The Role of Self-Concept and Motivation Within the "Computational Thinking" Approach to Early Computer Science Education

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
Luzia Leifheit
aus Greven

Tübingen
2020

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen.

Tag der mündlichen Prüfung: 17.12.2020
Stellvertretender Dekan: Prof. Dr. József Fortágh

1. Berichterstatter: Prof. Dr. Klaus Ostermann
2. Berichterstatterin: Prof. Dr. Enkelejda Kasneci

Contents

Danksagung

Interdisziplinäre Forschung ermöglicht neue fachdidaktische Erkenntnisse, stellt aber immer auch eine Herausforderung für die Beteiligten dar. Ich danke meinen Promotionsbetreuern Klaus Ostermann, Korbinian Moeller und Ulrich Trautwein für all die anregenden Diskussionen, die konstruktive Kritik und die Kompromissbereitschaft, durch die es uns gelungen ist, diese Herausforderung gemeinsam zu meistern. Auch bedanke ich mich bei Enkelejda Kasneci für Ihre Bereitschaft, die Berichterstattung zu dieser Dissertation zu übernehmen.

Für die Ermöglichung meines Promotionsvorhabens und die vielen Gelegenheiten zu Austausch und wissenschaftlicher Weiterbildung danke ich der LEAD Graduate School, im Besonderen Mareike Bierlich und Sophie Freitag.

Mein herzlicher Dank für die Unterstützung meiner Projektarbeit gilt dem Team der Wissenschaftlichen Begleitung der Hector Kinderakademien am Hector-Institut für Empirische Bildungsforschung, der Nachwuchsgruppe neuro-kognitive Plastizität am Leibniz-Institut für Wissensmedien und der Forschungsgruppe Programmiersprachen am Wilhelm-Schickard-Institut für Informatik. Besonders bedanke ich mich bei Kristin Funcke, Jessika Golle, Manuel Ninaus und Julian Jabs.

Mein besonderer Dank gilt Katerina Tsarava für die enge Zusammenarbeit und Freundschaft. Ich hätte mir keine bessere Projektpartnerin wünschen können.

Ich danke all meinen Mitdoktoranden und Freunden, die den Weg der letzten Jahre mit mir geteilt haben und ohne die dieses Abenteuer ein einsames Unterfangen gewesen wäre. Ganz besonders danke ich Molly für all die Schreibtipps, den gegenseitigen Ansporn und die geselligen Kaffeepausen, Christine für ihre ermutigenden und motivierenden Perspektiven, Heiko für seine ansteckende Begeisterung, Olga für ihre langjährige Freundschaft und Tutulla für ihre moralische Unterstützung und die vielen gemeinsamen Spaziergänge.

An meine gesamte Familie, allen voran meine Eltern Marie-Luise und Nils, meine Schwester Emily und mein Schwager Adrian, meine Oma Doris, meinen Opa Frido, meine Tante Lisa und meine Großtante Erika: Dankeschön, dass ihr immer an mich geglaubt habt, mich unterstützt habt und für mich da wart.

Meine größte Dankbarkeit möchte ich aber meinem Kollegen, Mann und Weggefährten Tobias aussprechen. Danke für jeden gemeinsamen Moment und für einfach alles. Ohne dich hätte ich diesen Weg wahrscheinlich nie beschritten.

Summary

As efforts for computer science education in elementary and secondary education are on the rise, there is an increasing number of efforts for integrating computing into official school curricula as well as extracurricular initiatives. Especially for elementary education, these efforts often follow the didactic approach of computational thinking, which places focus on the concepts, methods, and principles of computing rather than on specific technological applications.

Aside from advancing students' computational abilities, one of the most common goals of these efforts is fostering motivation and a positive self-concept with regard to computing. Motivation and self-concept have been found to be important predictors for educational outcome, particularly in mathematics and the sciences. However, due to a lack of reliable instruments for assessing computing-related motivation and self-concept in younger students, it is rarely empirically evaluated whether computing education efforts are successful in reaching their goal to foster such important motivational dispositions.

The aim of this dissertation is to address this gap by investigating (1) how student's motivation and self-concept can be assessed reliably within early computing education and (2) how computational thinking can be taught in a way that is motivating to elementary school students and beneficial for their self-concept.

These questions are explored in four empirical studies, progressing from an exploratory pilot study on methods for teaching computational thinking (Study 1) to the incremental development and evaluation of an instrument for assessing programming-related motivation and self-concept (Studies 2 and 3) to a hypothesis-driven randomized controlled field trial examining the effectiveness of a structured multi-component computational thinking training for fostering programming-related motivation and self-concept (study 4).

Zusammenfassung

Es gibt zunehmend Bemühungen, Informatik in die offiziellen Lehrpläne von Schulen zu integrieren oder in außerschulischen Angeboten zu unterrichten. Insbesondere Informatikbildung für jüngere Schülerinnen und Schüler folgt oft dem didaktischen Ansatz des informatischen Denkens, der den Schwerpunkt auf das Verständnis konzeptueller Grundlagen der Informatik und nicht auf die Benutzung bestimmter digitaler Anwendungen legt.

Dabei ist neben der Förderung informatischer Fähigkeiten eines der häufigsten Ziele, bei Kindern Motivation und positives Selbstkonzept in Bezug auf Informatik zu fördern. Motivation und Selbstkonzept haben sich als wichtige Prädiktoren für den Bildungserfolg erwiesen, insbesondere in der Mathematik und den Naturwissenschaften. Da es jedoch an zuverlässigen Instrumenten zur Beurteilung von informatischem Selbstkonzept und Motivation fehlt, die für jüngere Schülerinnen und Schüler geeignet sind, wird nur selten empirisch evaluiert, ob Bildungsmaßnahmen in der Informatik das Ziel erreichen, diese wichtigen motivationalen Einstellungen zu fördern.

Das Ziel dieser Dissertation ist es, diese Lücke zu schließen, indem untersucht wird, (1) wie Motivation und Selbstkonzept von Schülerinnen und Schülern in der frühen Informatikbildung zuverlässig erfasst werden können und (2) wie informatisches Denken so unterrichtet werden kann, dass informatisches Selbstkonzept und Motivation für Informatik bei Kindern gefördert werden.

Diese Fragen werden in vier empirischen Studien untersucht, die von einer explorativen Pilotstudie zur Lehr-Lernmethoden für den Unterricht in informatischem Denken (Studie 1) über die schrittweise Entwicklung und Evaluierung eines Instruments zur Erfassung von Motivation und Selbstkonzept in der Informatik (Studien 2 und 3) bis hin zu einer hypothesengeleiteten, randomisierten kontrollierten Feldstudie reichen, in der ein strukturiertes Training in informatischem Denken für Grundschulkinder auf seine Wirksamkeit für die Förderung von Motivation und Selbstkonzept in der Informatik untersucht wird (Studie 4).

List of Publications

The first three manuscripts included in this dissertation have already been published elsewhere. These manuscripts are reprinted here with permission from the original publishers. Contributorship for each manuscript is specified on the following pages.

1. **Leifheit, L.**, Jabs, J., Ninaus, M., Moeller, K., & Ostermann, K. (2018, October). Programming Unplugged: An Evaluation of Game-Based Methods for Teaching Computational Thinking in Primary School. In *Proceedings of the 12th European Conference on Game-Based Learning (ECGBL)*. Academic Conferences and publishing limited. (pp. 344–353).

2. **Leifheit, L.**, Tsarava, K., Moeller, K., Ostermann, K., Golle, J., Trautwein, U., & Ninaus, M. (2019, October). Development of a Questionnaire on Self-Concept, Motivational Beliefs, and Attitude Towards Programming. In *Proceedings of the 14th Workshop in Primary and Secondary Computing Education (WiPSCE)*. ACM. (pp. 1–9).

3. **Leifheit, L.**, Tsarava, K., Ninaus, M., Ostermann, K., Golle, J., Trautwein, U., & Moeller, K. (2020, June). SCAPA: Development of a Questionnaire Assessing Self-Concept and Attitudes Toward Programming. In *Proceedings of the 25th Conference on Innovation and Technology in Computer Science Education (ITiCSE)*. ACM. (pp. 138–144).

4. **Leifheit, L.**, Golle, J., Trautwein, U., Ostermann, K., Ninaus, M., & Moeller, K. (unsubmitted manuscript). Motivational Effects of an Extracurricular Computational Thinking Training for Elementary School Children: A Randomized Controlled Field Trial. (17 pages).

5. **Leifheit, L.**, Tsarava, K., Ninaus, M., & Moeller, K. (unsubmitted manuscript). "Verstehen wie Computer denken". Ein Trainingsprogramm zur Förderung von informatischem Denken und systematischen Problemlösefähigkeiten besonders begabter Kinder im Grundschulalter. Reihe Hector Core Courses. (254 pages).

Contributorship

1. **Leifheit, L.**, Jabs, J., Ninaus, M., Moeller, K., & Ostermann, K. (2018, October). Programming Unplugged: An Evaluation of Game-Based Methods for Teaching Computational Thinking in Primary School. In *Proceedings of the 12th European Conference on Game-Based Learning (ECGBL)*. Academic Conferences and publishing limited. (pp. 344–353).
   Author contributions:
   The study was initiated and funding was acquired by Ostermann and Jabs. Data were collected and prepared by Jabs. The research questions were mainly generated by me with input from Jabs and Ostermann. I analyzed and interpreted the data with input from Moeller and Ninaus. The manuscript was mainly written by myself with contributions from Jabs and editing from Ninaus, Moeller, and Ostermann. I estimate my contribution to this manuscript to be approximately 80% of the total work.

2. **Leifheit, L.**, Tsarava, K., Moeller, K., Ostermann, K., Golle, J., Trautwein, U., & Ninaus, M. (2019, October). Development of a Questionnaire on Self-Concept, Motivational Beliefs, and Attitude Towards Programming. In *Proceedings of the 14th Workshop in Primary and Secondary Computing Education (WiPSCE)*. ACM. (pp. 1–9).
   Author contributions:
   I designed the study and methodology together with Tsarava and with input from Moeller, Ninaus, Golle, and Trautwein. I designed the proposed assessment instrument with input from Golle and Ostermann. The research questions were generated by me. Data were collected and prepared by myself, Tsarava, and student assistants. I analyzed the data and interpreted the results with input from Moeller, Golle, and Ninaus. The manuscript was mainly written by myself with input from Tsarava, Moeller, Ninaus, Ostermann, Golle, and Trautwein. I estimate my contribution to this manuscript to be approximately 85% of the total work.

3. **Leifheit, L.**, Tsarava, K., Ninaus, M., Ostermann, K., Golle, J., Trautwein, U., & Moeller, K. (2020, June). SCAPA: Development of a Questionnaire Assessing Self-Concept and Attitudes Toward Programming. In *Proceedings of the 25th Conference on Innovation and Technology in Computer Science Education (ITiCSE)*. ACM. (pp. 138–144).

Author contributions:
I designed the study and methodology together with Tsarava and with input from Moeller, Ninaus, Golle, and Trautwein. I designed the proposed assessment instrument with input from Golle and Ostermann. The research questions were generated by me. Data were collected and prepared by myself, Tsarava, and student assistants. I analyzed and interpreted the data with input from Golle. The manuscript was mainly written by myself with input from Tsarava, Moeller, Ninaus, Ostermann, Golle, and Trautwein. I estimate my contribution to this manuscript to be approximately 80% of the total work.

4. **Leifheit, L.**, Golle, J., Tsarava, K., Trautwein, U., Ostermann, K., Ninaus, M., & Moeller, K. (unsubmitted manuscript). Motivational Effects of an Extracurricular Computational Thinking Training for Elementary School Children: A Randomized Controlled Field Trial. (17 pages).
   Author contributions:
   I designed the study and methodology together with Tsarava and with input from Moeller, Ninaus, Golle, and Trautwein. The training was designed by Tsarava and myself with input from Moeller, Ninaus, and Ostermann. The research questions were generated by me. Data were collected and prepared by myself, Tsarava, and student assistants. I analyzed and interpreted the data with input from Golle. The manuscript was mainly written by myself with input from Moeller, Tsarava, Ninaus, Ostermann, Golle, and Trautwein. I estimate my contribution to this manuscript to be approximately 80% of the total work.

5. **Leifheit, L.**, Tsarava, K., Ninaus, M., & Moeller, K. (unsubmitted manuscript). "Verstehen wie Computer denken". Ein Trainingsprogramm zur Förderung von informatischem Denken und systematischen Problemlösefähigkeiten besonders begabter Kinder im Grundschulalter. Reihe Hector Core Courses. (254 pages).
   Author contributions:
   The training was designed by Tsarava and myself. Tsarava and I conducted the literature review with input from Moeller and Ninaus. The manuscript was mainly written by myself with approximately 30% contribution from Tsarava and editing from Ninaus and Moeller. I estimate my contribution to this manuscript to be approximately 50% of the total work.

Erklärung nach § 5 Abs. 2 Nr. 8 der Promotionsordnung der Math.-Nat. Fakultät
-Anteil an gemeinschaftlichen Veröffentlichungen-
Nur bei kumulativer Dissertation erforderlich!

Declaration according to § 5 Abs. 2 No. 8 of the PhD regulations of the Faculty of Science
-Collaborative Publications-
For Cumulative Theses Only!

Leifheit, Luzia:

List of Publications

1. Programming Unplugged: An Evaluation of Game-based Methods for Teaching Computational Thinking in Primary School

2. Development of a Questionnaire on Self-concept, Motivational Beliefs, and Attitude Towards Programming

3. SCAPA: Development of a Questionnaire Assessing Self-Concept and Attitudes Toward Programming

4. Motivational Effects of an Extracurricular Computational Thinking Training for Elementary School Children: A Randomized Controlled Field Trial

5. „Verstehen wie Computer denken". Ein Trainingsprogramm zur Förderung von informatischem Denken und systematischen Problemlösefähigkeiten besonders begabter Kinder im Grundschulalter. (Manual zum gleichnamigen Hector Core Kurs)

Nr.	**Accepted publication yes/no**	**List of authors**	Position of candidate in list of authors	Scientific ideas by the candidate (%)	Data generation by the candidate (%)	Analysis and Interpretation by the candidate (%)	Paper writing done by the candidate (%)
			Optionally, you can also declare the above-stated categories in a written statement on a separate sheet of paper.				
1	yes	**Luzia Leifheit**, Julian Jabs, Manuel Ninaus, Korbinian Moeller, Klaus Ostermann	first author	70	0	90	90
2	yes	**Luzia Leifheit**, Katerina Tsarava, Korbinian Moeller, Klaus Ostermann, Jessika Golle, Ulrich Trautwein, Manuel Ninaus	first author	80	50	90	90

3	yes	**Luzia Leifheit**, Katerina Tsarava, Manuel Ninaus, Klaus Ostermann, Jessika Golle, Ulrich Trautwein, Korbinian Moeller	first author	80	50	90	90
4	no	**Luzia Leifheit**, Katerina Tsarava, Jessika Golle, Ulrich Trautwein, Klaus Ostermann, Manuel Ninaus, Korbinian Moeller	first author	80	50	90	90
5	no	**Luzia Leifheit**, Katerina Tsarava, Manuel Ninaus, Korbinian Moeller	first author	30	does not apply* (no data were collected)	does not apply* (no data were analyzed or interpreted)	60

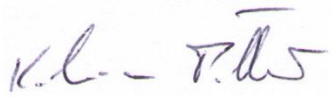
***** Manuscript 5 is not a research article, but a manual for instructors of a computational thinking course. It includes detailed information about the course regarding its educational objectives, didactic approach, instructional design, course structure, and learning materials. Because the manual does not report empirical results, no data were collected or analyzed.

I confirm that the above-stated is correct.

_____

Date, Signature of the candidate

I/~~We~~ certify that the above-stated is correct.



28.10.2020

Date, Signature of the doctoral committee or at least of one of the supervisors

1 Introduction

> *"The aim of education is to enable individuals to continue their education – or that the object and reward of learning is continued capacity for growth. [...] In our search for aims in education, we are not concerned, therefore, with finding an end outside of the educative process to which education is subordinate."*
>
> John Dewey, Democracy and Education

With the growing influence of computing technology in society, research, industry, and our private and social lives, the demand for computer science education is increasing – especially in elementary and secondary school (Balanskat & Engelhardt, 2015). However, efforts for integrating computing-related topics into school curricula vary widely across – and even within – countries (Balanskat & Engelhardt, 2015). The inconsistency of these efforts reflects the current state of empirical research on computing education, which does not yet provide conclusive results regarding which selection of content and which teaching methodologies might be most suitable for fostering programming skills and computational abilities in younger students.

Many of these efforts for promoting early computing education share the goal to stimulate students' interest, self-confidence, and motivation for programming and computing (e.g., Code.org; of Code; Kodable). This goal may be just as important as the goal to foster computing ability itself, considering the crucial role that attitudinal factors such as interest, confidence, or motivation generally play for educational outcomes.

Students' motivation for a school subject has been found to predict their achievement within that subject (Marsh, Köller, Trautwein, Lüdtke, & Baumert, 2005; Trautwein & Köller, 2005; Wilkins, 2004). Likewise, positive self-concept as an aspect of confidence has long been established to have an effect on students' educational outcome in science and mathematics and for driving continued pursuit of education in these domains (e.g., Armstrong & Price, 1982; Lantz & Smith, 1981;

Oliver & Simpson, 1988; Wilkins, 2004). Motivational dispositions students have as early as in elementary school can predict their educational trajectories, e.g., their course choices and performance in late high school (Wigfield & Cambria, 2010).

Considering the constitutive role self-concept and motivation play for educational achievement later in life, the goal to foster these characteristics in younger students seems worthwhile. However, it usually remains unclear whether computing efforts actually reach this goal. Owing to reasons such as a shortage of financial resources and lack of instruments for assessing such attitudinal factors (e.g., Vivian et al., 2020), the effect of computing education efforts on students' attitudinal characteristics is rarely evaluated empirically.

To make such evaluations possible and facilitate educational decisions regarding computing curricula in elementary and secondary school, it seems desirable (1) to reliably assess such student characteristics within early computing education and (2) to evaluate didactic approaches and teaching methods for promoting positive self-concept and motivation with regard to programming and computing.

1.1 Contribution goals

With the work presented in this dissertation, I seek to contribute to the scientific body of evidence on effective didactic approaches for early computing education, but also to the body of available evaluated teaching resources.

1.1.1 Scientific contribution

To facilitate further research on self-concept and motivation in early computing education, my goal was to design an assessment instrument for measuring these constructs and conduct studies aiming at the instrument's incremental validation. Such an instrument would be beneficial for evaluating the success of computing education efforts and could help teachers and educational policymakers make informed evidence-based decisions regarding the development or selection of computing education curricula and teaching materials.

Additionally, I wanted to contribute to the evidence on what works with regard to teaching computational thinking in a way that is motivating to elementary school students. Expecting that such evidence would provide orientation for fu-

ture development and evaluation of teaching resources, I analyzed the effectiveness of a multi-component computational thinking course for promoting students' programming-related self-concept and motivational dispositions toward programming.

1.1.2 Educational contribution

My aim also was to create and evaluate teaching materials in the form of a structured computational thinking course for elementary school students. Potential educational benefits of this are twofold. First, the availability of evaluated teaching resources can help educators design new resources or integrate the evaluated resources into their own lessons and curricula. Second, if such a course is established to be effective and can be scaled up and offered long-term, it would be of direct benefit to many cohorts of participating students.

1.2 Overview

The cumulative nature of this dissertation results in overlapping content between the manuscripts in the appendix and the frame of the dissertation. In particular, there is overlap between Chapter 2 Theoretical Framework and the theoretical background outlined in the individual manuscripts. For obvious reasons, the summaries of the research goals, study design, and findings of each study presented in Chapter 3 Research Questions and Studies and Section 4.1 Main outcomes overlap with the respective manuscript. Further, Chapter 1 Introduction and Section 4.2 Integrated discussion partially overlap with introductions and discussions of the individual manuscripts.

1.2.1 Theoretical framework

The shared theoretical basis of the four studies included in this dissertation is outlined in Chapter 2 Theoretical Framework. The chapter provides an overview of the discussion surrounding computational thinking, the current state of early computing education, and discusses the role of self-concept and motivation.

1.2.2 Studies

At the core of this dissertation are two different but interconnected lines of research. On the one hand, my goal was to investigate how students' motivational dispositions toward programming can be assessed reliably. On the other, I aimed to explore how computational thinking can be taught in a way that is motivating to elementary school students. The studies included in this dissertation follow these two lines of research, progressing from an exploratory pilot study identifying assessment and research design needs to instrument development and evaluation to a hypothesis-driven effectiveness study using a robust research design permitting causal interpretation.

Study 1 lays the groundwork for both lines of research by exploring unplugged game-based learning activities for teaching computational thinking (Manuscript 1, Leifheit, Jabs, Ninaus, Moeller, & Ostermann, 2018), opening up the questions how motivational dispositions can be assessed reliably and which teaching methods are effective for promoting such positive dispositions in elementary school students.

After identifying a lack of measurement instruments for assessing computing-related self-concept and motivation at the elementary level, we designed and initially evaluated an instrument for assessing these constructs in Study 2 (Manuscript 2, Leifheit et al., 2019) and further examined the instrument for reliability and construct validity in Study 3 (Manuscript 3, Leifheit et al., 2020).

Study 4 integrates the findings from all previous studies, reconnecting the two main research lines of this dissertation. The study investigates the effectiveness of a multi-component computational thinking training for promoting a positive programming-related self-concept and motivation for programming in elementary school students (Manuscript 4, Leifheit et al., in preparation).

The research goals, study design, and analysis procedure for each of these studies are outlined in Chapter 3 Research Questions and Studies. I present and discuss the findings of all studies in Section 4.1 Main outcomes.

2 Theoretical Framework

2.1 Computational thinking

In this section, I discuss the origins of the term computational thinking as well as some of the most prominent conceptions that are used today.

2.1.1 History and present discourse

The term computational thinking was coined by Papert (1980) to describe those thought processes that play a fundamental role in the systematic development of computational procedures. In the four decades since, computational thinking has been conceptualized and discussed from many different perspectives. The term became especially popular after Wing published the first in her series of articles on the topic, characterizing computational thinking as a fundamental and universally applicable skill set for solving problems that every child should learn (Wing, 2006).

Today, the term is used with a range of connotations. Some argue computational thinking to reflect computational skills learned through programming, and consequently hold that computational thinking is intrinsically linked to programming (Nardelli, 2019). Others consider computational thinking a novel and distinct way of thinking that can solve problems going beyond what can be achieved through other forms of problem solving, as pointed out by Denning (2017). In addition, computational thinking has been described as a cognitive ability (e.g., García Peñalvo et al., 2016; Shute, Sun, & Asbell-Clarke, 2017) and found to be empirically related to already defined cognitive abilities (e.g., Román-González, Pérez-González, & Jiménez-Fernández, 2017; Tsarava et al., 2019).

In the following sections, I further outline the ideas of computational thinking as problem solving and as a cognitive ability before discussing computational thinking as a didactic approach for conceptual learning.

2.1.2 Problem solving

Computational thinking can be considered a form of problem solving (e.g., Lu & Fletcher, 2009; Wing, 2006; Yadav, Hong, & Stephenson, 2016) that is narrower and more precise than that of general problem solving, i.e. skills and strategies to solve problems of any kind. Computational thinking is not only about finding solutions to problems, but about finding algorithmic solutions (Barr & Stephenson, 2011) that are algorithmic, systematic, abstract, and computable (Lu & Fletcher, 2009; Wing, 2008) – but it does not matter whether the computation is ultimately performed or the algorithm executed by a person or a machine (Wing, 2014). Because many problems can be understood and potentially solved in an algorithmic, systematic, abstract, and computable way, computational thinking can play an important role for general problem solving.

Complex problems whose solution engages computational thinking are usually defined by a multitude of factors. Solutions to such problems typically require processes of generalization, abstraction, decomposition into subproblems, and developing algorithmic solutions (Kazimoglu, 2013; Wing, 2006). An algorithmic solution is one that can be formulated in an unambiguous way that leaves no room for interpretation. This means that for a solution to be considered algorithmic, it must be clear after each step which step is to be executed next or which conditions are to be considered to decide which step is executed next. For the purpose of introducing children to programming, such algorithmic solutions are considered as consisting of a basic set of computational thinking concepts in the form of control-flow building blocks, such as sequences, loops, events, and conditionals (Brennan & Resnick, 2012).

2.1.3 Cognitive ability

According to cognitive conceptualizations, computational thinking refers to the cognitive processes that play an essential role in solving computational problems (García Peñalvo et al., 2016). At the same time, making a larger claim, computational thinking is argued to represent a cognitive ability with applicability in a wide spectrum of domains going beyond the field of computing (Shute et al., 2017).

In empirical research, computational thinking has been found to be closely associated with a number of established cognitive abilities, such as reasoning, spatial,

verbal, and arithmetic abilities (e.g., Román-González et al., 2017; Tsarava et al., 2019).

2.1.4 Didactic approach for conceptual learning

Although the scope and meaning of computational thinking have been debated extensively in recent years, the idea that underlying concepts are the most essential aspect of computer science has already been stressed by Donald Knuth (1974). While specific technological applications are developing rapidly, the underlying computational concepts remain the same (Wing, 2006) and are applicable more universally (Wing, 2011). Accordingly, when I refer to computational in this dissertation, I consider it to be the conceptual core of computing defined by concepts, methods, and principles rather than by specific technology or applications (Nardelli, 2019).

Consequently, the didactic approach of computational thinking teaches computing with a focus on conceptual understanding of fundamental computational concepts, rather than on the use of specific digital technologies or programming applications. For this reason, Prottsman (2014) advocates for the use of unplugged activities to first introduce computational thinking in elementary education.

2.2 Early computing education

To be successful, early computing education faces many challenges, including having to consider children's cognitive development and finding ways to be integrated into curricula. This section presents an overview of how computing education is currently situated in elementary as well as secondary school curricula and points out important factors to consider regarding the cognitive development of younger students.

2.2.1 Current situation

Already in the 1970s, it has been proposed that learning to program can help children reflect their own cognitive processes, because writing interactive programs requires contemplating and anticipating any possible misunderstanding or mistake that the program's users could encounter or make (e.g., Papert, 1972). On this

basis, Papert (2005) suggested that teaching programming in elementary school could be beneficial for the development of children's reasoning abilities.

Nowadays, promoting children's computing competences is considered to reflect the "21st century vision of students who are not just computer users but also computationally literate creators who are proficient in the concepts and practices of computer science" K-12 CS. It is not surprising that efforts for computing education at the elementary level are becoming increasingly popular, not only in the form of extracurricular initiatives (e.g., Code.org; of Code; Kodable), but also as classes integrated into official elementary school curricula (e.g., Balanskat & Engelhardt, 2015; Dredge, 2014; Wing, 2014).

Throughout Europe, efforts in education in logical thinking, problem solving and programming are on the rise. This is also reflected in the fact that more and more countries are already integrating or planning to integrate computational thinking and programming into curricula at the elementary school level (e.g., Balanskat & Engelhardt, 2015).

In 2012, the British Royal Society published the statement that "every child should have the opportunity to learn computing at school" (Wing, 2014). Subsequently, in September 2014, the Computing at School program was launched, which introduced computing as a subject for all students in each year of elementary and secondary education (Dredge, 2014). In addition, the curriculum for computer science in UK universities was revised in 2013, to include computational thinking ("Restart: The resurgence of computer science in UK schools", n.d.).

The relevance and importance of computational thinking is also reflected in the founding of the European Coding Initiative in 2014, aiming to promote the integration of computational thinking and programming into official curricula (Balanskat & Engelhardt, 2015). Moreover, in the United States, there currently are numerous programs – e.g. CE21, CPATH, BPC, CS 10K, CS4HS or Computing in the Core, to name but a few (Wing, 2014) – which aim to prepare teachers and to further computing education at the elementary and secondary level (Wing, 2014). Similar developments can also be observed in China, Korea and Singapore (Wing, 2014).

2.2.2 Cognitive development

Efforts for integrating computing into elementary education might seem promising. However, it is important to consider that students need to be at a stage of

cognitive development rendering the acquisition of computational skills possible.

The concrete operational stage at which first capabilities for abstraction are developed (according to Piaget, 1972) is typically reached between the age of 7 to 11 years. Considering how crucial abstraction is for computing, this suggests that kindergarten or preschool would be too early for children to learn computational thinking (Armoni & Gal-Ezer, 2014).

Instead, introducing elementary school students to programming seems developmentally appropriate – according to Neo-Piagetian models of cognitive development with regard to learning programming, students first show a purposeful approach to programming at this stage (Lister, 2016) and are able to write small programs when they are given well defined specifications (Teague et al., 2012). Elementary school students typically already have preconceptions of programming, commonly associating it with objects – such as computers, apps, viruses, or robots – as well as actions – such as creating, typing, inventing, adjusting, or combining (**?**).

For elementary education, following a didactic approach of computational thinking that uses unplugged exercises (e.g., board and card games or pen-and-paper exercises) is recommended to support comprehension of abstract computational concepts (Prottsman, 2014), which children will need to understand when they start learning to program (Kumar, 2014),

2.3 Self-concept and motivation

In this section, I introduce the constructs of self-concept and motivation, providing theoretical definitions, explaining how the constructs are related to educational achievement, and illustrating the gap in assessment instruments for measuring these constructs in early computing education.

Definitions for these motivational constructs and an overview of assessment instruments were already detailed in Manuscript 2 (Leifheit et al., 2019) and Manuscript 3 (Leifheit et al., 2020), but are included here as well to provide important context for the following chapters. Therefore, Section 2.3.1 Definitions largely overlaps with the theoretical background detailed in Manuscript 2 (Leifheit et al., 2019), while Section 2.3.3 Assessment largely overlaps with the theoretical background detailed in Manuscript 3 (Leifheit et al., 2020).

2.3.1 Definitions

Academic self-concept has been defined as self-perception with respect to achievement in school (Reyes, 1984). Correspondingly, students' self-concept regarding a specific subject reflects their confidence in their own ability to do well in that subject (Reyes, 1984; Wilkins, 2004). Positive self-concept was found to be an important predictor of achievement: When students have no confidence in their ability to perform well in a subject, they have no reason even attempting to succeed (Oliver & Simpson, 1988).

Motivational beliefs are attitudes toward a subject that motivate or demotivate a student to engage with a subject and make the effort required for achieving in it. As such, these beliefs reflect the motivational value a student attributes to a subject or task and thus are also termed value beliefs. Positive motivational beliefs have been associated with students' persistence in attempting to perform well even when their interest and intrinsic enjoyment of the subject decrease (Oliver & Simpson, 1988). Eccles and colleagues differentiate between four aspects of motivational value belief: intrinsic value belief, attainment value belief, utility value belief, and cost belief (Eccles et al., 1983; Wigfield & Eccles, 1992), all of which are characterized below.

Intrinsic value belief with respect to a subject is defined as the degree to which a student intrinsically enjoys and is interested in the subject. This construct reflects a student's attitude toward the subject. While a positive attitude toward a subject is not a necessary requirement for achievement, it can support and increase student' engagement (Oliver & Simpson, 1988). Furthermore, intrinsic value belief about a subject is a predictor for students' voluntary engagement with a subject in their leisure time (Durik, Vida, & Eccles, 2006; Nagengast et al., 2011).

Attainment value belief regarding a subject reflects the importance a student places on that subject (Gaspard, 2017).

Utility value belief about a subject represents students' expectation of the subject's usefulness in different areas of life, such as everyday life, school, future career, or social life (Gaspard, 2017).

Intrinsic value belief and attainment value belief are considered intrinsic motivational factors, while utility value belief is seen as an extrinsic motivational factor (Trautwein et al., 2013). All three of these beliefs have been linked with future career ambition and selection of classes (Durik et al., 2006).

Cost belief about about a subject reflects the negative consequences a student

expects to result from engaging in the subject, including assumed effort and exhaustion as well as negative emotions (Gaspard, 2017; Perez, Cromley, & Kaplan, 2014).

2.3.2 Connection to educational achievement

For other school subjects, motivational dispositions were repeatedly observed to be significant predictors of students' educational achievement (Marsh et al., 2005; Trautwein & Köller, 2005; Wigfield & Cambria, 2010; Wilkins, 2004). Positive self-concept as an aspect of confidence has long been established to have an effect on students' educational outcome, particularly in science and mathematics, and for driving continued pursuit of education in these domains (e.g., Armstrong & Price, 1982; Lantz & Smith, 1981; Oliver & Simpson, 1988; Wilkins, 2004). Students' motivational dispositions from as early as elementary school can predict their educational trajectories, e.g., their course choices and performance in late high school (Wigfield & Cambria, 2010).

Taking into account the important role motivational constructs play for educational outcomes, it seems desirable to also assess these factors for programming. Measuring these factors reliably would allow for assessing the development of students' self-concept and motivation toward programming and thus for evaluating the success and the actual effects of early computing education efforts.

2.3.3 Assessment

Several questionnaires for measuring students' attitudes toward computing have been proposed and evaluated in recent years. Generally, these questionnaires fall into one of two categories: the first type of questionnaire has a more narrowly defined focus on computing, while the latter more broadly assesses attitudes toward computer use.

One of the instruments from the first category is the Computing Attitudes Survey (CAS) for assessing college students' attitudes toward problem solving and their beliefs about the nature of knowledge within computer science (Dorn & Elliott Tew, 2015). Other instruments in this category include the questionnaire for measuring high school students' attitudes toward computer science and information technology (Heersink & Moskal, 2010) and the Computer Science Attitude Survey for undergraduate science and engineering students (Hoegh & Moskal,

2009).

The other category includes questionnaires measuring attitudes toward computer use, such as the Microcomputer Beliefs Inventory (MBI) (Riggs & Enochs, 1993) for assessing middle school students' self-efficacy and outcome expectancy beliefs regarding computer use (Riggs & Enochs, 1993). Another questionnaire from this category measures high school students' attitudes toward interacting with computers (Selwyn, 1997).

Due to the instruments' respective focus areas and target groups, questionnaires from neither of these categories seem well-suited for assessing elementary school students' self-concept and motivational dispositions toward programming. Existing domain-specific instruments developed for measuring attitudes toward computing – as is the case for questionnaires from the first category – are mainly aimed at high school and college students with previous computing experience (e.g., Dorn & Elliott Tew, 2015). Accordingly, item phrasing is not suited for the target group of elementary school students because it is too complex. Questionnaires from the second category are also aimed at younger students (i.e., at middle and high school students) and do not require previous computing experience. However, they do measure attitudes toward computer use rather than computing or programming, which makes them too broad for specifically assessing students' motivational dispositions toward computing and programming activities. Considering the assessments available, there still is a need for validated instruments for assessing students' self-concept and motivational dispositions toward programming, in particular for elementary school students.

3 Research Questions and Studies

This chapter provides an overview of the overarching research questions as well as the individual research questions of each study included in this dissertation before outlining the research design for each of the studies.

3.1 Overarching research questions

In this dissertation, I aim to answer two main interconnected research questions:

1. How can computing-related motivational dispositions be assessed reliably in early education?
2. How can computational thinking be taught in a way that is motivating to elementary school students and beneficial for their self-concept?

The following section explains how each of the studies included in this dissertation contributes to answering these main lines of research.

3.2 Studies

The studies with which I aspire to answer the aforementioned research questions progress from an exploratory pilot study to a hypothesis-driven study using a robust research design to examine the effectiveness of a computational thinking training for fostering self-concept and motivation for programming.

The goal of Study 1 was to explore the suitability of unplugged game-based activities for helping elementary school students form an initial understanding of computational thinking concepts and for motivating them to want to learn more about computing (Manuscript 1, Leifheit et al., 2018).

Studies 2 and 3 aimed to incrementally evaluate a new instrument for measuring students' self-concept and motivational beliefs regarding programming by

gathering evidence on its validity and reliability (Manuscript 2 and Manuscript 3, Leifheit et al., 2019, 2020).

Findings and considerations from the first three studies were connected in the design of Study 4. With this study, our goal was to evaluate the effectiveness of a structured computational thinking training for promoting a positive programming-related self-concept and motivation for programming, as well as to explore whether students' gender had any differential effect on their training outcomes (Manuscript 4, Leifheit et al., in preparation).

However, groundwork for this research started before any of the studies could be conducted and included the design of an assessment instrument for measuring students' programming-related motivational dispositions (Manuscript 2 and Manuscript 3, Leifheit et al., 2019, 2020) as well as the incremental development of a structured 10-lesson course for training elementary school students in computational thinking (Manuscript 5, Leifheit, Tsarava, Ninaus, & Moeller, n.d.).

The specific research goals and research designs for each of these studies are detailed in the following sections.

3.2.1 Study 1: Programming Unplugged: An Evaluation of Game-Based Methods for Teaching Computational Thinking in Primary School

This chapter summarizes the study design and research goals of previously published work (Leifheit et al., 2018). The original publication is included in the Appendix as Manuscript 1.

Research goals

Our goal for this study was to assess the suitability of unplugged game-based activities for helping elementary school students form an initial understanding of computational thinking concepts. To this end, we evaluated the outcome of three lessons that used unplugged game-based activities as their core component. Each of the evaluated lessons introduced one computational thinking concept using an unplugged game-based activity. Specifically, (1) debugging was introduced through a relay race game, (2) conditional branching was introduced through a card game, and (3) events were introduced through a sports game (Leifheit et al., 2018).

Study context

The evaluated lessons were part of a computational thinking course consisting of 18 lessons with 45 minutes each. The lessons were modeled closely on course material from code.org (Course 2, code.org), which is available under Creative Commons Attribution 4.0 International License. The main objectives of the course were to promote students' conceptual understanding of computational concepts and spark their interest in computing. Lessons primarily addressed concepts and processes essential for creating simple algorithms, such as sequences, loops, conditional branching, and events. Additionally, students were taught how to debug algorithms. All concepts and processes were first introduced through unplugged game-based activities using tangible everyday objects (e.g., paper, pencils, or playing cards) instead of digital environments and abstract code. In more advanced lessons, students' conceptual understanding was applied and deepened through simple plugged-in programming exercises.

Overall, 10 lessons included unplugged activities, while the remaining 8 lessons included programming exercises using tablet devices and desktop computers. Over the course of the curriculum, each computational thinking concept was first introduced in at least one unplugged lesson and consequently applied in a plugged-in lesson.

The three lessons evaluated in Manuscript 1 (Leifheit et al., 2018) used a game-based learning approach to increase students' engagement in the learning activity (Plass, Homer, & Kinzer, 2015). The unplugged learning activities used in the course have been argued to be especially well-suited for introducing elementary school students to computing, because they allow for placing focus on computational concepts rather than technology use (Prottsman, 2014).

All of the lessons evaluated in this article followed the same structure. At the beginning of the respective lesson, students were asked to recall the previous lesson's activities and were encouraged to ask questions they have come up with in the aftermath of the previous lesson. Then, the terms and vocabulary for any new computational thinking concepts introduced in the current lesson were presented and discussed with students. The central part of the lesson always consisted of the respective game-based learning activity. After the game, the teacher prompted them to discuss what they learned from this activity and how this may be implemented when working with an algorithm.

Participants

The study participants were 33 students (15 female) from 3rd and 4th grade of German elementary school. Written informed consent of parents or legal guardians was required for the students to sign up for the course.

Study design and measurements

At the end of each lesson, students were handed an assessment sheet on the computational concept introduced in the lesson and were given ten minutes to solve it. Each assessment tested assigned learning objectives (Code.org, Course 2) and students' results were analyzed descriptively based on which percentage of the respective learning objectives they reached.

Additionally, students' interest in and motivation for programming education were measured with pre- and post-course self-assessment questionnaires.

The pre-course questionnaire asked students to indicate whether they had any previous programming experience or whether their parents were working in IT-oriented jobs.

In both the pre- and post-course questionnaire, students were asked to indicate their answer to the question "Could you imagine working in an IT job?" on a 5-point Likert scale (ranging from 1 = not at all, to 5 = absolutely, with only the endpoints labeled accordingly).

In the post-course questionnaire only, students were asked to indicate on the same 5-point Likert scale (i) whether they liked the course in general, (ii) whether they learned something new in the course, and (iii) whether they were interested in learning more about computing.

Analysis

For our study, we evaluated all pre- and post-course questionnaires as well as the short tests from each of the three lessons with unplugged game-based activities as their core component.

Connection to the other studies

The results of this exploratory study were considered in the development of a new computational thinking training (see Manuscript 5, Leifheit et al., n.d.). Its

methodological limitations informed the development of a new instrument for assessing motivational dispositions in elementary school students (see Manuscript 2 and Manuscript 3, Leifheit et al., 2019, 2020) as well as the design of a new study on teaching computational thinking (see Manuscript 4, Leifheit et al., in preparation).

3.2.2 Study 2: Development of a Questionnaire on Self-concept, Motivational Beliefs, and Attitude Towards Programming

This chapter summarizes the study design and research goals of previously published work (Leifheit et al., 2019). The original publication is included in the Appendix as Manuscript 2.

Research goals

To address the lack of measurement instruments for assessing computing-related self-concept and motivation at the elementary level (e.g., Leifheit et al., 2019; Vivian et al., 2020), we developed such an instrument based on an existing instrument for measuring students' self-concept and motivational beliefs regarding mathematics (Brisson et al., 2017; Gaspard, 2017; Leifheit et al., 2019), which had already been adapted and validated for other school subjects, including biology, physics, and languages (Gaspard, Häfner, Parrisius, Trautwein, & Nagengast, 2017). The proposed instrument assesses aspects of students' self-concept and motivational disposition regarding programming on 50 items across seven scales: (1) self-reported programming experience and understanding, (2) self-concept, (3) intrinsic value belief, (4) attainment value belief, (5) utility value belief, (6) cost belief, and (7) compliance and persistence (Leifheit et al., 2019) (see Appendix, Manuscript 2). Definitions of these constructs were introduced in Section 2.3.3 Assessment.

The aim of our pilot study was to gather first evidence on the validity of the proposed instrument to explore its approach for assessing programming-related self-concept and motivation before proceeding with further validation efforts. To this end, we examined whether the intercorrelations of the subscales of the proposed instrument were similar to those of the instrument it was based on.

Hypotheses

We expected students' responses for the seven subscales of academic self-concept and motivational beliefs to be correlated because they measure related constructs (see corresponding findings for mathematics, Gaspard, 2017). However, since the subscales measure distinct aspects of self-concept and motivational beliefs, we did not expect very high correlations, nor did we expect all of the subscales to be correlated with each other. Our expectations are based on the correlational patterns between the corresponding subscales on the mathematics questionnaire on which we modeled the proposed instrument (Brisson et al., 2017; Gaspard, 2017).

Specifically, we expected programming-related self-concept to be correlated with three other aspects, namely with intrinsic value belief, utility value belief, and compliance and persistence. In previous research, confidence as an element of self-concept and interest as a form of intrinsic value belief have been found to be significantly correlated within computing education (Wang, Hejazi Moghadam, & Tiffany-Morales, 2017). In mathematics, self-concept was found to be a strong predictor for both students' intrinsic value belief, and their persistence and compliance (e.g., Denissen, Zarrett, & Eccles, 2007; Marsh et al., 2005; Trautwein, Lüdtke, Roberts, Schnyder, & Niggli, 2009). Moreover, students' utility value belief about mathematics has been found to correlate with their mathematics self-concept (e.g., Chouinard, Karsenti, & Roy, 2007; Husman & Hilpert, 2007; Trautwein & Lüdtke, 2009)).

We also expected intrinsic value belief to correlate with cost belief, because these two aspects were found to be highly correlated for motivational beliefs about mathematics (Gaspard, 2017).

Lastly, we expected self-concept, motivational beliefs, and compliance and persistence to correlate with computational thinking ability, because these aspects of motivation have been identified to predict achievement in other school subjects (e.g., Marsh et al., 2005; Oliver & Simpson, 1988; Wigfield & Cambria, 2010; Wilkins, 2004).

Participants

We evaluated data from 31 students in 3rd and 4th grade of elementary school who participated in a computational thinking course within an extracurricular en-

richment program for elementary school children in the German federal state of Baden-Württemberg. Teachers nominate students to participate in the program based on interest and achievement in school. Students who are nominated can then choose which courses from the program they want to attend. Before the start of the study, we obtained written informed consent from all participating students as well as from their parents or legal guardians.

Study design

We collected all data in a pilot field study with repeated measures. One week before the start of the course, all participating students took a pretest and one week after the end of the course, they took a posttest. In the two test sessions, students were assessed in the normal course classrooms using paper tests and questionnaires.

Measurements

In both test sessions, we collected data using three different instruments. First, we used the newly developed instrument to assess motivational constructs on seven subscales, each consisting of four to twelve 4-point Likert-scale items (1 = agree completely; 2 = rather agree; 3 = rather do not agree; 4 = do not agree at all).

Second, we assessed selected aspects of students' socioeconomic background as well as frequency of computer or tablet use with the TIMSS 2015 Context Questionnaire (Wendt et al., 2016), because correlations found in educational research are often mediated of socioeconomic status (Niepel, Stadler, & Greiff, 2019).

Last, we assessed students' computational thinking ability using the Computational Thinking test (CTt, Román-González, 2015). As the CTt is originally aimed at middle school students, we selected only the 21 easiest (out of 28) items to be included in our measurements.

For our study presented in Manuscript 2, we focused on students' ratings of their self-concept, motivational beliefs, and attitudes toward programming at pretest to evaluate their responses to the proposed instrument before their attitude was influenced by the course. To identify potential effects of these constructs on computational thinking ability, we also considered students' performance in a computational thinking ability test at pretest and posttest and calculated the change score.

Analysis

We calculated bivariate Pearson correlations to assess the relationship between the seven subscales for self-concept and motivation, as well as socioeconomic background, frequency of computer/tablet use, and the change score for computational thinking ability.

The amount of missing data ranged between 3.2% and 22.6%. As missing data treatment, we used multiple imputation, which represents the state of the art of missing data treatment Schafer and Graham (2002) and has been recommended for educational research Cheema (2014). Multiple imputation produces standard errors reflective of missing-data uncertainty Schafer and Graham (2002), realistically models random variation Cheema (2014), and provides high accuracy of estimation especially for smaller sample sizes Cheema (2014).

Connection to the other studies

After the manuscript reporting on our development and evaluation of the instrument was published (Manuscript 2, Leifheit et al., 2019), we named the instrument the Self-Concept and Attitude toward Programming Assessment (SCAPA) and took further steps toward its validation (see Manuscript 2, Leifheit et al., 2020). We later used the instrument to assess the effects of a computational thinking training on elementary school students' programming-related self-concept and motivation (Manuscript 4, Leifheit et al., in preparation).

3.2.3 Study 3: SCAPA: Development of a Questionnaire Assessing Self-Concept and Attitudes Toward Programming

This chapter summarizes the study design and research goals of previously published work (Leifheit et al., 2020). The original publication is included in the Appendix as Manuscript 3.

Research goals

Based on the promising findings of Study 2 (see Section 3.2.2, Manuscript 2, Leifheit et al., 2019), the present study takes a further step toward the validation of the proposed instrument for assessing students' self-concept and attitude toward programming. To provide first evidence regarding the internal consistency

and construct validity of the proposed Self-Concept and Attitude toward Programming Assessment (SCAPA), we examined the instrument's reliability (i.e., internal consistency on item and scale level) and construct validity (by means of confirmatory factor analysis).

Participants

In this study, we analyzed data collected from 197 elementary school students between 7 and 10 years old ($M = 8.83; SD = 0.73$) attending 3$^{\text{rd}}$ and 4$^{\text{th}}$ grade of German elementary school. All children participating in the present study attended a computational thinking course offered within the context of an extracurricular enrichment program in the German federal state of Baden-Württemberg. They had been nominated by their teachers to participate in the enrichment program based on interest and school achievement. Yet, they were not nominated for specific courses, but selected the courses they wanted to attend. Before the start of the study, we obtained written informed consent from all participating students as well as from their parents or legal guardians.

Study design

All data collection took place in the regular course classrooms. One week before the computational thinking course started, the participating students attended a test session, in which they were assessed using SCAPA in the form of a pen-and-paper questionnaire.

Measurements

SCAPA assesses students' self-concept and attitude toward programming on the seven scales (1) self-reported programming experience and understanding, (2) self-concept, (3) intrinsic value belief, (4) attainment value belief, (5) utility value belief, (6) cost belief, and (7) compliance and persistence. Definitions of these constructs were introduced in Section 2.3.3 Assessment.

Scales (2), (3), (4), (5), (6), and (7) were developed based on an existing instrument that had previously been validated for assessing the same constructs for mathematics (Brisson et al., 2017; Gaspard, 2017; Trautwein & Köller, 2005). Scale (1) was not modeled on the existing instrument for mathematics, but was

newly designed for SCAPA to allow for collecting indication of students' programming experience and understanding. All scales combined include 50 4-point Likert-type items (1 = agree completely; 2 = rather agree; 3 = rather do not agree; 4 = do not agree at all).

Analysis

We assessed all scales of the SCAPA instrument for internal consistency by calculating descriptive statistics and Cronbach's alpha to investigate the reliability of each of its scales.

To evaluate the construct validity of SCAPA's scales, we conducted a confirmatory factor analysis (CFA). Specifically, we calculated the model fit indices chi-squared test, RMSEA, SRMR, and CFI to assess how good the fit is between the model for each scale and the actual data. In addition, we examined to what extent the individual items of each scale contribute to the variance of the scale. To this end, we calculated the factor loading of each item on the scale it belongs to.

Connection to the other studies

Our evaluation SCAPA for construct validity and reliability built upon a previous study in which we had examined the pattern of intercorrelations between its scales to gather initial indications of the instrument's validity (Manuscript 2, Leifheit et al., 2019).

After establishing that the instrument had good model fit and its scales were largely reliable, we used SCAPA to assess the effectiveness of a computational thinking training for promoting programming-related self-concept and motivation in elementary school students (Manuscript 4, Leifheit et al., in preparation).

3.2.4 Study 4: Development and Evaluation of a Computational Thinking Course for Elementary School Students

This section presents a proposed training for computational thinking and a study investigating the effectiveness of the training. The first subsection (3.2.4.1) reports on the findings from our study on the effectiveness of the computational thinking training for promoting students' self-concept and motivation for programming. In the second subsection (3.2.4.2), I present the educational objectives, di-

dactic approach, instructional design, course structure, and learning materials of the computational thinking course evaluated in the effectiveness study.

3.2.4.1 Effects of an Extracurricular Computational Thinking Training for Elementary School Children: A Randomized Controlled Field Trial

This section summarizes the study design and research goals previously detailed in a manuscript not yet submitted for publication (Leifheit et al., in preparation). The manuscript is included in the Appendix as Manuscript 4.

Research goals

Our goal for this study was to evaluate the effectiveness of a structured computational thinking training for promoting a positive programming-related self-concept and motivation for programming.

The computational thinking training is aimed at 3rd and 4th grade elementary school students and follows the conceptually-oriented didactic approach of computational thinking. Thus, the focus of the training is not on using specific programming applications or technologies, but on understanding the following essential computational thinking concepts: sequences, loops, conditionals, events, variables, constants, and operators (as identified by Brennan & Resnick, 2012). The instructional design incorporates principles of embodied learning(e.g., Barsalou, 2008; Butz, 2016), game-based learning (e.g., Hamari, Koivisto, & Sarsa, 2014; Ninaus et al., 2015), scaffolding (Belland, 2014), and blended learning (unplugged/plugged-in) (del Olmo-Muñoz, Cózar-Gutiérrez, & González-Calero, 2020; Prottsman, 2014). A more detailed description of the computational thinking training is given in the following Section 3.2.4.2 Computational Thinking Course Manual (Manuscript 5, Leifheit et al., n.d.).

Our hypothesis was that the computational thinking training would have a positive effect on the development of students' programming-related self-concept and motivational dispositions. In addition, the aim of the study was to explore whether students' gender had any differential effect on their self-concept and motivation for programming. To assess this, our study followed a randomized controlled research design with repeated measures and a waiting control group. We used multiple linear regressions with maximum likelihood robust estimation to estimate the training and gender effects on seven outcome variables representing different

aspects of self-concept and motivation.

Furthermore, we assessed computational thinking ability and cognitive abilities to estimate the effects of the computational thinking training on computational thinking ability and reported the results separately (see Tsarava et al., in preparation).

Participants

Our participants for this study were 197 elementary school students ($female$: $N = 47;\ 24\%$) between 7 and 10 years old ($M = 8.83; SD = 0.73$) from 3rd and 4th grade of German elementary school. The computational thinking training – and, accordingly, our study on the effectiveness of the training – took place within the context of an extracurricular enrichment program for talented elementary school children. The program is being held at 66 local sites across the German federal state of Baden-Württemberg and offers courses, mostly from the STEM fields, that go beyond what children would learn in school. The proposed computational thinking training was offered as part of this program at 16 local sites. Elementary school teachers in Baden-Württemberg can nominate students for the program based on interest, motivation, and school performance. After students are nominated, they can choose which courses offered by the program they would like to participate in. Before students participated in our study along with the computational thinking training, we obtained written informed consent from all students as well as from their parents or legal guardians.

Study design

For our study, 29 course groups participated in the computational thinking training at 16 local sites. Each course group consisted of 4 to 6 students. At each site, both the training and data collection took place in the same regular classroom. Data collection was conducted by researchers or research assistants following detailed instructions during two test sessions.

All course groups were taught by teachers who had previously participated in a seminar instructing in teaching the course and had received a detailed course manual. The instructional seminar was taught by me (one of the developers of the computational thinking training) approximately one month before the beginning of the computational thinking course and study.

To allow for estimating the causal effect of the training (Torgerson & Torgerson, 2008) on both computational thinking ability and motivational attitudes toward programming, the study followed a randomized controlled trial design (e.g., Friedman, Furberg, & DeMets, 2010) with repeated measures and a waiting control group. After signing up for the computational thinking course and the accompanying study, students were randomly assigned to either the training condition or the control condition. One week prior to the start of the training, all participants took part in a pretest. At the end of the pretest session, students were informed about whether they had been randomly assigned to the training group or the waiting control group. Students in the training condition started attending their computational thinking courses one week after the pretest. The posttest for all participants from both conditions took place one week after the training group had completed their course. Students in the control condition had waited for the start of their course and did not receive any computational thinking training between pretest and posttest. One week after the posttest, the computational thinking course started for students in the waiting control condition. The study design is outlined in Figure 3.1.

In addition to permitting causal inference on the effectiveness of the training, this study design reduces the risk of drop-out among students in the control group and and allows all participating students to receive the same training. It is therefore considered to be in agreement with ethical standards for intervention trials (Emanuel, Abdoler, & Stunkel, 2010).

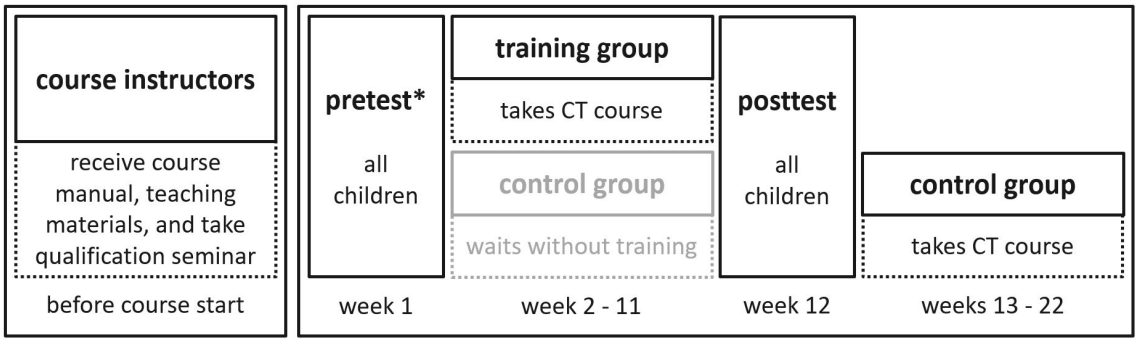


Figure 3.1: Study design: Randomized controlled field-trial with waiting control group and repeated measures (Figure taken from Leifheit et al., in preparation)

Measurements

In both test sessions, we measured variables falling into three categories: 1) self-reports on motivational constructs, 2) computational thinking ability, and 3) cognitive abilities (as control variables).

Motivational constructs, specifically programming-related self-concept and motivational beliefs about programming, were measured using the 7-scale, 50-item Self-Concept and Attitude toward Programming Assessment questionnaire (SCAPA Leifheit et al., 2019, 2020). This self-report instrument assesses aspects of motivation for programming on the seven scales (1) self-reported programming experience and understanding, (2) self-concept, (3) intrinsic value belief, (4) attainment value belief, (5) utility value belief, (6) cost belief, and (7) compliance and persistence. All SCAPA items items were rated on a 4-point Likert scale (1 = agree completely; 2 = rather agree; 3 = rather do not agree; 4 = do not agree at all).

As an indication of computational thinking ability, students' understanding of sequences, loops, conditionals, and simple functions was assessed using a shortened version of the Computational Thinking test (CTt, Román-González, 2015). All tasks on this multiple-choice type performance test consist of an algorithmic problem to be solved, a pictorial representation of the problem (e.g., a maze), a pseudocode-like representation of the problem (in visual block-based programming style), and four answer options. Because performance on the CTt has been found to be correlated with various cognitive abilities (Román-González et al., 2017; Tsarava et al., 2019), we assessed these abilities and used the measured variables as control variables for the CTt, but not for the motivational constructs.

The full line of reasoning for including these variables, as well as a detailed account of all measured variables, is provided in the Appendix in Manuscript 4 (Leifheit et al., in preparation).

Analysis

Before calculating any training effects, we tested for baseline equivalence of all dependent and control variables to establish the success of the randomized group assignment. To this end, we conducted two-tailed t-tests to check for potential differences at pretest between the two groups.

The amount of missing values in the collected data ranged between 4.1% and 16.2% per variable across both groups. Data were missing due to individual item

nonresponse and absence from posttest. Drop-out rates did not differ significantly between training group and control group ($\chi^2(1) = .45;\ p = .503$). When using T-tests to compare all dependent variable means of students present at posttest with students absent from posttest, we found no significant differences between the groups. These findings suggested data were missing at random (Enders, 2010). We confirmed this using Little's Missing Completely at Random test (Little, 1988), which revealed that data were indeed missing completely at random ($\chi^2(1) = 349.34;\ p = .503$). This permitted using the full-information maximum likelihood estimator as missing data treatment (Muthén & Muthén, 2012), which considers all available data when estimating model parameters and standard errors (Buhi, Goodson, & Neilands, 2008; Schafer & Graham, 2002).

To evaluate the effectiveness of the training, we used multiple linear regressions with maximum-likelihood robust estimation with study condition (training or control group) as predictor. Dependent variables were the seven motivational variables at posttest as assessed by the SCAPA questionnaire. For each variable, we included its pretest score as a control variable to increase the precision of the training effect estimate. As a result of the randomized controlled research design, the estimated training effects can be interpreted causally.

Additionally, we estimated the effect of gender on students' motivational development. To this end, we used another regression with gender as predictor and group assignment as a control variable. However, any gender effects estimated in this regression must only be interpreted correlationally and not causally, considering the predictor variable was not subject to randomized assignment.

We reported the training effects on computational thinking ability and cognitive abilities separately (for a detailed description of the analysis and results for cognitive abilities and computational thinking ability, see Tsarava et al., in preparation).

Connection to the other studies

The present study evaluated the effectiveness of the computational thinking training proposed in Section 3.2.4.2 (Manuscript 5, Leifheit et al., n.d.) for promoting a positive programming-related self-concept and motivation for programming. For the development of the computational thinking training and the design of this effectiveness study, we built on findings from all three of the previously outlined studies.

As previously mentioned in Section 3.2.1, the development of the computational

thinking training evaluated in the present study was informed by our findings from Study 1. One of the reasons for selecting the design of the present study was to overcome the methodological limitations of Study 1: Our new aim was to generate inferential, causally interpretable evidence regarding the effectiveness of the training through repeated measurements, a control group, and randomized group assignment (Manuscript 1, Leifheit et al., 2018).

To assess the outcome variables for motivational constructs, we used the Self-Concept and Attitude toward Programming Assessment (SCAPA). SCAPA was first proposed and described in Section 3.2.2 Study 2 (Manuscript 2, Leifheit et al., 2019) and later evaluated for reliability and construct validity in Section 3.2.3 Study 3 (Manuscript 3, Leifheit et al., 2020).

3.2.4.2 Computational Thinking Course Manual

This section summarizes the goals, didactic approach, and theoretical background of a computational thinking training previously detailed in the unpublished course manual for instructors of the training (Leifheit et al., n.d.). The manual was co-authored with Katerina Tsarava, Manuel Ninaus, and Korbinian Moeller. It incorporates some of the components and approaches suggested in a previous training proposal (see Tsarava et al., 2017) as well as considerations outlined in Section 3.2.1 Study 1 (Manuscript 1, Leifheit et al., 2018). The full course manual is included in the Appendix as Manuscript 5.

The computational thinking training was designed for groups of 6 to 10 students from 3rd and 4th grade and consists of 10 lessons. Each lesson is 90 minutes long and the training is intended to be taught as a weekly course over a timespan of 10 weeks.

In the following subsections, I will outline the educational objectives of the computational thinking course, its didactic approach and instructional design, the course structure and learning materials, and present an overview of the contents of the course manual.

Educational objectives

The computational thinking course aims to foster computational thinking abilities and to spark motivation for computing. More specifically, the course pursues the goal of helping children develop an initial understanding of basic computational

concepts and form a positive self-concept regarding their programming abilities.

Currently, elementary and lower secondary school curricula in Germany do not ensure that children can pursue computing education under professional and pedagogical guidance. It is unclear when students will have another opportunity to engage in computational thinking or programming activities after they complete the proposed computational thinking course. Therefore, the course also aims to enable students to program independently and pursue further computing education on their own.

To give students the skills they need in order to transfer their computational thinking understanding to different applications and familiarize themselves with new programming languages and environments, the course places focus on conceptual understanding and transfer. In the last lesson of the course, students are provided with a curated collection of links to further resources, such as online programming courses for children, to facilitate further engagement in computing-related activities independent of the availability of teachers or official school curricula.

Didactic approach and instructional design

To help students develop a conceptual understanding of computing, the course follows the conceptually-oriented didactic approach of computational thinking. Accordingly, the course does not focus on the use of specific programming applications or technologies, but on building an understanding of the following essential computational thinking concepts: sequences, loops, conditionals, events, variables, constants, and operators (as identified by Brennan & Resnick, 2012). The instructional design integrates principles of embodied learning(e.g., Barsalou, 2008; Butz, 2016), game-based learning (e.g., Hamari et al., 2014; Ninaus et al., 2015), scaffolding (e.g., Belland, 2014), and blended learning (unplugged/plugged-in) (e.g., del Olmo-Muñoz et al., 2020; Prottsman, 2014).

Embodied learning happens when students develop new understanding, knowledge, or skills through physical activity or manipulation of physical objects. Embodied teaching and learning methods facilitate implicit learning (Barsalou, 2008) and positively influence understanding and memory through increased spatial organisation of conceptual content (Noice & Noice, 2001). The theory of embodied cognition states that many aspects of cognition are dependent on the physical body, which can play a constitutive role in the cognitive processing of experiencs

and situations (Wilson & Foglia, 2015).

Game-based learning helps students form new understanding, knowledge, or skills through games or playful activities. The use of game-based teaching and learning methods can improve learners' performance (Ninaus et al., 2015), increase motivation, and incite students to actively engage in learning activities (Hamari et al., 2014; Plass et al., 2015).

Scaffolding is an instructional approach for helping students perform and advance in tasks they would not be able to complete without help (Belland, 2014). Scaffolding means that students are guided and – if necessary – supported in completing the learning tasks, e.g., through teacher support or written hints and instructions (Belland, 2014). The level of support is gradually decreased as students gain more skills in completing the tasks independently.

Blended unplugged and plugged-in learning takes place when an instructional setting integrates learning activities using digital media and technology (plugged-in) and learning activities without any digital or technological applications (unplugged). Unplugged activities are used to place focus on conceptual understanding rather than technology use and are accessible even to schools with little access to digital technology (Prottsman, 2014). Using a blended approach with a combination of unplugged and plugged-in activities can lead to better computational thinking performance and motivation compared to a plugged-in only approach (del Olmo-Muñoz et al., 2020).

Course structure and learning materials

Throughout the 10 lessons of the course, learning activities progress from concrete and playful unplugged activities (board games, card games, and pen-and-paper exercises) to more abstract and sophisticated plugged-in programming exercises (programming games, simulations, hardware, and simulated robots).

Students are initially familiarized with all computational thinking concepts introduced in the course through unplugged activities (as suggested by Prottsman, 2014) before they learn to apply the concepts in plugged-in activities.

The unplugged activities include the life-size board and card game series Crabs & Turtles (Tsarava, Moeller, & Ninaus, 2018a, 2018b) as well as pen-and-paper exercise sheets based on tasks from the Bebras international informatics challenge for children (Dagienė & Sentance, 2016).

In the first three lessons of the course, students are introduced to basic compu-

tational thinking concepts through playing Crabs & Turtles, which was developed specifically for this purpose and target group (Tsarava et al., 2018a, 2017). The Crabs & Turtles series – two board games and one card game – is deliberately designed to be independent of any specific programming languages or environments. Instead, Crabs & Turtles offers students a game-based and embodied experience for familiarizing themselves with basic computational thinking concepts.

In addition to unplugged games, the course also includes unplugged exercise sheets. At the end of each of the 10 lessons, students work on exercise sheets with puzzle-like tasks for 10 minutes. The tasks are based on the Bebras International Challenge on Informatics and Computational Thinking for children (Bebras), which has been established for 16 years and is being held in in 50 countries (Dagienė & Sentance, 2016). Bebras tasks engage students in computational thinking through motivating pen-and-paper puzzle exercises (Dagienė & Sentance, 2016). Within the computational thinking course, these exercise sheets are used so that students can recapitulate previously learnt computational thinking concepts in an unplugged way to strengthen their conceptual understanding.

Plugged-in activities are used to enable students to transfer and apply the computational thinking concepts to which they were introduced using unplugged activities. Scaffolded programming activity sheets guide students through three different programming languages and environments.

In the second lesson, students are introduced to plugged-in programming activities using the block programming language and environment Scratch 2.0, specifically developed for children (Diaz & Lopez, 2017). Subsequently, in the fourth to sixth lesson, Scratch allows students to apply the computational thinking concepts introduced through Crabs & Turtles and transfer their conceptual understanding to create basic programs (Diaz & Lopez, 2017), such as simple games and simulations.

In the seventh and eighth lesson, the possibilities of Scratch are extended through the open-hardware platform Arduino (Badamasi, 2014) as well as the programming language and environment S4A (Scratch for Arduino) (S4A), via which the Arduino's various ports and sensors can be controlled. After computational thinking concepts were introduced by Crabs & Turtles and applied in Scratch, the Arduino and S4A are used to transfer these concepts to an area going beyond the creation of simple screen applications to give students a first glimpse into hardware programming. This provides another embodied learning experience, as stu-

dents get to see that their program code can not only dictate what happens on the screen in front of them, but can also interact with their physical environment.

In the last two lessons of the course, the robot simulation Open Roberta gives students the opportunity to practice their acquired computational thinking understanding more independently by programming and controlling a simulated robot (Jost, Ketterl, Budde, & Leimbach, 2014). Unlike Scratch, Open Roberta offers no customizable graphical elements – the only way to control the simulated robot is through program code. Therefore, Open Roberta provides no distractions from programming itself, making the environment especially suited for open-ended exploration of programming.

Through all three of these programming environments, students are guided by scaffolded programming activity sheets. In the beginning of the course, these activity sheets provide students with clearly defined program goals and step-by-step instructions. As the course progresses, the programming activity sheets gradually provide less detailed instructions and, by the end of the course, even allow students to choose their own program goals.

At the end of the last lesson of the course, students receive a set of resources to help them to continue learning about computing without having to rely on teachers or further courses. Providing students with such resources is crucial for enabling them to have continued access to computing education materials, considering that elementary and lower secondary school curricula in Germany do not ensure students will soon have another opportunity to engage in computational thinking or programming.

Contents of the manual

The instructor's manual for the computational thinking course includes detailed information on the theoretical background of the course as well as practical teaching instruction and course materials.

First, the theoretical background of the course is explained. Specifically, the ideas of computational thinking, embodied learning, and game-based learning are illustrated. Second, the overarching learning objectives as well as individual lesson goals of the course are presented. Third, all learning materials of the course are introduced, including the unplugged games, exercise sheets, and programming languages and environments. Fourth, detailed lesson plans for each of the lessons are provided. Finally, the appendix of the course manual contains all

central learning materials, in particular game instructions, programming activity sheets, end-of-lesson exercise sheets, homework sheets, and resources for further computing education.

4 Discussion and Results

In the following Section 4.1 and its subsections, I will outline and discuss the main findings of each of the four studies included in this dissertation. Then, I will present an integrated discussion (Section 4.2) of these studies, including their strengths, weaknesses, and finally their combined contribution to the field of research on early computing education.

4.1 Main outcomes

In Study 1, we evaluated lessons from a pre-existing computational thinking course to explore the suitability of unplugged game-based activities for helping elementary school students form an initial understanding of computational thinking concepts and for motivating them to want to learn more about computing (Manuscript 1, Leifheit et al., 2018).

The goal of Studies 2 and 3 was the incremental development and evaluation of a new instrument for measuring students' self-concept and motivational beliefs regarding programming (Manuscript 2 and Manuscript 3, Leifheit et al., 2019, 2020).

Finally, for Study 4, we considered results from Study 1 and used the assessment instrument proposed in Studies 2 and 3 to evaluate the effectiveness of a structured computational thinking training for promoting a positive programming-related self-concept and motivation for programming (Manuscript 4, Leifheit et al., in preparation).

Findings from each of these studies are detailed in the following sections.

4.1.1 Study 1: Programming Unplugged: An Evaluation of Game-Based Methods for Teaching Computational Thinking in Primary School

The findings summarized in this section are based on previously published work (Leifheit et al., 2018). The original publication is included in the Appendix as Manuscript 1.

Research goals

Our goal for this exploratory study was to assess the suitability of unplugged game-based activities for helping elementary school students form an initial understanding of computational thinking concepts. Additionally, we wanted to find out whether students who participated in the course were motivated to learn more about computing. To this end, we evaluated pre- and post-course questionnaires as well as the outcome of three lessons that used unplugged game-based activities as their core component. Each of the evaluated lessons introduced one computational thinking concept using an unplugged game-based activity. Specifically, (1) debugging was introduced through a relay race game, (2) conditional branching was introduced through a card game, and (3) events were introduced through a sports game (Leifheit et al., 2018).

Study context

The lessons were part of a computational thinking course consisting of 18 lessons with 45 minutes each, which was based on course material from code.org (Course 2, code.org). The main objectives of the course were to promote students' conceptual understanding of computational concepts and spark their interest in computing. Lessons primarily addressed concepts and processes essential for creating simple algorithms. These concepts were first introduced through unplugged game-based activities using tangible everyday objects (e.g., paper, pencils, or playing cards) instead of digital environments and abstract code. In more advanced lessons, students' conceptual understanding was applied and deepened through simple plugged-in programming exercises.

The three lessons evaluated in Manuscript 1 (Leifheit et al., 2018) used a game-based learning approach to increase students' engagement in the learning activity (Plass et al., 2015). The unplugged learning activities used in the course have been argued to be especially well-suited for introducing elementary school students to

computing, because they allow for placing focus on computational concepts rather than technology use (Prottsman, 2014).

Participants and study design

The course was taught to 33 students (15 female) from 3rd and 4th grade of German elementary school. At the end of all unplugged lessons, students' understanding of newly introduced computational thinking concepts was assessed using short tests in the form of pen-and-paper exercise sheets. Students' interest in and motivation for programming education was measured with pre- and post-course self-assessment questionnaires.

Results and discussion

At the end of the lessons on debugging and events, students were able to mostly solve the short tests correctly. In particular, we observed that (a) after playing the debugging relay race game, students on average solved 89% of the end-of-lesson assessment on debugging correctly, and (b) after playing the events sports game, students on average solved 90% of the end-of-lesson assessment on events correctly. This suggests that the unplugged game-based learning activities employed in these two lessons were suitable for helping students develop an initial understanding of computational thinking concepts.

However, we found that after playing the conditional branching card game, students on average only solved 6% of the end-of-lesson assessment on conditional branching correctly. None of the students were able to solve all tasks of the assessment correctly. These results conflict with our qualitative observations from the conditional branching lesson, in which we noted that students were well able to understand the rules and play the card game, for which they had to apply conditional branching. To better understand this inconsistency between our observations, we took a closer look at the end-of-lesson assessment on conditional branching. We found that students' problems in solving the tasks might have arisen from the somewhat convoluted structure of the assessement: Students were largely able to solve the individual tasks, but were unable to tally them correctly to keep track of the score (Leifheit et al., 2018). The problems students faced in the end-of-lesson assessment on conditional branching might be related to programming novices' difficulties to understand 'else' cases (Guzdial, 2008) and to trace code

linearly (Kaczmarczyk, Petrick, East, & Herman, 2010). This is in line with literature arguing that conditional branching is an especially complex concept for learners new to computing (e.g., Guzdial, 2008).

In addition to the end-of-lesson assessments, students filled out a pre- and post-course questionnaire and responded to each item on a 5-point Likert scale (from 1 = not at all, to 5 = absolutely, with only the end points labeled). On the post-course questionnaire, students reported they liked the course overall ($M = 4.36;\ SD = 0.89$) and indicated that they felt they had made substantial learning progress($M = 4.39;\ SD = 0.98$). For interest in learning more about computing, there was a wider variance between students' responses, but on average they reported moderate interest in future computing education ($M = 3.36; SD = 1.34$).

In both the pre- and the post-course questionnaires, students were asked whether they could imagine working in an IT job. Before taking the computational thinking course, students rated their interest in such a job higher ($M = 3.27;\ SD = 1.39$) than they did after completing the course ($M = 2.82; (\ SD = 1.27$). However, only 9% of students reported previous experience in programming. Participation in the course provided the students with more experience to be able to judge whether IT jobs might be interesting for them. Thus, the descriptive mean decrease could reflect that students developed less strong and less unsubstantiated opinions (i.e., extreme ratings on the Likert scale) than before and were able to build a more differentiated and informed idea of their vocational interest.

Limitations

It is important to note the study's limitations. Due to the study design, no inferential analyses were possible. In other words, our statistical analyses of the end-of-lesson assessments as well as the pre- and post-course questionnaires were purely descriptive. Therefore, no causal conclusions can be drawn about the influence of the learning activities on students' learning outcomes nor the relationship between the course and the development of students' interest in computation.

Conclusion and outlook

In conclusion, 3rd and 4th grade students were able to understand debugging and events, to which they were introduced by playing unplugged games. This indicates that the lessons and their central activities were suitable for helping students un-

derstand and apply the respective lesson's central concepts. However, parts of the course appeared to be more challenging for the students. In particular, students had problems solving the end-of-lesson assessment on conditional branching. This finding is of particular interest for the development of future programming courses for elementary school students. Furthermore, results of the post-course questionnaire reflected that students had a positive learning experience and were interested in learning more about computer science.

The results of this exploratory study – especially the findings on conditional branching as a challenging concept and the suitability of unplugged game-based learning activities for teaching computational thinking – have proven helpful in informing the planning and design of a new computational thinking training (see Manuscript 5, Leifheit et al., n.d.). Furthermore, to investigate motivation and learning outcomes beyond the purely descriptive results of this study, we developed a new instrument for assessing motivational dispositions in elementary school students (see Manuscript 2 and Manuscript 3, Leifheit et al., 2019, 2020) and designed an effectiveness study to allow for drawing causal conclusions on the development of students' motivation and ability (see Manuscript 4, Leifheit et al., in preparation).

4.1.2 Study 2: Development of a Questionnaire on Self-Concept, Motivational Beliefs, and Attitude Towards Programming

The findings summarized in this section are based on previously published work (Leifheit et al., 2019). The original publication is included in the Appendix as Manuscript 2.

Research goals

In this pilot study, we aimed to gather preliminary evidence on the validity of an instrument for assessing programming-related self-concept and motivation in elementary school students. We had developed the proposed instrument based on an existing instrument for measuring students' self-concept and motivational beliefs regarding mathematics (Brisson et al., 2017; Gaspard, 2017; Leifheit et al., 2019), which had already been adapted and validated for other school subjects (Gaspard et al., 2017). To gather first evidence regarding its validity, we examined whether the intercorrelations of the subscales of the proposed instrument were

similar to those of the instrument it was based on.

Prior to developing the instrument, we had identified a lack of instruments for assessing computing-related self-concept and motivation in younger students, especially at the elementary level (Leifheit et al., 2019).

The proposed instrument assesses aspects of students' self-concept and motivational disposition regarding programming on seven scales: (1) self-reported programming experience and understanding, (2) self-concept, (3) intrinsic value belief, (4) attainment value belief, (5) utility value belief, (6) cost belief, and (7) compliance and persistence (Leifheit et al., 2019) (see Appendix, Manuscript 2). Definitions of these constructs were introduced in Section 2.3.3 Assessment.

Participants and study design

We evaluated data from 31 students in 3rd and 4th grade of elementary school who participated in a computational thinking course within an extracurricular enrichment program. Students were nominated to participate in the program by their teachers based on interest and achievement in school. After being nominated, students choose which course offered within the enrichment program they want to attend.

One week before the start of the course, all participating students were assessed in a pretest and one week after the end of the course, they were assessed in a posttest. Both test sessions included the same measurement instruments.

We collected data using the proposed instrument for assessing programming-related self-concept and motivational dispositions. In addition, we assessed students' computational thinking ability using the Computational Thinking test (CTt, Román-González, 2015) as well as selected aspects of students' socioeconomic background and frequency of computer or tablet use with the TIMSS 2015 Context Questionnaire (Wendt et al., 2016).

We focused on the pretest measurements to evaluate students' responses to the proposed instrument before their self-concept and motivation could be affected by the course. To identify potential effects of these constructs on computational thinking ability, we also considered students' performance on the Computational Thinking test at pretest and posttest and calculated the change score.

Results and discussion

In line with our expectations, the results of our correlation analysis indicated programming-related self-concept to be significantly correlated with other subscales, in particular with intrinsic value belief, utility value belief, cost belief, and self-reported compliance and persistence. Considering how constitutive self-concept is for motivation to do well in a subject (Oliver & Simpson, 1988), this is not surprising.

Intrinsic value belief, attainment value belief, utility value belief, and cost belief all correlate significantly with each other as well as with other variables, specifically with self-concept and compliance and persistence. This reflects that these four motivational value beliefs are integral to the instrument.

Overall, the correlation patterns of the subscales for self-concept and motivational beliefs regarding programming mirrored the patterns observed for the same motivational dispositions regarding mathematics. This reflects that the constructs measured by the proposed instrument for assessing programming-related self-concept and motivational beliefs have similar associations with each other as the constructs assessed by the validated instrument for mathematics. This gives a first indication of the validity of the proposed instrument.

Aside from correlations between the subscales of the instrument, we found a significant correlation between self-reported previous experience and understanding of programming and frequency of computer or tablet use. This could indicate that students might assume computer or tablet use to be closely associated with programming skills. However, this could also indicate that students with previous programming experience also use computers or tablets more frequently than those without such experience. As the results are correlational, we cannot draw a causal conclusion.

Contrary to what we expected, the constructs measured by the proposed instruments showed no correlations with computational thinking abilities as assessed by the Computational Thinking test. We had based our expectations on findings that for other subjects, self-concept and motivational beliefs were associated with achievement in the respective subject (e.g., Oliver & Simpson, 1988; Wilkins, 2004). However, this lack of significant correlation could be the result of power limitations due to the small sample size. Typically, correlations between motivational dispositions and achievement were found in studies with much larger sample sizes (e.g., Gaspard, 2017).

Limitations

All students who participated in this pilot study had chosen to participate in the computational thinking course voluntarily. Moreover, the study was conducted in the context of an extracurricular enrichment program for which students were nominated by their teachers based on interest and achievement in school. Thus, results cannot necessarily be generalized for a broader population and may contain a bias toward students with above-average motivation and school achievement.

Conclusion and outlook

Taking into account the important role of self-concept and motivation for educational outcomes, it would be desirable to reliably assess these constructs for programming. A valid and reliable assessment instrument would facilitate evaluating the effects of computing education efforts on important student characteristics that could influence their educational trajectories and success.

We observed that the relationships between the subscales of the proposed instrument show similar patterns to the instrument it is modeled on. This revealed preliminary evidence on the validity of the developed questionnaire, encouraging a follow-up study to evaluate the instrument's reliability and construct validity with a larger sample of students.

After publication of the manuscript in which we reported the development and evaluation of the proposed instrument (Manuscript 2, Leifheit et al., 2019), we named the instrument the Self-Concept and Attitude toward Programming Assessment (SCAPA) and took further steps toward its validation (see Manuscript 2, Leifheit et al., 2020). We later used the instrument to assess the effects of a computational thinking training on elementary school students' programming-related self-concept and motivation (see Manuscript 4, Leifheit et al., in preparation).

4.1.3 Study 3: SCAPA: Development of a Questionnaire Assessing Self-Concept and Attitudes Toward Programming

The findings summarized in this section are based on previously published work (Leifheit et al., 2020). The original publication is included in the Appendix as Manuscript 3.

Research goals

Following up on Study 2 (Manuscript 2, Leifheit et al., 2019), our goal for this study was to provide first evidence regarding the internal consistency and construct validity of the proposed Self-Concept and Attitude toward Programming Assessment (SCAPA). To this end, we examined the instrument's reliability (i.e., internal consistency on item and scale level) and construct validity (by means of confirmatory factor analysis).

SCAPA measures aspects of students' self-concept and motivational disposition regarding programming on seven scales: (1) self-reported programming experience and understanding, (2) self-concept, (3) intrinsic value belief, (4) attainment value belief, (5) utility value belief, (6) cost belief, and (7) compliance and persistence (see Manuscript 3, Leifheit et al., 2020).

Participants and study design

We collected and analyzed data from 197 elementary school students from 3rd and 4th grade of German elementary school who participated in the present study and the computational thinking training within the context of an extracurricular enrichment program. They had been nominated to attend the program by their teachers based on interest and school achievement, but made their own decision which courses they wanted to attend.

One week before the start of the computational thinking training, students attended a test session in their regular course classrooms and were assessed using SCAPA in the form of a pen-and-paper questionnaire.

Results and discussion

Overall, our results indicated adequate internal consistency for SCAPA. The scales of the instrument were shown to have sufficient to very good reliability, the only exceptions being the scale for self-reported programming experience and understanding and the subscale for programming persistence.

In addition, results of the confirmatory factor analysis mostly indicate a good fit between the models for the scales and the actual data. We had calculated the four model fit indices chi-squared test, RMSEA, SRMR, and CFI. While chi-squared test, SRMR, and CFI indicated adequate to very good model fit, RMSEA indicated the models for some of the scales to fit less well. It must be noted,

however, that RMSEA is known to yield inappropriately high values (indicating low fit) for models with smaller degrees of freedom and sample sizes up to N = 200 (e.g., Taasoobshirazi & Wang, 2016), as is the case for our data set. For such models, RMSEA should be considered with caution (Taasoobshirazi & Wang, 2016). Therefore, we place more emphasis on the other measures of model fit, which indicate adequate to very good construct validity.

A detailed account of all analyses and results for internal consistency on item and scale level, for construct validity, and for factor loadings of the individual items is reported in Manuscript 3 (Leifheit et al., 2020).

Limitations

It should be pointed out that the results found in this study cannot necessarily be generalized for all elementary school students. All participants in the present study were in 3rd and 4th grade of German elementary school and attended an extracurricular enrichment program after they had been nominated by their teachers based on their interest and school achievements. Thus, findings may not be representative for the overall student population but potentially only for a sample of better performing students.

Conclusion and outlook

Based on the promising findings of Study 2 (see Section 3.2.2, in which we had examined the pattern of intercorrelations between its scales to gather initial indications of the instrument's validity Manuscript 2, Leifheit et al., 2019), the present study takes a further step toward the validation of the proposed instrument for assessing students' self-concept and attitude toward programming. Results for internal consistency and construct validity of the scales provided further evidence for SCAPA's reliability and validity.

While our findings regarding SCAPA's reliability and construct validity are promising, further development and evaluation is necessary to substantiate the suitability of SCAPA. Future evaluations of SCAPA should aim for a broader, more representative sample to improve generalizability. Additionally, future evaluation studies should aim for larger sample sizes to increase power and overcome limitations as experienced with the RMSEA model fit index for some of the scales.

After the sufficient to very good results for reliability and construct validity

established SCAPA as a suitable instrument for assessing self-concept and motivational disposition regarding programming, we used the SCAPA questionnaire to assess the effectiveness of a computational thinking training for promoting programming-related self-concept and motivation in elementary school students (Manuscript 4, Leifheit et al., in preparation).

4.1.4 Study 4: Development and Evaluation of a "Computational Thinking" Course for Elementary School Students

The findings summarized in this section are based on a manuscript not yet submitted for publication (Leifheit et al., in preparation). The manuscript is included in the Appendix as Manuscript 4.

Research goals

Our goal for this study was to evaluate the effectiveness of a structured computational thinking training for promoting a positive programming-related self-concept and motivation for programming.

The computational thinking training was aimed at 3rd and 4th grade elementary school students and followed the conceptually-oriented didactic approach of computational thinking. Thus, the focus of the training was not on specific programming languages or technologies, but on understanding the following essential computational thinking concepts: sequences, loops, conditionals, events, variables, constants, and operators (as identified by Brennan & Resnick, 2012). The didactic and pedagogical approach of the training followed principles of scaffolding, embodied learning, game-based learning, and blended learning (unplugged/plugged-in).A detailed description of the computational thinking training and its instructional design is given in Section 3.2.4 Study 4 and in the instructor's manual for the computational thinking training (Manuscript 5, Leifheit et al., n.d.).

We expected the computational thinking training to have a positive effect on the development of students' programming-related self-concept and motivational dispositions. Further, we aimed to explore whether students' gender had any differential effect on their self-concept and motivation for programming.

To go beyond motivational dispositions, we also assessed computational thinking ability and cognitive abilities to estimate the effects of the computational thinking training on computational thinking ability and reported the results separately

(see Tsarava et al., in preparation).

Participants and study design

Our participants for this study were 197 elementary school students from 3rd and 4th grade of elementary school who participated in an extracurricular enrichment program. Students are nominated for the program by their teachers based on interest an school achievement, and can then choose from a range of courses. In total, 29 course groups participated in the computational thinking training at 16 local sites of the enrichment program. At each site, both the training and data collection took place in the same regular classroom. The course groups were taught by different course instructors, none of whom worked in research or were involved in the development of the course. Instead, they had received a detailed course manual and a one-day preparation seminar along with all teaching materials for the course.

To allow for estimating the causal effect of the training (Torgerson & Torgerson, 2008), we used a randomized controlled field trial design with repeated measures and a waiting control group. To analyze the data we collected, we calculated multiple linear regressions with maximum likelihood robust estimation to estimate the training and gender effects on seven outcome variables for different aspects of motivational dispositions. The study design and analysis procedure yield conservative estimates to minimize the risk of finding false positives (Dong & Peng, 2013). Outcome variables were motivational dispositions toward programming as assessed by the seven scales (P1) experience and understanding, (P2) self-concept, (P3) intrinsic value belief, (P4) attainment value belief, (P5) utility value belief, (P6) cost belief, and (P7) compliance and persistence.

Results and discussion

On the one hand, there is an increasing number of initiatives for teaching computational thinking at the elementary school level (e.g., Code.org; of Code; Kodable; Balanskat and Engelhardt, 2015). On the other, the effectiveness of these efforts is rarely evaluated empirically – among other reasons due to lack of funding and appropriate assessment instruments (Vivian et al., 2020). To contribute toward closing this gap in research, we developed a computational thinking training for elementary school children (Manuscript 5, Leifheit et al., n.d.) and explored to

what extent the training could foster positive motivational dispositions toward programming.

Overall, results indicated that the computational thinking training had a significant positive effect for three out of the seven motivational constructs. Students who received the computational thinking training, compared to the control group, displayed a more positive development of (P1) experience and understanding of programming ($B = .373;\ p < .001$), in (P2) programming-related self-concept ($B = .167;\ p = .025$), and (P3) intrinsic value belief about programming ($B = .197;\ p = .003$). All estimated effect sizes fell into the small to medium range (Cohen, 1992). No significant effects were found for (P4) attainment value belief, (P5) utility value belief, (P6) cost belief, and (P7) compliance and persistence with regard to programming.

Considering the research design, method of analysis, and sample of the study, we need to take several factors contributing to small training effect sizes into account.

To evaluate the effectiveness of the training under real-world conditions, the computational thinking course was not taught by researchers in standardized instructional settings, but by different course instructors in normal classrooms. All instructors had received a detailed course manual and a seminar on how to teach the course, but had not been involved in the course's development or research. Such real-world settings and instructor variability lead to smaller treatment effects compared to standardized instruction under controlled conditions (e.g., Greene, 2015; Hulleman & Cordray, 2009; Lendrum & Wigelsworth, 2013).

Moreover, all participants had been nominated for an extracurricular enrichment program based on interest and achievement in school and had chosen the computational thinking course over other courses offered within the program. It is therefore possible that the students participating in our study had above-average levels of motivation at pretest time, which may have resulted in ceiling effects at pretest and, consequently, smaller comparative training effects on the posttest variables (Resch & Isenberg, 2018).

Finally, multiple linear regression analyses with maximum-likelihood robust estimation minimize the risk of finding false positives and yield conservative estimates (Dong & Peng, 2013). As a result, any estimated training effects can be interpreted as relevant outcomes of the training and are not likely to be estimated too large.

In light of these factors reducing the estimated size of training effects, we can conclude that the computational thinking training was successful in fostering programming-related self-concept, and motivational beliefs about programming in elementary school students. Because there are no similar computing, programming, or computational thinking training studies reporting effect sizes, we cannot compare the observed effect sizes to similar studies.

Aside from the training effects, we also investigated the role of gender for the development of students' motivational dispositions. Our findings indicated that a student's gender does not affect the influence of the training on motivational beliefs about programming. Specifically, results revealed no significant effects of gender for (P1) experience and understanding, (P3) intrinsic value belief, (P4) attainment value belief, (P5) utility value belief, (P6) cost belief, or (P7) compliance and persistence with regard to programming. At the same time, results showed that after the computational thinking training, boys' programming-related self-concept improved significantly more than that of girls. Considering that only 24% of the participating students were female, this is in line with research reporting that a smaller ratio of girls within a class for gifted students negatively impacts girls' academic self-concept (Preckel, Zeidner, Goetz, & Schleyer, 2008). It must be noted, however, that our study design does not allow for the estimated gender effecs to be interpreted causally.

Limitations

Despite the promising findings of this study, it is important to bear in mind its limitations and point out areas in need of further research.

We evaluated the effectiveness of the computational thinking training for 3rd and 4th grade students within an extracurricular enrichment program for talented children. While it is recommended to focus on a clearly defined target group when first investigating the effectiveness of a training (Gottfredson et al., 2015), such narrow samples still limit the generalizability of the findings. Therefore, future studies should aim to adapt and evaluate the training for a broader sample.

Another limitation results from evaluating any training consisting of multiple components. Because the proposed computational thinking course is a multi-component training, the present study cannot determine which specific components of the training were effective in particular. Thus, follow-up studies should focus on assessing the effectiveness of the individual components of the training

in a more fine-grained research design.

Lastly, in evaluating the effectiveness of the computational thinking training, our focus was on the immediate training effects. Consequently, the employed research design does not permit to predict longitudinal effects. To address this, any further evaluation studies on the training should include additional measurement timepoints after the end of the computational thinking course to provide insight into potential long-term effects of the training.

Conclusion and outlook

Our findings revealed that the training components and instructional design principles applied in the proposed multi-component computational thinking course were generally successful in reaching our goal to promote 3rd and 4th grade students' programming-related self-concept and motivation for programming. Specifically, the training had significant positive effects on the development of students' self-reported experience and understanding of programming, programming-related self-concept, and intrinsic motivation for programming.

In addition to being effective for fostering positive motivational dispositions toward programming, we found the computational thinking training to have significant positive effects on students' computational thinking ability, revealing that the proposed training was not only motivating, but also beneficial for students' learning outcomes in the domain of computing. We reported the training effects on computational thinking ability and cognitive abilities separately (for a detailed description of the analysis and results for cognitive abilities and computational thinking ability, see Tsarava et al., in preparation).

With regard to gender, we found that for boys, the training had a significantly greater positive effect on the programming-related self-concept than it did for girls. It must be stressed that our study design does not allow for this effect to be interpreted causally. In light of the smaller ratio of girls who participated in the computational thinking training, our finding is in line with research reporting that academic self-concept of girls is negatively affected by a smaller ratio of girls within classes for gifted students (Preckel et al., 2008).

In conclusion, the present study provided initial evidence for the effectiveness of the proposed multi-component computational thinking course for improving students' computational thinking abilities and promoting positive self-concept and motivational beliefs with regard to programming. Additionally, aside from gender-

ratio effects unrelated to the proposed training, our results revealed that girls and boys benefit from the computational thinking course in equal measure.

4.2 Integrated discussion

The studies included in this dissertation followed two different but interconnected lines of research. On the one hand, my goal was to investigate how students' motivational dispositions toward programming can be assessed reliably. On the other, I aimed to explore how computational thinking can be taught in a way that is motivating to elementary school students.

Findings of Study 1 reflected that unplugged game-based teaching methods were generally suitable for teaching computational thinking in elementary school. After attending a blended unplugged and plugged-in computational thinking course, students felt motivated to learn more about computing. The results also indicated that conditional branching, especially nested conditionals, was a particularly challenging computational concept to grasp for elementary school students (Manuscript 1, Leifheit et al., 2018). These findings were considered for the development of the new computational thinking training evaluated in Study 4. Informed by Study 1, a step-wise, scaffolded approach was chosen to introduce and further explore conditional branching over the course of several lessons to meet the complexity of the concept (Manuscript 5, Leifheit et al., n.d.).

After identifying a lack of research instruments for assessing programming-related motivational constructs, I designed the Self-Concept and Attitude toward Programming Assessment (SCAPA) and put it to the test in a pilot evaluation study (Manuscript 2, Leifheit et al., 2019). In a follow-up study with a larger sample, SCAPA was examined for internal consistency and construct validity. Findings revealed high internal consistency for most of the scales and sufficient to very good construct validity, establishing SCAPA as a suitable instrument for assessing self-concept and motivational disposition regarding programming (Manuscript 3, Leifheit et al., 2020).

Study 4 builds on findings from all three of the previous studies and reconnects the two lines of research in this dissertation. I co-developed a computational thinking training and evaluated its effectiveness for fostering a positive programming-related self-concept and motivation for programming (Manuscript 4, Leifheit et al., in preparation). The effect of the training on these motivational constructs

was assessed using SCAPA. Because the research design of Study 1 only permitted descriptive analyses and thus limited the possibilities for interpretation, Study 4 applied a robust research design to allow for inferential analysis and causal interpretation of the training effects.

In addition to having a positive effect on students' learning outcomes in computational thinking, the training effectively fostered positive self-concept and motivational beliefs with regard to programming. Aside from gender-ratio effects unrelated to the training itself, results revealed that girls and boys benefited from the computational thinking training in equal measure.

4.2.1 Strengths and contribution

One of the key strengths of this dissertation is the evidence-oriented empirical research approach followed by all of the included studies. Reporting empirical evidence allows other researchers and educators to make informed decisions based on what has been found to work when tested practically and systematically.

In line with this approach, not only the included studies, but also the proposed computational thinking training was designed based on findings from education sciences, psychology of learning, and computing education research.

Results presented in this dissertation, particularly in Studies 2, 3, and 4, were calculated using state-of-the-art statistical analysis procedures. All applied missing data treatment was selected to ensure high accuracy and was specifically recommended for education research (e.g., Buhi et al., 2008; Cheema, 2014; Schafer & Graham, 2002).

The research design used in Study 4 – a field trial with randomized group assignment and a waiting control group – allows for drawing causal inference on the effectiveness of the computational thinking training (e.g., Friedman et al., 2010; Torgerson & Torgerson, 2008). For this reason, the reported effect sizes can serve as a benchmark with which future effectiveness research in the field of early computing education can be compared.

4.2.1.1 Scientific contribution

To facilitate further research on self-concept and motivation in early computing education, I designed an assessment instrument for measuring self-concept and motivation with regard to programming and incrementally worked toward vali-

dating the instrument in two studies. Results revealed that the instrument is a promising tool with sufficient to very good internal consistency and construct validity, which can be used in future motivation research or evaluation studies for assessing students' programming-related self-concept and motivation. Further, applying the instrument to evaluate the success of computing education efforts could help teachers and educational policymakers make informed evidence-based decisions regarding the development or selection of computing education curricula and teaching materials.

Additionally, I wanted to contribute to the evidence on what is effective for teaching computational thinking in a way that is motivating to elementary school students. For this purpose, I analyzed the effectiveness of a multi-component computational thinking course for promoting students' programming-related self-concept and motivational dispositions toward programming. The study followed a randomized controlled field trial design with repeated measures and a waiting control group to permit causal interpretation of the results. The computational thinking training was found to be effective not only for teaching computational thinking, but also for fostering programming-related motivation and self-concept. These findings can provide orientation for future development and evaluation of teaching resources while the estimated effect sizes for the training can serve as a benchmark against which future studies on early computing education can be measured.

4.2.1.2 Educational contribution

The computational thinking course proposed and evaluated in this dissertation is being offered continuously in most of the 66 local sites of the Hector Children's Academy Program in the German federal state of Baden-Württemberg. Thus far, over 100 instructors attended instructional seminars in which they were trained to teach the computational thinking course and subsequently qualified as official course instructors. From 2018 to 2020, already several hundred students were able to benefit from participating in an empirically evaluated course that has been established to be effective for fostering computational thinking as well as programming-related self-concept and motivation.

Aside from students participating in the course, another educational contribution of the evaluated computational thinking training is that the availability of evaluated teaching resources can help educators design new resources or inte-

grate the evaluated resources into their own lessons and curricula.

Furthermore, the computational thinking course inspired an online programming course in which over 1000 students participated while schools were closed in Germany during the COVID-19 pandemic of 2020.

4.2.2 Limitations and future perspective

This dissertation reveals promising findings, in particular the initial evidence on the reliability and validity of the SCAPA instrument and on the effectiveness of a training for fostering computational thinking, self-concept, and motivational beliefs at the elementary school level. Nevertheless, it is important to note its limitations and identify areas in which further research could help to answer the research questions in more detail.

None of the studies included in this dissertation collected data at more than two timepoints. Study 2 examined correlational patterns between the scales and Study 3 evaluated the SCAPA instrument for construct validity and internal consistency on item and scale level. Longitudinal data were not required for either of these goals. However, to also evaluate the instrument for retest reliability and construct stability over time, a research design would be useful in which the instrument is used to assess students at more than two timepoints over a longer period of time. Such a longitudinal design would also allow to explore whether the motivational constructs measured by SCAPA are predictive for future achievement and continued pursuit of computing education, as is the case for other school subjects. Study 4 focused on estimating immediate training effects. As a result, the structure of the collected data does not allow longitudinal effects to be predicted. In future studies, additional measurement timepoints after the completion of the training could provide insight into potential long-term effects of the training.

Another limitation of Study 4 results from evaluating any training consisting of multiple components. Because the proposed computational thinking course is a multi-component training, determining which specific components of the training were effective in particular lies beyond what this dissertation can accomplish. More fine-grained effectiveness studies on individual components and instructional approaches used in the training could provide insight into which of the training components and teaching methods are particularly effective for fostering computational thinking and positive motivational dispositions.

Moreover, finding appropriate instruments for measuring computational think-

ing ability in the target age group proved a challenge. Thus, a shortened version of the Computational Thinking test was used, which is originally aimed at middle school students (Román-González, 2015). In the coming years, I expect more instruments for assessing computational thinking to be made available. Hopefully, future studies regarding the effectiveness of the training and its components will be able to apply a wider array of assessment instruments, especially instruments developed specifically for the target age group.

Lastly, because of partially selective samples, findings presented in this dissertation cannot necessarily be generalized for the entire population of elementary school students. In all four studies, data were collected from students who were in 3rd and 4th grade of German elementary school and had voluntarily decided to attend computational thinking courses. Thus, results may be representative specifically for students already interested in learning about computing. Further, Studies 2, 3, and 4 were conducted in the context of an extracurricular enrichment program for which students were nominated by their teachers based on interest and achievement in school. While it is recommended to focus on a clearly defined target group when first evaluating instruments or investigating the effectiveness of a training (Gottfredson et al., 2015), such narrow samples still limit the generalizability of the findings. For future studies, it would be desirable to have a broader sample to fully validate SCAPA and to adapt and evaluate the computational thinking training for its general effectiveness.

4.3 Conclusion

With this dissertation, I aimed to find answers to two main interconnected research questions, namely (1) how computing-related motivational dispositions can be assessed reliably in early education and (2) how computational thinking can be taught in a way that is motivating to elementary school students and beneficial for their self-concept.

Being able to reliably assess student characteristics and to determine the effectiveness of didactic approaches and teaching methods within early computing education would help teachers and educational policymakers make informed evidence-based decisions regarding the design and implementation of computing courses or curricula for younger students. Likewise, such an assessment instrument and evidence on effective teaching methods can serve as tools or provide

orientation for future computing education research.

Thus, I designed the SCAPA instrument for measuring students' programming-related motivation and self-concept and conducted studies aiming at the instrument's incremental validation. Results revealed sufficient to very good results for reliability and construct validity of SCAPA, establishing the instrument as suitable for assessing self-concept and motivational disposition regarding programming.

Further, to gather evidence on how computational thinking can be taught in a way that is motivating to elementary school students, I co-developed a structured computational thinking course and evaluated its effectiveness for promoting students' self-concept and motivational dispositions with regard to programming. Findings indicated that the training components, didactic approach, and instructional design of the computational thinking course were generally successful in promoting students' self-concept and motivation for programming. In addition, the course was also effective for improving students' computational thinking ability. The study design and analysis procedures allowed for drawing causal inferences about the effectiveness of the computational thinking training. For this reason, the reported effect sizes can serve as a benchmark future reports of educational effectiveness in early computing education.

The substantial and original contribution of this dissertation lies in three accomplishments, specifically in (1) proposing an evaluated novel instrument for assessing computing-related motivational constructs, (2) presenting an empirical evaluation study using a research design which is unique in the field of early computing education in that it permits drawing causal inference, and (3) providing robust evidence that the proposed computational thinking training not only improves computational thinking in elementary school students, but is also motivating and beneficial for their self-concept.

I expect these contributions to be of value for the evidence-oriented development of new computing education materials and curricula, for facilitating future research on computing-related motivation, and for introducing state-of-the-art empirical study designs in early computing education research.

References

Armoni, M., & Gal-Ezer, J. (2014). Early computing education: Why? What? When? Who? *ACM Inroads*, *5*(4), 54–59.

Armstrong, J. M., & Price, R. A. (1982). Correlates and predictors of women's mathematics participation. *Journal for Research in Mathematics Education*, *13*(2), 99.

Badamasi, Y. A. (2014). The working principle of an Arduino. In *11th International Conference on Electronics Computer and Computation (ICECCO)* (pp. 1–4).

Balanskat, A., & Engelhardt, K. (2015). Computing our future. Computer programming and coding. Priorities, school curricula and initiatives across Europe. *European Schoolnet, Brussels*.

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to k-12: What is involved and what is the role of the computer science education community? *ACM Inroads*, *2*(1), 48–54.

Barsalou, L. W. (2008). Grounded cognition. *Annual Review of Psychology*, *59*, 617–645.

Bebras. (n.d.). *Bebras: International challenge on informatics and computational thinking*. Retrieved from `https://www.bebras.org/`

Belland, B. R. (2014). Scaffolding: Definition, current debates, and future directions. In *Handbook of Research on Educational Communications and Technology* (pp. 505–518). Springer.

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 Annual Meeting of the American Educational Research Association, Vancouver, Canada* (Vol. 1, p. 25).

Brisson, B. M., Dicke, A.-L., Gaspard, H., Häfner, I., Flunger, B., Nagengast, B., & Trautwein, U. (2017). Short intervention, sustained effects: Promoting students' math competence beliefs, effort, and achievement. *American Educational Research Journal*, *54*(6), 1048–1078.

Buhi, E. R., Goodson, P., & Neilands, T. B. (2008). Out of sight, not out of mind:

Strategies for handling missing data. *American Journal of Health Behavior*, *32*(1), 83–92.

Butz, M. V. (2016). Toward a unified sub-symbolic computational theory of cognition. *Frontiers in Psychology*, *7*, 925.

Cheema, J. R. (2014). Some general guidelines for choosing missing data handling methods in educational research. *Journal of Modern Applied Statistical Methods*, *13*(2), 53–75.

Chouinard, R., Karsenti, T., & Roy, N. (2007). Relations among competence beliefs, utility value, achievement goals, and effort in mathematics. *British Journal of Educational Psychology*, *77*(3), 501–517.

code.org. (n.d.). *Course 2*. https://studio.code.org/s/course2. (Accessed: 16.07.2017)

Code.org. (n.d.). *Cs fundamentals for grades k-5*. Retrieved from `https://code.org/educate/curriculum/elementary-school`

Cohen, J. (1992). A power primer. *Psychological Bulletin*, *112*(1), 155.

Dagienė, V., & Sentance, S. (2016). It's computational thinking! Bebras tasks in the curriculum. In *International Conference on Informatics in Schools: Situation, Evolution, and Perspectives* (pp. 28–39).

del Olmo-Muñoz, J., Cózar-Gutiérrez, R., & González-Calero, J. A. (2020). Computational thinking through unplugged activities in early years of primary education. *Computers & Education*, *150*, 103832.

Denissen, J. J. A., Zarrett, N. R., & Eccles, J. S. (2007). I like to do it, I'm able, and I know I am: Longitudinal couplings between domain-specific achievement, self-concept, and interest. *Child Development*, *78*(2), 430–447.

Denning, P. J. (2017). Remaining trouble spots with computational thinking. *Communications of the ACM*, *60*(6), 33–39.

Diaz, L. M., & Lopez, L. F. A. (2017). A classification of programming styles in Scratch. In *Proceedings of the 8th Latin American Conference on Human-Computer Interaction* (pp. 1–4).

Dong, Y., & Peng, C.-Y. J. (2013). Principled missing data methods for researchers. *SpringerPlus*, *2*(1), 222.

Dorn, B., & Elliott Tew, A. (2015). Empirical validation and application of the computing attitudes survey. *Computer Science Education*, *25*(1), 1–36.

Dredge, S. (2014). *Coding at school: A parent's guide to England's new computing curriculum* (Vol. 4). Retrieved from `https://www.theguardian.com/`

technology/2014/sep/04/coding-school-computing-children-programming

Durik, A. M., Vida, M., & Eccles, J. S. (2006). Task values and ability beliefs as predictors of high school literacy choices: A developmental analysis. *Journal of Educational Psychology*, *98*(2), 382–393.

Eccles, J. S., Alder, T. F., Futterman, R., Goff, S. B., Kaczala, C. M., Meece, J. L., & Midgley, C. (1983). Expectancies, values, and academic behaviors. In *Achievement and Achievement Motivation* (pp. 75–146). Freeman.

Emanuel, E., Abdoler, E., & Stunkel, L. (2010). Research ethics: How to treat people who participate in research. *National Institutes of Health (NIH) Clinical Center Department of Bioethics and Foundation for the NIH*.

Enders, C. K. (2010). *Applied missing data analysis*. Guilford Press.

Friedman, L. M., Furberg, C. D., & DeMets, D. L. (2010). Blindness. In *Fundamentals of Clinical Trials* (pp. 119–132). Springer.

García Peñalvo, F. J., Reimann, D., Tuul, M., Rees, A., Jormanainen, I., et al. (2016). An overview of the most relevant literature on coding and computational thinking with emphasis on the relevant issues for teachers. Retrieved from https://gredos.usal.es/bitstream/handle/10366/131863/TACCLE3O5Literaturereview%20-%20final.pdf?sequence=1

Gaspard, H. (2017). *Promoting value beliefs in mathematics : A multidimensional perspective and the role of gender*.

Gaspard, H., Häfner, I., Parrisius, C., Trautwein, U., & Nagengast, B. (2017). Assessing task values in five subjects during secondary school: Measurement structure and mean level differences across grade level, gender, and academic subject. *Contemporary Educational Psychology*, *48*, 67–84.

Gottfredson, D. C., Cook, T. D., Gardner, F. E., Gorman-Smith, D., Howe, G. W., Sandler, I. N., & Zafft, K. M. (2015). Standards of evidence for efficacy, effectiveness, and scale-up research in prevention science: Next generation. *Prevention Science*, *16*(7), 893–926.

Greene, J. A. (2015). Serious challenges require serious scholarship: Integrating implementation science into the scholarly discourse. *Contemporary Educational Psychology*, *40*, 112–120.

Guzdial, M. (2008). Education paving the way for computational thinking. *Communications of the ACM*, *51*(8), 25–27.

Hamari, J., Koivisto, J., & Sarsa, H. (2014). Does gamification work? – A literature

review of empirical studies on gamification. In *47th Hawaii International Conference on System Sciences* (pp. 3025–3034).

Heersink, D., & Moskal, B. M. (2010). Measuring high school students' attitudes toward computing. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (pp. 446–450). New York, NY, USA: ACM.

Hoegh, A., & Moskal, B. M. (2009). Examining science and engineering students' attitudes toward computer science. In *2009 39th IEEE Frontiers in Education Conference* (pp. 1–6).

Hulleman, C. S., & Cordray, D. S. (2009). Moving from the lab to the field: The role of fidelity and achieved relative intervention strength. *Journal of Research on Educational Effectiveness*, *2*(1), 88–110.

Husman, J., & Hilpert, J. (2007). The intersection of students' perceptions of instrumentality, self-efficacy, and goal orientations in an online mathematics course. *Zeitschrift fur Pädagogische Psychologie*, *21*(3-4), 229–239.

Jost, B., Ketterl, M., Budde, R., & Leimbach, T. (2014). Graphical programming environments for educational robots: Open Roberta – yet another one? In *2014 IEEE International Symposium on Multimedia* (pp. 381–386).

K-12 CS, F. S. C. (2016). *K-12 computer science framework.* ACM. Retrieved from `https://k12cs.org`

Kaczmarczyk, L. C., Petrick, E. R., East, J. P., & Herman, G. L. (2010). Identifying student misconceptions of programming. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (pp. 107–111). ACM.

Kazimoglu, C. (2013). *Empirical evidence that proves a serious game is an educationally effective tool for learning computer programming constructs at the computational thinking level* (Unpublished doctoral dissertation). University of Greenwich.

Knuth, D. E. (1974). Computer science and its relation to mathematics. *The American Mathematical Monthly*, *81*(4), 323–343.

Kodable. (n.d.). *Programming for kids.* Retrieved from `https://www.kodable.com/`

Kumar, D. (2014). *Welcome.* Retrieved from `https://dl.acm.org/doi/10.1145/2684721.2684733`

Lantz, A. E., & Smith, G. P. (1981). Factors influencing the choice of nonrequired mathematics courses. *Journal of Educational Psychology*, *73*(6), 825–837.

Leifheit, L., Jabs, J., Ninaus, M., Moeller, K., & Ostermann, K. (2018). Program-

ming unplugged: An evaluation of game-based methods for teaching computational thinking in primary school. In *Proceedings of the 12th European Conference on Game-Based Learning (ECGBL)* (pp. 344–353).

Leifheit, L., Tsarava, K., Golle, J., Trautwein, U., Ostermann, K., Ninaus, M., & Moeller, K. (in preparation). Motivational effects of an extracurricular computational thinking training for elementary school children: A randomized controlled field trial..

Leifheit, L., Tsarava, K., Moeller, K., Ostermann, K., Golle, J., Trautwein, U., & Ninaus, M. (2019). Development of a questionnaire on self-concept, motivational beliefs, and attitude towards programming. In *Proceedings of the 14th Workshop in Primary and Secondary Computing Education (WiPSCE). ACM.* (pp. 1–9). doi:http://dx.doi.org/10.1145/3361721.3361730.

Leifheit, L., Tsarava, K., Ninaus, M., & Moeller, K. (n.d.). *"Verstehen wie Computer denken". Ein Trainingsprogramm zur Förderung von informatischem Denken und systematischen Problemlösefähigkeiten besonders begabter Kinder im Grundschulalter. Reihe Hector Core Courses.* Available from the first author.

Leifheit, L., Tsarava, K., Ninaus, M., Ostermann, K., Golle, J., Trautwein, U., & Moeller, K. (2020). SCAPA: Development of a questionnaire assessing self-concept and attitudes toward programming. In *Proceedings of the 25th Conference on Innovation and Technology in Computer Science Education (ITiCSE). ACM.* (pp. 138–144). doi:http://dx.doi.org/10.1145/3341525.3387415.

Lendrum, A., & Wigelsworth, M. (2013). The evaluation of school-based social and emotional learning interventions: Current issues and future directions. *Psychology of Education Review*, *37*, 70–76.

Lister, R. (2016). Toward a developmental epistemology of computer programming. In *Proceedings of the 11th Workshop in Primary and Secondary Computing Education* (pp. 5–16).

Little, R. J. (1988). A test of missing completely at random for multivariate data with missing values. *Journal of the American Statistical Association*, *83*(404), 1198–1202.

Lu, J. J., & Fletcher, G. H. (2009). Thinking about computational thinking. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education* (pp. 260–264).

Marsh, H. W., Köller, O., Trautwein, U., Lüdtke, O., & Baumert, J. (2005). Aca-

demic self-concept, interest, grades, and standardized test scores: Reciprocal effects models of causal ordering. *Child Development*, *76*(2), 397–416.

Muthén, L. K., & Muthén, B. O. (2012). *Mplus version 7 user's guide*.

Nagengast, B., Trautwein, U., Scalas, L. F., Hau, K.-T., Marsh, H. W., & Xu, M. K. (2011). Who took the "×" out of expectancy-value theory? *Psychological Science*, *22*(8), 1058–1066.

Nardelli, E. (2019). Do we really need computational thinking? *Communications of the ACM*, *62*(2), 32–35.

Niepel, C., Stadler, M., & Greiff, S. (2019). Seeing is believing: Gender diversity in stem is related to mathematics self-concept. *Journal of Educational Psychology*.

Ninaus, M., Pereira, G., Stefitz, R., Prada, R., Paiva, A., Neuper, C., & Wood, G. (2015). Game elements improve performance in a working memory training task. *International Journal of Serious Games*, *2*(1), 3–16.

Noice, H., & Noice, T. (2001). Learning dialogue with and without movement. *Memory & Cognition*, *29*(6), 820–827.

of Code, H. (n.d.). *Hour of code: Join the movement.* Retrieved from `https://hourofcode.com/`

Oliver, J. S., & Simpson, R. D. (1988). Influences of attitude toward science, achievement motivation, and science self concept on achievement in science: A longitudinal study. *Science Education*, *72*(2), 143–55.

Papert, S. (1972). Teaching children thinking. *Programmed Learning and Educational Technology*, *9*(5), 245–255.

Papert, S. (1980). *Mindstorms: Computers, children, and powerful ideas*. NY: Basic Books.

Papert, S. (2005). You can't think about thinking without thinking about thinking about something. *Contemporary Issues in Technology and Teacher Education*, *5*(3), 366–367.

Perez, T., Cromley, J. G., & Kaplan, A. (2014). The role of identity development, values, and costs in college STEM retention. *Journal of Educational Psychology*, *106*(1), 315–329.

Piaget, J. (1972). Intellectual evolution from adolescence to adulthood. *Human Development*, *15*(1), 1–12.

Plass, J. L., Homer, B. D., & Kinzer, C. K. (2015). Foundations of game-based learning. *Educational Psychologist*, *50*(4), 258–283.

Preckel, F., Zeidner, M., Goetz, T., & Schleyer, E. J. (2008). Female 'big fish' swimming against the tide: The 'big-fish-little-pond effect'and gender-ratio in special gifted classes. *Contemporary Educational Psychology*, *33*(1), 78–96.

Prottsman, K. (2014). Computer science for the elementary classroom. *ACM Inroads*, *5*(4), 60–63.

Resch, A., & Isenberg, E. (2018). How do test scores at the ceiling affect value-added estimates? *Statistics and Public Policy*, *5*(1), 1–6.

restart: The resurgence of computer science in uk schools. (n.d.).

Reyes, L. H. (1984). Affective variables and mathematics education. *The Elementary School Journal*, *84*(5), 558–581.

Riggs, I. M., & Enochs, L. G. (1993). A microcomputer beliefs inventory for middle school students: Scale development and validation. *Journal of Research on Computing in Education*, *25*(3), 383–390.

Román-González, M. (2015). Computational thinking test: Design guidelines and content validation. In *Proceedings of EDULEARN15 Conference* (pp. 2436–2444).

Román-González, M., Pérez-González, J.-C., & Jiménez-Fernández, C. (2017). Which cognitive abilities underlie computational thinking? criterion validity of the computational thinking test. *Computers in Human Behavior*, *72*, 678–691.

S4A, S. f. A. (n.d.). *Scratch for Arduino (S4A)*. Retrieved from `http://s4a.cat`

Schafer, J. L., & Graham, J. W. (2002). Missing data: Our view of the state of the art. *Psychological Methods*, *7*(2), 147.

Selwyn, N. (1997). Students' attitudes toward computers: Validation of a computer attitude scale for 16–19 education. *Computers & Education*, *28*(1), 35–41.

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, *22*, 142–158.

Taasoobshirazi, G., & Wang, S. (2016). The performance of the SRMR, RMSEA, CFI, and TLI: An examination of sample size, path size, and degrees of freedom. *Journal of Applied Quantitative Methods*, *11*(3), 31–39.

Teague, D. M., Corney, M. W., Fidge, C. J., Roggenkamp, M. G., Ahadi, A., & Lister, R. (2012). Using neo-Piagetian theory, formative in-class tests and think alouds to better understand student thinking: A preliminary report on

computer programming. In *Proceedings of 2012 Australasian Association for Engineering Education (AAEE) Annual Conference*.

Torgerson, D., & Torgerson, C. (2008). *Designing randomised trials in health, education and the social sciences: An introduction*. Springer.

Trautwein, U., & Köller, O. (2005). Was lange währt, wird nicht immer gut. *Zeitschrift für Pädagogische Psychologie*, *17*(3/4), 199–209.

Trautwein, U., & Lüdtke, O. (2009). Predicting homework motivation and homework effort in six school subjects: The role of person and family characteristics, classroom factors, and school track. *Learning and Instruction*, *19*(3), 243–258.

Trautwein, U., Lüdtke, O., Roberts, B. W., Schnyder, I., & Niggli, A. (2009). Different forces, same consequence: Conscientiousness and competence beliefs are independent predictors of academic effort and achievement. *Journal of Personality and Social Psychology*, *97*(6), 1115–1128.

Trautwein, U., Nagengast, B., Marsh, H. W., Gaspard, H., Dicke, A.-L., Lüdtke, O., & Jonkmann, K. (2013). Expectancy-value theory revisited: From expectancy-value theory to expectancy-valueS theory? *Theory driving research: New wave perspectives on self-processes and human development*, 233–249.

Tsarava, K., Leifheit, L., Ninaus, M., Golle, J., Trautwein, U., & Moeller, K. (in preparation). Computational thinking training: Implementation and effects on elementary school children's cognitive and computational thinking skills.

Tsarava, K., Leifheit, L., Ninaus, M., Román-González, M., Butz, M. V., Golle, J., ... Moeller, K. (2019). Cognitive correlates of computational thinking: Evaluation of a blended unplugged/plugged-in course. In *Proceedings of the 14th Workshop in Primary and Secondary Computing Education (WiPSCE)*. *ACM*. (pp. 1–9).

Tsarava, K., Moeller, K., & Ninaus, M. (2018a). Board games for training computational thinking. In *International Conference on Games and Learning Alliance* (pp. 90–100).

Tsarava, K., Moeller, K., & Ninaus, M. (2018b). Training computational thinking through board games: The case of Crabs & Turtles. *International Journal of Serious Games*, *5*(2), 25–44.

Tsarava, K., Moeller, K., Pinkwart, N., Butz, M., Trautwein, U., & Ninaus, M. (2017). Training computational thinking: Game-based unplugged and

plugged-in activities in primary school. In *European Conference on Games Based Learning* (pp. 687–695).

Vivian, R., Franklin, D., Frye, D., Peterfreund, A., Ravitz, J., Sullivan, F., ... McGill, M. M. (2020). Evaluation and assessment needs of computing education in primary grades. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 124–130).

Wang, J., Hejazi Moghadam, S., & Tiffany-Morales, J. (2017). Social perceptions in computer science and implications for diverse students. In *Proceedings of the 2017 ACM Conference on International Computing Education Research - ICER '17* (pp. 47–55). New York, New York, USA: ACM Press.

Wendt, H., Bos, W., Selter, C., Köller, O., Schwippert, K., & Kasper, D. (2016). *TIMMS 2015 - Mathematische und naturwissenschaftliche Kompetenzen von Grundschulkindern in Deutschland im interntionalen Vergleich*.

Wigfield, A., & Cambria, J. (2010). *Students' achievement values, goal orientations, and interest: Definitions, development, and relations to achievement outcomes* (Vol. 30) (No. 1). Academic Press.

Wigfield, A., & Eccles, J. S. (1992, sep). The development of achievement task values: A theoretical analysis. *Developmental Review*, *12*(3), 265–310.

Wilkins, J. L. (2004). Mathematics and science self-concept: An international investigation. *The Journal of Experimental Education*, *72*(4), 331–346.

Wilson, R. A., & Foglia, L. (2015). *Embodied cognition.* Stanford University. Retrieved from `https://plato.stanford.edu/archives/spr2017/entries/embodied-cognition/`

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–35.

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *366*(1881), 3717–3725.

Wing, J. M. (2011). Research notebook: Computational thinking — what and why. *The Link Magazine*, *6*.

Wing, J. M. (2014). *Computational thinking benefits society.* Retrieved from `http://socialissues.cs.toronto.edu/index.html?p=279.html`

Yadav, A., Hong, H., & Stephenson, C. (2016). Computational thinking for all: pedagogical approaches to embedding 21st century problem solving in k-12 classrooms. *TechTrends*, *60*(6), 565–568.

Appendix

Manuscript 1

Title:
Programming Unplugged: An Evaluation of Game-Based Methods for Teaching Computational Thinking in Primary School

Authors:
Luzia Leifheit, Julian Jabs, Manuel Ninaus, Korbinian Moeller, and Klaus Ostermann

Published in:
Proceedings of the 12th European Conference on Game-Based Learning (ECGBL).

Time of publication:
October 2018

Programming Unplugged: An Evaluation of Game-Based Methods for Teaching Computational Thinking in Primary School

Luzia Leifheit[1,2,3], Julian Jabs[2], Manuel Ninaus[3,1], Korbinian Moeller[3,4,1], & Klaus Ostermann[1,2]

[1] LEAD Graduate School and Research Network, University of Tübingen, Tübingen, Germany

[2] Department of Computer Science, University of Tübingen, Tübingen, Germany

[3] Leibniz Institut für Wissensmedien, Tübingen, Germany

[4] Department of Psychology, University of Tübingen, Tübingen, Germany

luzia.leifheit@uni-tuebingen.de

julian.jabs@uni-tuebingen.de

m.ninaus@iwm-tuebingen.de

k.moeller@iwm-tuebingen.de

klaus.ostermann@uni-tuebingen.de

Abstract: Game-based approaches can be a motivating and engaging way of teaching and learning, in particular for younger students. To evaluate the suitability of game-based approaches for teaching programming in primary school, we conducted a field study on Computational Thinking (CT). CT can be characterized as the ability to understand, formulate, and systematically solve complex problems, which typically requires abstraction, generalization, parameterization, algorithmization, and partitioning as processes of CT, which are also vital to programming. The employed CT course focused on fostering students' conceptual understanding of computational thinking independent of specific technological applications and was based on course material from code.org. Lessons primarily addressed algorithms as a core CT concept and used game-based learning material to increase students' understanding of algorithmic CT concepts such as sequences, loops, branches, and events. These concepts were first introduced through unplugged game-based activities using tangible everyday objects (e.g. pencils, playing cards, etc.) instead of abstract code. In more advanced lessons, students' conceptual understanding was applied and deepened through plugged-in programming exercises. The 18 sessions (45 minutes each) of the course were taught to 33 3rd and 4th grade primary school students. At the end of unplugged lessons, students' understanding of newly introduced CT concepts was assessed by short tests. Students' interest in and motivation for programming education was measured with pre- and post-course self-assessment questionnaires. Results indicated benefits of the unplugged game-based approach for teaching CT concepts. In particular, we observed (a) for all but one of the CT concepts, students reached on average 82% of the learning objectives or more, and (b) students rated their learning experience in the course positively and reported high levels of interest in learning more about computation-related topics. In addition, qualitative analyses indicated part of the curriculum was very complex for the target group (e.g. nested conditionals). This finding is of particular interest for the development and evaluation of future programming courses for primary school students.

Keywords: computer science education, programming education, computational thinking, game-based learning

1. Introduction

In recent years, computer programming or coding has repeatedly been argued to be a so-called 21st century skill in our increasingly technological world, which is more and more dominated by algorithms (e.g., Willson, 2017). As a consequence, different authors and institutions suggested computer programming should be part of school curricula (e.g., Papert, 2005; Balanskat & Engelhardt, 2015). However, just teaching children one or the other programming language does not seem desirable, as programming languages develop fast. Instead, it seems more promising to introduce children to what has been termed computational thinking (henceforth CT). CT is characterized as the processes involved in understanding, formulating, and solving complex problems in a way that admits a computational solution, using abstraction, generalization, partitioning, and algorithmization (Wing, 2014). What teaching children – or anyone – to think computationally actually means is not to turn them into computer programmers, but to help them develop their ability to understand complex problems and to cultivate strategies for solving them systematically.

1.1 Computational thinking in primary education: Why and how?

Across Europe, the focus on logical thinking, problem solving, and programming skills in education is increasing as more and more countries are integrating or planning to integrate programming-related skills into school curricula for as early as primary education (e.g., Balanskat & Engelhardt, 2015). This development is in line with the notion that CT is a fundamental skill for the 21st century (Yadav et al., 2014).

Already Papert (1972) proposed letting children write interactive programs can be used for fostering their ability to articulate and simultaneously learn about their own thought processes, considering creating such programs requires children to contemplate and anticipate all possible misunderstandings and mistakes the program's users might encounter or make. On this basis, he argued programming can be used to support the fundamental ability of understanding one's own thought processes, and thus suggested integrating simple elements of computer science, such as programming, into primary education (Papert, 2005).

Primary school seems an appropriate time for introducing students to programming because the concrete operational stage (according to Piaget, 1972) is typically reached between the age of 7 to 11 years. This stage, according to Neo-Piagetian models of cognitive development with regard to learning programming, is "the first stage where students show a purposeful approach to writing code" (Lister, 2016) and "can write small programs from well-defined specifications" (Teague et al., 2012). For early computer science education, Prottsman (2014) suggested a CT approach making use of unplugged exercises (using concrete materials such as pencils, cards, etc.) to assist the progress of learning abstract computational concepts students will encounter when they begin to learn programming (Kumar, 2014).

Accordingly, we used unplugged games in our study to facilitate children's learning. In particular, we incorporated various forms of embodiment, which was argued to facilitate implicit learning (Barsalou, 2008) and was shown to positively influence comprehension and memory through increasing the organization of conceptualization (Noice & Noice, 2001). Moreover, the use of game elements in a working memory task was found to have the potential for improving learners' performance (Ninaus et al., 2015). In a review of empirical studies on gamification, Hamari, Koivisto, and Sarsa (2014) concluded gamification increases motivation and engagement in the learning task. Therefore, we expected our game-based approach on learning to program to be motivating and engaging for students and to yield significant learning effects. In the following, we will briefly describe the course curriculum employed before reporting the results of the study.

1.2 Course curriculum

The game-based CT teaching approach evaluated in this article was part of a CT course that was taught at two primary schools in Germany as part of an exploratory pilot project in 2016 and 2017. The course followed a concept-based curriculum modeled closely on the existing Course 2 (Code.org, Course 2) by Code.org (Code.org, About US), which is available under Creative Commons Attribution 4.0 International License and has been successfully evaluated in programs for primary school students across the United States (Code.org, Evaluation Summary Report). Code.org's Course 2 is aimed at children who are already able to read and write, but who have not yet taken any previous courses on computer science, programming, or CT. It follows a CT approach by placing its focus on conceptual understanding rather than specific implementations and makes use of many 'unplugged' exercises and teaching units. Exercises being 'unplugged' means they abstain from using technology and instead are carried out using pencils, paper, scissors, glue, playing cards, or even sports-like games.

The course started by introducing students to a key computer science concept – the *algorithm*, a sequence of commands that need to be followed to solve a problem. The following teaching units explored ways in which algorithms can be structured, improved, and the flow of algorithmic instructions can be controlled. In addition to algorithms, the fundamental concepts around which the curriculum is structured include *sequences*, *loops*, *conditional branching*, and *events*. Furthermore, students are intended to learn how to debug algorithms. To complement the 16 lessons we selected from Code.org's Course 2 and to relate them to the existing primary school curriculum, we added 2 unplugged lessons prompting children to transfer their acquired understanding of CT skills and concepts to solve mathematical problems.

Throughout the course, the focus was on helping students develop an understanding of these concepts. The course applied this conceptual approach to help students build a long-lasting and sustainable understanding of CT. Specific technological applications including programming languages are developing fast, but it is argued that the underlying computational concepts are what is fundamental to computer science (Wing, 2006) and that they are applicable more universally (Wing, 2010). Any technology used in the course was only utilized as means to increasing conceptual understanding rather than being an end in itself. For this reason, some of the teaching

units contain programming exercises closely interlinked with the respective unit's conceptual content. Overall, 10 lessons included unplugged activities, while the remaining 8 lessons included programming exercises using tablet devices and desktop computers. Over the course of the curriculum, each CT concept was first introduced in one or more unplugged lessons and consequently applied in a 'plugged-in' lesson.

2. Methods

2.1 Course framework

2.1.1 Lesson structure and course set-up

All of the lessons evaluated in this article followed the same structure. At the beginning of the respective lesson, students were asked to recall the previous lesson's activities and were encouraged to ask questions they have come up with in the aftermath of the previous lesson. Then, the terms and vocabulary for any new CT concepts introduced were presented and discussed with students. The central part of the lesson always consisted of the respective game activity. After the game, the teacher prompted them to discuss what they learned from this activity and how this may be implemented when working with an algorithm. At the end of the lesson, students were handed the assessment sheet on the CT concept introduced in the lesson and were given ten minutes to solve it. Each end-of-lesson assessment tested assigned learning objectives (Code.org, Course 2) and students' results were assessed based on which percentage of the respective learning objectives they reached.

The course was taught by four computer science undergraduate and graduate students who received formal training in teaching the CT course in the format of a weekly two-hour (120 minutes) course over a timespan of three months. Students taught the course in teams of two, which were assigned to the two schools respectively.

The course at school A (n = 22 students; 10 female; mean age = 8.76 years; SD = 0.75) was taught in lessons of 90 minutes each on a weekly basis over the course of 9 weeks. In contrast, the course at school B (n = 11 students; 5 female; mean age = 9.00 years; SD = 0.67) was taught in lessons of 45 minutes each on a weekly basis over the course of 18 weeks. Written informed consent of parents or legal guardians was required for the students to sign up for the courses, which were offered during the schools' afternoon programme. Both schools combined, the total number of participants was 33 3rd and 4th grade students between eight and ten years old.

2.1.3 Student background

Before the start of the course, children filled out self-assessment questionnaires. These questionnaires asked students to provide background information on, for instance, their previous programming experience or whether their parents were working in IT-oriented jobs.

Out of 33 students participating in the course, 28 filled out the questionnaires – the remaining 5 students had not been present when the questionnaires were administered. Evaluation of questionnaire data yielded the following results: 14 students (6 female) stated at least one of their parents' jobs had something to do with information technology. 6 students (2 female) stated they had some previous experience in programming. 2 of these students, and 3 others who had no programming experience (no female), stated they knew what the term 'computer science' means. 5 out of the 9 students who had previous programming experience or knew what the term 'computer science' means had at least one parent with an IT-related job.

2.2 Pre- and post-course questionnaires

At the start and at the end of the course, students filled out questionnaires to rate their perception of the course in general as well as their interest in computation-related topics and their self-perceived learning progress.

In both the pre- and post-course questionnaire, students were asked to indicate their answer to the question "Could you imagine working in an IT job?" on a 5-point Likert scale ranging from "not at all" (1) to "absolutely" (5) (with only the endpoints labelled accordingly). In the post-course questionnaire only, they were also asked to indicate on the same Likert scale (i) whether they liked the course in general, (ii) whether they learned something new in the course, and (iii) whether they were interested in learning more about computer science.

2.3 The unplugged games

The following three games each made up the central part of one of the lessons evaluated in this article.

2.3.1 "Relay Programming" – a debugging game

Debugging is the action of systematically searching for and consequently resolving problems or errors in an algorithm or program. The lesson's educational objective was for students to be able to find and fix problems in existing programs. This ability was introduced and practised using an unplugged game.

For this game, students were divided into teams of three to five players to compete in a race against the other teams and against time. For each team, the goal was to create a program – command by command – and to debug this program in case any errors occurred. The game's learning objective was for the students to learn to detect errors quickly and debug their program systematically. The time pressure brought about by the competitiveness and fast pace of the game was supposed to provoke students to make mistakes when writing down commands, thus providing the opportunity to learn debugging.

To play the game, each team had to form a queue on one side of the room, just like in a relay race. For each of the teams, the same image is placed on the other side of the room, as well as a sheet of paper on which the teams should write down their program. All images were grids of four times four squares, some of which are black and some are white. One corner of the grid was marked as the starting point for the program. See Figure 1 for an example picture and the commands that can be used to write the program.

When the game starts, the first student in line for each team had to run to the other side of the room, look at the picture, and write down the first command for the program that would reproduce that picture. Then the student had to run back to their team, clap the hand of the next student in line, and go to the end of the queue. The next student in line, who just got a handclap from the previous player, then had to run to the other side of the room, look at the picture, and either debug the team's current program by crossing out an incorrect command or add a new command. Then the student had to run back to their team, clap the next teammate's hand, and the cycle started over again. Importantly, on each turn, each player can only perform one action – either debug a previous command or add a new command. This cycle was repeated until one of the teams completed their program and thereby won the game. The game was played several times in a row with increasing difficulty, determined by the complexity of the respective picture.

Due to limited space in the course classroom at school A, it was impossible to ensure large enough spatial separation between the currently active player and the rest of the team during the "Relay Programming" game phase of the lesson. As a consequence, the active player sometimes received help from their teammates.

At the end-of-lesson assessment, students had to fill out a short test comprising four exercises. Each exercise consisted of one picture just like in the "Relay Programming" game and a sequence of commands (see Figure 1). When the sequence is executed, it was supposed to draw the respective picture. However, there is one faulty command in each sequence. It was the students' task to find each mistake, circle it, and write a new, correct sequence of commands for drawing the picture. This assessment tested how well students were able to detect a bug within a sequential algorithm and replace it with an error-free sequence of commands, which they had learned to do through playing the "Relay Programming" game.

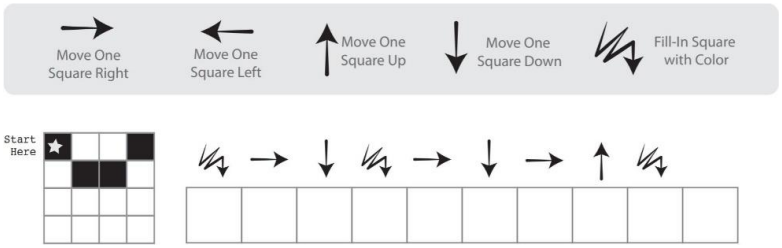


Figure 1: One of the four tasks referring to the debugging lesson's assessment sheet (Code.org, Assessment 9)

2.3.2 "Conditionals with Cards" – a conditional branching game

Conditionals are expressions used for control flow branching within an algorithm, depending on whether the condition is met, meaning whether the expression is true or false. Students had to learn about how conditionals work by playing an unplugged card game with two levels of difficulty, the rules of which were based on awarding points depending on whether a specific condition was met. To play the game, students were split into two teams. Both teams had to keep track of the current score for their own team as well as for their opponent.

For the first difficulty level, teams took turns in each drawing one card from a standard poker deck. The game's rules applied simple conditionals as shown in Figure 2.

```
If (CARD is RED)
    Award YOUR team 1 point
Else
    Award OTHER team 1 point
```

```
If (CARD is RED)
    Award YOUR team 1 point
Else
    If (CARD is higher than 9)
        Award OTHER team 1 point
    Else
        Award YOUR team the same
        number of points on the card
```

Figure 2: Example of simple (Panel A) and nested (Panel B) conditionals in "Conditionals with Cards"

When it was clear all students understood the rules, they moved on to the second level and were given a new set of rules, this time containing nested conditionals as shown in Figure 2.

For the respective end-of-lesson assessment, students filled out a short test instructing them to imagine they were watching a game of cards being played by two teams. Then participants were asked to write down the teams' scores after each turn. The game they "watched" was very similar to the "Conditionals with Cards" game they had just played and its rules also made use of nested conditionals (see Figure 4). This assessment was supposed to test how well students could trace conditional branching in an algorithm and award points to two teams accordingly, which they had learned to do by playing the "Conditionals with Cards" game.

```
If (CARD is lower than 5)
    If (CARD is BLACK)
        Award YOUR team the same
        number of points on the card
    Else
        Award OTHER team 1 point
Else
    If (CARD is HEARTS)
        Award YOUR team 1 point
```

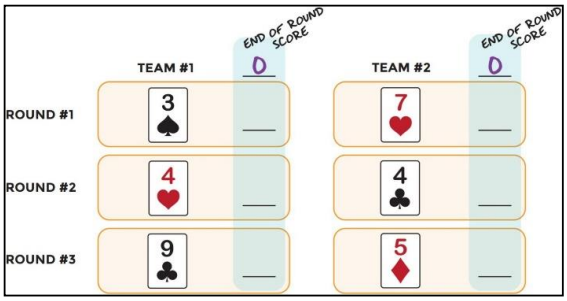


Figure 4: Exemplary assessment task on the conditional branching lesson's assessment sheet (Code.org, Assessment 12)

In case students follow the algorithm and evaluate all conditionals correctly, they keep track of the score as follows:

1st round, 1st turn: Team 1 draws the 3 of spades (lower than 5, black), awards 3 points to team 1.

1st round, 2nd turn: Team 2 draws the 7 of hearts (not lower than 5, hearts), awards 1 point to team 2.

2nd round, 1st turn: Team 1 draws the 4 of hearts (lower than 5, not black), awards 1 point to team 2.

2nd round, 2nd turn: Team 2 draws the 4 of clubs (lower than 4, black), awards 4 points to team 2.

3rd round, 1st turn: Team 1 draws the 9 of clubs (not lower than 5, not hearts), awards 0 points to either team.

3rd round, 2nd turn: Team 2 draws the 5 of diamonds (not lower than 5, not hearts), awards 0 points to either team.

Thus, the final score after the last turn is 3 points for team 1 and 6 points for team 2, which is the winning team.

2.3.3 "The Big Event" – an events game

Events are actions an algorithm is designed to recognize and react to. Typical events handled by computer programs include user actions such as mouse clicks, which can be used to make a program interactive.

A sports game was used to help students develop a basic understanding of how events work and what they can be used for. For this game, students were introduced to five geometric symbols, each of which was linked to one

specific action such as standing on one leg. Then, students were instructed to randomly run across the schoolyard until the teacher would hold up a cardboard sign showing one of the symbols. Upon the event of being shown a symbol, students had to react by performing the corresponding action. When the sign was taken down again, they resumed running until another symbol was shown.

To make the activity more challenging and to introduce chains of events, students were split into three teams for a second iteration of the game. Now each team only had to react to some of the symbols but not to all of them. Additionally, however, students did not only have to react to the event of a symbol being shown, but also to the event of another team performing a specific action. For example, when team X observes another team Y doing "jumping jacks", then team X has to react by clapping their hands three times.

At the respective end-of-lesson assessment, students had to fill out a short test with four short tasks. For each task, students had to find the correct reactions to a sequence of events. The events were represented by geometric shapes, just like in "The Big Event" game. Each event causes a picture of a specific animal to be displayed (see Figure 5). This assessment tested how well students were able to respond to individual events in a chain of events by linking them to the respective assigned reaction, which they had learned to do through playing the "Big Event" game.

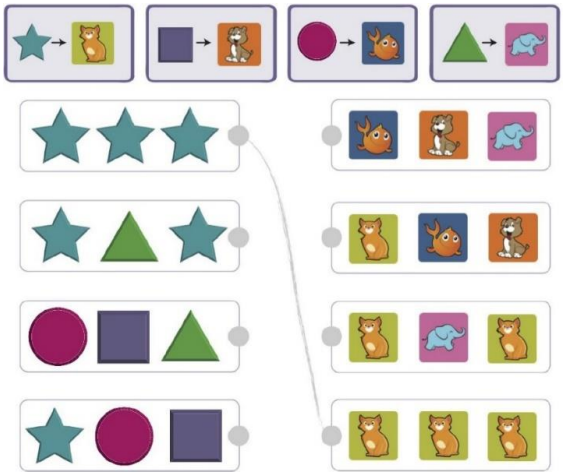


Figure 5: Exemplary task for the event lesson's assessment

3. Results

3.1 Pre- and post-course questionnaires

Out of 33 students who attended the course, 22 completed both the pre- and the post-course questionnaires. In both questionnaires, students were asked whether they could imagine working in an IT job, on a 5-point Likert scale. On average, interest in such a job decreased from M=3.27 (SD=1.39) to 2.82 (SD=1.27).

Three of the 22 students who answered both the pre- and post-course questionnaire rated their pre-course vocational interest as "not at all", and six of them as "absolutely". In both school A and school B there was one student who reported previous experience in programming. After the course, of the 22 students, 4 rated their vocational interest as "not at all", and two as "absolutely".

The post-course questionnaires were completed by 28 students. On the 5-point Likert scale, the students rated how they liked the course overall with on average M=4.36 (SD=0.89). For their self-assessed learning progress made in the course, they gave an average rating of M=4.39 (SD=0.98). The interest in learning more about computer science was rated on average with M=3.36 (SD=1.34).

3.2 The unplugged games

3.2.1 "Relay Programming" – a debugging game

Figure 6 provides an overview of how many of the students were able to solve which percentage of the assessment sheet correctly. Out of 25 students who completed the assessment at both schools combined, 16

were able to solve all tasks of the assessment correctly. 20 students' solutions were in the highest quartile of correctness. On average, students solved 89% of the assessment correctly (SD = 0.2).

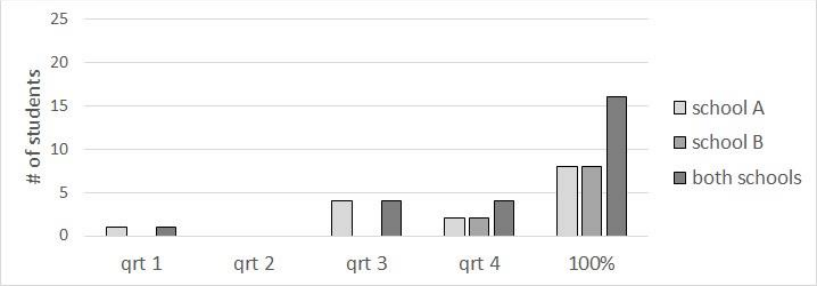


Figure 6: Results of the end-of-lesson assessment – how many students reached the debugging lesson's educational objective

3.2.2 "Conditionals with Cards" – a conditional branching game

Figure 7 provides an overview of how many of the students were able to solve which percentage of the assessment sheet correctly. Out of 27 students who completed the assessment at both schools combined, none were able to solve all tasks of the assessment correctly. 24 students' solutions were in the lowest quartile of correctness. On average, students solved 6% of the assessment correctly (SD=0.16).

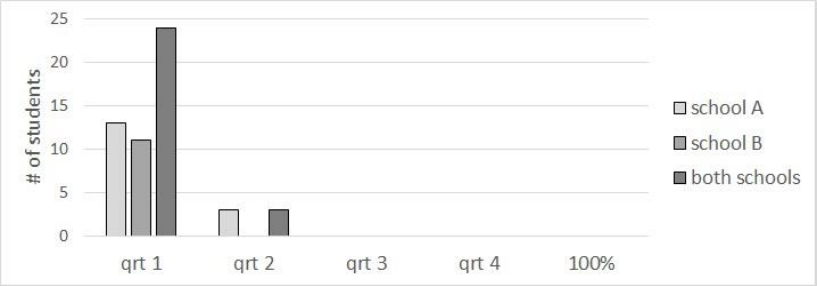


Figure 7: Results of the end-of-lesson assessment – how many students reached the conditional branching lesson's educational objective

These results conflict with our experience from the conditional branching lesson, in which we observed students were well able to understand the rules and play the "Conditionals with Cards" game. Some students had at first been confused by nested conditionals. The idea that it was not obvious at first glance which case should be followed was new to students, but by the end of the game, all students were able to follow the rules and correctly award points to the respective team.

To better understand this inconsistency between our observations from the lesson and the results of the end-of-lesson assessment, we analysed students' solutions of the tasks for types of mistakes on the level of individual turns and identified four solution types: For each turn, students either gave a solution that was (a) correct, (b) partially correct – the score was incorrectly added up due to a previous mistake –, (c) incorrect – the mistake came from following an instruction from the wrong block –, or (d) incorrect – the mistake was not in accordance with any of the instructions. We found for each respective turn, either the majority of students found the correct solution, or one specific type of mistake was dominant. There was no relevant difference in the distribution of correct solutions and types of mistakes between schools, which is why we discuss these mistakes for both schools combined.

For both turns of the *first round*, all or almost all students gave the correct solution. For the first turn of the *second round*, the dominant type of mistake was students coming to an incorrect solution as a result of following an instruction from the wrong block. In the second turn of the second round, the majority of students calculated an incorrect score as a consequence of the previous mistake. In both turns of the *last round* of the game, the prevalent type of mistakes was students specifying an incorrect solution as a result of following an instruction from the wrong block.

Possible reasons for the occurrence of these specific types of mistakes will be discussed in section 4.2 (Interpretation of the end-of-lesson assessment results).

3.2.3 "The Big Event" – an events game

Figure 8 provides an overview of how many of the students were able to solve which percentage of the respective assessment sheet correctly. Out of 21 students who completed the assessment at both schools combined, 19 were able to solve all tasks of the assessment correctly. 19 students' solutions were in the highest quartile of correctness. On average, students solved 90% of the assessment correctly (SD = 0.29).

Figure 8: Results of the end-of-lesson assessment – how many students reached the event lesson's educational objective

4. Discussion

4.1 Interpretation of questionnaire data

It is a long-term goal to get more children and students to be interested in studying computer science and in taking related jobs. Thus, the results for the questionnaire data indicating a descriptive decrease in vocational interest of students after the course seem surprising. However, it needs to be noted the difference was not significant and even more important additional qualitative feedback revealed most children liked the course, indicated they learned something valuable, and were interested in learning more about the topic. We consider this to necessitate further empirical studies to better understand this possible inconsistency. Moreover, we also think it would be desirable for future studies to use a more fine-grained questionnaire for inquiry into that problem.

More importantly, however, we actually observed another of our goals was clearly reached: Providing school children with information to be able to judge whether IT jobs might be interesting for them in a more differentiated way. As such, the descriptive mean decrease reflects a decrease in strong and possibly unsubstantiated opinions (i.e., extreme ratings on the Likert scale). In fact, 9 students (or 40% of participants) indicated strong interest or disinterest before the start of the course. Even if we assume the 2 students with previous programming experience gave their positive response on purpose, this still leaves us with 7 students (over 30%) with a strong but not possibly substantiated stance.

Finally, this data led us to be concerned that such strong opinions seem to be formed very early on. We think this again underlines the importance of starting in primary school with introducing children to CT and thus to concepts crucial for computer programming.

4.2 Interpretation of the end-of-lesson assessment results

Results of the end-of-lesson assessments on debugging and events revealed students on average reached 89% and, respectively, 90% of the lessons' learning objectives. This indicates the lessons and their central activities, the "Relay Programming" game and the "Big Event" game, were suitable for helping students understand and apply the CT concepts of debugging and events respectively.

Interestingly, at school A, only 10 out of 15 students scored in the highest quartile of the end-of-lesson assessment on debugging. It is possible the spatial limitations and the resulting aberrations from the intended gameplay at school A resulted in a smaller learning effect for students who received help and thus did not have the opportunity to actively practise applying the concept themselves.

Additionally, we observed students had specific problems with grasping the concept of conditional branching. The problems the students faced in some of the turns on the end-of-lesson assessment on conditional branching might be related to programming novices' difficulties to understand 'else' cases (Guzdial 2008) and to trace code linearly (Kaczmarczyk et al., 2010). This would explain why the students tended to follow instructions within the wrong outer branch if an inner conditional was true for the card that had been drawn. This is in line with the literature indicating conditional branching is indeed a complex concept to grasp for any computation novice (e.g., Guzdial, 2008) and requires to be dealt with in depth. Therefore, our first suggestion is to extend the teaching unit on branches from one lesson to two lessons.

During the first lesson, the concept of conditional branching might be introduced and students be familiarized with simple conditionals using the "Conditionals with Cards" game without any nested conditionals. In the second lesson then, the contents of the first lesson should first be recapitulated before introducing the concept of nested conditionals. The latter may then be practised in depth with the help of the card game from the first lesson augmented with nested conditionals. Teachers should also explain what happens when none of the conditions is met and there is no else statement, which students had difficulties to figure out themselves.

Our second suggestion would be to restructure the assessment worksheet to avoid dual-task interference – after all, the goal is to assess students' ability to trace an algorithm containing conditional branches, not their ability to perform more than one task at the same time. A restructured version of the end-of-lesson assessment on conditional branching avoiding dual-task interference and allows students to keep track of the score turn by turn rather than round by round might be more suitable.

These changes would also make it easier for teachers to evaluate students' performance on the assessment, as it would be observable at first glance how many points students awarded to which team for each card. This is not as clearly visible in the current layout of the task, which complicated assessing and interpreting the types of mistakes students made.

4.3 Limitations and future perspectives

There are several limiting factors to these interpretations of the results. It cannot be determined whether the difference in performance results might be explained by potential differences in students' cognitive abilities or socioeconomic status between the groups or by the differing temporal structures of the courses between both schools. The appropriateness of the evaluation of questionnaire data regarding students' interest in working in an IT-related job might be impaired due to CT and programming skills – which were central in the course – not being required for, and therefore not representative of, the entire IT sector. In addition, while the students who taught the children received a weekly two hours of teaching training over the course of three months prior to teaching the course, a difference in teaching quality compared to professional teachers might be expected.

The results of this pilot project – as well as its limitations – have proved helpful in informing the planning and design of a new study on teaching CT. Building on the experience gathered in this pilot project, we are currently conducting a study evaluating a game-based CT training program we developed (Tsarava et. al, 2017). To overcome such limitations as the ones described above, we are conducting a randomized controlled field trial with a pre- and post-test design using standardized and validated questionnaires and testing instruments. The CT training program is taught by professional teachers.

In conclusion, 3rd and 4th grade students seemed able to understand the respective CT concepts across all lessons and apply them while playing the unplugged games. This was reflected in the results of the post-course self-assessment questionnaire, in which students rated their course experience very positively and reported high levels of interest in learning more about computer science. Furthermore, considering students on average reached at least 82% of the learning objectives except for conditional branching, we are confident the unplugged game-based approach was motivating and beneficial for fostering the understanding of CT concepts in primary school students.

Acknowledgement

This project was supported by the Vector Foundation under the funding ID P2015-0036.

References

Balanskat, A. and Engelhardt, K. (2015) "Computing our future. Computer programming and coding – priorities, school curricula and initiatives across Europe", *European Schoolnet*, Brussels.

Barsalou, L. W. (2008) "Grounded cognition", *Annual review of psychology*, 59, pp. 617–645.

Code.org (n.d.). *About Us*. [online] Available at: https://code.org/about.

Code.org (n.d.) *Course 2*. [online] Available at: https://studio.code.org/s/course2.

Code.org (n.d.) "Assessment 9", *Course 2* [online] Available at: https://code.org/curriculum/course2/9/Assessment9-RelayProgramming.pdf.

Code.org (n.d.) "Assessment 12", *Course 2* [online] Available at: https://code.org/curriculum/course2/12/Assessment12-Conditionals.pdf.

code.org (n.d) *Evaluation Summary Report 2015 – 2016*. [online] Available at: https://code.org/files/EvaluationReport2015-16.pdf.

Guzdial, M. (2008) "Education: Paving the Way for Computational Thinking", *Commun. ACM*, Vol. 51, No. 8, pp. 25–27.

Hamari, J., Koivisto, J., and Sarsa, H. (2014) "Does gamification work? – A literature review of empirical studies on gamification", *Proceedings of the Annual Hawaii International Conference on System Sciences*, pp. 3025–3034.

Kaczmarczyk, L.C., Petrick, E.R., East, J.P., and Herman, G.L. (2010) "Identifying Student Misconceptions of Programming", *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, pp. 107–111.

Kumar, D. (2014) "Welcome", *ACM Inroads*, Vol. 5, No. 4, pp 52–53.

Lister, R. (2016) "Toward a developmental epistemology of computer programming", *Proceedings of the 11th workshop in primary and secondary computing education*, ACM, pp. 5–16.

Ninaus, M., Pereira, G., Stefitz, R., Prada, R., Paiva, A., and Wood, G. (2015) "Game elements improve performance in a working memory training task", *International Journal of Serious Games*, 2(1), pp. 3–16.

Noice, H. and Noice, T. (2001), "Learning dialogue with and without movement", *Memory & Cognition*, Vol. 29, No. 6, pp. 820–827.

Papert, S. (1972) "Teaching Children Thinking", *Programmed Learning and Educational Technology*, Vol. 9, No. 5, pp. 245–255.

Papert, S. (2005) "You can't think about thinking without thinking about thinking about something", Contemporary Issues in Technology and Teacher Education, Vol. 5, No. 3, pp. 366–367.

Piaget, J. (1972) "Intellectual evolution from adolescence to adulthood", *Human development*, Vol. 15, No. 1, pp. 1–12.

Prottsman, K. (2014) "Computer Science for the Elementary Classroom", *ACM Inroads*, Vol. 5, No. 4, pp 60–63.

Teague, D.M., Corney, M.W., Fidge, C.J., Roggenkamp, M.G., Ahadi, A., & Lister, R. (2012) "Using neo-Piagetian theory, formative in-Class tests and think alouds to better understand student thinking: a preliminary report on computer programming", *Proceedings of 2012 Australasian Association for Engineering Education (AAEE) Annual Conference*, pp. 772–780.

Tsarava, K., Moeller, K., Pinkwart, N., Butz, M., Trautwein, U., & Ninaus, M. (2017) "Training computational thinking: Game-based unplugged and plugged-in activities in primary school", *Proceedings of the 11th European Conference on Game Based Learning*, pp. 687-695.

Willson, M. (2017) "Algorithms (and the) everyday", *Information, Communication & Society*, Vol. 20, No. 1, pp. 137–150.

Wing, J.M. (2006) "Computational Thinking", *Communications of the ACM*, 49(3), pp. 33–35.

Wing, J.M. (2010) "Computational Thinking: What and Why?", T*he Link - The Magazine of the Carnegie Mellon University School of Computer Science*, pp. 1–6.

Wing, J.M. (2014) "Computational Thinking Benefits Society", [online], New York Academic Press, http://socialissues.cs.toronto.edu/index.html%3Fp=279.html

Yadav, A., Mayfield, C., Zhou, N., Hambrusch, S., and Korb, J.T. (2014) "Computational Thinking in Elementary and Secondary Teacher Education", *ACM Transactions on Computing Education*, 14(1), pp. 1–16.

Manuscript 2

Title:
Development of a Questionnaire on Self-concept, Motivational Beliefs, and Attitude Towards Programming

Authors:
Luzia Leifheit, Katerina Tsarava, Korbinian Moeller, Klaus Ostermann, Jessika Golle, Ulrich Trautwein, and Manuel Ninaus

Published in:
Proceedings of the 14th Workshop in Primary and Secondary Computing Education (WiPSCE). ACM.

https://doi.org/10.1145/3361721.3361730

Time of publication:
October 2019

Development of a Questionnaire on Self-concept, Motivational Beliefs, and Attitude Towards Programming

Luzia Leifheit
LEAD Graduate School and Research Network, University of Tübingen
Leibniz-Institut für Wissensmedien
Tübingen, Germany
luzia.leifheit@uni-tuebingen.de

Katerina Tsarava
Leibniz-Institut für Wissensmedien
LEAD Graduate School and Research Network, University of Tübingen
Tübingen, Germany
k.tsarava@iwm-tuebingen.de

Korbinian Moeller
Leibniz-Institut für Wissensmedien
Department of Psychology
LEAD Graduate School and Research Network, University of Tübingen
Tübingen, Germany
k.moeller@iwm-tuebingen.de

Klaus Ostermann
Department of Computer Science
LEAD Graduate School and Research Network, University of Tübingen
Tübingen, Germany
klaus.ostermann@uni-tuebingen.de

Jessika Golle
Hector Research Institute of Education Sciences and Psychology
LEAD Graduate School and Research Network, University of Tübingen
Tübingen, Germany
jessika.golle@uni-tuebingen.de

Ulrich Trautwein
Hector Research Institute of Education Sciences and Psychology
LEAD Graduate School and Research Network, University of Tübingen
Tübingen, Germany
ulrich.trautwein@uni-tuebingen.de

Manuel Ninaus
Leibniz-Institut für Wissensmedien
LEAD Graduate School and Research Network, University of Tübingen
Tübingen, Germany
m.ninaus@iwm-tuebingen.de

ABSTRACT

Academic self-concept, motivational beliefs, and attitudes towards a school subject are relevant for learning and educational achievement. A positive self-concept in science and mathematics is argued to motivate students to persist and advance in studying these subjects. In particular, self-concept, motivational beliefs, and attitudes towards STEM domains were found to be predictive of educational achievement. Recently, programming was suggested to be a key competence in education.

To assess self-concept, motivational beliefs, and attitudes towards programming, we developed a new questionnaire based on existing scales for mathematics. The new questionnaire assesses the same aspects for programming on seven subscales, such as self-concept, belief about usefulness, and self-reported persistence when working on programming tasks.

We conducted a pilot study in which we used this questionnaire to measure self-concept, motivational beliefs, and attitudes towards programming. The study was set in the context of an extracurricular course on computational thinking (CT) for elementary school students between the ages of seven and ten years. Before the start of the course, we assessed all 31 participating students' self-concept, motivational beliefs, and attitudes towards programming using the developed questionnaire and their CT skills using the Computational Thinking test (CTt).

Our results confirmed the expected associations between the aspects assessed by our questionnaire. However, we did not find significant associations of questionnaire results and CT skills. Consequently, future research involving a larger sample is needed to better understand the association between children's actual performance and their self-concept, motivational beliefs, and attitudes towards programming.

WiPSCE '19, October 23–25, 2019, Glasgow, Scotland

ACM ISBN 978-1-4503-9999-9/18/06…$15.00
DOI: 10.1145/3361721.3361730

CCS CONCEPTS

• **Social and professional topics** → **Computer science education**; **K-12 education**; **Computational thinking**; *Computing literacy*; • **Applied computing** → *Education.*

KEYWORDS

computer science education, computing education, computational thinking, computing literacy

ACM Reference Format:
Luzia Leifheit, Katerina Tsarava, Korbinian Moeller, Klaus Ostermann, Jessika Golle, Ulrich Trautwein, and Manuel Ninaus. 2019. Development of a Questionnaire on Self-concept, Motivational Beliefs, and Attitude Towards Programming. In *Proceedings of WiPSCE '19: Workshop in Primary and Secondary Computing Education (WiPSCE '19).* ACM, New York, NY, USA, 9 pages.

1 INTRODUCTION

1.1 Academic Self-concept, Motivational Beliefs, and Attitude

Academic self-concept, motivational beliefs, and attitude were repeatedly observed to be significant predictors of learning progress and achievement [24, 38, 48, 51]. In particular, a positive self-concept in science and mathematics has been found to motivate students to persist and advance in their studies of science and mathematics [1, 21, 51]. Additionally, motivational beliefs and self-concept in science have been shown to reliably predict academic achievement [28, 51]. Considering this impact of academic self-concept, motivational beliefs, and attitude towards STEM (Science, Technology, Engineering, Mathematics) subjects, it seems worthwhile to measure, monitor, and foster these variables in students – in particular because individuals trained in STEM competencies have increasing employment market opportunities with high monetary compensation and social recognition [27].

Importantly, in addition to STEM competencies, the increasing importance of computer science in general and programming in particular was emphasized repeatedly in recent years (e.g., [2, 18, 34]). So far, however, few studies evaluated the relevance of students' academic self-concept, motivational beliefs, and attitude towards programming. Therefore, the current study aimed at piloting a questionnaire on academic self-concept, motivational beliefs, and attitude towards programming based on an existing one for mathematics [4, 13, 38]. The same mathematics questionnaire was previously adapted and validated for other school subjects, such as biology, physics, English, and German [14].

Because the developed questionnaire is based on existing instruments for other school subjects, it would allow for assessing academic self-concept, motivational beliefs, and attitude towards programming in relation to and comparable to the same constructs for other subjects. The availability of instruments for assessing the same constructs across various school subjects would offer possibilities for research on the relationship between self-concept, motivational beliefs, and attitude in different subjects. In the following, we will first describe the constructs assessed by the questionnaire before we report empirical results of a first pilot study.

1.1.1 Constructs Measured by the Developed Questionnaire. The questionnaire we developed assesses the following aspects of academic self-concept, motivational beliefs, and attitude towards programming as well as students' self-reported previous experience and understanding of programming on seven subscales.

i) The first subscale measures students' self-reported *previous experience and understanding of programming*. Importantly, this aspect is not modeled on the existing questionnaire on mathematics. However, we included it because this scale may provide important information on the background against which the results on the other scales need to be interpreted.

ii) *Self-concept with regard to programming* is assessed on the second subscale. Academic self-concept has been defined as a person's self-perception with respect to achievement in school [31]. Correspondingly, individuals' self-concept regarding a specific subject reflects their confidence in their own ability to do well in that subject [31, 51]. Positive self-concept was found to be an important predictor of achievement in a subject: when students have no confidence in their ability to perform well in a subject, they have no reason even attempting to succeed [28].

Motivational beliefs are individual beliefs about a subject that motivate or demotivate a student to engage with a subject and make the effort required for achieving in it. Thus, these beliefs reflect the motivational value a person attributes to a subject or task and therefore are also termed value beliefs. Positive motivational beliefs have been associated with students' persistence in attempting to perform well even when their interest and intrinsic enjoyment of the subject decrease [28]. As specified in expectancy-value theory, Eccles et al. differentiate between four aspects of motivational value belief: intrinsic value belief, attainment value belief, utility value belief, and cost belief [11, 49]. Accordingly, we assess these aspects in four further subscales:

iii) *Intrinsic value belief about programming* is measured on the third subscale. Intrinsic value belief with respect to a subject is defined as the degree to which a person intrinsically enjoys and is interested in the subject. This construct reflects a person's attitude towards the subject. While a positive attitude towards a subject is not a necessary requirement for achievement, it can support and increase a student's engagement in a subject [28]. Furthermore, intrinsic value belief about a subject is a predictor for students' voluntary engagement with a subject in their leisure time [10, 25].

iv) The fourth subscale assesses *attainment value belief about programming*. With regard to any specific subject, attainment value belief reflects the importance an individual places on a subject [13].

v) *Utility value belief about programming* is measured on the fifth subscale. Utility value belief about a subject represents an individual's expectation of the subject's usefulness in different areas of life, such as everyday life, school, future career, or social life [13].

To distinguish between the terms, intrinsic value belief and attainment value belief are considered intrinsic motivational factors, while utility value belief is seen as an extrinsic motivational factor [41]. All three of these beliefs have been linked with future career ambition and selection of classes [10].

vi) The sixth subscale assesses students' *cost belief about programming*. This belief about a subject reflects the negative consequences a person expects to result from engaging in the subject, including assumed effort and exhaustion as well as negative emotions [13, 30].

Including such a broad spectrum of motivational beliefs in the questionnaire is one of its key strengths, as covering these four dimensions makes education attitude questionnaires powerful for predicting academic choices [50]. In qualitative analyses, intrinsic value belief – or enjoyment –, utility value belief, and cost belief have already been established as important factors influencing students' decision whether or not to pursue further education in computer science [23].

vii) Finally, *compliance and persistence with regard to programming* are measured on the seventh subscale. These variables were characterized as indicators for students' thoroughness in working on a task as well as their resilience in the face of challenging tasks [38]. Correspondingly, high persistence and compliance have been linked to better learning achievement [38].

In the present study, we evaluated these seven variables' correlation with each other and with students' computational thinking

(CT) abilities as a cognitive skill underlying programming and coding. We used an adapted version of the Computational Thinking test (CTt) [33] to measure students' ability to understand and apply CT concepts.

1.2 Related Work

1.2.1 Instruments for Assessing Students' Attitude towards Computing and Programming. To allow for an assessment of students' interest in or attitudes towards computing, some questionnaires have been developed in recent years. These questionnaires may broadly be divided into two categories: one subsuming questionnaires pertaining more narrowly to computer science or computing, and the other to computer use or interaction with computing devices more generally.

The first category includes questionnaires such as the Computing Attitudes Survey (CAS). The CAS was developed to assess students' attitudes and beliefs about problem solving and the nature of knowledge within computer science [9]. Four of the instrument's 26 items assess personal interest in and enjoyment of computer science, while the remaining 22 items were designed to assess students' self-estimated computing-related abilities, such as coming up with specific problem solving strategies. Items were mostly modeled on items of existing instruments for other subjects, such as biology [12]. Validity of the CAS instrument was evaluated for target groups in post-secondary computing education [9].

Other instruments in this category include a two-part questionnaire for measuring high school students' attitudes towards computing [15] and the Computer Science Attitude Survey [16]. The former includes two scales with the same five subscales each – one scale for "computer science" and one for "information technology" [15]. The latter has been developed for assessing science and engineering students' attitudes toward computer science and is aimed at undergraduate college students [16].

These instruments are all aimed at either high school or college students and their items are phrased to be appropriate and comprehensible for this target group. However, they would be inappropriate for a target group at elementary school age for several reasons. First, regarding grammar and vocabulary, their items seem too complex for younger children. Secondly, they have in part been developed for a target group with previous experience in computer science (e.g. CAS, [9]). Second, they aim at assessing attitude toward computer science, computing, or information technology – terms and domains which are unlikely to be comprehensible or familiar for students significantly younger than high school age.

The second category of instruments includes questionnaires aimed at assessing attitudes toward computer use or interaction with computers, such as the Microcomputer Beliefs Inventory (MBI) [32]. This instrument consists of 26 items for measuring self-efficacy and outcome expectancy beliefs regarding computer use and is aimed at middle school students [32].

Another instrument from this category is a questionnaire comprising 21 items for assessing attitudes towards interacting with computers in high school students between the ages of 16 and 19 years [36].

The instruments from this second category are mostly targeted at middle or high school students and therefore at a slightly younger target group than those from the first category. Additionally, they do not require previous experience in programming or computer science. On this basis, these instruments could more easily be adapted for assessing elementary school students than instruments from the first category. However, assessing attitude towards computer use or interaction with computing devices is not equivalent to assessing attitude towards the more narrowly defined domain of computing.

For these reasons, existing instruments from either of these two categories appear unsuited for being used or adapted for assessing elementary school students' attitude towards programming, either due to being too complex and requiring too much previous knowledge and experience, or due to not being domain-specific enough.

This motivated our decision to develop a new instrument for specifically assessing academic self-concept, motivational beliefs, and attitude towards programming partly based on instruments for assessing the same constructs for other subjects in students of the same age group.

1.2.2 The Role of CT. The idea of CT was first described by Seymour Papert [29], referring to cognitive processes that play a fundamental role in the systematic development of computational procedures. Today, the term is being used more broadly.

Jeannette Wing [52] characterized CT as an essential ability for understanding, formulating, and solving complex problems, which often requires partitioning, abstraction, generalization, parameterization, modeling, and algorithmization. Such solution approaches are typically strategic, systematic, abstract, reproducible, algorithmic, and, most importantly, computable. Nevertheless, it does not matter whether a person or a machine is going to execute the resulting computation [52].

For the purpose of introducing young students to programming, such algorithmic solutions have been identified as consisting of a basic set of CT concepts, such as sequences, loops, events, conditional branching, operators, and data [3].

While some argue CT to reflect conceptual competences obtained through – and therefore intrinsically linked to – programming and studying computer science [26], others frame it as a new and fundamental way of thinking with problem-solving benefits superior to other ways of thinking, as pointed out by Denning et al. [8]. Other definitions portray CT as a cognitive skill in addition to reflecting the practical skill of programming [37] and thus emphasize the wide spectrum of CT applicability. Therefore, CT is also suggested to be a 21st century skill [45, 53], which is valuable to be acquired and developed already in early education [53].

In recent years, the idea of CT has been the focal point of continuous discussion and attention. However, the notion that underlying concepts are the most important part of computer science is not new, but has already been emphasized by Donald Knuth [20].

In summary, when we refer to CT, we consider it the conceptual core of computing describing principles and methods rather than specific tools or technological systems [26].

1.3 Aim of the Study

The present pilot study aimed at providing a first indication of the potential validity of our newly developed questionnaire for assessing children's academic self-concept, motivational beliefs, and

attitude towards programming. We approach the validation process by first utilizing the instrument in this pilot study to appraise its general usability before initiating a full validation study.

We expected the questionnaire results for the seven subscales of academic self-concept, motivational beliefs, and attitude to be correlated because they are designed to measure related constructs (for similar results on mathematics, see [13]). However, we neither expected correlations to approach 1, nor all subscales to be correlated significantly because they assess different aspects of academic self-concept, motivational beliefs, and attitude towards programming. These expectations mirror the correlation pattern found for the mathematics-related questionnaire on which the developed questionnaire is based [4, 13].

In particular, we expected *programming self-concept* to correlate with *intrinsic value belief* and *compliance and persistence with respect to programming*, as well as with *CT performance*, because students' mathematics self-concept was found to strongly predict students' intrinsic value belief, persistence, and performance in mathematics (e.g., [7, 24, 40]). Furthermore, confidence as part of self-concept, and interest, which is an element of intrinsic value belief, were previously observed to be significantly correlated in computer science education [46].

We also expected a positive correlation between *programming self-concept* and *utility value belief about programming*, because positive associations were found between students' utility value belief about mathematics and their mathematics self-concept (e.g., [6, 17, 39]).

Moreover, *intrinsic value belief* about mathematics was found to be highly correlated with *cost belief* regarding mathematics [13]. Therefore we expected to replicate this association for the case of programming.

Finally, we expected *self-concept*, *motivational beliefs*, and *compliance and persistence* to be correlated with *CT performance*, as these variables have been found to be predictors of achievement in other subjects (e.g., [24, 28, 48, 51]).

2 METHODS

2.1 Participants

In this pilot study, we evaluated test and questionnaire data from 31 elementary school students between the ages of 7 and 10 years (M = 9.47; SD = 0.76) from 3rd and 4th grade who participated in a CT course at four Hector Children's Academies in Baden-Württemberg, Germany. The Hector Children's Academies offer extracurricular enrichment programs for elementary school children. Students are nominated for the program by their elementary school teachers based on their achievement in school. Nominated students can then select courses from the local academy's program and attend them free of charge. There are currently more than 60 Hector Children's Academies.

Participation in the study was voluntary. We obtained informed consent from each of the students as well as from their parents before they participated in the study.

2.2 Study Design

We piloted the questionnaire in the pretest and posttest evaluating a newly developed CT course. One week before the first course lesson, all participating students attended a pretest and one week after the end of the course, they attended a posttest session. These took place in classrooms of the children's schools. Students filled out anonymized paper tests and questionnaires.

In the present article, we focus on students' ratings of their academic self-concept, motivational beliefs, and attitudes towards programming at the pretest to evaluate their answers to the developed questionnaire before they attended the course. To identify potential effects of these variables on CT ability, we also assessed students' performance in the CTt at pre- and posttest time and calculated the increase in performance.

2.3 The CT Course

We utilized the new questionnaire in the context of a CT course consisting of ten lessons, each of them 90 minutes long, with students attending one lesson per week. The course aims to foster programming skills, systematic problem-solving ability, and interest in computation-related topics. Its purpose is to help students develop an initial understanding of basic programming concepts and their applications – both in the digital and the non-digital world.

In the course [44], students are introduced to basic concepts of CT, specifically sequencing, loops, parallelism, events, conditionals, operators, and data/variables (e.g. [3]). They course activities allow students to make use of CT processes, such as algorithmic thinking, conditional logic, decomposition, abstraction, pattern matching, parallelization, evaluation, and generalization (e.g. [18]). CT concepts and processes are taught using unplugged activities like the life-size educational board game "Crabs & Turtles: A Series of Computational Adventures" [43] as well as plugged-in activities making use of the educational visual block-programming language Scratch, the Scratch extension S4A (Scratch for Arduino), the Arduino open hardware platform and the Open Roberta Lab, a robots programming environment. Unplugged game-based methods for teaching CT were observed to support elementary school students' understanding of CT concepts [22], while plugged-in activities provide the opportunity for applying these concepts. The objective of the CT course is for students to get to know basic CT concepts and gain first experiences in programming while applying these concepts. Tsarava et al. provide a more detailed description of the CT course, its contents, methods, and learning objectives [42].

2.4 Measurements

2.4.1 The New Questionnaire. The developed questionnaire assesses different aspects of academic self-concept, motivational beliefs, and attitude towards programming on seven subscales, comprising four to twelve four-point Likert-scale items each. The subscales for assessing academic self-concept, motivational beliefs, and persistence regarding programming were developed based on an existing validated questionnaire for assessing academic self-concept and motivational beliefs related to mathematics [4, 13, 38]. Items were adapted by replacing the word "mathematics" with the word "programming". The same adaptation procedure was used in previous studies for adapting the mathematics instrument for the subjects biology, physics, English, and German [14]. All four

instruments developed this way were validated in a study with 830 participants [14].

In a first step, we identified critical terms to represent the focal area of the instrument. Despite the terms *programming* (German *Programmieren*) and *computer science* (German *Informatik*), we did not consider *computing* and *computational thinking*, both of which have no direct translation or commonly used equivalent in German. Other potential focal terms, such as *computer use* (German *Computernutzung*), we found too broad in scope to assess students' attitude towards computing-related activities.

To evaluate the appropriateness of the wording of the adapted questionnaire for 3rd and 4th graders (the math version was initially developed for 5th graders), we asked children attending a pre-pilot session of the CT course whether they had heard the focal terms *Informatik* or *Programmieren* before. In case they responded positively, we asked them to explain what they understood the respective word to mean. While only some of the children had heard the word *Informatik* before and none of them were able to explain its meaning, all children had heard the word *Programmieren* before and were able to either give a basic explanation or provide examples of what programming activities could look like. Thus, we decided on using the word *programming* (German *Programmieren*) as the focal word of the instrument, constituting a compromise between subject specificity and comprehensibility for 3rd and 4th grade students.

The additional subscale for assessing previous experience and understanding of programming was not modeled on the existing questionnaire on mathematics, but was developed new. We included it to provide important information on the background against which the results on the other subscales need to be interpreted.

The scales and their items are worded in German, but examples in this article are presented in English. The seven subscales were set up the following way:

- *(P1) self-reported previous programming experience and understanding*: four items, e.g. "I can explain what the word 'programming' means."
- *(P2) programming ability self-concept*: four items, e.g. "I am good at programming."
- *(P3) programming intrinsic value belief*: four items, e.g. "I enjoy programming."
- *(P4) programming attainment value belief*: seven items, e.g. "Being good at programming means a lot to me."
- *(P5) programming utility value belief*: twelve items, e.g. "Being able to program has a lot of advantages in school."
- *(P6) programming cost belief*: eleven items, e.g "After I work on programming tasks, I often feel exhausted."
- *(P7) self-reported programming compliance and persistence*: eight items, e.g. "Even when programming tasks get challenging, I try to do my best."

All items for these scales are Likert-type, therefore participants respond by means of checking one of four boxes indicating ordinal responses (1 = agree completely; 2 = rather agree; 3 = rather do not agree; 4 = do not agree at all). This means that a low result for each of the variables indicates a high score on the respective construct. For example, an average of 1.72 for *(P2)* implies students on average have a positive *self-concept in programming*.

The scales include reversed items to filter out invalid responses by detecting inconsistencies between responses to positively phrased items and potentially contradicting responses to their negatively phrased counterpart items [19]. Responses to these items were recoded prior to analysis.

2.4.2 TIMSS 2015 Context Questionnaire. Because a relevant share of correlations found in educational research can be explained by effects of socioeconomic status [27], we assessed selected aspects of students' socioeconomic background as well as their access to digital technology:

- *(lang) frequency of German language being spoken in the student's home*: four response options (from 1 = "I always speak German at home" to 4 = "I never speak German at home")
- *(soc) possession of wealth-indicating items in the student's home as an indicator of socioeconomic status*: eleven items with "yes"/"no" responses; total number of "yes" responses checked constitutes the score for this variable
- *(use) frequency of computer/tablet use at home, in school, or elsewhere*: three items, one each for home, school, and elsewhere; four response options (from 1 = "daily or almost daily" to 4 = "never or almost never")

These variables were assessed using scales 3, 5, and 6 of the Trends in International Mathematics and Science Study (TIMSS) 2015 Context Questionnaire [47].

2.4.3 CT Performance Assessment (CTt). To test students' CT ability – specifically for recognizing and understanding sequences, loops, events, conditional branching, operators, variables, and functions – we used the CTt [33], which originally consists of 28 items. However, because it has been validated for students of age 12 and 13, which is slightly above the age group of our sample, we used only the 21 items of lowest difficulty as indicated by the item difficulty ranking [33]. In particular, we selected items 1, 2, 3, 4, 5, 6, 7, 9, 10, 11,13, 14, 17, 18, 19, 20, 21, 24, 26, 27, 28.

We calculated the following variables from the CTt:

- *(prCTt) number of correctly solved CTt items at pretest time*
- *(poCTt) number of correctly solved CTt items at posttest time*
- *(gainCTt) = (postCTt) - (prCTt)*

2.5 Statistical Analysis

All statistical analyses were performed using IBM SPSS Statistics 25. To assess the relationship between the variables *(P1)*, *(P2)*, *(P3)*, *(P4)*, *(P5)*, *(P6)*, *(P7)*, *(lang)*, *(soc)*, *(use)*, *(prCTt)*, *(poCTt)*, and *(gainCTt)*, we calculated bivariate Pearson correlations.

Through a systematic search for contradictory answers to items within the same subscale using reversed items, we identified two students who each had only checked answer boxes on the same side of the Likert scale for three of the subscales. We therefore followed the suggested approach of labeling these values as invalid and excluded the variables for the respective subscales of these two students' responses from the analysis [19].

2.5.1 Missing Data. Out of 31 students participating in the course, one did not attend the pretest due to illness. Four students did not attend the posttest; three of them had been ill and one had

dropped out of the course. The range of missing data due to absence, nonresponse, or invalid response was between 3.2 % and 22.6 % per item. Please note these percentages include the missing data due to absence of the five students who only attended either the pretest or posttest.

We used the multiple imputation algorithm integrated into IBM SPSS Statistics 25 to impute the missing data. When performing a multiple imputation, each missing item is replaced by a list of $m > 1$ simulated values for the respective item [35]. This generates m plausible versions of the dataset. When analyzing an imputed dataset, the same analysis is calculated for each of the m datasets and the results are combined into an overall estimate whose standard errors reflect missing-data uncertainty and finite-sample variation [35].

Multiple imputation is a mathematically sophisticated missing data treatment method which realistically models random variation [5]. It represents the state of the art of missing data treatment [35] and has been recommended for educational research [5]. Especially for smaller sample sizes, multiple imputation provides high accuracy of estimation and, on average, performs better than other missing data treatment methods [5].

3 RESULTS

Our results showed *(soc) possession of wealth-indicating items in the student's home as an indicator of socioeconomic status* (M = 4.18; SD = 1.80) and *(lang) frequency of German language use at home* (M = 1.33; SD = 0.48) to have no significant correlation with any of the self-concept, motivational beliefs, attitude, or CT performance variables. Therefore, we can conclude that language and socioeconomic status as measured by the TIMSS 2015 scales had no effect on the other variables. Consequently, we dropped *(lang)* and *(soc)* from the correlation table (see Table 1), and it was not necessary to consider them as control variables in a partial correlation analysis.

Part of the analyzed variables were non-normally distributed. Technically, this violates the assumption of normal distribution which usually needs to hold true for many analyses to be valid. However, Pearson correlation analyses specifically have a high robustness towards violation of normality and can therefore also be calculated for non-normally distributed variables (e.g. Field, 2000, p. 87).

When assessing the relationship between variables *(P1)*, *(P2)*, *(P3)*, *(P4)*, *(P5)*, *(P6)*, *(P7)*, *(lang)*, *(soc)*, *(use)*, *(prCTt)*, *(poCTt)*, and *(gainCTt)*, we found 13 significant correlations (see Table 1).

In accordance with our expectations, we found *(P2) programming self-concept* (M = 1.64; SD = 0.49) to be significantly correlated with *(P3) intrinsic value belief* (M = 1.35; SD = 0.48), *(P5) utility value belief* (M = 1.93; SD = 0.63), and significantly correlated with *(P7) self-reported compliance and persistence* (M = 1.55; SD = 0.55). Thus, children who reported high scores on programming self concept also reported high scores on intrinsic value belief, utility value belief, and compliance and persistence.

Likewise, we observed the expected correlation between)*(P3 intrinsic value belief* and *(P6) cost belief* (M = 3.23; SD = 0.54). These two variables were significantly but negatively correlated. Due to the nature and direction of these two scales, this result indicates that children who reported high interest in and enjoyment of programming on the one hand also reported little negative consequences or emotions, such as exhaustion, on the other.

In addition, our results showed *(P2) self-concept* as well as *(P7) compliance and persistence* to be correlated significantly but negatively with *(P6) cost belief*. This reflects a relationship between little expected negative consequences or emotions associated with programming and, respectively, positive programming self-concept or high compliance and persistence.

Furthermore, we found a significant correlation between *(P3) intrinsic value belief* and *(P5) utility value belief*. This means that students who reported high enjoyment of and interest in programming also considered programming to be useful.

We also found *(P4) attainment value belief* (M = 1.81; SD = 0.72) to be highly significantly correlated with *(P5) utility value belief* and significantly correlated with *(P7) compliance and persistence*. Thus, there seems to be a relationship between placing high importance on programming one one hand, and regarding programming as useful and working on programming tasks thoroughly and resiliently on the other.

The results also indicated *(P1) previous programming experience and understanding* (M = 1.66; SD = 0.57) to be significantly correlated with *(P2) self-concept* and highly significantly correlated with *(use) frequency of computer/tablet use at home, in school, or elsewhere* (M = 2.79; SD = 0.90). This result indicates that students who already have some experience in and understanding of programming also have a positive programming self-concept and tend to use computers or tablets more frequently.

Contrary to our expectations, we found no significant correlations between *CT performance at pretest* (M = 11.55; SD = 3.61), *posttest* (M = 14.22; SD = 3.79), or *CT gain* (M = 2.96; SD = 4.53), and any of the self-concept, motivational beliefs, or attitude variables. This may indicate a lack of relationship between self-concept, motivational beliefs, and attitude variables, but may also be related to other reasons, which we discuss in section 4.1 on potential limitations of the current study.

4 DISCUSSION

This pilot study set off to develop a new questionnaire on self-concept, motivational beliefs, and attitudes towards programming. We modeled several subscales of the instrument on existing instruments for other subjects and applied it in a first pilot study to evaluate the general feasibility of our approach.

As shown by the results of the correlation analysis (see Table 1), academic *self-concept with regard to programming* correlated significantly with other subscales, namely with *intrinsic value belief*, *utility value belief*, *cost belief*, and self-reported *compliance and persistence*. This is not surprising, considering the central role self-concept plays as a prerequisite for achievement in a subject and as a deciding factor in whether a person attempts to succeed in a subject or not [28].

When examining the correlation pattern for the four motivational value beliefs – *intrinsic value belief*, *attainment value belief*, *utility value belief*, and *cost belief* – it is worth noting that they correlate significantly with each other and other variables such as

Table 1: Correlation between questionnaire variables, frequency of computer/tablet use, and CTt performance

Variable name	(P1)	(P2)	(P3)	(P4)	(P5)	(P6)	(P7)	(use)	(prCTt)	(poCTt)
(P1) self-reported programming understanding										
(P2) programming ability self-concept	.415*									
(P3) programming intrinsic value belief	.358	.414*								
(P4) programming attainment value belief	.064	.312	.225							
(P5) programming utility value belief	.210	.391*	.386*	.621**						
(P6) programming cost belief	-.175	-.544**	-.373*	-.253	-.102					
(P7) programming compliance and persistence	.057	.594**	.336	.407*	.310	-.483**				
(use) frequency of computer/tablet use	.472**	.188	.393	-.016	.114	-.026	.168			
(prCTt) CTt performance before training	-.156	-.278	.023	-.069	.012	.322	-.207	.005		
(poCTt) CTt performance after training	.170	-.058	.191	-.162	.068	.102	-.254	.066	.208	
(gainCTt) CTt performance gain	.259	.172	.137	-.077	.045	-.178	-.038	.048	-.623**	.635**

* Correlation is significant at the 0.05 level (2-tailed).
** Correlation is significant at the 0.01 level (2-tailed).

self-concept and *compliance and persistence*. Hence, they seem to constitute an integral part of the questionnaire.

Overall, the pattern of correlations among the subscales for self-concept, motivational beliefs, and attitude towards programming was very similar to the pattern observed for the same constructs on mathematics. In turn, this implies similar relationships between the variables within the newly developed questionnaire on academic self-concept, motivational beliefs, and attitudes towards programming as compared to the validated questionnaire on mathematics. Accordingly, this is first evidence for the validity of the newly developed questionnaire.

Unsurprisingly, we found self-reported *previous experience and understanding of programming* to be significantly correlated with *frequency of computer/tablet use at home, in school, or elsewhere*. This may imply students might have uninformed assumptions about computer or tablet use being closely connected with programming skills. On the other hand, this result may also reflect that students who already have some previous programming experience also are the ones who use computers or tablets more frequently. Because of the correlational results, no causal interpretation is possible.

Because the proposed instrument for assessing self-concept, motivational beliefs, and attitude towards programming is based on existing instruments for other school subjects, it allows for these attitudinal constructs to be assessed in relation to and comparable with these other subjects. This is an asset of the new questionnaire, as it is so far not clear whether the working mechanisms underlying influences of academic self-concept, motivational beliefs, and attitude towards specific subjects are comparable across subjects or not. A questionnaire allowing for the assessment of these same constructs across different subjects would make it possible to investigate whether these factors have the same influence on programming as they do on other subjects, and whether attitudes might be correlated across subjects.

Considering the educational impact of students' academic self-concept, motivational beliefs, and attitude towards programming and the increasing importance of the topic itself, it would be desirable to assess these constructs reliably. A validated questionnaire – based on the results of the present pilot study – for measuring them would allow for the assessment of the effects of specific programming, CT, or computer science programs or curricula on these constructs, which have been observed to strongly influence students' educational achievement and academic trajectories in other subjects such as mathematics (e.g. [24, 38, 48, 51]). In turn, any changes of these variables may be regarded as an indicator for the quality of a program or curriculum in terms of teaching methodology and fostering of student motivation.

4.1 Limitations

Unlike with other subjects (e.g. [28, 51]), our expectations of finding correlations between self-concept, motivational beliefs, and attitude towards programming on one hand and CT performance variables on the other could not be confirmed. Importantly, this lack of a significant association could be caused by power limitations resulting

from the small sample size. Studies reporting this association for mathematics, for instance, usually had much larger sample sizes (e.g., [13]). Therefore, we are planning to further investigate the relationship between the variables of the developed questionnaire and performance in programming and CT in future studies with larger samples.

Furthermore, it needs to be acknowledged that students who attended the course were nominated by their teachers based on their school achievements. Therefore, the sample may not be representative for the overall student population but represent a sample biased towards better performing students.

Additionally, it needs to be acknowledged that the original questionnaire on which we based the development of the current one was intended for students in 5th grade or higher. Therefore, future studies should evaluate in more detail whether the comprehensibility of the respective items is given.

4.2 Future Work

To gather further evidence and overcome power limitations, we are planning to use the developed questionnaire in a follow-up study using a randomized controlled field-trial design. This means all students participating in the study are randomly assigned to either the CT course group or the control group and all parts of course and assessment take place in actual classroom environments and thus in the contextual unpredictability of real teaching and learning situations. In such a randomized controlled field trial, it would be desirable to assess questionnaire and CT performance data of about 200 participating elementary school students. In this way, the instrument can undergo a validation process and already be utilized to analyze differential course effects on the measured constructs between the course group and the control group.

4.3 Conclusion

Being rooted in evidence-based education research and modeled on existing standardized and validated scales, the developed questionnaire goes beyond the current state of the art of attitude assessment in computer science education research at the primary and secondary education level. The relationships between the subscales of the questionnaire appear to show a similar pattern as compared to relationships between the subscales of the questionnaire it is modeled on. This provides a first implication of evidence on the validity of the developed questionnaire. However, the small sample size of this pilot study combined with the lack of observations of significant correlations between actual performance and the questionnaire's subscales for self-concept, motivational beliefs, and attitude towards programming calls for further research to fully substantiate the validity of the newly developed questionnaire.

ACKNOWLEDGMENTS

For their support of this study through teaching, data processing, and providing us with helpful feedback, we would like to thank Johannes-Dominik Krauß, Helmut Fritsch, Jana Hoffstadt, and Julian Jabs. Our gratitude also goes to the entire team of "Wissenschaftliche Begleitung der Hector Kinderakademien" at the Hector Research Institute of Education Sciences and Psychology, whose dedicated support we could always rely on while planning and implementing this study.

This research was partly funded by the Hector Stiftung II and supported by the Leibniz-Institut für Wissensmedien. Luzia Leifheit is a doctoral student of the LEAD Graduate School and Research Network [GSC1028], a project of the Excellence Initiative of the German federal and state governments.

REFERENCES

[1] Jane M. Armstrong and Richard A. Price. 1982. Correlates and Predictors of Women's Mathematics Participation. *Journal for Research in Mathematics Education* 13, 2 (mar 1982), 99. https://doi.org/10.2307/748357

[2] Marc R. Benioff and Edward D. Lazowska. 2005. *Computational Science: Ensuring America's Competitiveness.* Technical Report. President's Information Technology Advisory Committee. http://vis.cs.brown.edu/docs/pdf/Pitac-2005-CSE.pdf

[3] Karen Brennan and Mitchel Resnick. 2012. *New frameworks for studying and assessing the development of computational thinking.* Technical Report. http://scratched.gse.harvard.edu/ct/files/AERA2012.pdf

[4] Brigitte Maria Brisson, Anna-Lena Dicke, Hanna Gaspard, Isabelle Häfner, Barbara Flunger, Benjamin Nagengast, and Ulrich Trautwein. 2017. Short Intervention, Sustained Effects: Promoting Students' Math Competence Beliefs, Effort, and Achievement. *American Educational Research Journal* 54, 6 (2017), 1048–1078. https://doi.org/10.3102/0002831217716084

[5] Jehanzeb R. Cheema. 2014. Some General Guidelines for Choosing Missing Data Handling Methods in Educational Research. *Journal of Modern Applied Statistical Methods* 13, 2 (nov 2014), 53–75. https://doi.org/10.22237/jmasm/1414814520

[6] Roch Chouinard, Thierry Karsenti, and Normand Roy. 2007. Relations among competence beliefs, utility value, achievement goals, and effort in mathematics. *British Journal of Educational Psychology* 77, 3 (2007), 501–517. https://doi.org/10.1348/000709906X133589

[7] Jaap J. A. Denissen, Nicole R. Zarrett, and Jacquelynne S. Eccles. 2007. I Like to Do It, I'm Able, and I Know I Am: Longitudinal Couplings Between Domain-Specific Achievement, Self-Concept, and Interest. *Child Development* 78, 2 (2007), 430–447. https://www.psychologie.hu-berlin.de/de/prof/perdev/pdf/Denissen_et_al_2007_achievement_self-concept_interests.pdf

[8] Peter J. Denning, Matti Tedre, and Pat Yongpradit. 2017. Misconceptions about computer science. *Commun. ACM* 60, 3 (feb 2017), 31–33. https://doi.org/10.1145/3041047

[9] Brian Dorn and Allison Elliott Tew. 2015. Empirical validation and application of the computing attitudes survey. *Computer Science Education* 25, 1 (2015), 1–36.

[10] Amanda M. Durik, Mina Vida, and Jacquelynne S. Eccles. 2006. Task values and ability beliefs as predictors of high school literacy choices: A developmental analysis. *Journal of Educational Psychology* 98, 2 (2006), 382–393. https://doi.org/10.1037/0022-0663.98.2.382

[11] Jacquelynne S. Eccles, T. F. Alder, R. Futterman, S. B. Goff, C. M. Kaczala, J. L. Meece, and C. Midgley. 1983. Expectancies, values, and academic behaviors. In *Achievement and achievement motivation.* Freeman, 75–146. https://ci.nii.ac.jp/naid/10020820462/

[12] Allison Elliott Tew, Brian Dorn, and Oliver Schneider. 2012. Toward a Validated Computing Attitudes Survey. In *Proceedings of the Ninth Annual International Conference on International Computing Education Research (ICER '12).* ACM, New York, NY, USA, 135–142. https://doi.org/10.1145/2361276.2361303

[13] Hanna Gaspard. 2017. Promoting Value Beliefs in Mathematics : A Multidimensional Perspective and the Role of Gender. January 2015 (2017). https://doi.org/10.15496/publikation-5241

[14] Hanna Gaspard, Isabelle Häfner, Cora Parrisius, Ulrich Trautwein, and Benjamin Nagengast. 2017. Assessing task values in five subjects during secondary school: Measurement structure and mean level differences across grade level, gender, and academic subject. *Contemporary Educational Psychology* 48 (2017), 67–84.

[15] Daniel Heersink and Barbara M. Moskal. 2010. Measuring High School Students' Attitudes Toward Computing. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education (SIGCSE '10).* ACM, New York, NY, USA, 446–450. https://doi.org/10.1145/1734263.1734413

[16] Andrew Hoegh and Barbara M Moskal. 2009. Examining science and engineering students' attitudes toward computer science. In *2009 39th IEEE Frontiers in Education Conference.* IEEE, 1–6.

[17] Jenefer Husman and Jonathan Hilpert. 2007. The intersection of students' perceptions of instrumentality, self-efficacy, and goal orientations in an online mathematics course. *Zeitschrift fur Pädagogische Psychologie* 21, 3-4 (2007), 229–239. https://doi.org/10.1024/1010-0652.21.3.229

[18] M Jeannette. 2010. Wing. 2010. *Computational thinking: What and why* (2010).

[19] Krisztián Józsa and George A. Morgan. 2017. *Reversed Items in Likert Scales: Filtering out Invalid Responders.* Technical Report 1. 7–25 pages. https://fac.ksu.edu.sa/sites/default/files/likert2_0.pdf

[20] Donald E Knuth. 1974. Computer science and its relation to mathematics. *The American Mathematical Monthly* 81, 4 (1974), 323–343.
[21] Alma E. Lantz and Gregory P. Smith. 1981. Factors influencing the choice of nonrequired mathematics courses. *Journal of Educational Psychology* 73, 6 (1981), 825–837. https://doi.org/10.1037/0022-0663.73.6.825
[22] Luzia Leifheit, Julian Jabs, Manuel Ninaus, Korbinian Moeller, and Klaus Ostermann. 2018. Programming unplugged: An evaluation of game-based methods for teaching computational thinking in primary school. In *Proceedings of the European Conference on Games-based Learning*, Vol. 2018-Octob.
[23] Colleen M Lewis, Ken Yasuhara, and Ruth E Anderson. 2011. Deciding to major in computer science: a grounded theory of students' self-assessment of ability. In *Proceedings of the seventh international workshop on Computing education research*. ACM, 3–10.
[24] Herbert W Marsh, Olaf Köller, Ulrich Trautwein, Oliver Lüdtke, and Jürgen Baumert. 2005. Academic self-concept, interest, grades, and standardized test scores: Reciprocal effects models of causal ordering. *Child Development* 76, 2 (2005), 397–416. https://doi.org/10.1111/j.1467-8624.2005.00853.x
[25] Benjamin Nagengast, Ulrich Trautwein, L. Francesca Scalas, Kit-Tai Hau, Herbert W. Marsh, and Man K. Xu. 2011. Who Took the "×" out of Expectancy-Value Theory? *Psychological Science* 22, 8 (2011), 1058–1066. https://doi.org/10.1177/0956797611415540
[26] Enrico Nardelli. 2019. Do we really need computational thinking? *Commun. ACM* 62, 2 (2019), 32–35. https://doi.org/10.1145/3231587
[27] Christoph Niepel, Matthias Stadler, and Samuel Greiff. 2019. Seeing Is Believing: Gender Diversity in STEM Is Related to Mathematics Self-Concept. *Journal of Educational Psychology* (2019). https://doi.org/10.1037/edu0000340
[28] J. Steve Oliver and Ronald D. Simpson. 1988. Influences of attitude toward science, achievement motivation, and science self concept on achievement in science: A longitudinal study. *Science Education* 72, 2 (1988), 143–155. https://doi.org/10.1002/sce.3730720204
[29] Seymour Papert. 1980. *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc.
[30] Tony Perez, Jennifer G. Cromley, and Avi Kaplan. 2014. The role of identity development, values, and costs in college STEM retention. *Journal of Educational Psychology* 106, 1 (2014), 315–329. https://doi.org/10.1037/a0034027
[31] Laurie Hart Reyes. 2005. Affective Variables and Mathematics Education. *The Elementary School Journal* 84, 5 (2005), 558–581. https://doi.org/10.1086/461384
[32] Iris M Riggs and Larry G Enochs. 1993. A microcomputer beliefs inventory for middle school students: Scale development and validation. *Journal of Research on Computing in Education* 25, 3 (1993), 383–390.
[33] Marcos Román-González. 2015. Computational Thinking Test: Design Guidelines and Content Validation. *Proceedings of EDULEARN15 Conference* July 2015 (2015), 2436–2444. https://doi.org/10.13140/RG.2.1.4203.4329
[34] Mark Sanders. 2009. STEM, STEM education, STEMmania. *The Technology Teacher* 68, 4 (2009), 20–26.
[35] Joseph L. Schafer and John W. Graham. 2002. Missing data: Our view of the state of the art. *Psychological Methods* 7, 2 (2002), 147–177. https://doi.org/10.1037/1082-989X.7.2.147
[36] Neil Selwyn. 1997. Students' attitudes toward computers: Validation of a computer attitude scale for 16–19 education. *Computers & Education* 28, 1 (1997), 35–41.
[37] Valerie J Shute, Chen Sun, and Jodi Asbell-Clarke. 2017. Demystifying computational thinking. *Educational Research Review* 22 (2017), 142–158.
[38] Ulrich Trautwein and Olaf Köller. 2005. Was lange währt, wird nicht immer gut. *Zeitschrift für Pädagogische Psychologie* 17, 3/4 (2005), 199–209. https://doi.org/10.1024//1010-0652.17.34.199
[39] Ulrich Trautwein and Oliver Lüdtke. 2009. Predicting homework motivation and homework effort in six school subjects: The role of person and family characteristics, classroom factors, and school track. *Learning and Instruction* 19, 3 (jun 2009), 243–258. https://doi.org/10.1016/j.learninstruc.2008.05.001
[40] Ulrich Trautwein, Oliver Lüdtke, Brent W Roberts, Inge Schnyder, and Alois Niggli. 2009. Different forces, same consequence: Conscientiousness and competence beliefs are independent predictors of academic effort and achievement. *Journal of Personality and Social Psychology* 97, 6 (2009), 1115–1128. https://doi.org/10.1037/a0017048
[41] Ulrich Trautwein, Benjamin Nagengast, Herbert W. Marsh, Hanna Gaspard, Anna-Lena Dicke, Oliver Lüdtke, and Kathrin Jonkmann. 2013. Expectancy-Value Theory revisited: From Expectancy-Value Theory to Expectancy-ValueS Theory? - PsycNET. *Theory driving research: New wave perspectives on self-processes and human development* (2013), 233–249. https://psycnet.apa.org/record/2013-21161-010
[42] Katerina Tsarava, Luzia Leifheit, Manuel Ninaus, Marcos Román-González, Martin Butz, Jessika Golle, Ulrich Trautwein, and Korbinian Moeller. 2019. Cognitive Correlates of Computational Thinking in Elementary School: Evaluation of a Blended Unplugged/Plugged-In Course. In *Proceedings of the 14th Workshop in Primary and Secondary Computing Education - WiPSCE '19*. ACM Press, New York, New York, USA.
[43] Katerina Tsarava, Korbinian Moeller, and Manuel Ninaus. 2018. Training Computational Thinking through board games: The case of Crabs & Turtles. *International Journal of Serious Games* 5, 2 (2018), 25–44. https://doi.org/10.17083/ijsg.v5i2.248
[44] Katerina Tsarava, Korbinian Moeller, Niels Pinkwart, Martin Butz, Ulrich Trautwein, and Manuel Ninaus. 2017. Training computational thinking: Game-based unplugged and plugged-in activities in primary school. In *Proceedings of the 11th European Conference on Games Based Learning*. 687–695.
[45] Joke Voogt, Ola Erstad, Chris Dede, and Punya Mishra. 2013. Challenges to learning and schooling in the digital networked world of the 21st century. *Journal of computer assisted learning* 29, 5 (2013), 403–413.
[46] Jennifer Wang, Sepehr Hejazi Moghadam, and Juliet Tiffany-Morales. 2017. Social Perceptions in Computer Science and Implications for Diverse Students. In *Proceedings of the 2017 ACM Conference on International Computing Education Research - ICER '17*. ACM Press, New York, New York, USA, 47–55. https://doi.org/10.1145/3105726.3106175
[47] Heike Wendt, Wilfried Bos, Christoph Selter, Olaf Köller, Knut Schwippert, and Daniel Kasper. 2016. *TIMMS 2015 - Mathematische und naturwissenschaftliche Kompetenzen von Grundschulkindern in Deutschland im interntionalen Vergleich*.
[48] Allan Wigfield and Jenna Cambria. 2010. Students' achievement values, goal orientations, and interest: Definitions, development, and relations to achievement outcomes. In *Developmental Review*. Vol. 30. Academic Press, 1–35. https://doi.org/10.1016/j.dr.2009.12.001
[49] Allan Wigfield and Jacquelynne S. Eccles. 1992. The development of achievement task values: A theoretical analysis. *Developmental Review* 12, 3 (sep 1992), 265–310. https://doi.org/10.1016/0273-2297(92)90011-P
[50] Allan Wigfield, Stephen Tonks, and Susan Lutz Klauda. 2009. Expectancy-Value Theory. (sep 2009), 69–90. https://doi.org/10.4324/9780203879498-10
[51] Jesse L.M. Wilkins. 2004. Mathematics and science self-concept: An international investigation. *Journal of Experimental Education* 72, 4 (2004), 331–346. https://doi.org/10.3200/JEXE.72.4.331-346
[52] Jeannette M. Wing. 2014. Computational Thinking Benefits Society. http://socialissues.cs.toronto.edu/index.html?p=279.html
[53] Aman Yadav, Chris Mayfield, Ninger Zhou, Susanne Hambrusch, and John T Korb. 2014. Computational thinking in elementary and secondary teacher education. *ACM Transactions on Computing Education (TOCE)* 14, 1 (2014), 5.

Manuscript 3

Title:
SCAPA: Development of a Questionnaire Assessing Self-Concept and Attitudes Toward Programming

Authors:
Luzia Leifheit, Katerina Tsarava, Manuel Ninaus, Klaus Ostermann, Jessika Golle, Ulrich Trautwein, and Korbinian Moeller

Published in:
Proceedings of the 25th Conference on Innovation and Technology in Computer Science Education (ITiCSE). ACM.

https://doi.org/10.1145/3341525.3387415

Time of publication:
June 2020

SCAPA: Development of a Questionnaire Assessing Self-Concept and Attitudes Toward Programming

Luzia Leifheit
LEAD Graduate School and Research Network, University of Tübingen
Leibniz-Institut für Wissensmedien
Tübingen, Germany
luzia.leifheit@uni-tuebingen.de

Katerina Tsarava
Hector Research Institute of Education Sciences and Psychology
Leibniz-Institut für Wissensmedien
LEAD, University of Tübingen
Tübingen, Germany
k.tsarava@iwm-tuebingen.de

Manuel Ninaus
Leibniz-Institut für Wissensmedien
LEAD, University of Tübingen
Tübingen, Germany
m.ninaus@iwm-tuebingen.de

Klaus Ostermann
Department of Computer Science
LEAD, University of Tübingen
Tübingen, Germany
klaus.ostermann@uni-tuebingen.de

Jessika Golle
Hector Research Institute of Education Sciences and Psychology
LEAD, University of Tübingen
Tübingen, Germany
jessika.golle@uni-tuebingen.de

Ulrich Trautwein
Hector Research Institute of Education Sciences and Psychology
LEAD, University of Tübingen
Tübingen, Germany
ulrich.trautwein@uni-tuebingen.de

Korbinian Moeller
Centre for Mathematical Cognition, Loughborough University
Leibniz-Institut für Wissensmedien
LEAD, University of Tübingen
Loughborough, UK
k.moeller@lboro.ac.uk

ABSTRACT

There is a constantly growing number of initiatives asserting the relevance of programming already in primary education and offering respective interventions with the goal to foster interest in and positive attitudes toward programming. To evaluate to what extent this goal is achieved, assessing students' attitudes toward programming reliably is indispensable. However, there still is a need for validated instruments for assessing this in elementary school students. This seems particularly relevant as self-concept and attitudes toward a school subject were repeatedly observed to be significant predictors of learning motivation and achievement. The newly developed Self-Concept and Attitude toward Programming Assessment (SCAPA) is based on existing instruments for assessing students' self-concept and attitude toward mathematics. SCAPA measures aspects of students' self-concept and attitudes toward programming on seven scales: i) self-reported previous programming experience and understanding, ii) self-concept, iii) intrinsic value belief, iv) attainment value belief, v) utility value belief, vi) cost belief, and vii) compliance and persistence. We administered SCAPA to 197 elementary school students between seven and ten years of age in the context of an evaluation of a computational thinking intervention. Data were analyzed for reliability (i.e., internal consistency on item and scale level) and construct validity (by means of confirmatory factor analysis). Results indicated good reliability for all scales except for the self-reported previous programming experience and understanding scale. Overall, these results reflect SCAPA's suitability for assessing different aspects of elementary school students' self-concept and attitudes toward programming.

KEYWORDS

computer science education, computing education, assessment, instrument development, K-12 education, primary education

ACM Reference Format:
Luzia Leifheit, Katerina Tsarava, Manuel Ninaus, Klaus Ostermann, Jessika Golle, Ulrich Trautwein, and Korbinian Moeller. 2020. SCAPA: Development of a Questionnaire Assessing Self-Concept and Attitudes Toward Programming. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '20), June 15–19, 2020, Trondheim, Norway.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3341525.3387415

ITiCSE '20, June 15–19, 2020, Trondheim, Norway

ACM ISBN 978-1-4503-6874-2/20/06…$15.00
https://doi.org/10.1145/3341525.3387415

1 INTRODUCTION

Over the last decade, the relevance of programming in secondary and even primary education has become more recognized. On the one hand, more and more countries are integrating programming into school curricula already in elementary school (e.g., [3]). On

the other hand, initiatives such as code.org aim at broadening students' active engagement with programming (e.g., [5, 6, 16]). One of the most common goals of these initiatives is to foster interest, confidence, and positive attitudes toward programming (e.g., [6]).

For other school subjects, attitudes toward the subject were repeatedly observed to be significant predictors of students' educational achievement [18, 27, 29, 31]. In particular, for mathematics and science, positive self-concept and attitude have been found to reliably predict academic achievement and to motivate students to persist and advance in their studies of the respective subjects (e.g., [2, 17, 20, 31]). These longitudinal findings illustrate how strongly self-concept and attitude at a younger age influence educational outcomes later in life [18, 20], with motivational factors from as early as elementary school predicting achievement and course selection in late high school [29].

Taking into account the important role self-concept and attitude play for educational outcomes, it seems desirable to also assess these factors for programming. Measuring these factors reliably would allow for assessing the development of students' self concept and attitudes toward programming and thus for evaluating the actual effects of above mentioned initiatives to determine to what extent the respective interventions are successful. Such an assessment tool could prove beneficial for educators as well as educational policymakers when developing, evaluating, and adapting computing courses or curricula.

However, there still is a need for validated instruments for assessing students' self-concept and attitudes toward programming, in particular for elementary school students. Therefore, we developed the Self-Concept and Attitude toward Programming Assessment (SCAPA) and evaluated it for its reliability and construct validity in the present study.

1.1 Instruments Assessing Students' Attitude toward Computing

Several questionnaires for measuring students' attitudes toward computing have been proposed and evaluated in recent years. Generally, these questionnaires fall into one of two categories: the first type of questionnaire has a more narrowly defined focus on computing, while the latter more broadly assesses attitudes toward computer use.

One of the instruments from the first category is the Computing Attitudes Survey (CAS) for assessing college students' attitudes toward problem solving and their beliefs about the nature of knowledge within computer science [7]. Other instruments in this category include the questionnaire for measuring high school students' attitudes toward computer science and information technology [13] and the Computer Science Attitude Survey for undergraduate science and engineering students [14].

The other category includes questionnaires measuring attitudes toward computer use, such as the Microcomputer Beliefs Inventory (MBI) [23] for assessing middle school students' self-efficacy and outcome expectancy beliefs regarding computer use [23]. Another questionnaire from this category measures high school students' attitudes toward interacting with computers [24].

Due to the instruments' respective focus areas and target groups, questionnaires from neither of these categories seem well-suited for assessing elementary school students' self-concept and attitudes toward programming. Existing domain-specific instruments developed for measuring attitudes toward computing – as is the case for questionnaires from the first category – are mainly aimed at high school and college students with previous computing experience (e.g. CAS, [7]). Accordingly, item phrasing is not suited for the target group of elementary school students because it is too complex. Questionnaires from the second category are also aimed at younger students (i.e., at middle and high school students) and do not require previous computing experience. However, they do measure attitudes toward computer use rather than computing or programming, which makes them too broad for specifically assessing students' attitudes toward computing and programming activities.

Against this background, we developed and evaluated a new questionnaire – the Self-Concept and Attitude toward Programming Assessment (SCAPA), with more age-appropriate item phrasing for elementary school students and a focus on programming rather than general attitudes toward computer use.

1.2 Aspects Measured by SCAPA

All SCAPA scales except one are based on existing scales for mathematics [4, 10, 27], which have already been adapted and validated for other school subjects, such as biology, physics, or English [11].

SCAPA is designed to assess the following aspects of students' self-concept and attitude toward programming on seven scales:

i) *Self-reported previous experience and understanding of programming* is assessed on the first scale. Unlike the other SCAPA scales, this scale is not modeled on the existing questionnaire on mathematics, but has been added to provide relevant information on students' background with regard to previous programming experience.

ii) The second scale measures *self-concept with regard to programming*. Students' self-concept with regard to a subject was defined as confidence in their own ability to do well in the respective subject [22, 31]. Positive self-concept was repeatedly found to significantly predict achievement: when students have no confidence in their ability to do well in a subject, they tend to not even try [20].

Scales iii) through vi) appraise attitudes toward programming as specified in expectancy-value theory, which has been developed to model achievement motivation [9] and has been established in empirical research (e.g., [19, 28, 30]).

iii) *Intrinsic value belief about programming* is assessed on the third scale. Intrinsic value belief about a subject reflects how much a student enjoys and is interested in the subject. Positive intrinsic value belief can increase commitment to learning activities [20] and is a predictor for leisure-time engagement with the respective subject [8, 19].

iv) Scale four measures *attainment value belief about programming*, which represents the importance a student places on a subject [10], in this case on programming.

v) *Utility value belief about programming* is assessed on the fifth scale. A student's utility value belief about a subject reflects the expectation of the respective subject's usefulness in areas such as everyday life, school, or social life [10].

vi) The sixth scale measures *cost belief about programming*. The cost belief individuals hold about a subject reflects negative consequences – such as high effort and negative emotions – they fear as a result of engaging in the subject [10, 21].

vii) Last, *compliance and persistence with regard to programming* are assessed on the seventh scale. Compliance and persistence in a subject reflect thoroughness in working on a subject-related task as well as resilience in the face of challenges in that subject [27].

1.3 Aim of the Present Study

This study aims at providing first results on the internal consistency and construct validity of SCAPA for assessing students' self-concept and attitude toward programming. Data were collected and analyzed for reliability (i.e., internal consistency on item and scale level) and construct validity (by means of confirmatory factor analysis).

2 METHODS

2.1 Participants

In this study, we administered SCAPA to 197 elementary school students (*female* = 47) between 7 and 10 years of age (M = 8.83; SD = 0.73) attending 3rd and 4th grade of German elementary school. All these children participated in a computational thinking course offered as part of an extracurricular enrichment program, for which they were nominated by their teachers. All students participated in the study voluntarily. Before participation in the study, we obtained informed consent from all students as well as their parents or legal guardians. The study was approved by the local ethics committee.

2.2 Study Design

In a field study, we administered SCAPA to students in the form of anonymized paper-pencil based questionnaires. One week before the first session of the computational thinking course, all participating students attended a test session taking place in the regular course classrooms. Ten minutes were scheduled for students to fill in the SCAPA questionnaire.

2.3 The SCAPA Instrument

SCAPA assesses students' self-concept and attitude toward programming on seven scales, four of which comprise subscales to allow for a differentiated assessment of the scales' various facets.

The scales for measuring *self-concept*, *expectancy-value beliefs*, as well as *compliance and persistence regarding programming* were modeled on an existing validated questionnaire for assessing the same aspects for mathematics [4, 10, 27]. Items were adapted using the word "programming" as the focal term instead of the word "mathematics". The same procedure was used to adapt the mathematics instrument for biology, physics, English, and German [11].

To adapt the instrument appropriately, we considered relevant terms to represent the focus area of the questionnaire. Even though we considered the terms *programming* (German *Programmieren*) and *computer science* (German *Informatik*), we did not consider *computing* and *computational thinking*, as neither of these terms have a close translation or commonly used equivalent in German.

To ensure comprehensibility for SCAPA's target group, we asked children attending a session of an earlier computational thinking intervention whether they had heard the words *Informatik* or *Programmieren* before and asked them to explain what they understood each of the two words to mean. While only few of them had heard the word *Informatik* before and none were able to explain its meaning, all of the children had already heard the word *Programmieren* and were able to either provide a basic explanation or give examples of what programming activities could look like. This is in line with findings regarding the target group's understanding of the word *Programmieren*: German 3rd and 4th grade students most commonly associate programming with objects – such as computers, apps, viruses, or robots – as well as actions – such as creating, typing, inventing, adjusting, or combining [12]. There is no significant difference between associations stated by girls and boys [12]. Thus, we decided to use the word *programming* (German *Programmieren*) as the focal term of the instrument.

The scale for *previous experience and understanding of programming* was not based on the existing instrument for mathematics, but designed exclusively for the new questionnaire to provide additional information on students' previous exposure to programming.

The original scales and their items are in German, but in this article English translations are presented. This is the structure of the seven SCAPA scales and their subscales:

- *P1 (self-reported previous programming experience and understanding)*: four items, e.g. "I know what the word programming means."
- *P2 (programming ability self-concept)*: four items, e.g. "I'm good at programming."
- *P3 (programming intrinsic value belief)*: four items, e.g. "I enjoy programming."
- *P4 (programming attainment value belief)*: seven items across two subscales:
  - *P4ac (achievement)*: three items, e.g. "Being good at programming means a lot to me."
  - *P4ps (personal significance)*: four items, e.g. "Programming is very important to me personally."
- *P5 (programming utility value belief)*: twelve items across four subscales:
  - *P5dl (daily life)*: three items, e.g. "Programming skills have many benefits in my daily life."
  - *P5fj (future job)*: three items, e.g. "Good programming skills will help me in my future job."
  - *P5sc (school)*: four items, e.g. "Doing well in programming brings many advantages at school."
  - *P5so (social)*: two items, e.g. "If I know a lot about programming, I will leave a good impression on my classmates."
- *P6 (programming cost belief)*: eleven items across three subscales:
  - P6ef (effort): four items, e.g. "Programming is exhausting to me."
  - *P6em (emotional)*: four items, e.g. "When I deal with programming, I get annoyed."
  - *P6op (opportunity)*: three items, e.g. "To be successful at programming, I have to give up other activities I enjoy."
- *P7 (self-reported programming compliance and persistence)*: eight items across two subscales:

- *P7co (compliance)*: four items, e.g. "I always try to complete my programming tasks."
- *P7pe (persistence)*: four items, e.g. "Even with difficult programming tasks I don't give up quickly."

All items for these scales require four-point Likert-type ratings (1 = agree completely; 2 = rather agree; 3 = rather do not agree; 4 = do not agree at all). Scales *P2*, *P4*, and *P7* include reversed items (e.g. "Programming is not meaningful to me" rather than "Programming is meaningful to me") to evaluate response biases by detecting inconsistencies between responses to positively phrased items and potentially contradicting responses to their negatively phrased counterpart items [15]. Responses to these items were recoded (rating 1 := 4; 2 := 3; 3 := 2; 4 := 1) prior to analyses to ensure the same directionality as the positively phrased items.

2.4 Statistical Analysis

2.4.1 Reliability Analysis. To analyse SCAPA's internal consistency, i.e. its coherence in measuring constructs as assessed by its scales, we calculated descriptive statistics and Cronbach's alpha for each of the scales and subscales using IBM SPSS Statistics 25.

2.4.2 Confirmatory Factor Analysis. By means of confirmatory factor analysis (CFA) conducted using Mplus 8, we evaluated the validity of the respective scales and subscales. Therefore, we considered the model fit indices chi-squared test, RMSEA, SRMR, and CFI to evaluate how well the respective model for each (sub)scale fits to the data. Additionally, the confirmatory factor analysis provided each item's factor loadings on the respective (sub)scale, i.e. to what extent each item contributes to its associated (sub)scale's variance.

3 RESULTS

All available data from each of the 197 participants were used for the analyses presented in this article. Reported differences in sample sizes for scales or subscales (see Table 1) are the result of missing values due to nonresponse or invalid response to individual items.

3.1 Reliability

We performed analyses of internal consistency on item and scale level. Descriptive results and Cronbach's α for each of the seven scales and eleven subscales are presented in Table 1. Except from *P1 (self-reported previous programming experience and understanding)* and *P7pe (programming persistence)*, reliability analyses of all scales and subscales yielded Cronbach's α between .612 and .923 (see Table 1), ranging from moderate to strong reliability (e.g., [26]).

Moreover, Cronbach's α for each individual scale (see Table 1) did not change substantially when any of the items in the scale were excluded. The only exception to this is the last item of *P1 (self-reported previous programming experience and understanding)*: "I have programmed before". In case the item were excluded, the scale's Cronbach's α would increase to .642.

3.2 Confirmatory Factor Analysis

Model fit indices indicated mixed results (see Table 1). Subscales *P4ac (programming attainment value belief: achievement)*, *P5dl (programming utility value belief: daily life)*, *P5fj (programming utility value belief: future job)*, *P5so (programming utility value belief: social)* and *P6op (programming cost belief: opportunity)* are just-identified with degrees of freedom ≤ 0. Therefore, model fit for these subscales cannot be assessed.

Scale *P3 (programming intrinsic value belief)* was found to have high model fit according to all indices considered in the analysis. However, while model fit for (sub-)scales *P1 (self-reported previous programming experience and understanding)*, *P4ps (programming attainment value belief: personal significance)*, *P5sc (programming utility value belief: school)*, *P6ef (programming cost belief: effort)*, *P6em (programming cost belief: emotional)*, and *P7co (programming compliance)* appeared to be adequate according to chi-squared, SRMR, and CFI results, their model fit is not so good when considering RMSEA. For *P2 (programming ability self-concept)* and *P7pe (programming persistence)*, insufficient model fit indices were observed.

Standardized factor loadings were calculated to determine how strong the relationship is between each (sub-)scale and its contributing items (see Table 2). Out of SCAPA's 50 items, 41 load highly on their respective scale (i.e., loadings $\geq .50$), 6 moderately (i.e., loadings $\geq .30$), and only 3 items showed low loadings on their respective scale (i.e., loadings $< .30$) (factor loadings interpreted as correlation coefficients, e.g. [1]). *P3 (programming intrinsic value belief)*, *P4ac (programming attainment value belief: achievement)*, *P5dl (programming utility value belief: daily life)*, *P5fj (programming utility value belief: future job)*, *P5so (programming utility value belief: social)*, *P6ef (programming cost belief: effort)*, *P6em (programming cost belief: emotional)*, *P6op (programming cost belief: opportunity)*, and *P7co (programming compliance)* all consist exclusively of items with high factor loadings. *P1 (self-reported previous programming experience and understanding)* contains two items with high and two with moderate factor loadings, *P5sc (programming utility value belief: school)* has three items with high factor loadings and one item contributing moderately. *P2 (programming ability self-concept)* comprises two items with high factor loadings, one with moderate, and one with low loading. *P4ps (programming attainment value belief: personal significance)* contains three items with high factor loadings and one with a low loading. *P7pe (programming persistence)* has one high-loading item, two items with moderate factor loadings, and one item with low loading.

4 DISCUSSION AND OUTLOOK

Overall, SCAPA's scales showed sufficient to very good reliability in terms of internal consistency, with the only exceptions being *P1 (self-reported previous programming experience and understanding)* and *P7pe (programming persistence)*. These results indicate SCAPA being largely consistent with regard to how well its (sub)scales assess the respective constructs.

Moreover, CFA results provide first evidence for the construct validity of SCAPA (sub)scales with good model fit indices for most (sub)scales. This indicates the models for most of the (sub)scales fit the data well. For some, observed RMSEA indicated not so good model fit in spite of other model fit indices indicating adequate to very good model fit. Most probably, this discrepancy is a consequence of specific characteristics of the models tested and our sample. Models with smaller degrees of freedom and sample sizes up to N = 200 are known to often result in inappropriately high RMSEA values (e.g., [25]). Therefore, RMSEA may be considered with

Table 1: Scale- and subscale-level descriptive statistics, reliability (as indicated by Cronbach's α), and model fit indices (chi-squared, RMSEA, SRMR, CFI) for SCAPA

(Sub-)scale	N	# of items	M	SD	Cr. α	df	χ^2	RMSEA	SRMR	CFI
P1 (self-reported programming understanding)	182	4	3.419	.481	.547	2	8.317	.129	.039	.936
P2 (programming ability self-concept)	180	4	3.318	.128	.688	2	29.918	.274	.093	.843
P3 (programming intrinsic value belief)	178	4	3.496	.035	.923	2	1.877	.000	.007	1.000
P4 (programming attainment value belief)	170	7	3.408	.100	.802	13	32.605	.091	.052	.959
P4ac (programming attainment value belief: achievement)	178	3	3.466	.022	.877	0*	-	-	-	-
P4ps (programming attainment value belief: personal significance)	173	4	3.373	.129	.612	2	8.314	.131	.043	.937
P5 (programming utility value belief)	162	12	3.392	.097	.873	48	82.618	.063	.043	.959
P5dl (programming utility value belief: daily life)	177	3	3.416	.025	.710	0*	-	-	-	-
P5fj (programming utility value belief: future job)	168	3	3.423	.030	.854	0*	-	-	-	-
P5sc (programming utility value belief: school)	169	4	3.364	.166	.733	2	7.044	.118	.033	.977
P5so (programming utility value belief: social)	172	2	3.291	.008	.648	<0*	-	-	-	-
P6 (programming cost belief)	155	11	2.539	.360	.819	41	88.599	.081	.054	.927
P6ef (programming cost belief: effort)	164	4	2.948	.161	.722	2	6.729	.116	.030	.968
P6em (programming cost belief: emotional)	164	4	2.250	.217	.816	2	6.742	.116	.026	.980
P6op (programming cost belief: opportunity)	175	3	2.307	.087	.782	0*	-	-	-	-
P7 (programming compliance and persistence)	166	8	3.592	.208	.779	19	79.900	.136	.073	.876
P7co (programming compliance)	169	4	3.703	.042	.845	2	9.808	.150	.029	.971
P7pe (programming persistence)	169	4	3.469	.260	.485	2	26.348	.266	.097	.648

Note. N = number of cases; M = mean; SD = standard deviation; Cr. α = Cronbach's α; df = degrees of freedom; χ^2 = chi-squared value; RMSEA = root mean square error of approximation; SRMR = standardized root mean square residual; CFI = comparative fit index, *please note that due to the limited number of df, no model fit indices were estimated

caution in these cases [25] and more emphasis should be placed on the other measures of model fit.

Based on the model fit and reliability results for the (sub)scales, we derive suggestions for SCAPA's future use. Despite observed insufficiencies for reliability and construct validity, we suggest to keep *P7pe (programming persistence)* as a subscale for *P7 (self-reported programming compliance and persistence)* to allow for comparability with the same aspects for other school subjects [11].

For scale *P1 (self-reported previous programming experience and understanding)*, however, we suggest excluding the last item. This seems reasonable semantically as the remaining items "I know the word programming", "I know what the word programming means", and "I can explain what programming means" differ significantly in meaning from the to-be-excluded "I have programmed before". Moreover, deleting this item would improve the scale's reliability.

While these first results on SCAPA's reliability and construct validity are promising, further development and evaluation is necessary to substantiate the suitability of SCAPA. Amongst others, future development of SCAPA should aim for larger sample sizes to overcome limitations as experienced with RMSEA for some of the (sub)scales. Additionally, further validation efforts may include empirical evaluation of SCAPA's concurrent validity using one or several of the existing instruments presented in 1.1 to correlate SCAPA with. Nevertheless, while comparison to these instruments seems desirable, it needs to be considered that aforementioned instruments are not particularly well-suited for either the focus area of programming or for the target group of elementary school students. In addition, another validation study on SCAPA may be conducted with native English-speaking elementary school students to evaluate the validity of the English version of SCAPA.

4.1 Limitations

It needs to be acknowledged that all students who participated in the study to be assessed using SCAPA had previously been nominated by their teachers to attend an extracurricular enrichment program based on their school achievements. Therefore, the sample may not be representative for the overall student population but represent a sample biased toward better performing students.

4.2 Conclusion

Results of the present study provided first promising indications for SCAPA's reliability in terms of internal consistency and construct validity of its (sub)scales. For further validation efforts, we recommend using SCAPA in the form presented in this article to allow for further evaluation of scales, subscales, and items.

However, when using SCAPA to assess students' self-concept and attitudes toward programming, we suggest the following modifications: The last item for *P1 (self-reported previous programming experience and understanding)* should be dropped. Instead, we would include three open questions at the beginning of the questionnaire, asking students "Have you programmed before?", "If yes, what have you programmed? Give an example.", and "Please explain what you think the word programming means.". These questions would allow for collecting more precise information on students' previous programming experience and their comprehension of the term.

Table 2: Standardized factor loadings using the total sample for all SCAPA items (English translation)

Item	factor loading
P1 (self-reported programming understanding)	
I know the word programming	.467
I know what the word programming means	.652
I can explain what programming means	.743
I have programmed before	.328
P2 (programming ability self-concept)	
I simply have no talent for programming	.407
I'm not very good at programming	.279
I'm good at programming	.916
Programming is easy for me	.747
P3 (programming intrinsic value belief)	
Programming is fun to me	.882
I enjoy programming	.947
I simply like programming	.871
I enjoy dealing with programming topics	.787
P4ac (programming attainment value belief: achievement)	
It is important to me to be good at programming	.802
Being good at programming means a lot to me	.879
Programming achievements are important to me	.847
P4ps (programming attainment value belief: personal significance)	
Remembering what I learn about programming matters a lot to me	.694
Programming is not meaningful to me	.541
Programming is very important to me personally	.723
To be honest, I don't care about programming	.173
P5dl (programming utility value belief: daily life)	
Programming skills have many benefits in my daily life	.641
Programming skills come in handy in everyday life and leisure time	.707
What we learn about programming is directly beneficial in everyday life	.653
P5fj (programming utility value belief: future job)	
Good programming skills will help me in my future job	.843
For my future working life, it will pay off to be good at programming	.799
Programming skills will be helpful for my future career	.788
P5sc (programming utility value belief: school)	
Being good at programming will help me in the remaining years at school	.872
Being good at programming pays off because it is simply needed at school	.811
It is worth making an effort at programming because it will save me a lot of trouble at school during the coming years	.428
Doing well in programming brings many advantages at school	.654
P5so (programming utility value belief: social)	
Good programming skills count for something with my classmates	.691
If I know a lot about programming, I will leave a good impression on my classmates	.707
P6ef (programming cost belief: effort)	
Programming is exhausting to me	.587
After doing programming tasks, I often feel completely drained	.585
Dealing with programming drains a lot of my energy	.731
Learning programming exhausts me	.716
P6em (programming cost belief: emotional)	
Programming is a real burden to me	.702
Dealing with programming makes me really nervous	.738
I'd rather not deal with programming because it only worries me	.668
When I deal with programming, I get annoyed	.813
P6op (programming cost belief: opportunity)	
To be successful at programming, I have to give up other activities I enjoy	.730
I have to give up a lot to be good at programming	.842
To be good at programming, I would have to sacrifice a lot of leisure time	.645
P7co (programming compliance)	
I'm willing to make an effort at programming	.732
I do my best with programming tasks	.779
I work seriously on all programming tasks	.706
I always try to complete my programming tasks	.824
P7pe (programming persistence)	
Even if programming tasks are difficult, I make an effort	.698
If I don't quickly find a solution for a programming task, I stop working on it	.137
Even with difficult programming tasks, I don't give up quickly	.444
When programming tasks get difficult, I stop working on them	.420

Considering our findings and the suggested modifications, SCAPA could prove a useful assessment tool for developing, evaluating, and adapting computing courses or curricula. Therefore, it could be helpful for educators as well as educational policymakers and, consequently, could benefit students partaking in educational activities that have been evaluated and improved with the aim of fostering positive self-concept and attitude with regard to programming.

ACKNOWLEDGMENTS

For their support of this study, we would like to thank the entire team of "Wissenschaftliche Begleitung der Hector Kinderakademien" at the Hector Research Institute of Education Sciences and Psychology, whose dedicated support we could always rely on while planning and implementing this study.

This research was partly funded by the Hector Stiftung II and supported by the Leibniz-Institut für Wissensmedien. Luzia Leifheit is a doctoral student of the LEAD Graduate School and Research Network [GSC1028], a project of the Excellence Initiative of the German federal and state governments.

REFERENCES

[1] [n. d.]. Pearson's Correlation Coefficient. https://www.statisticssolutions.com/pearsons-correlation-coefficient/
[2] Jane M. Armstrong and Richard A. Price. 1982. Correlates and Predictors of Women's Mathematics Participation. *Journal for Research in Mathematics Education* 13, 2 (mar 1982), 99. https://doi.org/10.2307/748357
[3] Anja Balanskat and Katja Engelhardt. 2015. Computing our future: Computer programming and coding - Priorities, school curricula and initiatives across Europe.
[4] Brigitte Maria Brisson, Anna-Lena Dicke, Hanna Gaspard, Isabelle Häfner, Barbara Flunger, Benjamin Nagengast, and Ulrich Trautwein. 2017. Short Intervention, Sustained Effects: Promoting Students' Math Competence Beliefs, Effort, and Achievement. *American Educational Research Journal* 54, 6 (2017), 1048–1078. https://doi.org/10.3102/0002831217716084
[5] Code.org. [n. d.]. CS Fundamentals for grades K-5. https://code.org/educate/curriculum/elementary-school
[6] Code.org. [n. d.]. Hour of Code: Join the Movement. https://hourofcode.com/
[7] Brian Dorn and Allison Elliott Tew. 2015. Empirical validation and application of the computing attitudes survey. *Computer Science Education* 25, 1 (2015), 1–36.
[8] Amanda M. Durik, Mina Vida, and Jacquelynne S. Eccles. 2006. Task values and ability beliefs as predictors of high school literacy choices: A developmental analysis. *Journal of Educational Psychology* 98, 2 (2006), 382–393. https://doi.org/10.1037/0022-0663.98.2.382
[9] Jacquelynne S. Eccles, T. F. Alder, R. Futterman, S. B. Goff, C. M. Kaczala, J. L. Meece, and C. Midgley. 1983. Expectancies, values, and academic behaviors. In *Achievement and achievement motivation*. Freeman, 75–146. https://ci.nii.ac.jp/naid/10020820462/
[10] Hanna Gaspard. 2017. Promoting Value Beliefs in Mathematics : A Multidimensional Perspective and the Role of Gender. January 2015 (2017). https://doi.org/10.15496/publikation-5241
[11] Hanna Gaspard, Isabelle Häfner, Cora Parrisius, Ulrich Trautwein, and Benjamin Nagengast. 2017. Assessing task values in five subjects during secondary school: Measurement structure and mean level differences across grade level, gender, and academic subject. *Contemporary Educational Psychology* 48 (2017), 67–84.
[12] Katharina Geldreich, Alexandra Simon, and Elena Starke. 2019. Which Perceptions Do Primary School Children Have about Programming?. In *Proceedings of the 14th Workshop in Primary and Secondary Computing Education*. 1–7.
[13] Daniel Heersink and Barbara M. Moskal. 2010. Measuring High School Students' Attitudes Toward Computing. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education (SIGCSE '10)*. ACM, New York, NY, USA, 446–450. https://doi.org/10.1145/1734263.1734413
[14] Andrew Hoegh and Barbara M Moskal. 2009. Examining science and engineering students' attitudes toward computer science. In *2009 39th IEEE Frontiers in Education Conference*. IEEE, 1–6.
[15] Krisztián Józsa and George A. Morgan. 2017. *Reversed Items in Likert Scales: Filtering out Invalid Responders*. Technical Report 1. 7–25 pages. https://fac.ksu.edu.sa/sites/default/files/likert2_0.pdf
[16] Kodable. [n. d.]. Programming for Kids. https://www.kodable.com/
[17] Alma E. Lantz and Gregory P. Smith. 1981. Factors influencing the choice of nonrequired mathematics courses. *Journal of Educational Psychology* 73, 6 (1981), 825–837. https://doi.org/10.1037/0022-0663.73.6.825
[18] Herbert W Marsh, Olaf Köller, Ulrich Trautwein, Oliver Lüdtke, and Jürgen Baumert. 2005. Academic self-concept, interest, grades, and standardized test scores: Reciprocal effects models of causal ordering. *Child Development* 76, 2 (2005), 397–416. https://doi.org/10.1111/j.1467-8624.2005.00853.x
[19] Benjamin Nagengast, Ulrich Trautwein, L. Francesca Scalas, Kit-Tai Hau, Herbert W. Marsh, and Man K. Xu. 2011. Who Took the "×" out of Expectancy-Value Theory? *Psychological Science* 22, 8 (2011), 1058–1066. https://doi.org/10.1177/0956797611415540
[20] J. Steve Oliver and Ronald D. Simpson. 1988. Influences of attitude toward science, achievement motivation, and science self concept on achievement in science: A longitudinal study. *Science Education* 72, 2 (1988), 143–155. https://doi.org/10.1002/sce.3730720204
[21] Tony Perez, Jennifer G. Cromley, and Avi Kaplan. 2014. The role of identity development, values, and costs in college STEM retention. *Journal of Educational Psychology* 106, 1 (2014), 315–329. https://doi.org/10.1037/a0034027
[22] Laurie Hart Reyes. 2005. Affective Variables and Mathematics Education. *The Elementary School Journal* 84, 5 (2005), 558–581. https://doi.org/10.1086/461384
[23] Iris M Riggs and Larry G Enochs. 1993. A microcomputer beliefs inventory for middle school students: Scale development and validation. *Journal of Research on Computing in Education* 25, 3 (1993), 383–390.
[24] Neil Selwyn. 1997. Students' attitudes toward computers: Validation of a computer attitude scale for 16–19 education. *Computers & Education* 28, 1 (1997), 35–41.
[25] Gita Taasoobshirazi and Shanshan Wang. 2016. The performance of the SRMR, RMSEA, CFI, and TLI: An examination of sample size, path size, and degrees of freedom. *Journal of Applied Quantitative Methods* 11, 3 (2016), 31–39.
[26] Keith S. Taber. 2018. The Use of Cronbach's Alpha When Developing and Reporting Research Instruments in Science Education. *Research in Science Education* 48, 6 (01 Dec 2018), 1273–1296. https://doi.org/10.1007/s11165-016-9602-2
[27] Ulrich Trautwein and Olaf Köller. 2005. Was lange währt, wird nicht immer gut. *Zeitschrift für Pädagogische Psychologie* 17, 3/4 (2005), 199–209. https://doi.org/10.1024//1010-0652.17.34.199
[28] Ulrich Trautwein, Herbert W Marsh, Benjamin Nagengast, Oliver Lüdtke, Gabriel Nagy, and Kathrin Jonkmann. 2012. Probing for the multiplicative term in modern expectancy–value theory: A latent interaction modeling study. *Journal of educational psychology* 104, 3 (2012), 763.
[29] Allan Wigfield and Jenna Cambria. 2010. Students' achievement values, goal orientations, and interest: Definitions, development, and relations to achievement outcomes. In *Developmental Review*. Vol. 30. Academic Press, 1–35. https://doi.org/10.1016/j.dr.2009.12.001
[30] Allan Wigfield and Jacquelynne S. Eccles. 1992. The development of achievement task values: A theoretical analysis. *Developmental Review* 12, 3 (sep 1992), 265–310. https://doi.org/10.1016/0273-2297(92)90011-P
[31] Jesse L.M. Wilkins. 2004. Mathematics and science self-concept: An international investigation. *Journal of Experimental Education* 72, 4 (2004), 331–346. https://doi.org/10.3200/JEXE.72.4.331-346

Manuscript 4

Title:
Effects of an Extracurricular Computational Thinking Training for Elementary School Children: A Randomized Controlled Field Trial

Authors:
Luzia Leifheit, Jessika Golle, Katerina Tsarava, Ulrich Trautwein, Klaus Ostermann, Manuel Ninaus, and Korbinian Moeller

Unpublished manuscript

UNSUBMITTED MANUSCRIPT

Motivational Effects of an Extracurricular Computational Thinking Training for Elementary School Children: A Randomized Controlled Field Trial

Luzia Leifheit[abf], Katerina Tsarava[af], Jessika Golle[af], Ulrich Trautwein[acf], Klaus Ostermann[bf], Manuel Ninaus[df], and Korbinian Moeller[ef]

[a]Hector Research Institute of Education Sciences and Psychology, University of Tübingen; [b]Wilhelm Schickard Institute for Computer Science, University of Tübingen; [c]Department of Psychology, University of Tübingen; [d]Department of Psychology, University of Innsbruck; [e]Mathematics Education Centre, Loughborough University; [f]LEAD Graduate School and Research Network, University of Tübingen

ARTICLE HISTORY
Compiled November 7, 2020

ABSTRACT
An increasing number of initiatives are promoting computing education at the elementary school level, both within official curricula and in extracurricular education. However, the effectiveness of these initiatives is rarely evaluated empirically.

To address this gap in research, we developed a computational thinking training, aiming to foster elementary school students' programming-related self-concept and motivation for programming. The training introduces students to basic computational concepts – such as sequences, loops, conditionals, and events – and their applications. Didactically, the training follows instructional design principles of embodied learning, game-based learning, scaffolding, and blended learning (unplugged/plugged-in).

We collected data from 197 students in 3rd and 4th grade who participated in the training within the context of an extracurricular enrichment program. We investigated the effectiveness of the training by applying a robust study design in the form of a randomized controlled trial with a waiting control group and repeated measures, allowing for the estimated training effects to be interpreted causally. Additionally, we explored whether gender had any influence on the effectiveness of the training.

Results indicated that the training was effective for promoting a positive self-concept and motivation for programming in elementary school students. These findings reveal promising evidence that computational thinking training efforts can already be effective at the elementary education level. Further, estimated gender effects highlight the need to encourage girls as well as boys to take opportunities to participate in computing education offers.

Abbreviations:
CT: computational thinking
FIML: full-information maximum likelihood estimation
HCAP: Hector Children's Academy Program
MCAR: missing completely at random
MLR: maximum likelihood robust estimation

KEYWORDS
Computational thinking; computer science education; programming education; self-concept; motivation; expectancy-value beliefs; elementary education; primary education; K-4 education

1. Introduction

Efforts for computing education at the elementary level are on the rise, not only in the form of extracurricular initiatives (e.g., Code.org; HourofCode; Kodable), but also as classes inte-

CONTACT Luzia Leifheit. Email: luzia.leifheit@uni-tuebingen.de

grated into official elementary school curricula (e.g., Balanskat & Engelhardt, 2015; Dredge, 2014; Wing, 2014). However, due to reasons such as lack of financial resources and appropriate evaluation tools and procedures (Vivian et al., 2020), the effectiveness of these efforts is rarely evaluated using empirically valid research designs.

To address this gap in empirical research on elementary-level computing education, we developed, implemented, and evaluated an extracurricular 10-week course for 3rd and 4th grade students. The course aimed at fostering a positive programming-related academic self-concept and motivation for learning to program as well as training students in computational thinking. Participating students were introduced to basic algorithmic concepts, including sequences, loops, events, and conditionals through the use of board and card games, programming activities, and pen-and-paper exercises. Didactically, the course followed a conceptually-oriented computational thinking approach, as well as principles of embodied learning, game-based learning, scaffolding, and blended learning (unplugged/plugged-in).

We applied a robust study design with a randomized group assignment, a waiting control group, and repeated measures in a natural classroom setting. This research design allowed for drawing causal inferences regarding the course's effectiveness. We analyzed the collected data using multiple linear regression analysis with maximum-likelihood robust estimation. Outcome variables were motivational attitudes toward programming as measured by the Self-Concept and Attitude toward Programming Assessment (SCAPA, Leifheit et al., 2020).

1.1. Computational thinking

The term computational thinking (henceforth CT) was coined by Seymour Papert (1980) to describe those thought processes that play a fundamental role in the systematic development of computational procedures. In the four decades since – but especially within recent years – CT has been conceptualized and discussed from many different perspectives.

While some argue CT reflects computational skills learned through programming, and consequently is intrinsically linked to programming (Nardelli, 2019), others consider CT a novel and foundational way of thinking that goes beyond what can be achieved through other forms of problem solving, as pointed out by Denning (2017). According to cognitive conceptualizations, CT represents a cognitive ability with applicability in a wide spectrum of domains (Shute, Sun, & Asbell-Clarke, 2017).

Although the scope and meaning of CT have been debated extensively in recent years, the idea that underlying concepts are the most essential aspect of computer science has already been stressed by Donald Knuth (1974). While specific technological applications are developing rapidly, the underlying computational concepts remain the same (Wing, 2006) and are applicable more universally (Wing, 2011). Accordingly, when we refer to CT, we consider it to be the conceptual core of computing defined by concepts, methods, and principles rather than by specific technology or applications (Nardelli, 2019).

Complex problems whose solution engages CT are usually defined by a multitude of factors. Dealing with such problems typically requires solutions involving processes of generalization, abstraction, decomposition into subproblems, and developing an algorithmic solution (Kazimoglu, 2013; Wing, 2006). An algorithmic solution is one that can be formulated unambiguously and leaves no room for interpretation. This means that for a solution to be considered algorithmic, it must be clear after each step which step is to be executed next or on which conditions it is decided which step is to be executed next. For the purpose of introducing children to programming, such algorithmic solutions are considered as consisting of a basic set of CT concepts, such as sequences, loops, events, and conditionals (Brennan & Resnick, 2012).

1.2. Computing in elementary education

Fostering children's computing competences reflects the "21st century vision of students who are not just computer users but also computationally literate creators who are proficient in the

concepts and practices of computer science" (K-12 CS, 2016). The idea of starting to promote and teach programming as early as in elementary school is becoming increasingly popular (e.g., Balanskat & Engelhardt, 2015; Yadav, Mayfield, Zhou, Hambrusch, & Korb, 2014).

Across Europe, more and more countries are integrating or planning to integrate CT and programming into curricula at the elementary school level (Balanskat & Engelhardt, 2015). In 2012, the British Royal Society released a statement, proposing that "every child should have the opportunity to learn computing at school" (Wing, 2014). Subsequently, in September 2014, the Computing at School program was launched, introducing computing as a subject for all students in each year of elementary and secondary education (Dredge, 2014).

Papert (2005) suggests that teaching elementary school children to program could benefit the development of their reasoning abilities. However, it is important for students to be at a stage of cognitive development that renders the acquisition of computational skills possible. Considering that the concrete operational stage (according to Piaget, 1972) is typically reached between the age of 7 to 11 years, introducing elementary school students to programming seems developmentally appropriate. According to Neo-Piagetian models of cognitive development with regard to learning programming, the concrete operational stage is when students first show a purposeful approach to programming (Lister, 2016) and are able to write small programs when they are given well-defined specifications (Teague et al., 2012).

For early computing education, Prottsman (2014) proposes the didactic approach of CT, using unplugged exercises (e.g., board and card games or pen-and-paper exercises) to support comprehension of abstract computational concepts, which children will need to understand when they start learning to program (Kumar, 2014). Therefore, our CT training uses the didactic approach of CT in a blended format with unplugged as well as plugged-in training activities.

1.3. Self-concept and motivation

Motivational disposition toward a school subject has been found to be predictive of a student's achievement within that subject (Marsh, Köller, Trautwein, Lüdtke, & Baumert, 2005; Trautwein & Köller, 2005; Wigfield & Cambria, 2010; Wilkins, 2004). Especially, positive self-concept and attitude have been identified as motivational predictors for how well students do in science and mathematics and how persistent they are in continuing their education on these subjects (e.g., Armstrong & Price, 1982; Lantz & Smith, 1981; Oliver & Simpson, 1988; Wilkins, 2004). Even motivational beliefs of elementary school students can reliably predict which courses they select in late high school and how they perform in these courses (Wigfield & Cambria, 2010). Considering how influential self-concept and motivational beliefs are for educational achievement later in life, our training also aims to foster positive motivational dispositions toward programming.

1.4. The present study

To promote CT as well as programming-related self-concept and motivation, we developed a CT course for elementary school students. The course follows the conceptually-oriented didactic approach of CT. Thus, the focus of the training is not on using specific programming applications or technologies, but on understanding the following essential CT concepts: sequences, loops, conditionals, events, variables, constants, and operators (as identified by Brennan & Resnick, 2012). The instructional design incorporates principles of embodied learning(e.g., Barsalou, 2008; Butz, 2016), game-based learning (e.g., Hamari, Koivisto, & Sarsa, 2014; Ninaus et al., 2015), scaffolding (Belland, 2014), and blended learning (unplugged/plugged-in) (e.g., del Olmo-Muñoz, Cózar-Gutiérrez, & González-Calero, 2020; Prottsman, 2014). A more detailed description of the CT training is given in Section 2.3 Training: CT course.

We conducted a randomized controlled trial with 197 elementary school students (see Section 2.1 Participants) who were randomly assigned to either the training condition or the waiting control condition (Section 2.2 Study design). We analyzed the collected data for training effects

and gender effects using multiple linear regressions with maximum likelihood robust estimation (Section 2.6 Statistical Analysis). Outcome variables were seven different aspects of self-concept and motivation (see Section 2.5 Measurements).

We expected the CT training to have a positive effect on the development of students' self-reported experience and understanding of programming, on their programming-related self-concept, their motivational beliefs about programming, as well as their compliance and persistence when working on programming tasks (Hypothesis). Additionally, we wanted to explore whether there are any differential gender effects on students' learning outcomes, self-concept, and motivation.

Additionally, CT ability and cognitive abilities were assessed to measure students' learning progress. The analysis procedure and effects of the CT training on CT ability and cognitive abilities are reported separately (see Tsarava et al., in preparation).

2. Methods

2.1. Participants

We collected data from 197 elementary school students ($female: N = 47;\ 24\%$) between 7 and 10 years old ($M = 8.83; SD = 0.73$) attending 3rd and 4th grade of German elementary school.

The children voluntarily participated in the CT training, which was offered in the form of a course within the Hector Children's Academy Program (HCAP). The HCAP is a voluntary extracurricular enrichment program for talented elementary school children in the German federal state of Baden-Württemberg, currently offered at 66 local sites. Children are nominated for the HCAP by their teachers based on characteristics such as interest, motivation, and school performance. After being nominated for the HCAP, students choose from a variety of extracurricular enrichment courses at their local HCAP site (for further information about the HCAP, see Golle, Herbein, Hasselhorn, & Trautwein, 2017; Rothenbusch, Zettler, Voss, Lösch, & Trautwein, 2016). Before students participated in the study, we obtained informed written consent from all students as well as their parents or legal guardians.

2.2. Study design

The CT training was offered in 29 courses within the regular course program at 16 local HCAP sites. The training was developed for 6 to 10 children per course group. Due to dropout, the size of the actual course groups varied between 4 to 10 students per course. The courses were offered during the school year 2018/2019. The courses and testing sessions took place in regular classrooms at the respective local sites. All test sessions were conducted either by researchers or by research assistants following detailed instructions.

We used a randomized controlled trial design (e.g., Friedman, Furberg, & DeMets, 2010) with repeated measures and a waiting control group. This study design allows for estimating the causal effect of the training (Torgerson & Torgerson, 2008) on both CT ability and motivational attitudes toward programming.

Upon enrolling for the CT course, we randomly assigned students to either the training condition or the control condition using computer-generated randomization. One week before the first session of the CT course, all participating students attended a pretest session. At the end of the pretest session, students were informed about whether they were randomized to participate as part of the training group or the waiting control group. The CT courses for students in the training condition started one week after the pretest while students in the control condition waited for the start of their course. One week after the CT course for students in the training condition was completed, all children – including those assigned to the control condition – attended a posttest session. One week after the posttest, students in the waiting control condition started to take the CT course. The study design is outlined in Figure 1.

This research design has the benefit that, in accordance with ethical standards for interven-

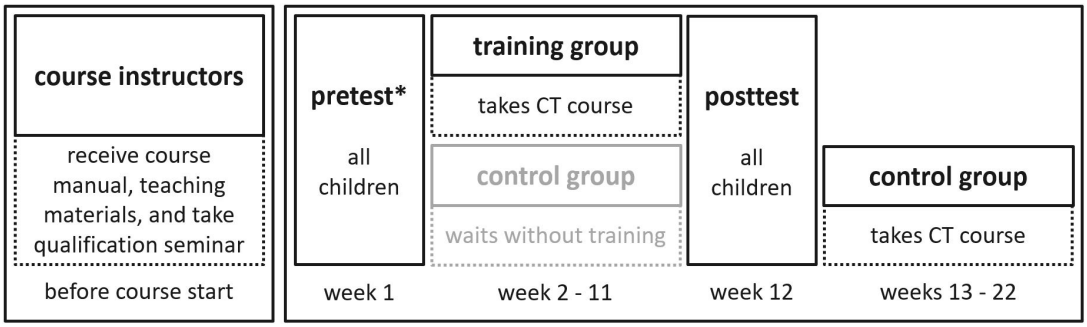


Figure 1. Study design: Randomized controlled field-trial with waiting control group and repeated measures.

tion trials, all participating students are treated equally and receive the same training (Emanuel, Abdoler, & Stunkel, 2010). This approach also reduces the risk of drop-out among students in the control group. The study was approved by the ethics committee of the Leibniz-Institut für Wissensmedien.

2.3. Training: CT course

2.3.1. Aims and instructional set-up

The objective of this CT training is to foster CT abilities and to spark motivation for computing. More specifically, the course aim is for children to develop an initial understanding of basic computational concepts and to form a positive self-concept regarding their programming abilities. The CT course comprises 10 lessons of 90 minutes each and was conceptualized to be taught for groups of 6 to 10 students across a timespan of 10 weeks.

The course follows the conceptually-oriented didactic approach of CT. Thus, the focus of the training is not on using specific programming applications or technologies, but on understanding the following essential CT concepts: sequences, loops, conditionals, events, variables, constants, and operators (as identified by Brennan & Resnick, 2012). The instructional design incorporates principles of embodied learning, game-based learning, scaffolding, and blended learning (unplugged/plugged-in).

Throughout the course, students are gradually guided from concrete and playful unplugged activities (board/card games and pen-and-paper exercises) to more abstract and sophisticated plugged-in programming exercises (programming games, simulations, hardware, and simulated robots).

2.3.2. Unplugged activities

All CT concepts are introduced using unplugged activities (as suggested by Prottsman, 2014) before being transferred and applied in plugged-in contexts. Unplugged activities can be motivating and beneficial for forming an initial understanding of CT concepts in elementary school students (Leifheit, Jabs, Ninaus, Moeller, & Ostermann, 2018). Using a blended approach with a combination of unplugged and plugged-in activities can lead to better CT performance and motivation compared to a plugged-in only approach (del Olmo-Muñoz et al., 2020).

The unplugged activities include the life-size board and card game series Crabs & Turtles (Tsarava, Moeller, & Ninaus, 2018a, 2018b) as well as pen-and-paper exercise sheets based on tasks from the Bebras international informatics challenge for children (Dagienė & Sentance, 2016).

In the first three lessons of the course, students are introduced to basic CT concepts through playing Crabs & Turtles, which was developed specifically for this purpose and target group (Tsarava et al., 2018a, 2017). Crabs & Turtles is deliberately designed to be independent of any

specific programming languages or environments.

The Crabs & Turtles series – two board games and one card game – allow for basic CT concepts to be experienced in an embodied way (e.g., Barsalou, 2008), on the basis of which conceptual abstractions can be naturally supported (Butz, 2016). Embodied experience means that one experiences something tangibly or physically, for example through moving one's body, instead of perceiving it only in a figurative or abstract way, which helps in understanding abstract concepts through increased mental organization (Noice & Noice, 2001). Furthermore, the use of game-based methods can improve students' performance (Ninaus et al., 2015) and increase their motivation and engagement in the learning task (Hamari et al., 2014).

In addition to unplugged games, the training also includes unplugged exercise sheets. At the end of each of the ten lessons, students work on exercise sheets with puzzle-like tasks for 10 minutes. The tasks are based on the Bebras International Challenge on Informatics and Computational Thinking for children (Bebras), which has been established for 16 years and is being held in in 50 countries (Dagienė & Sentance, 2016). Bebras tasks engage students in CT through motivating puzzle-like pen-and-paper exercises (Dagienė & Sentance, 2016). Within the CT course, these exercise sheets are used so that students can recapitulate previously learnt CT concepts in an unplugged way to deepen their conceptual understanding.

2.3.3. Plugged-in activities

Plugged-in activities are used to enable students to transfer and apply the CT concepts to which they were introduced using unplugged activities. Scaffolded programming activity sheets guide students through three different programming languages and environments.

In the second lesson, students are introduced to plugged-in programming activities using the block programming language and environment Scratch 2.0, specifically developed for children (Diaz & Lopez, 2017). Subsequently, in the fourth to sixth lesson, Scratch is used to enable students to apply the CT concepts introduced through Crabs & Turtles and transfer their conceptual understanding to create basic programs (Diaz & Lopez, 2017), such as simple games and simulations.

In the seventh and eighth lesson, the possibilities of Scratch are extended through the open-hardware platform Arduino (Badamasi, 2014) as well as the programming language and environment S4A (Scratch for Arduino) (S4A), via which the Arduino's various ports and sensors can be controlled. After CT concepts were introduced by Crabs & Turtles and deepened in Scratch, the Arduino and S4A are used to transfer these concepts to an area going beyond the creation of simple screen applications to give children a first glimpse into hardware programming. This provides another embodied learning experience, as students recognize their program code can not only determine what happens on the screen in front of them, but also interact with their physical environment.

In the last two lessons of the course, the robot simulation Open Roberta gives students the opportunity to practice their acquired CT understanding more independently by programming and controlling a simulated robot (Jost, Ketterl, Budde, & Leimbach, 2014). Unlike Scratch, Open Roberta offers no customizable graphical elements – the only way to control the simulated robot is through program code. Therefore, Open Roberta provides no distractions from programming itself, making the environment especially suited for open-ended exploration of programming.

Through all three of these programming environments, students are guided by scaffolded programming activity sheets. In the beginning of the course, these activity sheets provide students with clearly defined program goals and step-by-step instructions. As the course progresses, the programming activity sheets gradually provide less detailed instructions and, by the end of the course, even allow students to choose their own program goals.

2.3.4. Context and outlook

Because neither elementary nor lower secondary school curricula in Germany currently guarantee that children can pursue further computing education under professional and pedagogical guidance after completing the CT course, the course also aims to enable students to program independently and pursue further computing education on their own.

To this end, students are guided in transferring their CT understanding to different applications and encouraged to familiarize themselves with new programming languages and environments, rather than focusing on the specifics of just one. Upon completing the course, students are provided with a curated collection of links to further resources, such as online programming courses for children, to facilitate further engagement in computing-related activities independent of official school curricula.

2.4. Treatment fidelity

A training needs to be consistently implemented as intended across all groups to allow for drawing valid inferences regarding the training's effectiveness (Carroll et al., 2007). A comprehensive course manual was prepared to enable the course instructors to teach the course as intended (O'Donnell, 2008).

The manual was created and tested during a pilot run of the CT course with 31 students at 4 different HCAP sites. The first author taught the CT course at 3 of the participating HCAP sites, while a computer science teacher-in-training taught the course at the fourth HCAP site. This teacher-in-training gave regular qualitative feedback regarding his experiences with teaching the course based on the manual, including how well the activities worked, how clear the manual's written instructions were to him, and whether the specified time frames for each activity were feasible. Following the feedback and the first author's own experiences teaching the CT course, the manual was adapted and improved before being used in the present study.

The final version of the manual provides a theoretical introduction to CT, embodied learning, and game-based learning, before offering a summary of the learning targets as well as teaching materials of the training. A comprehensive lesson plan is given for each course unit, detailing all activities, materials, micro-level learning objectives, and exact time frames for each part of the lesson. Along with the course manual, we provided all course instructors with the same learning materials to be used during the training. The original German manual for the CT course is available from the first author (Leifheit, Tsarava, Ninaus, & Moeller, 2018).

In addition to receiving the manual and materials for the CT training, all course instructors were prepared for teaching the course in an 8-hour qualification seminar offered by the first author. The qualification seminar provided an overview of the theoretical background of the course and its learning targets, familiarized the instructors with the different types of learning materials, and included hands-on exercises in which the instructors learned to conduct the course activities. Providing all instructors with the course manual, the learning materials needed for the training, and the qualification seminar facilitates adherence to the course and ensures treatment fidelity (Graham & Harris, 2014).

2.5. Measurements

The variables we measured fall into three categories: 1) self-reports on motivational constructs, 2) CT ability, and 3) cognitive abilities (as control variables). Motivational constructs were measured by seven variables (self-reported), CT ability was measured by one variable (performance test), and cognitive abilities were measured by eight variables (performance tests).

All variables were measured twice – once at pretest and once at posttest – for both the training group and the waiting control group using paper-and-pencil testing materials and questionnaires.

2.5.1. Motivational constructs

Self-concept and motivational beliefs were assessed using the 7-scale, 50-item Self-Concept and Attitude toward Programming Assessment questionnaire (SCAPA Leifheit et al., 2019, 2020). The scales measure the following aspects of motivation and attitude toward programming:

(P1) experience and understanding, whether students have programmed before or are familiar with the term "programming" (e.g. "I know what the word programming means")
(P2) self-concept, how good students believe they are at programming (e.g. "I'm good at programming")
(P3) intrinsic value belief, how much students enjoy programming (e.g. "Programming is fun to me")
(P4) attainment value belief, how important being good at programming is to the students (e.g. "Being good at programming means a lot to me")
(P5) utility value belief, how useful students think programming is within different areas of their experience, such as school or social life (e.g. "Programming skills have many benefits in my daily life")
(P6) cost belief, how much effort students think programming – or learning to program – is for them (e.g. "To be successful at programming, I have to give up other activities I enjoy")
(P7) compliance and persistence, (e.g. "I always try to complete my programming tasks")

Items were rated on a 4-point Likert scale (1 = agree completely; 2 = rather agree; 3 = rather do not agree; 4 = do not agree at all).

2.5.2. CT ability

CT ability was measured using a shortened version of the Computational Thinking test (CTt, Román-González, 2015), which originally contains 28 items and was validated with a sample of 1,251 students from 5^{th} to 10^{th} grade (Román-González, Pérez-González, & Jiménez-Fernández, 2017). There was no validated instrument available for a younger age group. To have a more a appropriate instrument for our participants, we used only the 21 easiest items – as ranked by the instrument's authors – of the CTt (items 1, 2, 3, 4, 5, 6, 7, 9, 10, 11,13, 14, 17, 18, 19, 20, 21, 24, 26, 27, and 28).

The CTt assesses students' understanding of sequences, loops, conditionals, and simple functions (Román-González, 2015). All items consist of a problem to be solved, a pictorial representation of the problem (e.g., a maze), a pseudocode-like representation of the problem (in visual block-based programming style), and four multiple-choice style answers, only one of which is correct for each item (Román-González, 2015). Each correctly solved item was scored with one point, so students were able to score in between 0 and 21 points on the CTt.

It is important to note that we did not train to the test. None of the tasks of the CTt were used during the CT course. Likewise, none of the tasks used in the CT training were similar to any CTt tasks in form, problem statement, or visual representation. Furthermore, functions were not part of the CT training. Therefore, the CTt partly can be considered a far-transfer assessment (Barnett & Ceci, 2002).

2.5.3. Control variables

CT ability as measured by the CTt has been found to be correlated with reasoning abilities, specifically verbal, spatial, problem completion, and series continuation reasoning (Román-González et al., 2017). Confirming this further, CTt score has been found to correlate with nonverbal visuospatial reasoning and arithmetic reasoning abilities, specifically multiplication and problem completion in the form of arithmetic fill-in-the-blank tasks (Tsarava et al., 2019).

We therefore assessed these cognitive abilities using three different instruments. To measure reasoning and visuospatial ability, we used the subtests series and matrices of the Culture Fair Intelligence Scale (CFT 20-R, Weiß, Albinus, & Arzt, 2006). Verbal reasoning ability was mea-

sured using subtest V1 of the Cognitive Ability Test (*Kognitiver Fähigkeitstest*) (KFT 4-12+R, Heller & Perleth, 2000). Arithmetic ability was assessed using the five subtests i) writing speed, ii) addition, iii) subtraction, iv) multiplication, and v) problem completion of the Arithmetic Test for the Assessment of Foundational Mathematical Competencies at Elememtary-school Age (*Heidelberger Rechentest zur Erfassung mathematischer Basiskompetenzen im Grundschulalter*) (HRT, Haffner, 2005).

These control variables were not included for the analyses of the training effect on motivational constructs (for a more detailed description of the analysis and results of the training regarding cognitive abilities and CT ability, see Tsarava et al., in preparation).

2.6. Statistical analysis

To assess the result of randomizing students into the training and control groups, we examined baseline equivalence to establish whether the groups were reasonably similar with regard to the relevant constructs. We used two-tailed t-tests for all dependent and control variables measured at pretest to check for potential differences between the two groups.

2.6.1. Missing values

The share of missing data ranged between 4.1% and 16.2% across both groups (see Table **??**). The higher number of missing data was due to participants not being present at posttest. There was no differential drop-out between training group and control group ($\chi^2(1) = .45$; $p = .503$). We also ran t-tests to compare the means of students present at posttest with students missing at posttest for all dependent variables and found no significant differences between the groups (all p-values $> .05$). This is consistent with the assumption of data being missing at random (Enders, 2010).

To verify this, we ran Little's Missing Completely at Random (MCAR) test (Little, 1988) and found that data were indeed missing completely at random ($\chi^2(1) = 349.34$; $p = .503$). This allowed us to use the full-information maximum likelihood (FIML) estimator as missing data treatment (Muthén & Muthén, 2012). FIML takes all available data into account when estimating model parameters and standard errors (Buhi, Goodson, & Neilands, 2008; Schafer & Graham, 2002).

2.6.2. Effectiveness

We used multiple linear regressions with maximum-likelihood robust (MLR) estimation to assess the effectiveness of the training. MLR is robust for non-normally distributed data and corrects standard errors for non-normality. All regression analyses were conducted in Mplus Version 7 (Muthén & Muthén, 2012).

The regression's dependent variables were motivational constructs at posttest as assessed by the seven scales (P1) experience and understanding, (P2) self-concept, (P3) intrinsic value belief, (P4) attainment value belief, (P5) utility value belief, (P6) cost belief, and (P7) compliance and persistence with regard to programming. The pretest score of each respective variable was included as a control variable to increase the precision of the training effect estimate. We z-standardized all continuous variables before conducting the analyses. Binary-coded group assignment (control = 0; training = 1) was the predictor.

A training's effect size on any given outcome variable is the standardized mean difference between training and control group (Hedges, 2007). Because all continuous variables were z-standardized, the unstandardized regression coefficient B indicates the effect size of the training. Due to the randomized controlled study design, training effect sizes can be interpreted causally.

In an additional regression, we used binary-coded gender (female = 0; male = 1) as predictor and controlled for group assignment to estimate an effect of gender on students' motivational development. It must be emphasized, however, that any estimate of gender effect can only be interpreted correlationally and not causally because the predictor variable was not subject to

randomization within our study.

The analysis procedure and results for estimating the effect of the CT training on CT ability and cognitive abilities are reported separately (see Tsarava et al., in preparation).

3. Results

All descriptive statistics for training and control group at both measurement points are presented in Table 1. Using t-tests to check for differences between training and control group at pretest time, we found no significant baseline differences for any of the outcome or control variables.

The training effects are shown in Table 2 and gender effects are shown in Table 3.

3.1. Training effects

As stated in our Hypothesis, we expected to observe that the CT training would have a positive effect on students' self-concept and motivational beliefs. Students who participated in the CT training showed a significantly positive development for (P1) experience and understanding ($B = .373$; $p < .001$), (P2) self-concept ($B = .167$; $p = .025$), and (P3) intrinsic value ($B = .197$; $p = .003$) compared to children in the control group. Overall, three out of the seven motivational construct scales were significantly positively affected by the training (see Table 2).

3.2. Gender effects

In addition to investigating training effects, we wanted to explore whether there were any differences between girls and boys concerning the development of their motivational beliefs. We found a very small gender effect for P2 self-concept in favor of the male students ($B = .110$; $p = .036$). Aside from this construct, we could not identify any significant differences between girls and boys with regard to development of the children's motivational attitudes (see Table 3).

4. Discussion

While there is an increasing number of initiatives for teaching CT at the elementary school level (e.g., Code.org; HourofCode; Kodable; **?**, 2015), the effectiveness of these efforts is rarely evaluated empirically, for reasons such as lack of funding and appropriate assessment instruments (Vivian et al., 2020). With this in mind, experts in the field of evaluation and assessment at the elementary level have identified a key need for empirical research on how to teach and assess computing for this age group and, accordingly, how to best evaluate motivational development and learning progress in elementary education (Vivian et al., 2020).

To address this need and contribute to the body of empirical evidence, we developed a CT training for elementary school children and explored to what extent the training could foster

Table 1. Descriptive statistics

	Pretest					Posttest				
	training		control		missing	training		control		missing
	M	*SD*	*M*	*SD*		*M*	*SD*	*M*	*SD*	
P1	3.478	0.591	3.348	0.644	4.1	3.692	0.504	3.168	0.695	14.2
P2	3.326	0.601	3.306	0.678	7.6	3.311	0.675	3.085	0.724	15.2
P3	3.598	0.613	3.388	0.845	9.1	3.601	0.730	3.253	0.784	16.2
P4	3.452	0.627	3.366	0.629	6.6	2.996	0.708	2.962	0.740	14.2
P5	3.443	0.477	3.296	0.784	10.2	2.837	0.741	2.780	0.737	15.2
P6	2.535	0.828	2.492	0.866	11.2	1.802	0.688	1.741	0.621	14.2
P7	3.607	0.400	3.556	0.545	12.7	3.417	0.598	3.336	0.658	16.2

M: mean; *SD* = standard deviation; missing: amount of missing data in percent.

Table 2. Training effects

	B	SE	p
P1 experience/understanding	.373	.084	< .001
P2 self-concept	.167	.074	.025
P3 intrinsic value	.197	.067	.003
P4 attainment value	.001	.093	.989
P5 utility value	.024	.074	.746
P6 cost belief	.041	.070	.556
P7 compliance/persistence	.023	.065	.718

Table 3. Gender effects

	B	SE	p
P1 experience/understanding	.070	.079	.378
P2 self-concept	.110	.053	.036
P3 intrinsic value	.029	.066	.657
P4 attainment value	.006	.059	.916
P5 utility value	.068	.049	.166
P6 cost belief	.075	.072	.296
P7 compliance/persistence	−.024	.065	.632

B: unstandardized regression coefficient; SE: standard error; p: probability value (two-tailed), significance level set to $\alpha = .05$. Continuous variables were z-standardized before the analyses.

positive motivational attitudes toward programming and CT abilities (for a detailed report on the CT ability results, see Tsarava et al., in preparation). The course introduced students to CT concepts such as sequences, loops, events, and conditionals through the use of board games, programming activities, and pen-and-paper exercises. The didactic and pedagogical approach of the training followed principles of scaffolding, embodied learning, game-based learning, and blended learning (unplugged/plugged-in).

The CT training was taught by different course instructors in 16 towns in the German federal state of Baden-Württemberg. None of the instructors worked in research or were involved in the development of the course. Instead, they had received a detailed course manual and a one-day preparation seminar along with all teaching materials for the course.

We studied the CT training's effectiveness in a randomized controlled field trial with a waiting control group and repeated measures. The collected data were analyzed in a multiple linear regression analysis using maximum-likelihood robust estimation. The study design and analysis procedure yielded conservative estimates to minimize the risk of finding false positives. Outcome variables were motivational attitudes toward programming as assessed by the seven scales of the SCAPA instrument (Leifheit et al., 2020), namely (P1) experience and understanding, (P2) self-concept, (P3) intrinsic value belief, (P4) attainment value belief, (P5) utility value belief, (P6) cost belief, and (P7) compliance and persistence.

Overall, the collected data indicated positive effects for three out of the seven motivational constructs. Students who received the CT training, compared to the control group, were more positively affected in their motivational attitudes regarding programming experience and understanding, programming self-concept, and intrinsic value belief about programming.

The course also had a significant positive effect on students' CT ability (as reported in more detail in Tsarava et al., in preparation).

4.1. Classification and interpretation of the estimated effects

4.1.1. Training effects

Overall, the CT training had a significant positive effect on students' self-concept and motivational beliefs regarding programming. Compared to the control group, students in the training group reported a more positive development in (P1) experience and understanding of programming ($B = .373$; $p < .001$), in (P2) programming-related self-concept ($B = .167$; $p = .025$), and (P3) intrinsic value belief about programming ($B = .197$; $p = .003$). At the same time, we identified no significant effects for (P4) attainment value belief, (P5) utility value belief, (P6) cost belief, and (P7) compliance and persistence with regard to programming.

Considering the lack of similar studies, we interpreted effect sizes according to Cohen's commonly used classification of effect size (small: $d = 0.2$, medium: $d = 0.5$, large: $d = 0.8$) (Cohen, 1992), placing all estimated effect sizes in the small to medium range. Owing to our study design, analysis procedure, and participant sample, three factors contributing to small effect sizes must be taken into account.

First, the CT training had not been administered by researchers in a standardized instructional setting. Instead, the training took place under real-world conditions, i.e., in normal class-

rooms with different course instructors who had been trained to teach the CT course and had received a manual, but had not been involved in the course's development or research. Compared to standardized instruction under controlled conditions, real-world settings and instructor variability lead to smaller treatment effects (e.g., Greene, 2015; Hulleman & Cordray, 2009; Lendrum & Wigelsworth, 2013).

Second, the sample consisted of students who had been nominated for an extracurricular enrichment program based on interest and achievement in school. All students participating in our study voluntarily selected the CT course from a range of courses offered within the enrichment program. Students displaying above-average interest and achievement in school, as well as selecting the CT course by choice, could indicate high levels of motivation at pretest time resulting in potential ceiling effects and, consequently, smaller training effects (Resch & Isenberg, 2018).

Finally, the multiple linear regression analysis using maximum-likelihood robust estimation minimizes the risk of finding false positives and yields conservative estimates (Dong & Peng, 2013), meaning the effects were unlikely to be estimated too large and any identified effects are relevant outcomes of the training.

Considering these factors, our findings suggest the training was successful in fostering programming-related self-concept and motivational beliefs about programming in elementary school students. Because there are no similar training studies reporting effect sizes for elementary school children, we cannot compare the observed effect sizes to similar studies.

4.1.2. Gender effects

In our exploratory regression analysis, we did not find any significant effects of gender on (P1) experience and understanding, (P3) intrinsic value belief, (P4) attainment value belief, (P5) utility value belief, (P6) cost belief, or (P7) compliance and persistence with regard to programming. This indicates that a student's gender does not affect to what extent participating in the CT training positively influences their motivational beliefs about programming.

However, we found that participating in the CT course leads boys to develop a more positive (P2) programming-related self-concept than girls. Considering that only 24% of the children participating in our study were girls, this is in line with findings on how gender and gender ratio affect girls' academic self-concept within special classes for gifted students (Preckel, Zeidner, Goetz, & Schleyer, 2008).

In a study with 769 elementary and junior high school students attending special classes for gifted students, Preckel and colleagues (2008) found that the lower the percentage of girls within a class, the more negatively girls' academic self-concept was impacted. Conversely, gender ratio had no significant effect on the academic self-concept of boys, no matter if boys constituted a majority or minority within their class.

4.2. Limitations and future research

While the present study reveals promising insights into the effectiveness of a training for CT ability and motivational beliefs at the elementary school level, it is important to note its limitations and identify key areas for further research.

On the one hand, focusing on a clearly defined target group is recommended for initial investigation of a training's effectiveness (Gottfredson et al., 2015). On the other, this limits the generalizability of the findings. Because the effectiveness of the CT training was evaluated for a sample of 3rd and 4th grade students within an extracurricular enrichment program for talented children, the estimated effects cannot necessarily be generalized to all children in the age group. In a future study, the training could be adapted and evaluated for a broader sample.

Further, because the CT training consists of multiple components, it remains unclear which specific components of the training were effective. It seems desirable to conduct a more fine-grained follow-up study to better assess the effectiveness of the training's individual components.

Moreover, the present study focused on the immediate training effects. As a result, longitu-

dinal effects cannot be predicted. In future studies, additional measurement timepoints after the completion of the training could provide insight into potential long-term effects of the training.

Lastly, finding appropriate instruments to measure CT ability in the target age group proved a challenge. Thus, we used a shortened version selecting the 21 easiest items of the Computational Thinking test, which is originally aimed at middle school students (Román-González, 2015). We expect more instruments for assessing CT to be published within the coming years. In future research regarding the effectiveness of the training and its components, we hope to be able to apply a wider array of assessment instruments, especially instruments developed specifically for the target age group.

4.3. Conclusion

In the present study, we aimed to evaluate a multi-component CT training incorporating instructional principles of embodied learning, game-based learning, scaffolding, and blended learning (unplugged/plugged-in) in a real-world setting. Our findings indicate that the training was generally successful in reaching its goal to promote 3rd and 4th grade students' programming-related self-concept and motivation for programming.

According to our Hypothesis, we expected the CT training to have a positive effect on the development of students' self-reported experience and understanding of programming, on their programming-related self-concept, their motivational beliefs about programming, as well as their compliance and persistence when working on programming tasks. In agreement with this Hypothesis, we found that students who participated in the training displayed a more positive development of self-reported experience and understanding of programming, programming-related self-concept, and intrinsic motivation for programming. However, we observed no significant effects for attainment value belief, utility value belief, cost belief, or compliance and persistence.

Additionally, we found that after the CT training, boys' programming-related self-concept improved significantly more than that of girls. Considering that only 24% of the participating students were female, this is in line with research reporting that a smaller ratio of girls within a class for gifted students negatively impacts girls' academic self-concept (Preckel et al., 2008). However, our study design does not allow for this finding to be interpreted causally. No other gender effects were identified.

In conclusion, the present study indicated promising initial results regarding the effectiveness of the training components and instructional design principles applied in our CT course for fostering positive self-concept and motivation for programming in 3rd and 4th grade students participating in an extracurricular enrichment program.

Further, we found the CT training evaluated in the present study to be effective for improving students' CT ability (see Tsarava et al., in preparation).

Our study design and analysis procedures allow for drawing causal inferences about the effectiveness of the CT training. For this reason, the reported effect sizes can serve as a benchmark with which future effectiveness research in the field of early computing education can be compared. Yet, more research on the effectiveness of different instructional approaches is needed to understand how computing can best be taught at the elementary school level. We therefore hope that more empirical evaluation studies applying robust study designs will investigate the effectiveness of computing education efforts in elementary education.

Acknowledgements

For their support of this study, we would like to thank all involved research assistants and the members of the working group "Neurocognitive Plasticity" at Leibniz-Institut für Wissensmedien, as well as the entire team of "Wissenschaftliche Begleitung der Hector Kinderakademien" at the Hector Research Institute of Education Sciences and Psychology, whose dedicated support we could always rely on while planning and implementing this study.

Special thanks go to Johannes Dominik Krauß for his feedback during the early stages of the CT course development and for his help in improving the course manual.

This research was partly funded by the Hector Stiftung II and supported by the Leibniz-Institut für Wissensmedien. Luzia Leifheit is a doctoral student at the LEAD Graduate School & Research Network [GSC1028], which was funded within the framework of the Excellence Initiative of the German federal and state governments.

References

Armstrong, J. M., & Price, R. A. (1982, mar). Correlates and predictors of women's mathematics participation. *Journal for Research in Mathematics Education*, *13*(2), 99. Retrieved from `https://www.jstor.org/stable/748357?origin=crossref`

Badamasi, Y. A. (2014). The working principle of an arduino. In *2014 11th international conference on electronics, computer and computation (ICECCO)* (pp. 1–4).

Balanskat, A., & Engelhardt, K. (2015). Computing our future. *Computer programming and coding. Priorities, school curricula and initiatives across Europe*.

Barnett, S. M., & Ceci, S. J. (2002). When and where do we apply what we learn?: A taxonomy for far transfer. *Psychological bulletin*, *128*(4), 612.

Barsalou, L. W. (2008). Grounded cognition. *Annu. Rev. Psychol.*, *59*, 617–645.

Bebras. (n.d.). *Bebras: International Challenge on Informatics and Computational Thinking.* Retrieved from `https://www.bebras.org/`

Belland, B. R. (2014). Scaffolding: Definition, current debates, and future directions. In *Handbook of Research on Educational Communications and Technology* (pp. 505–518). Springer.

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American educational research association, Vancouver, Canada* (Vol. 1, p. 25).

Buhi, E. R., Goodson, P., & Neilands, T. B. (2008). Out of sight, not out of mind: Strategies for handling missing data. *American journal of health behavior*, *32*(1), 83–92.

Butz, M. V. (2016). Toward a unified sub-symbolic computational theory of cognition. *Frontiers in psychology*, *7*, 925.

Carroll, C., Patterson, M., Wood, S., Booth, A., Rick, J., & Balain, S. (2007). A conceptual framework for implementation fidelity. *Implementation science*, *2*(1), 40.

Code.org. (n.d.). *CS Fundamentals for grades K-5.* Retrieved from `https://code.org/educate/curriculum/elementary-school`

Cohen, J. (1992). A power primer. *Psychological bulletin*, *112*(1), 155.

Dagienė, V., & Sentance, S. (2016). It's computational thinking! Bebras tasks in the curriculum. In *International conference on informatics in schools: Situation, evolution, and perspectives* (pp. 28–39).

del Olmo-Muñoz, J., Cózar-Gutiérrez, R., & González-Calero, J. A. (2020). Computational thinking through unplugged activities in early years of primary education. *Computers & Education*, *150*, 103832.

Denning, P. J. (2017). Remaining trouble spots with computational thinking. *Communications of the ACM*, *60*(6), 33–39.

Diaz, L. M., & Lopez, L. F. A. (2017). A classification of programming styles in scratch. In *Proceedings of the 8th Latin American Conference on Human-Computer Interaction* (pp. 1–4).

Dong, Y., & Peng, C.-Y. J. (2013). Principled missing data methods for researchers. *SpringerPlus*, *2*(1), 222.

Dredge, S. (2014). *Coding at school: A parent's guide to England's new computing curriculum* (Vol. 4). Retrieved from `https://www.theguardian.com/technology/2014/sep/04/coding-school-computing-children-programming`

Emanuel, E., Abdoler, E., & Stunkel, L. (2010). Research ethics: How to treat people who participate in research. *National Institutes of Health (NIH) Clinical Center Department of Bioethics and Foundation for the NIH*.

Enders, C. K. (2010). *Applied missing data analysis.* Guilford press.

Friedman, L. M., Furberg, C. D., & DeMets, D. L. (2010). Blindness. In *Fundamentals of clinical trials* (pp. 119–132). Springer.

Golle, J., Herbein, E., Hasselhorn, M., & Trautwein, U. (2017). Begabungs-und Talentförderung in der

Grundschule durch Enrichment: Das Beispiel Hector-Kinderakademien [promoting gifted and talented elementary school students via enrichment: Using the example hector children's academy]. *Tests und Trends-Jahrbuch der pädagogisch-psychologischen Diagnostik*, *15*, 177–196.

Gottfredson, D. C., Cook, T. D., Gardner, F. E., Gorman-Smith, D., Howe, G. W., Sandler, I. N., & Zafft, K. M. (2015). Standards of evidence for efficacy, effectiveness, and scale-up research in prevention science: Next generation. *Prevention science*, *16*(7), 893–926.

Graham, S. E., & Harris, K. R. (2014). Conducting high quality writing intervention research: Twelve recommendations.

Greene, J. A. (2015). Serious challenges require serious scholarship: Integrating implementation science into the scholarly discourse. *Contemporary Educational Psychology*, *40*, 112–120.

Haffner, J. (2005). *HRT 1-4: Heidelberger Rechentest; Erfassung mathematischer Basiskompetenzen im Grundschulalter[TODO.* Hogrefe.

Hamari, J., Koivisto, J., & Sarsa, H. (2014). Does gamification work? – a literature review of empirical studies on gamification. In *2014 47th Hawaii international conference on system sciences* (pp. 3025–3034).

Hedges, L. V. (2007). Effect sizes in cluster-randomized designs.

Heller, K. A., & Perleth, C. (2000). *KFT 4-12+ R kognitiver Fähigkeitstest für 4. bis 12. Klassen, Revision[TODO.* Beltz Test.

HourofCode. (n.d.). *Hour of Code: Join the Movement.* Retrieved from `https://hourofcode.com/`

Hulleman, C. S., & Cordray, D. S. (2009). Moving from the lab to the field: The role of fidelity and achieved relative intervention strength. *Journal of Research on Educational Effectiveness*, *2*(1), 88–110.

Jost, B., Ketterl, M., Budde, R., & Leimbach, T. (2014). Graphical programming environments for educational robots: Open roberta – yet another one? In *2014 IEEE International Symposium on Multimedia* (pp. 381–386).

K-12 CS, F. S. C. (2016). *K-12 Computer Science Framework.* ACM. Retrieved from `https://k12cs.org`

Kazimoglu, C. (2013). *Empirical Evidence That Proves a Serious Game is an Educationally Effective Tool for Learning Computer Programming Constructs at the Computational Thinking Level* (Unpublished doctoral dissertation). University of Greenwich.

Knuth, D. E. (1974). Computer science and its relation to mathematics. *The American Mathematical Monthly*, *81*(4), 323–343.

Kodable. (n.d.). *Programming for Kids.* Retrieved from `https://www.kodable.com/`

Kumar, D. (2014). *Welcome.* Retrieved from `https://dl.acm.org/doi/10.1145/2684721.2684733`

Lantz, A. E., & Smith, G. P. (1981). Factors influencing the choice of nonrequired mathematics courses. *Journal of Educational Psychology*, *73*(6), 825–837. Retrieved from `http://content.apa.org/journals/edu/73/6/825`

Leifheit, L., Jabs, J., Ninaus, M., Moeller, K., & Ostermann, K. (2018). Programming unplugged: An evaluation of game-based methods for teaching computational thinking in primary school. In *Proceedings of the 12th European Conference on Game-Based Learning (ECGBL)* (pp. 344–353).

Leifheit, L., Tsarava, K., Moeller, K., Ostermann, K., Golle, J., Trautwein, U., & Ninaus, M. (2019). Development of a questionnaire on self-concept, motivational beliefs, and attitude towards programming. In *Proceedings of the 14th Workshop in Primary and Secondary Computing Education (WiPSCE). ACM.* (pp. 1–9).

Leifheit, L., Tsarava, K., Ninaus, M., & Moeller, K. (2018). *"Verstehen wie Computer denken". Ein Trainingsprogramm zur Förderung von informatischem Denken und systematischen Problemlösefähigkeiten besonders begabter Kinder im Grundschulalter. Reihe Hector Core Courses.[TODO].* Available from the first author.

Leifheit, L., Tsarava, K., Ninaus, M., Ostermann, K., Golle, J., Trautwein, U., & Moeller, K. (2020). SCAPA: Development of a questionnaire assessing self-concept and attitudes toward programming. In *Proceedings of the 25th Conference on Innovation and Technology in Computer Science Education (ITiCSE). ACM.* (pp. 138–144).

Lendrum, A., & Wigelsworth, M. (2013). The evaluation of school-based social and emotional learning interventions: Current issues and future directions. *Psychology of Education Review*, *37*, 70–76.

Lister, R. (2016). Toward a developmental epistemology of computer programming. In *Proceedings of the 11th workshop in primary and secondary computing education* (pp. 5–16).

Little, R. J. (1988). A test of missing completely at random for multivariate data with missing values. *Journal of the American statistical Association*, *83*(404), 1198–1202.

Marsh, H. W., Köller, O., Trautwein, U., Lüdtke, O., & Baumert, J. (2005). Academic self-concept, interest, grades, and standardized test scores: Reciprocal effects models of causal ordering. *Child Development*, *76*(2), 397–416.

Muthén, L. K., & Muthén, B. O. (2012). *Mplus Version 7 user's guide*.

Nardelli, E. (2019). Do we really need computational thinking? *Communications of the ACM*, *62*(2), 32–35.

Ninaus, M., Pereira, G., Stefitz, R., Prada, R., Paiva, A., Neuper, C., & Wood, G. (2015). Game elements improve performance in a working memory training task. *International journal of serious games*, *2*(1), 3–16.

Noice, H., & Noice, T. (2001). Learning dialogue with and without movement. *Memory & cognition*, *29*(6), 820–827.

Oliver, J. S., & Simpson, R. D. (1988). Influences of attitude toward science, achievement motivation, and science self concept on achievement in science: A longitudinal study. *Science Education*, *72*(2), 143–55.

O'Donnell, C. L. (2008). Defining, conceptualizing, and measuring fidelity of implementation and its relationship to outcomes in k–12 curriculum intervention research. *Review of educational research*, *78*(1), 33–84.

Papert, S. (1980). *Mindstorms: Computers, Children, and Powerful Ideas*. NY: Basic Books.

Papert, S. (2005). You can't think about thinking without thinking about thinking about something. *Contemporary Issues in Technology and Teacher Education*, *5*(3), 366–367.

Piaget, J. (1972). Intellectual evolution from adolescence to adulthood. *Human development*, *15*(1), 1–12.

Preckel, F., Zeidner, M., Goetz, T., & Schleyer, E. J. (2008). Female 'big fish' swimming against the tide: The 'big-fish-little-pond effect'and gender-ratio in special gifted classes. *Contemporary Educational Psychology*, *33*(1), 78–96.

Prottsman, K. (2014). Computer science for the elementary classroom. *ACM Inroads*, *5*(4), 60–63.

Resch, A., & Isenberg, E. (2018). How do test scores at the ceiling affect value-added estimates? *Statistics and Public Policy*, *5*(1), 1–6.

Román-González, M. (2015). Computational thinking test: Design guidelines and content validation. In *Proceedings of EDULEARN15 conference* (pp. 2436–2444).

Román-González, M., Pérez-González, J.-C., & Jiménez-Fernández, C. (2017). Which cognitive abilities underlie computational thinking? criterion validity of the computational thinking test. *Computers in Human Behavior*, *72*, 678–691.

Rothenbusch, S., Zettler, I., Voss, T., Lösch, T., & Trautwein, U. (2016). Exploring reference group effects on teachers' nominations of gifted students. *Journal of Educational Psychology*, *108*(6), 883.

S4A, S. f. A. (n.d.). *Scratch for Arduino (S4A)*. Edutec at Citilab. Retrieved from `http://s4a.cat`

Schafer, J. L., & Graham, J. W. (2002). Missing data: our view of the state of the art. *Psychological methods*, *7*(2), 147.

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, *22*, 142–158.

Teague, D. M., Corney, M. W., Fidge, C. J., Roggenkamp, M. G., Ahadi, A., & Lister, R. (2012). Using neo-piagetian theory, formative in-class tests and think alouds to better understand student thinking: A preliminary report on computer programming. In *Proceedings of 2012 Australasian Association for Engineering Education (AAEE) Annual Conference*.

Torgerson, D., & Torgerson, C. (2008). *Designing randomised trials in health, education and the social sciences: an introduction*. Springer.

Trautwein, U., & Köller, O. (2005). Was lange währt, wird nicht immer gut. *Zeitschrift für Pädagogische Psychologie*, *17*(3/4), 199–209.

Tsarava, K., Leifheit, L., Ninaus, M., Golle, J., Trautwein, U., & Moeller, K. (in preparation). Computational thinking training: Implementation and effects on elementary school children's cognitive and computational thinking skills.

Tsarava, K., Leifheit, L., Ninaus, M., Román-González, M., Butz, M. V., Golle, J., ... Moeller, K. (2019). Cognitive correlates of computational thinking: Evaluation of a blended unplugged/plugged-in course. In *Proceedings of the 14th Workshop in Primary and Secondary Computing Education* (pp. 1–9).

Tsarava, K., Moeller, K., & Ninaus, M. (2018a). Board games for training computational thinking. In *International Conference on Games and Learning Alliance* (pp. 90–100).

Tsarava, K., Moeller, K., & Ninaus, M. (2018b). Training computational thinking through board games: The case of crabs & turtles. *International Journal of Serious Games*, *5*(2), 25–44.

Tsarava, K., Moeller, K., Pinkwart, N., Butz, M., Trautwein, U., & Ninaus, M. (2017). Training computational thinking: Game-based unplugged and plugged-in activities in primary school. In *European Conference on Games Based Learning* (pp. 687–695).

Vivian, R., Franklin, D., Frye, D., Peterfreund, A., Ravitz, J., Sullivan, F., ... McGill, M. M. (2020). Evaluation and assessment needs of computing education in primary grades. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 124–130).

Weiß, R., Albinus, B., & Arzt, D. (2006). *Grundintelligenztest Skala 2-Revision (CFT 20-R)*. Hogrefe.

Wigfield, A., & Cambria, J. (2010, mar). *Students' Achievement Values, Goal Orientations, and Interest: Definitions, Development, and Relations to Achievement Outcomes* (Vol. 30) (No. 1). Academic Press. Retrieved from `https://www.sciencedirect.com/science/article/pii/S0273229709000392`

Wilkins, J. L. (2004). Mathematics and science self-concept: An international investigation. *The Journal of Experimental Education*, *72*(4), 331–346.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–35.

Wing, J. M. (2011). Research notebook: Computational thinking — what and why. *The link magazine*, *6*.

Wing, J. M. (2014, Jan). *Computational Thinking Benefits Society.* Retrieved from `http://socialissues.cs.toronto.edu/index.html?p=279.html`

Yadav, A., Mayfield, C., Zhou, N., Hambrusch, S., & Korb, J. T. (2014). Computational thinking in elementary and secondary teacher education. *ACM Transactions on Computing Education (TOCE)*, *14*(1), 5.

Manuscript 5

Title:
“Verstehen wie Computer denken”. Ein Trainingsprogramm zur Förderung von informatischem Denken und systematischen Problemlösefähigkeiten besonders begabter Kinder im Grundschulalter

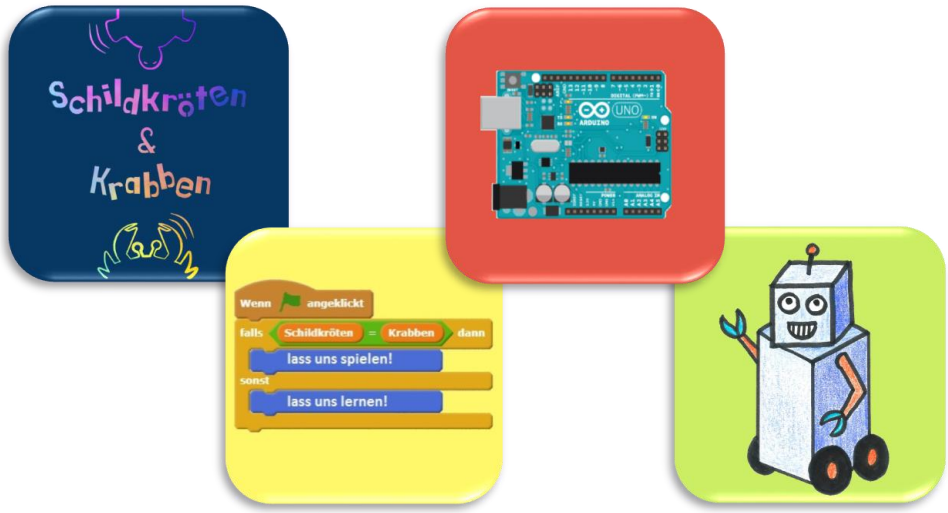
Authors:
Luzia Leifheit, Katerina Tsarava, Manuel Ninaus, and Korbinian Moeller

Unpublished manuscript

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Leibniz-Institut für Wissensmedien

„Verstehen wie Computer denken“



Ein Trainingsprogramm zur Förderung von
informatischem Denken und systematischen Problemlösefähigkeiten
besonders begabter Kinder im Grundschulalter

Reihe *Hector Core Courses*

Luzia Leifheit, Katerina Tsarava, Manuel Ninaus & Korbinian Moeller

Hector Stiftung II

Kontakt:

Luzia Leifheit

E-Mail: luzia.leifheit@uni-tuebingen.de

Eberhard Karls Universität Tübingen
LEAD Graduate School and Research Network
Walter-Simon-Straße 12 • 72072 Tübingen

Katerina Tsarava

E-Mail: k.tsarava@iwm-tuebingen.de

Leibniz-Institut für Wissensmedien
Schleichstraße 6 • 72076 Tübingen

Inhaltsverzeichnis

1 EINLEITUNG

Hector Kinderakademien

Im Land Baden-Württemberg entstanden durch die finanzielle Förderung der Hector Stiftung II seit dem Jahr 2010 66 Hector Kinderakademien (Stand September 2018). Bei den Hector Kinderakademien handelt es sich um ein freiwilliges, zusätzliches Angebot zur Förderung besonders begabter und hochbegabter Grundschulkinder.

Wissenschaftliche Begleitung der Hector Kinderakademien

Das Hector-Institut für Empirische Bildungsforschung (HIB) der Universität Tübingen ist gemeinsam mit dem Deutschen Institut für Internationale Pädagogische Forschung (DIPF) in Frankfurt für die formative Evaluation der Hector Kinderakademien verantwortlich (wissenschaftliche Begleitung). Die Erkenntnisse begleitender Untersuchungen fließen unmittelbar in die Arbeit der Hector Kinderakademien ein. Eine besondere Herausforderung ist es hierbei, die lokalen Besonderheiten der Hector Kinderakademien zu berücksichtigen und gleichzeitig eine an allen Standorten ähnliche und qualitätsgeprüfte Förderung zu etablieren.

Einheitliche Qualitätsstandards: *Hector Core Courses*

Um die Qualität der Kursangebote an allen Hector Kinderakademien sicherzustellen, hat die wissenschaftliche Begleitung damit begonnen, gemeinsam mit Dozentinnen und Dozenten aus den Hector Kinderakademien Kurse aus dem MINT-Bereich (Mathematik, Informatik, Naturwissenschaft, Technik) speziell für besonders begabte und hochbegabte Grundschulkinder zu entwickeln – die sogenannten *Hector Core Courses*. Alle Hector Core Courses basieren auf aktuellen Erkenntnissen der Fachdidaktik, der Psychologie und der Unterrichtsqualitätsforschung. Nach intensiven Evaluationsphasen stehen sie als flächendeckendes Angebot allen Hector Kinderakademien zur Verfügung.

Die Hector Core Courses

Seitens der Wissenschaftlichen Begleitung stehen allen Hector Kinderakademien im Schuljahr 2018/19 die folgenden Hector Core Courses zur Verfügung:

- Verstehen wie Computer denken
- Über Naturwissenschaften sprechen: Kleine Spezialisten – Wir präsentieren unser Wissen
- Kleine Forscher – Wir arbeiten wie Wissenschaftler
- Fit für die Mathematik-Olympiade
- Sicher experimentieren im Chemielabor
- Mathematik zum Anhören – Kinder komponieren mit LEGO
- Wie funktionieren Pflanzen
- Pneumatik
- Fischertechnik und elektrischer Strom
- Geheime Schriften

Kursentwicklung

Der Hector Core Course „Verstehen wie Computer denken“ wurde im Schuljahr 2017/18 in Zusammenarbeit mit den Hector Kinderakademien in Tübingen, Reutlingen, Loßburg und im Landkreis Esslingen erprobt und weiterentwickelt. Vielen Dank an Monika Reiff, Ottmar Glassmann, Dr. Andrea Krauß, Ingar Oppermann und Franziska Maier für ihren Einsatz und das Ermöglichen der Erprobung dieses Kurses an den von ihnen geleiteten Hector Kinderakademien.

Unser besonderer Dank gilt Helmut Fritsch für seine Erfahrungen als Leiter informatisch orientierter Kurse für begabte Kinder und für seine Unterstützung bei der erstmaligen Kursdurchführung.

Besonders bedanken möchten wir uns außerdem bei Dominik Krauß für seine engagierte Mitarbeit an der Kursentwicklung und für sein Feedback zur Umsetzbarkeit des Kurses.

Verstehen wie Computer denken

Das Ziel dieses Hector Core Courses ist die Förderung informatischen Denkens (engl. *computational thinking*) und das Wecken von Interesse an informatischen Fragestellungen bei begabten Grundschulkindern. Informatisch denken zu können bedeutet komplexe Probleme zu verstehen, präzise zu formulieren und in der Folge systematisch lösen zu können (Wing, 2014). Eine solch systematische Problemlösung erfordert typischerweise Fähigkeiten zur Verallgemeinerung, Abstraktion, Zerlegung in Teilprobleme und zum Entwickeln einer algorithmischen Lösungsstrategie (Wing, 2006; Briggs, 2014; Kazimoglu, 2013).

Die Kernkonzepte des informatischen Denkens (engl. *computational thinking*), die zur Entwicklung solch algorithmischer Lösungen verwendet werden, sind die grundlegenden Konzepte von Sequenzen, Schleifen, bedingten Verzweigungen, Ereignissen, Operatoren sowie Daten und Variablen (Brennan & Resnick, 2012). Die Förderung informatischen Denkens verfolgt das Ziel, die Entwicklung eines ersten Verständnisses dieser grundlegenden Konzepte und Prozesse zu ermöglichen. Deshalb werden die teilnehmenden Kinder auf spielerische Weise mit diesen Grundkonzepten vertraut gemacht und darin geschult, informatische Denkprozesse anzuwenden (z.B. Wing, 2010; Briggs, 2014).

Im Laufe des Kurses werden die Schülerinnen und Schüler von konkreten sogenannten „unplugged“-Aktivitäten (d. h. Aktivitäten, bei denen keinerlei technologische bzw. digitale Anwendungen zum Einsatz kommen) hin zu abstrakteren und anspruchsvolleren Programmierübungen in den „plugged in“-Aktivitäten (d. h. Aktivitäten, die technologische bzw. digitale Anwendungen einsetzen) geführt.

Die „unplugged“-Aktivitäten umfassen lebensgroße Lern-Brettspiele und -Kartenspiele sowie Übungsblätter mit Knobelaufgaben. Die Brett- und Kartenspiele wurden eigens für den Hector Core Course entwickelt (Tsarava et al. 2017; 2018). Diese Spiele ermöglichen eine verkörperlichte Erfahrung der grundlegenden Konzepte und Prozesse des informatischen Denkens (Barsalou, 2008), auf deren Basis konzeptionelle Abstraktionen auf natürliche Weise unterstützt werden können (Butz, 2016). Etwas verkörperlicht zu erfahren bedeutet, dass man etwas zum Beispiel durch Bewegung am eigenen Körper erlebt, statt es nur bildlich oder abstrakt dargestellt wahrzunehmen. Der spielbasierte Ansatz der eingesetzten Lernspiele motiviert die teilnehmenden Kinder zur vollen Teilnahme an den

Lernaktivitäten und zielt darauf ab, die Ausbildung und Entwicklung des informatischen Denkens der Schülerinnen und Schüler durch multimodale Darstellungen zu verbessern (Plass et al., 2015).

Bei den „plugged in"-Aktivitäten kommen unterschiedliche Programmieranwendungen zum Einsatz. Die speziell für Kinder entwickelte visuelle Blockprogrammiersprache und -umgebung *Scratch* wird verwendet, um es den Schülerinnen und Schülern zu ermöglichen, ihre mithilfe der Brettspiele erlernten informatischen Denkprozesse in konkrete Programmierprojekte einzubringen. Die Möglichkeiten von Scratch werden durch den Einsatz der „open hardware"-Plattform *Arduino* sowie der Programmiersprache und -umgebung *S4A* (*Scratch for Arduino*), über welche die unterschiedlichen Sensoren des Arduino angesteuert werden können, erweitert. Durch die Nutzung dieser Plattform erleben die Schülerinnen und Schüler, dass sie mit ihrem Programmcode nicht nur den Computer selbst befehligen, sondern auch mit der sie umgebenden physischen Welt interagieren können. Darüber hinaus können die Schülerinnen und Schüler ihr informatisches Denken selbstständig üben, indem sie in der interaktiven Robotersimulation *Open Roberta* einen simulierten Roboter mit Befehlen ihrer Wahl programmieren und steuern.

Der Einsatz informatischer Problemlösungsstrategien findet darüber hinaus viele Anwendungen in anderen MINT-Fächern (Sanders, 2009; Barr und Stephenson, 2011). Aus diesem Grund zielt der Kurs darauf ab, die in jeder Kurseinheit behandelten informatischen Prozesse auf unterschiedliche MINT-Fächer anzuwenden. So programmieren die Kinder in einer der Kurseinheiten beispielsweise Simulationen einer Nahrungskette oder eines Wasserkreislaufs, wodurch sie ihre bisher erarbeiteten informatischen Kompetenzen auf Themen aus dem Biologie- bzw. Erdkundeunterricht anwenden lernen. Dadurch soll ihnen die Perspektive eröffnet werden, dass informatische Kompetenzen für die Lösung von Problemen in unterschiedlichen Fachgebieten wertvoll sind.

Was macht diesen Kurs besonders?

Der Hector Core Course „Verstehen wie Computer denken" vermittelt Kompetenzen und Inhalte, die weit über das derzeitige Grundschulcurriculum hinausgehen, um begabte Schülerinnen und Schüler ihren Bedürfnissen entsprechend zu fordern und ihnen wertvolle Fähigkeiten und Kenntnisse für ihre persönliche Zukunft mitzugeben.

Der Kurs ist auf die Vermittlung derjenigen informatischen Konzepte und Denkprozesse ausgerichtet, die beim Programmieren eine wesentliche Rolle spielen. Das didaktische Konzept des Kurses basiert auf der multimodalen Einführung der einzelnen informatischen Konzepte durch „unplugged"- und „plugged in"-Aktivitäten und der Demonstration ihrer Anwendbarkeit in verschiedenen MINT-bezogenen Kontexten. Die Grundidee des Kurses folgt dem Motto "spielen – modifizieren – entwickeln". Die Schülerinnen und Schüler werden durch spielerische „unplugged"-Aktivitäten an die informatischen Konzepte und Prozesse herangeführt. Anschließend werden sie in den „plugged in"-Aktivitäten angeleitet, bestehende Programmierprojekte zu modifizieren, bevor sie schließlich eigenständig Ansätze für Programmierprojekte entwickeln.

Um jüngeren Schülerinnen und Schülern die Arbeit mit informatischen Problemstellungen und Themen nahezubringen, soll informatisches Denken durch den Einsatz von „unplugged"-Übungen gefördert werden (Prottsman, 2014). Dadurch werden diejenigen informatischen Konzepte vermittelt, die Schülerinnen und Schüler verstehen und anwenden müssen, wenn sie mit dem Erlernen des

Programmierens beginnen (Prottsman, 2014; Kumar, 2014). Dieser didaktische Ansatz stützt sich auf die Argumentation, dass ein Fokus auf *konzeptionelle* Grundlagen der Informatik den nachhaltigen Erwerb dieser Inhalte und Fähigkeiten sowie deren Transfer auf unterschiedliche Anwendungen erleichtert (Prottsman, 2014). Während spezifische technologische Anwendungen – Programmiersprachen eingeschlossen – sich fortlaufend entwickeln und verändern, bleiben die konzeptionellen Grundlagen der Informatik weitgehend dieselben (Wing, 2006) und sind universell auf unterschiedliche Anwendungen übertragbar (Wing, 2010).

Dabei kann der Einsatz *spielbasierter* Lehrmethoden die Leistung der Lernenden verbessern (Ninaus et al., 2015) und ihre Motivation sowie ihr Engagement beim Bearbeiten von Lernaufgaben erhöhen (Hamari et al., 2014). Deshalb werden den gesamten Kurs über immer wieder spielbasierte Lehrmethoden eingesetzt, angefangen bei den lebensgroßen Brett- und Kartenspielen bis hin zum Programmieren von Computerspielen.

Verkörperlichte Lernmethoden erleichtern das implizite Lernen (Barsalou, 2008) und haben durch die körperliche Organisation der konzeptionellen Inhalte einen positiven Einfluss auf das Vermögen, sich an diese Inhalte zu erinnern (Noice & Noice, 2001). Daher kommen in diesem Kurs lebensgroße Brett- und Kartenspiele zum Einsatz, in denen die Kinder verkörperlicht erfahren können, worin beispielsweise der Unterschied zwischen Sequenzen und Schleifen besteht. Darüber hinaus bietet die Arbeit mit der „open hardware“-Plattform Arduino den Kindern die Möglichkeit, verkörperlicht zu erleben, wie sie mit ihrem Programmcode die physische Welt beeinflussen können.

Aufgrund der angeführten positiven Auswirkungen dieser Lehr- und Lernmethoden basiert der Hector Core Course „Verstehen wie Computer denken“ auf einem didaktischen Ansatz, der (i) konzeptionell, (ii) spielbasiert und (iii) verkörperlicht ist.

2 THEORETISCHER HINTERGRUND

Die am Hector Core Course „Verstehen wie Computer denken“ teilnehmenden Schülerinnen und Schüler bringen in der Regel noch keine Programmiererfahrung oder Informatikkenntnisse mit, da die Vermittlung informatischer Inhalte und Kompetenzen in der Grundschule bisher nicht in den Bildungsplänen der Bundesländer verankert ist. Es ist daher wichtig, den teilnehmenden Kindern den Einstieg in das Fachgebiet möglichst einfach zugänglich zu machen.

Da die Grundschulcurricula sowie Unterstufencurricula der weiterführenden Schulen in den deutschen Bundesländern derzeit noch nicht gewährleisten, dass die Kinder nach Abschluss des Kurses weiterhin unter fachlicher sowie pädagogischer Anleitung ihre informatischen Kenntnisse und Kompetenzen erweitern können, sollen die Kinder durch den Kurs außerdem dazu befähigt werden, eigenständig programmieren und sich mit informatischen Fragestellungen beschäftigen zu können.

Mit dem Ziel, den Kindern schnell erlernbare, gut strukturierte und einfach auf unterschiedliche Anwendungen übertragbare Informatikkenntnisse zu vermitteln, folgt der Kurs dem fachdidaktischen Ansatz des *informatischen Denkens*. Damit die Kinder diese Kenntnisse möglichst nachhaltig verinnerlichen, um nach Abschluss des Kurses eigenständig an Programmierprojekten arbeiten und sich informatische Problemstellungen erschließen zu können, werden die Kursinhalte mit Methoden des *verkörperlichten Lernens* eingeführt und geübt. Damit die Kinder außerdem die Freude an der Arbeit mit informatischen Problemstellung vermittelt bekommen und sie zur aktiven Teilnahme an den Aktivitäten des Kurses motiviert werden, kommen außerdem *spielbasierte Lehrmethoden* zum Einsatz.

Um den Kursleitungen einen fundierten Überblick über den wissenschaftlich-theoretischen Hintergrund von informatischem Denken, verkörpertem Lernen und spielbasierten Lehrmethoden zur Verfügung zu stellen, werden diese drei theoretischen Ansätze in diesem Kapitel näher beleuchtet.

2.1 Informatisches Denken (engl. *computational thinking*)

Der Begriff des informatischen Denkens ist eine sinngemäße Übersetzung des in der englischsprachigen Fachliteratur gebräuchlichen Begriffs *computational thinking*, welcher von Seymour Papert (1980) geprägt wurde, um diejenigen Denkprozesse zu beschreiben, welche bei der systematischen Entwicklung informatischer Prozeduren eine grundlegende Rolle spielen.

Heutzutage wird der Begriff in unterschiedlichen Zusammenhängen verwendet. Zum einen wird mit ihm auf die kognitiven Prozesse und Fähigkeiten Bezug genommen, die für die Lösung informatischer Problemstellungen eine wesentliche Rolle spielen (Garcia-Peñalvo et al., 2016). Zum anderen werden durch diesen Begriff bestimmte erlernbare Denkstrategien bezeichnet, die zur systematischen Problemlösung eingesetzt werden können. Außerdem bezieht sich der Begriff des informatischen Denkens auf einen fachdidaktischen Ansatz zur Vermittlung grundlegender informatischer Inhalte und Kompetenzen.

Im Folgenden wird der Begriff des informatischen Denkens zunächst von Informatik, Programmierung und allgemeiner Problemlösefähigkeit abgegrenzt. Anschließend werden die kognitive,

strategiebezogene und didaktische Dimension des Begriffs des informatischen Denkens vorgestellt. Darüber hinaus wird die Relevanz informatischen Denkens im Bildungskontext der Grundschule erläutert.

2.1.1 Begriffliche Abgrenzung und Definition

Um in seinem begrifflichen Umfang definiert zu werden, muss das informatische Denken einerseits von der Informatik selbst sowie von der Programmierung und andererseits von einem weiter gefassten Begriff des Problemlösens unterschieden werden.

Während die verschiedenen Schwerpunkte in der Informatik von theoretischen Fragestellungen über die Entwicklung von Software bis hin zu den eher technologischen und physikalischen Anwendungen in der technischen Informatik reichen, steht beim informatischen Denken die Idee der algorithmischen Problemlösung im Mittelpunkt (Wing, 2006). Die Informatik selbst ist eine breit gefächerte akademische Disziplin, während informatisches Denken einen kognitiven Prozess oder eine grundlegendere Fähigkeit bezeichnet (Lu & Fletcher, 2009), die für den Erfolg in jedem Bereich der Informatik, aber auch darüber hinaus, erforderlich ist.

Informatisches Denken ist eine grundlegende Voraussetzung für das Programmieren, aber nicht mit ihm identisch. Programmieren ist das Schreiben algorithmischen Codes, um Anweisungen an einen „Rechner“ zu geben. Der „Rechner“, der diesen Code ausführt, kann ein Computer – also eine Maschine – oder ein Mensch sein (Wing, 2014). Programmieren unterscheidet sich von Informatik dadurch, dass sie als technische Fertigkeit erworben werden kann und nicht unbedingt eine breitere Kenntnis der theoretischen Details oder technologischen Implementierungen über den Code hinaus erfordert. Darüber hinaus besteht der Unterschied zwischen Programmieren und informatischem Denken darin, dass ersteres die Implementierung von Algorithmen in Form von Programmcode bezeichnet, während es bei letzterem um die grundlegende Fähigkeit geht, algorithmisch zu denken, ohne notwendigerweise auf tatsächliche Computerprogramme angewendet zu werden.

Schließlich hebt sich informatisches Denken von allgemeinem Problemlösen ab, also von Fähigkeiten und Strategien, Probleme jeglicher Art zu lösen. Der Begriff des informatischen Denkens ist präziser und enger gefasst als der des Problemlösens: Es geht nicht nur darum, Lösungen für Probleme zu finden, sondern darum, *algorithmische* Lösungen zu finden (Barr & Stephenson, 2011), die strategisch, systematisch, abstrakt, reproduzierbar und berechenbar sind – es spielt jedoch keine Rolle, ob die Berechnung letztendliche von einer Person oder einer Maschine ausgeführt wird (Wing, 2014). Während informatisches Denken also durchaus als eine Unterart des Problemlösens im allgemeinen Sinne angesehen werden kann, sind die Begriffe keineswegs identisch oder austauschbar. Da allerdings viele Probleme auf strategische, systematische, abstrakte, reproduzierbare, algorithmische und berechenbare Weise verstanden und potentiell gelöst werden können, kann informatisches Denken viel zur allgemeinen Problemlösung beitragen.

2.1.2 Die kognitive Dimension informatischen Denkens

Informatisches Denken wird als die Fähigkeit bezeichnet, komplexe Probleme zu verstehen, präzise zu formulieren und in der Folge systematisch lösen zu können (Wing, 2014).

Im Gegensatz zur kognitiven Grundlage von z.B. mathematischen oder sprachlichen Fähigkeiten ist jedoch unklar, auf welchen grundlegenden kognitiven Fähigkeiten und Prozessen informatisches Denken basiert, da zu dieser Frage bislang nur sehr wenig empirische Forschung durchgeführt wurde. Daher soll in der im Schuljahr 2018/19 stattfindenden Evaluationsstudie zum Hector Core Course „Verstehen wie Computer denken“ unter anderem untersucht werden, ob es Zusammenhänge zwischen informatischem Denken auf der einen Seite und mathematischen, verbalen, logischen sowie visuell-räumlichen Fähigkeiten auf der anderen Seite gibt.

Doch weshalb werden die als „informatisches Denken“ bezeichneten Fähigkeiten und Denkprozesse überhaupt als informatisch bezeichnet? Informatische Denkprozesse lassen sich definieren als „diejenigen Denkprozesse, die an der Formulierung von Problemen und deren Lösungen beteiligt sind, sodass die Lösungen in einer Form dargestellt werden, die von einem informationsverarbeitenden Akteur effektiv ausgeführt werden können" (Cuny et al., 2010, Übersetzung der Verfasserin). Informatisches Denken bezeichnet die Idee, eine verallgemeinerte Lösung für ein Problem zu entwickeln, indem man es zerlegt, relevante Variablen und Muster identifiziert und einen algorithmischen Lösungsprozess ableitet (z.B. Wing, 2006; Kazimoglu, 2013). Dieser Prozess stimmt weitgehend mit dem Vorgehen beim Programmieren oder bei der strategischen Lösung informatischer Problemstellungen überein. Informatisches Denken stützt sich insbesondere auf Prozesse wie algorithmisches Denken, konditionale Logik, Zerlegung in Teilprobleme, Abstraktion, Mustererkennung, Parallelisierung und Verallgemeinerung (z.B. Wing, 2010; Briggs, 2014) und reflektieren damit kognitive Instanziierungen von Konzepten, die beim Programmieren eine zentrale Rolle spielen. Wichtig ist, dass diese Konzepte im informatischen Denken nicht als domänspezifisch zu verstehen sind, da sie nicht nur innerhalb der Informatik oder beim Programmieren anwendbar sind. Daher wird informatisches Denken mitunter als eine grundlegende kognitive Fähigkeit bezeichnet, die in der schulischen Bildung ähnlich wie grundlegende Lese-, Schreib- und Rechenfähigkeiten gefördert werden sollte (Yadav et al., 2014).

2.1.3 Informatisches Denken als erlernbare Denkstrategie

Komplexe Probleme, deren Lösung informatisches Denken erfordert, werden häufig durch eine Vielzahl von Faktoren bestimmt. Deshalb erfordert der Umgang mit solchen Problemen oft Lösungswege, die Verallgemeinerung, Abstraktion, Zerlegung in Teilprobleme und das Entwickeln einer algorithmischen Lösungsstrategie beinhalten (Wing, 2006; Briggs, 2014; Kazimoglu, 2013). Algorithmisch ist eine Lösungsstrategie dann, wenn sie auf eine Weise formuliert werden kann, sodass sie eindeutig ist und keinen Spielraum für Interpretation offenlässt. Das heißt, dass in einer solchen Lösungsformulierung nach jedem Schritt klar ist, welcher Schritt als nächstes ausgeführt werden soll oder anhand welcher Bedingungen entschieden werden soll, welcher Schritt als nächstes ausgeführt wird.

Informatisches Denken von Kindern zu schulen, bedeutet also nicht, sie zu Programmierern oder Programmiererinnen zu machen, sondern ihnen zu helfen, ihre Fähigkeit weiterzuentwickeln, komplexe Probleme zu verstehen und Strategien zu deren systematischer Lösung zu finden (Wing, 2014). Dieser Prozess des Verstehens und Lösens von Problemen besteht aus mehreren Schritten, die wie folgt beschrieben werden können:

(1) Das Problem verstehen: Den Kern des Problems sowie seinen Kontext, seine bestimmenden Parameter und, falls erforderlich, seine mehrstufige Struktur verstehen.

(2) Das Problem präzise formulieren: Das Problem so ausdrücken, dass eine algorithmische Lösung möglich ist. Dazu gehört die Aufteilung des Problems in einfacher lösbare Teilprobleme sowie die Verallgemeinerung konkreter Elemente des Problems durch Abstraktion.

(3) Das Problem lösen: Einen Lösungsalgorithmus für das ausformulierte Problem entwickeln, der aus einer geordneten Menge von Anweisungen besteht.

(4) Implementierung und Bewertung der Lösung: Den im vorherigen Schritt entwickelten Algorithmus ausführen oder ausführen lassen. Die ausgeführte Lösung daraufhin bewerten, ob sie funktioniert und alle wichtigen Anforderungen des Problems erfüllt werden. Falls nicht, analysieren, warum die Lösung unbefriedigend oder nicht praktikabel ist.

(5) Die Lösung verbessern: Fehlerhafte oder unzureichende Teile des Algorithmus verbessern, um die erwünschten Ergebnisse zu erzielen.

Beim Befolgen dieser Schritte zum Entwickeln einer algorithmischen Problemlösung werden die bereits erwähnten informatischen Denkprozesse der Verallgemeinerung, Abstraktion und Zerlegung in Teilprobleme angewendet. Die konzeptionellen „Bausteine“ einer solchen algorithmischen Problemlösung sind für gewöhnlich informatische Denkkonzepte wie Sequenzen, Schleifen, bedingte Verzweigungen, Ereignisse, Operatoren sowie Daten und Variablen. Darüber hinaus kommen häufig Funktionen, Prozeduren oder Methoden zum Einsatz, die jedoch aufgrund ihrer vergleichsweise höheren Komplexität, im Vergleich zu den anderen genannten informatischen Denkkonzepten, nicht Teil des Kursinhaltes sein sollen.

2.1.4 Informatisches Denken als didaktischer Ansatz

Bei der Vermittlung der Inhalte und Kompetenzen folgt der Hector Core Course „Verstehen wie Computer denken“ dem fachdidaktischen Ansatz des informatischen Denkens. Das bedeutet, dass der Fokus des Kurses stets auf den konzeptionellen Grundlagen und nicht auf spezifischen Programmieranwendungen oder Technologien liegt. Der anfängliche Erwerb informatischen Denkens wird weitgehend durch den Einsatz von „unplugged“-Übungen unterstützt.

Ein solcher Ansatz wird von Prottsman vorgeschlagen, um jüngeren Schülerinnen und Schülern informatische Themen und Problemstellungen nahezubringen. Dabei sollen „unplugged“-Übungen genutzt werden, um das Erlernen der abstrakten informatischen Konzepte zu erleichtern, auf die die Schülerinnen und Schüler stoßen werden, wenn sie anfangen programmieren zu lernen (Prottsman, 2014), denn für den Erwerb praktischer Programmierfertigkeiten ist zumindest die Kenntnis grundlegender informatischer Konzepte unerlässlich (Balanskat & Engelhardt 2015; Garcia-Peñalvo et al., 2016). Gleichzeitig könnte die Förderung informatischen Denkens, losgelöst von tatsächlichem Programmieren, zu eher künstlichen und kontextlosen Lernszenarien führen. Daher ist es sinnvoll, informatisches Denken zu fördern, indem es konzeptionell gefestigt, in der praktischen

Programmierung angewendet und auf fächerübergreifende Kontexte übertragen wird (z.B. Yadav et al., 2016).

2.1.5 Informatik und informatisches Denken in der Grundschule

Informatisches Denken bei Kindern zu fördern, spiegelt die „Vision des 21. Jahrhunderts wider, dass Schülerinnen und Schüler nicht nur Computerbenutzer, sondern auch informatisch versierte Entwickler sein sollten“ (https://k12cs.org/). Es überrascht nicht, dass die Idee, bereits in der Grundschule mit der gezielten Förderung und Vermittlung von Programmierfähigkeiten zu beginnen, immer beliebter wird (z.B. Balanskat & Engelhardt, 2015; https://code.org/).

In ganz Europa nehmen die Bemühungen um Bildung in logischem Denken, Problemlösen und Programmierung zu. Das schlägt sich auch darin nieder, dass immer mehr Länder bereits in der Grundschule informatische Grundlagen und Programmierfähigkeiten in die Lehrpläne integrieren oder planen zu integrieren (z.B. Balanskat & Engelhardt, 2015).

Im Jahr 2012 veröffentlichte die British Royal Society in Großbritannien die Erklärung, dass „jedes Kind die Möglichkeit haben sollte, in der Schule Grundlagen der Informatik oder Programmierung zu lernen“ (Wing, 2014). Anschließend wurde im September 2014 das Programm „Computing at School“ (Computing at School) gestartet, welches „computing“ als Schulfach für alle Schülerinnen und Schüler in jedem Schuljahr der Grund- und Sekundarschulbildung einführte (Dredge, 2014). Darüber hinaus wurde 2013 der Lehrplan für Informatik an Universitäten in Großbritannien überarbeitet, wobei der Fokus auf einer Förderung von informatischem Denken als allgemein anwendbare und übertragbare Fähigkeit in der Informatik lag (Brown et al., 2014).

Die Relevanz und Bedeutung informatischen Denkens spiegelt sich auch in der Gründung der European Coding Initiative im Jahr 2014 wider. In Zusammenarbeit mit mehreren europäischen Bildungsministerien, Mitgliedern des Europäischen Schulnetzes und mithilfe der Unterstützung technologischer Großunternehmen (z.B. Microsoft, Facebook) zielt die Initiative darauf ab, eine sinnvolle Einbindung von informatischem Denken und Programmierung in die offiziellen Lehrpläne zu fördern (Balanskat & Engelhardt, 2015).

In den USA gibt es zahlreiche aktuell laufende Programme – z.B. *CE21*, *CPATH*, *BPC*, *CS 10K*, *CS4HS* oder *Computing in the Core*, um nur einige zu nennen (Wing, 2014) –, die sowohl darauf abzielen, den Informatikunterricht in der Grund- und Sekundarschulbildung zu erweitern, als auch darauf, Lehrerinnen und Lehrer auf das Unterrichten in diesem Fach vorzubereiten (Wing, 2014). Ähnliche Entwicklungen wie in Großbritannien und den USA sind außerdem in China, Korea und Singapur zu beobachten (Wing, 2014).

Diesen Entwicklungen entsprechend werden Programmierfähigkeiten und das Verständnis der zugrunde liegenden Konzepte und kognitiven Prozesse oft als eine grundlegende Fähigkeit für das 21. Jahrhundert beschrieben (z.B. Yadav et al., 2014). Bereits Papert (1972) deutete die Möglichkeit an, dass das Erlernen von Programmierfertigkeiten Kindern helfen könnte, ihre eigenen Denkprozesse zu reflektieren, denn das Entwickeln interaktiver Programme erfordert die Berücksichtigung und das Voraussehen aller möglichen Missverständnisse und Fehler, die einer

Benutzerin oder einem Benutzer des Programms begegnen oder unterlaufen könnten. Aufgrund der elementaren Wichtigkeit der Fähigkeit zum Verstehen und Reflektieren des eigenen Denkens schlägt Papert (2005) vor, dass das Unterrichten von grundlegenden informatischen Konzepten und Prozessen in der Grundschule vorteilhaft für die Entwicklung der Denkfähigkeiten von Kindern sein kann.

Es ist jedoch wichtig, dass die Schülerinnen und Schüler kognitiv weit genug entwickelt sind, um sich informatisches Denken aneignen zu können. Für erste Einblicke in die Grundlagen der Informatik ist das Grundschulalter deshalb ein geeigneter Zeitpunkt, da die konkret-operationale Phase der kognitiven Entwicklung (nach Piaget, 1972) typischerweise im Alter von 7 bis 11 Jahren erreicht wird. Diese Phase ist nach neopiagetischen Modellen der kognitiven Entwicklung im Hinblick auf den Erwerb von informatischen Fähigkeiten „die erste Phase, in der Schülerinnen und Schüler einen zielgerichteten Ansatz zum Entwickeln von Programmcode zeigen“ (Lister, 2016) und „anhand von klar definierten Vorgaben kleine Programme schreiben können“ (Teague et al., 2012). Für die frühe Informatikbildung schlägt Prottsman (2014) den didaktischen Ansatz des informatischen Denkens vor, bei dem „unplugged“-Übungen (mit greifbaren Materialien wie Stiften, Spielkarten usw.) verwendet werden, um das Erlernen abstrakter informatischer Konzepte zu unterstützen, die Kinder anwenden können müssen, wenn sie anfangen programmieren zu lernen (Kumar, 2014).

Aus diesem Grund wurden bei der Entwicklung des Hector Core Courses „Verstehen wie Computer denken“ Erkenntnisse der Entwicklungspsychologie und Bildungsforschung berücksichtigt und in die Konzeption des Kurses integriert. Um den Schülerinnen und Schülern einen altersgerechten Zugang zu den Inhalten, Arbeitsweisen und Problemstellungen des Kurses zu ermöglichen und ihr Verständnis der informatischen Konzepte sowie ihre Motivation zum Lernen zu fördern, kommen Methoden des spielbasierten und des verkörperten Lernens zum Einsatz.

2.2 Spielbasiertes Lernen (engl. *game-based learning*)

Lernaktivitäten, bei denen die Lernenden Inhalte und Kompetenzen anhand von Spielen oder spielerischen Aufgaben erarbeiten und vermittelt bekommen, werden als spielbasierte Lernaktivitäten bezeichnet. Der Einsatz von spielbasierten Lehr- und Lernmethoden kann die Leistung der Lernenden verbessern (Ninaus et al., 2015; Wouters et al., 2013) und ihre Motivation und ihren Einsatz bei der Lernaktivität erhöhen (Hamari et al., 2014).

Spiele und spielbasierte Aktivitäten sind ein zunehmend wichtiger Ansatz für kognitive Trainings-, Lern- und Bildungsmaßnahmen (Boyle et al., 2016), da sie die Lernenden motivieren, aktiv mit der Lernumgebung zu interagieren (Plass et al. 2015). Aktuelle Studien legen sogar nahe, dass spielbasiertes Lernen ein effektiveres Mittel für das Erlernen und langfristige Verinnerlichen von Inhalten und Kompetenzen sein könnte als herkömmliche Unterrichtsmethoden (Wouters et al., 2013).

In den letzten Jahren haben sich immer mehr Brettspiele als informeller und interaktiver Kontext erwiesen, in dem informatisches Denken angewendet und geübt werden kann. So sind beispielsweise Pandemie (Leacock, 2008) und RaBit EscApe (Apostolellis et al., 2014) zwei strategische Brettspiele, in denen informatisches Denken in ein kooperatives Spielszenario eingebettet wurde. Schülerinnen

und Schüler der dritten und vierten Klasse, die mithilfe von „unplugged"-Spielen in informatische Konzepte eingeführt werden, sind anschließend meist in der Lage, diese Konzepte zu verstehen und anwenden zu können (Leifheit et al., 2018). Vor diesem Hintergrund nutzt der Hector Core Course „Verstehen wie Computer denken" lebensgroße Brett- und Kartenspiele, die in Abschnitt 3.4 „Einsatz der lebensgroßen Brett- und Kartenspiele" im Detail vorgestellt werden (siehe auch Tsarava, Moeller, & Ninaus, 2018).

Mithilfe dieser Spiele bekommen die Schülerinnen und Schüler durch die Regeln und den Ablauf der Spiele vermittelt, wie unterschiedliche informatische Konzepte funktionieren. Dabei dient das aktive Handeln der Spielenden dazu, die Motivation zur Anteilnahme am Lerngeschehen zu erhöhen (siehe Echeverría et al., 2011 für einen Überblick). Darüber hinaus zielen die in den Kurs integrierten Brett- und Kartenspiele darauf ab, die Entwicklung abstrakten und symbolischen Denkens der Schülerinnen und Schüler durch multimodale Darstellungen (Plass et al., 2015) sowie die Vereinfachung und Konkretisierung informatischer Konzepte zu verbessern.

2.3 Verkörperlichtes Lernen (engl. *embodied learning*)

Die Theorie der Verkörperlichung (engl. *embodiment*) besagt, dass viele Aspekte der Kognition stark vom physischen Körper einer oder eines Handelnden abhängig sind, insofern der Körper dieser oder dieses Handelnden eine bedeutende kausale Rolle oder eine physisch konstitutive Rolle bei der kognitiven Verarbeitung von erlebten Vorgängen und Situationen spielt (Wilson & Foglia, 2017).

Lernaktivitäten, bei denen die Lernenden Inhalte und Kompetenzen mithilfe von körperlichen Aktivitäten oder durch die Manipulation physischer Gegenstände erlernen, werden als verkörperlichtes Lernen (engl. *embodied learning*) bezeichnet. Verkörperlichte Lehr- und Lernmethoden sollen implizites Lernen erleichtern (Barsalou, 2008) und Verstehen sowie Erinnerungsvermögen durch erhöhte räumliche Organisation konzeptioneller Inhalte positiv beeinflussen (Noice & Noice, 2001).

Die für den Kurs entwickelten lebensgroßen Brett- und Kartenspiele sollen eine verkörperlichte Erfahrung der grundlegenden Konzepte und Prozesse des informatischen Denkens (Barsalou, 2008) ermöglichen, auf deren Basis konzeptionelle Abstraktionen auf natürliche Weise unterstützt werden können (Butz, 2016). So wird in einem dieser Brettspiele beispielsweise das Konzept einer Schleife eingeführt und geübt, indem Wiederholungen in sequentiellen Bewegungsfolgen einer Spielfigur erkannt und anschließend als Schleifen von Bewegungsfolgen neu formuliert werden.

Darüber hinaus werden im Kurs die „open hardware"-Plattform *Arduino* sowie die Programmiersprache und -umgebung *S4A* (*Scratch for Arduino*) eingesetzt, um verkörperlichtes Lernen zu ermöglichen. Indem sie die unterschiedlichen Sensoren des Arduino ansteuern und die Funktionalität der elektronischen Komponenten programmieren, erleben die Schülerinnen und Schüler, dass sie beim Programmieren nicht nur beeinflussen können, was vor ihren Augen auf dem Bildschirm passiert, sondern durch die von ihnen geschriebenen Programme auch mit der sie umgebenden physischen Welt interagieren können.

3 KURSMATERIALIEN UND ORGANISATORISCHES

3.1 „Schildkröten & Krabben“

Schildkröten & Krabben ist eine Serie von drei lebensgroßen Lernspielen: 1. *Die Schatzsuche*, 2. *Die Muster* und 3. *Das Wettrennen*. Bei *Die Schatzsuche* und *Das Wettrennen* handelt es sich um Brettspiele, während *Die Muster* ein Kartenspiel ist. Die preisgekrönte Spieleserie belegte beim internationalen Lernspielwettbewerb 2018 den ersten Platz[1]. Derzeit sind alle drei Spiele auf Deutsch, Englisch ("Crabs & Turtles") und Griechisch ("Χελώνες & Κάβουρες") erhältlich.

Die Spieleserie kann über die auf der offiziellen Webseite angegebenen Kontakte bezogen werden: https://crabsturtles.iwm-tuebingen.de/index.php/contact/

Die Spiele führen Kinder spielerisch an grundlegende informatische Konzepte heran. Zielgruppe der Spiele sind Kinder im Grundschul- oder Unterstufenalter (8 bis 12 Jahre), die bereits lesen, schreiben und rechnen können, aber noch nicht über Programmierkenntnisse verfügen. *Schildkröten & Krabben* ist jedoch auch für ältere Schülerinnen und Schüler sowie für Erwachsene ohne Vorkenntnisse im Programmieren geeignet.

Schildkröten & Krabben wurde mit einem lebensgroßen Spieldesign entwickelt, um die aktive Beteiligung am Spielgeschehen zu fördern und damit die Motivation der Kinder zu erhöhen (für einen Überblick siehe Echeverría et al., 2011) und um den Lernerfolg durch konzeptionelle Abstraktion zu unterstützen, indem grundlegende informatische Konzepte verkörperlicht werden (Butz, 2016). Die Spiele sind bewusst so konzipiert, dass sie unabhängig von bestimmten Programmierumgebungen oder -sprachen sind. Sie wurden als Brett- und Kartenspiele und nicht als digitale Spiele entwickelt, um die Spielenden erleben zu lassen, dass Anwendungen informatischer Konzepte nicht auf digitale Zusammenhänge beschränkt sind, sondern auch auf nicht-digitale Kontexte verallgemeinert werden können (Tsarava et al., 2017).

Die Spiele können zwar eigenständig verwendet werden, wurden jedoch speziell für den Hector Core Course „Verstehen wie Computer denken“ entwickelt, dessen Kurssitzungen auf den im Spiel erworbenen Fähigkeiten aufbauen. Die Kursleitung spielt bei jedem dieser Spiele als Spielleitung eine wichtige Rolle und steht in engem Kontakt mit den Schülerinnen und Schülern.

Schildkröten & Krabben zielt darauf ab, kognitive Prozesse informatischen Denkens zu trainieren, wie z.B. algorithmisches Denken, Abstraktion oder Mustererkennung. Diese Prozesse können entweder auf spezifische informatische Konzepte (Sequenzen, Schleifen, bedingte Verzweigungen, Ereignisse und Muster) oder auf mathematische Fähigkeiten (Winkel, Addition, Subtraktion und Multiplikation) angewendet werden sowie auf Fähigkeiten, die sowohl beim Programmieren als auch in der Mathematik relevant sind (Umgang mit Operatoren, Variablen und Konstanten) (Tsarava et al., 2018).

[1] https://www.academic-conferences.org/conferences/ecgbl/ecgbl-future-and-past/

Im Folgenden werden die drei Spiele näher beschrieben und es wird erklärt, welche informatischen Konzepte und Prozesse in jedem von ihnen im Mittelpunkt stehen. Die detaillierten und bebilderten Anleitungen für alle drei Spiele befinden sich im Anhang des Kursmanuals.

Aktuelle Informationen zu *Schildkröten & Krabben* sind auf der offiziellen Webseite verfügbar: https://crabsturtles.iwm-tuebingen.de/

3.1.1 Schildkröten & Krabben: Die Schatzsuche

Die Schatzsuche ist das erste Spiel der Reihe *Schildkröten & Krabben*. Es wird in Zweierteams darum gespielt, welches Team als erstes eine bestimmte Anzahl Schätze sammeln kann. In diesem Spiel müssen Spielfiguren, die Schildkröten und Krabben darstellen, durch auf Spielkarten aufgedruckte Befehle gesteuert werden, um Schätze zu sammeln, die auf den Feldern des Spielbretts platziert sind. Um eine Krabbe oder Schildkröte zu bewegen, müssen die Spielenden aus Spielkarten Befehlsfolgen auf einem Klemmbrett erstellen. So müssen beispielsweise Sequenzen von Schritten und Drehungen erstellt werden, um eine Spielfigur schnell und effizient in Richtung der Schätze zu bewegen. Informatische Konzepte und – in geringerem Maße – mathematische Fähigkeiten, die durch dieses Spiel trainiert werden, sind räumliche Orientierung, Umgang mit Winkeln und Drehungen sowie die Bildung von Sequenzen und Schleifen (Tsarava et al., 2018). Wichtig ist, dass es einige Einschränkungen in Bezug auf Spielbrett und Art der Spielfigur (Schildkröte bzw. Krabbe) gibt, die die Strategie der Spielenden beeinflussen. Beispielsweise können sich Schildkröten nur Felder betreten, die Stein- oder Grasflächen anzeigen, während Krabben nur Felder betreten können, die Stein- oder Wasserflächen darstellen. Außerdem können sich die Schildkröten nur vorwärts und rückwärts gehen, während die Krabben nur nach links oder rechts laufen können. Unter bestimmten Bedingungen, z.B. wenn die Spielenden einer Krabbe eine Bonuskarte ziehen, die einer Figur erlaubt vorwärts oder rückwärts zu gehen, kann diese Karte auch von einer Krabbe so verwendet werden, wie sie von einer Schildkröte verwendet wird.

Die Hauptlernziele dieses Spiels sind die allgemeine Einführung in das algorithmische Denken, die Verwendung von Befehlen in bestimmter sequentieller Reihenfolge sowie die Berücksichtigung von Einschränkungen bei der Entwicklung der strategischen Lösung eines Problems (Tsarava et al., 2018). Mithilfe des Spiels sollen die Kinder ein Verständnis dafür entwickeln, was einfache Algorithmen sind und wie sie als Sequenzen mehrerer Befehle gebildet werden, die einem bestimmten strategischen Zweck dienen (Tsarava et al., 2018). Darüber hinaus sollten die Spielenden lernen, bei ihren Entscheidungen spezifische Einschränkungen zu berücksichtigen und kleine wiederholbare Muster zu erkennen, die in eine Schleife gefaltet werden können (Tsarava et al., 2018).

3.1.2 Schildkröten & Krabben: Die Muster

Die Muster ist ein Kartenspiel. Es wird nicht in Teams gespielt, sondern einzeln gegeneinander. In diesem Spiel müssen die Spielenden so schnell wie möglich Muster finden und Karten nach bestimmten Regeln vergleichen, um Paare zusammengehörender Karten zu finden. Dieses Verfahren bezeichnet man als Mustererkennung, welche beim Programmieren häufig angewendet

wird, z.B. bei der Zerlegung eines Problems in Teilprobleme, bei der Verallgemeinerung von Lösungen oder der Bildung von Schleifen (Tsarava et al., 2018).

Das Spiel ist in zwei Phasen gegliedert und benötigt eine Spielleitung. Ein Teil der Karten liegt offen und ungeordnet auf dem Boden, während die übrigen Karten vom Spielleiter nach und nach in zufälliger Reihenfolge aufgedeckt und den Spielenden gezeigt werden. Die Spieler sammeln jedes Mal eine Karte, wenn sie durch Mustervergleiche ein richtiges Kartenpaar finden. In der ersten Phase müssen die Spielenden so schnell wie möglich zwei Karten anhand der auf ihnen dargestellten Muster finden und zuordnen. Damit jemand ein Kartenpaar für sich beanspruchen kann, muss sie oder er am schnellsten das Kartenpaar finden und die Hand heben, um das Kartenpaar zu nennen. Die Spielleitung deckt nacheinander Karten auf, und die oder der erste, die oder der zwei Karten richtig abgleicht, erhält die aufgedeckte Karte und damit einen Punkt.

Um in der ersten Phase Karten zu kombinieren, müssen die folgenden drei Regeln befolgt werden: 1. Die beiden Karten sollten das gleiche Muster haben, 2. Sie sollten dasselbe Farbschema haben, aber 3. Sie sollten nicht die gleiche Farbe an der gleichen Position des Musters haben. In der zweiten Phase des Spiels müssen die Spielenden die Formen und Farben der aufgedeckten Karten anhand anderer Kriterien abgleichen und so schnell wie möglich der richtigen auf dem Boden liegenden Karte zuordnen.

Bei diesem Spiel handelt es sich um eine Einführung in das Konzept der Muster. Muster sind wichtige Konzepte im informatischen Denken, die sowohl zur Identifizierung von Abstraktionen als auch zur Verallgemeinerung verwendet werden (Curzon & McOwen, 2017). Lernziel dieses Spiels ist es, das Erkennen von Mustern nach bestimmten Regeln zu trainieren (Tsarava et al., 2018).

3.1.3 Schildkröten & Krabben: Das Wettrennen

Das Wettrennen ist ein Brettspiel und das dritte Spiel aus *Schildkröten & Krabben*. Ziel des Spiels ist, als erstes Zweierteam das Zielfeld zu erreichen, indem auf Ereignisse reagiert wird und mathematische Rätsel gelöst werden. Dazu müssen die Spielenden die Werte von Variablen verändern, mit Konstanten und Variablen rechnen und Entscheidungen in Abhängigkeit von Bedingungen treffen. Im Vergleich zu den anderen beiden Spielen liegt der Schwerpunkt von *Das Wettrennen* stärker auf informatischen Konzepten, die in engem Zusammenhang mit mathematischen Fähigkeiten stehen. Insbesondere das Verständnis von Operatoren, Variablen und Konstanten, der Umgang mit Ereignissen und das Befolgen bedingter Verzweigungen sind Fähigkeiten, die in diesem Spiel erarbeitet und geübt werden (Tsarava et al., 2018).

In diesem Spiel sind Zweierteams im Uhrzeigersinn nacheinander an der Reihe und müssen versuchen, mit ihrer Spielfigur so weit und so schnell wie möglich vorwärts zu kommen. Vor Spielbeginn wählt jedes Team aus, ob es eine Schildkröte oder eine Krabbe spielen möchte. Jedes Team erhält ein wiederbeschreibbares Variablen- und Notizbrett, einen wegwischbaren Filzstift und einen Schwamm. Zu Spielbeginn werden alle Spielfiguren auf dem Startfeld platziert. Wenn ein Team an der Reihe ist, würfelt es und geht so viele Schritte vorwärts, wie der Würfel angibt. Wenn seine Spielfigur ein kreisförmiges Feld betritt, zieht das Team eine Karte vom Kreiskartenstapel (Rätselkarten), ansonsten erhalten sie eine Karte vom Quadratkartenstapel (Ereigniskarten). Kreiskarten enthalten Rätsel in der Form mathematischer Aufgaben, die die Spielenden lösen

müssen. Dabei setzen sich die mathematischen Rätsel aus unterschiedlichen Variablen und Konstanten zusammen. Die Ereigniskarten beinhalten Anweisungen zum Verändern der Werte von Variablen, auf die die Teams reagieren müssen.

Hauptziel des Spiels ist die allgemeine Einführung informatischer Konzepte, die in engem Zusammenhang zur Mathematik stehen. Die Spielenden werden mit Variablen, Konstanten, Operatoren und Bedingungen vertraut gemacht. Das Spiel konzentriert sich auf das Üben des Umgangs mit Zahlenwerten innerhalb einfacherer arithmetischer Operationen, die Additionen, Subtraktionen und Multiplikationen enthalten. Die Operationen bestehen dabei aus visuellen Symbolen für Variablen und Konstanten, damit die Spielenden mit dem Erkennen symbolischer Darstellungen von Werten vertraut gemacht werden und sich an eine abstrakte Darstellungsweise gewöhnen (Tsarava et al., 2018). Darüber hinaus lernen die Spielenden auf Ereignisse zu reagieren und zu überprüfen, ob bestimmte Bedingungen erfüllt sind, um anhand dessen zu entscheiden, welche Verzweigung innerhalb eines Handlungsflusses gewählt werden muss.

3.2 Die Programmiersprache und -umgebung Scratch 2.0

Scratch ist eine visuelle Block-Programmiersprache, die am Massachusetts Institute for Technology (MIT) entwickelt wurde, um Kinder an das Programmieren heranzuführen (https://scratch.mit.edu).

Als Block-Programmiersprache wird Scratch deshalb bezeichnet, weil die einzelnen Befehle und Komponenten, aus denen sich Scratch-Programmcode zusammensetzen lässt, in Form vom bunten Blöcken visualisiert sind. Diese Blöcke können angeklickt und mit dem Mauszeiger zusammengefügt werden. Dabei lassen sich immer nur diejenigen Blöcke miteinander verbinden, die von ihrer Funktion her auch zusammenpassen und gemeinsam ausführbaren Programmcode ergeben.

Die Programmierumgebung sowie die Elemente der Sprache sind darauf ausgerichtet einzelne Objekte zu programmieren. Diese Objekte nehmen bei Scratch für gewöhnlich die Form von grafischen Figuren an, die durch den Programmcode verändert und bewegt werden können. Zum Beispiel ist das Standard-Objekt, das beim Start von Scratch bereits als Teil des Programms vorhanden ist, eine orangene Katze, deren Bewegung, Handlungen und Erscheinungsbild die Nutzerin oder der Nutzer von Scratch durch den eigenen Programmcode bestimmen kann. Durch das Programmieren der Eigenschaften und Funktionalität eines oder mehrerer solcher Objekte lassen sich Spiele oder filmartige Sequenzen programmieren, die in einem Bildschirmfenster ausgeführt werden können, sodass die Kinder genau sehen können, was sie programmiert haben. Deshalb wird Scratch auch als visuelle Programmiersprache bezeichnet.

In Modul 1 und 2 des Hector Core Courses „Verstehen wie Computer denken“ wird Scratch eingesetzt, um den Schülerinnen und Schülern zu ermöglichen, erste Erfahrungen im Anwenden informatischer Konzepte zum Entwickeln von einfachen Computerprogrammen zu nutzen. Dafür wird die Version Scratch 2.0 genutzt, da diese Version in der Onlineumgebung der offiziellen Scratch-Webseite verwendbar ist (https://scratch.mit.edu/projects/editor/), aber auch heruntergeladen und installiert werden kann (https://scratch.mit.edu/download).

3.3 Die „open hardware"-Plattform Arduino und S4A (Scratch for Arduino)

3.3.1 Der Arduino Uno

Der Arduino ist eine „open hardware"-Plattform, die aus einem mit einer Steckplatine verbundenen und mit diversen analogen und digitalen Ports ausgestatteten Microcontroller besteht (https://www.arduino.cc/en/Guide/Introduction).

Dass der Arduino eine „open hardware"-Plattform ist, bedeutet, dass seine Hardware nach frei verfügbaren Bauplänen erstellt wurde, die der Öffentlichkeit zugänglich sind und zur Entwicklung eigener Hardware-Projekte genutzt werden dürfen.

Der Arduino Uno ist eine Version der Arduino-Plattform, die speziell für den Einstieg in die Arbeit mit Hardware-Projekten und dem Programmieren von Microcontrollern entwickelt wurde. Detaillierte Beschreibungen der Komponenten des „Arduino Uno"-Kits sowie Erklärungen zu deren Funktionalität finden sich im Handbuch, das dem „Arduino Uno"-Kit beiliegt, oder unter:

https://www.arduino-tutorial.de/arduino-uno/

3.3.2 S4A (Scratch for Arduino)

Scratch for Arduino (S4A) ist eine auf Scratch basierende Programmiersprache und -umgebung, mit deren Hilfe sich die Funktionalität eines Arduino-Aufbaus programmieren lässt (http://s4a.cat).

S4A nutzt dieselben Blöcke und dieselbe Programmierumgebung wie Scratch 1.6. Zusätzlich dazu stellt S4A jedoch spezielle Blöcke zur Verfügung, mit denen die digitalen Ports des Arduino angesteuert werden können und sich kontrollieren lassen und mit denen Sensordaten eingelesen werden können, die die analogen Ports des Arduino empfangen. Darüber hinaus ist die Programmierumgebung von S4A gegenüber Scratch um ein zusätzliches Fenster erweitert, das eine Vorschau auf den Arduino sowie eine Anzeige der aktuell eingelesenen Sensorwerte enthält.

Das „Arduino Uno"-Kit und S4A werden in Modul 3 des Hector Core Courses „Verstehen wie Computer denken" eingesetzt, um die in Modul 1 eingeführten und in Modul 2 mithilfe des Programmierens von einfachen Computerspielen und Simulationen in Scratch vertieften informatischen Konzepte auf einen Anwendungsbereich zu übertragen, der über das Erstellen einfacher Bildschirmanwendungen hinausgeht und den Kindern erste Einblicke in das Gebiet der Hardware-Programmierung eröffnet. Außerdem dient die Arbeit mit S4A und dem Arduino dazu, die teilnehmenden Kinder in einer Form des verkörperlichten Lernens erleben zu lassen, dass sie mit ihrem Programmcode nicht nur bestimmen können, was auf dem Bildschirm direkt vor ihren Augen geschieht, sondern dass sie mittels Programmierung sogar Einfluss auf Geschehnisse in der sie umgebenden physischen Welt nehmen können.

3.4 Die interaktive Robotersimulation Open Roberta

Open Roberta ist eine für Kinder entwickelte Programmiersprache und -umgebung, mit der sowohl simulierte Roboter als auch real existierende Roboter gesteuert und programmiert werden können

(https://www.roberta-home.de). Aktuell ist die Sprache kompatibel mit den Lern-Robotern bzw. Hardware-Plattformen *WeDo, Ev3, NXT, micro:bit, Bot'n Roll, NAO, Nepo4Arduino, BOB3* und *Calliope Mini* (https://lab.open-roberta.org). Darüber hinaus bietet Open Roberta die Option, den simulierten Roboter „Roberta" in der Umgebung *Open Roberta SIM* zu programmieren.

Diese Robotersimulation gibt den Schülerinnen und Schülern im letzten Modul des Hector Core Courses „Verstehen wie Computer denken" die Möglichkeit, ihre im Laufe des Kurses erworbenen informatischen Kenntnisse und Kompetenzen in einem für sie neuen Kontext auszutesten und dabei eigenständiger zu arbeiten als in den Modulen 1 bis 3. Open Roberta SIM bietet für diese Art des Arbeitens in mehrerlei Hinsicht besonders förderliche Rahmenbedingungen.

Erstens bietet Open Roberta SIM eine einfach erlernbare Blocksprache, sodass die Kinder sich schnell in der neuen Programmiersprache und -umgebung zurecht finden können. Zweitens wird durch die Verwendung der vorgegebene, aber leicht zugänglichen Simulationsumgebung dafür gesorgt, dass die Kinder sich nicht lange mit den rein grafischen Elementen ihrer Programme aufhalten können, sondern tatsächlich programmieren und strategisch denken müssen, um Roberta zu steuern. Schließlich motiviert das Steuern des Roboters zu freien Problemlösungsversuchen, da es keinen vorgegebenen Problemlösungsansatz gibt, sondern das Aufgabenziel auf vielen verschiedenen Wegen erreicht werden kann.

3.5 Die Übungsblätter und der Wettbewerb „Informatik-Biber"

Der „Informatik-Biber" ist ein jährlich in Deutschland stattfinder Schülerwettbewerb, für den Kinder und Jugendliche informatische Problemstellungen lösen müssen. Dieser Wettbewerb ist die deutsche Version des internationalen Bebras-Wettbewerbs für Informatik und informatisches Denken, der in 61 Ländern auf der ganzen Welt ausgerichtet wird (https://www.bebras.org).

Beim Informatik-Biber „setzen sich Schülerinnen und Schüler mit altersgerechten informatischen Aufgaben auseinander – spielerisch und wie selbstverständlich. Der Wettbewerb weckt Interesse an Informatik, ohne dass die TeilnehmerInnen oder Lehrkräfte Vorkenntnisse haben müssen" (https://bwinf.de/biber/).

Die auf den Übungsblättern des Kurses „Verstehen wie Computer denken" eingesetzten Aufgaben basieren auf Aufgaben aus dem Informatik-Biber für die Klassenstufen 3 bis 6. Die einzelnen Aufgaben wurden entsprechend ihrer Übereinstimmung mit den in den einzelnen Kurssitzungen behandelten informatischen Konzepte ausgewählt.

4 Ziele, Inhalte und Struktur des Kurses

4.1 Kursziele

Der Kurs verfolgt drei Hauptziele:

- Die **Vermittlung** neun **grundlegender informatischer Konzepte**:
    1. Sequenz
    2. Schleife
    3. Muster
    4. Ereignis
    5. Bedingung
    6. Variable
    7. Konstante
    8. Operator
    9. Algorithmus
- Die Förderung der **Kompetenz zum Entwickeln algorithmischer Lösungsstrategien für informatische Problemstellungen**
- Das Wecken von **Interesse an informatischen Inhalten und Problemstellungen**

4.2 Beschreibung und Ablaufpläne der einzelnen Module und Kurssitzungen

Im Folgenden werden die einzelnen Module und die in ihnen enthaltenen Kurssitzungen im Detail beschrieben. Für jedes Modul und jede Kurssitzung sind die relevanten Lernziele und Lernmaterialien angegeben.

Zu Beginn des Kurses werden den Kindern mithilfe von lebensgroßen Brett- und Kartenspielen auf verkörperlichte und spielbasierte Weise grundlegende informatische Konzepte vermittelt, die sie im Laufe des Kurses in unterschiedlichen Kontexten anzuwenden lernen.

Dabei bearbeiten die Kinder zu Beginn des Kurses hinweg zunächst einfachere, konkretere Aufgaben, zu deren Lösung sie detaillierte Anleitungen befolgen können. Im Laufe des Kurses lernen sie daraufhin nach und nach, auch anspruchsvollere Aufgaben mit immer weniger Anleitung zu bearbeiten, bis sie zum Abschluss des Kurses schließlich größtenteils selbst Ansätze zur Lösung informatischer Problemstellungen entwickeln und diese eigenständig beim Programmieren umsetzen.

4.2.1 Modul 1 – Ein erstes Verständnis informatischer Denkkonzepte entwickeln

Kurssitzung	1) „Kennenlernen – uns und den Kurs“	2) „Muster erkennen und die Scratch-Umgebung erkunden“	3) „Wie entscheiden Computer? Bedingungen und Ereignisse“
Modus	unplugged	unplugged & plugged-in	unplugged
Kooperation	Zweier-Teams	Einzeln und Zweier-Teams	Zweier-Teams
Informatische Konzepte	• Sequenz • Schleife	• Muster • Ereignis	• Bedingung • Variable • Konstante • Operator
Kernmedium	Brettspiel „Schildkröten & Krabben: Die Schatzsuche“	Kartenspiel „Schildkröten & Krabben: Die Muster“ und Scratch	Brettspiel „Schildkröten & Krabben: Das Wettrennen“
MINT-Bezug	Informatik	Informatik	Mathematik
Das Übungsblatt enthält...	• Sequenzen • Schleifen	• Muster	• Bedingungen • Ereignisse

Lernziele und Inhalte von Modul 1

In diesem Modul bekommen die Schülerinnen und Schüler ein erstes Verständnis der grundlegenden informatischen Konzepte von Sequenzen, Schleifen, Mustern, Bedingungen, Ereignissen, Operatoren, Variablen und Konstanten vermittelt. All diese acht Konzepte werden mithilfe der lebensgroßen Brett- und Kartenspiele aus der „Schildkröten & Krabben“-Serie eingeführt.

Um die teilnehmenden Kinder auf die praktischen Programmierübungen in Modul 2 vorzubereiten und ihnen eine Perspektive auf die Anwendbarkeit grundlegender informatischer Konzepte zu eröffnen, werden sie in Modul 1 außerdem mit der visuellen Block-Programmiersprache und -umgebung Scratch vertraut gemacht.

Kurssitzung 1: „Kennenlernen – uns und den Kurs“
Kernaktivität, Lernziele und Materialien

Kernaktivität: Die Kinder spielen das lebensgroße Lern-Brettspiel „Schildkröten & Krabben: Die Schatzsuche“.

Lernziele: Dieses Spiel gibt den Kindern die Gelegenheit, sich ein erstes Verständnis der grundlegenden informatischen Konzepte von *Sequenzen* und *Schleifen* sowie von deren Beziehung zueinander erarbeiten. Darüber hinaus üben die Kinder sich darin, *Ansätze zum strategischen Problemlösen* zu entwickeln, indem sie während des Spiels eigene Lösungsansätze für ein Wegfindungsproblem ausprobieren.

Materialien:

Immer:
- Anwesenheitsliste
- Kurssitzungsplan
- Schreibutensilien

Sitzungsspezifisch:
- Illustrierte Definitionsposter zu den Begriffen *Sequenz* und *Schleife*
- Spielanleitung zu „Schildkröten & Krabben: Die Schatzsuche“
- Brettspiel „Schildkröten & Krabben: Die Schatzsuche“
- Übungsblatt 1
- Blatt „Zu Hause ausprobieren“ 1

Technik:
- Keine Technik erforderlich

Das Klassenzimmer sollte ...
- ... einen Stuhlkreis ermöglichen.
- … genügend freie Bodenfläche (mindestens 2 mal 2 Meter) für das lebensgroße Brettspiel haben.
- … genügend Stühle und Tische haben, dass die Kinder sich zum Bearbeiten der Übungsblätter hinsetzen können.

Vorbereitungen:
- Für alle teilnehmenden Kinder kopieren:
  - Übungsblatt 1
  - Blatt „Zu Hause ausprobieren“ 1
- Genügend freie Bodenfläche (mindestens 2 mal 2 Meter) für das Brettspiel schaffen
- Im Idealfall das Brettspiel bereits vor Beginn der Kurssitzung aufbauen und alle Spielmaterialien (Spielfiguren, Klemmbretter, Spielkarten, wegwischbare Stifte, Abzeichen usw.) bereit legen

Kurssitzung 1: „Kennenlernen – uns und den Kurs“					
Zeit	**Dauer**	**Ziel**	**Inhalt**	**Methode**	**Medien/Materialien**
0'	5'	Ankommen und Vorbereiten	Kinder begrüßen, Materialien vorbereiten	• Sämtliche **Materialien bereitlegen**. • Das **Brettspiel aufbauen**, siehe: Spielanleitung -> Spielaufbau. Für einen schnelleren Ablauf kann das Brettspiel auch schon **vor Kursbeginn** aufgebaut werden!	Alle Materialien
5'	5'	Gegenseitiges Kennenlernen	Vorstellungsrunde	Die Kursleitung setzt sich mit allen teilnehmenden Kindern in einen **Stuhlkreis**. • Die **Kinder stellen sich reihum der Gruppe vor**. Dafür sollen alle Folgendes über sich erzählen: o Ihren **Namen** o In der wievielten **Klasse** sie sind o Auf welche **Schule** sie gehen o Ein bis drei **Hobbies** • Auch die **Kursleitung** kann sich in dieser Runde den Kindern vorstellen und anstelle der eigenen Klassenstufe und Schule zum Beispiel den eigenen Beruf nennen.	
10'	10'	Einführung in den Kurs „Verstehen wie Computer denken“	Kurzes Gespräch: Was die Kinder vom Kurs erwarten und was sie im Kurs lernen können	**Was heißt „Verstehen wie Computer denken?“** • Die Kursleitung **leitet ein kurzes Gespräch** über die Erwartungen der Kinder. Dabei geht die Kursleitung auf die Antworten der Kinder ein und bietet ihnen zusätzliche Erklärungen (siehe Beispielerklärungen unter den Fragen). • **Fragen an die Kinder**: o **„Warum ist es beim Programmieren wichtig zu verstehen, wie Computer denken?“** ➢ Anders als wir Menschen denken Computer natürlich nicht so *richtig*. Trotzdem können sie auf Anweisungen reagieren, die Menschen ihnen beim Programmieren geben. Aber Computer sind sehr anders als wir Menschen und verstehen nicht alle Anweisungen, die auch ein Mensch verstehen würde. ➢ Wir wollen lernen, wie man Computern beim Programmieren Anweisungen gibt, die sie auch verstehen und ausführen können. Dazu müssen wir verstehen, wie Computer „denken“.	

Kurssitzung 1: „Kennenlernen – uns und den Kurs“					
Zeit	**Dauer**	**Ziel**	**Inhalt**	**Methode**	**Medien/Materialien**
				o **„Was glaubt ihr, was wir im Kurs machen werden?“** ➢ Hier im Kurs werden wir am Anfang Brettspiele und Kartenspiele spielen, bei denen ihr etwas darüber erfahrt, wie Computer „denken“. ➢ Danach werden wir richtig am Computer programmieren. Dafür wird es euch helfen, wenn ihr vorher die Brettspiele und Kartenspiele gespielt habt. o **„Was hofft ihr, was ihr hier lernen könnt?“** ➢ Natürlich könnt ihr hier im Kurs programmieren lernen! ➢ Ihr lernt aber auch, welche Anweisungen Computer verstehen und wie Computer Entscheidungen treffen können. ➢ Außerdem lernt ihr, dass man nicht nur Computerspiele programmieren kann, sondern dass man durch das Programmieren viele verschiedene Probleme lösen kann, die auf den ersten Blick vielleicht gar nichts mit Computern zu tun haben.	
20'	5'	Neue Begriffe lernen!	1. Sequenz 2. Schleife	**Jeder der neuen Begriffe wird wie folgt eingeführt:** • Die Kursleitung fragt die Kinder, ob sie das **Wort schonmal gehört** haben und was sie sich **darunter vorstellen**. • Die Kursleitung zeigt das **Definitionsposter** und bittet eines der Kinder, den Text auf dem Poster vorzulesen. **„Das ist, was das Wort bedeutet, wenn wir es im Kurs verwenden.“** • Um die Bedeutung der neuen Begriffe zu veranschaulichen, kann die Kursleitung **Beispiele aus dem Alltag** nennen: o **Sequenz**: ➢ **Stundenplan** für einen bestimmten Schultag („Montags habt ihr mehrere Schulfächer in einer bestimmten Reihenfolge.“) ➢ **Busfahrplan** („Der Bus fährt in einer bestimmten Reihenfolge von Haltestelle zu Haltestelle.“) o **Schleife**: ➢ **Telefonklingeln** („Wenn jemand bei euch anruft, wiederholt euer Telefon so lange die gleiche Melodie – Klingeling, Klingeling, Klingeling, Klingeling –, bis ihr ans Telefon geht.“)	Definitionsposter zu *Sequenz* und *Schleife*

Kurssitzung 1: „Kennenlernen – uns und den Kurs“					
Zeit	**Dauer**	**Ziel**	**Inhalt**	**Methode**	**Medien/Materialien**
				➢ **Bahnen schwimmen** („Wenn ihr von einer Beckenseite zur anderen schwimmen wollt, müsst ihr immer wieder die gleichen Schwimmbegungen wiederholen, bis ihr da seid.“)	
25'	15'	Spielregeln von „Schildkröten & Krabben: Die Schatzsuche“ erklären		Alle versammeln sich um das auf dem Boden liegende Spielbrett. • Die **Kursleitung erklärt die Spielregeln** von „Schildkröten & Krabben: Die Schatzsuche“ **anhand von Veranschaulichungen** mit den Spielmaterialien und den Figuren auf dem Spielbrett. • Dabei kann die **Spielanleitung als Orientierung** für die Reihenfolge der Erklärungen dienen. • Die Kursleitung sollte die Spielregeln jedoch **nicht aus der Spielanleitung vorlesen**, sondern frei **erklären und mithilfe der Spielfiguren auf dem Spielbrett veranschaulichen** (z.B. die 90°- und 270°-Drehungen mit einer Spielfigur zeigen.)	Spielanleitung, Brettspiel „Schildkröten & Krabben: Die Schatzsuche“
40'	35'	„Schildkröten & Krabben: Die Schatzsuche“ spielen		• Die **Kinder spielen** nach den soeben erlernten Spielregeln in Zweier-Teams das Spiel „**Schildkröten & Krabben: Die Schatzsuche**“. • Die **Kursleitung unterstützt** die Kinder bei Schwierigkeiten **mit erneuten Erklärungen und Veranschaulichungen**, vergibt **Abzeichen für richtig angewandte Konzepte** und achtet darauf, dass alle die Spielregeln befolgen. • **WICHTIG: Am Ende des Spiels sollten die Abzeichen zunächst auf den Spielfiguren stecken bleiben!**	Spielanleitung, Brettspiel „Schildkröten & Krabben: Die Schatzsuche“
75'	10'	Vertiefende Übungen/ Kontrolle des Lernfortschritts	Die Kinder bearbeiten Übungsblatt 1	Die Kursleitung teilt **Übungsblatt 1** an die Kinder aus. • Zu Beginn schreiben die Kinder den **Punktestand** ihres Teams **aus dem Spiel „Schildkröten & Krabben: Die Schatzsuche“** auf das Übungsblatt. Jedes Abzeichen ist einen Punkt wert. Abzeichen mit Zähler sind so viele Punkte wert, wie auf dem Zähler stehen. • Dann bearbeiten die Kinder das restliche Übungsblatt, das aus **Knobelaufgaben zu Sequenzen und Schleifen** besteht. • Die Kursleitung bittet die Kinder, sich zu melden, wenn sie etwas nicht verstehen und **unterstützt die Kinder bei Verständnisschwierigkeiten** mit Erklärungen. • Anschließend **sammelt die Kursleitung die Übungsblätter ein**.	Übungsblatt 1

Kurssitzung 1: „Kennenlernen – uns und den Kurs“					
Zeit	**Dauer**	**Ziel**	**Inhalt**	**Methode**	**Medien/Materialien**
85'	5'	Verabschiedung		Die Kursleitung teilt den Kindern ein Blatt mit Aufgaben aus, die sie **freiwillig zu Hause bearbeiten** können und **verabschiedet** sie bis zur nächsten Kurssitzung.	Blatt „Zu Hause ausprobieren“ 1
90'	**Ende von Kurssitzung 1**				

Probleme, die auftreten können:

- Die Erklärung der Spielregeln dauert länger als erwartet (15').
- Das Spiel dauert länger als erwartet und es gibt gegen Ende der angesetzten Spielzeit noch keinen Gewinner (35').
- Das Ziel des Spiels wird bereits deutlich vor Ende der angesetzten Spielzeit erreicht (35').

Mögliche Lösungen für diese Probleme:

- Noch vor dem Abschließen der Erklärung der Spielregeln kann eine Proberunde gespielt werden, in der die Teams die Regeln durch eigenes Ausprobieren verstehen lernen, indem Sie bereits das Spiel spielen.
- Die Kursleitung schlägt vor, das Spiel am Ende der aktuellen Runde zu beenden und das Team mit den bis dahin meisten gesammelten Abzeichen zum Gewinner des Spiels zu erklären.
- Die Gewinnbedingung wird so angepasst, dass die Teams ein Futter (egal welcher Farbe) zusätzlich sammeln müssen, um das Spiel zu gewinnen.

Kurssitzung 2: „Muster erkennen und die Scratch-Umgebung erkunden“

Kernaktivitäten, Lernziele und Materialien

Kernaktivitäten: Die Kinder spielen das Lern-Kartenspiel „Schildkröten & Krabben: Die Muster“ und erkunden Scratch mithilfe von Projektkarten.

Lernziele: Das Kartenspiel und die Arbeit mit dem Scratch-Projektkarten ermöglicht den Kindern, die *Erkennung bzw. den Vergleich von Mustern* zu üben, der in Vorbereitung auf die nächste Kurssitzung bereits das Überprüfen einfacher impliziter *Bedingungen* beinhaltet. Darüber hinaus lernen die Kinder sich auf Grundlage des Konzepts von *Ereignissen* eigenständig Elemente der *grundlegenden Funktionalität von Scratch* zu erarbeiten.

Materialien:

Immer:

- Anwesenheitsliste
- Kurssitzungsplan
- Schreibutensilien

Sitzungsspezifisch:

- Poster mit illustrierten Definitionen der Begriffe Sequenz, Schleife, Muster, Ereignis
- Blatt „Zu Hause ausprobieren“ 1
- Übungsblatt 1 aus der letzten Kurssitzung
- Kartenspiel und Spielanleitung zu „Schildkröten & Krabben: Die Muster“
- Scratch-Einführungsblatt
- Projektkarten
- Übungsblatt 2
- Blatt „Zu Hause ausprobieren“ 2

Technik:

- So viele Desktop-Computer oder Laptops, dass die Kinder je zu zweit an einem Computer arbeiten können
- Internetzugang ODER installierte Version von Scratch 2.0

Das Klassenzimmer sollte ...

- ... einen Stuhlkreis ermöglichen.
- … genügend freie Bodenfläche (mindestens 1 mal 1 Meter) für das Kartenspiel haben.
- … genügend Stühle und Tische haben, dass die Kinder sich zum Bearbeiten der Übungsblätter hinsetzen können.

Vorbereitungen:

- Optional: Scratch 2.0 auf allen Computern installieren, an denen die Kinder arbeiten werden
- Computer hochfahren
- Scratch-Projektkarten einmal kopieren und in der Mitte durchschneiden
- Für alle teilnehmenden Kinder kopieren:
  - Scratch-Einführungsblatt
  - Scratch-Projektkarten (müssen nicht durchgeschnitten werden)
  - Übungsblatt 2
  - Blatt „Zu Hause ausprobieren“ 2
- Genügend freie Bodenfläche (mindestens 1 mal 1 Meter) für das Kartenspiel schaffen

Kurssitzung 2: „Muster erkennen und die Scratch-Umgebung erkunden“					
Zeit	**Dauer**	**Ziel**	**Inhalt**	**Methode**	**Medien/Materialien**
0'	5'	Ankommen und Vorbereiten	Kinder begrüßen, Materialien vorbereiten	• Sämtliche **Materialien bereitlegen**. • Falls die für den Kurs verwendeten **Computer** sehr lange brauchen, um hochzufahren, kann die Kursleitung sie bereits jetzt **einschalten**.	Alle Materialien
5'	5'	Rückbezug auf die vergangene Kurssitzung und Besprechung der Hausaufgaben		Die Kursleitung setzt sich mit allen teilnehmenden Kindern in einen **Stuhlkreis**. • Die **Kursleitung fragt die Kinder**: o Wisst ihr noch, was wir in der **letzten Kurssitzung** gemacht haben? o Wer von euch hat zu Hause die **Hausaufgaben ausprobiert**? o Was sind eure **Lösungen**?	Blatt „Zu Hause ausprobieren“ 1
10'	5'	Wiederholung bisher gelernter Konzepte	Begriffe 1. Sequenz 2. Schleife	• Die **Kursleitung fragt die Kinder** (für jeden der Begriffe einzeln): o **Wisst ihr noch, was das Wort bedeutet**, wenn wir es hier im Kurs verwenden? (Kurzer Rückblick auf das jeweilige Definitionsposter.)	Definitionsposter zu *Sequenz* und *Schleife*
15'	5'	Besprechung von Übungsblatt 1		• Die Kursleitung teilt den Kindern ihre **Übungsblätter aus der vergangenen Kursstunde** aus und **bespricht die Ergebnisse** mit den Kindern. • Für jede der Aufgaben bittet die Kursleitung ein **Kind, das die richtige Lösung zu haben glaubt**, den anderen den **eigenen Lösungsweg** zu erklären. Dann sollen die anderen Kinder sagen, ob sie dieser **Lösung zustimmen** oder **begründen, falls sie nicht zustimmen**. Die **Kursleitung sammelt die alten Übungsblätter** danach **wieder ein**.	Übungsblatt 1
20'	5'	Neue Begriffe lernen!	1. Muster 2. Ereignis	**Jeder der neuen Begriffe wird wie folgt eingeführt:** • Die Kursleitung fragt die Kinder, ob sie das **Wort schonmal gehört** haben und was sie sich **darunter vorstellen**. • Die Kursleitung zeigt das **Definitionsposter** und bittet eines der Kinder, den Text auf dem Poster vorzulesen. **„Das ist, was das Wort bedeutet, wenn wir es im Kurs verwenden.“** • Um die Bedeutung der neuen Begriffe zu veranschaulichen, kann die Kursleitung **Beispiele aus dem Alltag** nennen: o **Muster**: ➢ **Schachbrett** („Ein Schachbrett besteht aus einem bestimmten Muster von schwarzen und weißen Feldern. Die unterschiedlichen Spielfiguren können sich auf verschiedene Arten über dieses Muster bewegen.“) ➢ **Dominosteine** („Jeder Dominostein hat ein bestimmtes	Definitionsposter zu *Muster* und *Ereignis*

Kurssitzung 2: „Muster erkennen und die Scratch-Umgebung erkunden“					
Zeit	**Dauer**	**Ziel**	**Inhalt**	**Methode**	**Medien/Materialien**
				Muster aus Punkten. Wenn man Domino spielen will, muss man die Muster auf den Steinen vergleichen, um zu wissen, welche Steine man aneinander legen darf.“) o **Ereignis**: ➢ **Ampelsignal** („Wenn die Ampel rot wird, müssen alle Autos anhalten und warten. Wenn die Ampel grün wird, dürfen die Autos wieder losfahren.“) ➢ **Anpfiff bei einem Fußballspiel** („Wenn der Schiedsrichter anpfeift, beginnt das Fußballspiel.“)	
25'	5'	Spielregeln von „Schildkröten & Krabben: Die Muster“ erklären		Die Kursleitung legt einen Teil der Spielkarten wie in der Spielanleitung beschrieben auf dem Boden aus. Die Kinder versammeln sich vor den auf dem Boden liegenden Spielkarten. • Die **Kursleitung erklärt die Spielregeln** von „Schildkröten & Krabben: Die Muster“ **anhand von Veranschaulichungen** mit den Spielkarten. • Dabei kann die **Spielanleitung als Orientierung** für die Reihenfolge der Erklärungen dienen. • Die Kursleitung sollte die Spielregeln jedoch **nicht aus der Spielanleitung vorlesen**, sondern frei **erklären und mit Spielkarten veranschaulichen**.	Spielanleitung, Brettspiel „Schildkröten & Krabben: Die Muster“
30'	15'	„Schildkröten & Krabben: Die Muster“ spielen		• Die **Kinder spielen** nach den soeben erlernten Spielregeln das Spiel „**Schildkröten & Krabben: Die Muster**“. • Die **Kursleitung unterstützt** die Kinder bei Schwierigkeiten **mit erneuten Erklärungen und Veranschaulichungen**, vergibt **Abzeichen für richtig angewandte Konzepte** und achtet darauf, dass alle die Spielregeln befolgen. • **WICHTIG: Am Ende des Spiels behalten die Kinder ihre gesammelten Karten bis zum Ende der Kurssitzung!**	Spielanleitung, Brettspiel „Schildkröten & Krabben: Die Muster“
45'	10'	Erarbeiten der Grundlagen zur Benutzung der Scratch-Umgebung	Scratch-Umgebung: Bühne, Objekte, Skripte, Figuren, Kostüme, Arten von Befehlen	• Die Kursleitung teilt den Kindern das **Scratch-Einführungsblatt** aus. • Die **Kinder** suchen sich eine Partnerin oder einen Partner und **setzen sich je zu zweit an einen Computer und öffnen die Scratch-Umgebung** nach der Anleitung auf dem Scratch-Einführungsblatt. • Die Zweier-Teams bearbeiten das **Scratch-Einführungsblatt**.	Scratch-Einführungsblatt, Computer, Scratch-Umgebung (online oder installiert)
55'	20'	Einfache Scratch-Projekte ausprobieren		• Die Kursleitung wählt aus den **Scratch-Projektkarten** halb so viele Projekte aus, wie Kinder im Kurs sind. **Jedes Projekt besteht aus zwei Projektkarten, die zusammen gehören** und bei der Lösung einer kleinen	Scratch-Projektkarten, Computer, Scratch-

Kurssitzung 2: „Muster erkennen und die Scratch-Umgebung erkunden“					
Zeit	**Dauer**	**Ziel**	**Inhalt**	**Methode**	**Medien/Materialien**
				Programmieraufgabe helfen können. • Die Kursleitung **mischt die einzelnen Projektkarten** und lässt jedes Kind eine **Karte ziehen**. • **Jedes Kind** sieht sich seine Karte an und **sucht** nun **das Kind, das die andere Karte hat,** die zum selben Projekt gehört, und bildet ein **Zweier-Team** mit diesem Kind. • Jedes Zweier-Team geht an einen **Computer** und versucht die **Programmieraufgabe** von den gezogenen Projektkarten zu **lösen**. • Wenn ein Team **in weniger als 20 Minuten fertig** wird, kann die Kursleitung ihm ein **neues Projekt** geben (zwei zusammen gehörende Projektkarten). Während dieser gesamten Phase steht die **Kursleitung** allen Teams zur **Beantwortung von Fragen** und für **Hilfestellungen** zur Verfügung.	Umgebung (online oder installiert)
75'	10'	Vertiefende Übungen/ Kontrolle des Lernfortschritts	Die Kinder bearbeiten Übungsblatt 2	Die Kursleitung teilt **Übungsblatt 2** an die Kinder aus. • Zu Beginn schreiben die Kinder ihren **Punktestand aus dem Spiel „Die Muster“** auf das Übungsblatt. Jede Karte ist einen Punkt wert. • Dann bearbeiten die Kinder das restliche Übungsblatt, das aus **Knobelaufgaben zu Mustererkennung und logischem Denken** besteht. • Die Kursleitung bittet die Kinder, sich zu melden, wenn sie etwas nicht verstehen und **unterstützt die Kinder bei Verständnisschwierigkeiten** mit Erklärungen. Anschließend **sammelt die Kursleitung die Übungsblätter ein**.	Übungsblatt 2
85'	5'	Verabschiedung		Die Kursleitungen teilt jedem Kind eine Kopie aller **Scratch-Projektkarten** aus, mit denen die Kinder zu Hause arbeiten können. Außerdem bekommen die Kinder ein Blatt mit Aufgaben, die sie **freiwillig zu Hause bearbeiten** können und die Kursleitung **verabschiedet** sie bis zur nächsten Kurssitzung. Die Kinder dürfen außerdem das Arbeitsblatt „Scratch-Einführung“ mit nach Hause nehmen.	Scratch-Projektkarten, Blatt „Zu Hause ausprobieren“ 2
90'	**Ende von Kurssitzung 2**				

Probleme, die auftreten können:

- Am Ende der Kurssitzung bleibt noch Zeit nach dem Ausfüllen des Übungsblattes.

Mögliche Lösungen für diese Probleme:

- Die Kursleitung lässt die Kinder entscheiden, ob sie die Scratch-Umgebung weiter erkunden und Dinge selbst ausprobieren möchten oder ob sie noch einmal „Schildkröten & Krabben: Die Muster“ spielen wollen.

Kurssitzung 3: „Wie entscheiden Computer? Bedingungen und Verzweigungen“

Kernaktivität, Lernziele und Materialien

Kernaktivität: Die Kinder spielen das lebensgroße Lern-Brettspiel „Schildkröten & Krabben: Das Wettrennen“.

Lernziele: Dieses Spiel bietet den Kindern die Möglichkeit, sich ein erstes Verständnis der grundlegenden informatischen Konzepte von *Bedingungen* und mathematiknahen Konzepte von *Variablen, Konstanten* und *Operatoren* zu erarbeiten. Währenddessen vertiefen sie ihr Verständnis von *Ereignissen* und knüpfen für die Erarbeitung des Konzepts von Vergleichen beinhaltenden *Bedingungen* an das Üben des Vergleichens von *Mustern* aus der vorherigen Kurssitzung an.

Materialien:

Immer:
- Anwesenheitsliste
- Kurssitzungsplan
- Schreibutensilien

Sitzungsspezifisch:
- Poster mit illustrierten Definitionen der Begriffe Muster, Ereignis, Bedingung, Variable, Konstante, Operator
- Blatt „Zu Hause ausprobieren“ 2
- Übungsblatt 2 aus der letzten Kurssitzung
- Spielanleitung zu „Schildkröten & Krabben: Das Wettrennen“
- Brettspiel „Schildkröten & Krabben: Das Wettrennen“
- Übungsblatt 3
- Blatt „Zu Hause ausprobieren“ 3

Technik:
- Keine Technik erforderlich

Das Klassenzimmer sollte ...
- ... einen Stuhlkreis ermöglichen.
- … genügend freie Bodenfläche (mindestens 2 mal 2 Meter) für das lebensgroße Brettspiel haben.
- … genügend Stühle und Tische haben, dass die Kinder sich zum Bearbeiten der Übungsblätter hinsetzen können.

Vorbereitungen:
- Für alle teilnehmenden Kinder kopieren:
  - Übungsblatt 3
  - Blatt „Zu Hause ausprobieren“ 3
- Genügend freie Bodenfläche (mindestens 2 mal 2 Meter) für das Brettspiel schaffen
- Im Idealfall das Brettspiel bereits vor Beginn der Kurssitzung aufbauen und alle Spielmaterialien (Spielfiguren, Spielkarten, wegwischbare Stifte, Abzeichen usw.) bereit legen

Kurssitzung 3: „Wie entscheiden Computer? Bedingungen und Verzweigungen“					
Zeit	**Dauer**	**Ziel**	**Inhalt**	**Methode**	**Medien/Materialien**
0'	5'	Ankommen und Vorbereiten	Kinder begrüßen, Materialien vorbereiten	• Sämtliche **Materialien bereitlegen**. • Das **Brettspiel aufbauen**, siehe: Spielanleitung -> Spielaufbau. Für einen schnelleren Ablauf kann das Brettspiel auch schon **vor Kursbeginn** aufgebaut werden!	Alle Materialien
5'	5'	Rückbezug auf die vergangene Kurssitzung und Besprechung der Hausaufgaben		Die Kursleitung setzt sich mit allen teilnehmenden Kindern in einen **Stuhlkreis**. • Die **Kursleitung fragt die Kinder**: o Wisst ihr noch, was wir in der **letzten Kurssitzung** gemacht haben? o Wer von euch hat zu Hause die **Hausaufgaben ausprobiert**? o Welche **Erfahrungen** habt ihr damit gemacht? o Was sind eure **Lösungen**?	Blatt „Zu Hause ausprobieren“ 2
10'	5'	Wiederholung bisher gelernter Konzepte	Begriffe 1. Muster 2. Ereignis	Die Kursleitung setzt sich mit allen teilnehmenden Kindern in einen **Stuhlkreis**. • Die **Kursleitung fragt die Kinder** (für jeden der Begriffe einzeln): • **Wisst ihr noch, was das Wort bedeutet**, wenn wir es hier im Kurs verwenden? (Kurzer Rückblick auf das jeweilige Definitionsposter.)	Definitionsposter zu *Muster* und *Ereignis*
15'	5'	Besprechung von Übungsblatt 2		• Die Kursleitung teilt den Kindern ihre **Übungsblätter aus der vergangenen Kursstunde** aus und **bespricht die Ergebnisse** mit den Kindern. • Für jede der Aufgaben bittet die Kursleitung ein **Kind, das die richtige Lösung zu haben glaubt**, den anderen den **eigenen Lösungsweg** zu erklären. Dann sollen die anderen Kinder sagen, ob sie dieser **Lösung zustimmen** oder **begründen, falls sie nicht zustimmen**. Die **Kursleitung sammelt die alten Übungsblätter** danach **wieder ein**.	Übungsblatt 2
20'	10'	Neue Begriffe lernen!	1. Bedingung 2. Variable 3. Konstante 4. Operator	**Jeder der neuen Begriffe wird wie folgt eingeführt:** • Die Kursleitung fragt die Kinder, ob sie das **Wort schonmal gehört** haben und was sie sich **darunter vorstellen**. • Die Kursleitung zeigt das **Definitionsposter** und bittet eines der Kinder, den Text auf dem Poster vorzulesen. **„Das ist, was das Wort bedeutet, wenn wir es im Kurs verwenden.“** • Um die Bedeutung der neuen Begriffe zu veranschaulichen, kann die Kursleitung **Beispiele aus dem Alltag** nennen:	Definitionsposter zu *Bedingung, Variable, Konstante* und *Operator*

Kurssitzung 3: „Wie entscheiden Computer? Bedingungen und Verzweigungen“					
Zeit	**Dauer**	**Ziel**	**Inhalt**	**Methode**	**Medien/Materialien**
				o **Bedingung**: ➢ **Deine Eltern** stellen bestimmt manchmal Bedingungen („Falls du direkt nach dem Mittagessen deine Hausaufgaben fertig machst, darfst du danach eine Stunde fernsehen.“) ➢ **Viele Regeln** sind als Bedingungen formuliert („Falls ein Kind mindestens 1,30 m groß ist, darf es mit dieser großen Achterbahn fahren. Sonst muss es mit der kleineren Achterbahn fahren.“) • Für **Variablen, Konstanten und Operatoren** müssen keine Beispiele aus dem Alltag gegeben werden, da es sich bei diesen Begriffen um etwas **abstraktere mathematische Konzepte** handelt. Damit die Kinder diese Konzepte trotzdem verstehen lernen, werden sie gleich **im Brettspiel „Schildkröten & Krabben: Das Wettrennen“ veranschaulicht**. • Erklärung zu **Operatoren**: „Manche Zeichen sehen **beim Programmieren anders** aus, **als ihr sie aus dem Mathe-Unterricht kennt**“ (siehe Multiplikation und Division auf dem Definitionsposter).	
30'	10'	Spielregeln von „Schildkröten & Krabben: Das Wettrennen“ erklären		Alle versammeln sich um das auf dem Boden liegende Spielbrett. • Die **Kursleitung erklärt die Spielregeln** von „Schildkröten & Krabben: Das Wettrennen“ **anhand von Veranschaulichungen** mit den Spielmaterialien und Figuren auf dem Spielbrett. • Dabei kann die **Spielanleitung als Orientierung** für die Reihenfolge der Erklärungen dienen. • Die Kursleitung sollte die Spielregeln jedoch **nicht aus der Spielanleitung vorlesen**, sondern frei **erklären und mithilfe der Spielmaterialien veranschaulichen**.	Spielanleitung, Brettspiel „Schildkröten & Krabben: Das Wettrennen“
40'	35'	„Schildkröten & Krabben: Das Wettrennen“ spielen		• Die **Kinder spielen** nach den soeben erlernten Spielregeln in Zweier-Teams das Spiel „**Schildkröten & Krabben: Das Wettrennen**“. • Die **Kursleitung unterstützt** die Kinder bei Schwierigkeiten **mit erneuten Erklärungen und Veranschaulichungen**, vergibt **Abzeichen für richtig angewandte Konzepte** und achtet darauf, dass alle die Spielregeln befolgen. • **WICHTIG: Am Ende des Spiels sollten die Abzeichen zunächst auf den Spielfiguren stecken bleiben!**	Spielanleitung, Brettspiel „Schildkröten & Krabben: Das Wettrennen“
75'	10'	Vertiefende Übungen/	Die Kinder bearbeiten Übungsblatt 3	Die Kursleitung teilt **Übungsblatt 3** an die Kinder aus.	Übungsblatt 3

Kurssitzung 3: „Wie entscheiden Computer? Bedingungen und Verzweigungen“					
Zeit	**Dauer**	**Ziel**	**Inhalt**	**Methode**	**Medien/Materialien**
		Kontrolle des Lernfortschritts		• Zu Beginn schreiben die Kinder den **Punktestand** ihres Teams **aus dem Spiel „Schildkröten & Krabben: Das Wettrennen“** auf das Übungsblatt. Jedes Abzeichen ist einen Punkt wert. Abzeichen mit Zähler sind so viele Punkte wert, wie auf dem Zähler stehen. • Dann bearbeiten die Kinder das restliche Übungsblatt, das aus **Knobelaufgaben zu Bedingungen, mathematischem Verständnis und Ereignissen** besteht. • Die Kursleitung bittet die Kinder, sich zu melden, wenn sie etwas nicht verstehen und **unterstützt die Kinder bei Verständnisschwierigkeiten** mit Erklärungen. Anschließend **sammelt die Kursleitung die Übungsblätter ein**.	
85'	5'	Verabschiedung		Die Kursleitung teilt den Kindern ein Blatt mit Aufgaben aus, die sie **freiwillig zu Hause bearbeiten** können und **verabschiedet** sie bis zur nächsten Kurssitzung.	Blatt „Zu Hause ausprobieren“ 3
90'	**Ende von Kurssitzung 3**				

Probleme, die auftreten können:

- Das Spiel dauert länger als erwartet und es gibt gegen Ende der angesetzten Spielzeit noch keinen Gewinner (35').
- Das Ziel des Spiels wird bereits deutlich vor Ende der angesetzten Spielzeit erreicht (35').

Mögliche Lösungen für diese Probleme:

- Die Kursleitung reduziert das Zielfeld des Wettrennens entsprechend der verbleibenden Unterrichtszeit, z.B. von 30 auf 25.
- Die Kursleitung passt den Endpunkt des Spiels an, indem sie das Zielfeld entsprechend der verbleibenden Unterrichtszeit erhöht.

4.2.2 Modul 2 – Anwendung und Vertiefung informatischer Denkkonzepte in Scratch

Kurssitzung	4) „Was sind Algorithmen?“	5) „Was sind Simulationen?“	6) „Mathematische Anwendungen simulieren“
Modus	plugged-in	plugged-in	plugged-in
Kooperation	Zweier-Teams	Zweier-Teams	Zweier-Teams
Informatische Konzepte	• Alle zuvor eingeführten Konzepte	• Alle zuvor eingeführten Konzepte	• Alle zuvor eingeführten Konzepte
Kernmedium	Scratch	Scratch	Scratch
MINT-Bezug	Informatik	Naturwissenschaft (Biologie, Erdkunde)	Mathematik
Das Übungsblatt enthält...	• Bedingungen • Sequenzen	• Sequenzen • Schleifen	• Bedingungen • Ereignisse • Variablen • Sequenzen • Schleifen

Lernziele und Inhalte von Modul 2

In diesem Modul lernen die Schülerinnen und Schüler, wie sie die in Modul 1 erarbeiteten informatischen Konzepte von Sequenzen, Schleifen, Mustern, Bedingungen, Ereignissen, Operatoren, Variablen und Konstanten beim Programmieren in Scratch anwenden können. Darüber hinaus lernen sie das die bisher erlernten Konzepte verknüpfende Konzept des Algorithmus kennen. Außerdem lernen sie, wie sie Algorithmen zur Simulation unterschiedlicher Vorgänge entwickeln können.

Kurssitzung 4: „Was sind Algorithmen?“
Kernaktivität, Lernziele und Materialien

Kernaktivität: Die Kinder erproben angeleitet durch das Arbeitsblatt „Spiele programmieren“ die Programmierung drei einfacher Computerspiele.

Lernziele: Die Kinder machen sich näher mit der Programmiersprache und -umgebung von Scratch vertraut. Währenddessen lernen sie, zur Lösung einfacher Problemstellungen *Algorithmen* zu entwickeln, indem sie die Konzepte von *Sequenzen, Schleifen, Bedingungen* und *Ereignissen* miteinander verknüpfen und zur Problemlösung einsetzen. So können sie beispielweise entdecken, wie sie grundlegende Spielmechaniken vieler Computerspiele (z.B. Bewegung einer Figur durch Pfeiltasten) programmieren können.

Materialien:

Immer:
- Anwesenheitsliste
- Kurssitzungsplan
- Schreibutensilien

Das Klassenzimmer sollte ...
- ... einen Stuhlkreis ermöglichen.
- … genügend Stühle und Tische haben, dass die Kinder sich zum Bearbeiten der Übungsblätter hinsetzen können.

Sitzungsspezifisch:
- Poster mit illustrierten Definitionen der Begriffe Sequenz, Schleife, Bedingung, Algorithmus
- Blatt „Zu Hause ausprobieren“ 3
- Übungsblatt 3 aus der letzten Kurssitzung
- Arbeitsblatt „Spiele programmieren“
- Übungsblatt 4
- Blatt „Zu Hause ausprobieren“ 4

Vorbereitungen:
- Computer hochfahren
- Für alle teilnehmenden Kinder kopieren:
  - Arbeitsblatt „Spiele programmieren“
  - Übungsblatt 4
  - Blatt „Zu Hause ausprobieren“ 4

Technik:
- So viele Desktop-Computer oder Laptops, dass die Kinder je zu zweit an einem Computer arbeiten können
- Internetzugang ODER installierte Version von Scratch 2.0

Kurssitzung 4: „Was sind Algorithmen?“					
Zeit	**Dauer**	**Ziel**	**Inhalt**	**Methode**	**Medien/Materialien**
0'	5'	Ankommen und Vorbereiten	Kinder begrüßen, Materialien vorbereiten	• Sämtliche **Materialien bereitlegen**. • Falls die für den Kurs verwendeten **Computer** sehr lange brauchen, um hochzufahren, kann die Kursleitung sie bereits jetzt **einschalten**.	Alle Materialien
5'	5'	Rückbezug auf die vergangene Kurssitzung und Besprechung der Hausaufgaben		Die Kursleitung setzt sich mit allen teilnehmenden Kindern in einen **Stuhlkreis**. • Die **Kursleitung fragt die Kinder**: o Wisst ihr noch, was wir in der **letzten Kurssitzung** gemacht haben? o Wer von euch hat zu Hause die **Hausaufgaben ausprobiert**? o Was sind eure **Lösungen**?	Blatt „Zu Hause ausprobieren“ 3
10'	5'	Wiederholung bisher gelernter Konzepte	1. Sequenz 2. Schleife 3. Bedingung 4. Ereignis	Die Kursleitung setzt sich mit allen teilnehmenden Kindern in einen **Stuhlkreis**. • Die **Kursleitung fragt die Kinder** (für jeden der Begriffe einzeln): • **Wisst ihr noch, was das Wort bedeutet**, wenn wir es hier im Kurs verwenden? (Kurzer Rückblick auf das jeweilige Definitionsposter.)	Definitionsposter zu *Sequenz, Schleife Bedingung* und *Ereignis*
15'	5'	Besprechung von Übungsblatt 3		• Die Kursleitung teilt den Kindern ihre **Übungsblätter aus der vergangenen Kursstunde** aus und **bespricht die Ergebnisse** mit den Kindern. • Für jede der Aufgaben bittet die Kursleitung ein **Kind, das die richtige Lösung zu haben glaubt**, den anderen den **eigenen Lösungsweg** zu erklären. Dann sollen die anderen Kinder sagen, ob sie dieser **Lösung zustimmen** oder **begründen, falls sie nicht zustimmen**. Die **Kursleitung sammelt die alten Übungsblätter** danach **wieder ein**.	Übungsblatt 3
20'	5'	Neue Begriffe lernen!	1. Algorithmus	**Der neue Begriff wird wie folgt eingeführt:** • Die Kursleitung fragt die Kinder, ob sie das **Wort schonmal gehört** haben und was sie sich **darunter vorstellen**. • Die Kursleitung zeigt das **Definitionsposter** und bittet eines der Kinder, den Text auf dem Poster vorzulesen. **„Das ist, was das Wort bedeutet, wenn wir es im Kurs verwenden.“** • Um die Bedeutung der neuen Begriffe zu veranschaulichen, kann die Kursleitung **Beispiele aus dem Alltag** nennen: o **Algorithmus**: ➢ **Bastelanleitung** („Wenn du etwas basteln möchtest, musst du oft eine ganz genaue Anleitung befolgen, die Sequenzen, Schleifen und Bedingungen enthalten kann.“)	Definitionsposter zu *Algorithmus*

Kurssitzung 4: „Was sind Algorithmen?“					
Zeit	**Dauer**	**Ziel**	**Inhalt**	**Methode**	**Medien/Materialien**
25'	50'	Erste Spiele programmieren in Scratch		Noch im Stuhlkreis teilt die Kursleitung das **Arbeitsblatt „Spiele programmieren“** aus. • Zu Beginn des Blattes findet sich eine **kurze Erklärung und Übung** zum **Achsensystem des Scratch-Vorschau-Fensters**. Diese Übung kann die Kursleitung mit den Kindern im Stuhlkreis besprechen. Anschließend teilt die Kursleitung die Kinder in **Zweier-Teams** ein. Jeweils ein Team kann jetzt an einen **Computer** gehen. • Jedes Team **programmiert** nun zuerst **Spiel 1 (Das Katzenspiel)** mithilfe der Anleitungen auf dem Arbeitsblatt. Sobald ein Team mit dieser Aufgabe fertig ist, soll es der Kursleitung das fertige Spiel zeigen. • Anschließend **programmiert** das Team mithilfe des **Arbeitsblattes Spiel 2 (Das Frosch- & Insektenspiel)**. Das fertige Spiel soll wieder der Kursleitung gezeigt werden. • Falls noch Zeit sein sollte, dass das Team nun mithilfe des Arbeitsblattes **Spiel 3 (Das Krabbenspiel) programmieren**. Während die Kinder programmieren, **unterstützt** die Kursleitung sie **bei Fragen und Schwierigkeiten**.	Arbeitsblatt „Spiele programmieren“, Computer
75'	10'	Vertiefende Übungen/ Kontrolle des Lernfortschritts	Die Kinder bearbeiten Übungsblatt 4	Die Kursleitung teilt **Übungsblatt 4** an die Kinder aus. Dann bearbeiten die Kinder das Übungsblatt, das aus **Knobelaufgaben zu Bedingungen, Sequenzen, räumlicher Orientierung und Raster-/Achsenverständnis** besteht. • Die Kursleitung bittet die Kinder, sich zu melden, wenn sie etwas nicht verstehen und **unterstützt die Kinder bei Verständnisschwierigkeiten** mit Erklärungen. Anschließend **sammelt die Kursleitung die Übungsblätter ein**.	Übungsblatt 4
85'	5'	Verabschiedung		Die Kursleitung teilt den Kindern ein Blatt mit Aufgaben aus, die sie **freiwillig zu Hause bearbeiten** können und **verabschiedet** sie bis zur nächsten Kurssitzung. Die Kinder dürfen außerdem das Arbeitsblatt „Spiele programmieren“ mit nach Hause nehmen.	Blatt „Zu Hause ausprobieren“ 4
90'	**Ende von Kurssitzung 4**				

Probleme, die auftreten können:	Mögliche Lösungen für diese Probleme:
• Es bleibt nicht genug Zeit, um die Programmierung von zwei oder allen drei Spielen in der angesetzten Zeit (50') auszuprobieren.	• Die Kursleitung kann die Kinder auf das Blatt „Zu Hause ausprobieren“ 4 verweisen, das die übrigen Programmieraufgaben als freiwillige Hausaufgaben enthält.
• Die Kinder werden vor Ablauf der angesetzten Zeit (50') mit der Programmierung der Spiele in Scratch fertig.	• Die Kursleitung fordert die Kinder dazu auf, mit ihrem Code zu experimentieren und gibt ihnen Ideen, wie sie den Spielen zusätzliche Funktionalitäten wie Zeitbeschränkungen usw. hinzufügen können.

Kurssitzung 5: „Was sind Simulationen?“
Kernaktivität, Lernziele und Materialien

Kernaktivität: Die Kinder programmieren auf Grundlage des Arbeitsblatts „Simulationen“ zwei Simulationen natürlicher Prozesse.

Lernziele: Die Kinder erfahren, was *Simulationen* sind und zu welchen Zwecken diese eingesetzt werden können. Dazu lernen sie, zur Nachbildung natürlicher Prozesse (z.B. einer Nahrungskette oder eines Wasserkreislaufs) einfache Simulationen zu programmieren, indem sie auf den Konzepten von *Sequenzen, Schleifen, Bedingungen* und *Algorithmen* aufbauen und ihr Verständnis dieser Konzepte vertiefen.

Materialien:

Immer:
- Anwesenheitsliste
- Kurssitzungsplan
- Schreibutensilien

Sitzungsspezifisch:
- Poster mit illustrierten Definitionen der Begriffe Sequenz, Schleife, Bedingung, Algorithmus, Simulation
- Blatt „Zu Hause ausprobieren“ 4
- Übungsblatt 4 aus der letzten Kurssitzung
- Arbeitsblatt „Simulationen“
- Übungsblatt 5
- Blatt „Zu Hause ausprobieren“ 5

Technik:
- So viele Desktop-Computer oder Laptops, dass die Kinder je zu zweit an einem Computer arbeiten können
- Internetzugang ODER installierte Version von Scratch 2.0

Das Klassenzimmer sollte ...
- ... einen Stuhlkreis ermöglichen.
- … genügend Stühle und Tische haben, dass die Kinder sich zum Bearbeiten der Übungsblätter hinsetzen können.

Vorbereitungen:
- Computer hochfahren
- Für alle teilnehmenden Kinder kopieren:
  - Arbeitsblatt „Simulationen“
  - Übungsblatt 5
  - Blatt „Zu Hause ausprobieren“ 5

Kurssitzung 5: „Was sind Simulationen?“					
Zeit	**Dauer**	**Ziel**	**Inhalt**	**Methode**	**Medien/Materialien**
0'	5'	Ankommen und Vorbereiten	Kinder begrüßen, Materialien vorbereiten	• Sämtliche **Materialien bereitlegen**. • Falls die für den Kurs verwendeten **Computer** sehr lange brauchen, um hochzufahren, kann die Kursleitung sie bereits jetzt **einschalten**.	Alle Materialien
5'	5'	Rückbezug auf die vergangene Kurssitzung und Besprechung der Hausaufgaben		Die Kursleitung setzt sich mit allen teilnehmenden Kindern in einen **Stuhlkreis**. • Die **Kursleitung fragt die Kinder**: o Wisst ihr noch, was wir in der **letzten Kurssitzung** gemacht haben? o Wer von euch hat zu Hause die **Hausaufgaben ausprobiert**? o Welche **Erfahrungen** habt ihr damit gemacht?	Blatt „Zu Hause ausprobieren“ 4
10'	5'	Wiederholung bisher gelernter Konzepte	1. Sequenz 2. Schleife 3. Bedingung 4. Algorithmus	Die Kursleitung setzt sich mit allen teilnehmenden Kindern in einen **Stuhlkreis**. • Die **Kursleitung fragt die Kinder** (für jeden der Begriffe einzeln): • **Wisst ihr noch, was das Wort bedeutet**, wenn wir es hier im Kurs verwenden? (Kurzer Rückblick auf das jeweilige Definitionsposter.)	Definitionsposter zu *Sequenz, Schleife, Bedingung* und *Algorithmus*
15'	5'	Besprechung von Übungsblatt 4		• Die Kursleitung teilt den Kindern ihre **Übungsblätter aus der vergangenen Kursstunde** aus und **bespricht die Ergebnisse** mit den Kindern. • Für jede der Aufgaben bittet die Kursleitung ein **Kind, das die richtige Lösung zu haben glaubt**, den anderen den **eigenen Lösungsweg** zu erklären. Dann sollen die anderen Kinder sagen, ob sie dieser **Lösung zustimmen** oder **begründen, falls sie nicht zustimmen**. Die **Kursleitung sammelt die alten Übungsblätter** danach **wieder ein**.	Übungsblatt 4
20'	5'	Neue Begriffe lernen!	1. Simulation	**Der neue Begriff wird wie folgt eingeführt:** • Die Kursleitung fragt die Kinder, ob sie das **Wort schonmal gehört** haben und was sie sich **darunter vorstellen**. • Die Kursleitung zeigt das **Definitionsposter** und bittet eines der Kinder, den Text auf dem Poster vorzulesen. **„Das ist, was das Wort bedeutet, wenn wir es im Kurs verwenden.“** • Um die Bedeutung der neuen Begriffe zu veranschaulichen, kann die Kursleitung **Beispiele aus dem Alltag** nennen: o **Simulation**: ➢ **Wetterbericht** („Der Wetterbericht im Fernsehen zeigt, wie das Wetter in der Zukunft sein soll. Aber die Wetterreporter können nicht in die Zukunft sehen. Deswegen wird der Wetterbericht mit einer Simulation berechnet.“)	Definitionsposter zu *Simulation*

Kurssitzung 5: „Was sind Simulationen?“					
Zeit	**Dauer**	**Ziel**	**Inhalt**	**Methode**	**Medien/Materialien**
25'	50'	Simulationen erstellen lernen in Scratch	Simulation einer Nahrungskette und eines Wasserkreislaufs Wasserkreislaufs	Die Kursleitung teilt das **Arbeitsblatt „Simulationen“** an die Kinder aus und teilt die Kinder in **Zweier-Teams** ein. Jeweils ein Team kann jetzt an einen **Computer** gehen. • Jedes Team **programmiert** nun zuerst die **Nahrungskette** mithilfe der Anleitungen auf dem Arbeitsblatt. Sobald ein Team mit dieser Aufgabe fertig ist, soll es der Kursleitung die fertige Simulation zeigen. • Anschließend **programmiert** das Team mithilfe des Arbeitsblattes den **Wasserkreislauf**. Die fertige Simulation soll wieder der Kursleitung gezeigt werden. Während die Kinder programmieren, **unterstützt** die Kursleitung sie **bei Fragen und Schwierigkeiten**.	Arbeitsblatt „Simulationen“
75'	10'	Vertiefende Übungen/ Kontrolle des Lernfortschritts	Die Kinder bearbeiten Übungsblatt 5	Die Kursleitung teilt **Übungsblatt 5** an die Kinder aus. Dann bearbeiten die Kinder das Übungsblatt, das aus **Knobelaufgaben zu logischem Denken, Sequenzen und Schleifen** besteht. • Die Kursleitung bittet die Kinder, sich zu melden, wenn sie etwas nicht verstehen und **unterstützt die Kinder bei Verständnisschwierigkeiten** mit Erklärungen. Anschließend **sammelt die Kursleitung die Übungsblätter ein**.	Übungsblatt 5
85'	5'	Verabschiedung		Die Kursleitung teilt den Kindern ein Blatt mit Aufgaben aus, die sie **freiwillig zu Hause bearbeiten** können und **verabschiedet** sie bis zur nächsten Kurssitzung. Die Kinder dürfen außerdem das Arbeitsblatt „Simulationen“ mit nach Hause nehmen.	Blatt „Zu Hause ausprobieren“ 5
90'	**Ende von Kurssitzung 5**				

Probleme, die auftreten können:

- Einige der Kinder schaffen es nicht, die Simulation im Kurs fertig zu programmieren.

Mögliche Lösungen für diese Probleme:

- Die Kinder können die Simulation mithilfe des Arbeitsblatts und des Blatts „Zu Hause ausprobieren“ 5 als freiwillige Hausaufgabe bearbeiten.

Kurssitzung 6: „Mathematische Anwendungen simulieren“

Kernaktivität, Lernziele und Materialien

Kernaktivität: Die Kinder programmieren mithilfe des Arbeitsblatts „Mathematische Simulationen“ einen simulierten Flohmarkt.

Lernziele: Die Kinder vertiefen ihr Verständnis von *Simulationen* und deren Einsatzmöglichkeiten. Dazu lernen sie, zur Nachbildung eines Flohmarktes, auf dem Waren für Geld gekauft werden können, eine Simulation zu programmieren. Diese Simulation erfordert die Anwendung mathematiknaher informatischer Konzepte von *Variablen, Konstanten* und *Operatoren* sowie den Einsatz von *Sequenzen, Schleifen, Bedingungen* und *Ereignissen*.

Materialien:

Immer:

- Anwesenheitsliste
- Kurssitzungsplan
- Schreibutensilien

Sitzungsspezifisch:

- Poster mit illustrierten Definitionen der Begriffe Schleife, Ereignis, Variable, Konstante und Operator
- Blatt „Zu Hause ausprobieren“ 5
- Übungsblatt 5 aus der letzten Kurssitzung
- Arbeitsblatt „Mathematische Simulationen“
- Übungsblatt 6
- Blatt „Zu Hause ausprobieren“ 6

Technik:

- So viele Desktop-Computer oder Laptops, dass die Kinder je zu zweit an einem Computer arbeiten können
- Internetzugang ODER installierte Version von Scratch 2.0

Das Klassenzimmer sollte ...

- ... einen Stuhlkreis ermöglichen.
- … genügend Stühle und Tische haben, dass die Kinder sich zum Bearbeiten der Übungsblätter hinsetzen können.

Vorbereitungen:

- Computer hochfahren
- Für alle teilnehmenden Kinder kopieren:
  - Arbeitsblatt „Mathematische Simulationen“
  - Übungsblatt 6
  - Blatt „Zu Hause ausprobieren“ 6

Kurssitzung 6: „Mathematische Anwendungen simulieren“					
Zeit	**Dauer**	**Ziel**	**Inhalt**	**Methode**	**Medien/Materialien**
0'	5'	Ankommen und Vorbereiten	Kinder begrüßen, Materialien vorbereiten	• Sämtliche **Materialien bereitlegen**. • Falls die für den Kurs verwendeten **Computer** sehr lange brauchen, um hochzufahren, kann die Kursleitung sie bereits jetzt **einschalten**.	Alle Materialien
5'	5'	Rückbezug auf die vergangene Kurssitzung und Besprechung der Hausaufgaben		Die Kursleitung setzt sich mit allen teilnehmenden Kindern in einen **Stuhlkreis**. • Die **Kursleitung fragt die Kinder**: o Wisst ihr noch, was wir in der **letzten Kurssitzung** gemacht haben? o Wer von euch hat zu Hause die **Hausaufgaben ausprobiert**? o Welche **Erfahrungen** habt ihr damit gemacht?	Blatt „Zu Hause ausprobieren“ 5
10'	5'	Wiederholung bisher gelernter Konzepte	1. Simulation 2. Variable 3. Konstante 4. Operator	Die Kursleitung setzt sich mit allen teilnehmenden Kindern in einen **Stuhlkreis**. • Die **Kursleitung fragt die Kinder** (für jeden der Begriffe einzeln): **Wisst ihr noch, was das Wort bedeutet**, wenn wir es hier im Kurs verwenden? (Kurzer Rückblick auf das jeweilige Definitionsposter.)	Definitionsposter zu *Simulation, Variable, Konstante* und *Operator*
15'	5'	Besprechung von Übungsblatt 5		• Die Kursleitung teilt den Kindern ihre **Übungsblätter aus der vergangenen Kursstunde** aus und **bespricht die Ergebnisse** mit den Kindern. • Für jede der Aufgaben bittet die Kursleitung ein **Kind, das die richtige Lösung zu haben glaubt**, den anderen den **eigenen Lösungsweg** zu erklären. Dann sollen die anderen Kinder sagen, ob sie dieser **Lösung zustimmen** oder **begründen, falls sie nicht zustimmen**. Die **Kursleitung sammelt die alten Übungsblätter** danach **wieder ein**.	Übungsblatt 5
20'	55'	Mathematische Anwendungen simulieren	Simulation eines Flohmarktes	Die Kursleitung teilt das **Arbeitsblatt „Mathematische Simulationen“** aus und teilt die Kinder in **Zweier-Teams** ein. Jeweils ein Team kann jetzt an einen **Computer** gehen. • Jedes Team **programmiert** nun die **Flohmarkt-Simulation** mithilfe der Anleitungen auf dem Arbeitsblatt. Sobald ein Team mit dieser Aufgabe fertig ist, soll es der Kursleitung das fertige Spiel zeigen. • Anschließend kann das Team die eigene **Flohmarkt-Simulation** entsprechend der Vorschläge auf dem Arbeitsblatt **erweitern** und **mit dem eigenen Programmcode experimentieren.** Während die Kinder programmieren, **unterstützt** die Kursleitung sie **bei Fragen und Schwierigkeiten**.	Arbeitsblatt “Mathematische Simulationen”
75'	10'	Vertiefende Übungen/	Die Kinder bearbeiten Übungsblatt 6	Die Kursleitung teilt **Übungsblatt 6** an die Kinder aus. • Dann bearbeiten die Kinder das Übungsblatt, das aus **Knobelaufgaben zu**	Übungsblatt 6

Kurssitzung 6: „Mathematische Anwendungen simulieren“					
Zeit	**Dauer**	**Ziel**	**Inhalt**	**Methode**	**Medien/Materialien**
		Kontrolle des Lernfortschritts		**Bedingungen, Ereignissen, Variablen, Sequenzen, Schleifen und Parallelismus** besteht. • Die Kursleitung bittet die Kinder, sich zu melden, wenn sie etwas nicht verstehen und **unterstützt die Kinder bei Verständnisschwierigkeiten** mit Erklärungen. Anschließend **sammelt die Kursleitung die Übungsblätter ein**.	
85'	5'	Verabschiedung		Die Kursleitung teilt den Kindern ein Blatt mit Aufgaben aus, die sie **freiwillig zu Hause bearbeiten** können und **verabschiedet** sie bis zur nächsten Kurssitzung. Die Kinder dürfen außerdem das Arbeitsblatt „Mathematische Simulationen“ mit nach Hause nehmen.	Blatt „Zu Hause ausprobieren“ 6
90'	**Ende von Kurssitzung 6**				

4.2.3 Modul 3 – Übertragung informatischer Denkkonzepte auf Hardware-Anwendungen

Kurssitzung	7) „Die ‚Scratch für Arduino'-Umgebung erkunden“	8) „Sensoren und Hardware-Ansteuerung mit Scratch für Arduino“
Modus	plugged-in	plugged-in
Kooperation	Zweier-Teams	Zweier-Teams
Informatische Konzepte	• Alle zuvor eingeführten Konzepte	• Alle zuvor eingeführten Konzepte
Kernmedium	S4A (Scratch für Arduino)	S4A (Scratch für Arduino)
MINT-Bezug	Elektrotechnik, Ingenieurswissenschaft	Elektrotechnik, Ingenieurswissenschaft
Das Übungsblatt enthält...	• Muster • Variablen • Schleifen	• Muster • Schleifen

Lernziele und Inhalte von Modul 3

In diesem Modul lernen die Schülerinnen und Schüler, ihre bisher im Kurs erworbenen Kompetenzen und Kenntnisse informatischer Konzepte auf den Bereich der Hardware-Programmierung anzuwenden. Indem sie die unterschiedlichen Sensoren des Arduino ansteuern und die Funktionalität der elektronischen Komponenten programmieren, erleben die Schülerinnen und Schüler außerdem, dass sie beim Programmieren nicht nur beeinflussen können, was vor ihren Augen auf dem Bildschirm passiert, sondern durch die von ihnen geschriebenen Programme sogar Einfluss auf Geschehnisse in der sie umgebenden physischen Welt nehmen können.

Kurssitzung 7: „Die ‚Scratch für Arduino'-Umgebung erkunden"

Kernaktivität, Lernziele und Materialien

Kernaktivität: Die Kinder steuern mithilfe des Arbeitsblatts „Arduino-Projekte 1" verschiedene Komponenten des Arduino an. Dabei programmieren sie die Funktionalität eines Tasters, mit dem sich ein Licht einschalten lässt, und nutzen den Taster, um Morsezeichen zu geben.

Lernziele: Die Kinder vertiefen ihr Verständnis von *Ereignissen* und *Bedingungen* und deren Nutzen beim Auslesen der Sensordaten einer Hardware. Dabei lernen sie, informatischen Konzepte auf einen Anwendungsbereich zu übertragen, der über das Erstellen einfacher Bildschirmanwendungen hinausgeht und bekommen erste Einblicke in die Hardware-Programmierung.

Materialien:

Immer:
- Anwesenheitsliste
- Kurssitzungsplan
- Schreibutensilien

Sitzungsspezifisch:
- Poster mit illustrierten Definitionen der Begriffe Schleife, Bedingung, Ereignis
- Blatt „Zu Hause ausprobieren" 6
- Übungsblatt 6 aus der letzten Kurssitzung
- Arduino-Einführungsblatt
- Arduino-Einführung für die Kursleitung1
- Arbeitsblatt „Arduino-Projekte 1"
- Übungsblatt 7
- Blatt „Zu Hause ausprobieren" 7

Technik:
- So viele Desktop-Computer oder Laptops, dass die Kinder je zu zweit an einem Computer arbeiten können
- „Arduino Uno"-Kit mit installierter Firmware
- Software: S4A (Scratch for Arduino)

Das Klassenzimmer sollte ...
- ... einen Stuhlkreis ermöglichen.
- … genügend Stühle und Tische haben, dass die Kinder sich zum Bearbeiten der Übungsblätter hinsetzen können.

Vorbereitungen:
- Die Software S4A (Scratch for Arduino) auf allen Computern installieren, an denen die Kinder im Kurs arbeiten werden
- Die Firmware auf dem Arduino-Board installieren
- Den ersten Arduino-Aufbau vornehmen
- Computer hochfahren
- Für alle teilnehmenden Kinder kopieren:
  - Arduino-Einführungsblatt
  - Arbeitsblatt „Arduino-Projekte 1"
  - Übungsblatt 7
  - Blatt „Zu Hause ausprobieren" 7

Kurssitzung 7: „Die ‚Scratch für Arduino'-Umgebung erkunden"					
Zeit	**Dauer**	**Ziel**	**Inhalt**	**Methode**	**Medien/Materialien**
0'	5'	Ankommen und Vorbereiten	Kinder begrüßen, Materialien vorbereiten	• Sämtliche **Materialien bereitlegen**. • Falls die für den Kurs verwendeten **Computer** sehr lange brauchen, um hochzufahren, kann die Kursleitung sie bereits jetzt **einschalten**.	Alle Materialien
5'	5'	Rückbezug auf die vergangene Kurssitzung und Besprechung der Hausaufgaben		Die Kursleitung setzt sich mit allen teilnehmenden Kindern in einen **Stuhlkreis**. • Die **Kursleitung fragt die Kinder**: o Wisst ihr noch, was wir in der **letzten Kurssitzung** gemacht haben? o Wer von euch hat zu Hause die **Hausaufgaben ausprobiert**? o Welche **Erfahrungen** habt ihr damit gemacht?	Blatt „Zu Hause ausprobieren" 6
10'	5'	Wiederholung bisher gelernter Konzepte	1. Schleife 2. Bedingung 3. Ereignis	Die Kursleitung setzt sich mit allen teilnehmenden Kindern in einen **Stuhlkreis**. • Die **Kursleitung fragt die Kinder** (für jeden der Begriffe einzeln): **Wisst ihr noch, was das Wort bedeutet**, wenn wir es hier im Kurs verwenden? (Kurzer Rückblick auf das jeweilige Definitionsposter.)	Definitionsposter zu *Schleife, Bedingung* und *Ereignis*
15'	5'	Besprechung von Übungsblatt 6		• Die Kursleitung teilt den Kindern ihre **Übungsblätter aus der vergangenen Kursstunde** aus und **bespricht die Ergebnisse** mit den Kindern. • Für jede der Aufgaben bittet die Kursleitung ein **Kind, das die richtige Lösung zu haben glaubt**, den anderen den **eigenen Lösungsweg** zu erklären. Dann sollen die anderen Kinder sagen, ob sie dieser **Lösung zustimmen** oder **begründen, falls sie nicht zustimmen**. Die **Kursleitung sammelt die alten Übungsblätter** danach **wieder ein**.	Übungsblatt 6
20'	10'	Den Arduino und S4A vorstellen		• Die Kursleitung teilt den Kindern das **Arduino-Einführungsblatt** aus und geht es Schritt für Schritt mit den Kindern durch, um: o die **Komponenten des Arduino-Kits** zu erklären. o die **"Scratch für Arduino"-Programmierumgebung** vorzustellen. • Dabei zeigt die Kursleitung auch den **Arduino,** das **Steckbrett** und die **elektronischen Komponenten**.	Arduino-Einführungsblatt
30'	45'	Erarbeiten der S4A-Umgebung	Arbeit an ersten Arduino-Projekten	Die Kursleitung teilt das **Arbeitsblatt „Arduino-Projekte 1"** an die Kinder aus und teilt die Kinder in **Zweier-Teams** ein. Jeweils ein Team kann jetzt an einen **Computer** gehen. • Während die Kinder programmieren, geht die Kursleitung mit dem **Arduino** herum und **schließt ihn wiederholt an die Computer der Teams an**, um den **Programmcode der Teams zu testen**.	Computer, Arduino, S4A-Umgebung, Arbeitsblatt „Arduino-Projekte 1"

Kurssitzung 7: „Die ‚Scratch für Arduino'-Umgebung erkunden"					
Zeit	**Dauer**	**Ziel**	**Inhalt**	**Methode**	**Medien/Materialien**
				• Jedes Team **programmiert** zuerst den **Lichttaster** mithilfe der Anleitungen auf dem Arbeitsblatt. Sobald ein Team mit dieser Aufgabe fertig ist, wartet es darauf, dass die Kursleitung den **Arduino** an den Computer anschließt, um den **Programmcode** zu **testen**. • Anschließend **programmiert** das Team mithilfe des Arbeitsblattes den **Morsecode**. Der fertige **Programmcode** soll wieder der Kursleitung gezeigt und mit dem **Arduino getestet** werden.	
75'	10'	Vertiefende Übungen/ Kontrolle des Lernfortschritts	Die Kinder bearbeiten Übungsblatt 7	Die Kursleitung teilt **Übungsblatt 7** an die Kinder aus. • Dann bearbeiten die Kinder das Übungsblatt, das aus **Knobelaufgaben zu logischem Denken, Mustererkennung, Variablen und Schleifen** besteht. • Die Kursleitung bittet die Kinder, sich zu melden, wenn sie etwas nicht verstehen und **unterstützt die Kinder bei Verständnisschwierigkeiten** mit Erklärungen. Anschließend **sammelt die Kursleitung die Übungsblätter ein**.	Übungsblatt 7
85'	5'	Verabschiedung		Die Kursleitung teilt den Kindern ein Blatt mit Aufgaben aus, die sie **freiwillig zu Hause bearbeiten** können und **verabschiedet** sie bis zur nächsten Kurssitzung.	Blatt „Zu Hause ausprobieren" 7
90'	**Ende von Kurssitzung 7**				

Probleme, die auftreten können:

- Die LED funktioniert nicht.
- Der Code wird nicht ausgeführt und der Arduino reagiert nicht wie gewünscht oder erwartet.

Mögliche Lösungen für diese Probleme:

- Der längere Draht der LED sollte in Richtung des Widerstands gesteckt werden und der kürzere in Richtung Minus.
- Auf vielen Computern werden nicht alle Ports des Arduino angezeigt. Falls dieses Problem auftritt, sollten die Komponenten, die mit diesen Ports verbunden sind, stattdessen in einen der vom Computer angezeigten Ports gesteckt werden.
  Wichtig: Dann muss auch der Code entsprechend geändert werden, damit die richtigen Ports angesteuert werden können (z.B. Port 10 statt Port 9, falls 9 nicht angezeigt wird).

Kurssitzung 8: „Sensoren und Hardware-Ansteuerung mit Scratch für Arduino“
Kernaktivität, Lernziele und Materialien

Kernaktivität: Die Kinder programmieren anhand des Arbeitsblatts „Arduino-Projekte 2“ die Funktionalität eines Lichtsensors, der je nach Raumhelligkeit ein Licht einschaltet, sowie die Funktionsweise einer Fußgängerampel in Schleifenschaltung.

Lernziele: Die Kinder erweitern ihre Kompetenzen im Umgang mit den Sensordaten einer Hardware. Dabei wenden sie Konzepte von *Sequenzen, Schleifen, Ereignissen, Bedingungen, Variablen, Konstanten* und *Operatoren* auf den Bereich der Hardware-Programmierung an.

Materialien:

Immer:
- Anwesenheitsliste
- Kurssitzungsplan
- Schreibutensilien

Sitzungsspezifisch:
- Poster mit illustrierten Definitionen der Begriffe Ereignis, Bedingung, Variable, Konstante
- Blatt „Zu Hause ausprobieren“ 6
- Übungsblatt 6 aus der letzten Kurssitzung
- Arduino-Einführung für die Kursleitung 2
- Arbeitsblatt „Arduino-Projekte 2“
- Übungsblatt 8
- Blatt „Zu Hause ausprobieren“ 8

Technik:
- So viele Desktop-Computer oder Laptops, dass die Kinder je zu zweit an einem Computer arbeiten können
- „Arduino Uno“-Kit mit installierter Firmware
- Software: S4A (Scratch for Arduino)

Das Klassenzimmer sollte ...
- ... einen Stuhlkreis ermöglichen.
- … genügend Stühle und Tische haben, dass die Kinder sich zum Bearbeiten der Übungsblätter hinsetzen können.

Vorbereitungen:
- Den ersten Arduino-Aufbau vornehmen
- Computer hochfahren
- Für alle teilnehmenden Kinder kopieren:
  - Arbeitsblatt „Arduino-Projekte 2“
  - Übungsblatt 8
  - Blatt „Zu Hause ausprobieren“ 8

Kurssitzung 8: „Sensoren und Hardware-Ansteuerung mit Scratch für Arduino“					
Zeit	**Dauer**	**Ziel**	**Inhalt**	**Methode**	**Medien/Materialien**
0'	5'	Ankommen und Vorbereiten	Kinder begrüßen, Materialien vorbereiten	• Sämtliche **Materialien bereitlegen**. • Falls die für den Kurs verwendeten **Computer** sehr lange brauchen, um hochzufahren, kann die Kursleitung sie bereits jetzt **einschalten**.	Alle Materialien
5'	5'	Rückbezug auf die vergangene Kurssitzung und Besprechung der Hausaufgaben		Die Kursleitung setzt sich mit allen teilnehmenden Kindern in einen **Stuhlkreis**. • Die **Kursleitung fragt die Kinder**: o Wisst ihr noch, was wir in der **letzten Kurssitzung** gemacht haben? o Wer von euch hat zu Hause die **Hausaufgaben ausprobiert**? o Welche **Erfahrungen** habt ihr damit gemacht?	Blatt „Zu Hause ausprobieren“ 7
10'	5'	Wiederholung bisher gelernter Konzepte	1. Ereignis 2. Bedingung 3. Variable 4. Konstante	Die Kursleitung setzt sich mit allen teilnehmenden Kindern in einen **Stuhlkreis**. • Die **Kursleitung fragt die Kinder** (für jeden der Begriffe einzeln): **Wisst ihr noch, was das Wort bedeutet**, wenn wir es hier im Kurs verwenden? (Kurzer Rückblick auf das jeweilige Definitionsposter.)	Definitionsposter zu *Ereignis,Bedingung, Variable* und *Konstante*
15'	5'	Besprechung von Übungsblatt 7		• Die Kursleitung teilt den Kindern ihre **Übungsblätter aus der vergangenen Kursstunde** aus und **bespricht die Ergebnisse** mit den Kindern. • Für jede der Aufgaben bittet die Kursleitung ein **Kind, das die richtige Lösung zu haben glaubt**, den anderen den **eigenen Lösungsweg** zu erklären. Dann sollen die anderen Kinder sagen, ob sie dieser **Lösung zustimmen** oder **begründen, falls sie nicht zustimmen**. Die **Kursleitung sammelt die alten Übungsblätter** danach **wieder ein**.	Übungsblatt 7
20'	55'	Vertiefendes Arbeiten in der S4A-Umgebung	Arbeit an weiteren Arduino-Projekten	Die Kursleitung teilt das **Arbeitsblatt „Arduino-Projekte 2“** aus und teilt die Kinder in **Zweier-Teams** ein. Jedes Team kann jetzt an einen **Computer** gehen. • Während die Kinder programmieren, geht die Kursleitung mit dem **Arduino** herum und **schließt ihn wiederholt an die Computer der Teams an**, um den **Programmcode der Teams zu testen**. Sobald alle Teams ihr **erstes Projekt testen** konnten und es funktioniert, **ändert** die Kursleitung den **Arduino-Aufbau**, sodass die Kinder am zweiten Projekt arbeiten können. • Jedes Team **programmiert** zuerst den **Lichtsensor** mithilfe der Anleitungen auf dem Arbeitsblatt. Sobald ein Team mit dieser Aufgabe fertig ist, wartet es darauf, dass die Kursleitung den **Arduino** an den Computer anschließt, um den **Programmcode** zu **testen**. • Anschließend **programmiert** das Team mithilfe des Arbeitsblattes die **Fußgängerampel**. Der fertige **Programmcode** soll wieder der Kursleitung gezeigt und mit dem **Arduino getestet** werden.	Computer, Arduino, S4A-Umgebung, Arbeitsblatt „Arduino-Projekte 2“

Kurssitzung 8: „Sensoren und Hardware-Ansteuerung mit Scratch für Arduino“					
Zeit	**Dauer**	**Ziel**	**Inhalt**	**Methode**	**Medien/Materialien**
75'	10'	Vertiefende Übungen/ Kontrolle des Lernfortschritts	Die Kinder bearbeiten Übungsblatt 8	Die Kursleitung teilt **Übungsblatt 8** an die Kinder aus. • Dann bearbeiten die Kinder das Übungsblatt, das aus **Knobelaufgaben zu Mustererkennung, logischem Denken und Schleifen** besteht. • Die Kursleitung bittet die Kinder, sich zu melden, wenn sie etwas nicht verstehen und **unterstützt die Kinder bei Verständnisschwierigkeiten**. Anschließend **sammelt die Kursleitung die Übungsblätter ein**.	Übungsblatt 8
85'	5'	Verabschiedung		Die Kursleitung teilt den Kindern ein Blatt mit Aufgaben aus, die sie **freiwillig zu Hause bearbeiten** können und **verabschiedet** sie bis zur nächsten Kurssitzung.	Blatt „Zu Hause ausprobieren“ 8
90'	**Ende von Kurssitzung 8**				

Probleme, die auftreten können:

- Die LED funktioniert nicht.
- Der Code wird nicht ausgeführt und der Arduino reagiert nicht wie gewünscht oder erwartet.
- Sie sind sich nicht sicher, mit welchem Wert Sie die vom Lichtsensor gegebene Lichtvariable vergleichen sollen.

Mögliche Lösungen für diese Probleme:

- Der längere Draht der LED sollte in Richtung des Widerstands gesteckt werden und der kürzere in Richtung Minus.
- Auf vielen Computern werden nicht alle Ports des Arduino angezeigt. Falls dieses Problem auftritt, sollten die Komponenten, die mit diesen Ports verbunden sind, stattdessen in einen der vom Computer angezeigten Ports gesteckt werden.
  Wichtig: Dann muss auch der Code entsprechend geändert werden, damit die richtigen Ports angesteuert werden können (z.B. Port 10 statt Port 9, falls 9 nicht angezeigt wird).
- Zuerst muss eingelesen werden, welcher Wert für den Port angezeigt wird, der mit dem Lichtsensor verbunden ist (im Lösungsbeispiel Analog0). Die LED soll dann eingeschaltet werden, wenn es dunkler wird, z.B. wenn die Umgebung des Sensors abgedunkelt wird, indem man eine Hand darüber hält. Für die Bedingung (Licht < …) sollte ein Wert gewählt werden, der unter dem Wert liegt, der bei normaler Raumhelligkeit für den Port angezeigt wird, der aber unterschritten wird, wenn man z.B. eine Hand über den Sensor hält.

4.2.4 Modul 4 – Eigenständiges Anwenden grundlegender informatischer Konzepte in der interaktiven Robotersimulation Open Roberta

Kurssitzung	9) „Die ‚Open Roberta'-Umgebung erkunden"	10) „Interaktive Robotersteuerung mit ‚Open Roberta'"
Modus	plugged-in	plugged-in
Kooperation	Zweier-Teams	Zweier-Teams
Informatische Konzepte	• Alle zuvor eingeführten Konzepte	• Alle zuvor eingeführten Konzepte
Kernmedium	Open Roberta	Open Roberta
MINT-Bezug	Robotik, Ingenieurswissenschaft, Informatik	Robotik, Ingenieurswissenschaft, Informatik
Das Übungsblatt enthält...	• Sequenzen	• Bedingungen • Ereignissen • Sequenzen • Schleifen

Lernziele und Inhalte von Modul 4

In diesem Modul haben die Schülerinnen und Schüler abschließend die Gelegenheit, alle bisher im Kurs erworbenen Kompetenzen und Kenntnisse informatischer Konzepte eigenständig auf das für sie neue Problemfeld der Steuerung eines simulierten Roboters anzuwenden..

Die Robotersimulation gibt den Schülerinnen und Schülern die Möglichkeit, eigenständiger zu arbeiten als in den Modulen 1 bis 3. Beim Programmieren des simulierten Roboters werden die Schülerinnen und Schüler zu freien Problemlösungsversuchen ermutigt, da es keinen vorgegebenen Problemlösungsansatz gibt, sondern das Aufgabenziel auf vielen verschiedenen Wegen erreicht werden kann.

Kurssitzung 9: „Die ‚Open Roberta'-Umgebung erkunden"
Kernaktivität, Lernziele und Materialien

Kernaktivität: Die Kinder entwickeln anhand der „Roberta-Einführung" und eigener Ideen einen Algorithmus, der Roberta durch den Einsatz ihrer Farbsensoren tanzen lässt.

Lernziele: Die Kinder übertragen ihre bisher erworbenen informatischen Fähigkeiten und Kompetenzen auf die Programmierung in einer für sie neuen Programmiersprache und -umgebung. Dabei wenden sie eigenständig Konzepte von *Sequenzen, Schleifen, Ereignissen* und *Bedingungen* zur Steuerung eines simulierten Roboters an.

Materialien:

Immer:
- Anwesenheitsliste
- Kurssitzungsplan
- Schreibutensilien

Sitzungsspezifisch:
- Poster mit illustrierten Definitionen der Begriffe ...
- Blatt „Zu Hause ausprobieren" 8
- Übungsblatt 8 aus der letzten Kurssitzung
- Roberta-Einführungsblatt
- Übungsblatt 9
- Blatt „Zu Hause ausprobieren" 9

Technik:
- So viele Desktop-Computer oder Laptops, dass die Kinder je zu zweit an einem Computer arbeiten können
- Internetzugang

Das Klassenzimmer sollte ...
- ... einen Stuhlkreis ermöglichen.
- … genügend Stühle und Tische haben, dass die Kinder sich zum Bearbeiten der Übungsblätter hinsetzen können.

Vorbereitungen:
- Computer hochfahren
- Für alle teilnehmenden Kinder kopieren:
  - Roberta-Einführungsblatt
  - Übungsblatt 9
  - Blatt „Zu Hause ausprobieren" 9

Kurssitzung 9: „Die ‚Open Roberta'-Umgebung erkunden"					
Zeit	**Dauer**	**Ziel**	**Inhalt**	**Methode**	**Medien/Materialien**
0'	5'	Ankommen und Vorbereiten	Kinder begrüßen, Materialien vorbereiten	• Sämtliche **Materialien bereitlegen**. • Falls die für den Kurs verwendeten **Computer** sehr lange brauchen, um hochzufahren, kann die Kursleitung sie bereits jetzt **einschalten**.	Alle Materialien
5'	5'	Rückbezug auf die vergangene Kurssitzung und Besprechung der Hausaufgaben		Die Kursleitung setzt sich mit allen teilnehmenden Kindern in einen **Stuhlkreis**. • Die **Kursleitung fragt die Kinder**: o Wisst ihr noch, was wir in der **letzten Kurssitzung** gemacht haben? o Wer von euch hat zu Hause die **Hausaufgaben ausprobiert**? o Welche **Erfahrungen** habt ihr damit gemacht?	Blatt „Zu Hause ausprobieren" 8
10'	5'	Wiederholung bisher gelernter Konzepte	1. Simulation 2. Ereignis 3. Bedingung 4. Algorithmus	Die Kursleitung setzt sich mit allen teilnehmenden Kindern in einen **Stuhlkreis**. • Die **Kursleitung fragt die Kinder** (für jeden der Begriffe einzeln): • **Wisst ihr noch, was das Wort bedeutet**, wenn wir es hier im Kurs verwenden? (Kurzer Rückblick auf das jeweilige Definitionsposter.)	Definitionsposter zu *Simulation, Ereignis, Bedingung* und *Algorithmus*
15'	5'	Besprechung von Übungsblatt 8		• Die Kursleitung teilt den Kindern ihre **Übungsblätter aus der vergangenen Kursstunde** aus und **bespricht die Ergebnisse** mit den Kindern. • Für jede der Aufgaben bittet die Kursleitung ein **Kind, das die richtige Lösung zu haben glaubt**, den anderen den **eigenen Lösungsweg** zu erklären. Dann sollen die anderen Kinder sagen, ob sie dieser **Lösung zustimmen** oder **begründen, falls sie nicht zustimmen**. Die **Kursleitung sammelt die alten Übungsblätter** danach **wieder ein**.	Übungsblatt 8
20'	55'	Robertas Sensoren kennenlernen	Programmierung von Open Roberta mithilfe der Farbsensoren	Die Kursleitung teilt das **Roberta-Einführungsblatt** an die Kinder aus und teilt die Kinder in **Zweier-Teams** ein. Jeweils ein Team kann jetzt an einen **Computer** gehen. • Die Kinder folgen dem Roberta-Einführungsblatt entsprechend der **Tour durch die „Open Roberta"-Umgebung**. • Jedes Team **probiert nun aus**, mithilfe der Anweisungen vom Roberta-Einführungsblatt **Roberta** über die **bunten Felder** fahren und dabei **tanzen zu lassen**. • Sobald ein Team mit dieser Aufgabe fertig ist, soll es der Kursleitung den fertigen Programmcode zeigen.	Roberta-Einführungsblatt

Kurssitzung 9: „Die ‚Open Roberta'-Umgebung erkunden"					
Zeit	**Dauer**	**Ziel**	**Inhalt**	**Methode**	**Medien/Materialien**
				Während die Kinder programmieren, **unterstützt** die Kursleitung sie **bei Fragen und Schwierigkeiten**.	
75'	10'	Vertiefende Übungen/ Kontrolle des Lernfortschritts	Die Kinder bearbeiten Übungsblatt 9	Die Kursleitung teilt **Übungsblatt 9** an die Kinder aus. • Dann bearbeiten die Kinder das Übungsblatt, das aus **Knobelaufgaben zu Sequenzen, räumlicher Orientierung und logischem Denken** besteht. • Die Kursleitung bittet die Kinder, sich zu melden, wenn sie etwas nicht verstehen und **unterstützt die Kinder bei Verständnisschwierigkeiten** mit Erklärungen. Anschließend **sammelt die Kursleitung die Übungsblätter ein**.	Übungsblatt 9
85'	5'	Verabschiedung		Die Kursleitung teilt den Kindern ein Blatt mit Aufgaben aus, die sie **freiwillig zu Hause bearbeiten** können und **verabschiedet** sie bis zur nächsten Kurssitzung.	Blatt „Zu Hause ausprobieren" 9
90'	**Ende von Kurssitzung 9**				

Probleme, die auftreten können:

- Der Roberta-Roboter bewegt sich nicht gemäß den angegebenen Befehlen.

Mögliche Lösungen für diese Probleme:

- Der Roberta-Roboter dreht manchmal nicht exakt um die angegebene Gradzahl. Falls also der Programmcode eines Teams nicht wie gewünscht funktioniert, kann es sich auch um eine Fehlfunktion von Roberta handeln. Bitte versuchen Sie, das Programm zu stoppen, Roberta an die Ausgangsposition zurückzusetzen und das Programm erneut auszuführen.

Kurssitzung 10: „Interaktive Robotersteuerung mit ‚Open Roberta'"

Kernaktivität, Lernziele und Materialien

Kernaktivität: Die Kinder bearbeiten das Arbeitsblatt „Roberta-Projekt" und entwickeln eigenständig einen Wegfindungsalgorithmus in Form von Programmcode, der Roberta nacheinander auf zwei unterschiedlichen Parkplätzen einparken lässt.

Lernziele: Die Kinder setzen ihre bisher erworbenen informatischen Fähigkeiten und Kompetenzen ein, um eigenständig einen Algorithmus zur Lösung des vorgegebenen Programmierproblems zu entwickeln. Dabei entscheiden sie eigenständig, welche informatischen Konzepte sie dafür einsetzen müssen.

Materialien:

Immer:

- Anwesenheitsliste
- Kurssitzungsplan
- Schreibutensilien

Das Klassenzimmer sollte ...

- ... einen Stuhlkreis ermöglichen.
- … genügend Stühle und Tische haben, dass die Kinder sich zum Bearbeiten der Übungsblätter hinsetzen können.

Sitzungsspezifisch:

- Poster mit illustrierten Definitionen der Begriffe Schleife, Bedingung, und Ereignis
- Blatt „Zu Hause ausprobieren" 9
- Übungsblatt 9 aus der letzten Kurssitzung
- Arbeitsblatt „Roberta-Projekt"
- Übungsblatt 10
- Blatt „Zu Hause ausprobieren" 10

Vorbereitungen:

- Computer hochfahren
- Für alle teilnehmenden Kinder kopieren:
  - Arbeitsblatt „Roberta-Projekt"
  - Übungsblatt 10
  - Blatt „Zu Hause ausprobieren" 10

Technik:

- So viele Desktop-Computer oder Laptops, dass die Kinder je zu zweit an einem Computer arbeiten können
- Internetzugang

Kurssitzung 10: „Interaktive Robotersteuerung mit ‚Open Roberta'"					
Zeit	**Dauer**	**Ziel**	**Inhalt**	**Methode**	**Medien/Materialien**
0'	5'	Ankommen und Vorbereiten	Kinder begrüßen, Materialien vorbereiten	• Sämtliche **Materialien bereitlegen**. • Falls die für den Kurs verwendeten **Computer** sehr lange brauchen, um hochzufahren, kann die Kursleitung sie bereits jetzt **einschalten**.	Alle Materialien
5'	5'	Rückbezug auf die vergangene Kurssitzung und Besprechung der Hausaufgaben		Die Kursleitung setzt sich mit allen teilnehmenden Kindern in einen **Stuhlkreis**. • Die **Kursleitung fragt die Kinder**: o Wisst ihr noch, was wir in der **letzten Kurssitzung** gemacht haben? o Wer von euch hat zu Hause die **Hausaufgaben ausprobiert**? o Welche **Erfahrungen** habt ihr damit gemacht?	Blatt „Zu Hause ausprobieren" 9
10'	10'	Wiederholung aller bisher gelernten Konzepte	1. Sequenz 2. Schleife 3. Muster 4. Ereignis 5. Bedingung 6. Variable 7. Konstante 8. Operator 9. Algorithmus 10. Simulation	Die Kursleitung setzt sich mit allen teilnehmenden Kindern in einen **Stuhlkreis**. • **„Heute wiederholen wir zum Abschluss nochmal alle Begriffe, die ihr in diesem Kurs gelernt habt!"** • Die **Kursleitung fragt die Kinder** (für jeden der Begriffe einzeln): • **Wisst ihr noch, was das Wort bedeutet**, wenn wir es hier im Kurs verwenden? (Kurzer Rückblick auf das jeweilige Definitionsposter.)	Definitionsposter zu *Sequenz, Schleife, Muster, Ereignis, Bedingung, Variable, Konstante, Operator, Algorithmus* und *Simulation*
20'	5'	Besprechung von Übungsblatt 9		• Die Kursleitung teilt den Kindern ihre **Übungsblätter aus der vergangenen Kursstunde** aus und **bespricht die Ergebnisse** mit den Kindern. • Für jede der Aufgaben bittet die Kursleitung ein **Kind, das die richtige Lösung zu haben glaubt**, den anderen den **eigenen Lösungsweg** zu erklären. Dann sollen die anderen Kinder sagen, ob sie dieser **Lösung zustimmen** oder **begründen, falls sie nicht zustimmen**. Die **Kursleitung sammelt die alten Übungsblätter** danach **wieder ein**.	Übungsblatt 9
25'	50'	Roberta intelligent Wege finden lassen	Programmierung von Open Roberta mithilfe der Farbsensoren	Die Kursleitung teilt das **Arbeitsblatt „Roberta-Projekt"** an die Kinder aus und lässt die Kinder zum Abschluss des Kurses **Zweier-Teams** mit einem Wunschpartner bilden. Jeweils ein Team kann jetzt an einen **Computer** gehen. • Jedes Team **probiert zunächst aus**, anhand des Arbeitsblatts „Roberta-Projekte" **Roberta** durch Einsatz der **Farbsensoren** ihren Weg zum **ersten eingekreisten Parkplatz** finden zu lassen. • Sobald ein Team mit dieser Aufgabe fertig ist, soll es vor der Kursleitung den fertigen Programmcode ausführen und zeigen, wie Roberta ihr Ziel findet.	Arbeitsblatt „Roberta-Projekte"

Kurssitzung 10: „Interaktive Robotersteuerung mit ‚Open Roberta'"					
Zeit	**Dauer**	**Ziel**	**Inhalt**	**Methode**	**Medien/Materialien**
				• Wenn ein Team mit der ersten Aufgabe fertig ist, kann es **ausprobieren**, anhand des Arbeitsblatts „Roberta-Projekte" **Roberta** mithilfe der **Farbsensoren** den Weg zum **zweiten eingekreisten Parkplatz** zu steuern. • Sobald ein Team mit dieser Aufgabe fertig ist, soll es vor der Kursleitung wieder den fertigen Programmcode ausführen und zeigen, wie Roberta ihr Ziel findet. Während die Kinder programmieren, **unterstützt** die Kursleitung sie **bei Fragen und Schwierigkeiten**.	
75'	10'	Vertiefende Übungen/ Kontrolle des Lernfortschritts	Die Kinder bearbeiten Übungsblatt 10	Die Kursleitung teilt **Übungsblatt 10** an die Kinder aus. • Dann bearbeiten die Kinder das Übungsblatt, das aus **Knobelaufgaben zu Bedingungen, Ereignissen, Sequenzen, Schleifen, räumlicher Orientierung und Variablen** besteht. • Die Kursleitung bittet die Kinder, sich zu melden, wenn sie etwas nicht verstehen und **unterstützt die Kinder bei Verständnisschwierigkeiten** mit Erklärungen. Anschließend **sammelt die Kursleitung die Übungsblätter ein**.	Übungsblatt 10
85'	5'	Verabschiedung		Die Kursleitung teilt den Kindern ein Blatt mit über den Kurs hinausgehenden Lernmaterialien aus, die sie **freiwillig zu Hause bearbeiten** können und **verabschiedet** alle Kinder aus dem Kurs „Verstehen wie Computer denken".	Blatt „Zu Hause ausprobieren" 10
90'	**Ende von Kurssitzung 10**				

Probleme, die auftreten können:

- Der Roberta-Roboter bewegt sich nicht gemäß den angegebenen Befehlen.

Mögliche Lösungen für diese Probleme:

- Der Roberta-Roboter dreht manchmal nicht exakt um die angegebene Gradzahl. Falls also der Programmcode eines Teams nicht wie gewünscht funktioniert, kann es sich auch um eine Fehlfunktion von Roberta handeln. Bitte versuchen Sie, das Programm zu stoppen, Roberta an die Ausgangsposition zurückzusetzen und das Programm erneut auszuführen.

5 LITERATURVERZEICHNIS

Apostolellis, P., Stewart, M., Frisina, C. and Kafura, D. (2014) 'RaBit EscAPE: A Board Game for Computational Thinking', *Proceedings of the 2014 conference on Interaction design and children - IDC '14*, pp. 349–352.

Balanskat, A. & Engelhardt, K. (2015). *Computing our future. Computer programming and coding – priorities, school curricula and initiatives across Europe*. Brussels: European Schoolnet. Retrieved from http://fcl.eun.org/documents/10180/14689/Computing+our+future_final.pdf/746e36b1-e1a6-4bf1-8105-ea27c0d2bbe0

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: what is Involved and what is the role of the computer science education community?. *ACM Inroads*, *2*(1), 48-54. doi:10.1145/1929887.1929905

Barsalou, L.W. (2008). Grounded cognition. *Annual review of psychology*, *59*, 617-645.

Boyle E.A., Hainey T., Connolly T.M., Gray G., Earp J., Ott M.,Lim T., Ninaus M., Pereira J., & Riberio C. (2016). An update to the systematic literature review of empirical evidence of the impacts and outcomes of computer games and serious games, *Computers and Education*, *94*.

Brennan, K. & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. *Annual American Educational Research Association Meetin*g, 1-25. Retrieved from http://scratched.gse.harvard.edu/ct/files/AERA2012.pdf

Briggs, J. (2014). Computational Thinker: Concepts & Approaches. CAS Barefoot. Retrieved from http://barefootcas.org.uk/wp-content/uploads/2016/08/Barefoot-Computational-Thinking-Poster.pdf .

Brisson, B.M., Dicke, A.L., Gaspard, H., Häfner, I., Flunger, B., Nagengast, B., & Trautwein, U. (2017). Short intervention, sustained effects: Promoting students' math competence beliefs, effort, and achievement. *American Educational Research Journal*, *54*(6), 1048-1078. doi:10.3102/0002831217716084

Butz, M. V. (2016). Toward a unified sub-symbolic computational theory of cognition. *Frontiers in Psychology*, *7*, 1-19. doi:10.3389/fpsyg.2016.00925

Code.org (n.d.). *About Us*. https://code.org/about

Computing at School (n.d.). Educate, Engage, Encourage. http://www.computingatschool.org.uk

Cuny, J., Snyder L. and Wing J. M. (2010). Demystifying Computational Thinking for Non-Computer Scientists. (work in progress)

Curzon, P. & McOwen, P. W. (2017). The power of Computational Thinking. *World Scientific*, *1*, https://doi.org/10.1142/q0054

Dredge, S. (2014). Coding at school: a parent's guide to England's new computing curriculum. https://www.theguardian.com/technology/2014/sep/04/coding-schoolcomputing-children-program-ming

Echeverría, A., García-Campo, C., Nussbaum, M., Gil, F., Villalta, M., Améstica, M. and Echeverría, S. (2011) A framework for the design and integration of collaborative classroom games. *Computers & Education.* 57(1), pp. 1127–1136.

Fullerton, T. (2008). Game Design Workshop: A Playcentric Approach to Creating Innovative Games, United States: Elsevier.

García-Peñalvo, F., Reimann, D., Tuul, M., Rees, A. & Jormanainen, I. (2016). TACCLE 3, O5: An overview of the most relevant literature on coding and computational thinking with emphasis on the relevant issues for teachers. S. 3-8.

Gaspard, H., Dicke, A.L., Flunger, B., Schreier, B., Häfner, I., Trautwein, U., & Nagengast, B. (2015). More value through greater differentiation: Gender differences in value beliefs about math. *Journal of educational psychology*, *107*(3), 663. doi:10.1037/edu0000003

Guzdial, M. (2008). Education: Paving the Way for Computational Thinking. *Communications of the Association for Computing Machinery (ACM)*, *51*(8), 25-27. doi:10.1145/1378704.1378713

Haffner, J., Baro, K., Parzer, P. & Resch, F. (2005). HRT 1-4. Heidelberger Rechentest. Erfassung mathematischer Basiskompetenzen im Grundschulalter. Göttingen: Hogrefe.

Hamari, J., Koivisto, J., & Sarsa, H. (2014). Does gamification work? – A literature review of empirical studies on gamification. *Proceedings of the Annual Hawaii International Conference on System Sciences*, 3025–3034. doi:10.1109/HICSS.2014.377

Heller, K.A. & Perleth, C. (2000). KFT 4-12+R - Kognitiver Fähigkeits-Test für 4. bis 12. Klassen, Revision. Göttingen: Beltz.

Hughes, J. (1989). Why functional programming matters. *The computer journal*, *32*(2), 98-107.

Informatik-Biber (n.d.) http://bwinf.de//biber/

K-12 Computer Science Framework (n.d.). https://k12cs.org/

Kazimoglu, C. (2013) 'Empirical evidence that proves a serious game is an educationally effective tool for learning computer programming constructs at the computational thinking level'.

Kumar, D. (2014). Welcome. *ACM Inroads*, *5*(4), 52-53. doi:10.1145/2684721.2684733

Leacock, M. (2008) 'Pandemic' [Board game], Z-Man Games: Mahopac, NY.

Leifheit, L., Jabs, J., Ninaus, M., Moeller, K., & Ostermann, K. (2018). *Programming Unplugged: An Evaluation of Game-Based Methods for Teaching Computational Thinking in Primary School,* Proceedings of the 12th European Conference on Game Based Learning.

Lister, R. (2016). Toward a developmental epistemology of computer programming. In *Proceedings of the 11th workshop in primary and secondary computing education*, ACM, 5-16. doi:10.1145/2978249.2978251

Lu, J.J., & Fletcher, G.H. (2009). Thinking about computational thinking. *ACM SIGCSE Bulletin*, *41*(1), 260-264.

Ninaus, M., Pereira, G., Stefitz, R., Prada, R., Paiva, A., & Wood, G. (2015). Game elements improve performance in a working memory training task, *International Journal of Serious Games*, *2*(1), 3-16. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.672.7664&rep=rep1&type=pdf

Noice, H. & Noice, T. (2001). Learning dialogue with and without movement. *Memory & Cognition*, *29*(6), 820-827. doi:10.3758/BF03196411

Open-Roberta Lab (n.d.) http://lab.open-roberta

Papert, S. (1972). Teaching Children Thinking. *Programmed Learning and Educational Technology*, *9*(5), 245-255. doi:10.1080/1355800720090503

Papert, S. (1980). Mindstorms: Children, computers, and powerful ideas. Basic Books, Inc.

Papert, S. (2005). You can't think about thinking without thinking about thinking about something. *Contemporary Issues in Technology and Teacher Education*, *5*(3), 366-367. Retrieved from https://www.learntechlib.org/p/21845/

Piaget, J. (1972). Intellectual evolution from adolescence to adulthood. *Human development*, *15*(1), 1-12. doi:10.1159/000271225

Plass, J.L., Homer, B.D. & Kinzer, C.K. (2015). Foundations of Game-Based Learning. *Educational Psychologist*, *50*(4), 258-283. doi:10.1080/00461520.2015.1122533

Prottsman, K. (2014). Computer Science for the Elementary Classroom. *ACM Inroads*, *5*(4), 60-63. doi:10.1145/2684721.2684735

Román-González, M. (2015). Computational Thinking Test: Design Guidelines and Content Validation. In *Proceedings of EDULEARN15 Conference*, 2436-2444.

Sanders, M. (2009). STEM, STEM Education, STEMAnia, *Education*, *68*(4), 20-27. doi:10.1080/14926156.2016.1166298

Schanzer, E.T. (2015). *Algebraic Functions, Computer Programming, and the Challenge of Transfer*. Doctoral dissertation, Harvard Graduate School of Education. Retrieved from https://dash.harvard.edu/handle/1/16461037

Schildkröten & Krabben. https://crabsturtles.iwm-tuebingen.de/

Scratch for Arduino – S4A (n.d.) http://s4a.cat

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking, *Educational Research Review*, *22*. DOI: https://doi.org/10.1016/j.edurev.2017.09.003

Swidan, A., Hermans, F., & Smit, M. (2018). Programming Misconceptions for School Students. In *2018 International Computing Education Research Conference: ICER '18*, Espoo, Finland. doi:10.1145/3230977.3230995

Teague, D.M., Corney, M.W., Fidge, C.J., Roggenkamp, M.G., Ahadi, A., & Lister, R. (2012). Using neo-Piagetian theory, formative in-Class tests and think alouds to better understand student thinking: a preliminary report on computer programming. In *Proceedings of 2012 Australasian Association for Engineering Education (AAEE) Annual Conference*, 772-780. Retrieved from https://eprints.qut.edu.au/55828/1/Teague2012.pdf

Tsarava, K. (2016). Programming in Greek with Python, Aristotle University of Thessaloniki.

Tsarava, K., Moeller, K., & Ninaus, M. (in press). Board Games for Training Computational Thinking. Proceedings of Games and Learning Alliance conference (GALA 2018) - Lecture Notes in Computer Science. Springer.

Tsarava, K., Moeller, K., & Ninaus, M. (2018). Training Computational Thinking through board games: The case of Crabs & Turtles. International Journal of Serious Games, 5(2), 25-44. https://dx.doi.org/10.17083/ijsg.v5i2.248

Tsarava, K., Moeller, K., Pinkwart, N., Butz, M., Trautwein, U., & Ninaus, M. (2017). Training computational thinking: Game-based unplugged and plugged-in activities in primary school. *Proceedings of the 11th European Conference on Game Based Learning*. Reading, UK: Academic Conferences and Publishing International Limited, 687-695.

Weiß, R.H. (2006). CFT 20-R. Grundintelligenztest Skala 2 - Revision. Göttingen: Hogrefe.

Wendt, H., Bos, W., Selter, C., Köller, O., Schwippert, K., & Kasper, D. (Eds.) (2016). TIMSS 2015 – Mathematische und naturwissenschaftliche Kompetenzen von Grundschulkindern in Deutschland im internationalen Vergleich. Münster: Waxmann.

Wilson, R. A. & Foglia, L. (2017). Embodied Cognition. The Stanford Encyclopedia of Philosophy (Spring 2017 Edition), Edward N. Zalta (ed.), https://plato.stanford.edu/archives/spr2017/entries/embodied-cognition/

Wing, J.M. (2006). Computational Thinking. *Communications of the Association for Computing Machinery (ACM)*, *49*(3), 33-35. doi:10.1145/1118178.1118215

Wing, J.M. (2010). Computational Thinking: What and Why?. *The Link - The Magazine of the Carnegie Mellon University School of Computer Science*, 1-6. Retrieved from https://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why

Wing, J.M. (2014). Computational Thinking Benefits Society. New York: Academic Press. Retrieved from http://socialissues.cs.toronto.edu/index

Wouters, P., van Nimwegen, C., van Oostendorp, H., & van Der Spek, E. D. (2013). A meta-analysis of the cognitive and motivational effects of serious games, *Journal of Educational Psychology*, *105*(2). DOI: https://doi.org/10.1037/a0031311

Yadav, A., Mayfield, C., Zhou, N., Hambrusch, S., & Korb, J.T. (2014). Computational Thinking in Elementary and Secondary Teacher Education. *ACM Transactions on Computing Education*, *14*(1), 1-16. doi:10.1145/2576872

6 ANHANG

Allgemein:

- Definitionsposter zu informatischen Begriffen

Kurssitzung 1:

- Spielanleitung – „Schildkröten & Krabben: Die Schatzsuche“
- Übungsblatt 1
- Zu Hause ausprobieren 1

Kurssitzung 2:

- Spielanleitung – „Schildkröten & Krabben: Die Muster“
- Scratch-Einführung
- Scratch-Projektkarten
- Übungsblatt 2
- Zu Hause ausprobieren 2

Kurssitzung 3:

- Spielanleitung – „Schildkröten & Krabben: Das Wettrennen“
- Übungsblatt 3
- Zu Hause ausprobieren 3

Kurssitzung 4:

- Spiele programmieren
- Übungsblatt 4
- Zu Hause ausprobieren 4

Kurssitzung 5:

- Simulationen
- Übungsblatt 5
- Zu Hause ausprobieren 5

Kurssitzung 6:

- Mathematische Simulationen
- Übungsblatt 6
- Zu Hause ausprobieren 6

Kurssitzung 7:

- Arduino-Einführung
- Arduino-Einführung für die Kursleitung 1
- Arduino-Projekte 1
- Übungsblatt 7
- Zu Hause ausprobieren 7

Kurssitzung 8:

- Arduino-Einführung für die Kursleitung 2
- Arduino-Projekte 2
- Übungsblatt 8
- Zu Hause ausprobieren 8

Kurssitzung 9:

- Roberta-Einführung
- Übungsblatt 9
- Zu Hause ausprobieren 9

Kurssitzung 10:

- Roberta-Projekt
- Übungsblatt 10
- Zu Hause ausprobieren 10

Bedingung

Mit einer Bedingung kann man **überprüfen**, ob etwas **wahr oder falsch** ist. Dann kann man **entscheiden**, was **passieren** soll, wenn die Bedingung **erfüllt** ist und was, wenn sie **nicht erfüllt** ist.

Ereignis

Ein Ereignis ist **etwas, das passiert**.
Auf viele Ereignisse müssen wir **reagieren**. Dazu bestimmen wir, auf welches **Ereignis** welche **Reaktion** folgen soll.

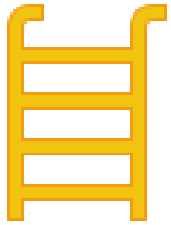
Variable

Eine Variable ist ein **Platzhalter für eine Zahl**, die sich immer wieder ändert.
Um **mit einer Variablen** zu **rechnen**, müsst ihr nachschauen, für **welche Zahl** sie **im Moment** steht.

Konstante

Eine Konstante ist eine **Zahl**, die **immer gleich** bleibt und die man **nicht verändern** kann.

- **Sequenz**:



Eine Sequenz ist eine **Liste von Befehlen**, die in einer bestimmten **Reihenfolge** ausgeführt werden.

- **Schleife**:

Eine Schleife ist eine **Liste von Befehlen**, die mehrmals hintereinander **wiederholt** ausgeführt wird.

- **Muster**:

Ein Muster ist eine **Kombination von Zeichen**, die einer bestimmten Regel folgt.

- Muster können sich **wiederholen**.
- Muster können **Vorbilder** für andere Kombinationen von Zeichen sein.

- **Operator**: + − * /

Ein Operator ist eine **Funktion**, mit der **Zahlen verändert** werden können.

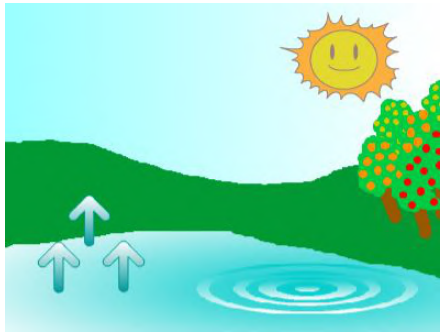
+ Plus	* Mal
− Minus	/ Geteilt

- **Algorithmus**:

Ein Algorithmus ist eine **Liste von Befehlen**, denen man folgen kann, um ein bestimmtes **Ziel** zu **erreichen**.

Ein Algorithmus kann zum Beispiel aus **Sequenzen**, **Schleifen** und **Bedingungen** bestehen.

Simulation:



Simulationen kann man einsetzen, um **Abläufe** und **Vorgänge** nachzuprogrammieren oder um auszuprobieren "**was wäre, wenn**".

Beispiele:
Wetterbericht, Flugsimulator, Autorennen-Computerspiel

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Kurssitzung 1

- Spielanleitung – „Schildkröten & Krabben: Die Schatzsuche“
- Übungsblatt 1
- Zu Hause ausprobieren 1

Schildkröten & Krabben: Die Schatzsuche
Spielanleitung

Inhaltsverzeichnis

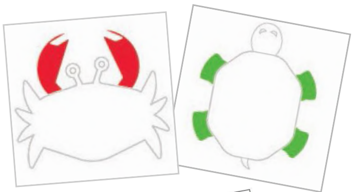
1. Das Spielmaterial

Für “Schildkröten & Krabben: Die Schatzsuche” werden folgende Materialien benötigt:

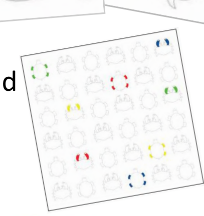
- Das Schatzsuche-**Spielbrett** (auf dem Brett sind 8 Schatztruhen zu sehen)
- Großer **Filzwürfel** mit Augen von 1 bis 6

- Eine große 30-Sekunden-**Sanduhr**

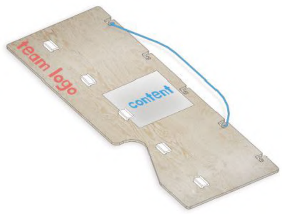
- 8 **Tierfiguren** (4 **Schildkröten**, 4 **Krabben**, jeweils einmal in rot, gelb, grün und blau)



- 8 x 5 **Bewegungskarten** (auf der Rückseite ist das jeweilige Tier abgebildet, zu dem die Bewegungskarten gehören)



- 18 **gemeinsame Bewegungskarten** (auf der Rückseite sind viele kleine Tiere in allen vier Farben abgebildet)



- 8 **Klemmbretter** (am linken oberen Rand des Brettes ist das jeweilige Tier abgebildet, zu dem das Klemmbrett gehört)

- 8 wegwischbare **Filzmarker** (Farbe egal)
- 8 **Schwämme**

- **Abzeichen** (können den Tieren auf den Rücken gesteckt werden):

  - **Sequenz-Abzeichen**: gelbe Leitern
  - **Schleifen-Abzeichen**: runde gelbe Pfeile
  - **Drehungs-Abzeichen**: durchsichtige Räder
  - **Effizienz-Abzeichen**: gelbe Glühbirnen

  - **Zähler**: Holzplättchen mit Zahlen

  - **Futter**: grüne und pinke Pflanzen

2. Der Spielaufbau

Das **Spielbrett** wird so ausgelegt, dass zu allen Seiten etwas Platz ist, damit die Spielenden sich um das Spielbrett bewegen können. Die **gemeinsamen Bewegungskarten** werden gut gemischt und als Stapel verdeckt in die Mitte des Spielfeldes gelegt. Der **Würfel** wird ebenfalls auf das Spielbrett gelegt und die **Sanduhr** wird so aufgestellt, dass alle Spielenden sie gut sehen können.
Auf die vier **Schatztruhen-Felder** in den Ecken des Spielbretts (die übrigen vier Schatztruhen werden ignoriert) wird das **Futter** wie folgt verteilt:

- eine Ecke: zwei gleich aussehende grüne Pflanzen
- diagonal gegenüberliegende Ecke: zwei gleich aussehende pinke Pflanzen
- eine der anderen beiden Ecken: eine grüne Pflanze und eine pinke Pflanze, die jeweils anders aussehen als die Pflanzen, die bereits in den anderen Schatztruhen liegen
- die übrige Ecke: eine grüne Pflanze und eine pinke Pflanze, die jeweils anders aussehen als die Pflanzen, die bereits in den anderen Schatztruhen liegen

3. Die Teams

Die Spielenden bilden **Zweier-Teams**. Falls es eine ungerade Anzahl an Spielenden geben sollte, darf auch ein Dreier-Team gebildet werden.
Jedes Team sucht sich eine **Tierfigur** aus und nimmt:

- Die jeweilige **Tierfigur**
- Das **zugehörige Klemmbrett**
- Die **5 zugehörigen Bewegungskarten**
- Einen **wegwischbaren Filzmarker**
- Einen **Schwamm**

Jedes Team würfelt einmal. Das Team, das die **höchste Augenzahl** gewürfelt hat, **beginnt** mit dem Spiel.

4. Das Spiel erklären!

Die Kursleitung nimmt eine der übrig gebliebenen **Tierfiguren** und die dazugehörigen Materialien (**Klemmbrett, Bewegungskarten, Filzmarker** und **Schwamm**) und nutzt diese, um den Teams das Spiel mittels Veranschaulichungen zu erklären. Dafür setzen sich alle mitspielenden Teams um das Spielbrett. Die Kursleitung erklärt das Spiel:

"In diesem Spiel steuert jedes Team ein Tier (*zeigen*) mithilfe von Befehlen, die zu Sequenzen und Schleifen zusammengesetzt werden können. Alle Tiere sind auf der Suche nach Futter (*zeigen*). Das Team, dessen Tier zuerst drei Pflanzen (davon mindestens eine pinke und eine grüne) ein aus den Schatztruhen (*zeigen*) eingesammelt hat, gewinnt.

Jetzt schaut euch das **Spielbrett** mal genau an. Was für Felder seht ihr?

- Die bunten Felder in der Mitte sind die **Startfelder** (*zeigen*). Dort beginnt das Spiel.
- Die Felder mit den Schatztruhen sind die **Schatzfelder** (*zeigen*). Dort müssen eure Tiere hingehen, um Futter zu finden.
- Die grauen Felder sind **Steinfelder** (*zeigen*).
- Die grünen Felder sind **Grasfelder** (*zeigen*).
- Die blauen Felder sind **Wasserfelder** (*zeigen*).

Alle Tiere starten auf den bunten Feldern in der Mitte (*zeigen*). Die Tiere können von Feld zu Feld gehen und können sich nicht diagonal bewegen (*mit einer Spielfigur veranschaulichen*). Aber Achtung: Die **Schildkröten** sind Landschildkröten und können die Wasserfelder (*zeigen*) nicht betreten! Und die **Krabben** können die Grasfelder (*zeigen*) nicht betreten. Aber alle Tiere können die Startfelder (*zeigen*), die Steinfelder (*zeigen*) und die Schatzfelder (*zeigen*) betreten."

4.1 Wie die Tiere sich bewegen

“Aber **wie bewegen die Tiere sich**? Schaut mal eure fünf **Bewegungskarten** (*zeigen*) an. Ihr seht, dass es fünf verschiedene Karten gibt. Drei von diesen Karten sind bei allen Teams gleich:

- wiederhole 2 mal (*zeigen*)
- drehe ↶ …….° (*zeigen*)
- drehe ↷ …….° (*zeigen*)

Zwei Karten haben nur die **Schildkröten**:

- gehe vorwärts ……. (*zeigen*)
- gehe rückwärts ……. (*zeigen*)

Und zwei Karten haben nur die **Krabben**:

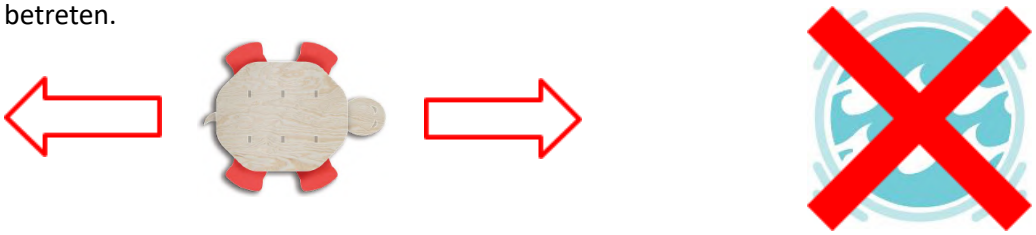
- gehe nach links ……. (*zeigen*)
- gehe nach rechts ……. (*zeigen*)

Das ist deshalb so, weil Krabben normalerweise seitwärts laufen!

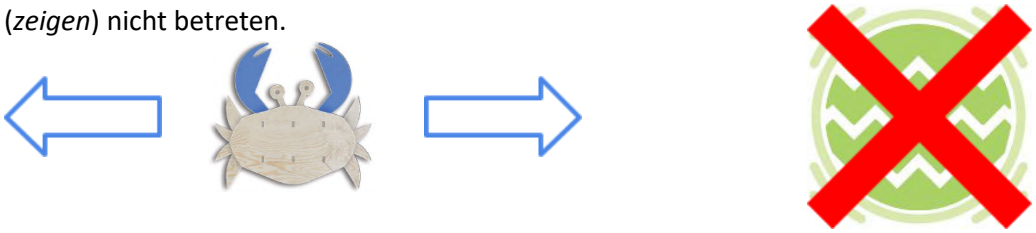
Außerdem zieht am Anfang jedes Team eine Karte vom Stapel mit den **gemeinsamen Bewegungskarten** (*zeigen*) in der Mitte des Spielbretts (*jetzt die Teams ziehen lassen*). Hier gibt es eine **Sonderregel**:

Wenn ein Krabben-Team eine “gehe vorwärts”- oder “gehe rückwärts”-Karte gezogen hat, darf die Krabbe diese Karte einmal benutzen! Umgekehrt ist es genauso. Wenn ein Schildkröten-Team eine “gehe nach links”- oder “gehe nach rechts”-Karte gezogen hat, darf die Schildkröte ihr diese Karte einmal benutzen!

Ihr seht also, es gibt **Unterschiede zwischen den Tieren**. Die **Schildkröten** laufen normalerweise vorwärts (*mit einer Spielfigur veranschaulichen*) oder rückwärts (*mit einer Spielfigur veranschaulichen*) und können die Wasserfelder (*zeigen*) nicht betreten.



Die **Krabben** laufen normalerweise nach links (*mit einer Spielfigur veranschaulichen*) oder nach rechts (*mit einer Spielfigur veranschaulichen*) und können die Grasfelder (*zeigen*) nicht betreten.



Jetzt schauen wir uns mal an, wie die verschiedenen **Bewegungskarten** funktionieren. Wenn euer Team dran ist, plant ihr euren Spielzug. Dafür steckt ihr die Karten, die ihr benutzen wollt, von oben nach unten an eurer **Klemmbrett**. Der Spielzug beginnt mit der obersten Karte, die ihr gewählt habt, und endet mit der untersten Karte. Ihr könnte jede Karte nur einmal benutzen, weil ihr jede Karte nur einmal habt! Ich erkläre euch jetzt, wie die verschiedenen Karten funktionieren:

- Die Karten "**gehe vorwärts** …….", "**gehe rückwärts** …….", "**gehe nach links** ……." und "**gehe nach rechts** ……." sind ganz einfach. Auf die **gestrichelte Linie** schreibt ihr, wieviele Felder euer Tier laufen soll. Zum Beispiel "gehe nach links 3" (*Krabbe nehmen und auf dem Spielbrett zeigen: die Krabbe geht 3 Felder nach links*).

- Seht ihr den Unterschied zwischen den beiden Karten “**drehe ↶ …….°**” und “**drehe ↷ …….°**”? Genau! Einmal sagt der **Pfeil**, dass euer Tier sich **nach links drehen** muss und einmal sagt der **Pfeil**, dass euer Tier sich **nach rechts drehen** muss (*mit einer Spielfigur veranschaulichen*).
  Bei diesen beiden Karten müsst ihr auf die **gestrichelte Linie** schreiben, wie weit sich euer Tier drehen soll. Dafür benutzen wir drei bestimmte Zahlen. Diese Zahlen sind wichtig, weil ihr sie später wieder brauchen werdet, wenn wir an den Computern programmieren. Die drei Zahlen sind **90, 180** und **270** (*an die Tafel oder groß auf ein Blatt Papier schreiben*).
  Ich zeige euch jetzt, wie das funktioniert:
  - Wenn euer Tier sich um **90° nach links** dreht, sieht das so aus (*mit einer Spielfigur veranschaulichen*).
  - Wenn euer Tier sich um **180° nach links** dreht, sieht das so aus (*mit einer Spielfigur veranschaulichen*).
  - Wenn euer Tier sich um **270° nach links** dreht, sieht das so aus (*mit einer Spielfigur veranschaulichen*).
  - … nach dem gleichen Schema Drehungen **nach rechts** veranschaulichen.

  Seht ihr, dass euer Tier nach der Drehung in die gleiche Richtung schaut, wenn ihr es um **180° nach links** dreht, wie wenn ihr es um **180° nach rechts** dreht? (*nochmal mit einer Spielfigur veranschaulichen*)
  Seht ihr auch, dass euer Tier nach der Drehung in die gleiche Richtung schaut, wenn ihr es um **90° nach links** dreht, wie wenn ihr es um **270° nach rechts** dreht? (*nochmal mit einer Spielfigur veranschaulichen*)
  Seht ihr, dass euer Tier nach der Drehung in die gleiche Richtung schaut, wenn ihr es um **90° nach rechts** dreht, wie wenn ihr es um **270° nach links** dreht? (*nochmal mit einer Spielfigur veranschaulichen*)”

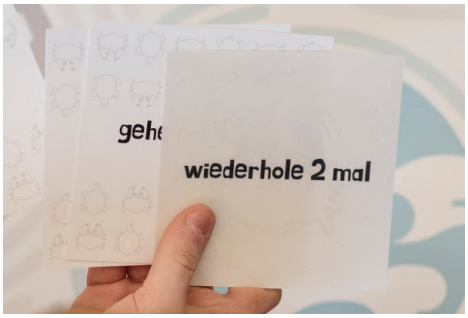
4.2 Beispiel-Spielzug mit Sequenz

"Ich zeige euch mal ein Beispiel, wie ein **Spielzug** aussehen kann. (*Ein Tier zum Zeigen auf eines der bunten Felder in der Mitte des Spielfelds setzen.*) Ich möchte **mit diesem Tier eine Schatztruhe erreichen**. Wie komme ich zu einer Schatztruhe? Indem ich dem Tier eine **Sequenz von Befehlen** gebe.

(*Eine Sequenz von Karten an das Klemmbrett klemmen, mit der das Tier zur Schatztruhe kommt. Dabei jeweils erklären, weshalb die jeweilige Karte ausgewählt wurde. Jeweils die benötigte Anzahl Schritte bzw. die benötigte Gradzahl für die Drehung auf die Karte schreiben.*)

So, ich glaube, so kann es funktionieren. Probieren wir das mal aus. **Ich führe meine Sequenz aus!** (*Mit dem Tier die Sequenz ausführen.*) Habt ihr das verstanden? (*Bei Bedarf Fragen beantworten und Teile des Spielzugs nochmal erklären.*)"

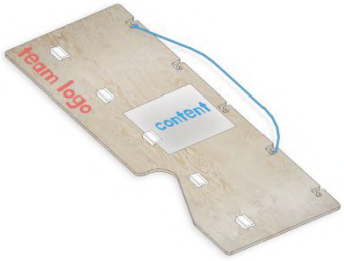
4.3 Beispiel-Spielzug mit Schleife



"Habt ihr bemerkt, dass ich eine Karte noch gar nicht erklärt habe? Welche Karte ist das? Genau, "**wiederhole 2 mal**" (*zeigen*). Diese Karte ist besonders und funktioniert ein bisschen anders als die anderen Karten.

Mit der "wiederhole 2 mal"-Karte könnt ihr, wie der Name sagt, andere **Befehle wiederholt ausführen**. Das heißt, mit dieser Karte kann ich eine **Schleife** erzeugen. Ich zeige euch mal, wie das funktioniert. (*Die "wiederhole 2 mal"-Karte ganz oben an das Klemmbrett klemmen. Darunter andere Befehle klemmen, z.B. "gehe vorwärts 3" und "drehe ↶90°".*) So kann ich die Befehle, die unter der "wiederhole 2 mal"-Karte stehen, zweimal hintereinander ausführen!

Aber was passiert, wenn ich danach **noch einen Befehl ausführen** möchte, der aber **nicht Teil der Schleife** sein soll? (*Einen weiteren Befehl, z.B. "drehe ↷90°" unter die Schleife klemmen.*) Dafür ist die Schnur da. Immer, wenn ich die "wiederhole 2 mal"-Karte benutze, muss ich auch eine **Schnur** benutzen, um zu zeigen, **wo die Schleife aufhört**. Dafür klemme ich das eine Ende der Schnur an die gleiche Klammer wie die "wiederhole 2 mal"-Karte und das andere Ende der Schnur an die gleiche Klammer wie die unterste Karte, die noch zu meiner Schleife gehören soll (*veranschaulichen*).

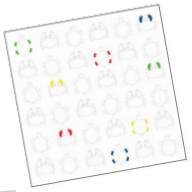


Ich glaube, so kann es funktionieren. Probieren wir das mal aus. **Ausführen!** (*Mit dem Tier die Schleife und den nachfolgenden Befehl ausführen.*) Habt ihr das verstanden? (*Bei Bedarf Fragen beantworten und Teile des Spielzugs nochmal erklären.*)"

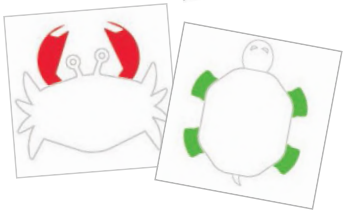
4.4 Spielzug-Zusammenfassung

"Wenn euer Team an der Reihe ist, wird die **Sanduhr** umgedreht (und wenn sie abgelaufen ist noch ein weiteres Mal umgedreht), sodass ihr **eine Minute** Zeit habt, euren Zug fertig zu planen. Sobald ihr fertig geplant habt und "**ausführen**" sagt, dürft an den Karten auf eurem Klemmbrett nichts mehr verändern! Wenn die Sanduhr abgelaufen ist, bevor ihr "**ausführen**" sagt, müsst ihr die Befehle auf eurem Klemmbrett so ausführen, wie sie gerade dastehen.

Falls ihr in eurem Zug einen **Fehler** macht und ein Befehl nicht ausgeführt werden kann, führt ihr den Zug solange aus, bis ihr zum **falschen Befehl** kommt. Dann müsst ihr aufhören. Wenn ihr die Befehle auf eurem Klemmbrett ausgeführt habt oder ein Fehler aufgetreten ist, ist euer **Spielzug zu Ende** und das nächste Team ist an der Reihe. In jedem Zug startet ihr auf dem Feld, auf dem ihr euren vorigen Zug beendet habt.



Am Ende eures Spielzuges legt ihr die Karte, die ihr vom gemeinsamen **Bewegungskartenstapel** (*zeigen*) in der Mitte genommen habt, zurück unter den Stapel und **zieht eine neue Karte**.



Eure eigenen Bewegungskarten , auf deren Rückseite euer Tier aufgedruckt ist (*zeigen*), dürft ihr **immer behalten** und in jedem Zug benutzen.

Denkt daran, als Team zusammenzuarbeiten!"

4.5 Abzeichen sammeln

“Abgesehen von den Pflanzen können eure Tiere noch weitere Dinge sammeln! Wenn ihr im Spiel bestimmte Sachen richtig macht, bekommt ihr dafür folgende Belohnungen:

Sequenz-Abzeichen (gelbe Leiter): wenn ihr eine Sequenz mit drei oder mehr Befehlen erfolgreich ausführt (kann man nur einmal pro Zug bekommen)

Schleifen-Abzeichen (runder gelber Pfeil): wenn ihr eine Schleife erfolgreich ausführt (kann man nur einmal pro Zug bekommen)

Drehungs-Abzeichen (hölzernes Rad mit Gradzahlen): wenn ihr eine Drehung erfolgreich ausführt (mehrere erfolgreiche Drehungen in einem Zug werden auch mit mehreren Abzeichen belohnt!)

Effizienz-Abzeichen (gelbe Glühbirne): wenn ihr alle fünf Plätze des Klemmbrettes mit Befehlen füllt, sie alle erfolgreich ausführt und alle Befehle **nützlich** sind (kann man nur einmal pro Zug bekommen)

Wenn ihr eines dieser Abzeichen schon habt, bekommt ihr es danach nicht nochmal, sondern bekommt stattdessen einen kleinen **Zähler**, den ihr auf das Abzeichen setzen könnt, um zu zählen, wie oft ihr es schon bekommen habt.”

4.6 Futter sammeln

"Wenn ihr mit eurem Tier ein **Schatzfeld** betretet, dürft ihr euch **eine der beiden Pflanzen** nehmen, die dort liegen, und ihn eurem Tier auf den Rücken setzen. Denkt daran, dass ihr zwei **verschiedenfarbige** Pflanzen sammeln müsst!

Aber Achtung: Ihr dürft aus jeder Schatztruhe nur einmal eine Pflanze nehmen! Auch wenn ihr später noch einmal zu derselben Schatztruhe geht, dürft ihr dort keine Pflanze mehr nehmen. Ihr müsst mit eurem Tier also mindestens zu zwei unterschiedlichen Schatztruhen laufen."

4.7 Ende des Spiels

"Wenn ein Team **mindestens drei Pflanzen aus drei unterschiedlichen Schatztruhen** (davon mindestens eine pinke und eine grüne Pflanze) gesammelt hat, wird die aktuelle **Spielrunde noch fertig gespielt**, damit alle Teams gleich oft dran waren. **Dann ist das Spiel zu Ende.**

Wenn **nur ein Team** drei Pflanzen gesammelt hat, hat dieses Team **gewonnen**. Wenn **mehrere Teams** die drei Pflanzen gesammelt haben, **gewinnt von diesen Teams** dasjenige, das **die meisten Punkte für Abzeichen** gesammelt hat."

5. Spielvarianten für verkürzte oder verlängerte Spielzeit

5.1 Verkürzte Spielzeit

Um die Spielzeit zu verkürzen, kann bis zu zwei verschiedenfarbigen gesammelten Futterpflanzen gespielt werden anstatt bis zu dreien. Alternativ oder zusätzlich können auf allen vier Schatztruhen in den Ecken des Spielbretts je pinke und grüne Futterpflanzen ausgelegt werden.
Außerdem können Futterpflanzen auf alle acht Schatztruhen-Felder verteilt werden, sodass sie auf kürzeren Wegen erreicht und dadurch schneller gesammelt werden können.

5.2 Verlängerte Spielzeit

Um die Spielzeit zu verlängern, kann die Anzahl der Futterpflanzen, die gesammelt werden muss, erhöht werden.

Name: _____

Übungsblatt 1

1. Die Schatzsuche

Wieviele Punkte hat dein Team im Spiel “Schildkröten & Krabben: Die Schatzsuche” bekommen?

Für **Futter:** _____ Punkte

Für **Drehungen:** _____ Punkte

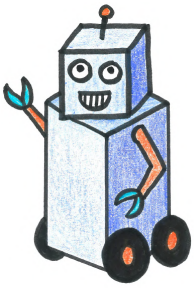
Für **Sequenzen:** _____ Punkte

Für **Schleifen:** _____ Punkte

Für **Effizienz**: _____ Punkte

2. Dein Roboter

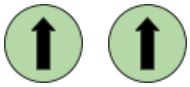
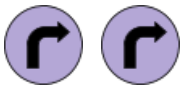
Du hast einen kleinen Roboter mit zwei Knöpfen:



Wenn du diesen Knopf drückst, fährt dein Roboter ein Stück geradeaus.

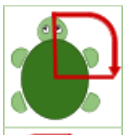
Wenn du diesen Knopf drückst, dreht sich dein Roboter auf der Stelle um 90° nach rechts.

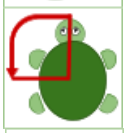
Du willst, dass dein Roboter sich so bewegt, dass er am Ende um 90° nach links gedreht ist. **Kreuze an, welche Knöpfe du dafür drücken musst!**

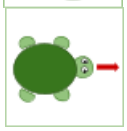
A) ☐		B) ☐	
C) ☐		D) ☐	

3. Der Schildkrötenroboter

Du hast einen Schildkrötenroboter zum Geburstag bekommen, der folgende einfache Befehle ausführen kann:

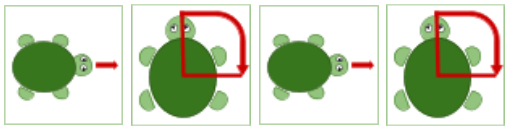
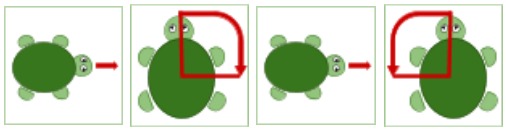
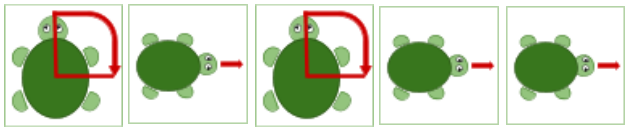
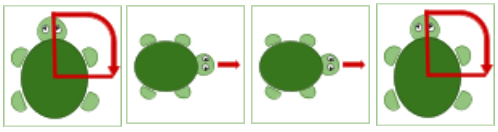
Drehe dich um 90 Grad nach rechts 

Drehe dich um 90 Grad nach links 

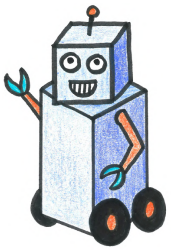
Fahre 30 Zentimeter vorwärts 

Der Schildkrötenroboter ist so eingestellt, dass er eine ihm gegebene Folge von Befehlen solange wiederholt, bis man ihn ausschaltet.

Welche Folge von Befehlen lässt den Schildkötenroboter ein Quadrat fahren?

A) ☐	
B) ☐	
C) ☐	
D) ☐	

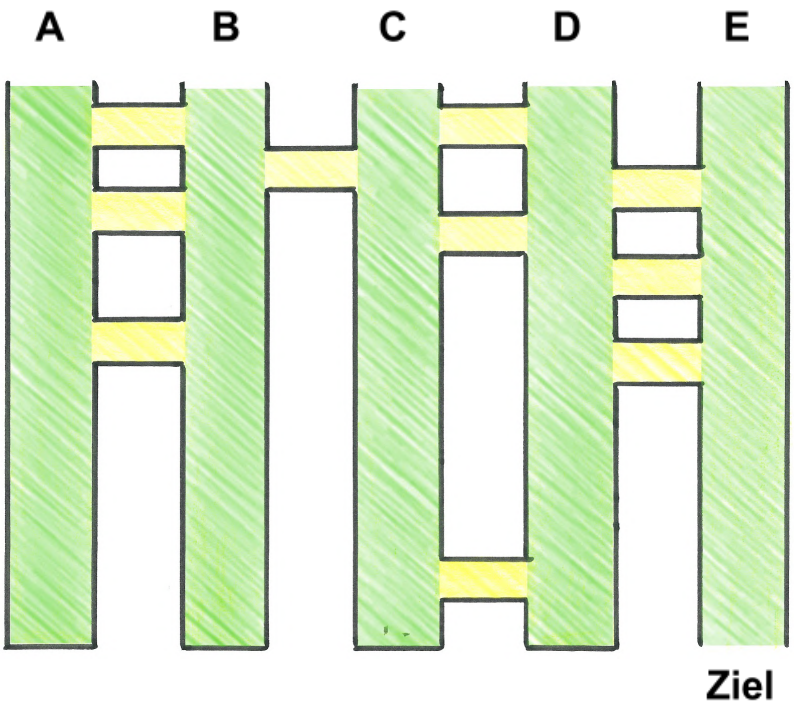
4. Das Labyrinth



Dein Roboter ist wie folgt programmiert:

- Wiederhole die folgenden beiden Befehle, bis du am Ziel angekommen bist:
  - Fahre den grünen Gang entlang nach unten, bis ein gelber Quergang kommt.
  - Fahre durch den gelben Quergang.

In welchen Gang muss dein Roboter hineinfahren, damit er zum Ausgang findet?



Deine Antwort: _____

Aufgabe 1, 2 und 3 auf diesem Übungsblatt basieren auf Aufgaben aus dem Informatik-Schülerwettbewerb "Informatik-Biber" der Initiative "Bundesweit Informatik fördern", https://bwinf.de/biber/downloads/
Wir empfehlen die auf dieser Webseite zusammengestellten Materialien als Knobelaufgaben für eine vertiefende Beschäftigung mit Problemstellungen der Informatik.

Lösung zu Übungsblatt 1

2. Dein Roboter

Antwort C

3. Der Schildkrötenroboter

Antwort A

4. Das Labyrinth

Antwort C

Zu Hause ausprobieren:

Sequenzen suchen

Sei ganz aufmerksam: Suche in deinem Alltag nach fünf Abläufen und Sachen, die wie eine **Sequenz** funktionieren!
Für jede richtige Sequenz gibt es einen Punkt:

Beispiel: *Wenn ich den Weg zu meiner Schule beschreibe*

1. _____
2. _____
3. _____
4. _____
5. _____

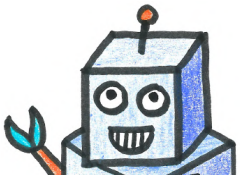
Schleifen suchen

Suche in deinem Alltag außerdem nach fünf Abläufen und Sachen, die wie eine **Schleife** funktionieren!
Für jede richtige Schleife gibt es einen Punkt:

Beispiel: *Die Bewegung der Zeiger auf einer Uhr*

1. _____
2. _____
3. _____
4. _____
5. _____

Viel Spaß beim Suchen!



EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Kurssitzung 2

- Spielanleitung – „Schildkröten & Krabben: Die Muster“
- Scratch-Einführung
- Scratch-Projektkarten
- Übungsblatt 2
- Zu Hause ausprobieren 2

Schildkröten & Krabben: Die Muster

Spielanleitung

Inhaltsverzeichnis

1. Das Spielmaterial

Für “Schildkröten & Krabben: Die Muster” werden folgende Materialien benötigt:

- 24 **längliche Musterkarten** (mit **eingekreistem** oberen Symbol)
- 24 **längliche Musterkarten** (ohne eingekreiste Symbole) für **Phase 1**
- 24 **quadratrische Musterkarten** für **Phase 2**

2. Der Spielaufbau

Die 24 **länglichen Musterkarten** (mit eingekreistem oberen Symbol) werden **zufällig verteilt auf dem Boden** ausgelegt. Es ist nicht wichtig, dass sie auf eine bestimmte Weise angeordnet oder ordentlich verteilt sind. Es sollte aber darauf geachtet werden, dass sie alle **gleich ausgerichtet** sind, d.h. dass **bei allen das eingekreiste Symbol oben** ist, wenn man von der Seite, auf der die Spielerinnen und Spieler stehen werden, auf die Karten schaut.

Die 24 **quadratrische Musterkarten** werden **verdeckt auf einen Stapel** gelegt, sodass die Spielerinnen und Spieler die Vorderseiten der Karten nicht sehen können.

Die Kursleitung nimmt die 24 **länglichen Musterkarten** (ohne eingekreiste Symbole) **als Stapel in die Hand** und hält sie so, dass die Spielerinnen und Spieler die Vorderseiten der Karten nicht sehen können.

3. Das Spiel erklären!

“Stellt euch bitte so auf, dass das **eingekreiste Symbol auf den Karten für euch oben** ist. Steht **alle nebeneinander** und achtet darauf, ob ihr die Karten auf dem Boden **gut sehen** könnt. In diesem Spiel geht es darum, wie schnell ihr **Muster erkennen** könnt. Jede Karte ist einen Punkt wert. Wer die **meisten Karten** richtig rät, **gewinnt** das Spiel.

Das Spiel hat **zwei Phasen** und in beiden Phasen gibt es **gleich viele Karten**. Fangen wir mit **Phase 1** an. Danach erkläre ich euch **Phase 2**.”

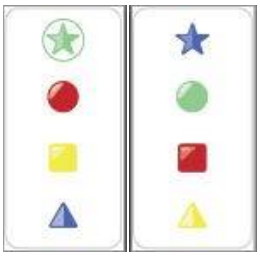
3.1 Phase 1

Zum Erklären am besten eine der **länglichen Musterkarten** (ohne eingekreiste Symbole) auswählen, auf der **vier unterschiedliche Farben** zu sehen sind. Die Karte offen zeigen.

"Zu jeder Karte in diesem Stapel (*Stapel mit länglichen Karten in der Hand zeigen*) gibt es eine passende Karte, die schon auf dem Boden liegt (*zeigen*). Immer wenn ich euch eine Karte zeige, müsst ihr auf dem Boden die Karte finden, die dazu passt. **Die Karten, die zusammen gehören, sehen nicht genau gleich aus**! Aber es gibt **drei Regeln**, mit denen ihr erkennen könnt, ob zwei Karten zusammen gehören:

1. **Auf beiden Karten kommen die gleichen Farben vor**. (Auf beiden Karte kommen alle Farben vor, die auch auf der anderen vorkommen. Auf keiner Karte kommen Farben vor, die auf der anderen nicht vorkommen.)
2. Auf beiden Karten sind **die gleichen Symbole** zu sehen **in der gleichen Reihenfolge von oben nach unten**. (Bei allen Karten auf dem Boden ist das oberste Symbol eingekreist.)
3. **Kein Symbol hat auf beiden Karten dieselbe Farbe**.

Welche Karte auf dem Boden gehört also zu der Karte, die ich euch gerade zeige?"
(*Kinder antworten lassen*)



"**Seht ihr, wieso die zwei Karten zusammengehören**? Sie haben beide die **gleichen Farben**. Sie haben beide die **gleichen Symbole in der gleichen Reihenfolge von oben nach unten**. Und **kein Symbol hat auf beiden Karten dieselbe Farbe**." (*Bei Bedarf nochmal erklären*)

“Ich zeige euch jetzt **immer eine Karte** von diesem Stapel (*Stapel in der Hand zeigen*). Sobald ihr seht, welche Karte dazu gehört, **meldet ihr euch** (*vormachen*). Ich rufe dann denjenigen oder diejenige auf, der oder die sich zuerst gemeldet hat. Dann darf der- oder diejenige **sagen, welches die richtige Karte ist**! Wenn ihr **falsch ratet oder zu lange braucht** (*nach subjektivem Ermessen der Kursleitung*), lege ich die Karte wieder **unter den Stapel**. Wenn ihr **richtig rate**t, dürft ihr die **Karte behalten**! **Seid ihr bereit**?”

Beim Spielen darauf achten, dass die Karte, die gezeigt wird, immer **richtig herum** gehalten wird, also **mit dem oberen Ende nach oben**. Wenn ein Kartenpaar richtig erraten wird, darf das Kind **die Karte behalten**, die **vom Handkartenstapel** gezeigt wurde. **Die Karten auf dem Boden bleiben immer liegen**!

Wenn eine Karte **falsch geraten** wurde oder das Kind **zu lange gebraucht** hat, wird die Handkarte **zurück unter den Handkartenstapel** gelegt. Diese Karten kommen dadurch später im Spiel automatisch nochmal an die Reihe und es kann **erneut geraten** werden.

Während der ersten paar Spielzüge: Wenn ein Kind geraten hat, die **Handkarte neben die geratene Karte auf dem Boden** legen und **Symbol für Symbol durchgehen, ob die drei Regeln erfüllt sind**.

Wenn alle Karten erraten wurden: “**Ihr habt alle Karten aus Phase 1 erraten**! **Seid ihr bereit für Phase 2**?”

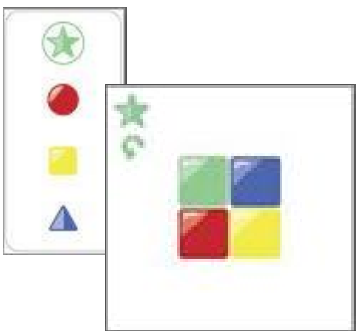
3.2 Phase 2

Zum Erklären am besten eine der **quadratischen Musterkarten** auswählen, auf der **vier unterschiedliche Farben** zu sehen sind.

"Zu jeder Karte in diesem Stapel (*Stapel mit quadratischen Karten in der Hand zeigen*) gibt es eine passende Karte, die schon auf dem Boden liegt (*zeigen*). Immer wenn ich euch eine Karte zeige, müsst ihr auf dem Boden die Karte finden, die dazu passt. Es gibt diesmal **vier Regeln**, mit denen ihr erkennen könnt, ob zwei Karten zusammen gehören:

1. **Auf beiden Karten kommen die gleichen Farben vor**.
2. Das **Symbol in der Ecke der quadratischen Karte** hat die **gleiche Form und Farbe** wie das **eingekreiste Symbol auf der schmalen Karte**.
3. Auf beiden Karten sind **die gleichen Farben** zu sehen **in der gleichen Reihenfolge**. Aber: Diesmal müsst ihr die Reihenfolge **auf der schmalen Karte von oben nach unten** beachten und **auf der quadratischen Karte im Kreis herum**!
4. Der **Pfeil auf der quadratischen Karte** gibt an, in welche **Kreisrichtung** ihr die quadratische Karte "lesen" müsst.

Welche Karte auf dem Boden gehört also zu der Karte, die ich euch gerade zeige?"
(*Kinder antworten lassen*)



"**Seht ihr, wieso die zwei Karten zusammengehören**? Sie haben beide die **gleichen Farben**. Sie haben beide das **gleiche Symbol eingekreist bzw. in der oberen Ecke**. Und **die Farbe haben dieselbe Reihenfolge, wenn ihr sie von oben nach unten bzw. in Preilrichtung vergleicht**." (*Bei Bedarf nochmal erklären*)

“Die Spielregeln sind genauso wie in **Phase 1**. Ich zeige euch **immer eine Karte**. Sobald ihr seht, welche Karte dazu gehört, **meldet ihr euch** (*vormachen*). **Seid ihr bereit**?”

Beim Spielen darauf achten, dass die Karte, die gezeigt wird, immer **richtig herum** gehalten wird, also **mit dem Symbol in der linken oberen Ecke** (*aus Perspektive der Kinder*). Wenn ein Kartenpaar richtig erraten wird, darf das Kind **die Karte behalten**, die **vom Handkartenstapel** gezeigt wurde. **Die Karten auf dem Boden bleiben immer liegen**!

Wenn eine Karte **falsch geraten** wurde oder das Kind **zu lange gebraucht** hat, wird die Handkarte **zurück unter den Handkartenstapel** gelegt. Diese Karten kommen dadurch später im Spiel automatisch nochmal an die Reihe und es kann **erneut geraten** werden.

Während der ersten paar Spielzüge: Wenn ein Kind geraten hat, die **Handkarte neben die geratene Karte auf dem Boden** legen und **Symbol für Symbol durchgehen, ob die vier Regeln erfüllt sind**.

3.3 Ende des Spiels

“Zählt jetzt, **wieviele Karten** ihr gesammelt habt. Jede Karte ist einen Punkt wert. **Wer hat die meisten Punkte bekommen**?”

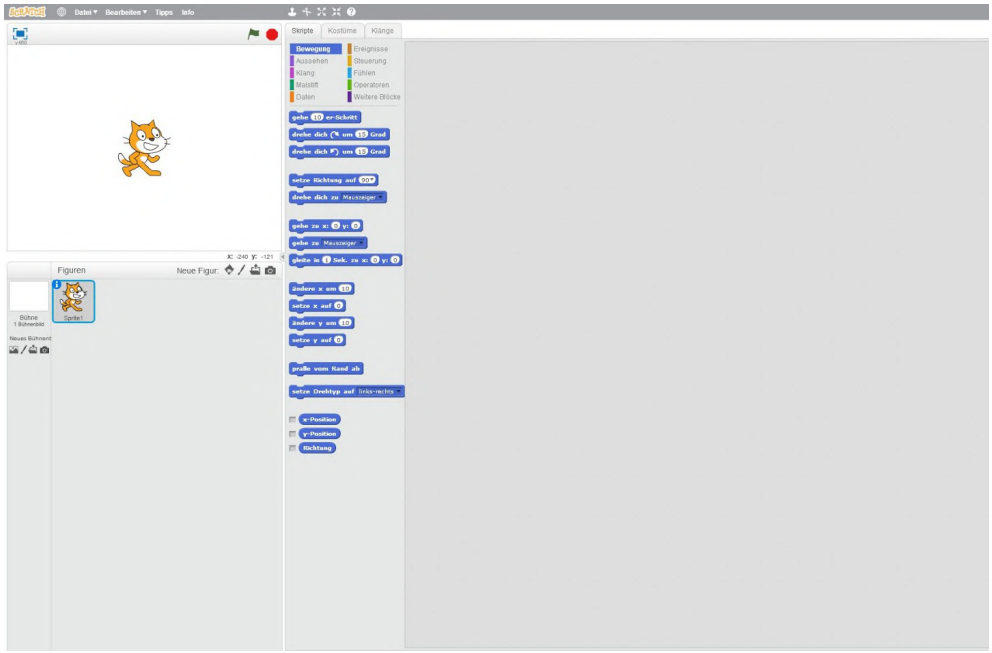
4. Spielvariante für verkürzte Spielzeit

Um die Spielzeit zu verkürzen, kann mit einer geringeren Anzahl Karten gespielt werden. Beim Verringern der Kartenanzahl während der Spielvorbereitung ist zu beachten, dass Karten immer zusammen mit den zu ihnen gehörenden Karten (jeweils eine längliche mit eingekreistem oberen Symbol, eine ohne eingekreistem oberen Symbol und eine quadratische) entfernt werden sollten.
Um zugleich die Schwierigkeit des Spiels zu senken, können einfachere Karten (mit nur zwei Farben) im Spiel behalten werden, während ein Teil der schwierigeren Karten (mit allen vier Farben) aussortiert werden kann.

Scratch-Umgebung

Die Scratch-Umgebung besteht aus drei Fenstern:

- Das **Block-Programmier**-Fenster (rechts)



Hier programmieren wir unser Skript. Dafür benutzen wir bunte Blöcke.

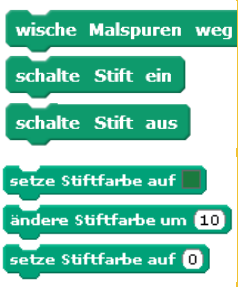
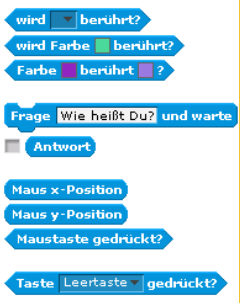
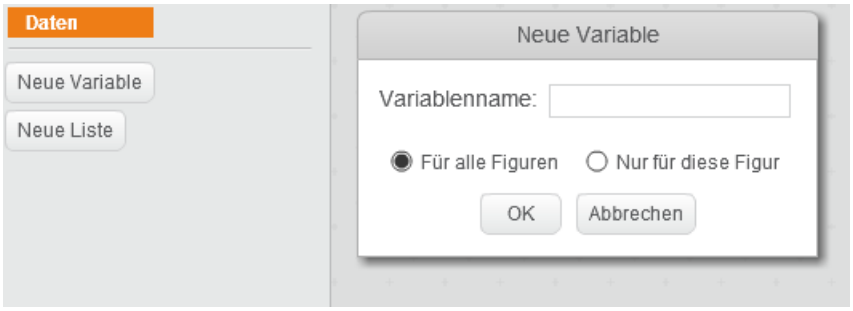
- Das **Objekt-**Fenster (unten links)

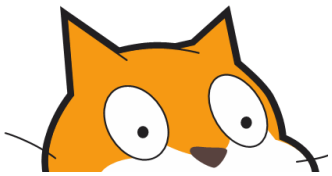
Hier erzeugen wir neue Objekte und bearbeiten den Hintergrund unseres Programms.

- Und das **Vorschau-**Fenster (oben links)

Hier testen wir unseren Code und überprüfen, wie das Programm aussieht.

Nachdem du im Objekt-Fenster ein Objekt erzeugt hast, kannst du es programmieren, damit es bestimmte Sachen macht. Dafür musst du verschiedene Befehls-Blöcke kombinieren. Es gibt verschiedene Arten von Blöcken und Befehlen, die wie folgt in verschiedene Farben eingeteilt sind:

Bewegung	Aussehen	Klang	Malstift
			
Ereignisse	**Steuerung**	**Fühlen**	**Operatoren**
			
Daten			
			



Lasst uns anfangen!

Probiert die folgenden Schritte aus. Vergesst nicht, als Team zu arbeiten!

Öffnet Scratch...
ODER geht ins **Internet** und schreibt folgende Adresse in die **URL-Zeile**:

https://scratch.mit.edu/projects/editor

Dann drückt die **Enter-Taste**.

- Löscht die Scratch-Katze.
- Erzeugt eine **neue Figur aus der Bibliothek**. Wenn ihr ein Team seid, müsst ihr beide eine eigene Figur erzeugen!
- Sucht euch einen **Hintergrund** aus.
- Klickt eine der Figuren an und **öffnet das Skript**.
- Zieht folgenden Block aus **Ereignisse** in euer Skript:

  Das ist ein **Ereignis-Block**. Wenn ihr die grüne Flagge über dem Vorschau-Fenster anklickt, werden alle an diesem Block hängenden Befehle Schritt für Schritt ausgeführt.

- Benutzt Blöcke aus **Bewegung** und **Klang**, um euch vorzustellen, indem ihr eure **Figuren sprechen** lasst.
- Probiert aus, wie ihr **Blöcke aus eurem Skript löschen** könnt. Zieht sie einfach aus dem Skript raus!
- Ihr könnt euer **Programm ausführen**, indem ihr **auf die grüne Flagge klickt**!
- **Wenn ihr mit dem Skript fertig seid, macht ein Skript für die andere Figur.**
- Führt den Code aus! Gibt es Fehler oder Sachen, die ihr ändern wollt? **Experimentiert damit!**

Jetzt nehmt euch eine **Projektkarte** und fangt an zu programmieren!

Die auf diesem Arbeitsblatt verwendeten Bilder sind Screenshots der Webseite: https://scratch.mit.edu/projects/editor/
Wir empfehlen diese Webseite für vertiefende Übungen im Programmieren in Scratch.

Gründe eine Band!

Baue dein eigenes musikalisches Scratch-Projekt, indem du Figuren mit Klängen verknüpfst, um interaktive Musikinstrumente zu bauen.

Anleitung

- Füge Figuren ein.
- Suche einen passenden Hintergrund aus.
- Füge Klangblöcke hinzu und mach deine Instrumente interaktiv.
- Lasse deine Bandmitglieder tanzen, während sie singen.

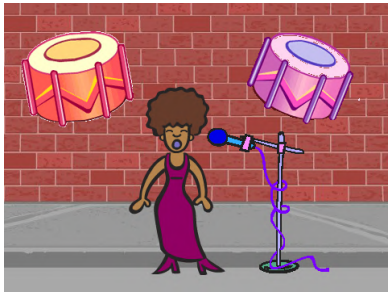
Arten von Blöcken

- Ereignisse
- Steuerung
- Bewegung
- Klang

Tipps

- Benutze “wiederhole”-Blöcke, um einen Klang mehr als einmal abzuspielen.
- Probiere die “Tempo”-Blöcke aus, um den Rhythmus schneller oder langsamer zu machen.
- Du kannst mehrere Leute singen oder Musik machen lassen!
- Du darfst auch andere Befehle benutzen.
- Experimentiere mit deinem Code!

Hintergrund



Figuren

Befehle

Bring Bewegung rein!

Bringe Figuren, Bilder und Ideen als Animationen zum Leben, indem du eine Reihe von Kostüm- änderungen programmierst.

Anleitung

- Such dir eine Figur aus.
- Füge ein anderes Kostüm hinzu.
- Füge Blöcke hinzu, damit das Bild lebendig wird.
- Wiederhole.

Arten von Blöcken

- Ereignisse
- Steuerung
- Bewegung
- Aussehen

Tipps

- Probiere verschiedene Figuren und Kostüme aus, bis du etwas findest, das dir gefällt.
- Benutze “wiederhole”-Blöcke, damit deine Figur sich bewegt.
- Du kannst Musik zum Hintergrund hinzufügen!
- Du darfst auch andere Befehle benutzen.
- Experimentiere mit deinem Code!

Hintergrund

Figuren

Befehle

Baue eine interaktive Wand!

Probiere deine Figuren so zu programmieren, dass sie verschiedene Sachen tun! Experimentiere mit Klängen, Sprechen und Reaktionen!

Anleitung

- Such einen Hintegrund aus, der deine interaktive Wand wird.
- Such ein paar Figuren aus, um deine Wand zu dekorieren.
- Füge Blöcke hinzu, damit die Figuren auf Anklicken reagieren.
- Füge Blöcke hinzu, um sie auf verschiedene Arten reagieren zu lassen.

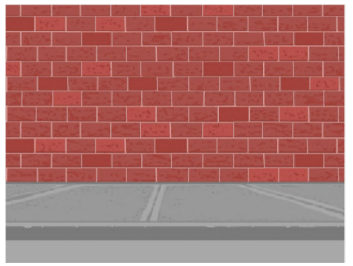
Arten von Blöcken

- Ereignisse
- Steuerung
- Klang
- Aussehen

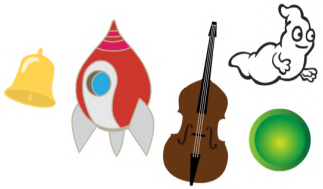
Tipps

- Experimentiere mit verschiedenen Hintergründen und Figuren, bis du etwas findest, das dir gefällt.
- Programmiere jede Figur einzeln! Du kannst Klänge wiederholen, um die Reaktion länger dauern zu lassen.
- Du darfst auch andere Befehle benutzen.
- Experimentiere mit deinem Code!

Hintergrund



Figuren



Befehle

Lass sie sprechen!

Probiere verschiedene Arten aus, deine Figuren dazu zu zu programmieren, miteinander zu sprechen! Experimentiere mit Bewegungen und Zeiteinteilung!

Anleitung

- Such dir ein paar Figuren aus.
- Such dir einen Hintergrund aus, der dir gefällt.
- Füge Blöcke hinzu, mit denen die Figuren sich aufeinander zu bewegen.
- Füge Blöcke hinzu, um die Figuren sprechen zu lassen.

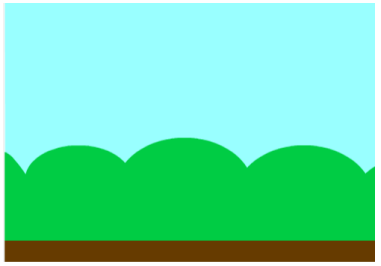
Arten von Blöcken

- Ereignisse
- Steuerung
- Bewegung
- Aussehen

Tipps

- Experimentiere mit verschiedenen Hintergründen und Figuren, bis du etwas findest, das dir gefällt.
- Programmiere jede Figur einzeln und achte auf die Zeiteinteilung! Die Figuren sollen nicht gleichzeitig sprechen.
- Du darfst auch andere Befehle benutzen.
- Experimentiere mit deinem Code!

Hintergrund



Figuren

Befehle

Bewegung wie im Videospiel!

Probiere, die Figur wie in einem Videospiel mit den Pfeiltasten auf der Tastatur zu bewegen! Experimentiere mit Bewegungen und Ereignissen!

Anleitung

- Suche dir eine Figur aus, die man von oben sieht.
- Such dir einen Hintergrund aus, der dir gefällt.
- Kombiniere Blöcke so, dass die Figur nach oben geht, wenn der “oben”-Pfeil gedrückt wird, nach unten, wenn der “unten”-Pfeil gedrückt wird, nach links, wenn der “links”-Pfeil gedrückt wird und nach rechts, wenn der “rechts”-Pfeil gedrückt wird.

Beispiel:

Arten von Blöcken

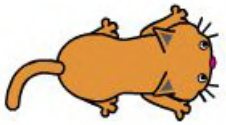
- Ereignisse
- Bewegung

Tipps

- Du kannst auch Bewegungen für andere Tasten hinzufügen.
- Füge noch weitere Figuren mit eigener Tastensteuerung hinzu!
- Experimentiere mit deinem Code!

Hintergrund

Figuren



Befehle

Die Aufgaben “Gründe eine Band” und “Bring Bewegung rein” basieren auf Lernressourcen von https://scratch.mit.edu/
Die auf diesem Arbeitsblatt verwendeten Bilder sind Screenshots der Webseite: https://scratch.mit.edu/projects/editor/
Wir empfehlen diese Webseite für vertiefende Übungen im Programmieren in Scratch.

Name: _____

Übungsblatt 2

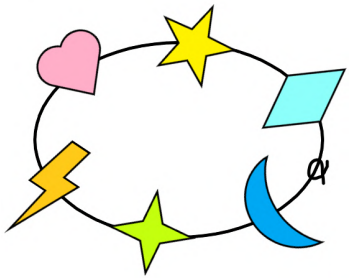
1. Die Muster

Wie viele Punkte hast du im Spiel “Die Muster” bekommen?

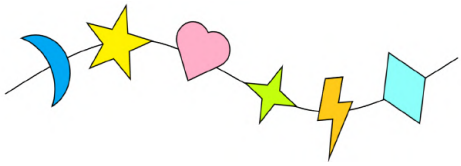
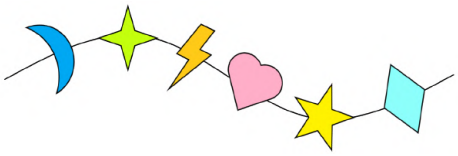
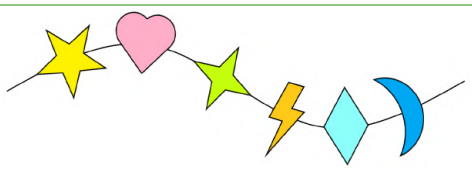
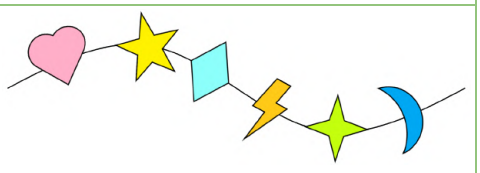
_____ Punkte

2. Die Glücksbringer

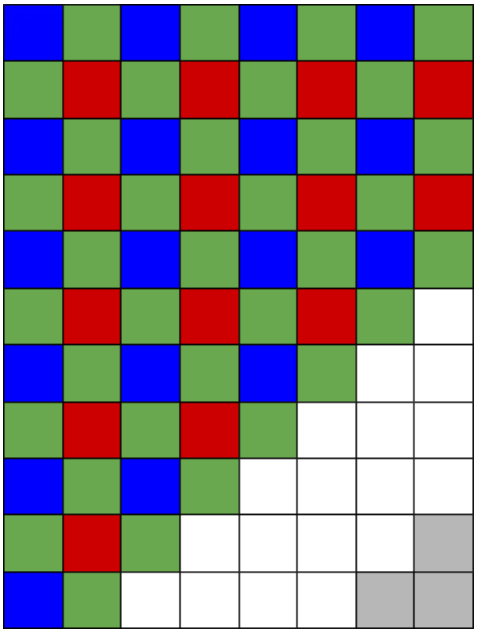
Sarah hat ein selbstgemachtes Glücksbringer-Armband aus bunten Perlen. So sieht es aus:



Für ihre beste Freundin möchte Sarah das gleiche Armband machen. Wie muss sie die Perlen auffädeln, damit ihre Freundin auch wirklich genau das gleiche Armband bekommt?

A) ☐	B) ☐
	
C) ☐	D) ☐
	

3. Farbmuster



Hier siehst du ein teilweise gefärbtes Raster mit 8 mal 11 Quadraten. Beachte das Farbmuster!

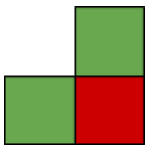
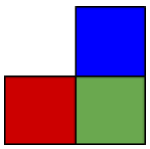
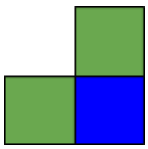
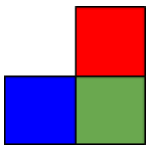
In der ersten Zeile sind die Quadrate abwechselnd blau und grün.

In der zweiten Zeile sind die Quadrate abwechselnd grün und rot.

In der dritten Zeile sind die Quadrate wieder abwechselnd blau und grün.

Und so weiter.

Wenn das ganze Raster wird mit diesem Farbmuster gefüllt wird, welche Farben bekommen dann die drei grauen Quadrate in der rechten unteren Ecke?

A) ☐	B) ☐	C) ☐	D) ☐
			

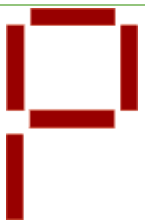
4. Fünf Hölzchen

Fünf Hölzchen liegen so auf dem Tisch:

Ali nimmt ein Hölzchen und legt es woanders hin. Jetzt liegen den Hölzchen so:

Danach nimmt Tanja ein Hölzchen und legt es woanders hin.

Wie können die Hölzchen jetzt <u>NICHT</u> liegen?

A) ☐	B) ☐
	
C) ☐	D) ☐
	

Aufgabe 1, 2 und 3 auf diesem Übungsblatt basieren auf Aufgaben aus dem Informatik-Schülerwettbewerb "Informatik-Biber" der Initiative "Bundesweit Informatik fördern", https://bwinf.de/biber/downloads/
Wir empfehlen die auf dieser Webseite zusammengestellten Materialien als Knobelaufgaben für eine vertiefende Beschäftigung mit Problemstellungen der Informatik.

Lösung zu Übungsblatt 2

2. Die Glücksbringer

Antwort B

3. Farbmuster

Antwort D

4. Fünf Hölzchen

Antwort D

Zu Hause ausprobieren:

Muster suchen

Sei ganz aufmerksam: Suche in deinem Alltag nach fünf Abläufen und Sachen, die **Muster** bilden!
Für jedes richtige Muster gibt es einen Punkt:

Beispiel: *Die Felder auf einem Schachbrett*

1. _____
2. _____
3. _____
4. _____
5. _____

Scratch-Projektkarten

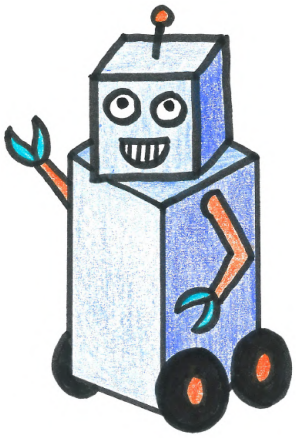
Schau dir nochmal die **Projektkarten** an! Hast du sie schon alle ausprobiert? Probiere die Projekte aus, die du noch nicht bearbeitet hast!

Gehe an einem **Computer** ins **Internet** und schreibe folgende Adresse in die **URL-Zeile**:

https://scratch.mit.edu/projects/editor

Dann drücke die **Enter-Taste**.

Bestimmt erkennst du die Seite, die sich öffnet.
Aber vielleicht ist sie auf einer anderen Sprache! Klicke auf die kleine **Weltkugel** oben links. Dann kannst du **Deutsch** als Sprache auswählen.



Viel Spaß beim Suchen und Programmieren!

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Kurssitzung 3

- Spielanleitung – „Schildkröten & Krabben: Das Wettrennen“
- Übungsblatt 3
- Zu Hause ausprobieren 3

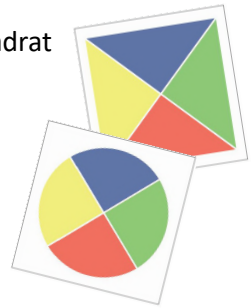
Schildkröten & Krabben: Das Wettrennen
Spielanleitung

Inhaltsverzeichnis

1. Das Spielmaterial

Für “Schildkröten & Krabben: Das Wettrennen” werden folgende Materialien benötigt:

- Das Wettrennen-**Spielbrett** (auf dem Brett sind bunte Kreise und Quadrate spiralförmig angeordnet)
- Großer **Filzwürfel** mit Augen von 1 bis 6
- Eine große 30-Sekunden-**Sanduhr**
- 8 **Tierfiguren** (4 **Schildkröten**, 4 **Krabben**, jeweils einmal in rot, gelb, grün und blau)



- 46 **Ereigniskarten** (auf der Rückseite ist ein buntes Quadrat abgebildet)
- 40 **Rästelkarten** (auf der Rückseite ist ein bunter Kreis abgebildet)
- 8 lamininierte **“Werte der Variablen”-Tafeln** (DIN A4)
- 8 **“Rechnen”-Tafeln** (DIN A4)
- 8 wegwischbare **Filzmarker** (Farbe egal)
- 8 **Schwämme**

- **Abzeichen** (können den Tieren auf den Rücken gesteckt werden):
  - **Additions-Abzeichen**: 
    grüne Plus-Zeichen
  - **Subtraktions-Abzeichen**: 
    grüne Minus-Zeichen
  - **Multiplikations-Abzeichen**: 
    grüne Mal-Zeichen
  - **Bedingungs-Abzeichen**: 

    orangene Bedingungsblöcke
  - **Variablen-Abzeichen**: 
    durchsichtige Räder
  - **Konstanten-Abzeichen**: 
    hölzerne Räder

  - **Zähler**:
    Holzplättchen mit Zahlen

2. Der Spielaufbau

Das **Spielbrett** wird so ausgelegt, dass zu allen Seiten etwas Platz ist, damit die Spielenden sich um das Spielbrett bewegen können. Die **Ereigniskarten** werden gut gemischt und als Stapel verdeckt in die Mitte des Spielfeldes gelegt. Auch die **Rätselkarten** werden gut gemischt und als separater Stapel verdeckt in die Mitte des Spielfeldes gelegt. Der **Würfel** wird ebenfalls auf das Spielbrett gelegt.

3. Die Teams

Die Spielenden bilden **Zweier-Teams**. Falls es eine ungerade Anzahl an Spielenden geben sollte, darf auch ein Dreier-Team gebildet werden.
Jedes Team sucht sich eine **Tierfigur** aus und nimmt:

- Die jeweilige **Tierfigur**
- Eine **“Werte der Variablen”-Tafel** und eine **“Rechnen”-Tafel**
- Einen **wegwischbaren Filzmarker** und einen **Schwamm**

4. Das Spiel erklären!

Die Kursleitung nimmt eine der übrig gebliebenen **Tierfiguren** und die dazugehörigen Materialien (**“Werte der Variablen”-Tafel, “Rechnen”-Tafel, Filzmarker** und **Schwamm**) und nutzt diese, um den Teams das Spiel mittels Veranschaulichungen zu erklären. Dafür setzen sich alle mitspielenden Teams um das Spielbrett. Die Kursleitung erklärt das Spiel:

“In diesem Spiel laufen die Tiere (*zeigen*) aller Teams miteinander um die Wette. Um schnell voran zu kommen, müsst ihr Rätselkarten (*zeigen*) ziehen und mathematische Rätsel lösen, wenn ihr auf Rätselfelder (*zeigen*) kommt, und Ereigniskarten (*zeigen*) ziehen und auf Ereignisse reagieren, wenn ihr auf Ereignisfelder (*zeigen*) kommt. Das Team, dessen Tier zuerst das Ziel erreicht, gewinnt!”

Jetzt schaut euch das **Spielbrett** mal genau an. Was seht ihr alles?

- Die große **Spirale** ist der Weg, den ihr gehen müsst, um zum Ziel zu kommen.
- Die eckigen Felder sind die **Ereignisfelder** (*zeigen*). Ereignisfelder können rot, gelb, grün oder blau sein. Wenn ihr würfelt und auf ein Ereignisfeld kommt, dürft ihr eine **Ereigniskarte** (*zeigen*) ziehen, auf die ihr reagieren müsst.
- Die runden Felder sind die **Rätselfelder** (*zeigen*). Rätselfelder können rot, gelb, grün oder blau sein. Wenn ihr würfelt und auf ein Rätselfeld kommt, dürft ihr eine **Rätselkarte** (*zeigen*) ziehen, die ihr lösen müsst.

- Die kleinen **Dreiecke** (*zeigen*) in jedem Feld beinhalten die **Feldnummern**. Feldnummern können nicht verändert werden, da sie auf dem Spielbrett aufgedruckt sind. Deshalb sind die Feldnummern **Konstanten**.

Alle Tiere starten auf **Feld 1** (*zeigen*). Die Tiere laufen immer so viele Felder weit, wie ihr würfelt (*mit einer Spielfigur veranschaulichen*).

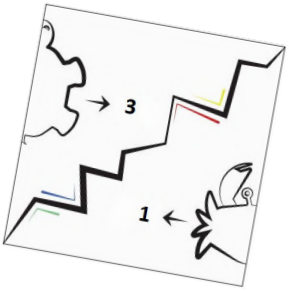
- Bei 3 oder weniger Teams: **Feld 40 ist das Ziel**! (*zeigen*)
- Bei 4 Teams: **Feld 30 ist das Ziel**! (*zeigen*)
- Bei 5 oder mehr Teams: **Feld 25 ist das Ziel**! (*zeigen*)

Jedes Team würfelt einmal. Das Team, das die **höchste Augenzahl** gewürfelt hat, **beginnt** mit dem Spiel."

4.1 Was passiert auf den Ereignisfeldern?

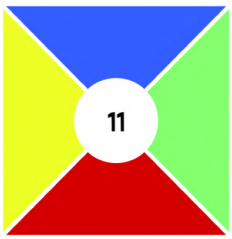
"Aber **was passiert, wenn euer Tier auf ein Ereignisfeld** (*zeigen*) **kommt**? Dann zieht ihr eine **Ereigniskarte** (*zeigen*) und müsst auf sie reagieren. Es gibt verschiedene Arten von Ereigniskarten:

- Wenn ihr so eine Karte (*auf der Karte ist **je eine Schildkröte und eine Krabbe mit einem Pfeil und einer Zahl daneben** abgebildet - passende Beispielkarte aus dem Stapel suchen und zeigen*) zieht, dürft ihr **x Felder vorwärts gehen, falls euer Tier eine Schildkröte** ist und **y Felder, falls euer Tier eine Krabbe** ist. Keine Sorge, die Zahlen auf den Karten sind für beide Tierarten **fair verteilt**!



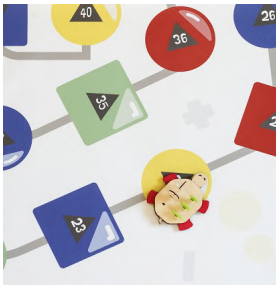
Achtung: Nur das Team, das die Karte gezogen hat, darf vorwärts gehen!

- Wenn ihr so eine Karte (*auf der Karte steht eine **Zahl inmitten eines bunten Quadrats** - passende Beispielkarte aus dem Stapel suchen und zeigen*) zieht, müssen **alle Teams** ihre **“Werte der Variablen”-Tafel** (*zeigen*) in die Hand nehmen!



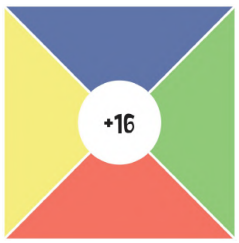
  - Schaut euch die **“Werte der Variablen”-Tafel** mal genau an. Was seht ihr? Es gibt **vier Farben**: grün, rot, blau und gelb (*zeigen*). **In diesem Spiel ist jede Farbe eine Variable**. Das heißt, **jede Farbe steht für eine Zahl**. Aber für welche Zahl sie steht, kann sich durch die Ereigniskarten (*zeigen*) verändern!

  - Jetzt schaut nach, **welche Farbe das Feld hat, auf dem euer Tier gerade steht**. Stellt euch vor, **euer Tier steht auf gelb** (*mit einer Spielfigur veranschaulichen*). Dann müsst ihr **die Zahl, für die gelb gerade steht**, **durchstrei chen** (*auf der “Werte der Variablen”-Tafel veranschaulichen*) und **durch die Zahl ersetzen, die auf der Karte steht** (*auf der “Werte der Variablen”-Tafel veranschaulichen*). Wir sagen: Der **Wert** der **Variablen** hat sich **verändert**!



  - **Achtung**: Die Werte der Variablen ändern sich immer **für alle Teams**! Die **“Werte der Variablen”- Tafel** (*zeigen*) muss also **für alle Teams immer gleich** aussehen. Wir legen jetzt für jede der vier Variablen einen **Startwert zwischen 0 und 100** fest. Meldet euch und wünscht euch eine Zahl für grün (rot, blau, gelb)! Der Startwert wird jeweils in die erste Zeile auf der Variablentafel eingetragen und ist **für alle Teams gleich**, z.B. rot = 24, blau = 0, gelb = 100, grün = 77.

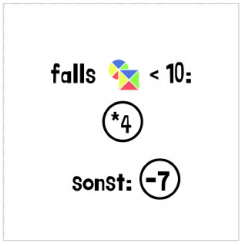
- Wenn ihr so eine Karte (*auf der Karte stehen **ein Plus und eine Zahl inmitten eines bunten Quadrats** - passende Beispielkarte aus dem Stapel suchen und zeigen*) zieht, müssen wieder alle Teams ihre **“Werte der Variablen”-Tafel** (*zeigen*) in die Hand nehmen!



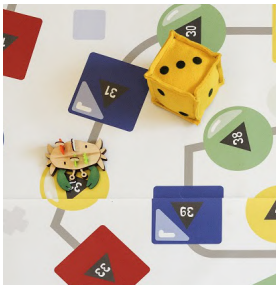
    - Schaut wieder nach, **welche Farbe das Feld hat, auf dem euer Tier gerade steht**. Stellt euch vor, **euer Tier steht auf grün** (*mit einer Spielfigur veranschaulichen*). Dann müsst ihr auf **die Zahl, für die grün gerade steht**, **durchstreichen** (*auf der “Werte der Variablen”-Tafel veranschaulichen*) und **auf diese Zahl den Wert der Zahl addieren, die auf der Karte steht, und das Ergebnis als neuen Wert aufschreiben** (*auf der “Werte der Variablen”-Tafel veranschaulichen*).

    - **Denkt daran**: Die Werte der Variablen ändern sich immer **für alle Teams**! Die **“Werte der Variablen”-Tafel** (*zeigen*) muss also **für alle Teams immer gleich** aussehen.

- Wenn ihr so eine Karte (*auf der Karte steht eine **"falls-sonst"-Anweisung** - passende Beispielkarte aus dem Stapel suchen und zeigen*) zieht, müssen wieder alle Teams ihre **"Werte der Variablen"-Tafel** (*zeigen*) in die Hand nehmen!



  - Schaut wieder nach, **welche Farbe das Feld hat, auf dem euer Tier gerade steht**. Stellt euch vor, **euer Tier steht auf gelb** (*mit einer Spielfigur veranschaulichen*). Dann müsst ihr **die Zahl, für die gelb gerade steht**, anschauen und überprüfen, ob die **Bedingung erfüllt** ist.
  - Wenn gelb gerade kleiner als 10 ist, ist die Bedingung erüllt. Dann müsst ihr **die Zahl, für die gelb gerade steht**, durchstreichen und **diese Zahl mit 4 multiplizieren und das Ergebnis als neuen Wert aufschreiben** (*auf der "Werte der Variablen"-Tafel veranschaulichen*).



  - Wenn gelb gerade nicht kleiner als 10 ist, ist die Bedingung nicht erüllt. Dann müsst ihr **die Zahl, für die gelb gerade steht**, durchstreichen und **von dieser Zahl 7 subtrahieren und das Ergebnis als neuen Wert aufschreiben** (*auf der "Werte der Variablen"-Tafel veranschaulichen*).
  - **Denkt daran**: Die Werte der Variablen ändern sich immer **für alle Teams**! Die **"Werte der Variablen"-Tafel** (*zeigen*) muss also **für alle Teams immer gleich** aussehen."

4.2 Abzeichen für Ereignisse

“Wenn ihr auf die **Ereignisse** richtig reagiert, könnte ihr dafür **Abzeichen** bekommen. Bei Den Ereignissen gilt: **Nur das Team, das gerade am Zug ist, kann Abzeichen bekommen!**

Subtraktions-Abzeichen: Wenn ihr richtig auf ein **Ereignis** reagiert, das eine **Subtraktion** enthält

Multiplikations-Abzeichen: Wenn ihr richtig auf ein **Ereignis** reagiert, das eine **Multiplikation** enthält

Bedingungs-Abzeichen: Wenn ihr richtig auf ein **Ereignis** mit einer **Bedingung** reagiert, also die **richtige Variable richtig ändert**

Variablen-Abzeichen: Wenn ihr richtig auf ein **Ereignis** reagiert, das eine **Variable** enthält

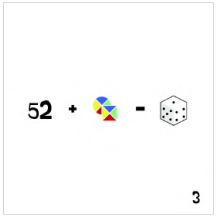
Wenn ihr eines dieser Abzeichen schon habt, bekommt ihr es danach nicht nochmal, sondern bekommt stattdessen einen kleinen **Zähler**, den ihr auf das Abzeichen setzen könnt, um zu zählen, wie oft ihr es schon bekommen habt.”

4.3 Was passiert auf den Rätselfeldern?

"Aber **was passiert, wenn euer Tier auf ein Rätselfeld** (*zeigen*) **kommt**? Dann zieht ihr eine **Rätselkarte** (*zeigen*) und **alle** müssen ein mathematisches Rätsel lösen. Dafür müssen **alle Teams** ihre **"Rechnen"-Tafel** (*zeigen*) in die Hand nehmen! In den Rätseln gibt es verschiedene Zeichen:

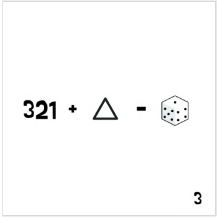
- △ Das **Dreieck** (*zeigen*) ist eine **Konstante** und steht für die **Nummer des Feldes** (*zeigen*), auf dem ihr gerade steht. Wie ihr seht, steht die **Feldnummer** auf dem Spielbrett (*zeigen*) immer in einem Dreieck.
- Das **bunte Farbsymbol** (*zeigen*) ist eine **Variable** und steht für den **Wert der Farbe des Feldes** (*zeigen*), auf dem ihr gerade steht. Wie ihr seht, haben die Farben auf der **"Werte der Variablen"-Tafel** (*zeigen*) das gleiche Symbol.
- Der **Würfel** (*zeigen*) ist eine **Variable** und steht für die **Zahl, die ihr im aktuellen Zug gewürfelt habt**.

Wenn ihr eine **Rätselkarte** gezogen habt, zeigt ihr sie so, dass alle sie sehen können. Dann schreiben **alle Teams** die Rechnung und die **Lösung des Rätsels** auf ihre **"Rechnen"-Tafel** (*zeigen*). Dafür habt ihr **30 Sekunden** Zeit. Passt auf, dass die anderen Teams **nicht abschreiben** können! Wenn ich es sage, halten alle Teams ihre **"Rechnen"-Tafel** hoch und ich kontrolliere die Ergebnisse. **Alle Teams mit einer richtigen Lösung** bekommen **Abzeichen**!



Denkt daran: **Eine Rätselkarte kann nie die Variablen ändern**!

Außerdem seht ihr auf allen Rätselkarten **unten rechts eine Zahl** (*zeigen*). Wenn ihr das Rätsel gelöst habt, darf **das Team, das die Karte gezogen** hat, nochmal **so viele Felder vorwärts gehen, wie die Zahl sagt**! Achtung: Das Team darf dann aber nicht nochmal eine Karte ziehen.



4.4 Abzeichen für Rätsel

“Wenn ihr die **Rätsel** richtig löst, könnte ihr dafür **Abzeichen** bekommen. Bei Den Rätseln gilt: **Alle Teams können Abzeichen bekommen!**

Additions-Abzeichen: Wenn ihr ein **Rätsel** mit einer **Addition** richtig löst

Subtraktions-Abzeichen: Wenn ihr ein **Rätsel** mit einer **Subtraktion** richtig löst

Multiplikations-Abzeichen: Wenn ihr ein **Rätsel** mit einer **Multiplikation** richtig löst

Variablen-Abzeichen: Wenn ihr ein **Rätsel** mit einer **Variablen** richtig löst

Konstanten-Abzeichen: Wenn ihr ein **Rätsel** mit einer **Konstanten** richtig löst

Wenn ihr eines dieser Abzeichen schon habt, bekommt ihr es danach nicht nochmal, sondern bekommt stattdessen einen kleinen **Zähler**, den ihr auf das Abzeichen setzen könnt, um zu zählen, wie oft ihr es schon bekommen habt.”

4.4 Ende des Spielzugs

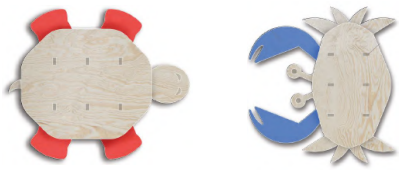
“Wenn ihr (a) auf ein Ereignis reagiert habt oder (b) ein Rätsel bearbeitet habt, ist euer **Spielzug zu Ende** und das nächste Team ist an der Reihe.

Am Ende eures Spielzuges legt ihr die Karte, die ihr vom **Ereigniskartenstapel** (*zeigen*) oder vom **Rätselkartenstapel** (*zeigen*) in der Mitte genommen habt, zurück unter den Stapel.

4.7 Ende des Spiels

“Wenn ein Team **das Ziel erreicht** hat, wird die aktuelle **Spielrunde noch fertig gespielt**, damit alle Teams gleich oft dran waren. **Dann ist das Spiel zu Ende.**

Wenn dann **nur ein Team** das Ziel erreicht hat, hat dieses Team **gewonnen**. Wenn **mehrere Teams** das Ziel erreicht haben, **gewinnt von diesen Teams** dasjenige, das **die meisten Punkte für Abzeichen** gesammelt hat.”



5. Spielvarianten für verkürzte oder verlängerte Spielzeit

5.1 Verkürzte Spielzeit

Um die Spielzeit zu verkürzen, kann ein Zielfeld mit einer niedrigeren Nummer Nummer gewählt werden. Statt z.B. Feld 30 zum Zielfeld zu nehmen, könnte Feld 25 als neues Zielfeld bestimmt werden.

Um außerdem den Schwierigkeitsgrad des Spiels zu senken, können schwierigere Rätselkarten (mit der Zahl 3 in der rechten unteren Ecke) aussortiert werden.

5.2 Verlängerte Spielzeit

Um die Spielzeit zu verlängern, kann ein Zielfeld mit einer höheren Nummer Nummer gewählt werden. Statt z.B. Feld 30 zum Zielfeld zu nehmen, könnte Feld 40 als neues Zielfeld bestimmt werden.

Name: _____

Übungsblatt 3

1. Das Wettrennen

Wieviele Punkte hat dein Team im Spiel “Schildkröten & Krabben: Das Wettrennen” bekommen?

Für **Additionen**: _____ Punkte

Für **Subtraktion**: _____ Punkte

Für **Multiplikationen**: _____ Punkte

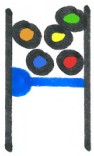
Für **Bedingungen**: _____ Punkte

Für **Variablen**: _____ Punkte

Für **Konstanten**: _____ Punkte

2. Der Kaugummiautomat

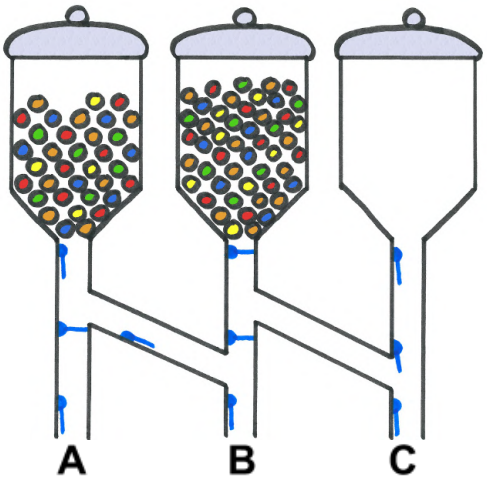
Wenn das Ventil offen ist, kommen Kaugummis durch.



Wenn das Ventil geschlossen ist, kommen keine Kaugummis durch.

Unter welchen Ausgang musst du deine Hand halten, um Kaugummis zu bekommen?

Deine Antwort: _____



3. Krabben-Geld

Um den Warenaustausch zwischen den verschiedenen Krabbenvölkern zu erleichtern, haben die Krabben eine gemeinsame Währung eingeführt, den “Kreuro”.

Es gibt Kreuro-Münzen mit den Werten 1, 2, 4 und 8 Kreuro.

Die Krabben lieben die neuen Münzen und wollen beim Bezahlen immer möglichst wenige davon weggeben.

Wie lautet die kleinste <u>Anzahl</u> Münzen, mit denen eine Krabbe den Betrag von 13 Kreuro passend bezahlen kann?

Deine Antwort: _____

4. Der Krabben-Wettlauf

Drei entschlossene Krabben treten zum Geländelauf an.

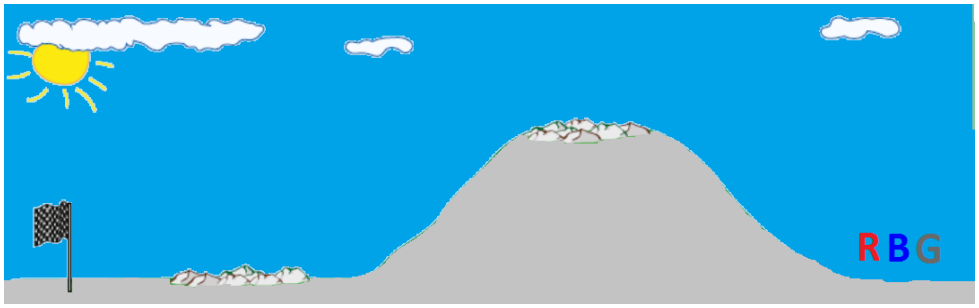
Jedesmal wenn es bergab geht, überholt Frau Rot genau eine Krabbe.

Jedesmal wenn es bergauf geht, überholt Herr Blau genau eine Krabbe.

Jedesmal wenn es über Felsen geht, überholt Herr Grau genau eine Krabbe.

Im Bild sieht man, dass die Strecke erst bergauf führt, dann folgen Felsen. Danach geht es bergab und schließlich folgen wieder Felsen. Zuerst startet Frau Rot, als nächstes Herr Blau und zuletzt Herr Grau.

In welcher Reihenfolge kommen die Krabben am Ziel an?



A) ☐ **Frau Rot, Herr Blau, Herr Grau (R B G)**

B) ☐ **Herr Blau, Herr Grau, Frau Rot (B G R)**

C) ☐ **Herr Grau, Frau Rot, Herr Blau (G R B)**

D) ☐ **Herr Blau, Frau Rot, Herr Grau (B R G)**

Aufgabe 1, 2 und 3 auf diesem Übungsblatt basieren auf Aufgaben aus dem Informatik-Schülerwettbewerb "Informatik-Biber" der Initiative "Bundesweit Informatik fördern", https://bwinf.de/biber/downloads/
Wir empfehlen die auf dieser Webseite zusammengestellten Materialien als Knobelaufgaben für eine vertiefende Beschäftigung mit Problemstellungen der Informatik.

Lösung zu Übungsblatt 3

2. Der Kaugummi-Automat

Antwort B

3. Krabben-Geld

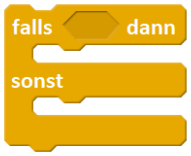
Antwort: 3

4. Der Krabben-Wettlauf

Antwort B

Zu Hause ausprobieren:

Bedingungen suchen



Sei ganz aufmerksam: Suche in deinem Alltag nach fünf Abläufen und Sachen, die wie eine **Bedingung** funktionieren!
Für jede richtige Bedingung gibt es einen Punkt:

Beispiel: *Falls ich direkt nach dem Mittagessen meine Hausaufgaben fertig mache, dann darf ich danach eine Stunde fernsehen*

1. _____
2. _____
3. _____
4. _____
5. _____

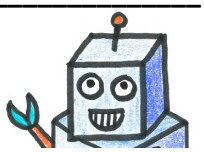
Ereignisse suchen

Suche in deinem Alltag außerdem nach fünf Abläufen und Sachen, die wie ein **Ereignis** funktionieren!
Für jedes richtige Ereignis gibt es einen Punkt:

Beispiel: *Wenn ich den Lichtschalter drücke, geht das Licht an*

1. _____
2. _____
3. _____
4. _____
5. _____

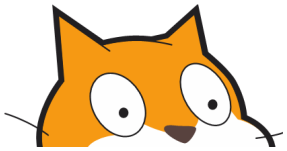
Viel Spaß beim Suchen!



EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Kurssitzung 4

- Spiele programmieren
- Übungsblatt 4
- Zu Hause ausprobieren 4



Lasst uns ein Spiel programmieren!

Wir wollen uns heute weiter mit der Scratch-Programmierumgebung vertraut machen!

- Öffne den Internet-Browser (Mozilla Firefox, Chrome, Safari oder Internet Explorer) auf deinem Desktop.

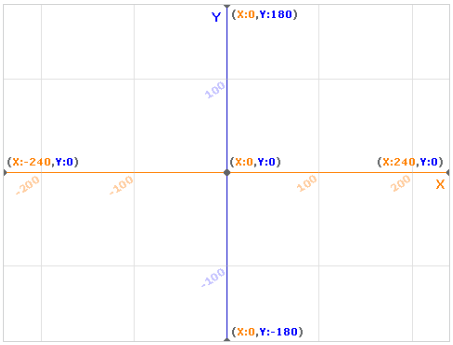
Schreibe Folgendes in die URL-Zeile:

https://scratch.mit.edu/projects/editor

- Drücke auf Enter.
- Starte dein eigenes Projekt.

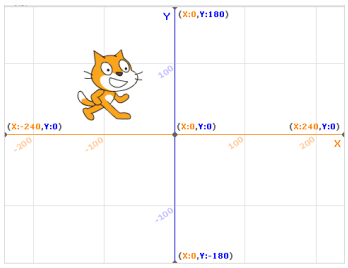
Koordinatensystem mit x-Achse und y-Achse

Für jede Stelle im Scratch-Bildschirm gibt es eine Kombination von zwei Zahlen, sodass man die Stelle ganz genau finden kann. Für die zwei Zahlen verwendet man die Variablen x und y als Platzhalter.
x legt fest, ob die Stelle weiter links (kleinere Zahlen) oder rechts (größere Zahlen) im Bild ist.
y legt fest, ob die Stelle weiter oben (kleinere Zahlen) oder unten (größere Zahlen) im Bild ist.
Die Stelle ganz in der Mitte ist x = 0 und y = 0.

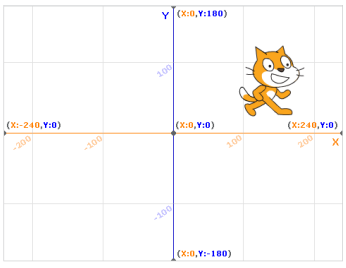


Wenn x = 50 und y = -50 ist, wo befindet sich dann die Scratch-Katze?

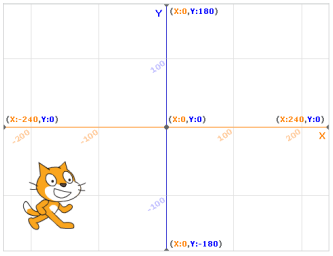
A) ☐



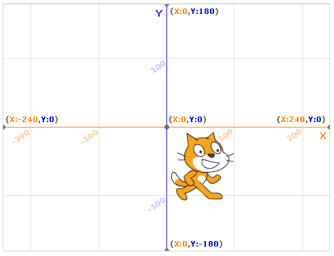
B) ☐



C) ☐

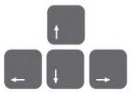


D) ☐



1 Das Katzenspiel!

Die meisten Computerspiele nutzen Tasten wie zum Beispiel die Pfeiltasten nach oben/unten/links/rechts!



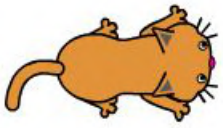
Lasst uns ein einfaches Spiel programmieren, in dem eine Katze sich auf dem Bildschirm bewegt, wenn wir die Pfeiltasten drücken!

- **Wähle diesen Hintergrund aus:**

Denke daran: Um einen Hintergrund auszuwählen, musst du unten links in den Bereich "Bühne" gehen und auf klicken. Dann kannst du den Hintergrund aussuchen und auf "Ok" klicken.

- **Wähle diese Figur aus der Scratch-Bibliothek aus:**

Gehe unten links in den Bereich "Figuren" und klicke auf (Figur aus der Bibliothek wählen).



- **Erzeuge jetzt für das Objekt Katze den folgenden Block-Code.**
- **Wähle dafür zuerst das passende Objekt aus und gehe dann zum Skript-Tab.**

Katze:

- **Diese Blöcke bestimmen die Bewegungen der Katze. Wenn die Pfeiltasten gedrückt werden, bewegt sie sich nach links oder nach rechts, nach oben oder nach unten.**

Kannst du auch die anderen beiden Befehlsblöcke für Bewegungen nach links und nach rechts zusammensetzen?

Verstehst du die Gradzahlen und wie sie funktionieren?

Was passiert, wenn du eine andere Anzahl Schritte angibst?

2 Das Frosch- & Insektenspiel!

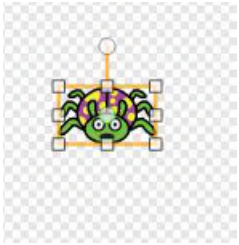
Jedes Spiel kann man gewinnen! Um deinen Punktestand im Blick zu behalten, musst du ihn irgendwo speichern. Dafür brauchst du eine Variable mit veränderbarem Wert!
Schauen wir uns an, wie das in Scratch funktioniert. Im folgenden Spiel geht es um einen Frosch, der versucht, Insekten zu fangen und zu fressen. Jedesmal, wenn er ein Insekt fängt, bekommt die Spielerin oder der Spieler einen Punkt!

- **Wähle diese beiden Figuren aus der Scratch-Bibliothek aus.**

Gehe unten links in den Bereich "Figuren" und klicke auf (Figur aus der Bibliothek wählen).

- **Achte auf ihre Größe. Sie sollten nicht zu klein oder zu groß sein, also verändere ihre Größe!**

Dazu musst du die Figur auswählen, nachdem du sie erzeugt hast.
Dann musst du auf den "Kostüme"-Tab klicken und dort ihre Größe verändern.



Wie erzeugen wir Variablen?

Gehe dazu in den "Daten"-Tab und klicke auf "Neue Variable". Dann kannst du deiner Variable einen Namen geben (z.B. Punkte).

- **Erzeuge jetzt für die Bühne den folgenden Block-Code.**
- **Wähle dafür zuerst die Bühne aus und gehe dann zum Skript-Tab.**

Frosch:

- **Erzeuge deine Pfeiltasten-Befehle. Du willst, dass der Frosch nach rechts geht, wenn du den Pfeil nach rechts drückst, und dass er nach links geht, wenn du den Pfeil nach links drückst.**

Fast so wie im Spiel mit der Katze!

Sieht etwas anders aus? Experimentiere!

- **Lass uns jetzt den Punktestand auf 1 setzen, wenn der Frosch das Insekt berührt.**

Du musst das Spiel starten lassen, wenn die grüne Flagge angeklickt wird.

Das Spiel muss mit 0 Punkten starten.

Wenn der Frosch die Figur1 (Käfer, Insekt, oder ein anderes Tier) berührt, muss der Punktestand auf 1 gesetzt werden.

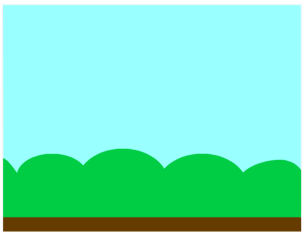
Warum brauchen wir hier die "wiederhole fortlaufend"-Schleife?

Was passiert, wenn wir diese Schleife weglassen? Warum?

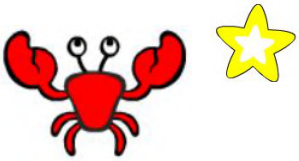
Bonusaufgabe: Das Krabbenspiel!

Fangen wir an, etwas anspruchsvollere Spiele zu programmieren! In diesem Spiel versucht die Hauptfigur so viele Dinge aufzufangen wie möglich. Wenn etwas nicht aufgefangen wird, ist die Spiel vorbei und die Spielerin oder der Spieler verliert!

- **Wähle diesen Hintergrund aus:**



- **Wähle diese beiden Figuren aus der Scratch-Bibliothek aus.**
- **Achte auf ihre Größe. Sie sollten nicht zu klein oder zu groß sein, also verändere ihre Größe!**



- **Erzeuge diese beiden Figuren in der "Neue Figur zeichnen"- Umgebung von Scratch.**

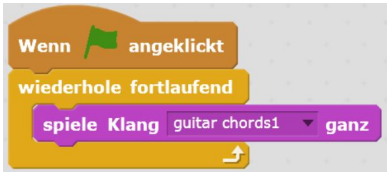
Klicke dazu auf  ***und dann auf***  ***. Du musst zwei verschiedene Figuren erzeugen! Teste verschiedene Schriftarten, Farben und Größen!***

Diese beiden Spielergebnisse dürfen nicht auf den Hintergrund geschrieben werden, sondern müssen als Objekte erzeugt werden, damit man sie verstecken kann. Sonst werden sie beide das ganze Spiel lang angezeigt!

- **Erzeuge jetzt für die verschiedenen Objekte den folgenden Block-Code.**
- **Wähle dafür zuerst das passende Objekt aus und gehe dann zum Skript-Tab.**

Bühne:

- **Diese Blöcke lassen das Spiel hindurch Musik erklingen. Die Musik ist dann mit dem Hintergrund verbunden.**



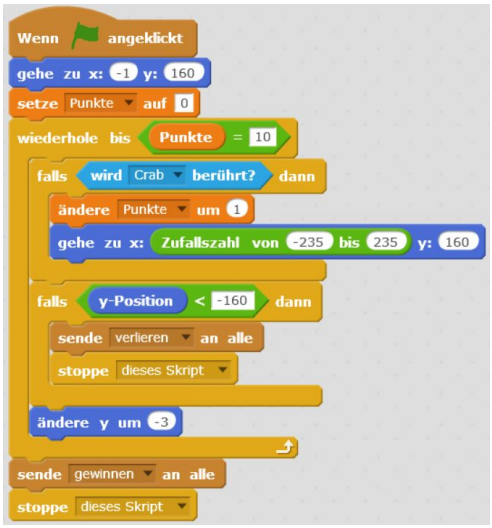
Crab (die Krabbe):

- **Diese Blöcke bestimmen die Bewegungen der Krabbe. Wenn die Pfeiltasten gedrückt werden, bewegt sie sich nach links oder nach rechts.**

Star1 (der Stern):

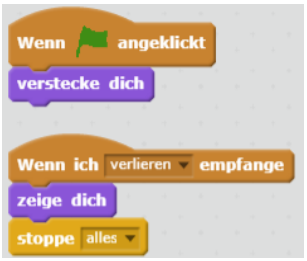
- **Diese Blöcke bestimmen, wie die Sterne fallen. Sie geben an, an welchem Ort die Sterne anfangen zu fallen und wie schnell sie fallen.**
- **Sie bestimmen den Punktestand.**
- **Sie bestimmen, wann das Spiel vorbei ist.**
- **Sie benutzen eine Schleife, um das Fallen der Sterne bis zum Ende zu wiederholen.**

➪ *Wo ist die **Schleife**? Welche Farbe hat sie?*
➪ *Wo sind die **Bedingungen**? Welche Farbe haben sie?*
➪ *Wo ist die **Variable**? Welche Farbe hat sie?*
➪ *Wo sind die mathematischen **Operatoren**? Welche Farbe haben sie?*



Objekt1 ("Du verlierst!"):

- **Diese Blöcke werden ausgeführt, wenn die Bedingung erfüllt ist, dass das Spiel verloren wurde.**



Objekt2 (“Du gewinnst!”):

- **Diese Blöcke werden ausgeführt, wenn die Bedingung erfüllt ist, dass das Spiel gewonnen wurde.**

Probiere diese Möglichkeiten aus. Was passiert dann?

1. Mache die Bewegungen der Krabbe schneller.
2. Ändere das Programm so, dass das Spiel erst nach 20 aufgefangenen Sternen endet.
3. Ändere die Anfangsposition der Sterne.
4. Ändere den Klang des Hintergrundes.
5. Ändere das Hintergrundbild.

Gehe jetzt auf die folgende Seite: **https://scratch.mit.edu/projects/245431123/**

➪ *Ist dein Spiel so ähnlich?* ➪ *Was ist anders?* ➪ *Warum?*

Die auf diesem Arbeitsblatt verwendeten Bilder sind Screenshots der Webseite: https://scratch.mit.edu/projects/editor/
Wir empfehlen diese Webseite für vertiefende Übungen im Programmieren in Scratch.

Name: _____

Übungsblatt 4

1. Das Urlaubsfoto

Du hast im Urlaub einige Fotos gemacht. Dein bester Freund möchte ein bestimmtes Foto von einer Krabbe haben, das du am Strand gemacht hast. Du willst herausfinden, welches Foto er gerne haben möchte.

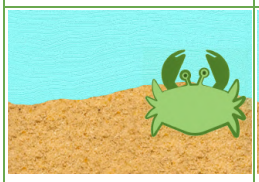
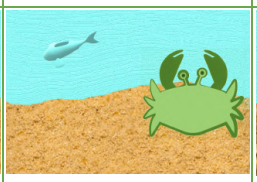
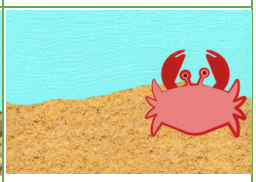
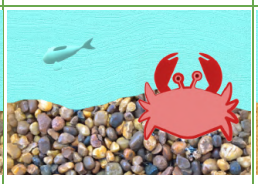
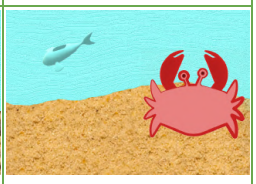
Dazu stellst du ihm einige Fragen:

“Möchtest du ein Foto, auf dem auch ein Fisch zu sehen ist?” - “Ja.”

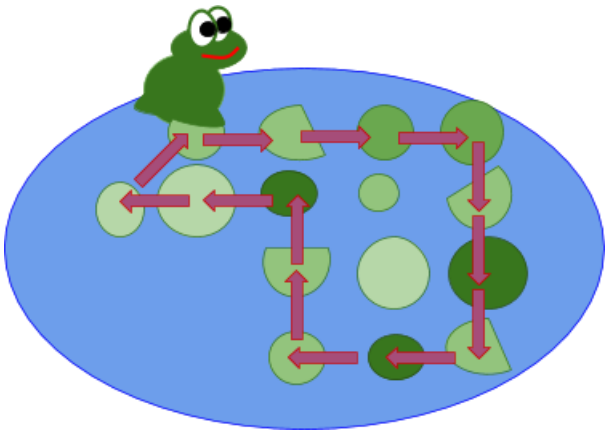
“Möchtest du ein Foto von einer grünen Krabbe?” - “Nein.”

“Möchtest du ein Foto, auf dem ein Sandstrand zu sehen ist?” - “Ja.”

Welches Foto möchte dein Freund haben?

A) ☐	B) ☐	C) ☐	D) ☐
			
E) ☐	F) ☐	G) ☐	H) ☐
			

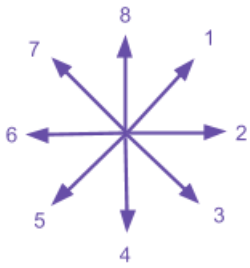
2. Froschhüpfen



Ein Frosch will vor Freude herumhüpfen. In seinem Teich sind viele Seerosenblätter. Auf einem Seerosenblatt sitzt der Frosch.

Er hüpft so von Blatt zu Blatt, wie es im Bild zu sehen ist. Am Ende sitzt der Frosch wieder dort, wo er losgehüpft ist.

Die acht möglichen Hüpfrichtungen des Frosches sind so nummeriert:



Mit den Nummern kann man Hüpfwege beschreiben.

Welchen Weg ist der Frosch gehüpft?

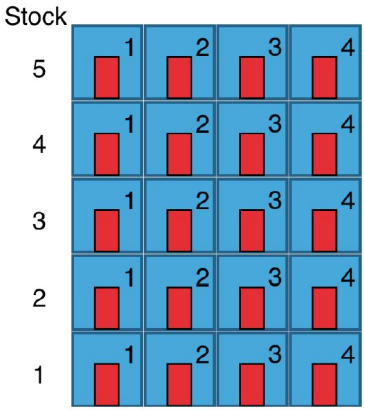
A) ☐ **2, 2, 3, 4, 5, 7, 7, 8**

B) ☐ **2, 2, 2, 4, 4, 4, 6, 6, 8, 8, 6, 6, 1**

C) ☐ **5, 2, 2, 4, 4, 2, 2, 8, 8, 8, 6, 6, 6**

D) ☐ **5, 2, 2, 3, 3, 7, 3, 3, 7, 3, 3, 7, 8**

3. Malerarbeiten



In einem Wohnblock leben viele Schildkröten-familien. Alle Türen zu den einzelnen Wohnungen sind ursprünglich mit roter Farbe angemalt.

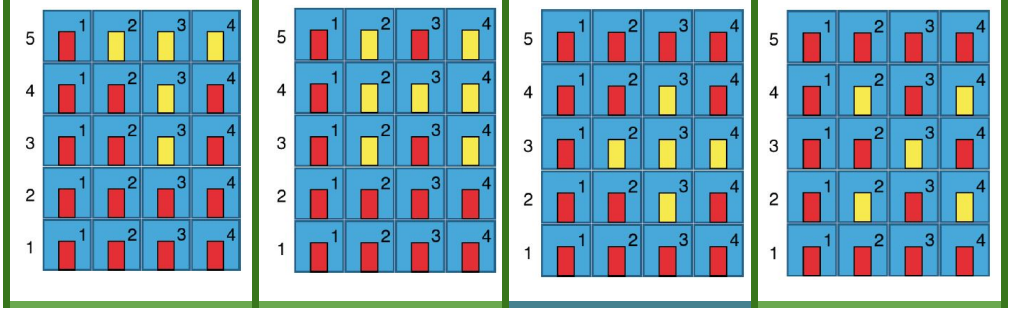
Ein Maler hat den Auftrag, bei den folgenden Wohnungen die Türen gelb zu streichen:

- Wohnung (2,2)
- Wohnung (4,2)
- Wohnung (3,3)
- Wohnung (2,4)
- Wohnung (4,4)

Wohnung (x,y) bedeutet: Im x-ten Stock die y-te Tür.

Wie sieht der Wohnblock aus, nachdem der Maler seine Arbeit beendet hat?

A) ☐ B) ☐ C) ☐ D) ☐



Aufgabe 1, 2 und 3 auf diesem Übungsblatt basieren auf Aufgaben aus dem Informatik-Schülerwettbewerb "Informatik-Biber" der Initiative "Bundesweit Informatik fördern", https://bwinf.de/biber/downloads/
Wir empfehlen die auf dieser Webseite zusammengestellten Materialien als Knobelaufgaben für eine vertiefende Beschäftigung mit Problemstellungen der Informatik.

Lösung zu Übungsblatt 4

1. Das Urlaubsfoto

Antwort H

2. Froschhüpfen

Antwort B

3. Malerarbeiten

Antwort D

Zu Hause ausprobieren:

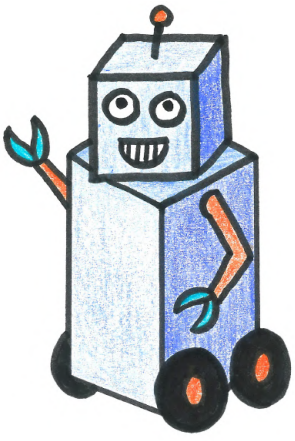
Hast du das **Krabbenspiel** schon fertig programmiert?
Falls nicht, probiere es aus!

Außerdem kannst du **Scratch-Projekte von anderen Programmierern** erkunden:

- Öffne das **Internet** (Mozilla Firefox, Chrome, Safari oder Internet Explorer).
- Schreibe Folgendes in die URL-Zeile:
  https://scratch.mit.edu/explore/projects
- Drücke auf **Enter**.
- Wähle ein Projekt aus und klicke es an.
- Klicke auf **Schau hinein**, um den Programm-Code zu sehen!
- Klicke auf die kleine **Weltkugel** eben dem Scratch-Logo oben links. Wähle **Deutsch** aus, um den Block-Code auf Deutsch zu sehen!
- Starte dein **eigenes Projekt**, indem du auf **Datei** klickst und dann auf **Neu**.

Hier kannst du nochmal das **Krabbenspiel** anschauen:
https://scratch.mit.edu/projects/245431123/

Viel Spaß beim Programmieren!



Die auf diesem Arbeitsblatt verwendeten Bilder (mit Ausnahme des Roboters) sind Screenshots der Webseite:
https://scratch.mit.edu/projects/editor/
Wir empfehlen diese Webseite für vertiefende Übungen im Programmieren in Scratch.

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Kurssitzung 5

- Simulationen
- Übungsblatt 5
- Zu Hause ausprobieren 5

Lasst uns eine Simulation programmieren!

Wir wollen uns heute weiter mit der Scratch-Programmierumgebung vertraut machen!

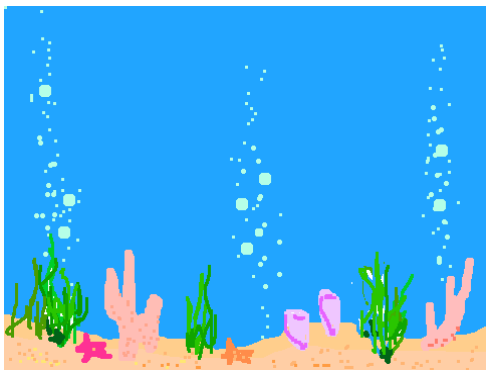
- Öffne den Internet-Browser (Mozilla Firefox, Chrome, Safari oder Internet Explorer) auf deinem Desktop.
- Schreibe Folgendes in die URL-Zeile:

  https://scratch.mit.edu/projects/editor
- Drücke auf Enter.
- Starte dein eigenes Projekt.

Das Nahrungsketten-Spiel

Lasst uns wiederholen, was wir schon alles gelernt haben. Jetzt programmieren wir ein Unterwasser-Nahrungsketten-Spiel! Der Hai versucht, den mittelgroßen Fisch zu fangen. Der mittelgroße Fisch versucht, die kleinsten Fische zu fangen.
Der Spieler oder die Spielerin steuert den mittelgroßen Fisch mit den vier Pfeiltasten. Um zu gewinnen, musst du alle drei kleinen Fische fangen und darfst dich nicht vom Hai fressen lassen!

- **Wähle diesen Hintergrund aus der Scratch-Bibliothek aus.**

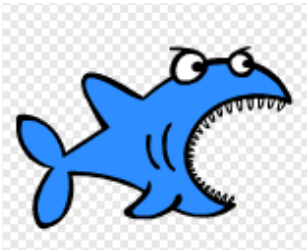


- **Wähle diese Figuren aus der Scratch-Bibliothek aus.**

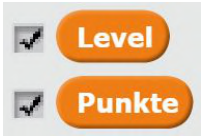
Manche Figuren, z.B. der Hai, haben verschiedene Kostüme. Du kannst die Kostüme finden, wenn du die Figur anklickst und dann auf "Kostüme" klickst. Dort kannst du dir ein Kostüm aussuchen.

Weil du 3 kleine Fische fangen musst, musst du 3 unterschiedliche Figuren für den kleinen Fisch erzeugen.
Denke daran: Du kannst mit rechts auf ein Objekt klicken und "Duplizieren" auswählen, um das Objekt nochmal zu erstellen.

Vergiss nicht, deine Figuren größer oder kleiner zu machen, damit es zum Spiel passt.
Der Hai ist größer als der bunte Fisch. Der bunte Fisch ist größer als die orangenen Fische.



Erzeuge zwei Variablen, eine für das Level und eine für die Punkte.



Spieler/in (bunter Fisch):

Denke daran: Du kannst deinen Objekten neue Namen geben, indem du sie mit rechts anklickst und "Info" auswählst. Dann kannst du ihren Namen ändern.

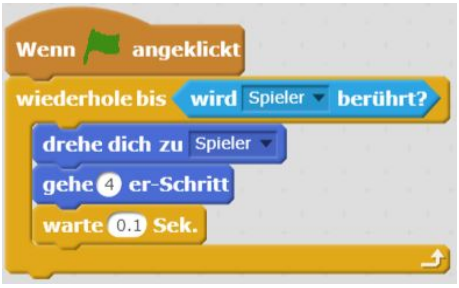
- **Setze das Level auf 1 und die Punkte auf 0.**
- **Benutze die Beweguns-funktionen der Pfeiltasten auf der Tastatur!**

- **Falls alle 3 kleinen Fische gefangen wurden, gewinnst du und das Spiel endet.**
- **Falls der Hai es schafft, dich zu fangen, verlierst du und das Spiel endet.**

Hai:

- **Programmiere den Hai so, dass er an einer zufälligen Stelle erscheint.**

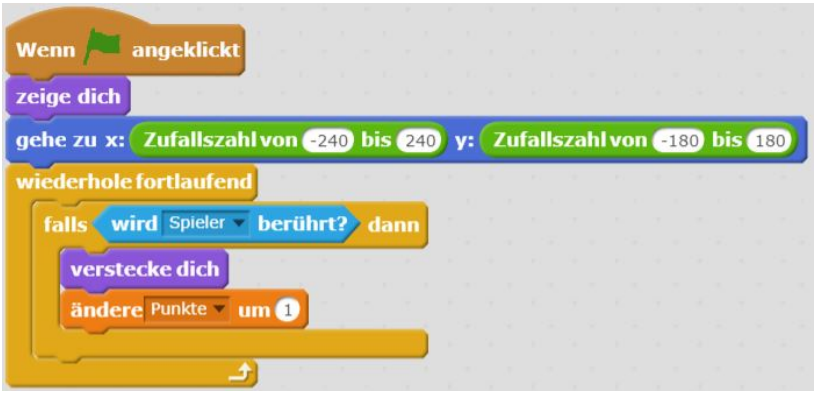
- **Programmiere den Hai so, dass er sich auf die Spielerin oder den Spieler (bunter Fisch) zu bewegt. So jagt der Hai den bunten Fisch!**



Kleiner Fisch:

Du musst diesen Block-Code für alle 3 kleinen Fische erzeugen. Um das Skript nicht dreimal schreiben zu müssen, kannst du zuerst den ersten kleinen Fisch programmieren und ihn dann zweimal duplizieren. Das spart Zeit!

- **Programmiere den kleinen Fisch so, dass er an einer zufälligen Stelle erscheint.**
- **Erhöhe die Punkte, wenn ein kleiner Fisch vom bunten Fisch gefangen wird!**



- **Programmiere den kleinen Fisch so, dass er zu zufälligen Stellen schwimmt.**

Probiere diese Möglichkeiten aus. Was passiert dann?

1. Programmiere ein weiteres Level.
2. Füge in diesem Level noch einen kleinen Fisch hinzu.

Gehe jetzt auf die folgende Seite: **https://scratch.mit.edu/projects/245186216/**

Was ist anders? Warum?
Könntest du noch mehr Levels programmieren?

Bonusaufgabe: Der Wasserkreislauf

Lasst uns ein Scratch-Programm erstellen, das simuliert, wie der natürliche Wasserkreislauf funktioniert!

- **Male einen passenden Hintergrund:**

Um einen Hintergrund auszuwählen, musst du unten links in den Bereich "Bühne" gehen und auf  *klicken.*

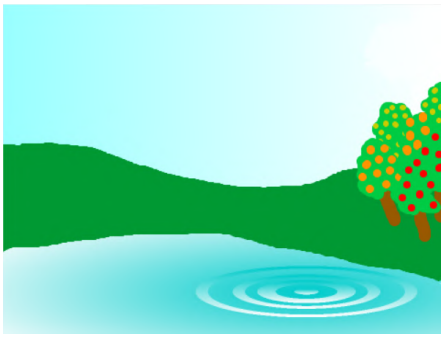
Dein Hintergrund muss folgende Dinge enthalten:

- **blauer Himmel**
- **Wiese**
- **See**

Probiere dafür die folgenden Werkzeuge aus ➡ ➡ ➡ ➡ ➡ ➡ ➡ ➡

Denke daran: Mit  *kannst du radieren, wenn dir etwas in deinem Bild nicht gefällt.*

Außerdem kannst du  *benutzen, um etwas rückgängig zu machen oder etwas wiederherzustellen.*



- **Wähle diese beiden Figuren aus der Scratch-Bibliothek aus:**

Gehe unten links in den Bereich "Figuren" und klicke auf  *. Mache die Wolke richtig groß und die Sonne ein bisschen kleiner!*

- **Lass die Sonne scheinen:**

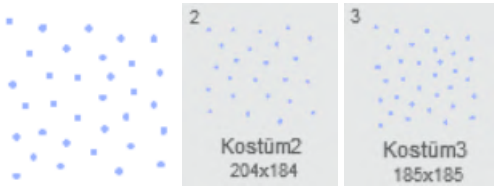
Wähle die Sonne aus und klicke auf Kostüme. Füge ein zweites Kostüm hinzu und male Strahlen.
Verschiebe nicht die Sonne im Kostüm!

- **Male den Wasserdampf:**

Gehe unten links in den Bereich "Figuren" und klicke auf . Male drei blaue Pfeile, die nach oben zeigen!

- **Male den Regen:**

Male eine neue Figur! Die Figur soll aus vielen kleinen Regentropfen bestehen. Füge zwei neue Kostüme mit noch mehr Regentropfen hinzu!



Erzeuge jetzt den Block-Code!
Wähle immer das passende Objekt aus, gehe zu "Skripte" und erstelle den Code!

Nachrichten:

Damit die verschiedenen Figuren in der richtigen Reihenfolge aktiviert werden, brauchen wir Nachrichten. Nachrichten kannst du mit "neue Nachricht..." erstellen.

Erstelle folgende Nachrichten:

- **Start**
- **Wärme**
- **Wolke**
- **Regen**

Bühne:

Auf der Bühne startet der Kreislauf!

Du kannst außerdem eine Musikschleife hinzufügen, die während des gesamten Kreislaufs gespielt wird.

Sonne:

Am Anfang des Kreislaufs muss die Sonne scheinen. Zeige die Sonnenstrahlen, indem du das Kostüm der Sonne wechselst!

Siehst du die Schleife? Was macht sie?

Die Sonnenstrahlen geben Wärme ab. ➡

Wasserdampf (Pfeile):

Am Anfang des Kreislaufs ist noch kein Wasserdampf zu sehen. Deshalb muss er zuerst versteckt werden!

Das Wasser erwärmt sich, wenn es von den Sonnenstrahlen berührt wird. Wenn Wasser warm wird, steigt Wasserdampf auf.

Siehst du die Schleife? Was macht sie?

Wenn der Dunst den Himmel erreicht, wird er durchsichtig und bildet eine Wolke ➡ ➡ ➡ ➡ ➡ ➡ ➡

Wolke:

Am Anfang des Kreislaufs ist noch keine Wolke zu sehen. Deshalb muss sie zuerst versteckt werden!

Wenn die Wolke sich bildet, wird sie immer weniger durchsichtig. Wenn sie richtig dicht ist, beginnt es zu regnen.

Siehst du die Schleife? Was macht sie?

Wenn es regnet, löst sich die Wolke langsam wieder auf.

Regen:

Am Anfang des Kreislaufs ist noch kein Regen zu sehen. Deshalb muss er zuerst versteckt werden!

Wenn es regnet, fallen viele Wasser-tropfen vom Himmel. Damit man das gut sehen kann, wechseln die Kostüme des Regens.

Siehst du die Schleife? Was macht sie?

Wenn es fertig geregnet hat, startet der Wasserkreislauf von vorne!

Die auf diesem Arbeitsblatt verwendeten Bilder sind Screenshots der Webseite: https://scratch.mit.edu/projects/editor/
Wir empfehlen diese Webseite für vertiefende Übungen im Programmieren in Scratch.

Name: _____

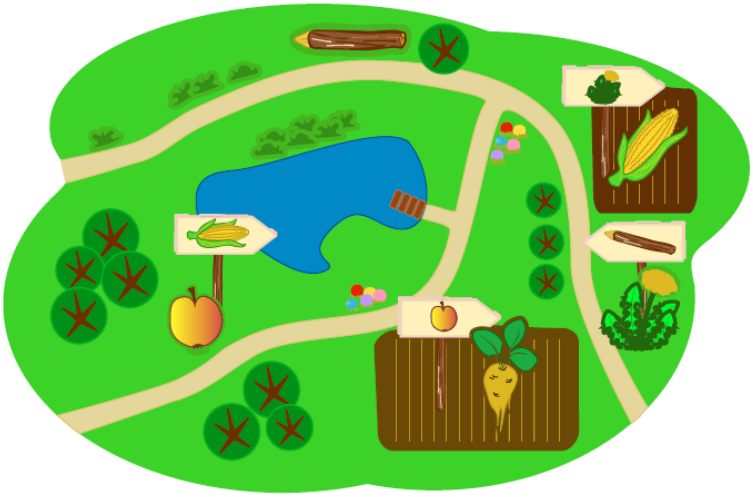
Übungsblatt 5

1. Die vegetarische Schnitzeljagd

Hase Jens möchte einen Bibersalat machen. Jens muss fünf Zutaten in die Schüssel tun.

In einem Garten findet Jens die Zutaten. Bei jeder Zutat zeigt ein Schild, welche Zutat er als nächste in die Schüssel tun muss. Zum Beispiel muss er nach dem Apfel den Mais in die Schüssel tun.

Welche Zutat muss Jens als erste in die Schüssel tun?

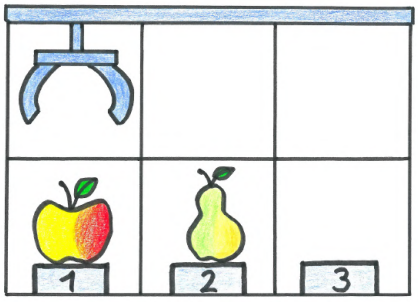


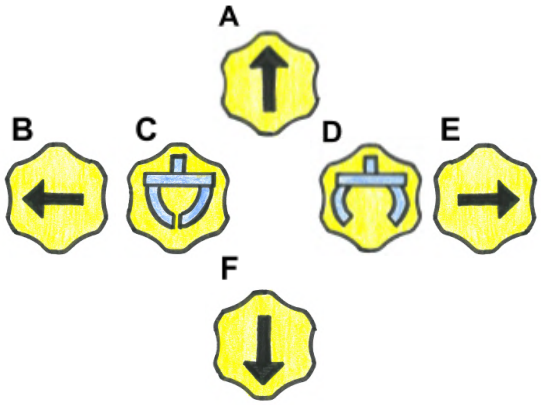
A) ☐ B) ☐ C) ☐ D) ☐ E) ☐

2. Der Automat

Der Automat kann Dinge greifen und sie bewegen. Und du sollst den Automaten bedienen!

Vertausche den Apfel und die Birne. Das heißt, der Apfel soll auf Platz 2 sein und die Birne soll auf Platz 1 sein.





Schreibe auf, in welcher Reihenfolge du die Knöpfe drücken musst!

Deine Antwort:

_____

_____

_____

_____

_____

_____

_____

3. Blumen malen

Du schreibst ein Programm, das Blumen malt. Dafür kannst du folgende Befehle benutzen:

- Mit dem Befehl **`Farbe`** kannst du festlegen, in welcher Farbe das Programm malt. Du kannst die folgenden Farben verwenden: **`gelb`**, **`orange`**, **`blau`**, **`rosa`**.
- Mit dem Befehl **`Blatt`** malt dein Programm ein Blütenblatt.
- Mit dem Befehl **`drehe nach rechts`** kannst du die Malrichtung bestimmen. Du kannst festlegen, um wieviel Grad (z.B. 90) die Malrichtung gedreht werden soll.
- Der Befehl **`wiederhole`** … **`mal [`*`Befehle`*`]`** führt Befehle in einer Schleife aus.

Beispiel - Diese Blume wird gemalt mit:

```
Farbe gelb
Blatt nach rechts 45
Farbe blau
wiederhole 3 mal
[Blatt nach rechts 90]
```

Welches Programm malt die folgende Blume?

A) ☐

```
Farbe gelb
wiederhole 2 mal [Blatt nach rechts 45]

Farbe orange
wiederhole 2 mal [Blatt nach rechts 90]

Farbe blau
wiederhole 2 mal [Blatt nach rechts 45]

Farbe rosa
wiederhole 2 mal [Blatt nach rechts 90]
```

B) ☐

```
Farbe gelb
Blatt nach rechts 45

Farbe orange
Blatt nach rechts 45

Farbe blau
Blatt nach rechts 45

Farbe rosa
Blatt nach rechts 45
```

C) ☐

```
Farbe gelb
wiederhole 2 mal [Blatt nach rechts 45]

Farbe orange
wiederhole 2 mal [Blatt nach rechts 45]

Farbe blau
wiederhole 2 mal [Blatt nach rechts 45]

Farbe rosa
wiederhole 2 mal [Blatt nach rechts 45]
```

D) ☐

```
Farbe gelb
wiederhole 2 mal [Blatt nach links 45]

Farbe orange
wiederhole 2 mal [Blatt nach links 45]

Farbe blau
wiederhole 2 mal [Blatt nach links 45]

Farbe rosa
wiederhole 2 mal [Blatt nach links 45]
```

Aufgabe 1, 2 und 3 auf diesem Übungsblatt basieren auf Aufgaben aus dem Informatik-Schülerwettbewerb "Informatik-Biber" der Initiative "Bundesweit Informatik fördern", https://bwinf.de/biber/downloads/
Wir empfehlen die auf dieser Webseite zusammengestellten Materialien als Knobelaufgaben für eine vertiefende Beschäftigung mit Problemstellungen der Informatik.

Lösung zu Übungsblatt 5

1. Die vegetarische Schnitzeljagd

Antwort E

2. Der Automat

Mögliche Antwort: F, C, A, E, E, F, D, A, B, F, C, A, B, F, D, A, E, E, F, C, A, B, F, D, A

Hier kann es mehrere richtige Lösungen geben! Es ist sowohl zulässig, wenn die Früchte abgesetzt werden, als auch, wenn sie fallen gelassen werden. Es ist egal, in welcher Reihenfolge die Früchte wohin gesetzt werden, solange sie am Ende an der richtigen Stelle sitzen.

3. Blumen malen

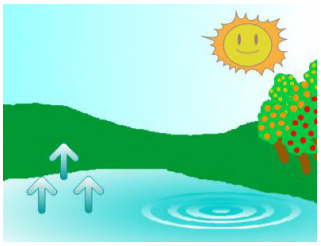
Antwort C

Zu Hause ausprobieren:

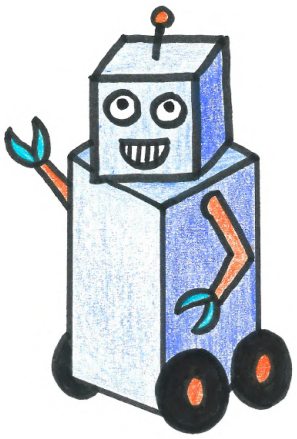
Hast du schon das **Nahrungsketten-Spiel** und den **Wasserkreislauf** schon fertig programmiert? Falls nicht, probiere es aus!

- Öffne das **Internet** (Mozilla Firefox, Chrome, Safari oder Internet Explorer).
- Schreibe Folgendes in die URL-Zeile:
  https://scratch.mit.edu/explore/projects
- Drücke auf **Enter**.
- Wähle ein Projekt aus und klicke es an.
- Klicke auf **Schau hinein**, um den Programm-Code zu sehen!
- Klicke auf die kleine **Weltkugel** eben dem Scratch-Logo oben links. Wähle **Deutsch** aus, um den Block-Code auf Deutsch zu sehen!
- Starte dein **eigenes Projekt**, indem du auf **Datei** klickst und dann auf **Neu**.

Hier kannst du nochmal den **Wasserkreislauf** anschauen:
https://scratch.mit.edu/projects/218891378/



Und hier kannst du nochmal das **Nahrungsketten-Spiel** anschauen:
https://scratch.mit.edu/projects/245186216/



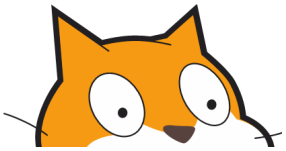
Viel Spaß beim Programmieren!

Die auf diesem Arbeitsblatt verwendeten Bilder (mit Ausnahme des Roboters) sind Screenshots der Webseite:
https://scratch.mit.edu/projects/editor/
Wir empfehlen diese Webseite für vertiefende Übungen im Programmieren in Scratch.

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Kurssitzung 6

- Mathematische Simulationen
- Übungsblatt 6
- Zu Hause ausprobieren 6



Lasst uns einen kleinen Rechner programmieren!

Wir wollen uns heute weiter mit der Scratch-Programmierumgebung vertraut machen!

- Öffne den Internet-Browser (Mozilla Firefox, Chrome, Safari oder Internet Explorer) auf deinem Desktop.
- Schreibe Folgendes in die URL-Zeile:

  https://scratch.mit.edu/projects/editor
- Drücke auf Enter.
- Starte dein eigenes Projekt.

Im Flohmarkt: Die Mathematik hinter dem Einkaufen

Lasst uns ein Scratch-Programm erstellen, das simuliert, wie ein Flohmarkt funktioniert!

- **Wähle diesen Hintergrund aus der Scratch-Bibliothek aus.**

- **Wähle noch zwei Hintergründe.**

Dieser Hintergrund wird als Spielende-Bild angezeigt, wenn die Zeit abgelaufen ist.

Dieser Hintergrund wird als "Dankeschön"-Bild angezeigt.

- **Wähle die folgenden Objekte aus der Scratch-Bibliothek aus:**

- **Einen Kunden oder eine Kundin**

  Klicke dafür auf , um eine Figur zu malen. Lasse die Figur aber leer!
  Die Kundin oder der Kunde ist ein "Platzhalter" für dich, deshalb muss man die Figur nicht sehen können, denn du sitzt ja vor dem Computer!

- **2 Musikinstrumente**

  Der Preis für ein Musikinstrument ist 10 €.

- **2 unterschiedliche Bälle**

  Der Preis für einen Ball ist 3 €.

- **Einen "Einkauf abgeschlossen"-Button**

  Suche dafür eine passende Figur aus und schreibe den Text "Einkauf abgeschlossen" in die Figur.

- **Verteile alle Produkte in passender Größe auf dem Hintergrund.**

Erzeuge zwei Variablen, eine für das Taschengeld und eine für das Geld, das noch übrig ist.

Taschengeld ist das Geld, das du am Anfang dabei hast. Das Taschengeld ändert sich nie, während du einkaufst.

Geld übrig ist am Anfang des Einkaufs gleich viel wie Taschengeld. Immer, wenn du etwas kaufst, wird der Preis des gekauften Produkts von Geld übrig abgezogen.

Erzeuge eine Variable für einen Timer (Stoppuhr). Der Timer speichert, wieviel Zeit noch bleibt, bis der Flohmarkt schließt.

Setze alle drei Zähler (Geld übrig, Taschengeld, Timer) in eine Ecke des Bildschirms. Dann kannst du sie während des Einkaufens immer sehen.

Erzeuge jetzt den Block-Code!
Wähle immer das passende Objekt aus, gehe zu "Skripte" und erstelle den Code!

Alle Produkte, die man kaufen kann:
Die Produkte erscheinen, wenn das Programm startet.

Achtung: Die Geld übrig-Variable reagiert nicht genau gleich auf alle Produkte! Diese Variable kann auch negativ sein, z.B. -3. ***Warum?***

Wenn der Kunde oder die Kundin fertig eingekauft hat und der Hintergrund sich zu "Vielen Dank!" ändert, sollen alle Produkte verschwinden.

Auch wenn der Hintergrund sich zu "Leider ist der Flohmarkt geschlossen" ändert, sollen alle Produkte verschwinden.

Der Kunde oder die Kundin:

Setze alle Variablen auf bestimmte Startwerte.

- *Lasse den normalen Hintergrund erscheinen.*
- *5 Minuten mal 60 Sekunden = 300 Sekunden*
- *20 € ist das Taschengeld, das der Kunde oder die Kundin am Anfang zur Verfügung hat.*
- *20 € ist zu Spielbeginn auch das Geld, das noch übrig ist.*

Programmiere den Timer als Schleife!

Wenn die Zeit abgelaufen ist, ändert sich der Hintergrund zu "Leider ist der Flohmarkt geschlossen" und das Programm endet.

"Einkauf abgeschlossen"-Button:

- *Entscheide, wann der Button erscheint und wann er verschwinden soll.*
- *Ändere den Hintergrund zu "Vielen Dank", wenn der Button angeklickt wird.*

Teste jetzt dein Programm!

- *Was fällt dir auf?*
- *Kann Geld ein negativer Betrag sein?*
- *Wie könntest du das Programm so ändern, dass das Geld nicht mehr negativ werden kann?*

Du musst eine Bedingung angeben, dass du genug Geld haben musst, um etwas zu kaufen. Wenn du nicht genug Geld hast, bekommst du eine Nachricht.

1. **Du musst festlegen, an welcher Stelle dieses Skript geschrieben werden soll.**
   z.B. für jedes der Produkte? Oder für den Kunden?

2. **Wie muss diese Bedingung aufgebaut sein?**
   z.B. nur mit "falls"? Oder mit "falls … sonst"?

3. **Überprüfe den folgenden Code. Verstehst du, was dieser Code macht und an welcher Stelle er im Skript stehen sollte? Was muss in die leeren weißen Felder? Ist dieser Code für alle Objekte gleich?**

- *Weshalb steht in der Bedingung ein "oder"?*
- *Weshalb gibt es zwei Bedingungen?*
- *Was macht der "ändere"-Befehl?*
- *Schreibe und teste deinen Code!*

4. **Versuche den Code aus dem letzten Schritt zu folgendem Code zu ändern:**

- *Wie ändert sich dein Programm durch diesen veränderten Code?*
- *Was ist der Unterschied zwischen "ändere (Variable) um …" und "setze (Variable) auf …"?*
- *Überprüfe nochmal deinen Code und mache dann auf der nächsten Seite weiter!*

Probiere die folgenden Aufgaben alleine aus, um dein Programm interessanter und anspruchsvoller zu machen!

1. Erzeuge 4 neue Produkte.

2. Ändere die Preise der alten Produkte.

3. Setze den Timer auf 3 Minuten.

4. Füge zu allen Produkten Preisschilder hinzu.

   z.B.: 

5. Erstelle eine Frage zu Beginn des Programms, die den Kunden oder die Kundin fragt, wieviel Geld er oder sie dabei hat. Speichere diesen Wert in der "Taschengeld"-Variablen ab.

6. Erstelle einen "Exit"-Button für das Spiel. Wenn dieser Button angeklickt wird, endet das ganze Programm.

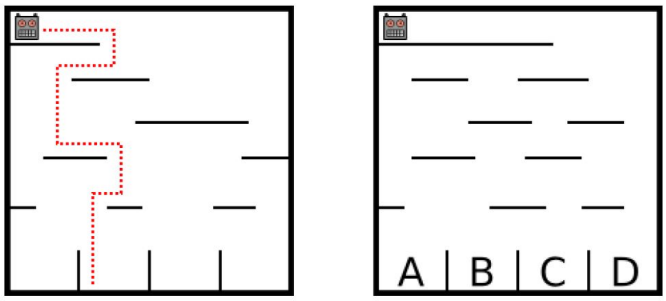
Die auf diesem Arbeitsblatt verwendeten Bilder sind Screenshots der Webseite: https://scratch.mit.edu/projects/editor/
Wir empfehlen diese Webseite für vertiefende Übungen im Programmieren in Scratch.

Name: _____

Übungsblatt 6

1. Der fallende Roboter

Ein Roboter läuft durch ein senkrecht stehendes Labyrinth. Er fällt dabei von einer Plattform auf eine darunter liegende. Nachdem er dort gelandet ist, ändert er seine Richtung. Am Schluss landet er ganz unten in einem der Fächer (siehe linkes Bild).



In welchem Fach landet der Roboter im rechten Bild?

A) ☐ **Er landet in Fach A.**

B) ☐ **Er landet in Fach B.**

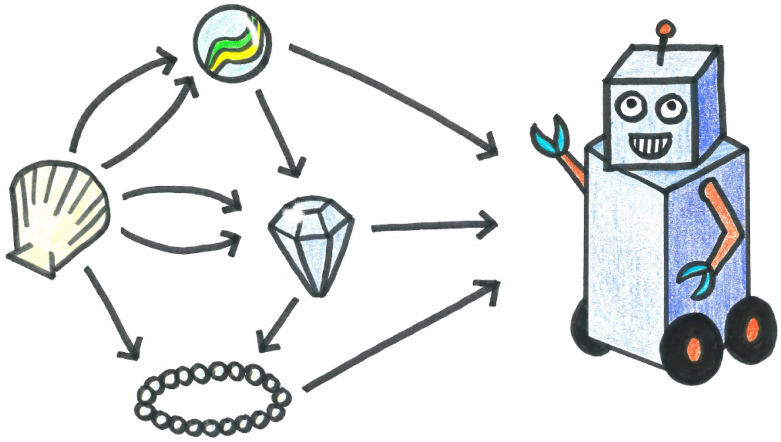
C) ☐ **Er landet in Fach C.**

D) ☐ **Er landet in Fach D.**

2. Der Händler

Du hast am Strand viele Muscheln gesammelt. Bei einem Händler kannst du sie gegen verschiedene Sachen eintauschen.

- Für zwei Muscheln bekommst du eine Murmel.
- Für zwei Muscheln und eine Murmel bekommst du einen Edelstein.
- Für eine Muschel und einen Edelstein bekommst du eine Perlenkette.
- Für eine Murmel, einen Edelstein und eine Perlenkette bekommst du einen Roboter.



Tipp: Schreibe auf, wieviele Muscheln diese Dinge Wert sind:

 = _____ Muscheln

 = _____ Muscheln

 = _____ Muscheln

Wieviele Muscheln musst du eintauschen, um einen Roboter zu bekommen?

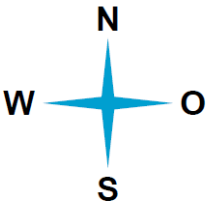
Deine Antwort: _____

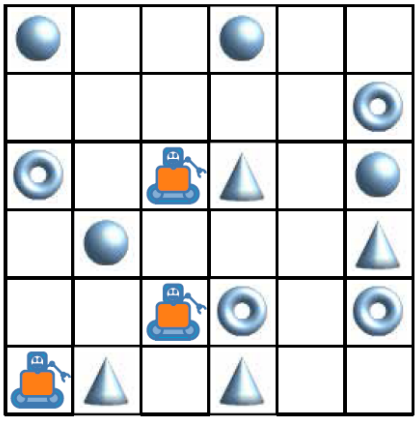
3. Das Roboter-Team

Drei Roboter arbeiten als Team zusammen. Du kannst das Team mit Richtungs-befehlen steuern: N, S, O oder W. Mit einem Richtunsbefehl steuerst du alle drei Roboter gleich: Um ein Feld weiter in diese Richtung.

Du sollst die Roboter zu den Dingen steuern, die sie am Ende nehmen sollen. Damit sie nichts Falsches nehmen, musst du sie vorher um andere Dinge herum steuern.

Ein **Beispiel**: Du steuerst die Roboter mit diesen Befehlen: N, N, S, S, O.
Dann nehmen die Roboter am Ende zwei Kegel und einen Ring.





Die Roboter sollen einen Ball, einen Ring und einen Kegel nehmen.

Mit welchen Befehlen musst du sie steuern?

A) ☐ **N, O, O, O**

B) ☐ **N, O, O, S, O**

C) ☐ **N, N, S, O, N**

D) ☐ **N, O, O, S, W**

Aufgabe 1, 2 und 3 auf diesem Übungsblatt basieren auf Aufgaben aus dem Informatik-Schülerwettbewerb "Informatik-Biber" der Initiative "Bundesweit Informatik fördern", https://bwinf.de/biber/downloads/
Wir empfehlen die auf dieser Webseite zusammengestellten Materialien als Knobelaufgaben für eine vertiefende Beschäftigung mit Problemstellungen der Informatik.

Lösung zu Übungsblatt 6

1. Der fallende Roboter

Antwort C

2. Der Händler

Antworten:

Murmel = 2 Muscheln

Edelstein = 4 Muscheln

Perlenkette = 5 Muscheln

Roboter = 11 Muscheln

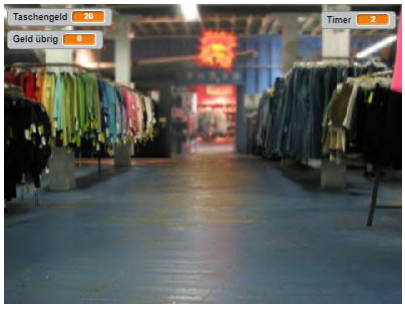
3. Das Roboter-Team

Antwort B

Zu Hause ausprobieren:

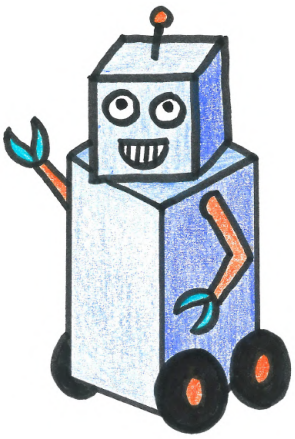
Hast du den **Flohmarkt** schon fertig programmiert? Falls nicht, probiere es aus!

- Öffne das **Internet** (Mozilla Firefox, Chrome, Safari oder Internet Explorer).
- Schreibe Folgendes in die URL-Zeile:
  https://scratch.mit.edu/explore/projects
- Drücke auf **Enter**.
- Wähle ein Projekt aus und klicke es an.
- Klicke auf **Schau hinein**, um den Programm-Code zu sehen!
- Klicke auf die kleine **Weltkugel** eben dem Scratch-Logo oben links. Wähle **Deutsch** aus, um den Block-Code auf Deutsch zu sehen!
- Starte dein **eigenes Projekt**, indem du auf **Datei** klickst und dann auf **Neu**.



Hier kannst du nochmal den **Flohmarkt** anschauen:
https://scratch.mit.edu/projects/220122727/

Viel Spaß beim Programmieren!



Die auf diesem Arbeitsblatt verwendeten Bilder (mit Ausnahme des Roboters) sind Screenshots der Webseite:
https://scratch.mit.edu/projects/editor/
Wir empfehlen diese Webseite für vertiefende Übungen im Programmieren in Scratch.

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

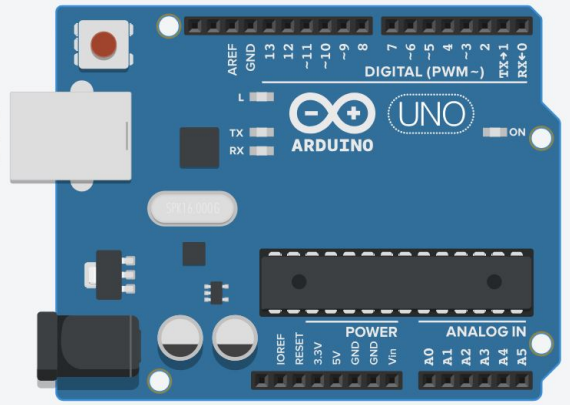
Kurssitzung 7

- Arduino-Einführung
- Arduino-Einführung für die Kursleitung 1
- Arduino-Projekte 1
- Übungsblatt 7
- Zu Hause ausprobieren 7

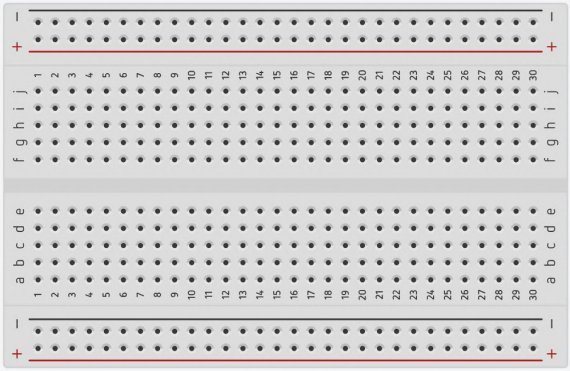
Der Arduino

Unser Arduino besteht aus…

… einer **Platine**:



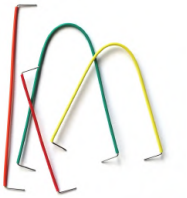
… einer **Steckplatine**:



… und verschiedenen **elektronischen Komponenten**:

(Sensoren, LED-Lichter, Kabel und Widerstände)

Die elektronischen Komponenten



Kabel kannst du benutzen, um *Komponenten* auf der **Steckplatine** miteinander oder mit der **Arduino-Platine** zu *verbinden*.

LED-Lichter leuchten, wenn *Strom* durch sie fließt.

Sensoren kannst du benutzen, wenn du möchtest, dass der Arduino auf etwas *reagieren* soll. Mit Sensoren kann der Arduino *etwas messen* (z.B. Licht, Bewegung oder Wärme) und die gemessenen Werte als *Ereignisse* verstehen.

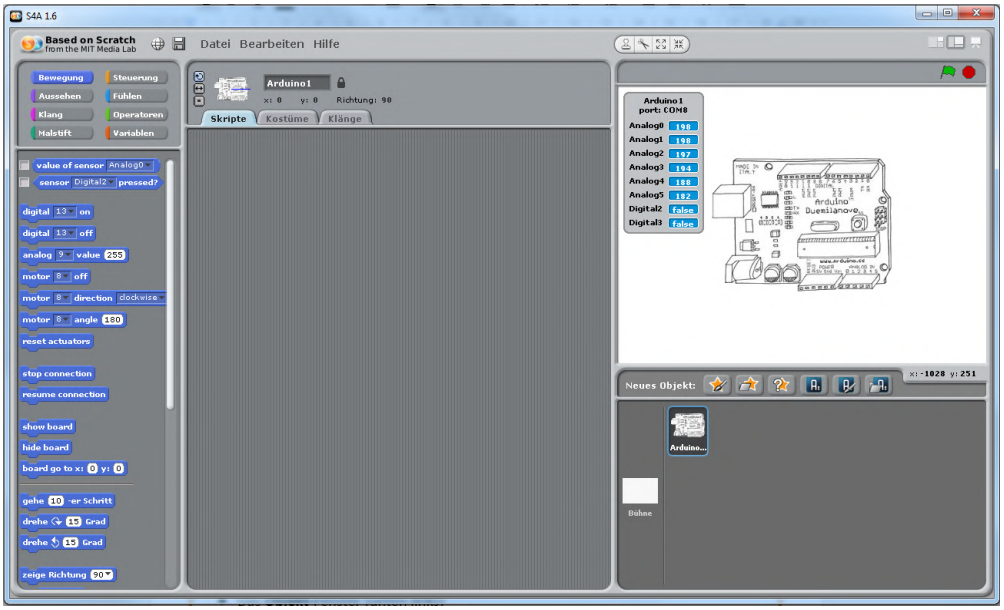
Taster sind eine Art von Sensoren. Den Knopf auf einem Taster kannst du drücken, um einen *Schaltkreis* zu schließen und *Strom fließen* zu lassen. Du kannst sie benutzen, wenn du etwas *ein- oder ausschalten* möchtest.

Lichtsensoren kannst du benutzen, um zu messen, wie hell ein Licht ist.

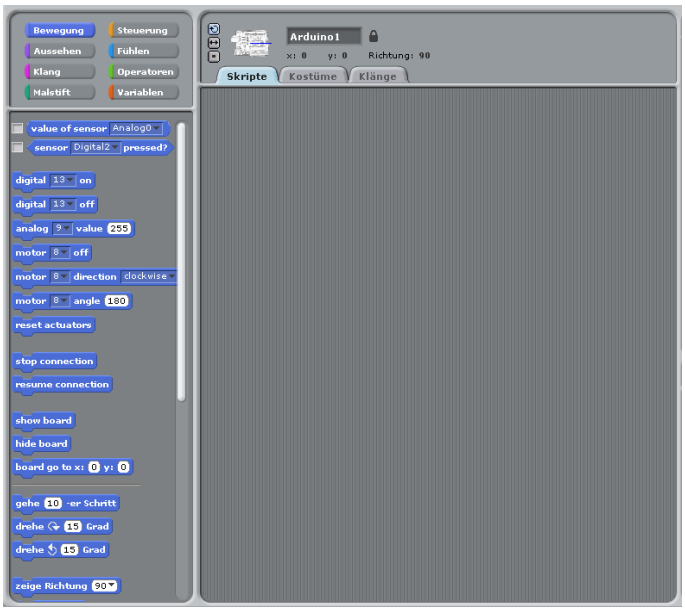
Widerstände muss man oft vor andere *Komponenten* schalten, um zu *begrenzen*, wieviel *Strom* zu diesen Komponenten fließen soll.

Die “Scratch für Arduino”-Umgebung

Die **“Scratch für Arduino”-Umgebung** besteht aus drei Fenstern:



1. Die **Block-Programmier**-Fenster (links und in der Mitte):

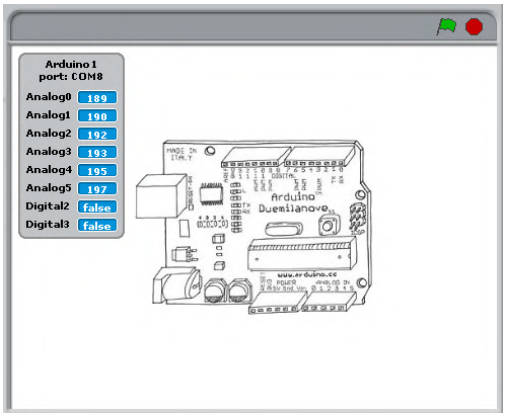


Hier programmieren wir unser **Skript**. Dafür benutzen wir bunte Blöcke.

2. Das **Objekt**-Fenster (unten rechts):

Hier bearbeiten unsere **Arduino-Platine**.

3. Und das **Vorschau**-Fenster (oben rechts):



Hier **testen** wir unseren **Code** und überprüfen, wie das Programm aussieht.

Du kannst den **Arduino programmieren**, damit er bestimmte Sachen macht. Dafür musst du verschiedene **Befehls-Blöcke kombinieren**. Es gibt die gleichen Blöcke wie in Scratch. Zusätzlich kannst du vier neue Blöcke verwenden:

Überprüft, ob ein **Sensor** (z.B. ein Knopf an einem Taster) **gedrückt** ist, der mit einem digitalen Port des Arduino verbunden ist (in diesem Fall der digitale Port 3)

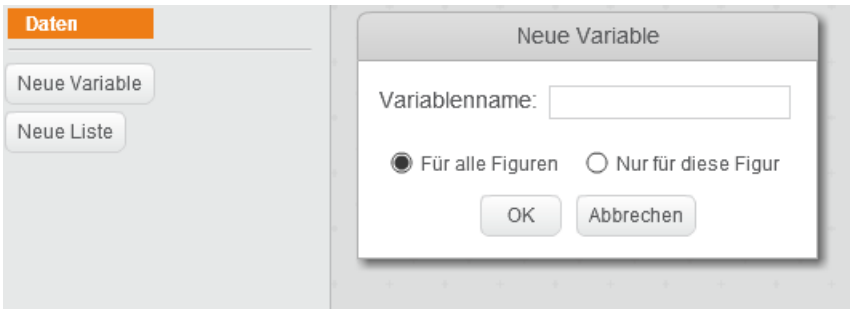
Dieser Block kann verwendet werden, um den **aktuellen Wert eines Sensors** (z.B. Lichtsensor) zu **ermitteln**, der mit in einem bestimmten Analogen Port verbunden ist (in diesem Fall der analoge Port 0)

Schaltet die Komponente **ein**, die mit einem bestimmten digitalen Port des Arduino verbunden ist (in diesem Fall der digitale Port 13)

Schaltet die Komponente **aus**, die mit einem bestimmten digitalen Port des Arduino verbunden ist (in diesem Fall der digitale Port 13)

Variablen

Die Variablen funktionieren genau wie in Scratch:

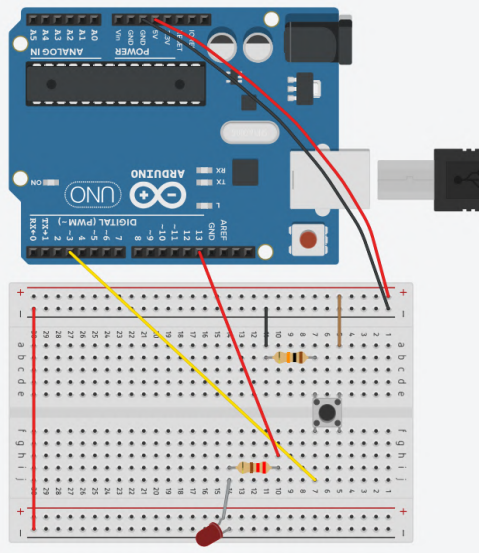


Jetzt nehmt euch das **Arbeitsblatt Arduino-Projekte 1** und fangt an zu programmieren!

Die auf diesem Arbeitssblatt verwendeten Bilder des Arduino-Aufbaus sind Screenshots der Webseite:
https://www.tinkercad.com/#/
Die auf diesem Informationsblatt verwendeten Bilder des S4A-Codes sind Screenshots aus S4A (Scratc for Arduino):
http://s4a.cat

Arduino-Einführung für die Kursleitung

Für den Lichttaster mit Knopf brauchen wir:

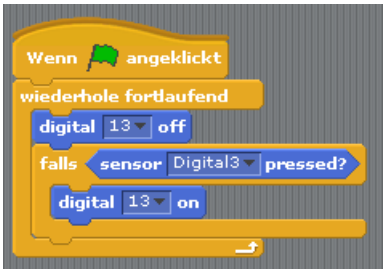


- 1 x Arduino
- 1 x Protoboard
- 1x Taster mit Knopf
- 1 x LED (rot)
- 1 x 220 Ω Widerstand (rot-rot-braun)
- 1 x 10 kΩ Widerstand (braun-schwarz-orange)
- genügend Kabel

Zu Beachten:

- **LED**: Der längere Draht zum Widerstand, der kürzere Draht zu -
- **Analoge und digitale Ports**: Auf vielen Computern werden nicht alle Ports des Arduino angezeigt. Falls dieses Problem auftritt, sollten die Komponenten, die mit diesen Ports verbunden sind, stattdessen in einen der vom Computer angezeigten Ports gesteckt werden.
  Wichtig: Dann muss auch der Code entsprechend geändert werden, damit die richtigen Ports angesteuert werden können (z.B. Port 10 statt Port 9, falls 9 nicht angezeigt wird).

Lösungsbeispiel:



Die auf diesem Informationsblatt verwendeten Bilder des Arduino-Aufbaus sind Screenshots der Webseite:
https://www.tinkercad.com/#/
Die auf diesem Informationsblatt verwendeten Bilder des S4A-Codes sind Screenshots aus S4A (Scratc for Arduino):
http://s4a.cat

1. Licht: Taster mit Knopf!

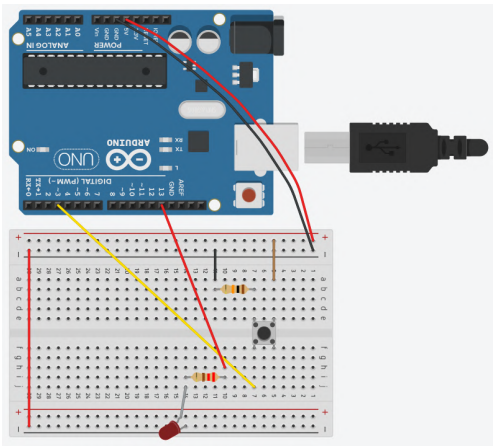
Lass deinen Arduino Lichter an- und ausschalten, wenn ein Knopf auf einem Taster gedrückt wird!

- ❑ Schau dir den Aufbau des Arduino an.
- ❑ Programmiere die Funktionalität des Arduino mit S4A.
- ❑ Verbinde das Arduino-Kit mit deinem PC oder Laptop und teste deinen Code!

Arten von Blöcken

- Steuerung
- Bewegung

Arduino Start-Aufbau



WICHTIG: Ändere den Aufbau des Arduino nur dann, wenn er nicht am PC eingesteckt ist!

Tipps

- Öffne die S4A-Umgebung und programmiere die Funktionalität dieses Arduino-Aufbaus.
- Starte den Arduino, wenn die grüne Flagge geklickt wird.

- Benutze eine Schleife, damit dein Code den Arduino die ganze Zeit auf dieselbe Art laufen lässt.

- Lege einen Startzustand für das LED-Licht fest: an/aus (= on/off). Dafür musst den richtigen digitalen Port ansteuern.

- Benutze eine Bedingung, um das LED-Licht einzuschalten, wenn der Knopf auf dem Taster gedrückt ist.

- Lege die Bedingung so fest, dass der Sensorzustand berücksichtigt wird.

- Gib mit den Befehlsblöcken an, was passieren soll, wenn die Bedingung erfüllt ist.

2. Morsecode!

Wir programmieren ein Morse-Licht zum Senden von geheimen Nachrichten!

- ❑ Benutze denselben Arduino-Aufbau wie in der letzten Übung.
- ❑ Programmiere die Funktionalität des Arduino mit S4A.
- ❑ Verbinde das Arduino-Kit mit deinem PC oder Laptop und teste deinen Code!

Arten von Blöcken

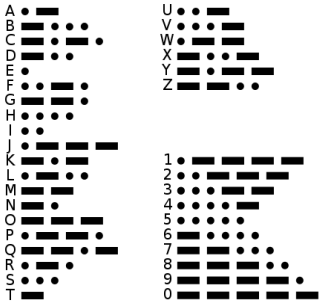
- Steuerung
- Bewegung

Schritte:

1. Ihr braucht Papier und Stift.
2. Legt fest: Eine/r in eurem Team kodiert und die/der andere dekodiert.
3. Wer kodiert, überlegt sich ein Wort und schreibt es in Morsecode auf Papier.
4. Wer kodiert hat, hält jetzt das Blatt, auf dem der Morsecode steht, so dass der oder die andere es nicht sehen kann. Dann führt die/der Kodierer/in den Morsecode mit dem Knopf auf dem Arduino aus, sodass das Licht blinkt.
   Denkt daran: **Ein Punkt heißt kurz drücken, ein Strich heißt lang drücken!**
   Und: **Zwischen den einzelnen Buchstaben eine größere Pause machen!**
5. Die/der Dekodierer/in muss jetzt das geheime Wort herausfinden! Dafür könnt ihr wieder Stift und Papier benutzen. Schaut auf das blinkende Licht und schreibt den Morsecode auf. Dekodiert den aufgeschriebenen Morsecode!
6. Tauscht die Zugreihenfolge und startet von vorne!

Der Morsecode

▬ Langes Drücken
● Kurzes Drücken



Beispiele

- „sos“ ●●●- - -●●●
- „heute“ ●●●● ● ●●- - ●

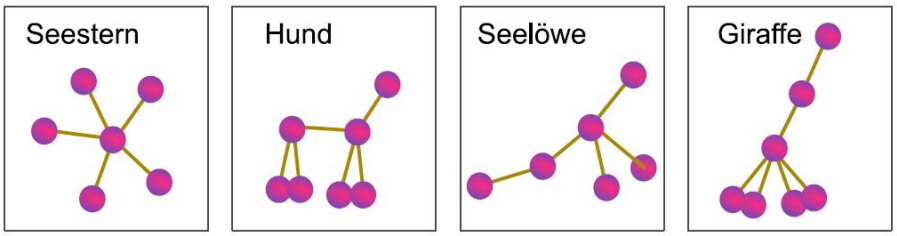
Die auf diesem Informationsblatt verwendeten Bilder des Arduino-Aufbaus sind Screenshots der Webseite:
https://www.tinkercad.com/#/
Die auf diesem Informationsblatt verwendeten Bilder des S4A-Codes sind Screenshots aus S4A (Scratc for Arduino):
http://s4a.cat

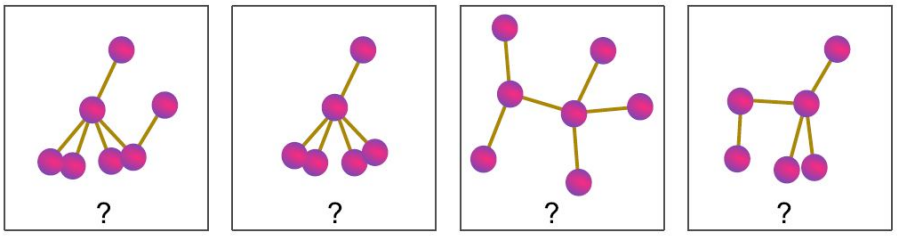
Name: _____

Übungsblatt 7

1. Knetetierchen

Yasmin hat aus Knetekugeln und Stäbchen vier verschiedene Knetetierchen gebastelt: Einen Seestern, einen Hund, einen Seelöwen und eine Giraffe.





Doch nun hat ihr kleiner Bruder mit den Knetetierchen gespielt.
Dabei hat jedes Tierchen eine neue Form bekommen.
Aber die Stäbchen stecken noch in den gleichen Kugeln wie vorher.

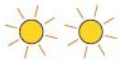
Was war was?

Male von jedem Knetetierchen oben eine Linie zu neuen Form unten.

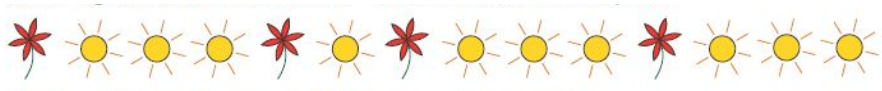
2. Blumen und Sonnen

Barbara hat zwei Stempel bekommen. Einer druckt eine Blume, der andere eine Sonne. Sie überlegt, wie sie nur mit Blumen und Sonnen ihren Namen stempeln kann.

Für verschiedene Buchstaben bestimmt sie verschiedene Folgen von Blumen und Sonnen:

Buchstabe	B	A	R	E	Y
Folge					

Ihren eigenen Namen “Barbara” muss sie dann so stempeln:



Nun stempelt Barbara den Namen eines ihrer Freunde:

Welchen Namen hat sie gestempelt?

A) ☐ **Abby**

B) ☐ **Anna**

C) ☐ **Barry**

D) ☐ **Ray**

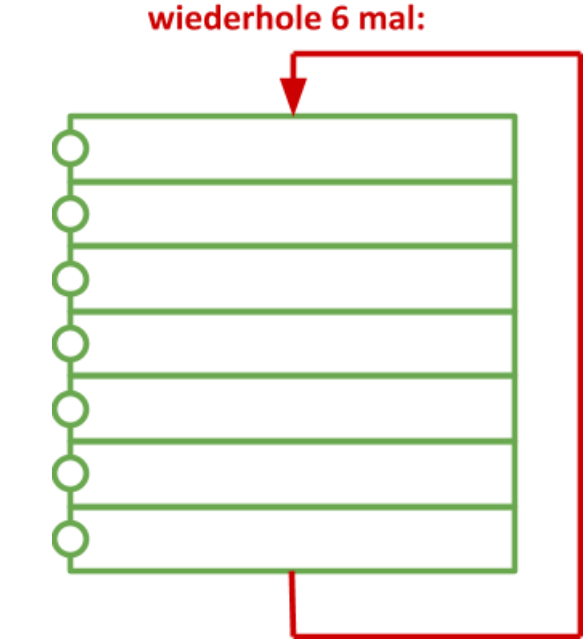
3. Die Musterschleife

Du willst ein Programm schreiben, das ein Muster malt. Dafür kannst du vier verschiedene Befehle benutzen. Ziehe eine Linie von dem Befehl, den du verwenden möchtest, zu der Stelle im Programm, an der er ausgeführt werden soll.

Du darfst alle Befehle auch mehrmals verwenden.

Die Befehle:

- **gehe ein Feld nach oben**
- **gehe ein Feld nach unten**
- **gehe ein Feld nach rechts**
- **gehe ein Feld nach links**



So soll das Muster aussehen:

Das Programm startet **auf** dem gelben Feld, das heißt dieses Feld ist schon gemalt.

Aufgabe 1, 2 und 3 auf diesem Übungsblatt basieren auf Aufgaben aus dem Informatik-Schülerwettbewerb "Informatik-Biber" der Initiative "Bundesweit Informatik fördern", https://bwinf.de/biber/downloads/
Wir empfehlen die auf dieser Webseite zusammengestellten Materialien als Knobelaufgaben für eine vertiefende Beschäftigung mit Problemstellungen der Informatik.

Lösung zu Übungsblatt 7

1. Knetetierchen

Antwort (von links nach rechts): Giraffe, Seestern, Hund, Seelöwe

2. Blumen und Sonnen

Antwort A

3. Die Musterschleife

Antwort:

gehe ein Feld nach rechts

gehe ein Feld nach unten

gehe ein Feld nach rechts

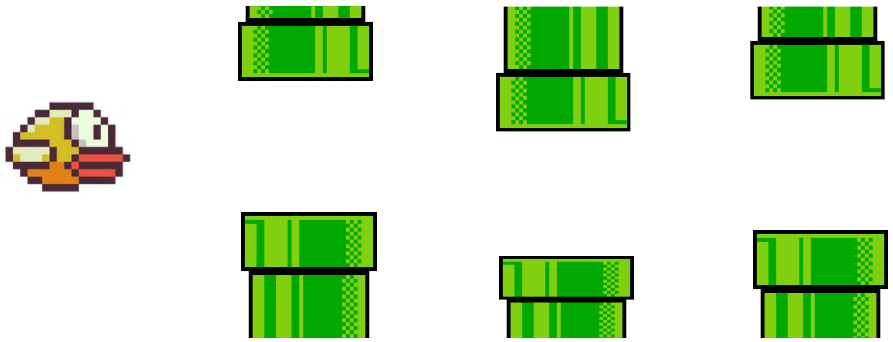
gehe ein Feld nach rechts

gehe ein Feld nach oben

gehe ein Feld nach rechts

Zu Hause ausprobieren:

Programmiere das Spiel **Flappy Bird** und füge immer schwierigere Level hinzu!

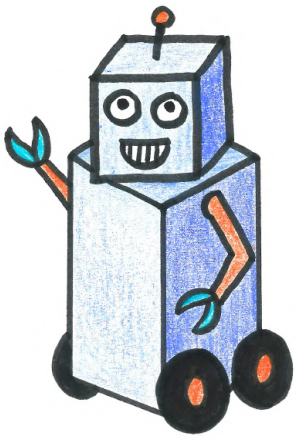


- Öffne das **Internet** (Mozilla Firefox, Chrome, Safari oder Internet Explorer).
- Schreibe Folgendes in die URL-Zeile:

https://studio.code.org/s/flappy

- Drücke auf **Enter**.
- Klicke auf **Weiter** und fang an zu programmieren!
- Wer hat **bis zur nächsten Kurssitzung die meisten Level** programmiert?

Viel Spaß beim Programmieren!



EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

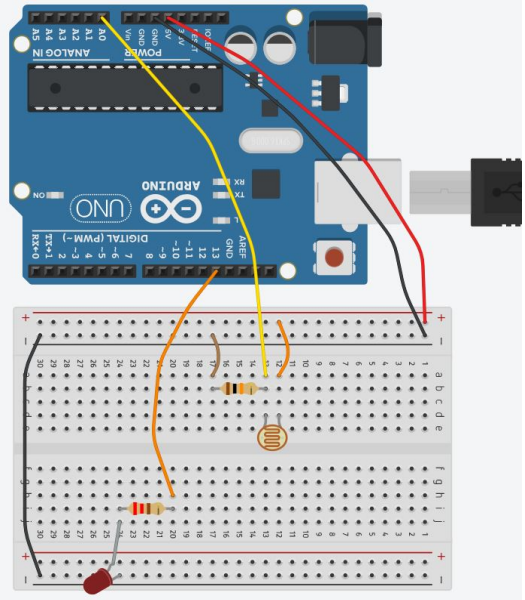
Kurssitzung 8

- Arduino-Einführung für die Kursleitung 2
- Arduino-Projekte 2
- Übungsblatt 8
- Zu Hause ausprobieren 8

Arduino-Einführung für die Kursleitung

Für den Lichtsensor brauchen wir:

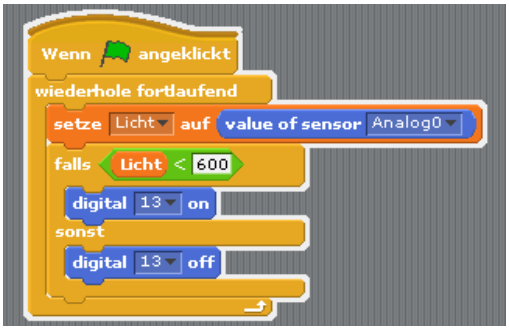
- 1 x Arduino
- 1 x Protoboard
- 1 x Photoresistor (Lichtsensor)
- 1 x LEDs (rot)
- 1 x 220 Ω Widerstand (rot-rot-braun)
- 1 x 10 kΩ Widerstand (braun-schwarz-orange)
- genügend Kabel

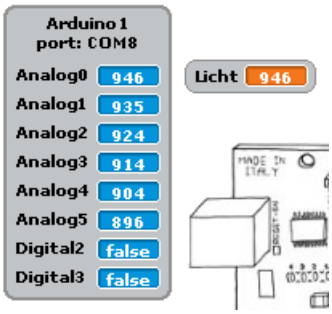


Zu Beachten:

- **LED**: Der längere Draht zum Resistor, der kürzere Draht zu -
- **Analoge und digitale Ports**: Auf vielen Computern werden nicht alle Ports des Arduino angezeigt. Falls dieses Problem auftritt, sollten die Komponenten, die mit diesen Ports verbunden sind, stattdessen in einen der vom Computer angezeigten Ports gesteckt werden.
  Wichtig: Dann muss auch der Code entsprechend geändert werden, damit die richtigen Ports angesteuert werden können (z.B. Port 10 statt Port 9, falls 9 nicht angezeigt wird).
- **Lichtsensor und Licht-Variable**: Zuerst muss eingelesen werden, welcher Wert für den Port angezeigt wird, der mit dem Lichtsensor verbunden ist (im Lösungsbeispiel Analog0). Die LED soll dann eingeschaltet werden, wenn es dunkler wird, z.B. wenn die Umgebung des Sensors abgedunkelt wird, indem man eine Hand darüber hält. Für die Bedingung (Licht < …) sollte ein Wert gewählt werden, der unter dem Wert liegt, der bei normaler Raumhelligkeit für den Port angezeigt wird, der aber unterschritten wird, wenn man z.B. eine Hand über den Sensor hält.

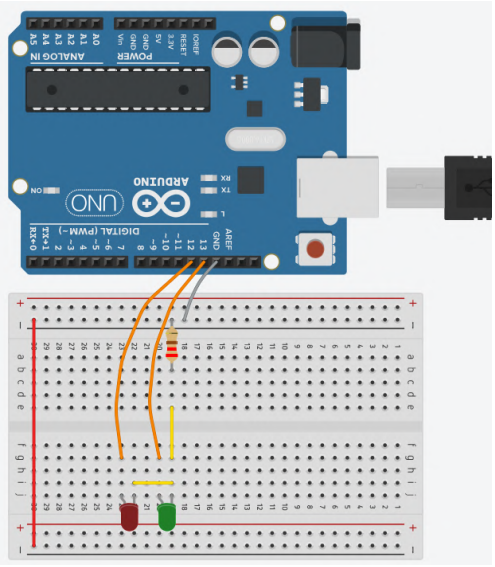
Lösungsbeispiel:





Für die **Fußgängerampel** brauchen wir:

- 1 x Arduino
- 1 x Protoboard
- 2 x LEDs (rot und grün)
- 1 x 220 Ω Widerstände (rot-rot-braun)
- genügend Kabel



Lösungsbeispiel

Die auf diesem Informationsblatt verwendeten Bilder des Arduino-Aufbaus sind Screenshots der Webseite:
https://www.tinkercad.com/#/
Die auf diesem Informationsblatt verwendeten Bilder des S4A-Codes sind Screenshots aus S4A (Scratc for Arduino):
http://s4a.cat

1. Lichtsensor!

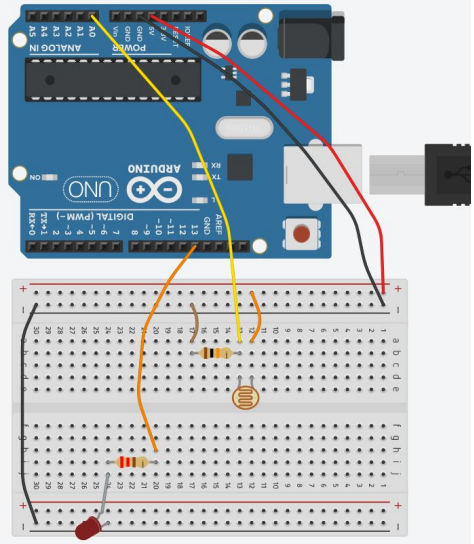
Programmiere einen Sensor, der auf Licht reagiert und schalte Lichter automatisch an und aus!

- ❑ Schau dir den Arduino-Aufbau an.
- ❑ Programmiere die Funktionalität des Arduino mit S4A.
- ❑ Verbinde das Arduino-Kit mit deinem PC oder Laptop und teste deinen Code!

Arten von Blöcken

- Steuerung
- Bewegung
- Operatoren
- Variablen

Arduino Start-Aufbau



WICHTIG: Ändere den Aufbau des Arduino nur dann, wenn er nicht am PC eingesteckt ist!

Tipps

- Öffne die S4A-Umgebung und programmiere die Funktionalität dieses Arduino-Aufbaus.
- Starte den Arduino, wenn die grüne Flagge geklickt wird.

- Benutze eine Schleife, damit dein Code den Arduino die ganze Zeit auf dieselbe Art laufen lässt.

- Erzeuge eine Variable für das Licht, das vom Sensor gemessen wird. Lege einen Startwert (= value) für den Lichtsensor fest.

- Benutze eine Schleife, damit dein LED-Licht an- und ausgeht.
- Benutze eine Bedingung, die überprüft, wie hell das Licht ist. Probiere unterschiedliche Zahlen für die Lichtempfindlichkeit aus.

- Gib mit den Befehlsblöcken an, was passieren soll, wenn die Bedingung erfüllt ist.

Experimentiere mit deinem Code!

- Probiere unterschiedliche Bedingungen für die Lichtempfindlichkeit aus.
- Drehe die Funktion deines Programms um!

2. Fußgängerampel!

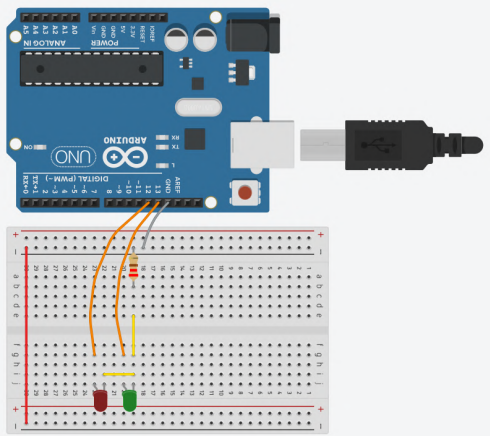
Lasst uns eine Ampel programmieren, die eine **Fußgängerampel** simuliert!

- ❑ Schau dir den Aufbau des Arduino an.
- ❑ Programmiere die Funktionalität des Arduino mit S4A.
- ❑ Verbinde das Arduino-Kit mit deinem PC oder Laptop und teste deinen Code!

Arten von Blöcken

- Steuerung
- Bewegung

Arduino Start-Aufbau



WICHTIG: Ändere den Aufbau des Arduino nur dann, wenn er nicht am PC eingesteckt ist!

Tipps

- Öffne die S4A-Umgebung und programmiere die Funktionalität dieses Arduino-Aufbaus.
- Starte den Arduino, wenn die grüne Flagge geklickt wird. Du brauchst eine Schleife, ein paar on/off-Befehle und Befehle, um zu warten oder die Sequenz zu stoppen.

Kleine Variantenübungen

1. Programmiere den Arduino als Fußgängerampel, die immer abwechselnd rot und grün leuchtet. Welches Farbe sollte länger leuchten?
2. Programmiere eine Füßgängerampel, die sich immer wiederholt, bei der Rot 6 Sekunden lang leuchtet und Grün 2 Sekunden.
3. Programmiere eine Fußgängerampel, die sich nur fünfmal wiederholt. Rot soll 10 Sekunden lang leuchten und Grün 3 Sekunden.

Die auf diesem Arbeitsblatt verwendeten Bilder des Arduino-Aufbaus sind Screenshots der Webseite:
https://www.tinkercad.com/#/
Die auf diesem Informationsblatt verwendeten Bilder des S4A-Codes sind Screenshots aus S4A (Scratc for Arduino):
http://s4a.cat

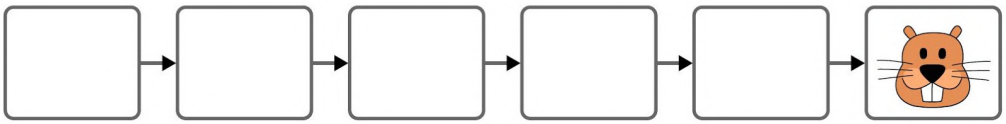
Name: _____

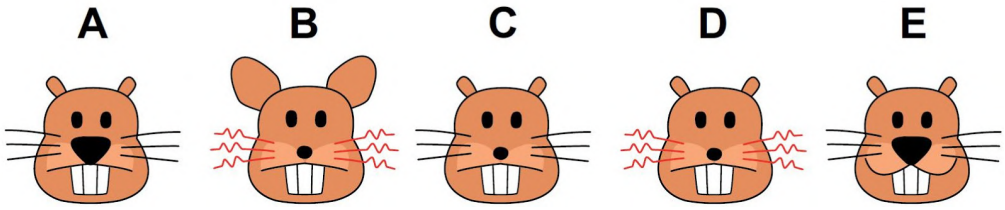
Übungsblatt 8

1. Biber-Animation

Aus sechs Biber-Bildern soll eine Animation entstehen. Dazu müssen die Bilder so angeordnet werden, dass sich von einem Bild zum nächsten nur ein Merkmal ändert: Barthaare, Mund, Nase, Ohren und Zähne. Das letzte Bild steht schon fest.

Schreibe die richtigen Buchstaben in die freien Kästchen!



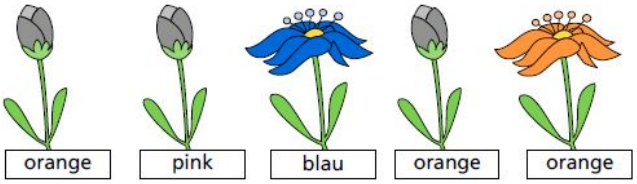


2. Blumenfarben

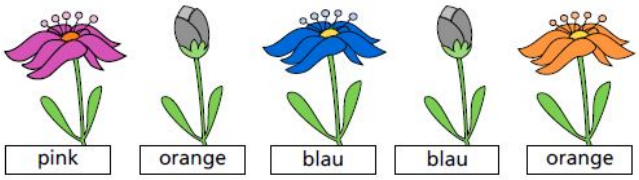
Julia spielt ein Computerspiel. Sie soll die Farben von fünf Blumen raten. Mögliche Farben sind blau, orange oder pink. Die Blumen sind geschlossen.

Julia rät zuerst die Farben so: orange, pink, blau, orange und orange.

Sie hat zweimal richtig geraten. Die richtigen Blumen gehen auf:



Für die falschen Blumen rät sie andere Farben. Nun ist noch eine Blume richtig:



Welche Farben muss sie raten, damit die beiden letzen Blumen sich öffnen?

pink, _____ , blau, _____ , orange

3. Magischer Roboter

Der magische Roboter bewegt sich auf einer langen, geraden Reihe von Feldern.

Er wird durch Symbole kommandiert.

Der magische Roboter bewegt sich ein Feld vorwärts. 

Der magische Roboter zaubert auf dem Feld vor sich einen Frosch herbei. 

Um ein Kommando mehrmals hintereinander zu geben, werden Zahlen benutzt:

Der magische Roboter führt ein Kommando (hier z.B. ein Feld vorwärts) vier Mal aus. Er bewegt sich also 4 Felder vorwärts.

Wenn mehr als ein Kommando mehrmals gegeben werden soll, werden Klammern benutzt:

Der magische Roboter führt die in Klammern gesetzten Kommandos (hier z.B. ein Feld vorwärts, dann noch ein Feld vorwärts) vier Mal aus. Er bewegt sich also acht Felder vorwärts.

Der magische Roboter darf sich auf ein Feld bewegen, auf dem ein Frosch sitzt.

Welche dieser Symbolfolgen bringt den magischen Roboter dazu, auf vier nebeneinanderliegenden Feldern vier Frösche herbeizuzaubern?

A) ☐ 

C) ☐ 

B) ☐ 

D) ☐ 

Aufgabe 1, 2 und 3 auf diesem Übungsblatt basieren auf Aufgaben aus dem Informatik-Schülerwettbewerb "Informatik-Biber" der Initiative "Bundesweit Informatik fördern", https://bwinf.de/biber/downloads/ .
Wir empfehlen die auf dieser Webseite zusammengestellten Materialien als Knobelaufgaben für eine vertiefende Beschäftigung mit Problemstellungen der Informatik.

Lösung zu Übungsblatt 8

1. Biber-Animation

Antwort B, D, C, A, E

2. Blumenfarben

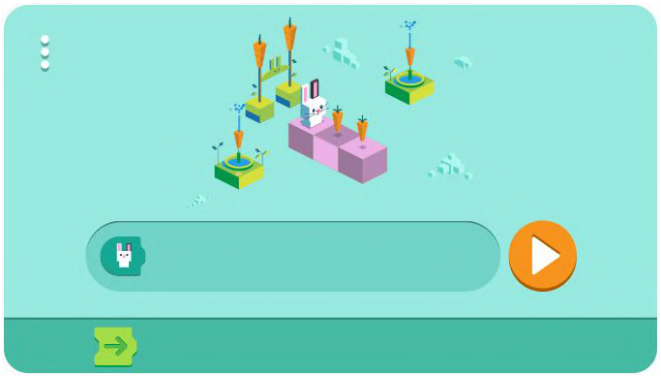
Antwort pink, blau, blau, pink, orange

3. Magischer Roboter

Antwort B

Zu Hause ausprobieren:

Möchtest du **mit einem Spiel testen, wie gut du schon programmieren kannst**? Dann kannst du das folgende Spiel ausprobieren, in dem du **klug denken** und **richtig programmieren** musst, **um von Level zu Level aufzusteigen**:

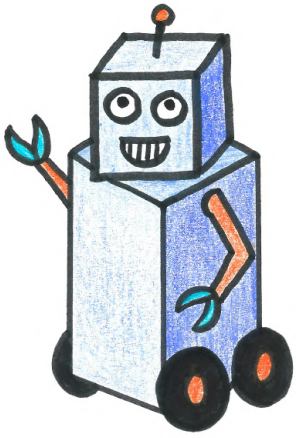


- Öffne das **Internet** (Mozilla Firefox, Chrome, Safari oder Internet Explorer).
- Schreibe Folgendes in die URL-Zeile:

  https://www.google.com/doodles/celebrating-50-years-of-kids-coding

- Klicke auf das **Start-Symbol** (das weiße Dreieck) und fang an zu spielen!
- Wer hat **bis zur nächsten Kurssitzung das höchste Level** erreicht?

Viel Spaß beim Spielen!



EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Kurssitzung 9

- Roberta-Einführung
- Übungsblatt 9
- Zu Hause ausprobieren 9

Lasst uns anfangen!

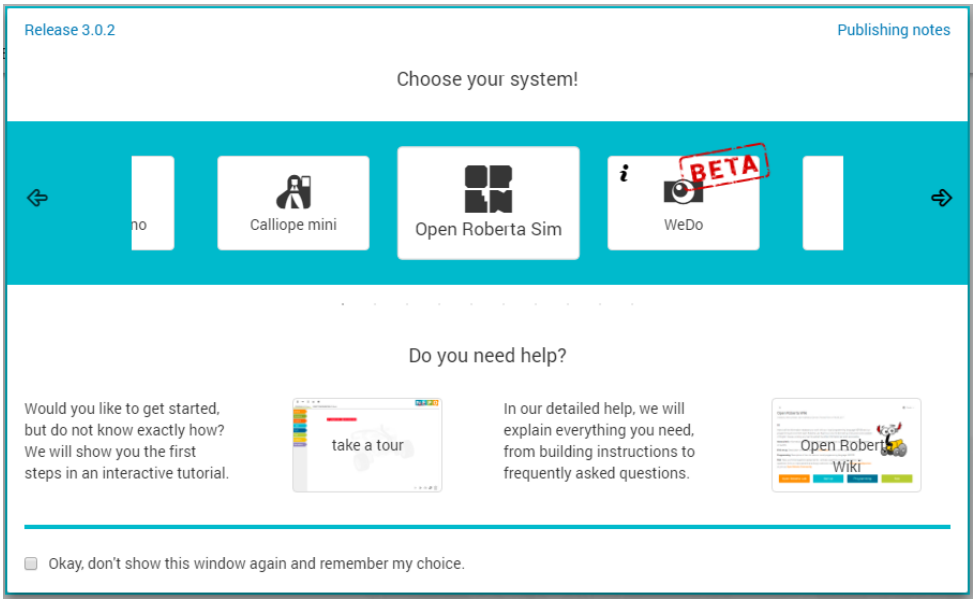
Probiert die folgenden Schritte aus. Vergesst nicht, als Team zu arbeiten!

- Tippt folgende Adresse in die URL-Zeile ein:

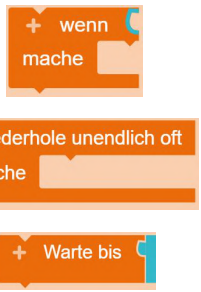
  https://lab.open-roberta.org/#overview

- Drückt auf Enter!
- Jetzt seht ihr das **Open Roberta Lab**. Das ist die neue **Programmierumgebung**, in der wir ab heute arbeiten.
- **Folgt der Tour** durch das Open Roberta Lab.
- **Lest die Texte** und klickt dann auf "**Weiter**"!

Wir werden mit dem "**Open Roberta Sim**"-System arbeiten:



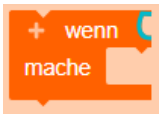
Das sind die Blöcke, die ihr heute benutzen könnt:

Aktion	Kontrolle	Farben
		
Sensoren	**Logik**	**Mathematik**
		

1 Roberta soll tanzen

Der Farbsensor

Mit dem Farbsensor kann Roberta Farben erkennen, die genau unter ihr in dem Kreis sind.



Der Befehl "wenn … mache …"

Mit diesem Befehl könnt ihr eine **Bedingung** aufstellen. Ihr könnt Roberta sagen: *Wenn etwas passiert, dann soll sie etwas machen.*

Wenn die Bedingung erfüllt wird, die ihr gestellt habt, dann soll Roberta das machen, was ihr möchtet.

Dazu müsst ihr unter 

den Befehl 

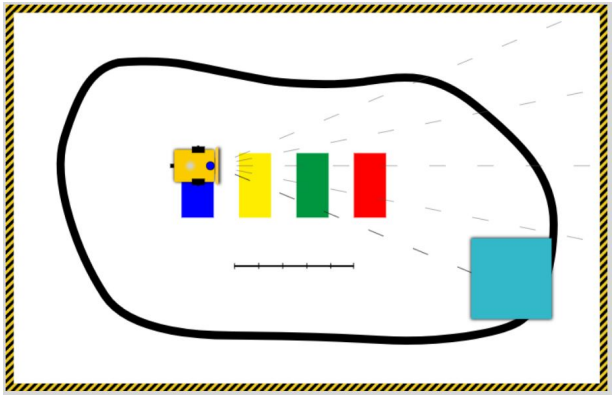
benutzen, um darin eure Bedingung zu formulieren. Ihr könnt zum Beispiel sagen, "**Wenn** der Farbsensor etwas Rotes sieht". Dazu müsst ihr unter 

den Befehl 

auswählen und vor das Gleichheitzeichnen einsetzen. Unter **Farbe** wählt ihr dann die Farbe Rot aus und zieht sie ins Feld hinter das Gleichheitszeichen.

Neben "**mache …**" könnt ihr eine Aktion einfügen, zum Beispiel vorwärts fahren.

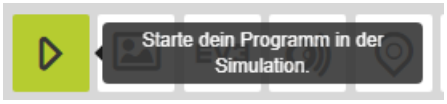
Für diese Aufgabe nutzen wir den folgenden Hintergrund:



Zieht Roberta als erstes einmal mit dem Farbensensor auf die schwarze Linie. Dann bewegt sie langsam runter auf die weiße Fläche. Was fällt euch auf?

Probiert es aus und lasst Roberta mit Hilfe von **“wenn … mache …”** *geradeaus fahren, sobald sie etwas Blaues sieht.*

Tipp: Zieht Roberta mit der Maus auf das blaue Feld, und drücke dann noch einmal Start:



Nun versucht mithilfe der neuen Befehle und den vier Farben, die ihr auf dem Hintergrund seht, Roberta tanzen zu lassen! Beginnt ganz einfach. Roberta soll vorwärts fahren, aber was soll sie machen, wenn sie auf die bunten Felder kommt?
Überlegt euch für jede Farbe eine andere Tanzbewegung, zum Beispiel:

Dann soll Roberta zur nächsten Farbe weiterfahren und dort eine neue Tanzbewegung ausführen!

Dieses Arbeitsblatt basiert auf: “Roberta EV3 Simulator. Einstieg” und “Roberta EV3 Simulator. Station 3: Der schwarzen Linie folgen”, Tobias Rütten, Team InfoSphere-Schülerlabor Informatik der RWTH Aachen,
http://schuelerlabor.informatik.rwth-aachen.de/module/open-roberta
Die auf diesem Arbeitsblatt verwendeten Bilder sind Screenshots der Webseite: http://lab.open-roberta.org
Wir empfehlen die auf diesen Webseiten zusammengestellten Materialien für eine vertiefende Beschäftigung mit dem Programmieren in Open Roberta.

Name: _____

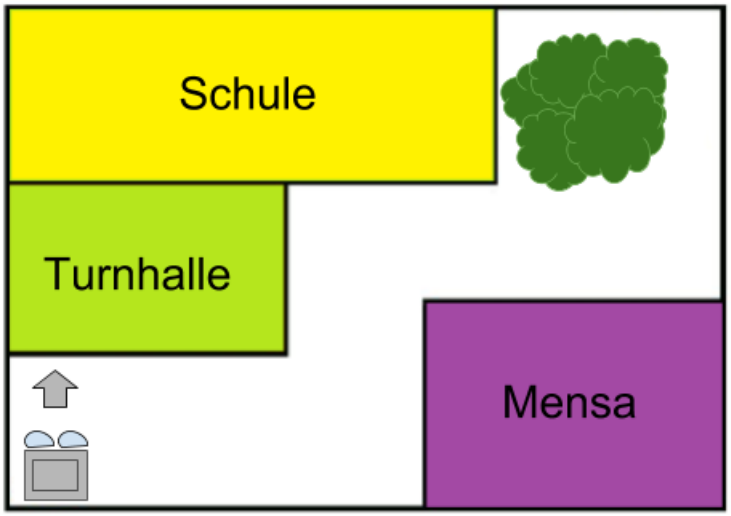
Übungsblatt 9

1. Roboter spielen

Die Kinder spielen Roboter. Lucky ist der Roboter und hört nur noch auf diese drei Kommandos: ***Vor! Links!*** und ***Rechts!***

Rufen die Kinder ***Vor!***, dann geht Lucky vorwärts, bis er an ein Gebäude stößt.
Rufen die Kinder ***Links!***, dann dreht sich Lucky nach links.
Rufen die Kinder ***Rechts!***, dann dreht er sich nach rechts.

Lucky steht in einer Schulhofecke. Man sieht ihn im Bild von oben. Er schaut in Richtung der Turnhalle. Die Kinder wollen ihn nun zur anderen Seite des Schulhofs hinter den Busch steuern.



Welche Folge von Kommandos können die Kinder rufen, um Lucky hinter den Busch zu steuern?

A) ☐ *Vor! Rechts! Vor! Links! Vor! Rechts! Vor! Links! Vor!*

B) ☐ *Rechts! Vor! Links! Vor! Links! Vor!*

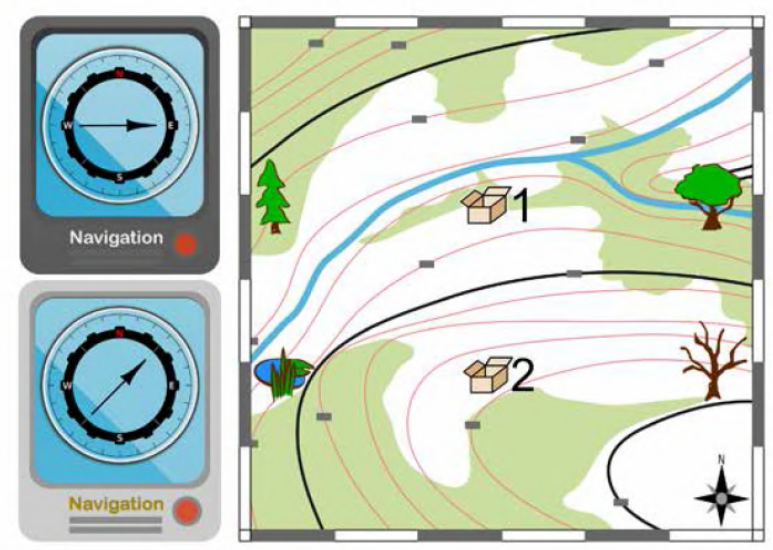
C) ☐ *Rechts! Vor! Links! Vor! Rechts! Vor! Rechts! Vor!*

D) ☐ *Vor! Rechts! Vor! Links! Vor! Links! Vor! Links! Vor!*

2. Geocaching

Lumi und Katja sind auf der Suche nach zwei Kisten, die für sie versteckt wurden. Dabei benutzen sie zwei Navigationsgeräte. Ein Gerät zeigt die Richtung zu Kiste 1, das andere die Richtung zu Kiste 2. Leider weißt du nicht, welches Gerät gerade zu welcher Kiste zeigt.

Im Bild siehst du links, welche Richtungen die beiden Geräte gerade zeigen.
Auf der Landkarte rechts siehst du sechs Orte.
Dazu gehören die Stellen, wo die Kisten versteckt sind.



An welchem Ort sind Lumi und Katja gerade?

A) ☐ 	B) ☐ 	C) ☐ 	D) ☐ 

3. Roboterkäfer

Ein Roboterkäfer kann sich auf diesem Spielbrett wie folgt bewegen:

Der Roboterkäfer startet auf einem beliebigen Feld der Spalten A bis D.

Steht der Roboterkäfer auf einem Feld, zählt er, wie viele Pfeile sich in diesem Feld befinden.

Dann bewegt er sich genauso viele Felder in Pfeilrichtung geradeaus und bleibt stehen.

Steht er zum Beispiel auf B4, dann bewegt er sich drei Felder nach oben und steht danach auf B1.

Der Roboterkäfer macht solange weiter, bis er entweder aus dem Spielbrett gelaufen ist oder in einem Feld der Spalte E steht.

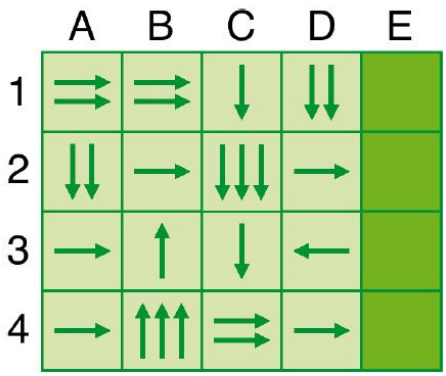
Von welchen Feldern der Spalte A aus kann der Käfer starten, um in einem Feld der Spalte E stehen zu bleiben?

A) ☐ **A1 und A2**

B) ☐ **A2, A3 und A4**

C) ☐ **A2 und A4**

D) ☐ **A1 und A4**



Aufgabe 1, 2 und 3 auf diesem Übungsblatt basieren auf Aufgaben aus dem Informatik-Schülerwettbewerb "Informatik-Biber" der Initiative "Bundesweit Informatik fördern", https://bwinf.de/biber/downloads/
Wir empfehlen die auf dieser Webseite zusammengestellten Materialien als Knobelaufgaben für eine vertiefende Beschäftigung mit Problemstellungen der Informatik.

Lösung zu Übungsblatt 9

1. Roboter spielen

Antwort A

2. Geocaching

Antwort C

3. Roboterkäfer

Antwort C

Zu Hause ausprobieren:

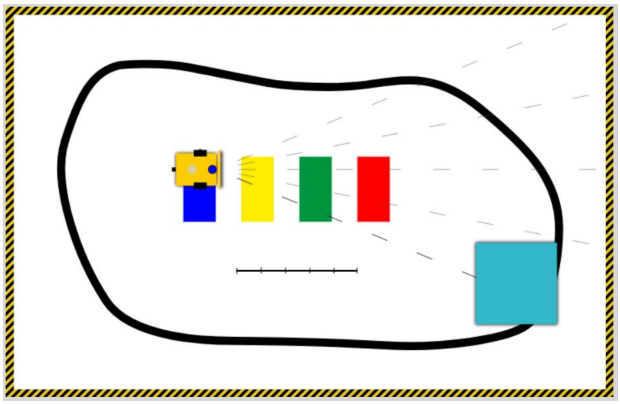
Schau dir nochmal das **Roberta-Projekt mit dem Tanzen** an! Konntest du Roberta schon zum Tanzen bringen? Probiere es nochmal aus!

Gehe an einem **Computer** ins **Internet** und schreibe folgende Adresse in die **URL-Zeile**:

https://lab.open-roberta.org/#

Dann drücke die **Enter-Taste**.

Jetzt kannst du das System **Open Roberta Sim** auswählen. Bestimmt erkennst du die Seite, die sich öffnet.

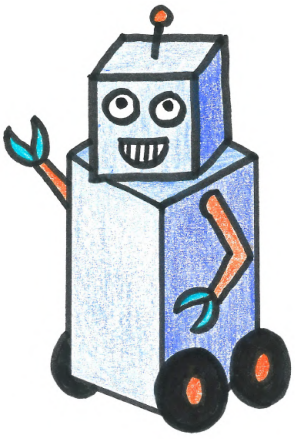


Schau nochmal auf die **Roberta-Einführung** und suche die richtige Umgebung aus.

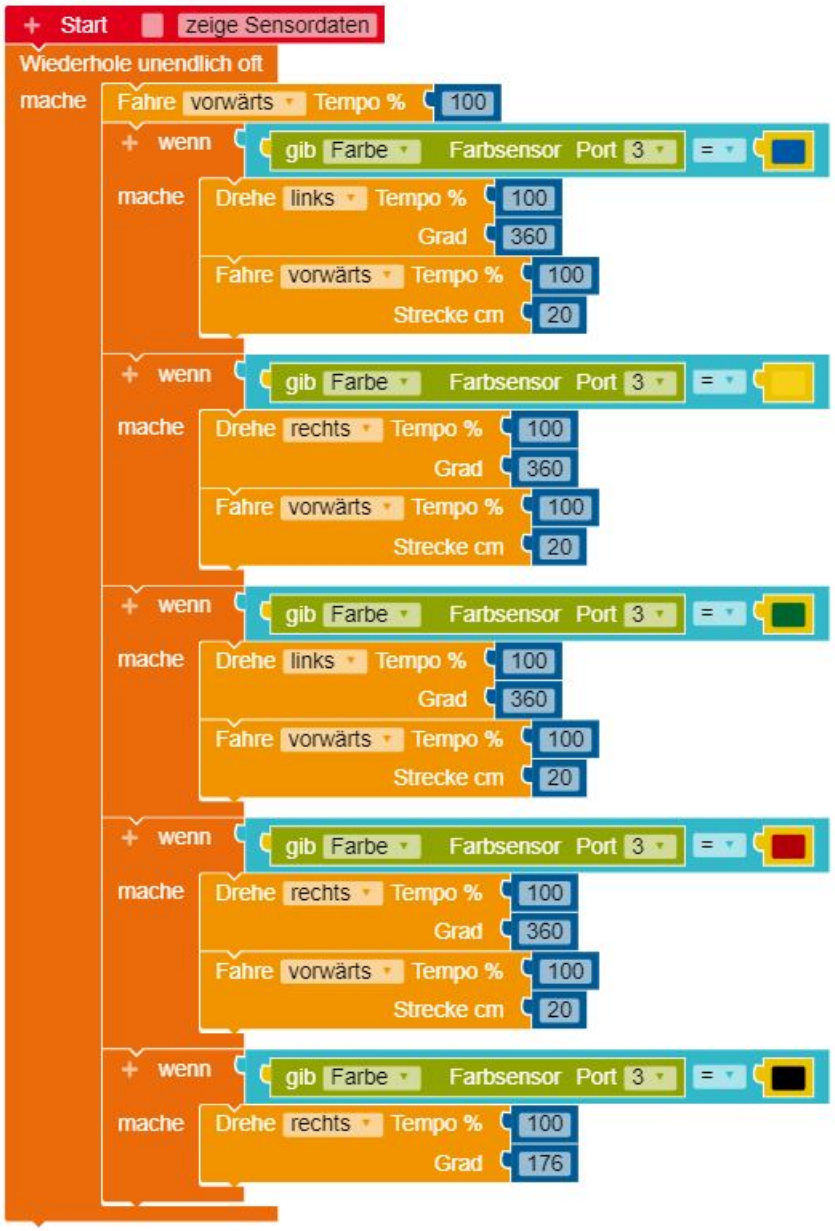
Bringe Roberta einen neuen Tanz bei!

Du kannst natürlich auch eigene Projekte ausprobieren!

Viel Spaß beim Programmieren!



Tipp: So kann man Roberta zum Beispiel tanzen lassen



Fallen dir noch mehr Ideen ein, wie Roberta tanzen könnte?

Die auf diesem Arbeitsblatt verwendeten Bilder (mit Ausnahme des blauen Roboters) sind Screenshots der Webseite:
http://lab.open-roberta.org
Wir empfehlen die auf dieser Webseite zusammengestellten Materialien für eine vertiefende Beschäftigung mit dem Programmieren in Open Roberta.

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Kurssitzung 10

- Roberta-Projekt
- Übungsblatt 10
- Zu Hause ausprobieren 10

Lasst uns anfangen!

Probiert die folgenden Schritte aus. Vergesst nicht, als Team zu arbeiten!

- Tippt folgende Adresse in die URL-Zeile ein:

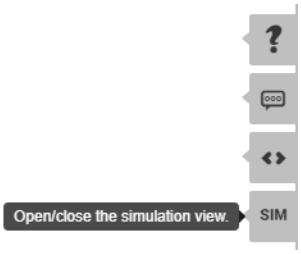
  https://lab.open-roberta.org/

- Drückt auf Enter!
- Wählt **Open Roberta Sim** aus.
- Klickt auf der rechten Seite des Bildschirms auf **SIM**, um das Simulations-fenster zu öffnen, in dem **Roberta** und ihre **Umgebung** angezeigt werden.

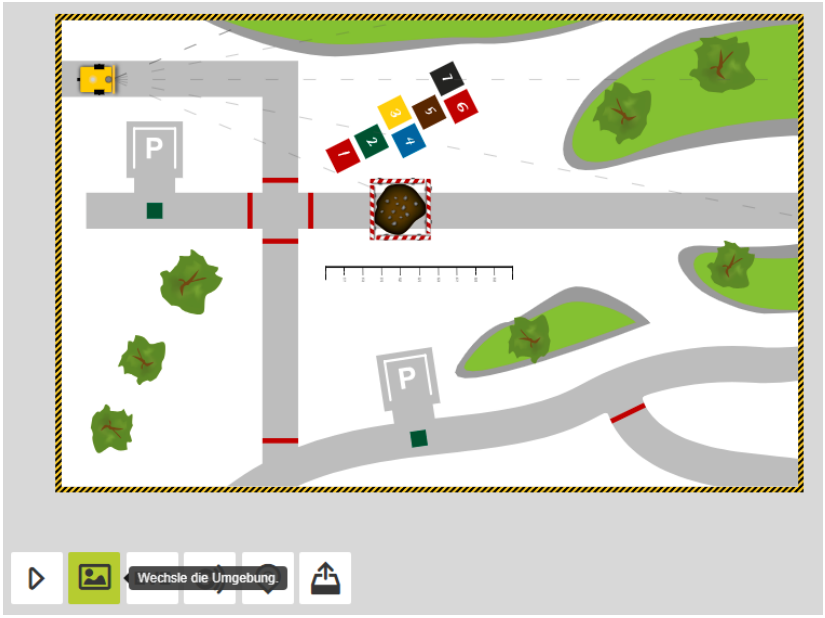
Einparken

Mit Hilfe der Motoren und Sensoren den **Roboter in eine Parklücke einparken** lassen. Als erstes lasst ihr ein beliebiges Programm starten, um den Bildschirm mit dem Roboter zu sehen.

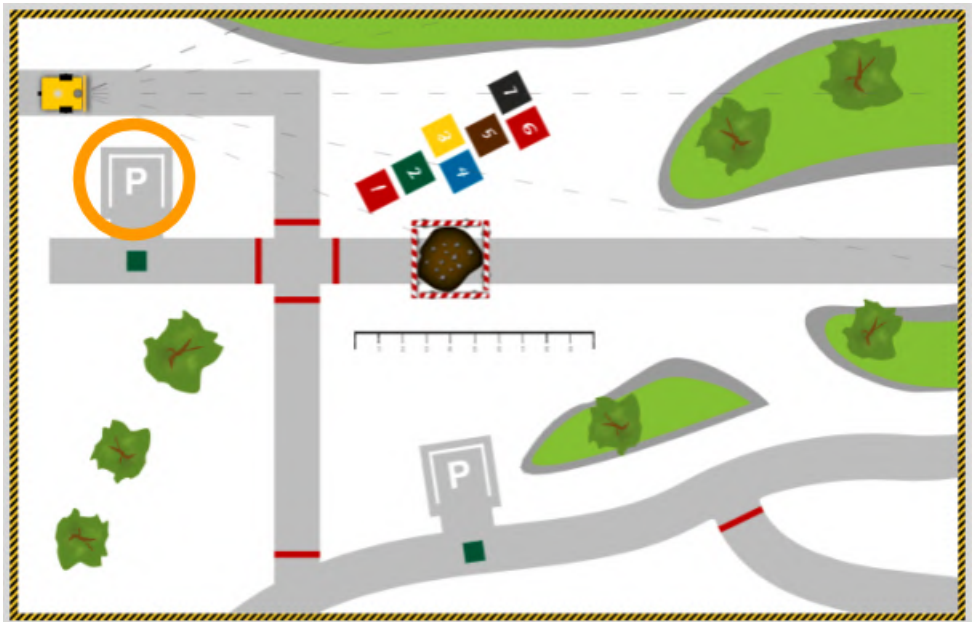
Öffnet danach die Simulationsübersicht (**SIM**).



Klickt dann auf das Symbol mit dem **Bilderrahmen**, bis ihr den **Hintergrund mit der Straße** gefunden habt.



Eure Aufgabe ist, den Roboter so zu programmieren, dass er der Straße folgt, an der Kreuzung rechts abbiegt und anschließend rückwärts auf den Parkplatz fährt.



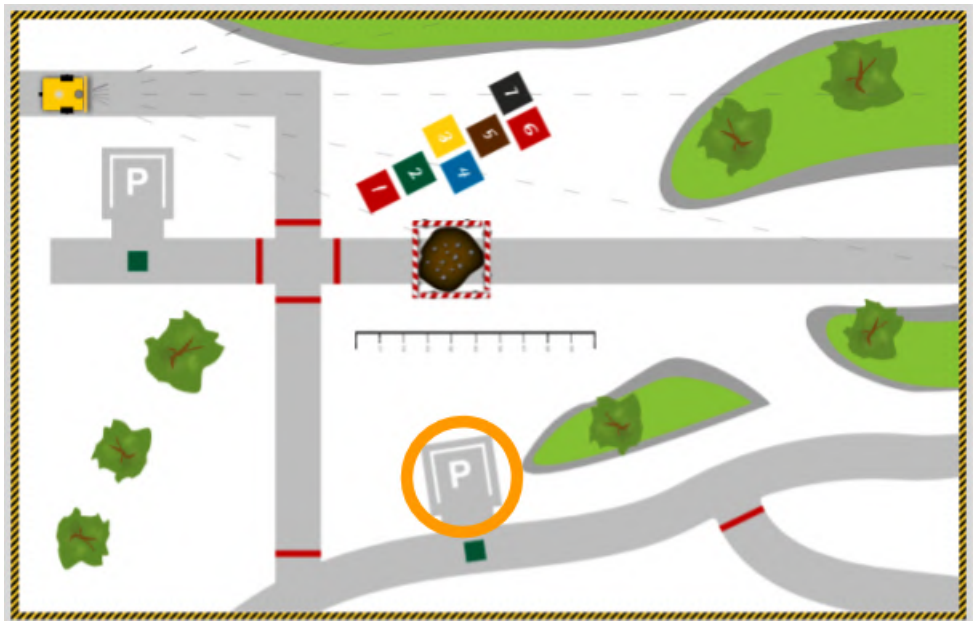
Sammelt Ideen, welcher Sensor euch dabei helfen kann, und probiert es aus!

Unten bekommt ihr einige Tipps, die ihr nutzen könnt, wenn ihr das möchtet.

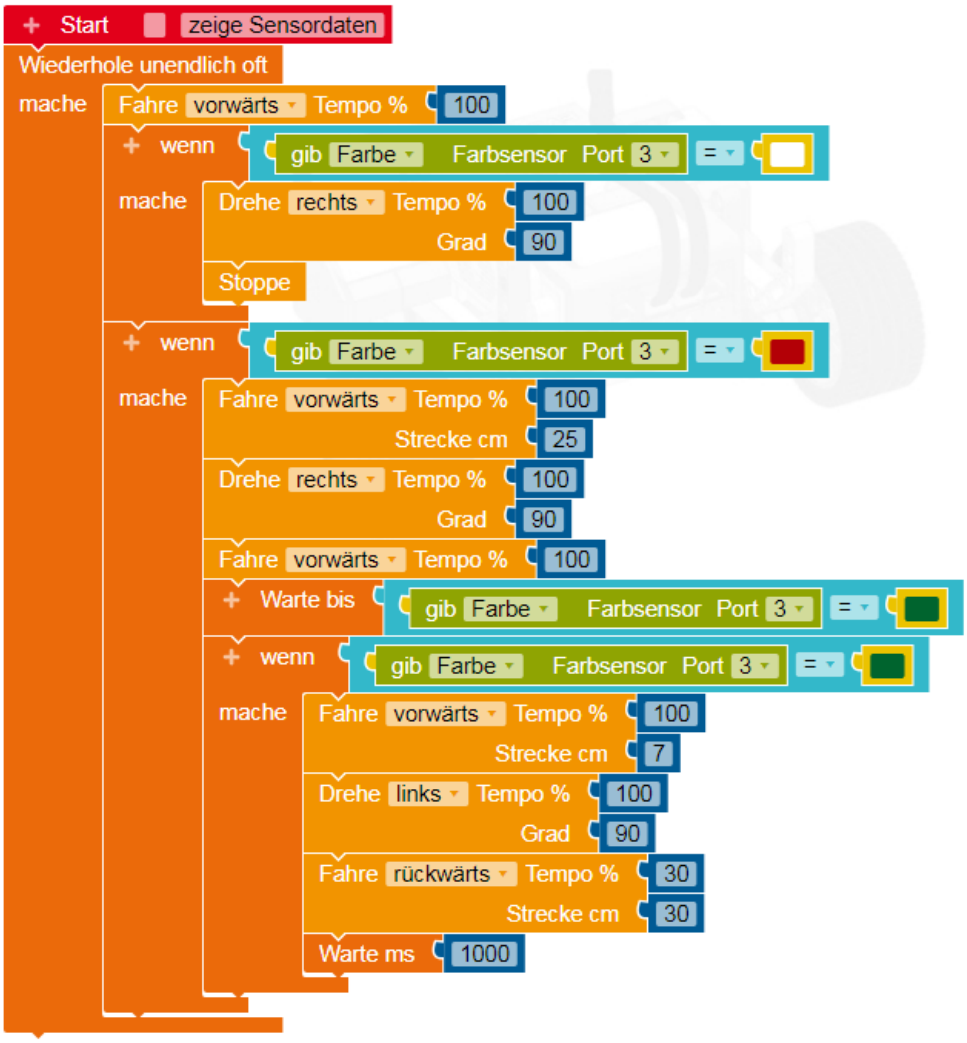
- **Tipp 1:** Der Farbensensor kann auch die Farben weiß und grau erkennen!
- **Tipp 2:** Wenn euer Roboter auf dieser Straße abbiegen möchte, muss er sich um 90 Grad drehen.
- **Tipp 3:** Nachdem euer Roboter die rote Linie an der Kreuzung gefunden hat, muss er noch einige Zentimeter geradeaus fahren, bevor er sich nach rechts dreht.
- **Tipp 4:** Auch auf dem grünen Quadrat muss der Roboter noch ein paar Zentimeter geradeaus fahren, bevor er sich dreht.

Einparken

Jetzt versucht mithilfe der Motoren und Sensoren euren **Roboter in die zweite Parklücke einparken** zu lassen.



Tipp: So kann man Aufgabe 1 lösen



Kannst du auch Aufgabe 2 lösen?

Dieses Arbeitsblatt basiert auf: "Roberta EV3 Simulator. Station 1: Einparken", Tobias Rütten, Team InfoSphere-Schülerlabor Informatik der RWTH Aachen,http://schuelerlabor.informatik.rwth-aachen.de/module/open-roberta
Die auf diesem Arbeitsblatt verwendeten Bilder sind Screenshots der Webseite: http://lab.open-roberta.org
Wir empfehlen die auf diesen Webseiten zusammengestellten Materialien für eine vertiefende Beschäftigung mit dem Programmieren in Open Roberta.

Name: _____

Übungsblatt 10

1. Erreicht er sein Ziel?

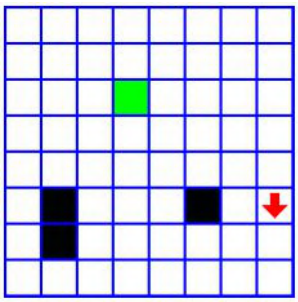
Unser Roboter soll auf einem Spielbrett sein Ziel erreichen: Das grüne Feld.

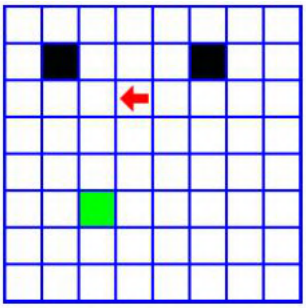
Das Feld mit dem Pfeil ist sein Startfeld. Die schwarzen Felder sind Hindernisse.

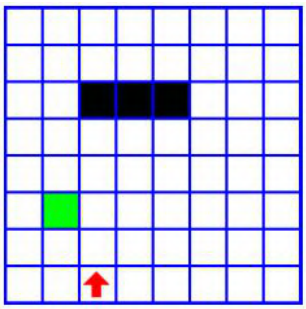
Der Roboter ist so programmiert: Er bewegt sich in Richtung des Pfeils geradeaus, bis er auf ein Hindernis oder den Spielbrettrand trifft. Dann dreht er sich um 90 Grad nach rechts und bewegt sich wieder geradeaus so weit es geht. Und so weiter.

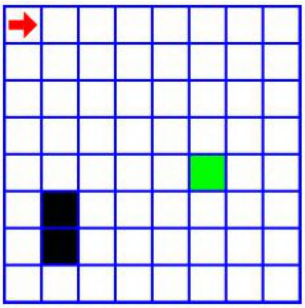
Jedes Feld, über das der Roboter kommt, wird sofort zu einem weiteren Hindernis, auch das Startfeld.

Auf welchem Spielbrett erreicht der Roboter NICHT sein Ziel?

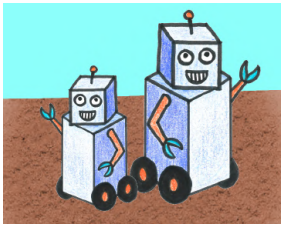
A) ☐ 

B) ☐ 

C) ☐ 

D) ☐ 

2. Die Gartenroboter



Du hast zwei Gartenroboter, die Blumen für dich Pflanzen. Der kleinere Roboter hat kürzere Arme und kleinere Räder als der große Roboter. Deshalb macht der kleinere Roboter auch kürzere Schritte und pflanzt die Blumen näher beieinander als der größere Roboter.

Am Anfang stehen die Roboter Rücken an Rücken im Beet und schauen in entgegengesetzte Richtungen. Dann bewegen sie sich beide nach diesen Anweisungen:

```
wiederhole 2 mal:
    pflanze eine Blume auf deiner rechten Seite
    bewege dich einen Schritt vorwärts
    pflanze eine Blume auf deiner linken Seite
    bewege dich einen Schritt vorwärts
```

Wie sieht das Blumenbeet anschließend aus?

A) ☐

B) ☐

C) ☐

D) ☐

3. Das Schubladenspiel

Für das Schubladenspiel brauchst du viele Schubladen mit unterschiedlichen Zeichen und ganz viele Kugeln, die alle gleich aussehen.

Das Spiel ist ganz einfach. Es funktioniert so:

[☆ := 3;] Ändere die Anzahl der Kugeln in der Schublade mit dem Zeichen ☆ in drei Kugeln!

[✨ := 🌙;] Ändere die Anzahl der Kugeln in der Schublade mit dem Zeichen ✨ in die Anzahl der Kugeln, die gerade in der Schublade mit dem Zeichen 🌙 liegen!

Und nach dem Spiel **[✨ := 1; 🌙 := 2; ☆ := 3; 🌙 := ☆;]** liegen in den benutzten Schubladen **⟨✨1, 🌙3, ☆3⟩** Kugeln.

Welches Schubladenspiel ändert die Kugeln in den Schubladen

von ⟨🌈7, ☁0, 🌊6⟩ **zu ⟨🌈6, ☁6, 🌊7⟩ ?**

A) ☐	**[🌈 := 🌊; ☁ := 🌊; 🌊 := 🌈;]**
B) ☐	**[🌈 := ☁; ☁ := 🌊; 🌊 := 7;]**
C) ☐	**[☁ := 🌈; 🌈 := 🌊; 🌊 := ☁; ☁ := 🌈;]**
D) ☐	**[🌈 := 6; 🌊 := 7; ☁ := 🌈; 🌈 := 🌊;]**

Aufgabe 1, 2 und 3 auf diesem Übungsblatt basieren auf Aufgaben aus dem Informatik-Schülerwettbewerb "Informatik-Biber" der Initiative "Bundesweit Informatik fördern", https://bwinf.de/biber/downloads/
Wir empfehlen die auf dieser Webseite zusammengestellten Materialien als Knobelaufgaben für eine vertiefende Beschäftigung mit Problemstellungen der Informatik.

Lösung zu Übungsblatt 10

1. Erreicht er sein Ziel?

Antwort C

2. Die Gartenroboter

Antwort A

3. Das Schubladenspiel

Antwort C

Zu Hause ausprobieren:

Verstehen wie Computer Denken

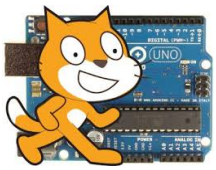
Hector Core Course

Verwendete Software

Du hast viel im Kurs gelernt! Jetzt kannst du schon mit Scratch, S4A und Open Roberta programmieren. Hier findest du die drei Programmier-umgebungen wieder:

Scratch
Info: **https://scratch.mit.edu/**
Programmierumgebung:
https://scratch.mit.edu/projects/editor/

Scratch for Arduino - S4A
Info: **http://s4a.cat/**
(Die Programmierumgebung ist nur im Download verfügbar)



Roberta
Info: **https://www.roberta-home.de/**
Programmierumgebung:
https://lab.open-roberta.org/

Weiterführende Lernmaterialien

Wenn du noch **mehr programmieren und dabei Neues lernen** möchtest, kannst du diese Links ausprobieren. Die Materialien laden zum spielerischen Umgang mit Programmierung und Informatik ein und sind **im Internet frei verfügbar**:

- **CODE.org**
  https://studio.code.org/courses (DE)
- **Blockly-Spiele**
  https://blockly-games.appspot.com/?lang=de (DE)
- **Lightbot**
  http://lightbot.com/flash.html (DE)
- **Internet-ABC**
  https://www.internet-abc.de (DE)
- **Kleine Forscher**
  http://www.meine-forscherwelt.de/spielen-und-wissen/computer (DE)
- **Scratch für Arduino (Simulation)**
  http://scratchx.org/?url=http://khanning.github.io/scratch-arduino-extension/arduino_extension.js&lang=de#scratch (DE)
- **Thinking myself**
  http://games.thinkingmyself.com (EN)
- **Turtle Stitch**
  http://turtlestitch.org/run (EN)
- **Pencil Code**
  http://pencilcode.net (EN)
- **Tynker**
  https://www.tynker.com/dashboard/student/#/dashboard/missions (EN)
- **Computer Science Unplugged**
  http://csunplugged.org/activities (EN)

DE heißt, dass die Materialien auf **Deutsch** sind.
EN heißt, dass die Materialien auf **Englisch** sind.

Viel Spaß beim Programmieren!