



UNIVERSITY OF TM
KWAZULU-NATAL

INYUVESI
YAKWAZULU-NATALI

Exploring the use of Robotics in the learning of programming

by

**Reginald Gerald Govender
(207501841)**

A thesis submitted in fulfilment of the academic requirements for the
Degree of Doctor of Philosophy in Computer Science Education,
School of Education, College of Humanities
University of KwaZulu-Natal, Edgewood campus
Durban, South Africa

Supervisor: Prof. Desmond W. Govender

November 2021

Abstract

Computer Programming is seen as a valuable skill in the digital era that we presently live in. However, for the novice programmer, it is often accompanied with difficulties resulting in negative reactions. The dawning of the Fourth Industrial Revolution has catapulted many initiatives local and global to promote Computer Programming and Robotics. A major initiative by the South African government is the planning and implementing of a new subject in school to raise the awareness of coding at an early age. The lack of coding exposure and awareness leads to little or no interest in Computer Programming related courses after schooling years. This study focuses on exploring the learning of coding through the use of Robotics among computer registered students with no prior coding knowledge at a University in South Africa.

Unlike the traditional use of block-based programming to introduce Computer Programming, which is limited to screen output, the study opted to use a physical manipulative by using a robotic element through prototype building using text-based programming, resulting in live autonomous output of code. The Arduino kit was used as the robot element to acquire knowledge development to the fundamental concepts of Computer Programming using the Python programming language. Participants' coding knowledge was assessed through a series of hands-on online activities.

Design Based Research was adopted with the integration of Kolb's Experiential Learning Cycle, framed within the second-generation Activity Theory. Mix methods were supported as it is in accordance with the pragmatic paradigm favoured by Design Based Research. All data collection took place online through workshops, surveys, questionnaires and a focus group interview. The sample size was 75 achieving a significant partial least squares structural equation model as a minimum of 50 participants was needed based on the ten times rule.

The results show that students acquiring a direct learning experience with text-based code with the aid of the robotic element proved to be successful. The robot coding simplified the assimilation of text-based coding as participants could see the execution of their code on the prototype in reality. The eradication of the abstract nature of Computer Programming through Robotics as a physical manipulative solidified the understanding of coding structures.

Furthermore, students' belief, interest, motivation, confidence, and Mathematics skill set were found to contribute success in Computer Programming. It was revealed that learning to code in a text-based environment can be made fun. In addition, learning programming with the use of the robot is effective for first time learning of text-based code. The researcher proposes that the introduction of learning programming integrated through the building of prototypes and coding resulting in autonomous robots enhances the learning experience of text-based code.

Table of contents

Abstract	ii
Table of contents	iv
Dedication	ix
Acknowledgments	x
Preface	xi
Declaration	xii
List of Abbreviations	xiii
List of Figures	xvi
List of Tables	xviii
PRELIMINARY RESEARCH	1
Chapter one: Introduction	1
1.1 Background.....	1
1.2 Purpose of the study	1
1.3 Significance of the study	2
1.4 Research objectives	3
1.5 DBR overview and structure of the dissertation.....	5
1.5.1 Preliminary research	5
1.5.2 Prototyping.....	6
1.5.3 Assessment.....	6
Chapter two: Literature review	7
2.1 Introduction	7
2.2 Everyone’s coding	8
2.3 Teaching and learning of code.....	12
2.4 Styles of programming	12
2.5 Proficiency in coding.....	15
2.5.1 Skill expectancy	15
2.5.2 Lack of early exposure.....	17

2.5.3 Computational Thinking	18
2.5.4 The influence of Mathematics	21
2.6 Robotics	23
2.6.1 Why Robotics?.....	23
2.6.2 Robotics as a learning aid	24
2.6.3 Lego Mindstorm series	26
2.6.4 Arduino	29
2.7 Programming languages	30
2.7.1 Hello world	30
2.7.2 Python Language	36
2.7.3 Integrated Developmental Environment (IDE).....	39
2.8 Setup	40
2.9 Workplace 2030.....	41
2.10 Conclusion.....	41
Chapter three: Theoretical framework.....	43
3.1 Introduction	43
3.2 Approach to the study.....	43
3.3 Activity theory.....	47
3.4 Conclusion.....	51
Chapter four: Research design and methodology	53
4.1 Introduction	53
4.2 Design Based Research – DBR	53
4.3 Integrating Kolb’s Experiential Learning Cycle	55
4.4 Research questions	56
4.5 Qualitative and quantitative data	57
4.6 Data collection techniques.....	58
4.6.1 Pre-workshop session.....	58
4.6.1.1 Pre-survey.....	58
4.6.1.2 Problem solving and logic.....	58
Part A: Pre-test based on Computational Thinking.....	58
Part B: Abstract Reasoning Test	59
4.6.2 Workshop sessions	59
4.6.3 Post-workshop session:.....	60

4.6.3.1 Post-survey	60
Selection of constructs/latent variables	60
4.6.3.2 Post-test based on programming	61
4.6.3.3 Interview.....	61
4.7 Fitting it all together	62
4.8 Study setting	64
4.8.1 Location and population	64
4.8.2 The pilot study	65
4.9 Validity, reliability and rigour	66
4.10 Permission and ethical considerations.....	66
4.11 Conclusion.....	67
PROTOTYPING.....	69
Chapter five: Iteration.....	69
5.1 Introduction	69
5.2 Pre-workshop session (excluded from the six micro cycles)	69
5.3 Micro cycle: Workshop session one.....	70
5.4 Micro cycle: Workshop session two.....	71
5.5 Micro cycle: Workshop session three.....	72
5.6 Micro cycle: Workshop session four	74
5.7 Micro cycle: Workshop session five	75
5.8 Micro cycle: Workshop session six.....	76
5.9 Composite accomplishment ratings.....	77
5.10 Conclusion.....	78
ASSESSMENT.....	79
Chapter six: Analysis and findings.....	79
6.1 Introduction	79
6.2 Data presentation and analysis	79
6.3 Pre-workshop session	79
6.3.1 Pre-survey	79
6.3.2 Questionnaire test one~ Problem solving and logic.....	88
6.3.2.1 Part A: Pre-test based on Computational Thinking.....	88
6.3.2.2 Part B: Abstract Reasoning Test	102

6.4 Post-workshop session.....	109
6.4.1 Post-survey.....	109
6.4.1.1 Measurement model.....	112
Indicator reliability.....	112
Convergent reliability.....	114
Discriminant validity.....	115
Heterotrait-Monotrait ratio of correlations, bootstrapping and normality.....	117
6.4.1.2 Structural model.....	119
Collinearity issues.....	120
Path coefficients.....	120
Significance of the relationships- t values and p values.....	122
Level of R ²	123
Effect size (f ²).....	125
Predictive relevance (Q ²).....	125
6.4.2 Questionnaire test two~ Post-test based on programming.....	127
6.6 Conclusion.....	132
Chapter seven: Discussion and conclusion.....	133
7.1 Introduction.....	133
7.2 Discussion.....	134
7.3 Closing remarks.....	139
7.4 Limitation and further research.....	141
7.5 Conclusion.....	142
References.....	143
Appendices.....	176
Appendix A - Permission from Registrar.....	177
Appendix B - Informed consent letter.....	178
Appendix C - Ethical clearance.....	179
Appendix D - Turn- it- in Report.....	180
Appendix E - Editor’s Report.....	181
Appendix G - Questionnaire one Part A: Pre-test based on computational thinking.....	183
Appendix H - Questionnaire one Part B: Abstract reasoning test.....	186
Appendix I – Pre workshop session.....	194

Appendix J - Workshop session one	207
Appendix K – Self-evaluation workshop session one.....	213
Appendix L - Workshop session two	214
Appendix M – Self-evaluation workshop session two.....	220
Appendix N - Workshop session three.....	221
Appendix O – Self-evaluation workshop session three	226
Appendix P - Workshop session four.....	227
Appendix Q – Self-evaluation workshop session four.....	232
Appendix R - Workshop session five.....	233
Appendix S – Self-evaluation workshop session five.....	238
Appendix T - Workshop session six	239
Appendix U – Self-evaluation workshop session six.....	245
Appendix V – Post- survey (survey two).....	246
Appendix W - Post survey (survey two) categorised under constructs.....	248
Appendix X – Post-test (questionnaire on programming language).....	250
Appendix Y - Interview.....	253
Appendix Z - Interview transcript.....	254

Dedication

This thesis is dedicated to my:

To my parents for the sacrifices made in difficult times and instilling in me the value of education. Your love has moulded and made me into the person that I am today.

Dear Mum, as I would not have made it thus far without her selfless sacrifices, support and inspiration

&

my Dad who passed away before seeing me achieve everything he knew I was capable of.

Acknowledgments

I would like to thank the unseen but all-knowing God for guiding me through this incredible journey- *Veni Vidi Vici!*

I am grateful to my supervisor Prof D.W. Govender for his support and aspiration for me to succeed in my studies, which has resulted in the completion of this study.

I like to thank my international mentor Prof F. Mensah from Columbia University and my local mentor Prof A. Philips from University of KwaZulu-Natal, for their advice and thought engaging discussions during my research.

To my colleagues in the Mathematics and Computer Science cluster and especially in the discipline of Computer Science Education for your unwavering support.

I am thankful for my Dean Prof T. Mbisi who prompted me in joining the AALDP programme and my fellow PhD cohort for their constructive feedback and advice.

I wish to acknowledge and thank the National Research Foundation for funding my study and the UCDF for their support.

Most importantly a huge thank you to all the participants who were part of the study and hope that you will continue to explore the world of coding and Robotics.

Anyone I have not mentioned please accept my apologies and my gratefulness for your support.

There is nothing more divine than education. It is only through education that one truly becomes man. ~Plato

Preface

The research undertaken was supported wholly by the National Research Foundation (grant number 122017).

The study was supported and guided by Columbia University, TC, New York City during the researcher's stint at the university.

The following academic articles/activities emanate from this study:

Govender, R. G. & Govender, D. W., (2020), ROBOPROG: Learning of Flowcharts through a Gamified Experience, *International Journal of Business and Management Studies*, 12 (2).https://www.sobiad.org/eJOURNALS/journal_IJBM/archives/IJBM_2020-2ek/r-govender.pdf

Govender, R. G., & Govender, D. W. (2021). A Physical Computing Approach to the Introduction of Computer Programming among a Group of Pre-service Teachers. *African Journal of Research in Mathematics, Science and Technology Education*, 25(1), 54-65. <https://doi.org/10.1080/18117295.2021.1924440>

Declaration

I Reginald Gerald Govender declare that

- i. The research reported in this dissertation, except where otherwise indicated, is my original work.
- ii. This dissertation has not been submitted for any degree or examination at any other university.
- iii. This dissertation does not contain other persons' data, pictures, graphs or other information, unless specifically acknowledged as being sourced from other persons.
- iv. This dissertation does not contain other persons' writing, unless specifically acknowledged as being sourced from other researchers. Where other written sources have been quoted, then:
 - v. their words have been re-written but the general information attributed to them has been referenced;
 - vi. where their exact words have been used, their writing has been placed inside quotation marks, and referenced.
- vii. Where I have reproduced a publication of which I am an author, co-author or editor, I have indicated in detail which part of the publication was actually written by myself alone and have fully referenced such publications.
- viii. This dissertation does not contain text, graphics or tables copied and pasted from the Internet, unless specifically acknowledged, and the source being detailed in the dissertation and in the References sections.

Date: Nov 2021...

Reginald Gerald Govender (207501841)

List of Abbreviations

2D: Two-dimensional

3D: Three-dimensional

4IR: Fourth Industrial Revolution

AI: Artificial Intelligence

AALDP: Accelerated Academic Leadership Development Programme

APS: Admission Points Score

ART: Abstract Reasoning Test

AVE: Average Variance Extracted

CA: Cronbach's Alpha

CB-SEM: Covariance Based-Structural Equation Modelling

CHAT: Cultural Historical Activity Theory

CI-LL: Confidence Interval-Lower Limit

CI-UL: Confidence Interval-Upper Limit

COVID-19: Coronavirus 2019

CPU: Central Processing Unit

CR: Composite Reliability

CRC: Class Responsibility Collaboration

CS: Computer Science

CT: Computational Thinking

DBE: Department of Basic Education

DBR: Design Based Research

EV3: Evolution 3

FET: Further Education Training

GET: General Education and Training

GUI: Graphical User Interface

HTMT: Heterotrait-Monotrait

ICT: Information and Communications Technology

IDE: Integrated Development Environments

IDLE: Integrated Development and Learning Environment

IEEE: Institute of Electrical and Electronics Engineers

IoT: Internet of Things

IPO: Input Processing Output

IT: Information Technology

KELC: Kolb's Experiential Learning Cycle

LDR: Light Dependent Resistor

LED: Light Emitting Diode

LMS: Learning Management System

LV: Latent variable

MCQ: Multiple-choice questions

MIT: Massachusetts Institute of Technology

NXT: Next generation

OLS: Ordinary least squares

OOP: Object-Oriented Programming

PLS-SEM: Partial Least Squares-Structural Equation Modelling

POP: Procedural-Oriented Programming

PRIMM: Predict-Run-Investigate-Modify-Make

Q-Q: Quantile-Quantile

RCX: Robotics Invention System

SD: Standard deviation

SE: Standard error

SGAT: Second-generation Activity Theory

SONA: State of the Nation Address

STEM: Science, Technology, Engineering and Mathematics

TIMSS: Trends in International Mathematics and Science Study

TOE: Task Object Event

UCDP: University Capacity Development Programme

USB: Universal Serial Bus

VIF: Variance Inflated Factor

ZPD: Zone of Proximal Development

List of Figures

Figure 1: <i>Structure and implementation of study according to DBR</i>	5
Figure 2 <i>An illustration of Alice</i>	9
Figure 3 <i>An illustration of Scratch</i>	9
Figure 4 <i>An illustration of Jeliot3</i>	10
Figure 5 <i>An illustration of UUhistle</i>	11
Figure 6 <i>Cognitive levels between digital literacy, computer literary and computational thinking and skills</i>	20
Figure 7 <i>Microsoft Windows Logo output of a square</i>	25
Figure 8 <i>Code in Logo commander window</i>	25
Figure 9 <i>Turtle device</i>	26
Figure 10 <i>First generation RCX</i>	27
Figure 11 <i>Second generation NXT</i>	28
Figure 12 <i>Third generation EV3</i>	28
Figure 13 <i>EV3 program that detects a black line</i>	29
Figure 14 <i>Arduino UNO R3 pin diagram</i>	30
Figure 15 <i>Hello world code in Java</i>	31
Figure 16 <i>Hello world code in C</i>	31
Figure 17 <i>Hello world code in Delphi</i>	32
Figure 18 <i>Hello world code in C++</i>	32
Figure 19 <i>Hello world code in Python</i>	32
Figure 20 <i>Hello world code in C#</i>	32
Figure 21 <i>Fibonacci code in C</i>	33
Figure 22 <i>Fibonacci code in Java</i>	34
Figure 23 <i>Fibonacci code in C++</i>	34
Figure 24 <i>Fibonacci code in C#</i>	34
Figure 25 <i>Fibonacci code in Delphi</i>	35
Figure 26 <i>Fibonacci code in Python</i>	35
Figure 27 <i>Relationship between low-level and high-level language</i>	36
Figure 28 <i>Ranking of programming languages</i>	37
Figure 29 <i>Schematic of the PC connection with Arduino</i>	40
Figure 30 <i>An illustration of first-generation Activity Theory</i>	47

Figure 31 <i>An illustration of second-generation Activity Theory Theory</i>	49
Figure 32 <i>An illustration of third-generation Activity Theory</i>	50
Figure 33 <i>Second-generation Activity Theory applied in study</i>	51
Figure 34 <i>An illustration of the flow of DBR based on three phases</i>	54
Figure 35 <i>Linkng the philosophical principles underpinning this study</i>	63
Figure 36 <i>Data collection process</i>	67
Figure 37 <i>Visual overview of micro cycle one feedback</i>	71
Figure 38 <i>Visual overview of micro cycle two feedback</i>	72
Figure 39 <i>Visual overview of micro cycle three feedback</i>	73
Figure 40 <i>Visual overview of micro cycle four feedback</i>	74
Figure 41 <i>Visual overview of micro cycle five feedback</i>	75
Figure 42 <i>Visual overview of micro cycle six feedback</i>	76
Figure 43 <i>Distribution of responses from pre-survey</i>	80
Figure 44 <i>Right vs Wrong responses for Part A</i>	91
Figure 45 <i>Summary of marks for Part A</i>	94
Figure 46 <i>Distribution of marks for Part A</i>	94
Figure 47 <i>Visualisation depicting normal distribution; based on data from Table 31</i>	97
Figure 48 <i>Q-Q Plot Mathematical Literacy</i>	101
Figure 49 <i>Q-Q Plot Pure Mathematics</i>	102
Figure 50 <i>Summary of scores for Part B: ART</i>	104
Figure 51 <i>Q-Q Plot Part B: ART</i>	105
Figure 52 <i>Spread of Part B: ART scores</i>	108
Figure 53 <i>The initial form of the model</i>	111
Figure 54 <i>Model showing valid path loadings</i>	113
Figure 55 <i>Visual representation of path coefficients</i>	121
Figure 56 <i>R Square values (R^2) and inner model depicting t values.</i>	124
Figure 57 <i>Right vs Wrong responses for programming test</i>	128
Figure 58 <i>Summary of marks for Programming test</i>	129

List of Tables

Table 1	<i>National count of Grade 12 learners enrolled in IT at schools</i>	18
Table 2	<i>Use of Python code in the private sector</i>	38
Table 3	<i>Kolb’s Experiential Learning Cycle vs the DBR prototyping phase</i>	56
Table 4	<i>Data instruments versus research questions</i>	62
Table 5	<i>Overview of micro cycle one feedback</i>	70
Table 6	<i>Overview of micro cycle two feedback</i>	72
Table 7	<i>Overview of micro cycle three feedback</i>	73
Table 8	<i>Overview of micro cycle four feedback</i>	74
Table 9	<i>Overview of micro cycle five feedback</i>	75
Table 10	<i>Overview of micro cycle six feedback</i>	76
Table 11	<i>Composite accomplishment ratings based on the majority responses</i>	77
Table 12	<i>Reliability statistics of non-reverse coded items</i>	81
Table 13	<i>Reliability statistics of reverse coded items</i>	81
Table 14	<i>Response to item 1. I have an interest in programming</i>	82
Table 15	<i>Response to item 2. I lack a basic mathematical background.</i>	83
Table 16	<i>Response to item 3. I think programming is too technical.</i>	83
Table 17	<i>Response to item 4. I can succeed in learning Computer Programming.</i>	84
Table 18	<i>Response to item 5. I am good at problem solving.</i>	85
Table 19	<i>Response to item 6. I think it would be interesting to use programming to solve problems.</i>	85
Table 20	<i>Response to item 7. From my own understanding of programming, it is boring.</i>	86
Table 21	<i>Response to item 8. My perception of programming is that it is difficult to learn.</i> ..	86
Table 22	<i>Response to item 9. I think programming is hard.</i>	87
Table 23	<i>Response to item 10. I have an interest in microcontrollers (robotic element).</i>	87
Table 24	<i>Bibliographic information</i>	88
Table 25	<i>Overall summary response from Part A (pre-test)</i>	89
Table 26	<i>Distribution of Part A (pre-test) marks obtained (out of 10)</i>	90
Table 27	<i>Summary of responses to question one</i>	92
Table 28	<i>Summary of responses to question nine</i>	92
Table 29	<i>Summary of responses to question ten</i>	92

Table 30 <i>Five-point number summary with mean and standard deviation (SD) for Part A (pre-test)</i>	93
Table 31 <i>Standardisation of values (raw marks out of 10)</i>	95
Table 32 <i>Differences depending on the type of Mathematics vs MCQ mark obtained using descriptive statistics</i>	98
Table 33 <i>Mark distribution based on percentage weighting per mark obtained</i>	98
Table 34 <i>Descriptive statistics based on the type of mathematics</i>	99
Table 35 <i>Calculation of the skewness z-value and kurtosis z-value</i>	100
Table 36 <i>Shapiro-Wilk test of normality</i>	101
Table 37 <i>Five-point number summary with mean and SD Part B</i>	103
Table 38 <i>Kolmogorov-Smirnov test of normality</i>	104
Table 39 <i>One-sample t-test</i>	107
Table 40 <i>Pearson correlation</i>	108
Table 41 <i>Set of hypotheses based on model</i>	112
Table 42 <i>Measurement model including LV (construct) reliability and validity</i>	115
Table 43 <i>Discriminant Validity- indicator item cross loading</i>	116
Table 44 <i>Discriminant Validity- Fornell and Larcker criterion</i>	117
Table 45 <i>Assessing Normality- Univariate and multivariate skewness and kurtosis</i>	118
Table 46 <i>Assessing Normality- Mardia's multivariate skewness and kurtosis</i>	118
Table 47 <i>Discriminant Validity- HTMT</i>	119
Table 48 <i>Structural model- Variance Inflated Factor</i>	120
Table 49 <i>Path coefficients</i>	121
Table 50 <i>Direct relationships for Hypothesis testing</i>	122
Table 51 <i>R Square (R^2) values</i>	123
Table 52 <i>Effect Size (f square)</i>	125
Table 53 <i>Construct Cross validated Redundancy</i>	126
Table 54 <i>Overall summary response from post- test</i>	127
Table 55 <i>Distribution of post-test marks obtained (out of 10)</i>	128
Table 56 <i>Five-point number summary with mean and SD for programming test (post-test)</i>	129
Table 57 <i>Paired Samples Statistics</i>	130
Table 58 <i>Paired Samples Test (pre-test versus post-test)</i>	130

PRELIMINARY RESEARCH

Chapter one: Introduction

“Learning to write programs stretches your mind, and helps you think better, creates a way of thinking about things that I think is helpful in all domains.” ~ Bill Gates

1.1 Background

South Africa continues to endure a critical shortage of digital skills, including computer programmers (Lategan, 2020; Maisiri & van Dyk, 2021; Malinga, 2021). Computer Programming¹ can be viewed as a series of instructions given to the computer to perform technical tasks. Further, as described by Akcay et al. (2018), programming code is a language for “describing computation and expressing a set of instructions on what tasks a computer needs to execute” (p. 1). It would seem that the greatest challenge faced by most students is the understanding of programming basics. This is especially true for novice programmers with no formal learning of programming; or those in their first year of study at tertiary level with no prior exposure at school level (Lo et al., 2015; Saeli et al., 2011). Furthermore, the teaching of programming is classified as difficult, since computer programs and algorithms are complex constructs that require abstract thinking. Consequently, these concepts and processes are often regarded as difficult to teach and learn (Bati et al., 2014; Mendes et al., 2012; Olsson et al., 2015). In recent years much attention has been given to Robotics and coding (SONA, 2019). It is possible that learning to code using robotics could be a strategy to alleviate the difficulties and challenges that learning programming presents.

1.2 Purpose of the study

Numerous strategies have been employed to introduce students to programming, to simplify the process and make understanding easy. Such strategies include visualisation tools and GUI environments to enhance understanding. However, such methods have not always proven effective (Kalelioglu & Gülbahar, 2014; Olsson et al., 2015; Tanrikulu & Schaefer, 2011; Techapalokul & Tilevich, 2017). There is limited research on best practices in learning to program and therefore the need for this study. The selected programming language used in this

¹ “Computer programming” and “coding” are used interchangeable in this study.

research is Python, as it is one of the most widely adopted general-purpose high-level programming languages (Summerfield, 2010).

As reported by Shein (2015), more people use Python as an “introductory programming language because it has a very large set of highly useful libraries that have been built over the years” (p. 19). This study used the Python language in a procedural paradigm to introduce students to the basic principles of programming together with the use of a robot element offering prototype building as an educational tool. Other paradigms, like object-oriented approaches, may not be ideal for introducing programming basics (Govender & Govender, 2016). It was hoped that the proposed strategy would allow the acquisition of knowledge about text-based programming seamless, achieving better results than with other past strategies. Factors such as motivation, interest and belief in the subject content are essential for student success, as these factors make the content interesting and appealing. Therefore, this study focuses on a strategy that enhances best practices for learning programming.

1.3 Significance of the study

With the dawn of the Fourth Industrial Revolution (4IR), the South African government has emphasised the skills needed for the future, including coding and Robotics, through its Vision 2030 (Ramaphosa, 2019). These skills prepare young people for future jobs and boost the national economy and demand for South Africa in the global market.

Coding is considered to be difficult to assimilate and requires many years to master (Robins et al., 2003). It would seem to necessitate a complex set of cognitive activities that demand the development of problem-solving and decision-making skills, as well as logical reasoning. Furthermore, the verbose programming language and Integrated Development Environments (IDE) do little to make it easier (Chen et al., 2017).

In South Africa, there is a lack of early exposure to the fundamentals of coding, let alone Robotics, resulting in no foundation that could lead further perusal or interest. This is in contrast to other specialisation subjects in school, such as, geography, physical science, biology, history, etc., where subject content in the prior phases lays a foundation for these subjects in the Further Education Training (FET) phase. Coding as a subject (Information Technology)

can only be selected in the later years of schooling, at the FET phase², making the Zone of Proximal Development (ZPD) difficult to scaffold.

In accordance with the constructivist approach, ZPD can be described as knowledge that a student can attain (by building on prior knowledge) with the assistance of facilitation to scaffold student development across the ‘gap’ between what is known and the construction of new knowledge (Vygotsky, 1978).

Students who lack prior exposure to programming cannot build new knowledge from any foundation, thereby adding to the abstract nature of the subject. As a result, many students may avoid fields at tertiary level that involve Computer Programming, due to their lack of prior exposure and, thus, a foundation in the subject. Computer Programming courses are sometimes perceived as uninteresting and demotivating, leading to high dropout rates (Lin & Kuo, 2010). In contrast, there have been innovative methods for learning programming over the years, including block-based coding (Erol & Kurt, 2017; Tanrikulu & Schaefer, 2011). Block-based coding is limited to 2D animation on the computer screen and restricts creativity due to its non-text-based setup. Hence, the competent computer programmer must, ultimately, be able to demonstrate proficiency in a text-based environment.

This study acts as a guide for consultation by the DBE when considering the development of new subjects that will introduce digital skills by incorporating Robotics and coding. Furthermore, the study seeks to promote and provide an innovative method to introduce Computer Programming through the use of Robotics. In doing so, the research offers a practical solution to developing scarce skills for the 4IR, and raising awareness among tertiary students.

In response to the above-mentioned key-significance areas of the study, the researcher’s intention was to explore the use of Robotics in the learning of basic coding using a text-based environment.

1.4 Research objectives

There is no fool-proof strategy for the introduction of programming to attain programming knowledge. Therefore, the following questions arise: What are the best practices for

² FET phase consists of Grades 10, 11 and 12 which are the final three school finishing years.

teaching/learning programming? How can the teaching of programming be promoted among students at tertiary level since programming is perceived as being difficult?

The proposed study sets out the following aims/objectives:

1. *An alternative educational tool*: To explore the use of Robotics to enhance the learning of Computer Programming.

The robot kit provides a hands-on approach to programming (i.e., physical computing) since the student designs, programs and executes the code in reality on the prototype rather than in a 2D representation on a screen.

2. *Problem solving and reasoning*: To explore external factor/s that contribute to the learning of Computer Programming.

It is well documented in literature that internal issues around learning and understanding programming exist, such as the programming paradigm preference, the programming language used and the programming style (Dmitrieva et al., 2019; Hendrix & Weeks, 2018; Hourani et al., 2019; Husain et al., 2016; Kurdi, 2013; Nasrawt & Lam, 2019; Samuel, 2017). However, there are contradictory accounts of external factors such as students' mathematical background, development of Computational Thinking, higher-order thinking skills, etc., that would deem a student proficient in Computer Programming (Bubica and Boljat, 2015; Elaine, 2013; Kurland, 1989; Pea & Rosen, 2018).

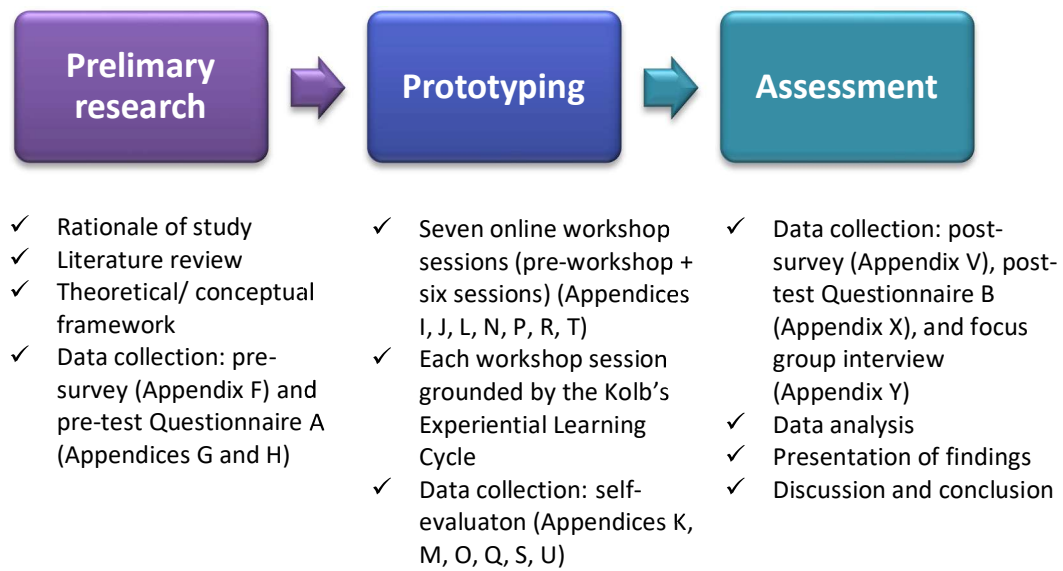
3. *Easy to understand*: To assess the effectiveness of Robotics in the understanding of Computer Programming.

This study used the programming of Robotics to introduce the basics of programming using a text-based environment rather than a block-based environment. The goal was that students would acquire a direct learning experience with text-based code in the hope of simplifying the learning process of programming.

1.5 DBR overview and structure of the dissertation

The three phases of Design Based Research (DBR) will be superimposed onto the structure of this dissertation as follow (Figure 1):

Figure 1: *Structure and implementation of study according to DBR*



Note. Overview of study according to Design Based Research

This study is structured into the following seven chapters under the three phases of DBR.

1.5.1 Preliminary research

Chapter one – Introduction introduces the study, discusses the background, scope and context. It presents the rationale and significance of the study by unpacking the purpose, focus and objectives. This chapter concludes with an overview of the study.

Chapter two – Literature review presents a literature review to gain a retrospective of the research by offering a corpus of relevant literature that guided and supported this study.

Chapter three – Theoretical framework presents a comprehensive write-up of the theoretical framework employed for this study, which is Activity Theory. A historical outline of the development of Activity Theory and its application to the research study is discussed. Pertinent literature that underpins the pragmatic paradigm that grounds the study will also be discussed in this chapter.

Chapter four – Research design and methodology outlines the research methodology used in this study, which is Design Based Research (DBR). It presents a systematic methodology and how the methods were applied in the research. In addition, the data collection instruments, location, ethical considerations and informed consent are discussed in chronological order. The chapter also outlines and justifies the research design and the trustworthiness or rigor of the study.

1.5.2 Prototyping

Chapter five – Iteration provides a discussion around each workshop session. For coherence purposes and in accordance with DBR, the analysis and discussion of each workshop, represented as cycles, appear in this chapter.

1.5.3 Assessment

Chapter six – Analysis and findings present and discuss the data analysis report and findings. The analysis includes data collected during the *preliminary* stage and the current *assessment* stage in accordance with DBR.

Chapter seven – Discussion and conclusion presents a holistic discussion highlighting the findings across all data collected, providing recommendations and suggestions for further research and a summative conclusion.

Chapter two: Literature review

“Talk is cheap. Show me the code.” ~ Linus Torvalds

2.1 Introduction

As South Africa embraces the Fourth Industrial Revolution (4IR), we need to prepare the future workforce with skills that are pertinent to the country's development. Among these skills, Computer Programming is essential in this Digital Age. According to the Royal Society Report (2012), Computer Science (CS) can be described as “the scientific discipline, encompassing principles such as algorithms, data structures, programming, systems architecture, design; and problem-solving” (p. 5). Bubnó et al. (2014) point out that Computer Programming is the core learning goal of any CS course. Similarly, Lo et al. (2015) state that “programming is a fundamental ability for Computer Science majors” (p. 225). Aspects of CS, particularly programming, can be found in some of the following courses offered at tertiary level like computer engineering, forensics, networking, cybersecurity, database administration, information security, technology, software engineering and web development. Saeli et al. (2011) point out that “programming is only one of the topics concerning the teaching of Informatics” (p. 74).

According to Margulieux et al. (2016), the problems with programming in education are twofold. Firstly, the techniques for teaching programming are relatively undeveloped as compared with other disciplines. The teaching of CS is based on industry practices; thus, most CS teaching methodologies are not informed by educational psychology (Knox et al., 1996). Secondly, the number of teachers qualified to teach programming is insufficient. Generally, the topics covered in a core programming syllabus, whether at school or tertiary level, focus on the student's development of algorithms (flowchart and pseudocode), Input Processing Output (IPO) charts, Task Object Event (TOE) charts, case diagrams, and debugging techniques like trace tables. This includes decomposing problems into strands, reasoning, developing systematic plans, and debugging steps repeatedly until a refined solution is reached. A programmer can choose different approaches to writing a program since there is no single approach to problem solving in this context. Programming demands complex cognitive skills involving a mix of procedural and conceptual understanding, out-of-the-box thinking, non-linear reasoning, and a no-one solution fits all stance (Lahtinen et al., 2005; Olsson et al., 2015).

In addition, the nature of CS incorporates ways of thinking such as Computational Thinking (CT), following rigorous processes and non-routine methods.

2.2 Everyone's coding

Computing courses and CS as a science discipline, particularly programming, have gained immense attention in recent years (Azad et al., 2018; Javidi & Sheybani, 2018; Yadav et al., 2016). As pointed out by Xinogalos et al., (2018), “the course with the title *Introduction to Programming* or similar exists almost in all bachelor studies of Computer Science, and Information and Communications Technology (ICT)” (p. 288). Consequently, learning an appropriate programming language during the first course in CS, is of paramount importance. Furthermore, Papert (1980) elaborates that programming shares deep notions of disciplines such as Science, Mathematics, Technology and Model Building. Thus, courses not related directly to computers, can include Computer Programming, such as Engineering, Analytics or Business. There is a growing popularity of Computer Programming, due to initiatives such as *Computer Science for All* in the United States of America, *Computing at School* in the United Kingdom, *Digital Technologies* in Australia and *Africa Code Week* in Africa. There is the *hour to code* campaign on a global platform that offers an online (<https://hourofcode.com>) interface to promote Computer Programming. As remarked by Combéfis et al. (2016), “learning of programming, and more generally, of Computer Science concepts is now reaching the public at large. It is not only reserved for people who studied Informatics or programming anymore” (p. 39).

There is a popular trend to move away from text-based coding to a more visual graphical coding that offers a Graphical User Interface (GUI) with the drop and dragging of blocks, colourful connectors and sometimes based on gamification principles. This coding style is commonly referred to as block-based programming (Homer & Noble, 2017; Inayama & Hosobe, 2018; Rodríguez et al., 2017). Integrated Development Environment (IDE) or workflow is a software editor that is used to create programs. Block-based coding IDEs offer a non-text-based interface to create code like Alice (Figure 2, next page), Beetle blocks, Codingame, Code Flights, Leekwars, Kodu, and Scratch (Figure 3, next page). Calao et al. (2015) argue that such non-text base programming languages have aroused the interest of the educational community in

coding, but not simply for learning code, but as a tool to develop other skills and motivation among students.

Figure 2

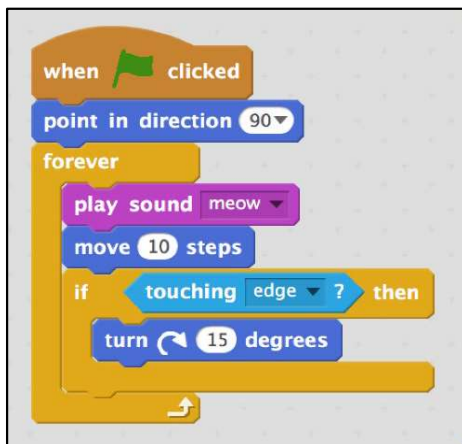
An illustration of Alice



Note. A snippet of the Alice software.

Figure 3

An illustration of Scratch



Note. A snippet of the Scratch software.

The exposure to block-based programming provides a visual graphical experience of coding. However, it can offer a lenient expectation of Computer Programming. There could be challenges when one transits to a text-based language (Chetty & Barlow-Jones, 2012) due to the lack of authenticity when coding with a block-based editor rather than a text-based editor.

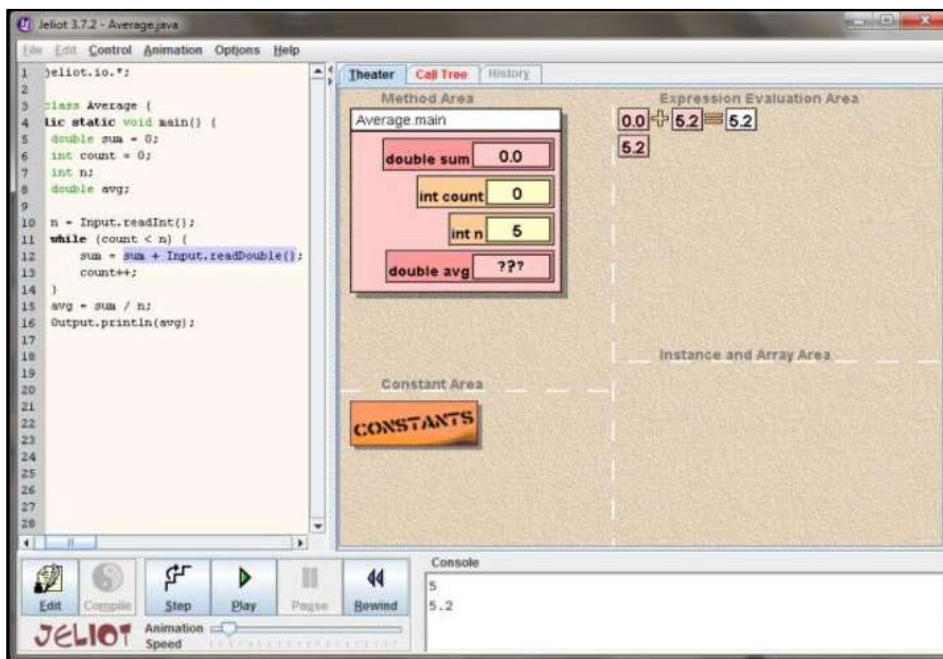
Block-based programming environments offer highly graphically interfaces that are popular for introducing young children and beginners to programming.

However, mastery of text-based programming continues to be the educational goal for students who aim to program into their teenage years and beyond (Kölling et al., 2015). The transitioning from block-based programming to text-based programming forms a significant gap between the two editing styles and presents a difficult challenge in the learning and teaching of programming. Many studies have reported a successful transition from block-based programming to text-based programming (Boldbaatar & Sendurur, 2018; Armoni et al., 2015), but have also reported difficulties when students attempt to transfer concepts from block-based to text-based programming (Chetty & Barlow-Jones, 2012; Kölling et al., 2015).

The very essence of program code is an execution of a well-designed algorithm. There are a number of program visualisation and simulation tools that can illustrate program-code execution steps and runtime behaviour. As pointed out by Calao et al. (2015) and Rajala et al. (2008), such tools have proven to bring about an improvement of students' experiences in introductory programming courses at school and at tertiary level. Such visualisation tools include Jeliot3 (Figure 4), UUhistle (Figure 5, next page) for Python, JIVE for Java, and Teaching Machine for C++.

Figure 4

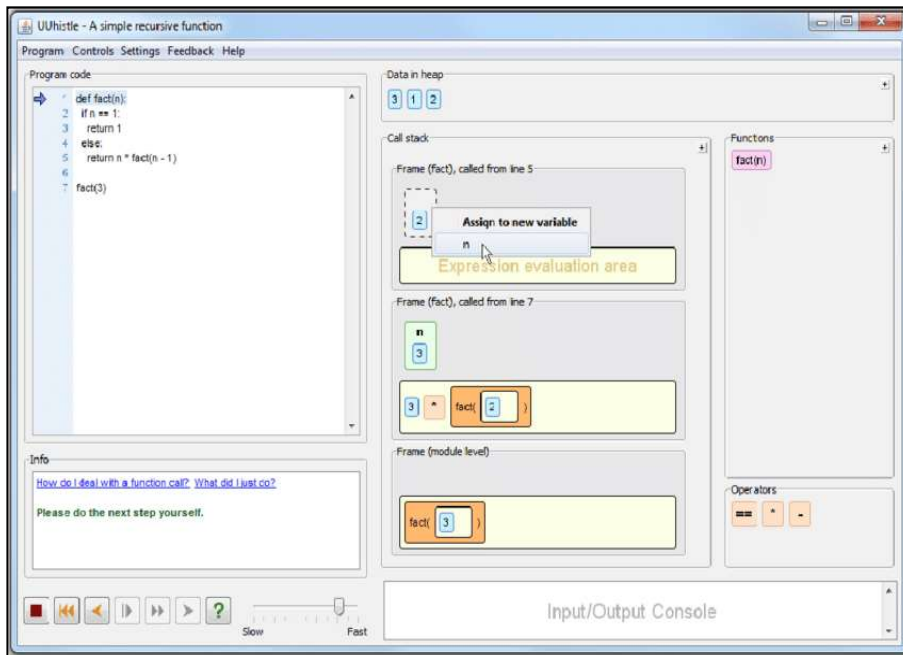
An illustration of Jeliot3



Note. A snippet of the Jeliot3.

Figure 5

An illustration of UUhistle



Note. A snippet of the UUhistle.

Learning how to code consists of a two-phase learning process – firstly familiarising oneself with the IDE, and secondly understanding the general syntax of the programming language. Any Computer Programming language consists of common key concepts such as variables, data types and recursive structures, which are not only important for coding, but also for Mathematics and Physics (Derus & Ali, 2012; Miles, 2016; Rogalsi & Samurca, 1993). It is well-known that such subject concepts can be challenging to teach with traditional teaching styles such as chalk and talk, more so relating to the programming language, making it difficult to comprehend (Bubica & Boljat, 2015). Although chalk-and-talk pedagogy lacks interactive learning engagement, this problem cannot be mitigated by the incorporation of pedagogies that use ICTs like multimedia, projector, clickers, etc. These, are not sufficient to make Computer Programming easy. Amid the two-phase learning process, one needs to develop a strong understanding of the syntax of the programming language and develop problem-solving abilities. Research carried out by Bubica and Boljat (2015) found that students who enrolled in a CS introductory course achieved much lower than their lecturer’s expectation. A similar study carried by Utting et al. (2013) produced similar results. Computer Programming structures requiring verbose syntax can be too abstract for the new programmer. In addition, the lack of physical representation, with only a mental representation of the code available until execution provides a visual representation, can also present challenges for the programmer.

2.3 Teaching and learning of code

While there might be a global trend towards learning to program, it is vital to train a significant number of computing teachers so they are well equipped (Chandler, 2017; Gal-Ezer & Stephenson, 2010; Lye & Koh, 2014; Ni & Guzdial, 2012). Programming modules in CS Education play a major role in developing a successful CS teacher. Knowledge of programming is essential for CS and is a core skill that a computer scientist should be competent in. The complexities of coding structures and concepts such as variables, loops, arrays, functions and syntax can be barriers to learning programming (Topalli & Cagiltay, 2018). However, the known knowledge, experience and skills that students possess when they first start programming are likely to facilitate the acquisition of programming knowledge and skills. This known knowledge is captured in the construct of the Zone of Proximal Developmental (ZPD), which is related to work by Vygotsky. Such knowledge and skills include natural language competencies, various reasoning skills and prior mental exercises that involved abstract thought.

Tending towards collaborative learning, the peer-to-peer pedagogical method has many benefits for learning programming. It stimulates creative thinking, which contributes to swiftly finding a solution to the programming problem (Govender & Govender, 2016; Padmanabhuni et al., 2012; Porter et al., 2016). Thus, in practice, it is crucial to embrace and encourage such pedagogy. However, the downside of peer-to-peer learning is that students may differ in their degree of understanding of code. Although this is the core advantage of collaboration among students, some students may be hindered in their progress due to their peers not having the same level of understanding and skill. A case study by Leyk et al. (2017) suggested that careful implementation and planning are needed for peer-to-peer learning to prove successful. As pointed out by Govender & Govender (2016), “not all learning and teaching environments are conducive to implementing peer-to-peer” (pg. 64).

2.4 Styles of programming

Distinguishing the level and complexity of programming style falls into two broad categories, namely, sequential and parallel programming. Sequential programming involves one instruction at a time carried out in a linear sequence as opposed to multiple instructions. The latter describes parallel programming where instructions are carried out concurrently. The

programming style has a huge impact, especially in scientific, engineering and commercial sectors (Feng et al., 2017). The type of environment should determine Central Processing Unit (CPU) utilisation and program efficiency at run-time.

There are different programming trends, namely, imperative, procedural, declarative, reflective structured, object-oriented, functional, etc. (Samuel, 2017). These trends or approaches to programming are termed paradigms. According to Samuel (2017), “the Object-Oriented Programming (OOP) paradigm can be considered as the dominant programming paradigm” (p. 38). On the other hand, Procedural-Oriented Programming (POP) is found to be the more common and favoured approach (Zuhud et al., 2013). As stated by Samuel (2017), “most widely and extensively used programming paradigms are Object-Oriented and Procedural-Oriented (Husain et al., 2016; Samuel, 2017). However, it is important to note that Hendrix and Weeks (2018) argue that POP is better suited to beginners. Similarly, a study by Husain et al. (2016) recommends POP at an earlier stage with a later switch over to OOP later. This switch, known as an Object-Later approach, has proved to be the best when teaching beginner programmers (Dmitrieva et al., 2019). This could mainly be due to the fact that POP will not require as much abstract or complex thought as OOP, and that the foundations of basic programming structures can be mastered before transitioning to OOP. Many academic planning committees face a predicament about which programming paradigm and language to use in introductory courses (Aleksić & Ivanović, 2016) as this has a ripple effect as one progresses and develops as a computer programmer.

These two commonly used paradigms, POP and OOP, both utilise high-level programming languages to solve problems and are largely dependent on the language used (Dmitrieva et al., 2019; Kurdi, 2013). Paradigms are simply a design concept or ideology and are not directly related to any particular programming language. The OOP paradigm focuses on data, revolves around classes and objects, and follows principles of inheritance, abstraction, encapsulation and polymorphism. Some common examples of languages that can be coded using an OOP approach are Java, C++, Python, R (Dmitrieva et al., 2019; Hendrix & Weeks, 2018; Samuel, 2017; Zuhud et al., 2013).

Object-Oriented Programming follows a bottom-up approach for designing a program (Kühn & Cazzola, 2016). The primary focus is on the data, achieved by dividing a program into methods that are bundled within the objects. As remarked by Samuel (2017), “Object-Oriented

Programming is an engineering approach for building software systems which are based on concepts of classes and objects that are used for modelling the real-world entities, which changes the focus of attention from code to data” (p. 40). For example, a *Car* main class/unit will probably derive properties from a blueprint (object) class/unit called *Vehicle*; thus, the programmer will start from the basics and develop a more complex design. Finally, communication between the driver/main class and blueprint/object classes is created, which enhances data security and modularity and avoids redundancy.

The POP paradigm follows a step-by-step approach that breaks down a task into variables, routines, and subroutines through a sequence of instructions. Execution systematically takes place and the program is divided into small parts called functions. Then it follows a series of computational steps to be carried out in order. It follows a top-down approach to solve a problem (Kühn & Cazzola, 2016). The solution is divided up into functions that are required to accomplish the task. Typical languages include C and GO (Nanz & Furia, 2015; Zuhud, 2013). As mentioned earlier, some programming languages can be coded following many paradigms; for example, a Python solution can be approached in object-oriented, imperative, functional, procedural or reflective paradigms (Ferrari et al., 2016; Nanz & Furia, 2015).

Multiple studies have reported that students face difficulties transitioning from POP to OOP and less likely vice versa (Biju, 2013; Govender, 2010; Liu et al., 2016; Nasrawt & Lam, 2019). As contended by White & Sivitanides (2005), “learning to program in an object-oriented style seems to be very difficult after being used to a procedural style” (p.333). Object-Oriented Programming is generally considered difficult to comprehend. This is likely due to its design concept, coupled with abstraction, encapsulation, polymorphisms and inheritance that affect the coding design, structure and style (Hourani et al., 2019). The complexity of OOP is more applicable to the student able to make a paradigm shift from POP to OOP, since it appears to require more cognitive power. However, a study by White & Sivitanides (2005) argued that “cognitive requirements are not the cause for the difficulty in shifting from procedural to OOP” (p.333). They advocate that the interference of learning POP prior to learning OOP makes the transition difficult. Thus, the move away from of POP is favoured, and many universities have opted to teach their first programming course in OOP (Al-Jepoori & Bennett, 2018; Hosanee & Panchoo, 2015; Kölling, 1999; White & Sivitanides, 2005).

In either case, OOP remains a problem as students have difficulties in comprehending the essence of OOP, and thus perform dismally. Even though there are well research approaches that make OOP easy, it remains a challenge (Hosanee & Panchoo, 2015).

A study by Stueben (2018) reports that there is no substantial difference between OOP and POP in terms of productivity. In general, programming is done with a fusion of approaches. For example, in OOP one can subdivide the problem by examining the objects, which is a top-down approach; followed by polishing the code and merging those into the final program, which is a bottom-up approach. This study chose to utilise a POP to introduce students to programming. In support of this choice, OOP is cumbersome for small projects (Huang et al., 2018) and first-time programmers (Dmitrieva et al., 2019); thus, the nature of the paradigm will not be appreciated to its full potential.

The study will focus on the introductory programming concepts (further discussed in *Chapter four: Research design and methodology* and *Chapter five: Iteration*): data types, variables, arguments, iteration, conditions and calculations. As concluded in a comparative study by Dmitrieva et al. (2019), the introduction of programming using POP followed by OOP, as compared with just using OOP, was imperative as it provided the necessary knowledge and skills required for OOP.

2.5 Proficiency in coding

2.5.1 Skill expectancy

Computer Programming has always been challenging, especially to novices (Bati et al., 2014; Lo et al., 2015; Olsson et al., 2015;). As remarked by Saeli et al. (2011), “programming is a skill that is considered hard to learn and even after two years of instruction” (p. 74). In the earlier years of programming, Sleeman (1986) likened Computer Programming to Latin in the curriculum. As this description is supported by Lions and Peña (2016), Latin was by no means an easy language to learn. However, it is peculiar that Computer Programming is seen as a challenging learning curve since there are many other complex subjects. The art of programming can translate from one language to another, like English to isiZulu or Afrikaans, while keeping the meaning intact.

This means that, if programming based on a particular language is understood, the same concepts can be applied in another language with only the syntax of the language changing. Some of the expected skills one needs to develop to become a programmer, and which may explain the steep learning curve, are as follows:

- Being able to error detect, debug code (Lee et al., 2018);
- Being able to develop CT skills (Barr & Stephenson, 2011; Voogt et al., 2015);
- Being a problem solver (Mead et al., 2006; Xinogalos et al., 2018);
- Being able to translate a problem solution into the required syntax of the chosen programming language by forming a program (Tsai et al., 2019; Yoshiaki et al., 2015);
- Being able to engage in structured thinking, which involves being able to plan, construct and manage the use of IPO charts, Algorithms (pseudocode and flowcharts), UML diagrams, case diagrams, etc. (Calderon et al., 2015; Soloway, 1986);
- Being able to interpret larger entities of a program instead of smaller details (Hamzah et al., 2019);
- Being able to recognise that a problem may have many different solutions; thus, no one solution fits all problems (Bogdan, 2018; Martin & Soares, 2017), but they will all have something in common;
- Being able to identify which code structure to use and when to use it (Bau et al., 2017), and its efficiency (Fagan, 2002).

Computer Programming can be challenging because the skills, as listed above, involve one to think on an abstract level. Unfortunately, South African learners are often not afforded the opportunity to develop the core knowledge and skills needed for proficiencies, creating a learning gap that becomes evident when they get to tertiary level. The South African component of TIMSS (Trends in International Mathematics and Science Study) has assessed Mathematics and Science achievements among students since 1995.

The latest TIMSS conducted in 2015 found that Grade 9 levels showed notable improvements in Mathematics at the lower and higher end of achievement scores as compared with a prior study in 2011 (Reddy et al., 2016). Even though South Africa showed improvement in its performance in Mathematics scores since 2003, it is still a concern to see that South Africa remains at the lower end of the rank order as compared with other countries.

As remarked Reddy et al. (2016), “South Africa is one of the lower performing countries in Mathematics in comparison with other participating countries, and the national average falls short of the lowest international benchmark set by TIMSS” (p. 16).

These learners likely lack cognitive progression, which enhances abstract thought development. They, therefore, struggle with abstract concepts when presented with subjects such as Computer Programming. Students sometimes develop a superficial understanding, but when posed with a computer program problem that requires a conceptual understanding, they can encounter difficulties. It becomes a challenge to determine which program constructs and procedures to utilise in solving the problem. Learners who are not cognitively ready for the abstract and complex thinking required for Computer Programming at tertiary level experience programming as challenging, especially weaker academic school leavers with no prior exposure to Computer Programming (Olsson et al., 2015).

2.5.2 Lack of early exposure

None of the school subjects in the South African GET³ (General Education and Training) phase expose the learner to Computer Programming knowledge. The first exposure to Computer Programming is in the FET⁴ (Further Education and Training) phase at Grade 10, provided that the learner selects the subject Information Technology (IT). Information Technology is unlike many other subjects in the FET phase, which build on prior knowledge from GET phase subjects. For example, Physical Science offered in the FET phase succeeds Natural Sciences from the GET phase and, likewise, History and Geography replaces Social Sciences.

These GET phase subjects lay a foundation and prepare learners for the FET phase. It is crucial to emphasise that an IT learner in the FET phase does not have any prior knowledge of coding. As a result, learners are more likely to select subjects that are familiar and relatable to their previous learning experiences. This is illustrated in Table 1, which shows a general decline over a ten-year period in the number of schools offering IT and in the enrolment of learners at Grade 12, the school finishing grade.

³ Grades 6-8 late primary school to early high school

⁴ Grades 10-12 late high school

Table 1*National count of Grade 12 learners enrolled in IT at schools*

Year	National schools	National learners
2018	333	4108
2017	323	4095
2016	340	4346
2015	349	4326
2014	371	4820
2013	346	4874
2012	359	4428
2011	353	4313
2010	381	4884
2009	425	6246
2008	439	6787

Note. Figures extracted from the Department of Basic Education (DBE Annual Report, 2018).

The lack of exposure from GET to FET impacts the future of the learner's tertiary education and field of specialisation after school. This suggests a sense of inadequacy and reluctance to further studies in Computer Programming and related fields. This reluctance is likely to be exacerbated if IT was not selected as a FET subject.

There are numerous approaches to the learning of programming that can make it more easily and simply understood. For example, simulation software, gamified learning environments (Olsson et al., 2015), pair programming (Govender & Govender, 2016), test-first approach (Doshi & Patil, 2016), test-driven approach (Erdogmus et al., 2005), read before write approach, Predict-Run-Investigate-Modify-Make (PRIMM) (Sentance & Waite, 2017); and Class Responsibility Collaboration cards (CRC) (Börstler & Schulte, 2005). Even though there are varying approaches, there is still no agreement about the best approach to the introduction of coding (Cazzola & Olivares, 2015; Plonka et al., 2015; Thota & Whitfield, 2010). This study hopes to provide an approach that is relevant in the Digital Age and creates a positive influence on the learning of programming.

2.5.3 Computational Thinking

To reach a solution to a problem, the student needs to be well-grounded and acquainted with the necessary basic structures in programming. García-Peñalvo (2018) states that “computational thinking can be based on programming, Robotics, control of devices, wearable

or simply unplugged concepts, that is, without any technology and aimed at developing a way to solve problems” (p. 18). This engenders the problem-solving skills and higher-order thinking skills that one needs to develop when learning how to code, which is also strongly related to Mathematics and other scientific fields. Computational Thinking (CT) is an essential skill that needs to be nurtured as it encapsulates other relevant thinking skills that are crucial for a programmer. Algorithmic thinking can be considered a type of mathematical thinking needed for the execution of performing a set of steps. Moreover, it is noted that CT embeds algorithmic thinking (Jacob et al., 2018). The theorisation of CT was popularised by Wing (2006) as “a way of solving problems, designing systems; and understanding human behaviour that draws on concepts fundamental to computer science” (p. 33).

Consensus on a precise definition of CT has not yet been reached (Barr & Stephenson, 2011; Grover & Pea, 2013). In general, however, scholars agree that four elements form the common core of CT:

- Decomposition is breaking down a complex problem into manageable easier parts. These smaller parts can be solved and designed individually since they are simpler to work with;
- Pattern recognition involves an analysis that sifts out similarities and trends that form a repetitive sequence. These patterns can help solve complex problems more efficiently;
- Abstraction is the removal of unnecessary parts of a problem. This filtering-out process ignores the non-essentials and focuses on the relevant characteristic so that the solution works on multiple problems;
- Algorithmic design is the step-by-step instruction or set of rules to solve a problem (Kalelioglu et al., 2016; Romero et al., 2017; Turchi et al., 2019).

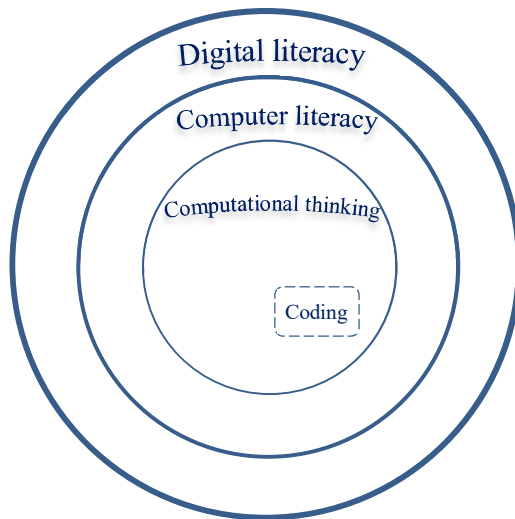
Computational Thinking is an important skill for a new programmer as it facilitates the understanding of the problem and the design/build of the code. It can be agreed from the above four elements that constitute CT, that knowing how to use a computing device or being computer literate does not by any means relate to CT.

Computational Thinking is not exclusively associated with computer/technology activities. Although CT was popularised in the 2000s, it was not a new term. Seminal work by Seymour Papert explains CT as a series of steps for finding solutions to problems related to computer system designs, but also that it can be applied to real-world phenomena (Cansu & Cansu, 2019).

However, when CT is used in a computer context, skills like digital literacy and computer literacy can be a starting point for the development of CT skills. Figure 6 depicts the relationship between various cognitive levels, showing that CT resides in the inner core of the relationship levels.

Figure 6

Cognitive levels between digital literacy, computer literacy and computational thinking and skills



Note. Visual illustration created by Author.

Computational Thinking requires efficient and well-thought-out decisions, problem-solving practices and exercising different thinking systems (Weintrop et al., 2016). As mentioned, CT is not a skill exclusively related to CS, as the problem-solving process may be generalised and transferred to a wide variety of problems (García-Peñalvo, 2018). Thus, CT is by no means synonymous with Computer Programming, but rather a way of thinking that is not exclusive to a person trained in a computer discipline. As remarked by García-Peñalvo (2018), learning activities based on STEM (Science, Technology, Engineering and Mathematics), require the development of CT. There is also a clear distinction between CS and CT, as Denning (2009) pointed out, as the latter contributes only a subset of skills possessed by a Computer Scientist. He derived four components in CS: programming, engineering of systems, modelling and applying. It is important to point out that CT is not part of Computer Programming, but rather Computer Programming is part of CT (Figure 6).

2.5.4 The influence of Mathematics

Mathematics comprises various disciplines such as arithmetic, trigonometry, geometry, calculus, measurement, etc. Mathematics is essential in the development of technology and it is a universal mode of reasoning, logic and constructive thought. Science, Technology, Engineering and Mathematics (STEM) subjects require the exercise and development of CT (García-Peñalvo, 2018), which develop the knowledge needed in specialised fields like architecture, commerce, aviation, etc. As agreed in Agenda 2063, of those who enter tertiary institutions 70% should graduate in STEM subjects (SONA, 2019). It can be concluded that the fundamentals of Mathematics are essential when learning Computer Programming, as the two subjects share the same cognitive stance (Elaine, 2013). Computer Programming, if not CS as a discipline, is rooted in the fundamental definitions, axioms, theorems and proof techniques of Mathematics. In essence, Mathematics provides a language for working with ideas relevant to CS (ACM/IEEE-CS, 2009). Furthermore, Rosen (2018) confirms that discrete Mathematics is essential in Computer Science, Engineering and other disciplines since it covers topics like logic, number theory, counting, algorithms, cryptography and number sequences. Conversely, programming concepts such as algorithmic constructs like flow statements (if... else, case, for... do, repeat... until, while, etc.), variables, data types, etc., are deeply rooted in Mathematics and CT (Lie et al., 2017).

In a seasoned view, Graham et al., (1989) stated that “I would advise the casual student to stay away from this course” (p. 107) in reference to learning Mathematics as a foundation for CS. It is relevant to note that computers were first developed to help solve difficult mathematical problems, and date back to the first computer, the Abacus. According to Misfeldt and Ejsing-Duun (2015), Computer Programming and Mathematics should be viewed as closely connected disciplines since the first computers were conceptualised and built by mathematicians. The Abacus eventually evolved over the centuries to be replaced by the electronic calculator: a tool used to calculate complex Mathematics. As remarked by Lie et al., (2017), from a “historical point of view, computers were constructed to perform mathematical computations” (p. 29). It would seem that Mathematics is a commodity necessary to learn how to program. Some of the mathematical content explored in Computer Programming are number sequences, frequency, abundant numbers, deficient numbers, perfect numbers, triangular numbers, squares, cubes, palindromes, factorials, Fibonacci numbers, maximum and minimum common

divider of two numbers, prime and composite numbers, to mention a few. The use of computers has perpetuated the belief that programming is the domain of the strong Mathematics adepts.

Findings by Bubica and Boljat (2015) establish that assessing university programming students based on those who have previous experience of programming at school and those who have poor Mathematics knowledge, has proven to be ineffective. Thus, indicating that Mathematics might not be the essential element needed to understand programming. However, Pea & Kurlan (2018) are adamant that Mathematics and programming skills are related once general intelligence is factored out. This gives rise to what is defined as general intelligence, and does this omit one's mathematical ability? However, Mathematics is still seen as an effective benchmark of having the required cognitive level to learn Procedural-Oriented Programming (POP) (Elaine, 2013; Kalelioğlu, 2015; Soykan & Kanbul, 2018; White & Sivitanides, 2003).

Some intuitions have adopted new approaches to the pre-requisites that must be met for a student to register for a Computer Programming course. Jackson and Miller (2009) emphasise that students must possess the ability to think abstractly since most enter the course with minimal mathematical knowledge, or only school-leaving Mathematics. Hence, all first-year applicants are required to complete two calculus modules. Along similar lines, ACM/IEEE-CS (2013) strongly suggests that all undergraduate courses in CS must include Mathematics as part of the curricula; however, it was noted that such requirements vary by institution due to several factors.

Among South African universities, the general pre-requisite into a course with Computer Programming is that the school leaver must have achieved at least a level 5 or 6 in Mathematics (Stellenbosch University handbook, 2019; University of Cape Town handbook, 2019; University of Kwa-Zulu Natal, 2019; University of the Witwatersrand handbook, 2019). This includes disciplines such as CS, CS Education, and Information Technology – all of which contain Computer Programming modules.

The link between Mathematics and Computer Programming has led to some countries like France to make Computer Programming compulsory in the school curriculum, similar to Mathematics and language studies (Pea & Kurlan, 1984). On the other hand, in recent times, countries like England and Sweden have linked programming to design and engineering subjects in schools (Misfeldt & Ejsing-Duun, 2015). Along similar lines, the National Council

of Teachers of Mathematics (2016) has announced that many states in America allow CS courses with particular emphasis on Computer Programming to satisfy either a Mathematics or a science course requirement. Furthermore, Saeli et al. (2011) state that in the “Netherlands, Informatics has been defined as a new generation discipline, because it is linked with Mathematics, Physics, Engineering, Linguistics, Philosophy, Psychology, Economy, Business, and Social Sciences” (p. 74). The use of Robotics and Mathematics have a history together as a result of the seminal work by Papert. According to Papert (1980) Robotics are to be used to, “externalize learner’s ideas and to make mathematical concepts more accessible to reflection (p.145). This has significant implications for visual reasoning in Mathematics. Papert’s work dealt with the Logo turtle program and geometric figures.

2.6 Robotics

2.6.1 Why Robotics?

The conception and design of robotics was begun in the 1950s by George Devol, who created a robot called Unimate (Gasparetto & Scalera, 2019). In the Digital Age, we harmoniously interact with robotics and Artificial Intelligence (AI) systems on a daily basis without being aware of such encounters, as such interactions have become a norm in our daily lives. Robotics and AI are used in assembly plants, automatic opening and closing doors, boom-gate detection, and voice control commands on smartphones such as Siri, Alexa and Cortana. The use of robotic equipment is found in many work sectors whereby humans assign them tasks that facilitate their work (Gasparetto & Scalera, 2019; Othman et al., 2018).

In summary, Robotics are machines that are instructed to carry out tasks that a human would normally perform. There are some cases where Robotics are used to perform tasks that are too dangerous or hazardous to human beings (Moniruzzaman et al., 2018). Such cases include bomb diffusion, military tactics, a rescue mission in rough terrains, exploration of unknown regions, etc.

Humanoid development is active on a progressive scale in many parts of the world. Humanoids are systems that, in appearance, resemble a human and have a highly interactive bodily autonomy. Famous humanoids such as Honda’s Asimo, and Atlas by Boston Dynamics, are well known in the Robotics community. However, more recently, the development of Sophia

has gripped the attention of many. Sophia was created by Hansen Technologies and is the first Robotic AI system to gain citizenship in a county (Kalra & Chadha, 2018; Weller, 2017).

The use of ICTs in education are becoming more readily available and intertwined with learning environments, and are changing the landscape in which one learns. As South Africa moves from the Third to the Fourth Industrial Revolution, increasing attention is being given to technological advancements such as Robotics, AI, Big data and the Internet of Things (IoT), to mention a few. It is vital to integrate such advancements into learning since the current generation should learn about skills that are relevant to Workplace 2030 and future economy growth (further discussed in sub chapter 2.9 *Workplace 2030*).

2.6.2 Robotics as a learning aid

As mentioned earlier, robot-based activities started with the work of Seymour Papert. This involved the learning of Geometry and Logo platforms under the pedagogical framework of Constructionism. The focal point being an unintentional learning encounter on how to code using the Logo platform while learning Geometry. The Logo platform consisted of special words like fd- forward, rt- right turn, etc., that used to control the actions of an arrow referred to as 'turtle'. Figure 7 depicts a drawing of a square. The code for the output in Figure 7 is shown in the commander window in Figure 8.

Figure 7

Microsoft Windows Logo output of a square

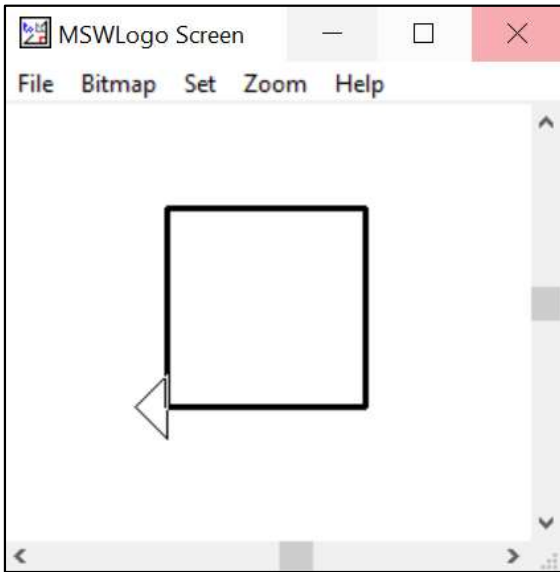
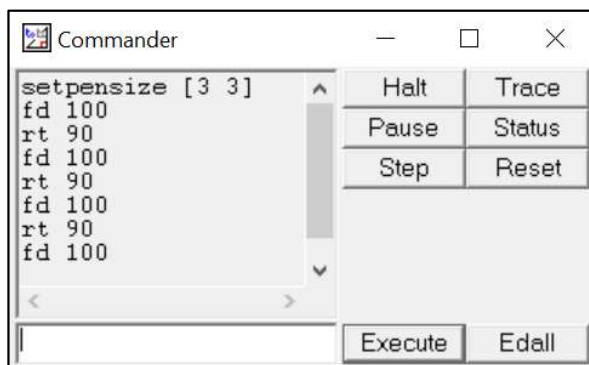


Figure 8

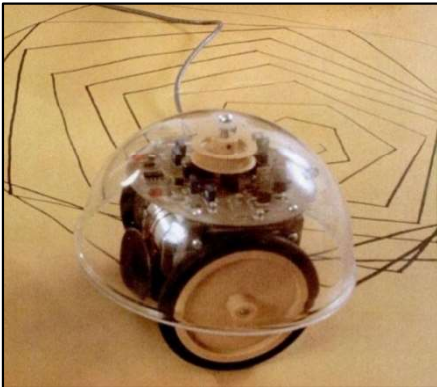
Code in Logo commander window



In the 1960s, Papert and the MIT (Massachusetts Institute of Technology) AI laboratories saw the dawn of the physical turtle device, which had a pencil attached for drawing (Figure 9). According to Papert (1980), the turtle is seen as a metaphor, an "object-to-think-with" (p. 12).

Figure 9

Turtle device



Note. Extracted from <http://cyberneticzoo.com/cyberneticanimals/1969-the-logo-turtle-seymour-papert-marvin-minsky-et-al-american/>

After many years it is important to note that there were some key limitations with this early robotic device: a lack of mobility since it had to be connected with a cord; no additional sensors; and a non-modular design. There is a scarcity of studies examining the use of Robotics in the classroom (Toh et al., 2016) and coding in a text-based editor. The studies that do involve robots and the learning of code focus on elementary level programming. Thus, utilising block-based environments similar to that of Scratch. In addition, many studies, if not all, target the development of thinking skills, problem-solving skills, motivation (Chin et al., 2014; Karim et al., 2016) and teamwork (Toh et al., 2016) of students. Therefore, this study will explore a unique way to introduce text-based programming by using Robotics as an educational tool to make coding concepts in text-based programming simple to understand while developing students' interests in programming.

2.6.3 Lego Mindstorm series

Several robot kits of different types and brands are available on the educational market. The activities with robots involving design-build, problem-solving, block-based programming and engineering skills have gained much popularity (Karp et al., 2010; Kucuk & Sisman, 2017; Toh et al., 2016; Varney et al., 2012). The Mindstorm series is a set of buildable and programmable robotic kits made by Lego. The Lego Mindstorm provides a stable interface, strongly built components, and a modular design. It appears that most of Papert's work over the years, if not all, was intended for the development of the Lego Mindstorm series (Catlin et al., 2018; Chesher, 2018; Resnick et al., 1988).

The Lego Mindstorm kit is provided with a microcontroller (usually referred to as the intelligent brick) modular motors, sensors and Lego bricks. There are two size motors supplied (large and medium) which can be moved in rotations, degrees or seconds. Each motor has the ability to read in data, such as, the state of the motor, whether moving or stalled, and the number of degrees rotated. The sensors read in data from their immediate surroundings and feed the data to the intelligent brick. The colour, ultrasonic, gyroscope, touch and light sensor are commonly supplied with the kits (Aslam et al., 2018; Kovács, 2019).

There are three generations of Lego Mindstorm, namely RCX (Robotics Invention System), NXT (next generation) and EV3 (evolution 3). Each generation has its design with different parts and intended use. Coupled with the modular design of the intelligent brick, the motors and the sensors, accuracy has been progressively improved in the latest Lego Mindstorm, the EV3. The releases are as follows:

- RCX (Figure 10) launched in January 1998 with four available kits. One basic kit, one educational kit, and two upgraded versions of the basic kit;
- NXT (Figure 11) launched in August 2006 with three kits available. One basic kit, an upgraded version of the basic kit and an educational kit;
- EV3 (Figure 12) launched in September 2013. Currently, two kits are available, a basic kit and an educational core kit (Aslam et al., 2018; Catlin et al., 2018; Kovács, 2019; Kuncoro et al., 2018). *⁵

Figure 10

First generation RCX



⁵ During this study, Lego had launched the Robot Inventor in 2020, which is said to replace the EV3.

Figure 11

Second generation NXT



Figure 12

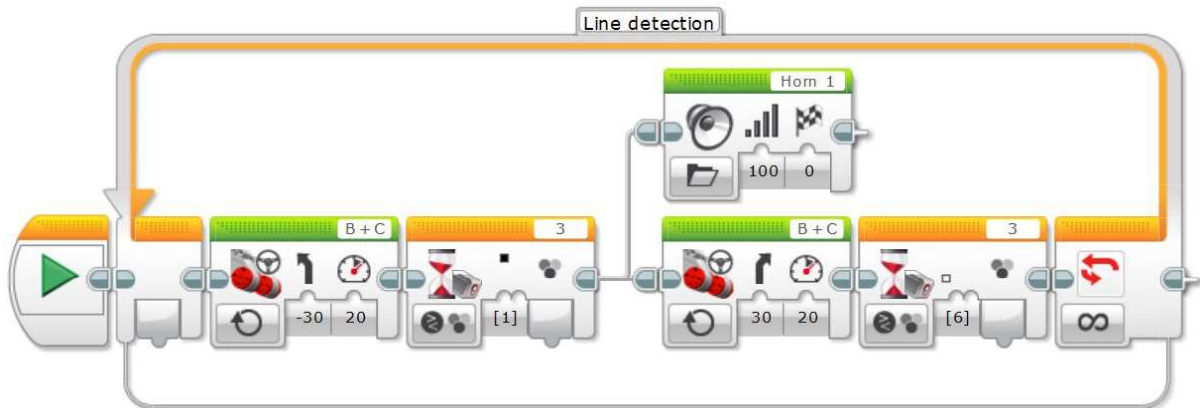
Third generation EV3



LEGO Mindstorm programming software (IDE) allows the user to create code and upload it to the intelligent brick. The programming software has a rich GUI design allowing actions of the brick to be created by dragging and dropping blocks together to make a program. Figure 13 (next page) shows an exemplar program using LEGO Mindstorm programming software.

Figure 13

EV3 program that detects a black line



The block-based code in Figure 13 runs in an infinite loop that sets the steering direction of motors B and C (wheels) -30 (left) at 20% power. The colour sensor continuously reads in data; once the colour black is detected, the program splits into two streams of blocks that run simultaneously. The first stream contains one block that outputs a motor horn noise while, at the same time, the second stream sets the steering direction of motors B and C +30 (right) at 20% power until the colour white is picked up. This process continues in an infinite loop until the user terminates the process. In summary, the program keeps the robot off the black line.

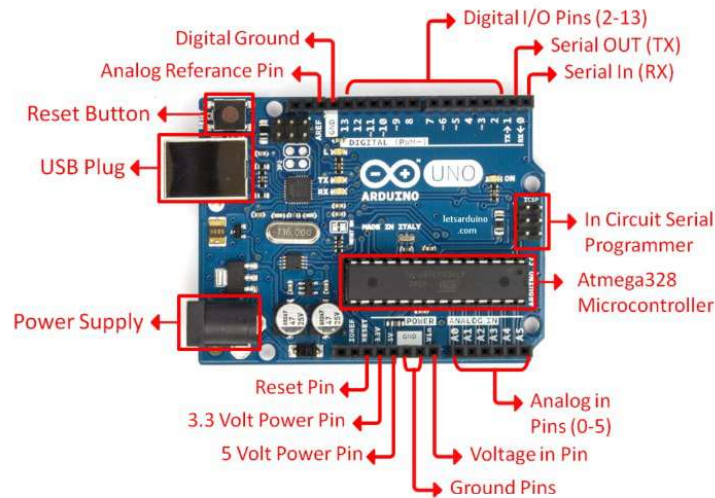
2.6.4 Arduino

The Arduino microcontroller board was first launched in 2005 in Italy (Barrett, 2020). There are many variations of the microcontroller board due to Arduino being an open-source hardware and software company. Compared with Lego and similar kits, Arduino provides a cost-effective way for people to code and design prototypes⁶ that can interact with the immediate environment. Arduino kits are generally supplied with the single-board microcontroller, LEDs (Light-Emitting Diodes), LDRs (Light Dependent Resistors), jumper wires, resistors, pin headers, a breadboard, a 9V power connector, and USB (Universal Serial Bus) connector. One of the most popular Arduino microcontroller boards is the UNO R3. The UNO R3 board (Figure 14, next page) has 14 digital pins, six analogue pins and can produce either a 3.3V or 5V output from a USB or power connector.

⁶ A prototype is an early model which is used for test purposes.

Figure 14

Arduino UNO R3 pin diagram.



Note. Adapted from <https://www.elprocus.com/what-is-arduino-uno-r3-pin-diagram-specification-and-applications/>

Unlike commercially based educational robot kits that are difficult to modify mechanically and electronically despite their modular structure, Arduino allows for such flexibility (Pérez & López, 2019). Arduino has various add-on sensors and actuators that can be connected to the microcontroller and coded in the default Arduino language and IDE. The Arduino language requires a steep learning curve which can hold back many users (Russell et al., 2020). The Arduino language is based upon C and C++. The programming of the microcontroller forms the basics of Robotics. Hence, it is autonomous in its immediate surrounding environment. There are many add-on libraries to the Arduino IDE providing unrestricted access due to the open-source community.

2.7 Programming languages

2.7.1 Hello world

There are many different programming languages used to teach programming. Some of the most popular include Java, C, Python, and C++ (Mannila & Raadt, 2006; Sebesta, 2016; Hendrix & Weeks, 2018). Every programming language has its own syntax set; that is the rules one would adhere to when typing code. To select the best programming language can be difficult, as pointed out by Nanz and Furia (2015): “the question on which is the best programming language is often asked, but well-founded answers are not easily available” (p.

779). For this reason, this study presents the code for two tasks in some popular programming languages. The first task examines *Hello World*, which does not involve any calculations; this code outputs text on the screen. The second task is the Fibonacci sequence which demands some complex calculations coupled with nested coding constructs.

In Figures 15 to 20, the famous *Hello World* program is examined in some of the common programming languages like Java (Figure 15), C (Figure 16), Delphi (Figure 17), C++ (Figure 18), Python (Figure 19) and C # (Figure 20). The *Hello World* program is generally the first program that students code in during an introductory programming course.

Figure 15

Hello world code in Java

```
Java
public class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println("Hello, World!");
    }
}
```

Figure 16

Hello world code in C

```
C
#include <stdio.h>
int main()
{
    printf("Hello World");
    return 0;
}
```

Figure 17

Hello world code in Delphi

```
Delphi
program HelloWorld;
begin
    Writeln('Hello, world!');
end.
```

Figure 18

Hello world code in C++

```
C++
int main()
{
    std::cout << "Hello, world!";
    return 0;
}
```

Figure 19

Hello world code in Python

```
Python
print("Hello, World!")
```

Figure 20

Hello world code in C#

```
C#
using System;
class Program
{
    public static void Main(string[] args)
    {
        Console.WriteLine("Hello, world!");
    }
}
```

The Python (Figure 19) programming language has the least lines of code from the selected programming languages. The code is as follows: `print ("Hello, World!")`, which is

all that is needed to display *Hello world!* on the screen. The fundamental difference between C and C++ is that C is procedural-based (Dmitrieva et al., 2019). This means it does not support the coding of classes and objects. Java (Figure 15), C (Figure 16), C++ (Figure 18) and C# (Figure 20) all seem to be verbose and heavily laden with syntax due to the syntactical need for parenthesis, round brackets, semi-colons and apostrophes. Java and C# seem to share very similar syntactical structures. A study carried by Farag et al. (2013) compared the effectiveness of C++ among a control group and Java among an experimental group. There were no significant differences between the groups regarding academic results and gratification. This is because both languages are based on C and C++ (Dmitrieva et al., 2019).

Python uses indenting to recognise the beginning and end of code structures compared with the parenthesis in the other languages. Except for Delphi, which does not require parenthesis as compared with Java, C, C++ and C#, it utilises the actual words *begin* and *end*. Overall, it can be said that Python is a coding language with less typing, requiring however its own syntactic needs.

The Fibonacci sequence is a well-known number pattern used in introductory courses in Computer Programming. The pattern starts with 1, followed by 1, and continues to add the current term to the previous term to get the next term which is 2; 3; 5; 8 and so on. In the code segments (Figure 21-26), the user enters *n* as the number of terms that must be generated for the Fibonacci sequence.

Figure 21

Fibonacci code in C

```
C
long long int fibb(int n)
{
    int fnow = 0, fnext = 1, tempf;
    while(--n>0)
    {
        tempf = fnow + fnext;
        fnow = fnext;
        fnext = tempf;
    }
    return fnext;
}
```


Figure 22

Fibonacci code in Java

```
Java
public static long itFibN(int n)
{
    long ans = 0;
    long n1 = 0;
    long n2 = 1;
    if (n < 2)
        return n;
    for(n--; n > 0; n--)
    {
        ans = n1 + n2;
        n1 = n2;
        n2 = ans;
    }
    return ans;
}
```

Figure 23

Fibonacci code in C++

```
C++
#include <iostream>
int main(int target)
{
    unsigned int a = 1, b = 1;
    for(unsigned int n = 3; n <= target; ++n)
    {
        unsigned int fib = a + b;
        std::cout << "F("<< n << ") = " << fib << std::endl;
        a = b;
        b = fib;
    }
    return 0;
}
```

Figure 24

Fibonacci code in C#

```
C#
public static ulong Fib(uint x)
{
    if (x == 0) return 0;
    ulong prev = 0;
    ulong next = 1;
    for (int i = 1; i < x; i++)
    {
        ulong sum = prev + next;
        prev = next;
        next = sum;
    }
    return next;
}
```

Figure 25

Fibonacci code in Delphi

```
Delphi
function Fib(inum: integer):String;
var num, first, second, next, c:Integer;
toString:String;
begin
  first := 0;
  second := 1;
  C:=0;
  num:=inum;
  while C<num do
  begin
    if C<=1 then
    begin
      next:=C;
    end else
    begin
      next:=first+second;
      first:=second;
      second:=next;
    end;
    inc(C);
    toString:=toString+IntToStr(next)+'#9;
  end;
  result:= toString ;
end;
```

Figure 26

Fibonacci code in Python

```
Python
def fibIter(n):
  if n < 2:
    return n
  fibPrev = 1
  fib = 1
  for num in xrange(2, n):
    fibPrev, fib = fib, fib + fibPrev
  return fib
```

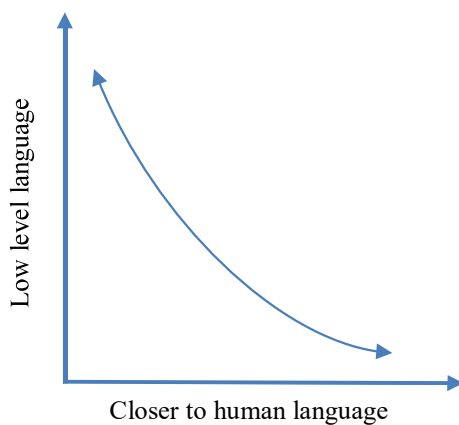
Examining the code indicates that Delphi (Figure 25) uses the most lines of code, whereas C (Figure 21) and Python (Figure 26) uses the least number of lines to code for the Fibonacci sequence. Despite the differing number of lines of code, a study conducted by Xinogalos et al. (2017) found that, between Delphi and C, there was no statistical significance as to which programming language should be used to teach and learn, since students encountered the same level of difficulty with both. This indicates that fewer lines of code do not mean an easy level of understanding. Likewise, more lines of code do not necessarily imply a difficult level of

understanding. The iteration structures and the algorithmic problem solving leading to the calculation of the Fibonacci numbers remain the same across all selected programming languages. When coding in Python, it is important to note that one does not need to declare data types (Nasrawt & Lam, 2019); this means type errors are not likely to occur. Due to the indenting system utilised in Python to mark the *begin* and *end*, this provides less visual clutter as compared with the other programming languages. It is important to point out that the semi-colon is used to denote the end of a line in the selected programming languages, except for Python, which recognises that if code appears on a new line visually, then it must denote a new line. This makes better sense than using a semi-colon, thus earning Python the title of being a truly high-level language. High-level programming languages are closer or similar to human languages and further away from machine language (Dmitrieva et al., 2019).

Figure 27 depicts the relationship between low-level language and human understanding, indicating the level of abstraction related to the language level. A high-level language is closer to human languages and is alien to a computer. Whereas low-level language is difficult to debug and understand (Chapman & Irwin, 2015), but takes up less storage space (Comer, 2017).

Figure 27

Relationship between low-level and high-level language



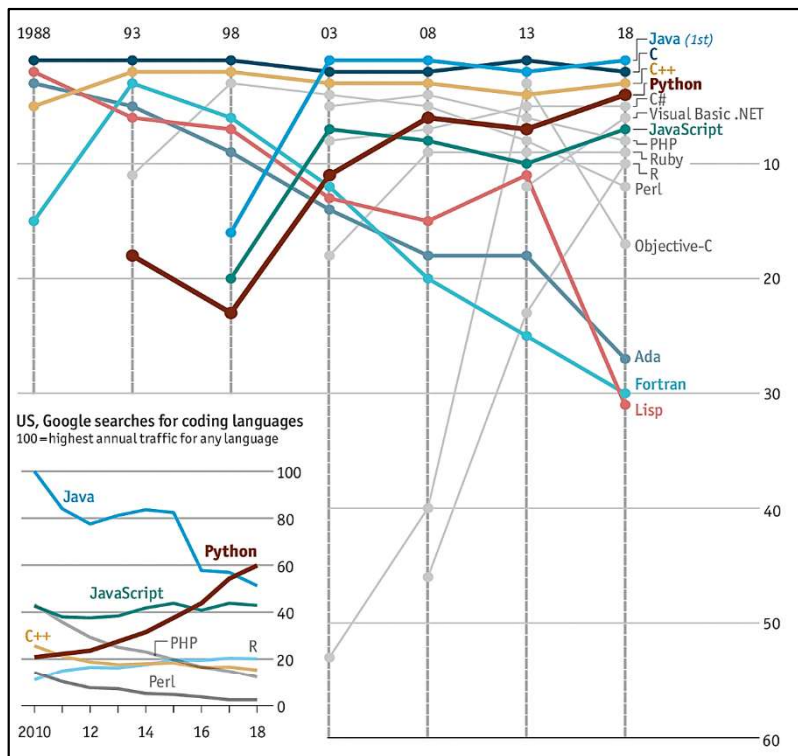
Note. The structure of low-level language is similar to the processor's instructions hence referred to as machine code.

2.7.2 Python Language

Python was developed in the 1980s and first released in 1991 by Dutch programmer Guido van Rossum; and has been increasing in popularity as a recommended language for programming (Dmitrieva et al., 2019; Sharma, 2018; Xinogalos et al., 2017).

Figure 28

Ranking of programming languages



Note. Extract from the Economist, 26 July 2018.

The Economist (2018) reported that Python is becoming the world’s most popular coding language and will soon overtake C, C++ and Java. As can be seen in Figure 28 under US Google searches for coding languages, Python is steadily increasing in popularity. Python is rated first on the IEEE (Institute of Electrical and Electronics Engineers) spectrum ranking 2021 programming language rating and is reported as the fastest growing programming language (Cass, 2021).

This increase could be because Python is a scripting language that takes less time to develop when compared with C++ and Java; and programs written in Python are often shorter than equivalent programs written in other languages because of the built-in high-level data types and its dynamic typing (Shein, 2015).

Python was selected as the programming language used in this study. Mannila & Raadt (2006) provide reasonable evidence that Python is the preferred programming language for learning. In addition, Shein (2015) states that “once someone grasps the logic behind Python, the concepts can be more easily transferred to other languages” (p. 19). Nevertheless, a study by

Aleksić and Ivanović (2016) found that the first programming language used for coding is not significant if the performance is the only measure of standard. Since the results from two non-parametric tests showed no significant differences in the language used and academic achievement.

In another study by Xinogalos et al. (2017) two groups of students were surveyed – twenty-eight without prior programming experience and twenty-seven with some prior experience. The results showed that both groups considered Python as a suitable first programming language. This preference was probably because Python is regarded as a multi-paradigm language with a simple and straightforward syntax, while its popularity in the private sector is growing (Shein, 2015).

An example of Python coding in the private sector can be illustrated by a study that investigated the learning of Python as a first programming language among biomedical students with no prior programming experience (Chapman & Irwin, 2015). The students used Python code in a software editor package for data science, analysis and visualization. Key findings (Table 2), where 4 = *Agree* and 5 = *Strongly agree*, were in favour of Python as compared with other programming languages.

Table 2

Use of Python code in the private sector

Statement	1	2	3	4	5
Learning Python was valuable for helping me subsequently learn additional programming language(s)	1	1	3	12	9
Learning Python was valuable for my career development	0	1	1	10	14

Note. Usefulness of learning Python (data extracted from Chapman & Irwin, 2015, p. 15).

Python supports multiple paradigms (Dmitrieva et al., 2019) and has a simpler syntactical structure that provides students with the opportunity to learn various programming languages (Lo et al., 2015). Consequently, Python is favoured as the first language for learning for novices. Python is one of the most widely adopted high-level programming languages (Summerfield, 2010). Shein (2015) added that more people use Python as an “introductory

programming language because it has a very large set of highly useful libraries that have been built over the years” (p. 19). On the other hand, languages such as C, C++ or Java, are the first learning languages for novices designing real applications; and are therefore popular in industry (Lo et al., 2015). However, the complex syntax of these languages is challenging for novices (Kalelioglu & Gülbahar, 2014), as mentioned in section 2.7.1 *Hello World*. In contrast, Python has some robust characteristics that make it an ideal language for novice programmers (Tollervey, 2015), such as, ease of learning, the existence of an interactive shell that allows programs to execute efficiently, the existence of various libraries, portability and installation flexibility. Action research aimed at reducing the dropout rate after introductory courses to Computer Programming through the use of Python language reported an improvement of 77% of students passing and progressing (Yadin, 2011). Yadin (2011) adds that Python’s simple syntax allows students to focus on algorithm designs and problem solving. This is further supported by Dmitrieva et al. (2019), who states that “Python can be definitely recommended to be used on the first level of study to receive initial programming skills, as well as the second level of study while learning object-oriented programming” (p. 203).

2.7.3 Integrated Developmental Environment (IDE)

There is no single IDE (Integrated Developmental Environment) that is compatible with all the available programming languages (Zayour & Hajjdiab, 2013). This reiterates the two-phase learning process mentioned earlier – learning how to use the IDE and learning the coding language. The IDE used can have a potential impact on the learning process – the individual can direct all their efforts in understanding how to use the IDE and overlook learning about the programming language. As a consequence, this prevents proper learning of the programming language.

Therefore, the choice of IDE that is used is very important. Some factors that need to be considered when choosing an IDE include: Does it support the design of GUI components or text-base only? Does it have predictive auto-completion? Is the program open-source? Does it have any additional syntax? Are debug and trace features available?

The IDE chosen for Python was Integrated Development and Learning Environment (IDLE). The Python IDLE is the default environment that is installed with the Python language package. The default mode when opening IDLE is Python shell. Through the shell window, the user can type and execute code and also create individual Python files (.py) for each set of code.

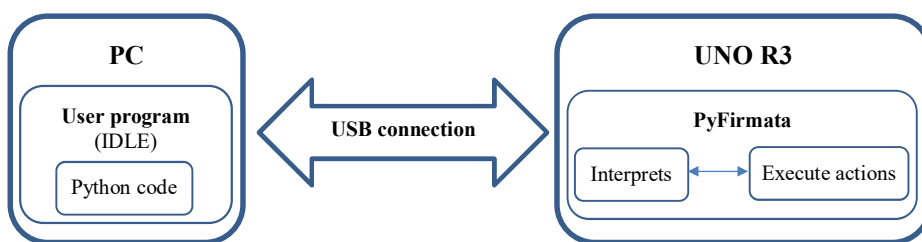
2.8 Setup ⁷

This study makes use of the Arduino UNO R3 microcontroller and Python programming language. The UNO R3 required some setup so that the firmware was able to understand the Python code, rather than its default Arduino language. This was achieved by loading library package PyFirmata to the microcontroller. PyFirmata allows the Python code to be interpreted on the Arduino UNO R3 microcontroller.

It is important to note that reference to the COM port number, which represents the PC port that the microcontroller is connected to, must be made within the Python code. This ensures that the UNO R3 is in a ready state mode waiting for instructions from the PC. The schematics of an UNO R3 loaded with PyFirmata connected via USB to a PC running IDLE are shown in Figure 29.

Figure 29

Schematic of the PC connection with Arduino



During the first connection to the PC, the Arduino UNO R3 drivers are installed automatically to the PC. Instead of being USB powered, the UNO R3 can be operated on a separate 9V battery power. However, for convince, only the USB connection was used as it serves two purposes: providing power to the microcontroller and as a medium for data transfer. In Figure 29, the PC is like the client and the UNO R3 is the server. This type of client-server setup can allow the communication of multiple servers with a single client through different ports. The text-based program is transferred from the client to the server, which gets interpreted and executed. The execution of the output can involve an unlimited number of sensors and actuators running simultaneously.

⁷ The original plan was to use the LEGO EV3 and Python in a form of a face-to-face workshop. However, due to the COVID-19 pandemic and cost factors, the researcher supplied each participant with an Arduino kit and the workshop took place on an online platform.

2.9 Workplace 2030

Currently we are experiencing a transition towards 4IR on the African continent, which includes the promotion and unveiling of AI, Big Data, Robotics and other related developments. In readiness to “ensure that South Africa effectively embraces the Fourth Industrial Revolution, the President has appointed a task team to focus on the Fourth Industrial Revolution” (SONA, 2019). As society begins to change, concomitant changes can be expected in terms of the types of knowledge and skills needed for the future. Thus, educational decision makers have a responsibility to explore new avenues to keep learning relevant. Consequently, it is necessary to explore new ways of introducing concepts such as programming to programming students in the 21st century. As the need to learn in a manner different from their predecessors, future graduates of today need to learn how to apply, design and understand in practice. This study will explore a new style of programming introduction at a tertiary level, which aims to bridge the gap for students who lack prior programming knowledge.

As mentioned in the SONA (2019), key jobs in South Africa where progress is being made are: installation, repair, maintenance jobs and digital/tech jobs like coding and data analytics. Thus, the study is in alignment with goals set out by the South African Government. As remarked by the President of South Africa in the SONA (2019) that “...Framework for Skills for a Changing World, we are expanding the training of both educators and learners to respond to emerging technologies including the internet of things, robotics and artificial intelligence” (p. 25).

2.10 Conclusion

Coding and Robotics are among the most valuable set of skills needed for the Fourth Industrial Revolution. Such digital skills are essential to enable graduates to integrate into the world of work. Instead of rote learning, the future generation needs skills development in unstructured problem solving and reasoning. These critical skills, coupled with coding and Robotics, will drive economic growth into the future.

It is widely accepted that Computer Programming is complex, challenging and students dislike the course, resulting in poor performance and motivation (Chen et al., 2017). Studies found in the literature reveal that the difficulty in learning programming initiates from the complexity of coding structures such as variables, loops, arrays, functions and syntax in programming languages. These complexities may become barriers to learning programming and diminish student motivation.

Studies also reveal that many research projects have been carried out incorporating Robotics and programming in a block-based environment. In most cases, the focus was on developing reasoning and logic and not programming. Those studies that did focus on programming did so in a block-based style environment, mostly with young students. As reviewed, this programming style can be misleading when the individual is later introduced later to text-based programming. The advantage of programming environments that offer a drop-drag interface (block-based) allows one to focus more on logic and reasoning than coding. This is achieved because the programming environment is not based on a particular programming language, so there is no verbose text-based programming and thus no syntax.

This study opts to explore a move away from classical learning platforms, such as block-based programming and computer visualisation tools, towards a more reformed approach using robots to introduce programming.

This approach envisions the use of a real-time three-dimensional learning experience of Computer Programming, rather than an artificial two-dimensional experience on a computer screen. The type of pedagogy required for Computer Programming should be demonstrative and practical in nature, which this study aims to achieve through the use of Robotics as an educational tool (robot coding). Since it is clearly argued in the literature that there is no single most-suitable methodology for the introduction to programming, it is hoped that the strategy employed in this study will yield an authentic experience to facilitate the development of programming knowledge – unlike with block-based programming.

Many of the studies reviewed did not sufficiently adhere to the requirements for rigorous education research. Thus, there remains a dearth of empirical studies on the potential of learning Computer Programming through Robotics. The existing literature on the learning of text-based coding says very little about the use of Robotics, indicating a gap in the literature and the rationale for this study. This study aims to reduce the research gap between text-based coding and Robotics to achieve the objectives using the Arduino, PyFirmata and Python language. Despite many well-researched and grounded approaches that offer strategies to simplify and make the introduction to programming easy and simple to the novice, they have not proven to be the most effective and efficient. Therefore, introducing the student to basic constructs of programming using Robotics may influence their learning and understanding of coding, which is what this study sets out to explore.

The next chapter presents the theoretical framework that underpinned this study.

Chapter three: Theoretical framework

“There is an intimate and necessary relation between the process of actual experience and education.” ~ John Dewey

3.1 Introduction

This chapter engages with the relevant theories that apply to the study. Theoretical frameworks guide the research by relying on formal theories, thus playing a pivotal role in the generation of data, interpretation and findings (Osanloo & Grant, 2016). This study explored the use of Robotics in the learning of programming and was supported by two theoretical frameworks, Activity Theory and Kolb’s Experiential Learning. Engeström (2001) contends that there are three generations of Activity Theory: first-generation, second-generation, and third-generation Activity Theory, which has evolved over the years.

The practical nature of this study – involving Robotics and coding – needed ideally to be underpinned by a pragmatic paradigm. Pragmatic knowledge is rooted in the theoretical framework of Engeström’s second-generation Activity Theory; and integrated with Kolb’s Experiential Learning Cycle. This chapter begins with an account of how Deweyan pragmatism was adapted for the study by outlining the study’s broader philosophical approach, followed by an exploration of Activity Theory. For reasons of plausibility and coherence, some theories will be dealt with in the next chapter (*Chapter four: Research design and methodology*), where their relevance will be contextualised.

3.2 Approach to the study

The pivotal axis of this study is the pragmatic paradigm – an approach that is often associated with mixed-methods research (Feilzer, 2010). Pragmatism offers an alternative worldview to those of positivists, interpretivists and critics, while focusing on the problem to be researched and the consequences of that research (Feilzer, 2010; Kivunja & Kuyini, 2017). Hence, it is research that develops through its design. The history of pragmatism spans many decades and is known for being ill-defined and open to many interpretations (Rorty, 1991), as it can be used as a framework or a paradigm. The development of pragmatism stems from a social science research movement influenced by Charles Peirce from 1839-1914, William James from 1842-1910 and John Dewey from 1859-1952 (Revez & Borges, 2018).

There is a distinction between pragmatism used as a framework and as a paradigm, the latter having been advocated by the work of John Dewey (Lincoln et al., 2011). This study draws upon the interpretations of Deweyan pragmatism. As stated by Revez and Borges (2018), the “Deweyan pragmatism offers a sensible, practical explanation” (p. 6). Pragmatism is accepted as a paradigm, according to Feilzer (2010), and maintains that “there are singular and multiple realities that are open to empirical inquiry and orient itself toward solving practical problems in the real world” (pg. 8). Dewey (1925) likens the pragmatist view to existential reality, which is a reference to an experiential world with different layers, some objective, some subjective and some a mixture of both. John Dewey referred to human transactions as *experiences*, since the view of the world for the inquirer is a continuous transaction between researchers as actors within their environment (Dewey, 1925; Dewey, 1905). Thus, in this transaction (the study), the inquirer (the researcher) behaves in a way that adjusts the surroundings (learning of Computer Programming) and monitors the consequences of these actions. Dewey did not classify these as separate entities, but rather regarded this interactive relationship between the world and actors as one; and constituting reality.

Lincoln et al. (2011) point out that the major philosophical element that cuts across different paradigms is ontology, referring to the question of what constitutes reality. The ontology or nature of reality of the pragmatic view is determined by real-life social issues (Creswell et al., 2011; Feilzer, 2010). As in real life, reality is constantly changing and interpreted in its usefulness in new unpredictable situations. As aligned to this study, pragmatism addresses the problems associated with learning to code and how they can be resolved. As remarked by Creswell (2003), pragmatist researchers focus on the “what” and “how” of the research problem (p.11). Pragmatism is not devoted to one system of philosophy, unlike the objective and measurable reality of positivism; and the subjectivity of interpretivism and criticism, which adhere to a set of prescribing methods (Kivunja & Kuyini, 2017). Pragmatism favours research through a designed theoretical perspective, thus making it a real-world practice-oriented paradigm.

According to Dewey (1925), paradigms based on objectivism and subjectivism derive from the same homogeneity since they seek to find the truth, whether it is an objective truth or relative truth of multiple realities. The pragmatic approach relies on reasoning that combines both induction and deduction (Morgan, 2014) and emphasises inter-subjectivity (Merriam, 1998), which captures the relationship between the subjective and objective approaches of qualitative

and quantitative enquiry respectively. Pragmatism is an approach that uses the logical process of abduction, as well as deduction and induction (Revez & Borges, 2018). Deduction is achieved by reaching a conclusion from generally accepted data, while induction leads to generalisation based on the known data. On the other hand, abduction is formed by selecting the best explanation given the known data (Thomas & Georg, 1995). As pointed out by Yin (2014) and Merriam (1998), when concepts are abstract, it is important to utilise processes that help interpret, sort and manage information. This will lead to findings that convey clarity and relevance to the study.

Epistemological issues centre on the process of knowing and knowledge (Lincoln et al., 2011). The epistemological stance of pragmatism is that the nature of knowledge is determined by the method for solving a problem, thus allowing for change, which is an underlying principle. Hence, the researcher is allowed the freedom to choose the necessary methods to reach the objectives of the study (Revez & Borges, 2018). According to Dewey (1929), a “spectator theory of knowledge” (p. 163) that promotes an objectivist stance, ought to be rejected in pragmatism as the researcher or inquirer is an integral part of the inquiry experienced through their actions. Thus, a subjectivist approach favoured by interpretivists where the inquirer’s or researcher’s subjective knowledge plays an essential role in knowing should also be rejected. Pragmatism does not place emphasis on the researcher nor the world, since the researcher is part of a particular inquiry experience or regarded as a parameter in the interaction with the world (Dewey, 1929; Dewey, 1925). Furthermore, Dewey asserted that the outcomes of a pragmatist study are only warranted within the context or the unique set of parameters of the inquiry. Likewise, the outcomes in this study are warranted within the context of learning programming with Robotics within the parameters of this study.

Pragmatism has been advocated as the underlying philosophical stance for mixed-methods research (Morgan, 2014; Teddlie & Tashakkori, 2011). Such methods include interaction, experimenting, questionnaires, focus-group interviews, observations, open-ended questions, surveys, data mining, usability testing, physical prototyping, etc., thus favouring a mixed-method methodology that includes a collection of quantitative and qualitative data. According to Creswell and Clark (2011), pragmatism is oriented towards “*what works*” (p. 42) and it assumes there is no single scientific method that is able to discover the truth (Morgan, 2014). Furthermore, Howe (1988) argues that “no incompatibility between quantitative and qualitative methods exists at either the level of practice, or that of epistemology, and that there are thus no

good reasons for educational researchers to fear forging ahead with “*what works*” (p. 10). The use of a mixed-methods approach allowed for a complete understanding of the results during the analytical and interpretation phases of this study. This was achieved through triangulation, which assisted in validating the results obtained from different methods (further discussed in *Chapter four: Research design and methodology*).

It can be concluded that pragmatic research aims to act on a situation, rather than simply describing it. In addition, pragmatic research acts on the propositional knowledge that will be implemented in similar future situations. After all, the pragmatic paradigm is problem-centred and based on real-world practice. The argument is that pragmatism is essential to explain how design-driven research approaches can facilitate change and enhance creativity (Collatto et al., 2018). As explained by Feilzer (2010), the “pragmatic approach to problem-solving in the social world offers an alternative, and more reflexive guide to research design and grounded research” (p. 7). Dewey (1929) advocates that research be regarded as a type of experience in the world. This study is founded on the basis that the researcher is presented with an unresolved situation (way/method of learning Computer Programming) where wonted actions (block-based programming and use of computer visualisation tools) do provide an effective resolution to the situation. This means that the researcher needs to identify the problem and introduce an intentional change (use of Robotics) that will result in resolution. To conclude, research grounded in a pragmatic paradigm will have the following characteristics, set out by Kivunja and Kuyini (2017):

- “A rejection of the positivist notion that social science inquiry can uncover the ‘truth’ about the real world;
- An emphasis on ‘workability’ in research;
- The use of ‘what works’ to allow the researcher to address the questions being investigated, with minimum concern as to whether the questions are wholly quantitative or qualitative in nature;
- Adoption of a worldview that allows for a research design and methodology that are best suited to the purpose of the study;
- Utilising lines of action that are best suited to studying the phenomenon being investigated;
- A rejection of the need to locate your study either in a positivist (postpositivist) paradigm or an interpretivist (constructivist) paradigm;

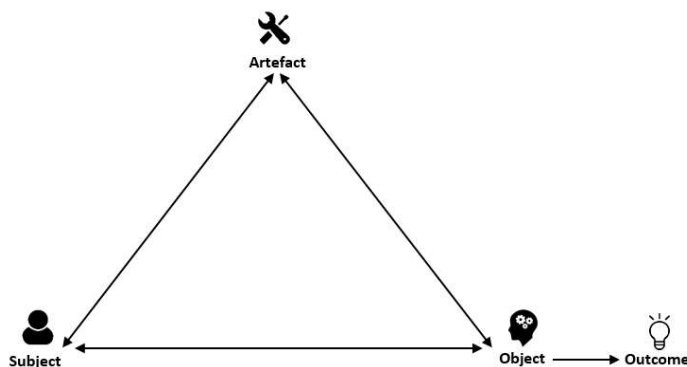
- Seeking to utilise the best approaches to gaining knowledge using any methodology that helps with knowledge discovery;
- Choice of research methods depending on the purpose of the research; and
- A search for useful points of connection within the research project that facilitate understanding of the situation” (p. 36).

3.3 Activity theory

The theoretical framework applied to this study was founded by Russian psychologist Vygotsky and his student Leontiev and was initiated in the 1920s and 1930s (Clemmensena et al., 2016; Engeström, 2001). It is recommended that Activity theory creates an ideal framework for HCI (Human Computer Interaction) research (Clemmensena et al., 2016; Jonassen & Ronrer-Murphy, 1999; Kuutti, 1996), thus, making Activity Theory relevant to the study that utilised Artificial Intelligence (Robotics) in learning how to code. Activity Theory builds on Vygotsky’s cultural-historical psychology, CHAT (Cultural Historical Activity Theory) and has evolved through three generations of research (Engeström, 2001). Activity Theory is not a methodology but a philosophical framework that aids in studying different forms of human praxis as developmental processes at an individual and social level (Jonassen & Rohrer-Murphy, 1999). Activity Theory is founded on the premise that human activity is purposefully carried out by a set of actions through the use of *artefacts/tools*, which can be physical or psychological (Engeström, 2001). This premise led to the development of first-generation Activity Theory, illustrated in Figure 30, with bi-directional arrows showing the flexibility of movement centered around the idea of mediation, where the artefact is a tool used to mediate learning.

Figure 30

An illustration of first-generation Activity Theory



Note. Adapted from Engeström (2001).

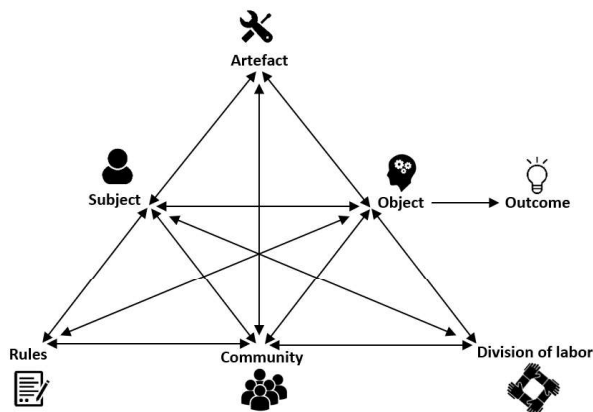
The *subject* is the individual within the learning environment working towards the *object*, the purpose of the actions. The *object*, in turn leads to the *outcome*, which is the result that the environment seeks to create. The *subject* generally is aware of the outcome but may not be consciously aware of their motives (Carvalho et al., 2015). Therefore, this can possibly lead to self-discovery learning, where learning and knowledge building takes place concomitantly. The *subject* refers to the individual who will acquire the relevant knowledge and the *object* relates to the motive, which is unplanned and through sense making results in the outcome (Engeström, 2001).

The actions of the *subject* are mediated by the *tools*. The *artefact* resembles any tool, physical or symbolic, such as machines, equipment, speaking, writing, gestures, music, etc., and that mediates the interaction between *subject* and *object*. Activity Theory focuses on how one utilises *tools* in a multidimensional social context to achieve specific *objects* that lead to the anticipated *outcomes*. As remarked by Carvalho et al. (2015), “the activity is directed at a motive” (p. 167); in other words, the motive refers to the *object* that the *subject* pursues.

Objects are the immediate results of actions, whereas *outcomes* are the products that emerge from the activity system. As pointed out by Carvalho et al. (2015), the activity is the basic unit of analysis for all human efforts. The activity is considered as a meaningful interaction between *subject*, *artefact* and *object*, as a process during which mutual change is accomplished. This is aligned to the principles of the pragmatic paradigm whereby some sort of invention introduces a change to a situation, which results in resolution. However, first-generation Activity Theory excluded the social context, which is the collaboration with others. As remarked by Engeström (2001), “the limitation of the first generation was that the unit of analysis remained individually focused” (p. 134). This led to three elements being added to the first model by Leontiev around 1978, which resulted in the bi-directional SGAT as illustrated in Figure 31 (next page).

Figure 31

An illustration of second-generation Activity Theory



Note. Adapted from Engeström (2001).

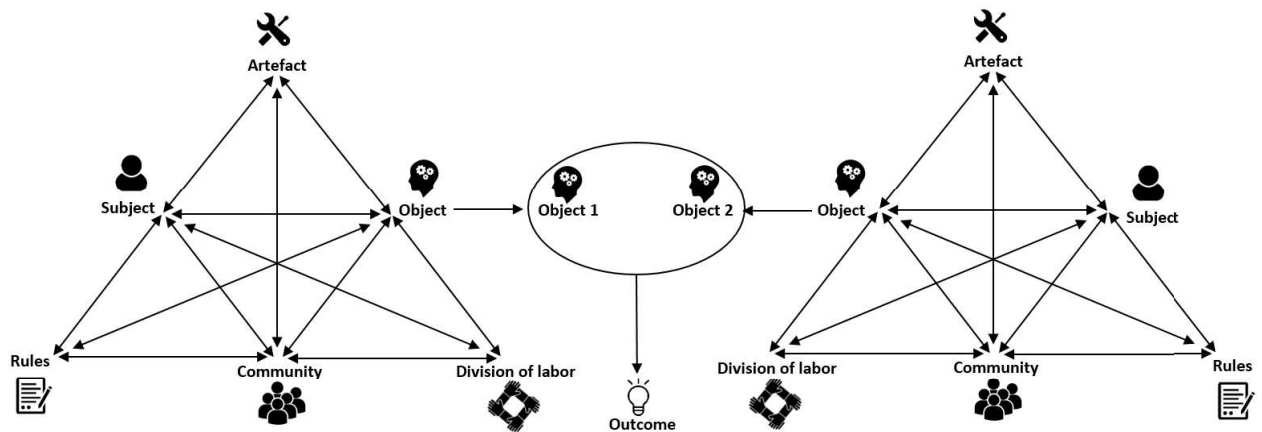
The three elements, also known as actors, that were added to the complexity of the first generation, were *rules* (referring to practices, learning, collaboration, etc.); a *community* (referring to the learning group, school, institution, profession, community, society, etc.); and *division of labor* (referring to the division of work between students, co-students, tutors, mentors, supervisors, managers, etc.). Thus, *rules*, *community* and *division of labor* have a critical influence on the actions that eventually lead to the *outcome*. It is noted that within the second generation, the first generation (represented by the upper triangle) focused on the individual, but is now supported by and grounded in *rules*, *community* and *division of labor*.

However, despite modifications to first-generation Activity Theory, which led to a more refined complex second generation, a third generation of the model was required to take account of two or more activities sharing a common outcome (as illustrated in Figure 32). When learning new concepts and material, outcomes can and should overlap – constantly building on prior knowledge (ZPD or Zone of Proximal Development) and reinforcing what has been learnt. As Hasan (1999) remarked, it is possible to realise the same activity by different sets of actions and operations, and the same actions can be part of different activities simultaneously aimed at similar or different outcomes. It is important to note that literature reveals that the third-generation model had sparked academic debate on Activity Theory and Actor-Network Theory due to the boundary crossing regarding the *outcomes* (Engeström, 2001).

It can be said that the Activity Theory can be related directly or indirectly to many philosophical concepts. As remarked, the assumptions of Activity Theory are accordant with constructivism and situated learning (Jonassen & Rohrer-Murphy, 1999).

Figure 32

An illustration of third-generation Activity Theory



Note. Adapted from Engeström (2001).

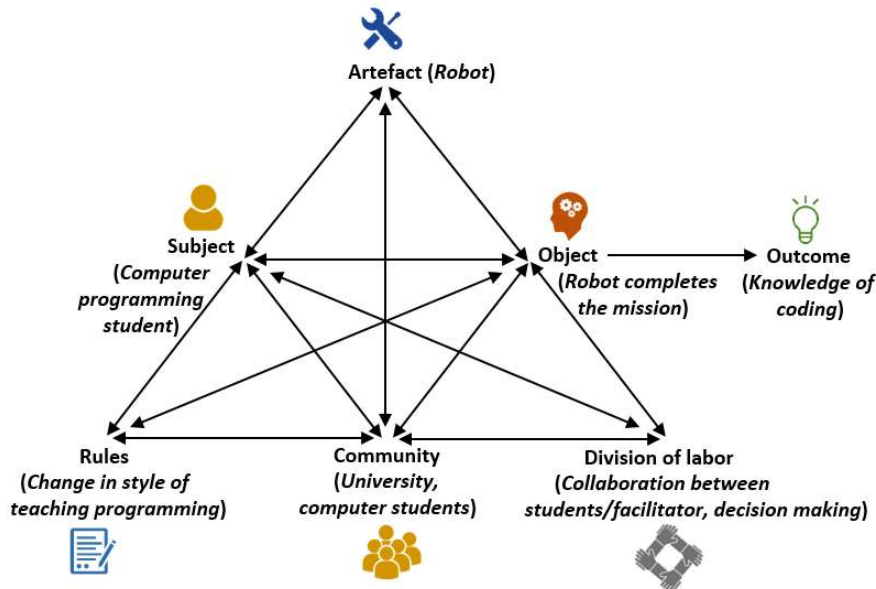
The bi-directional complexity of the third-generation model is clearly depicted in Figure 32. The objects from the two activities move to object 1 and object 2, a jointly shared space that leads to the common *outcome*. Engeström (2001) explained that “the object of activity is a moving target, not reducible to conscious short-term goals” (p.136). Thus, third-generation Activity Theory does not support a ‘shortcut’ or easy learning pathway to attaining the required knowledge.

Second-generation Activity Theory was deemed appropriate for understanding the use of Robotics in acquiring the knowledge for programming, since the study required a theoretical framework that accounted for human and mediating artefactual influences on achieving a common outcome, i.e., learning how to program in a system where the *tool*, *rules*, *community*, *division of labour* and *object* remain unaltered. The need for overlapping objects was not necessary. In addition, it is widely viewed that human knowledge is formed after the completion of an activity.

However, Jonassen and Rohrer-Murphy (1999) state that “Activity Theory posits that conscious learning emerges from activity (performance), not as a precursor to it” (p. 62). Thus, this theory provides a different view on human thinking and activity. Second-generation Activity Theory was implemented in this study, as illustrated in Figure 33.

Figure 33

Second-generation Activity Theory applied in study



Note. An illustration of the second-generation Activity Theory applied to this study.

The participants had to complete an activity with the outcome of attaining Computer Programming knowledge. The activity was divided into sub-activities. As confirmed by Carvalho et al. (2015), “the activity is not a static entity” (p.167), in the sense of Activity Theory. Kolb’s Experiential Learning Cycle was integrated as a philosophical enhancement to the research methodology, further discussed in the next chapter, sub chapter 4.7 *Fitting all together*.

3.4 Conclusion

This chapter discussed the theoretical underpinning of this study. Being practical in nature through the use of Robotics in the learning of programming, the study needed to be grounded in a suitable paradigm.

An account of why the study is deemed as a pragmatic study, grounded in Deweyan pragmatism is provided. In addition, it established the rationale for the use of Activity Theory as a framework. It was shown that the overlapping of objects was not necessary, so second-

generation Activity theory was deemed best suited. This chapter closes with an illustration of the second-generation Activity Theory that was implemented to provide a meaningful framework by which to describe the exploration of Robotics in the learning of programming.

Having established the paradigm and theoretical framework, the next chapter describes and discusses the methods used in this study. This includes Design Based Research (DBR) methodology and the integration of Kolb's Experiential Learning Cycle.

Chapter four: Research design and methodology

“Programs must be written for people to read, and only incidentally for machines to execute.” ~ Harold Abelson

4.1 Introduction

The previous chapter outlined the epistemology and ontology underpinning the study; as well as the theoretical frameworks that informed the study, arguing for the choice of second-generation Activity Theory. Although this is not the conventional format for capturing aspects of methodology, reasons for using this approach were given in Chapter 4 (*Theoretical framework*).

The current chapter provides a synthesis of the nature of research undertaken, the research tools used to gather information and how the collected data was analysed. It starts by presenting the research design and methodology starting with the design of the study, which is based on DBR (Design-Based Research), Kolb’s Experiential Learning Cycle, data collection tools and finally an overview of how everything integrates and aligns to the objectives set out.

4.2 Design Based Research – DBR

Aligned to the pragmatic paradigm, the Design Based Research (DBR) was selected for the research design. Design Based Research is closely aligned, and complementary to, the pragmatic paradigm (Alghamdi & Li, 2013; Cochrane et al., 2017; Owens et al., 2015). Design Based Research is founded on Design Science Research (DSR), but unlike DSR, it strikes a balance between theory and practice. The value of theory is appraised by the extent to which principles inform and improve practice. As remarked by Bakker and van Eerde (2015) “design-based research (DBR) can be characterised as research in which the design of educational materials (e.g., computer tools, learning activities or a professional development program) is a crucial part of the research ... the design of learning environments is interwoven with the testing or developing of theory” (p.430). Along similar lines Plomp (2013) states that DBR sets out to “design and develop an intervention (such as programs, teaching-learning strategies and materials, products and systems) as a solution to a complex educational problem; as well as to advance our knowledge about the characteristics of these interventions and the processes to design; and develop them – or alternatively to design and develop educational interventions” (p.15).

As highlighted earlier, Computer Programming is complex and abstract, thus qualifying for a DBR methodology as used in this study. As remarked by Plomp and Nieveen (2013) “design-based research also may contribute to the growth of human capacity for subsequent educational reform” (p.182).

DBR comprises three distinct phases: the preliminary research phase; the prototyping phase; and the assessment phase (Clark, 2015). It is worthy to note that most literature points to research carried out by Thomas Reeves, a scholar in learning, design, and technology who offers a modified version of DBR with four phases (Reeves, 2000; Reeves, 2006). An important consideration was which principle of DBR to base the study on: three phases or four phases? Goff and Getenet (2017) offer an extensive practical examination of DBR based on both principles. These authors found that three phases were suitable for research where the “researcher included the creation of particular teaching and learning materials and methods designed to realise participants’ learning gains” (p.112). Thus, the three-phase principle was selected for the current study, which sets out to accomplish the use of Robotics to harness students’ programming knowledge. Therefore, the study will adhere to the three distinct stages as illustrated in Figure 34 as derived from the work of Plomp (2009) and Nieveen (2009), and advocated by Palalas and Wark (2017); Clark (2015); and Abdallah (2013).

Figure 34

An illustration of the flow of DBR based on three phases



Note. Author created illustration.

The *preliminary research* phase requires a needs and context analysis for the study (Plomp, 2013). This includes a review of relevant literature that sets the platform for the study and other key theories that underpin the study, such as the development of the conceptual/theoretical framework.

The *prototyping* phase is also known as the iterative design phase. This phase consists of a series of iterations, each being a micro cycle of the research, followed by formative evaluation (Plomp, 2013).

These iterations can be accomplished with the use of the same tool, but for different group of participants; the same group of participants but a different/progressed tool; or a different group of participants with different/progressed tools (Clark, 2015).

The *assessment* phase involves an overall summative evaluation (Plomp, 2013), concluding if the solution or intervention meets the pre-determined specifications of solving the problem. This will entail evaluating the findings against the achievement of the objectives set for the study.

4.3 Integrating Kolb's Experiential Learning Cycle

As indicated in the previous chapter, Kolb's Experiential Learning Cycle is discussed in this chapter to create coherence. A short online course, hosted on a Learning Management System (LMS) platform and consisting of six online workshop sessions, formed part of the data collection process. Kolb's Experiential Learning Cycle (KELC) was infused into the DBR model. The second phase of DBR was the *prototyping* phase, an iterative process where each workshop session represented a micro cycle that followed the principles of KELC. KELC postulates that learning occurs in four stages, forming a cycle in the following order:

1. *Concrete experience*: Involves active engagement or experience;
2. *Reflective observation*: Reflecting on the activity or experience;
3. *Abstract conceptualisation*: Gaining knowledge from the experience; and
4. *Active experimentation*: Testing out the acquired skills or abilities (Kolb & Kolb, 2018; Kolb, 2015).

Each iterative process is a micro cycle in the form of a workshop session involving coding, designing and testing prototypes using the Arduino robot kit. Each of the six workshops consisted of three activities:

- **Activity 1:** A guided step-by-step activity that introduced and explored a programming concept;
- **Activity 2:** A semi-guided self-discovery programming mission that formed the foundation of the next activity; and
- **Activity 3:** A task based on Activity A and Activity B. This final activity was completed without any help.

Each of the six workshops was designed to continuously build on the previous, thus targeting the Zone of Proximal Development and building on existing knowledge.

It was important to facilitate the learning online using a set learning path created in LMS and with criteria in place to prevent the skipping of workshops or activities. The integration of KELC with the DBR prototyping phase is summed up in Table 3. This table shows how Activities 1, 2 and 3 are linked to KELC. Activity 1, provided the participant with an encounter with a concrete experience. While reflective observation and abstract conceptualisation are encountered in Activity 2. Lastly, Activity 3 allowed for active experimentation.

Table 3

Kolb’s Experiential Learning Cycle vs the DBR prototyping phase

Kolb’s Experiential Learning Cycle	Prototyping – micro cycle (workshop)
Concrete experience	Activity 1
Reflective observation	Activity 2
Abstract conceptualisation	
Active experimentation	Activity 3

Note. Kolb’s Experiential Learning Cycle imposed on the three activities per workshop.

4.4 Research questions

There is no strategy to the introduction of programming guaranteed to lead to the attainment of programming knowledge. Therefore, the following questions arise: What are the best practices for learning programming? In light of the fact that students at tertiary level perceive programming as difficult, the researcher asks: How can the learning of programming be promoted?

To recap, the study sets out the following objectives:

1. *An alternative educational tool:* To explore the use of Robotics to enhance the learning of Computer Programming.

The Arduino robot kit offered a hands-on approach to programming (physical computing) since the student designs, programs and executes the code in reality on the prototype rather than in a 2D representation on a screen.

2. *Problem solving and reasoning*: To explore external factor/s that contribute towards the learning of Computer Programming.

It is well documented in the literature (*Chapter 2*) that internal issues around the learning and understanding of programming exist, such as programming paradigm preference, the programming language used and programming style. However, there are contradictory accounts of external factors such as students' Mathematics background, development of Computational Thinking, higher-order thinking skills, etc., that may impact student proficiency in Computer Programming.

3. *Easy to understand*: To assess the effectiveness of Robotics in the understanding of Computer Programming.

This study used the programming of Robotics to introduce the basics of programming; using a text-based environment rather than a block-based environment. This approach was used to provide students with a direct learning experience of text-based code, and to simplify the learning of programming.

In line with the study's purpose and objectives, the following questions were formulated:

1. What are students' perceptions of Robotics when learning to program?
2. Does a high rational ability contribute to attaining programming knowledge?
3. How does the use of Robotics contribute to the understanding of programming?

4.5 Qualitative and quantitative data

As mentioned earlier, the pragmatic paradigm underpins DBR. Thus, according to pragmatism, a researcher should use whichever methods work best for data collection. This study was carried out using mixed-methods data collection, which combines qualitative and quantitative methods.

There are three types of mixed-methods approaches: firstly, where qualitative and quantitative data are equally important; secondly, where qualitative is dominant over quantitative; and thirdly, where quantitative is dominant over qualitative (Check & Shutt, 2012). The third type

was used in this study as the large number of participants allowed for the use of inferential statistics.

Nonetheless, qualitative data was used to complement the quantitative data in order to provide a deeper understanding and explain the results in relation to the objectives. Furthermore, the collection of both data types allowed for the construction of detailed descriptions and complete explanations.

4.6 Data collection techniques

In order to answer the research questions and achieve the study's objectives, data was gathered using the following data collection instruments⁸: surveys, questionnaires and an interview.

4.6.1 Pre-workshop session

4.6.1.1 Pre-survey

A 5-point, 10-item Likert scale survey entitled *Computer Programming survey one* (Appendix F) consisted of items that served as a pre-survey to capture the participant's self-efficacy, perceptions and attitude towards programming and Robotics. This data collection instrument was administered first to prevent any prior influence from other instruments that might have affected an individual response to programming.

4.6.1.2 Problem solving and logic

Questionnaire test one consisted of close-ended responses. This instrument was split into two parts, Part A (Appendix G) and Part B (Appendix H).

Part A: Pre-test based on Computational Thinking

This part of the questionnaire (Appendix G) captured bibliographic information about participants, followed by multiple-choice questions (MCQ) consisting of 10 items. These questions were based on reasoning and logic, which were set to determine the individual's problem-solving ability. These MCQs were aligned to concepts the participant would be likely to encounter during the workshop sessions.

⁸ Given the COVID-19 pandemic all data was collected online by means of online forms via the Moodle and Google Forms.

Part B: Abstract Reasoning Test

An Abstract Reasoning Test (ART) with 25 items (Appendix H) made up Part B. This test took the form of a psychometric test, which dates back to psychologists Charles Spearman's work around the 1920s (Chen et al., 2019).

Spearman indicated that intelligence is made up mainly of a *g* (general ability) factor, which is determined by mechanical, spatial, numerical and verbal factors referred to as *s* (specific ability) factors.

Abstract Reasoning Tests (ARTs) use symbols and shapes instead of words and numbers. All participants were given 25 minutes to answer the test, ensuring all participants had equal time for completion.

Holistically, *Questionnaire test one*, responds directly to research question two. This was achieved when coupled with data captured from the biographic section, such as Mathematics background, multiple-choice responses and the ART score.

4.6.2 Workshop sessions

There were six online workshop sessions (Appendix J, L, N, P, R and T), each comfortably completed within an hour, that covered programming aspects through robot (educational tool) coding using the Arduino kit. In addition to the six workshop sessions, an introductory workshop (Appendix I) served the purpose of familiarising participants with the components and software environment. Each participant received their own personal Arduino kit. The workshops were hosted online allowing for queries and communication among peers to be posted on a forum, in accordance with the pragmatic paradigm's social aspect. Forums were setup for each workshop session. Although a workshop session could be completed within an hour, each workshop ran for a week giving participants the flexibility of completing the activities at any time during that week. The full solutions for Activity 3 (no help given) in each workshop were made available at the end of the week via the discussion forum. Therefore, allowance was made for tally of participants who had queries or encountered problems during the week. Most importantly, it gave participants a chance to access feedback from the forums and to fix their prototypes or code in order to reach a successful solution.

These six workshops served as micro-cycles in the iteration process, which is in accordance with the prototyping phase of DBR. Workshop sessions 1, 2 and 3 introduced the basics of

programming: syntax, input, output, if statements, for loops, and while loops. Workshop sessions 4, 5 and 6 provided consolidation and further exploration of the robotic components.

At the end of each micro cycle, participants were prompted to complete a self-formative evaluation in the form of a 5-point Likert scale: 1 - *Very difficult*, 2 - *Difficult*, 3 - *Neutral*, 4 - *Easy* and 5 - *Very easy*.

Each online workshop session started with a guided step-by-step activity taking participants through an activity: Activity 1, followed by a semi guided self-explorative activity; Activity 2 and a no-help activity; and Activity 3. All activities involved hands-on interaction with the Python programming language in a text-based environment. The core purpose of the micro cycles was to expose participants to the intervention, and to help formulate meaningful data used to respond to the research questions.

4.6.3 Post-workshop session:

4.6.3.1 Post-survey

A 5-point 30-item Likert scale survey called *Computer Programming survey two* (Appendix V) was administered at the end of the six workshop sessions and served as a post-survey to capture the immediate afterthoughts of Computer Programming using the Arduino robot. To collate and give structure to the survey, items were formed based on the six constructs. Each construct comprised 5 items. These constructs or latent variables (LVs) were found via literature (*Chapter 2: Literature review*) to be contributing factors to an individual's coding knowledge.

Selection of constructs/latent variables

Regression analysis carried out in a study by Tsai et al. (2019) revealed that *Mathematics* was a positively correlated predictor of programming knowledge. The correlation was found in pre-test and post-test. Additionally, an individual's *motivation* to learn Computer Programming is a crucial factor in their acquisition of programming knowledge. As Carbone et al. (2009) elaborate, those students who exhibited motivation “usually undertook to learn programming in their own time, sometimes prior to the course commencing, working hard at developing their skills” (p. 4). As pointed out earlier (*Chapter 2: Literature review*), Computer Programming can be perceived as difficult and intimidating, especially text-based language-specific

programming. A study carried out by Blanchard et al. (2019) found that “hybrid programming⁹ environments can help to transition students from blocks to text-based programming while minimising negative perceptions of programming” (p. 25).

Thus, a student’s *belief* about coding is an important factor that contributes to their programming knowledge. This study, being practical and hands-on in nature, had the potential to build and maintain students’ *interest* in Computer Programming.

As remarked by Biggers et al. (2008), “highly interactive hands-on introductory CS courses ... provide a broader overview of potential CS” (p. 406). An individual’s *interest* in coding can thus affect their knowledge of programming. Students enrolled in a programming course are prone to show anxiety, such as, being less confident – especially if they do not have prior computer experience (Byrne & Lyons, 2001). Similarly, from a comparative study between coding IDEs, Daly (2011) commented that “learning abstract programming concepts and programming in an environment ... can cause students to become frustrated, lose confidence” (p.23). Therefore, *confidence* in learning coding has a potential effect on the development of programming knowledge.

Based on the corpus of the literature reviewed, the survey design took into account the following six constructs that have been shown to influence knowledge of coding: student *confidence* in their ability to learn programming; student *interest* in programming; student *motivation* to use Robotics; student intrinsic *belief* that they can solve problems; student perception of mathematical influence on programming; and student *knowledge* of programming through the use of Robotics (Appendix W).

4.6.3.2 Post-test based on programming

A closed-ended questionnaire called *Questionnaire two* (Appendix X) was based directly on the programming language whereby concepts covered were used to gauge participants’ understanding. This instrument was made up of ten multiple-choice questions (MCQs).

4.6.3.3 Interview

Focus-group interviews are planned discussions where the researcher uses a set sequence of questions to probe for participants’ insights on a particular topic (Krueger & Casey, 2015).

⁹ A mix of text-base and block-base environments

Eight leading questions made up the interview, which lasted approximately 45 minutes. The focus-group interview was held on Zoom¹⁰, with participants who volunteered to be part of the discussion.

The interview (Appendix Y) was conducted after the six workshops, as this prompted in-depth explanations that provided more information on participants' experiences of learning code through the use of the Arduino robot. To prevent the risk of some participants dominating the discussion, the chair (researcher) encouraged every participant to share their views, thus contributing equally to the discussion.

4.7 Fitting it all together

It was important that the appropriate data collection instruments were used to respond to each research question. Table 4 below describes the data collection instrument/s and how they were used to answer the respective research question/s.

Table 4

Data instruments versus research questions

	Data instrument	Research question/s
i.	The pre-survey: A 5-point Likert scale questionnaire that served as a pre-survey at the start of the study to capture the participant's self-efficacy perceptions and attitude towards programming.	RQ1 RQ3
ii.	Questionnaire test one: This problem solving and logic questionnaire had two parts: Part A (pre-test based on Computational Thinking) a 10-item multiple-choice pre-test; and Part B (Abstract Reasoning Test) a psychometric test. Both instruments were based on principles that set to determine the individual's problem-solving skills and logical reasoning.	RQ2
iii.	Micro cycles: Six workshop sessions (excludes pre-survey), each running for about an hour covering aspects of programming using the Arduino robot kit. Participants worked online and posted queries in accordance with the social aspect of the pragmatic paradigm. The six workshops also served as the iteration process in accordance with the prototyping phase of DBR. After each workshop session, a self-evaluation in the form of a Likert scale was completed by each participant.	RQ3
iv.	Questionnaire test two: A post-test based directly on the programming language used and concepts covered to gauge participants' understanding.	RQ2

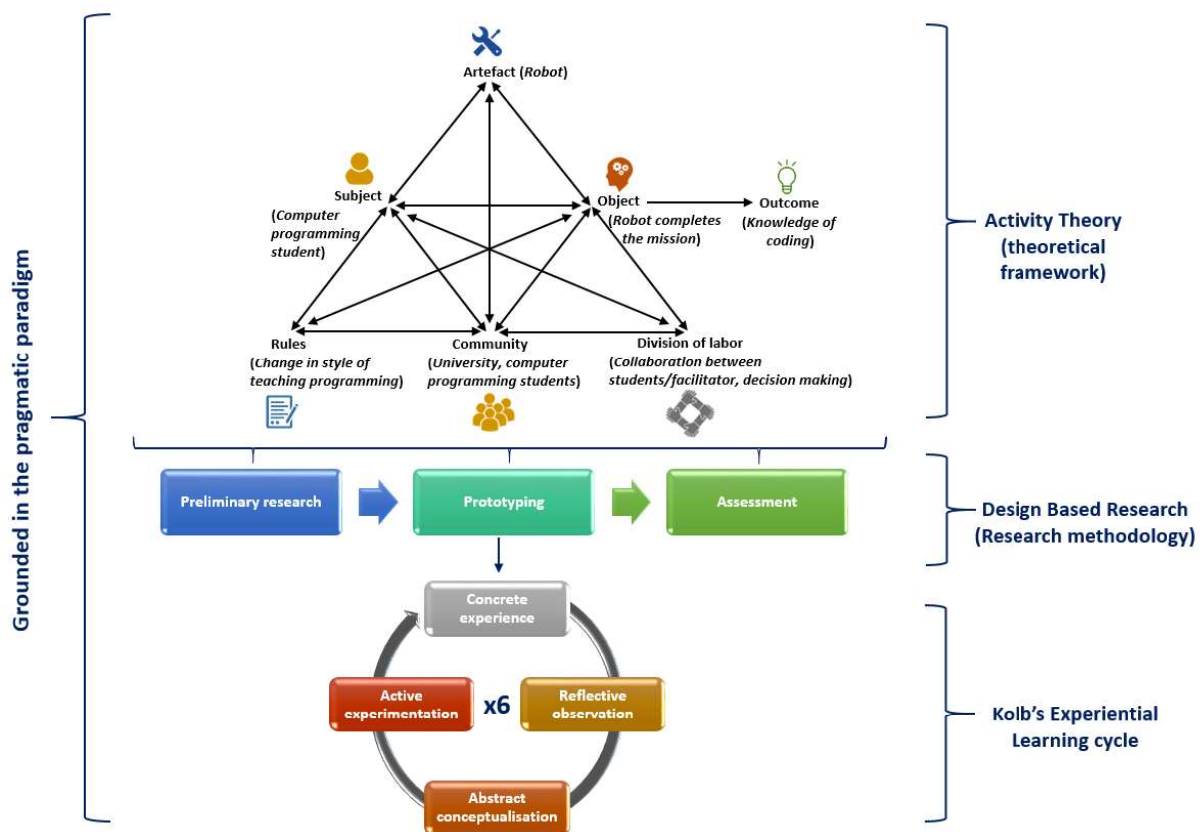
¹⁰ Zoom is an online platform that allows video and audio communication.

v.	The post-survey: A 5-point Likert scale questionnaire at the end of the study served as a post-survey to capture participants' afterthoughts about programming using the robot.	RQ1 RQ3
vi.	The focus-group interview: A focus-group interview in the form of a discussion.	RQ1, RQ2 and RQ3

Figure 35 provides a visual explanation of the philosophical principles that underpin this study. Second-generation Activity Theory conforms to Design Based Research methodology. In addition, Kolb's Experiential Learning Cycle was superimposed onto the prototyping phase of DBR. At the same time, the entire philosophical system in this study was viewed through the lens of the pragmatic paradigm.

Figure 35

Linking the philosophical principles underpinning this study



Note. The illustration depicts how the second-generation Activity Theory, DBR and Kolb's Experiential Learning Cycle were integrated for this study.

As mentioned earlier, the study used the Arduino UNO R3 kit, and each participant was provided with a kit. The kit contained several different types of components in addition to the UNO R3 microcontroller board, such as: a breadboard, jumper wires, resistors including a Light Dependant Resistor (LDR), Light Emitting Diodes (LEDs), buttons, a buzzer, actuators, and sensors (like flame sensors, tilt sensors, etc.). The IDE chosen was Python IDLE because of its design and simplicity of use. Many different programming languages are used to teach programming. Some of the most popular include Java, C, Python, and C++ (Hendrix & Weeks, 2018; Sebesta, 2016). However, Mannila and Raadt (2006) provided good evidence in their study that Python is the preferred programming language for learning. In addition, Shein (2016) states that “once someone grasps the logic behind Python, the concepts can be more easily transferred to other languages” (p. 19). As mentioned earlier, the Python programming language was used in the study.

4.8 Study setting

4.8.1 Location and population

The site for this study was one university campus in KwaZulu-Natal, South Africa. The study adopted a combination of non-probability sampling techniques, namely quota and purposeful sampling. At this campus, students are registered mainly for a four-year Bachelor of Education degree (B.Ed.). Students registered in any computer-related course were included in the study population. Quota sampling was used on the study population. Quota sampling establishes certain pre-requisites known as strata, which are considered important in selecting the sample (Descombe, 2014).

An invite was sent to all members of the study population via email and SMS (School Messaging System). SMS is a communication service hosted by the University that filters the intended receivers, hence possible participants. Participants needed to meet the following quota to be selected:

1. Registered in a computer course/module; and
2. No exposure to Computer Programming during the duration of the degree.

It was anticipated that the sample will most likely be made up of first-year students who had just started a computer course and no prior exposure to Computer Programming at the university. It is mandatory for students registered in the B.Ed. programme to complete a

computer endorsement course to ascertain who can specialise in Computer Science (focus on programming) or Information Systems (focus on user applications). Hence, the second quota or criterion was crucial in selecting the sample, which was not limited to B.Ed. students. The selection of participants with prior university Computer Programming comprised a second sample that allowed for a comparison with the first sample, i.e., those participants with no Computer Programming at all. To determine the appropriate sample size from the population, the ten-times rule was followed. The ten-times rule was especially important for the post-survey (Computer Programming survey two) that contained six constructs, each with 5 items forming 5 LVs. Therefore, in order to develop a significant PLS SEM, the minimum sample required = $5 * 10 = 50$.

4.8.2 The pilot study

A pilot study allows for the testing and refining of the data collection instruments before the actual data is collected during the main study (Cohen et al., 2002; Descombe, 2014). The pilot study allows for the checking of data instruments to ensure they are reasonable and have no defects or ambiguity. As a result of the pilot study, a few typographical errors were corrected and, most importantly, two issues related to time management were addressed. The necessary changes and modifications were made to ensure that no problems were encountered during the main data collection event. The process of conducting the pilot study ensures the reliability, validity, and practicability of using the data collection tools (Cohen et al., 2002). In regards to time management, two issues were encountered during the pilot study. Firstly, building and then coding the prototype required extra time. Thus, participants were given a week to complete the three activities that made up a workshop session in the main study. This provided sufficient time and offered flexibility to the participant to complete the workshop at any time during the week, given that it was done online. Secondly, the surveys, questionnaires and setup needed a separate time allocation for completion. This would allow participants to respond to the surveys and questionnaires without interfering with the workshop session – wasting no time in starting the activities, and not tiring participants. To overcome this problem, a pre-workshop session was added, which was dedicated to the pre-survey, Questionnaire A and the setup before workshop session one (first micro cycle) in the following week. (As mentioned earlier, each workshop session spanned a week). Similarly, Questionnaire B and the post-survey were completed in a separate session, reducing participant anxiety.

4.9 Validity, reliability and rigour

DBR is regarded as practical research. Thus, to ensure that the study meets the requirements for research rigour, *objectivity*, *validity* and *reliability* (Alghamdi & Li, 2013) form an integral part of the study. The issue of objectivity can be contentious in a DBR study where the researcher plays multiple roles, e.g., that of a developer, designer, facilitator and evaluator (Clark, 2015). However, this issue can be mitigated by the use of a mixed-methods, thus promoting triangulation (Goff & Getenet, 2017). Mixed-methods research helps to reduce bias in the research procedures and interpretation of results. Dikko (2016) posit that “one way to ensure that validity is achieved in any research is to conduct a pilot study of research instruments” (p. 521). Lincoln and Guba (1985) note that no validity exists without reliability; thus, ensuring validity also ensures reliability. Hence, reliability and validity are intertwined concepts that are juxtaposed. The validity of research as stated by Bertram and Christiansen (2014), is “the extent to which we trust the research” (p.42).

External validity is the extent to which the findings can be applied to the wider population, i.e., the extent to which the findings can be generalised (Bakker & van Eerde, 2015); while internal validity is the extent to which the conclusion correctly portrays the data collected (Ford et al., 2017). Alghamdi and Li (2013) posit that DBR cannot be generalised from a sample to a larger population because it is regarded as contextualised research, which relies on thick description for analysis. However, the mixed-methods approach used allowed for triangulation of data, thus increasing the external validity, and hence the generalisability, of the findings. Reliability of a study refers to the extent to which the same results would be produced by multiple occurrences of the same study (Bertram & Christiansen, 2014). In this study, the prototyping phase and the iterative cycles of DBR promoted reliability, adding to the robustness of the research findings.

4.10 Permission and ethical considerations

A letter requesting permission to use the study population was sent to the registrar of the university (Appendix A). All participants who accepted the invitation signed a consent form (Appendix B). The consent form outlined the details of the study with the option of withdrawing at any stage of the research. Prior to any letters being sent out, full ethical clearance was obtained to conduct this research (Appendix C). The duration of the data collection was eight weeks, with one session per week (pre-workshop session, six micro cycles

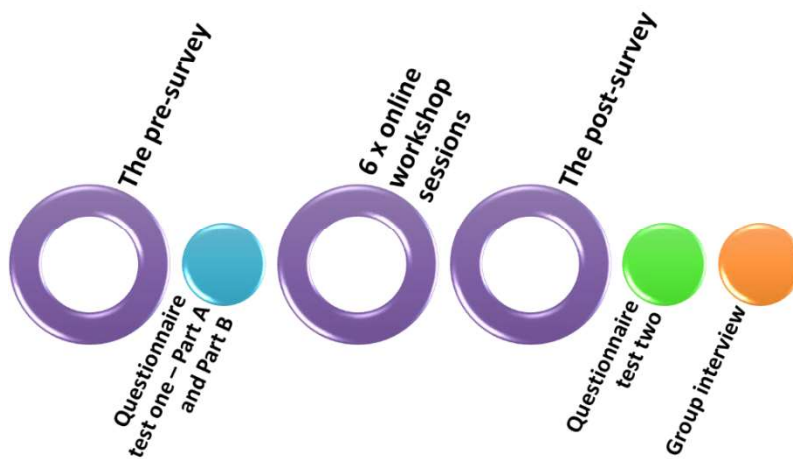
and post-workshop session). The use of pseudonyms guaranteed the anonymity and confidentiality of participants. The participants in the study were given pseudonyms starting with P1, P2, P3¹¹, etc.

4.11 Conclusion

The study is practical in nature, whereby it proposes a solution to the learning of code, followed by the testing of this solution. Since this is a developing solution to the learning of programming (an intervention), the study follows the methods advocated by Design Based Research (DBR). The DBR methodology used is described in detail in the next chapter, which also details how the Arduino robot kit and code were set up and used by the participants to enhance their learning of Computer Programming. Figure 36 illustrates the arrangement of the data collection process.

Figure 36

Data collection process



Note. Author illustration of research process.

To summarise, this study used surveys, questionnaires, workshops and an interview to collect data, thus ensuring triangulation. Being a pragmatic study, a DBR design was used employing mixed-methods data collection. Questionnaire surveys were used to produce quantitative data

¹¹ The P stands for programmer

that could be evaluative, followed by group discussion interviews that produced qualitative descriptive data to solidify the findings. The piloting of the data collection tools, assisted in identifying and removing errors, such as, changing to the appropriate layout and determining a sufficient period for completion of each workshop. The use of quantitative and qualitative data collection techniques enhances triangulation by affirming or negating results. This, in turn strengthens the body of knowledge arising from this study. These multiple methods of data collected and triangulation serve to enhance the validity and reliability of the study's findings.

PROTOTYPING

Chapter five: Iteration

“The role of the teacher is to create the conditions for invention rather than provide ready-made knowledge.” ~ Seymour Papert

5.1 Introduction

The previous chapter presented the research methods employed in this study to attain the research objectives and address the research questions. The data collection instruments used were described, namely, 5-point Likert scale surveys, questionnaires and six online workshops comprised of three activities each (Activity 1, 2 and 3). This chapter provides an overview and analysis of each workshop, which represents an iteration according to Kolb's Experiential Learning Cycle, and which is merged into DBR's prototyping phase. The activities in each workshop were designed around Kolb's Experiential Learning Cycle. Activity 1 was guided, providing a *concrete experience*. Activity 2 was semi-guided/self-discovery, allowing for *reflective observation* and *abstract conceptualisation*. Lastly, Activity 3 was unguided, providing *active experimentation*. Each online workshop and its three activities represented a micro cycle that plays a crucial role in developing and nurturing an individual's understanding of programming through the use of robotic elements (electronic components). In total, there were seven workshops, six micro cycles and one introduction. After each cycle, participants were made to rate their accomplishments based on the three activities. The rating system was in the form of a Likert scale with 1 - *Very difficult*, 2 - *Difficult*, 3 - *Neutral*, 4 - *Easy*, and 5 - *Very easy*. Each workshop session was designed to gradually increase cognitive demand as one progressed through the three activities.

5.2 Pre-workshop session (excluded from the six micro cycles)

This session (Appendix I) was excluded from the iteration as its purpose was to familiarise participants with the IDLE (Integrated Development and Learning Environment), Python language, and the setup. In addition, the background to Computer Programming and an introduction to coding was covered. Most importantly, the session familiarised participants with the Arduino components that were given to them.

5.3 Micro cycle: Workshop session one

In this workshop (Appendix J), participants were acquainted with the components of the Arduino kit. This included the exploration of the microcontroller, breadboard and jumper cables. Participants started coding their first program called *Hello world* while familiarising themselves with the Python language. The highlight of this introduction workshop was the use of the *for-loop* structure in controlling a LED switching ON following the display of *Hello world* on the PC screen.

Table 5

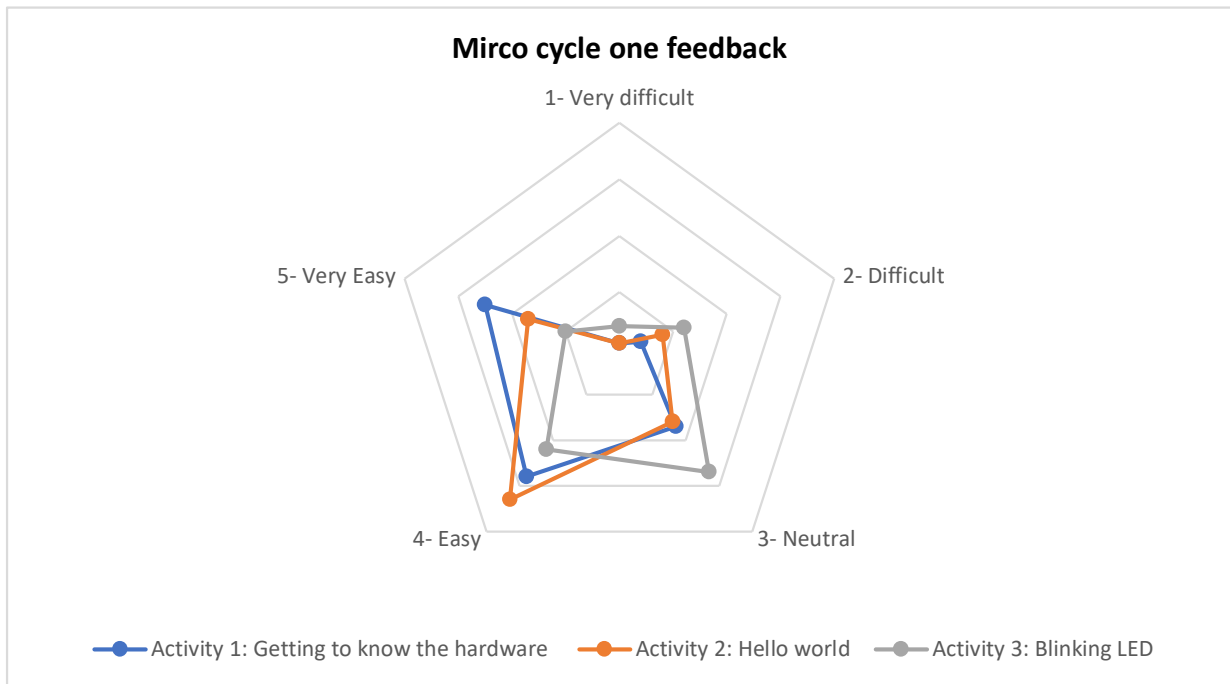
Overview of micro cycle one feedback

	Activity 1: Getting to know the hardware	Activity 2: Hello world	Activity 3: Blinking LED
1 - Very difficult	1%	1%	5%
2 - Difficult	5%	11%	16%
3 - Neutral	23%	21%	36%
4 - Easy	37%	44%	29%
5 - Very easy	33%	23%	13%
	100%	100%	100%

Note. Feedback based on self-evaluation for workshop session one (Appendix K).

Figure 37

Visual overview of micro cycle one feedback



Note. A graphical representation of the responses to the three activities in micro cycle one.

The majority of participants found Activity 2 to be *Easy* (44%), followed by Activity 1 being *Easy* (37%) and *Neutral* for Activity 3 (36%).

5.4 Micro cycle: Workshop session two

On the basis that participants were now familiar with the Arduino UNO R3, IDLE and basic Python code setup, this session (Appendix L) introduced user input and conditional statements (*if statement*). The highlight of this session was coding a traffic robot system prototype using red, yellow and green LEDs.

Table 6

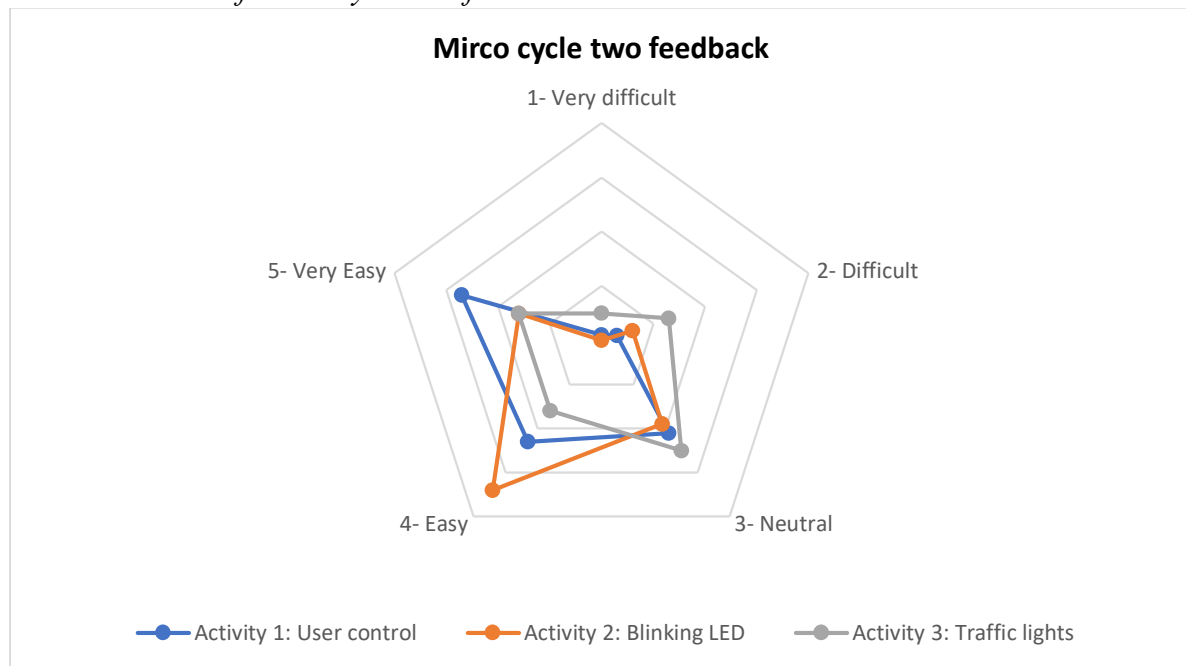
Overview of micro cycle two feedback

	Activity 1: User control	Activity 2: Blinking LED	Activity 3: Traffic lights
1- Very difficult	1%	0%	7%
2- Difficult	4%	8%	17%
3- Neutral	28%	25%	33%
4- Easy	31%	45%	21%
5- Very easy	36%	21%	21%
	100%	100%	100%

Note. Feedback based on self-evaluation for workshop session two (Appendix M).

Figure 38

Visual overview of micro cycle two feedback



Note. A graphical representation of the responses to the three activities in micro cycle two.

The majority of participants found Activity 2 to be *Easy* (45%), followed by Activity 1 being *Very easy* (36%) and, lastly, Activity 3 being *Neutral* (33%).

5.5 Micro cycle: Workshop session three

This session (Appendix N) furthered the use of the looping structures by emphasising and introducing the *while-loop* structure. Participants were made to code nested structures. The

highlight in this session was creating a night sensor prototype using an LDR (Light Dependent Resistor).

Table 7

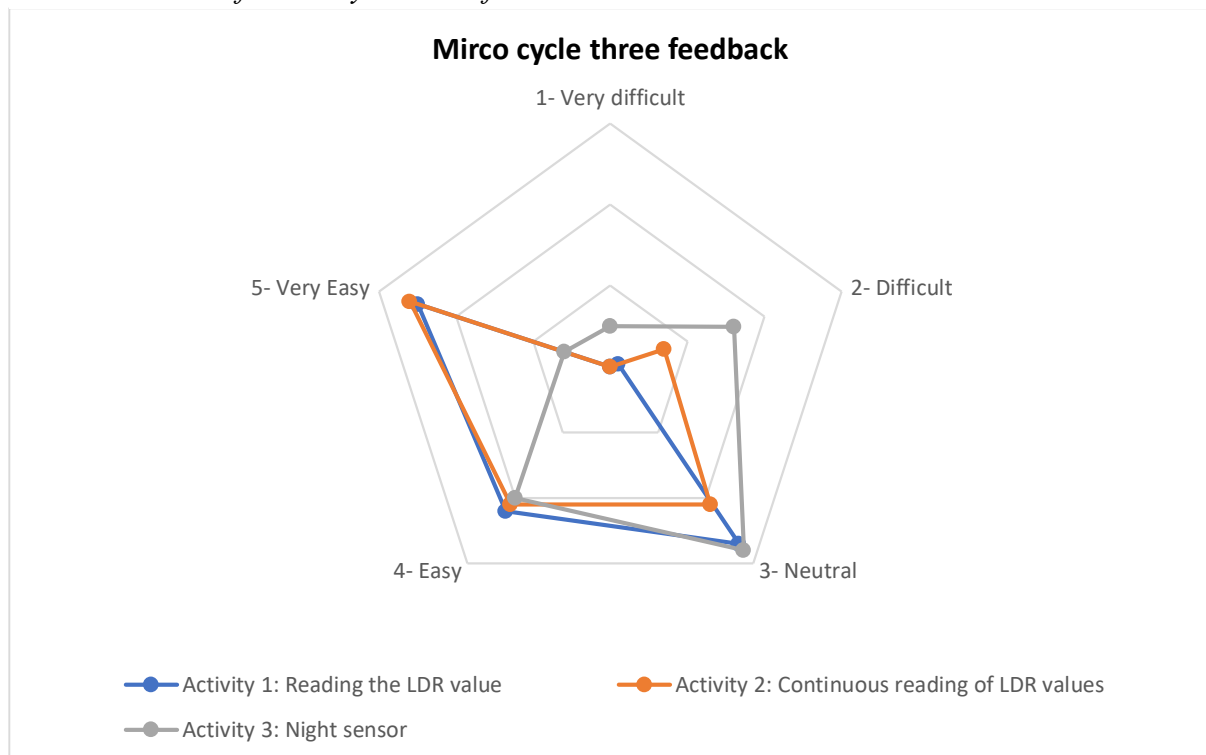
Overview of micro cycle three feedback

	Activity 1: Reading the LDR value	Activity 2: Continuous reading of LDR values	Activity 3: Night sensor
1- Very difficult	0%	0%	7%
2- Difficult	1%	9%	21%
3- Neutral	36%	28%	37%
4- Easy	29%	28%	27%
5- Very easy	33%	35%	8%
	100%	100%	100%

Note. Feedback based on self-evaluation for workshop session three (Appendix O).

Figure 39

Visual overview of micro cycle three feedback



Note. A graphical representation of the responses to the three activities in micro cycle three.

The majority of participants found Activity 3 to be *Neutral* (37%), followed by Activity 1 being *Neutral* (36%) and, lastly, Activity 2 being *Very easy* (35%).

5.6 Micro cycle: Workshop session four

Session four (Appendix P) reinforced the need for looping structures while using different Arduino parts. The highlight in this session was creating a fire alarm by building a prototype using a flame sensor and a buzzer.

Table 8

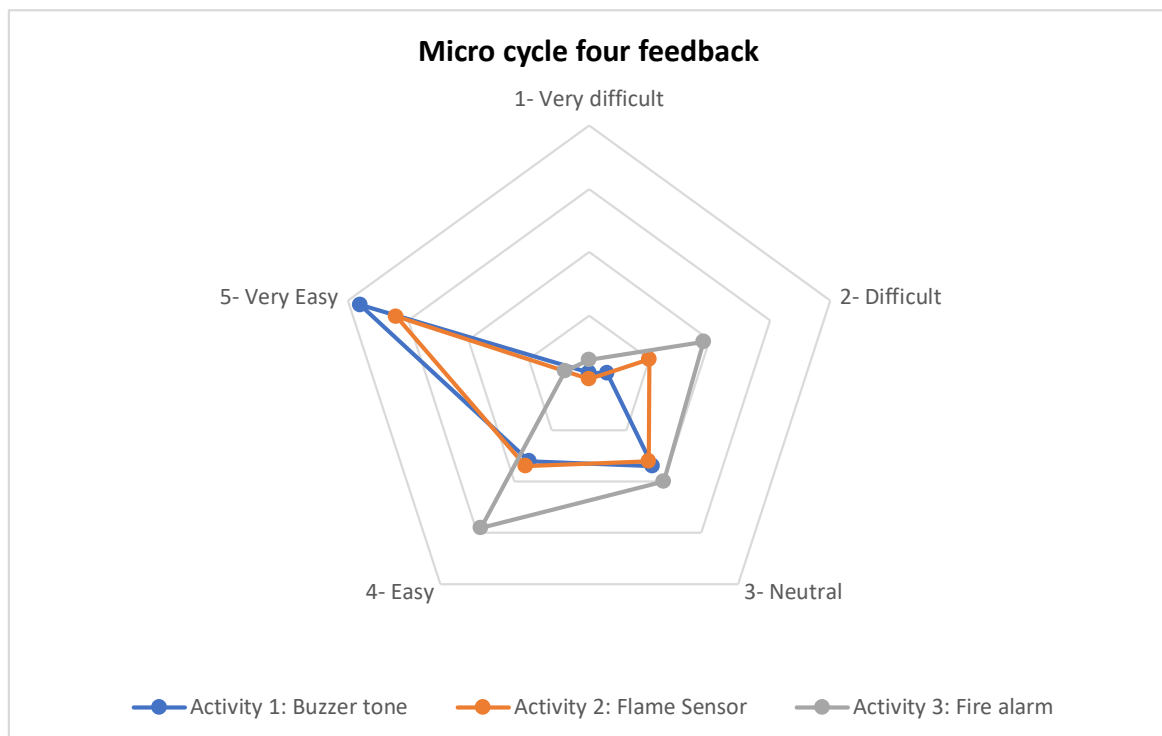
Overview of micro cycle four feedback

	Activity 1: Buzzer tone	Activity 2: Flame Sensor	Activity 3: Fire alarm
1- Very difficult	1%	0%	4%
2- Difficult	4%	13%	25%
3- Neutral	23%	21%	27%
4- Easy	21%	23%	39%
5- Very easy	51%	43%	5%
	100%	100%	100%

Note. Feedback based on self-evaluation for workshop session four (Appendix Q).

Figure 40

Visual overview of micro cycle four feedback



Note. A graphical representation of the responses to the three activities in micro cycle four.

The majority of participants found Activity 1 to be *Very easy* (51%), followed by Activity 2 being *Very easy* (43%) and, lastly, Activity 3 being *Easy* (39%).

5.7 Micro cycle: Workshop session five

This workshop session (Appendix R) used the buzzer introduced from the previous session, but with the tilt sensor. The highlight of this consolidation session was creating an earthquake detector.

Table 9

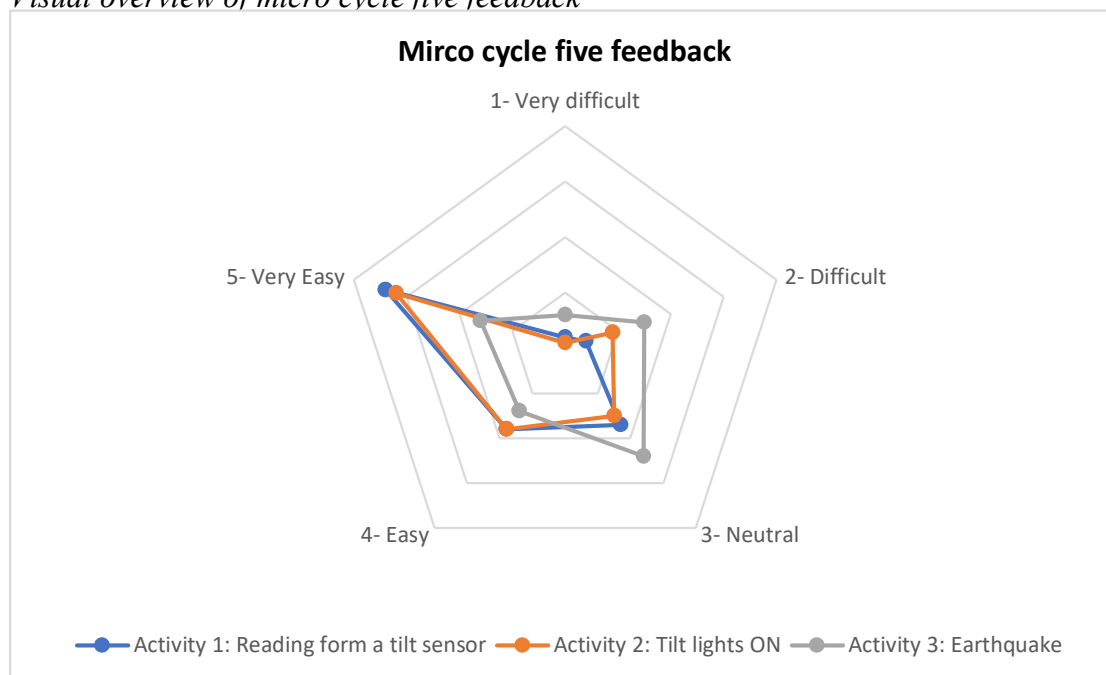
Overview of micro cycle five feedback

	Activity 1: Reading from a tilt sensor	Activity 2: Tilt lights ON	Activity 3: Earthquake
1- Very difficult	3%	1%	8%
2- Difficult	5%	12%	20%
3- Neutral	23%	20%	32%
4- Easy	24%	24%	19%
5- Very easy	45%	43%	21%
	100%	100%	100%

Note. Feedback based on self-evaluation for workshop session five (Appendix S).

Figure 41

Visual overview of micro cycle five feedback



Note. A graphical representation of the responses to the three activities in micro cycle five.

The majority of participants found Activity 1 to be *Very easy* (45%), followed by Activity 2 being *Very easy* (43%) and, lastly, Activity 3 being *Neutral* (32%).

5.8 Micro cycle: Workshop session six

This session (Appendix T) incorporated all previous coding concepts: conditional structures (*if statements*), looping structures (*for loops* and *while loops*) and nested statements. The highlight was coding the movement of an actuator (servo motor) according to the Fibonacci sequence.

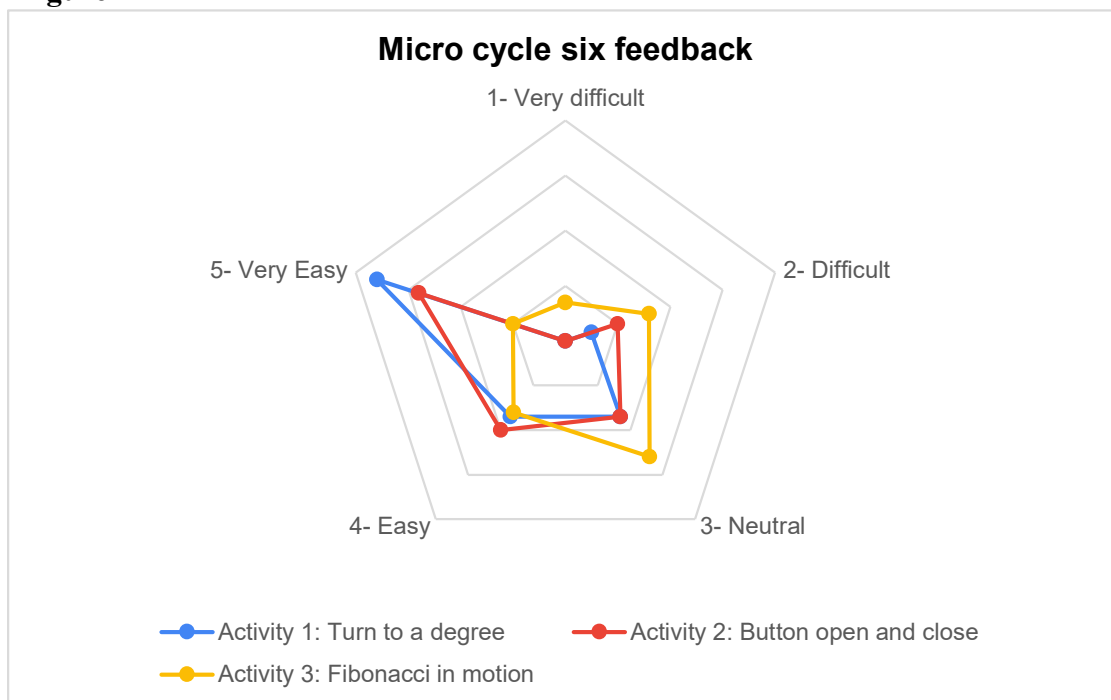
Table 10

Overview of micro cycle six feedback

	Activity 1: Turn to a degree	Activity 2: Button open and close	Activity 3: Fibonacci in motion
1- Very difficult	0%	0%	9%
2- Difficult	7%	13%	21%
3- Neutral	23%	23%	35%
4- Easy	23%	27%	21%
5- Very easy	48%	37%	13%
	100%	100%	100%

Note. Feedback based on self-evaluation for workshop session six (Appendix U).

Figure 42



Note. A graphical representation of the responses to the three activities in micro cycle six.

The majority of participants found Activity 1 to be *Very easy* (48%), followed by Activity 2 being *Very easy* (37%) and, lastly, Activity 3 being *Neutral* (35%).

5.9 Composite accomplishment ratings

Based on most responses, all accomplishments for the activities were ranged between *Neutral*, *Easy* and *Very easy* (Table 11).

Table 11

Composite accomplishment ratings based on the majority responses

	Activity 1	Activity 2	Activity 3
Workshop 1	Easy	Easy	Neutral
Workshop 2	Very easy	Easy	Neutral
Workshop 3	Neutral	Very easy	Neutral
Workshop 4	Very easy	Very easy	Easy
Workshop 5	Very easy	Very easy	Neutral
Workshop 6	Very easy	Very easy	Neutral

It is not surprising that most of the last activities (Activity 3) had a *Neutral* response since this activity was unguided, requiring participants to use their understanding and knowledge development from Activities 1 and 2. Similarly, most Activities 1 and 2 elicited a response of *Easy* and *Very easy*, as these activities were guided (Activity 1) or partially guided (Activity 2). Workshop 3 and 4 stand out from the other responses. In workshop 3, Activities 1 and 3 had *Neutral* responses. Activity 1 involved reading values from the Light Dependent Resistor (LDR) introduced in this workshop, while Activity 3 involved the use of the LDR and LED from the previous workshops. The *Neutral* accomplishment responses, especially for Activity 1, were likely due to the sensitivity of the LDR sensor. The sensitivity issue was highlighted to the participants during the activity workshop worksheet online. The values are quick to fluctuate due to light intensity and light exposure in the surroundings. All workshop Activity 4s were rated *Easy/Very easy*. This workshop involved prototypes using the flame sensor and buzzer introduced during the workshop.

5.10 Conclusion

All workshops utilised the UNO R3 microcontroller, jumper wires, breadboard, LEDs and resistors. Output was displayed on the PC screen in addition to output in the form of autonomous or physical change noted on the prototype. Workshop sessions 4, 5 and 6 consolidated the coding concepts, while sessions 1, 2 and 3 introduced the basics. Each workshop session introduced a new sensor leading to the design of a new prototype specific to the scenario. Based on the most accurate responses to the activities, all ranged between *Neutral*, *Easy* and *Very easy*. It is noted that the introduction to basic structures is challenging when using robotic elements to teach programming concepts. To design and create a meaningful prototype, one needs to apply basic programming structures much earlier in the coding. All accomplishment responses were fair and expected. It is important to note that Activities 1 and 2 in consolidation workshops 4, 5 and 6 were all *Very easy*. This supports the gradual cognitive development and confidence in Computer Programming. The next chapter will examine the findings and analyses from the data collection coalesced with discussion.

The findings and details of the analysis are discussed in the next chapter, which falls within the *Assessment phase* of DBR.

ASSESSMENT

Chapter six: Analysis and findings

“Should array indices start at 0 or 1? My compromise of 0.5 was rejected without, I thought, proper consideration.” ~ Stan Kelly-Bootle

6.1 Introduction

The previous chapter detailed and analysed the prototyping stage of the study. For coherence, *Chapter five: Prototyping* appears before *Chapter six: Analysis and Findings*, as in accordance with DBR. The purpose of *Chapter six* was to present the analysis and findings of the data collected using the instruments and methods described in the methodology chapter. This chapter provides analyses according to the study's aims and to answer the research questions. As the research questions were a guide to identifying crucial areas for investigation. All data was collected online through e-forms and e-questionnaires due to the COVID-19 pandemic.

6.2 Data presentation and analysis

The data collected was fed into statistical software packages that yielded descriptive statistics, inferential statistics and visual explanations. The latter was achieved through the creation of tables and graphs. The use of thick description together with statistical testing was applied; aligned to the principles of pragmatism and the practices of mixed-methods research. A significance level of 0.05 (*) was considered acceptable, which is common among social sciences and humanities. If a significance level of 0.01 (**) was achieved, it was noted during analysis. All interview recordings were transcribed verbatim, played more than once. Transcripts were cross-checked transcripts with the original recordings to ensure accuracy.

6.3 Pre-workshop session

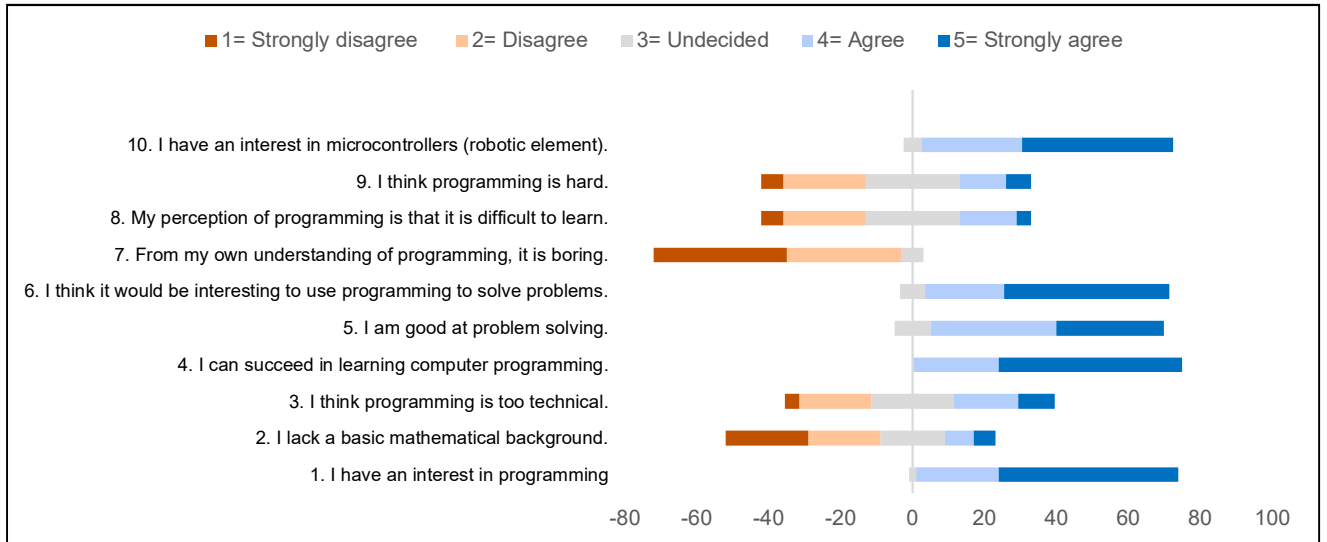
6.3.1 Pre-survey

The pre-survey was entitled *Computer Programming survey one* (Appendix F) and was given first to students to complete. This survey, in the form of a Likert scale, was given prior to any exposure to coding or to the robotic elements (prototype building). This survey did not collect biographical information. Instead, it sought to capture students' self-efficacy, perceptions and

attitude towards programming. The survey consisted of 10 items designed to gather initial perceptions from students regarding coding and Robotics.

Figure 43

Distribution of responses from pre-survey



Note. A visual representation of the responses from the pre-survey.

Shown in Figure 43, most responses fell in the Likert scale's extremes: *Strongly agree* and *Strongly disagree*, except items 3, 8 and 9. These exceptions indicate a more even distribution resulting in mixed reactions to items 3: *I think programming is too technical*; 8. *My perception of programming is that it is difficult to learn*; and 9. *I think programming is hard*.

Five items consisted of negatively worded statements and 5 items consisted of positively worded statements. For determining the Cronbach's Alpha (α), the five positively worded statements remained as is (Table 12, next page) while the five negatively worded statements were reverse coded before calculating α (Table 13, next page).

Table 12*Reliability statistics of non-reverse coded items*

Item	α if item Deleted
1. I have an interest in programming.	.684
4. I can succeed in learning Computer Programming.	.669
5. I am good at problem solving.	.712
6. I think it would be interesting to use programming to solve problems.	.623
10. I have an interest in microcontrollers (robotic element).	.637
*Reliability	.714

Note. Positively worded statements.

A Cronbach's Alpha above 0.70 is generally an acceptable value (Nunnally, 1978). As remarked by Hulin et al. (2001), an α of 0.60 - 0.70 indicates an acceptable reliability level. As shown in Table 12, the non-reverse coded items are therefore acceptable $\alpha \geq 0.7$. It is important to note that all the non-reverse coded items were framed from a personal perspective, with all statements starting with "I ..." prompting a response.

Table 13*Reliability statistics of reverse coded items*

Item	α if item Deleted
2. I lack a basic mathematical background.	.619
3. I think programming is too technical.	.635
7. From my own understanding of programming, it is boring.	.751
8. My perception of programming is that it is difficult to learn.	.621
9. I think programming is hard.	.502
*Reliability	.690

Note. Negatively worded statements.

Table 13 shows all reverse coded items are acceptable as $\alpha = 0.690$ (Hulin et al., 2001). As supported by Hair et al. (2006), one can accept values near 0.60. To probe further, each item from the pre-survey questionnaire follows (Tables 14 to 23). These tables provide a clear understanding of student self-efficacy, perceptions and attitudes towards programming. For reporting purposes, the percentages for *Strongly disagree* and *disagree* are combined for some; similarly, with *Agree* and *Strongly agree* in the excerpts that follow in Tables 14 to 23.

Table 14

Response to item 1. I have an interest in programming

	N	%
Undecided	2	2.67%
Agree	23	30.67%
Strongly agree	50	66.67%

The majority of students had a keen interest in programming (97.34% - *Agree* + *Strongly agree*) while a small percentage are undecided (2.67%). This is supported by the following excerpts:

“It was interesting to learn how we can use basic programming to do tasks” (P17, Focus Group Interview).

“Coding was interesting for me it is something that I have wanted to learn for a long time and I think it is a very valuable skill” (P15, Focus Group Interview).

Undecided responses may arise from students unsure of whether they find the robotic element or the coding aspect motivating, since the robotic element introduces a tangible component to the learning process and acts as a manipulative. The use of manipulatives in learning how to code are found to be a key motivation for students (Nugent et al., 2009; Merkouris et al., 2017) as they are actively involved in the learning process. The latter is affirmed by the following excerpts:

“It was an interesting journey, learning to work with a microcontroller” (P60, Focus Group Interview).

“The robot just makes the learning and understanding quicker” (P62, Focus Group Interview).

“Coding of the buzzer was very interesting. I feel like I can use that information in developing solutions in the future (knowledge development of coding the robotic element)” (P17, Focus Group Interview).

“Strategy I used was build my prototype then; I will code because after building I would have an idea of what the code must do and what must take place” (P18, Focus Group Interview).

Similar Yilmaz and Koc (2021) found that the student’s interest in learning how to code is stimulated through the manipulative. Alike Ioannou and Makridou (2018) found that students displayed high levels of engagement due to the physical object.

Table 15

Response to item 2. I lack a basic mathematical background.

	N	%
Strongly disagree	23	30.67%
Disagree	20	26.67%
Undecided	18	24.00%
Agree	8	10.67%
Strongly agree	6	8.00%

Most students (57.34% - *Disagree* + *Strongly disagree*) are of the view of having acquired the basics in mathematic skills. Interestingly, some students (18.67% - *Agree* + *Strongly agree*) say they do not have an essential mathematical background, while 24.00% are unsure. Khasawneh, et al. (2021) reason that maths confidence is associated with self-awareness and self-efficacy; thus having a positive outlook on one's mathematical ability may improve performance.

Table 16

Response to item 3. I think programming is too technical.

	N	%
Strongly disagree	4	5.33%
Disagree	20	26.67%
Undecided	23	30.67%
Agree	18	24.00%
Strongly agree	10	13.33%

Many students are unsure (30.67%) if programming is technical, while the majority agree (37.33% - *Agree* + *Strongly agree*) that programming is technical. Similar to the findings by Sobral (2021) that even with interventions, students find programming difficult, leading to a lack in motivation and high dropout rates. Interestingly some students do not think programming is too technical (32.00% - *Disagree* + *Strongly disagree*). Thus, the responses represent a relatively mixed view on the technicality of programming. It is important to note that all students met the criteria set out for the study, i.e., not having any exposure to Computer

Programming during the duration of their degree, but being currently registered in a computer course/module. The following excerpts support these mixed views:

“The coding started out to be really motivating at the beginning and then eventually I will just get bored of it... although it was amazing to code” (P42, Focus Group Interview).

“Coding somethings is tedious and then it's not like you can really see your progress when you stuck” (P1, Focus Group Interview).

“The thing that I liked the most was you could read the programming language and easily understand what it does” (P18, Focus Group Interview).

Table 17

Response to item 4. I can succeed in learning Computer Programming.

	N	%
Agree	24	32.00%
Strongly agree	51	68.00%

All students perceived that they could succeed in learning programming. In other words, none of the students doubt their capabilities, with the majority (68.00%) strongly agreeing that they could succeed in learning Computer Programming. Similar to the findings from Table 15, the cohort of students exhibit high confidence related to self-awareness and self-efficacy (Khasawneh et al., 2021). This 68.00% of students display intrinsic motivation and perceive programming as enjoyable and interesting (Zainal et al., 2012), as confirmed by the following:

“I really enjoyed this workshop it was an interesting journey, learning to work with a microcontroller. It was a bit difficult and it did require a bit of thinking, but nothing was impossible” (P60, Focus Group Interview).

“In high school, I did Java as a programming language. But now doing Python, it's less complicated and simple...everything is straightforward” (P15, Focus Group Interview).

Table 18

Response to item 5. I am good at problem solving.

	N	%
Undecided	10	13.33%
Agree	35	46.67%
Strongly agree	30	40.00%

Like the results in Table 17, most students held the view that they could excel in learning to program; the majority responded that they were good at problem solving (86.67%, *Agree + Strongly agree*). On the other hand, some were undecided (13.33%). Belski (2009) found that some students doubt their problem-solving abilities only to reflect their potential later.

Table 19

Response to item 6. I think it would be interesting to use programming to solve problems.

	N	%
Undecided	7	9.33%
Agree	22	29.33%
Strongly agree	46	61.33%

Interestingly, no student refuted the use of programming to solve problems, with a few being unsure (9.33%, undecided). Based on the sample selection criterion, it was expected for some students to be undecided as they had no previous exposure to Computer Programming and Robotics, as supported by the following excerpts:

“I got an idea on how they (robotic element) are really used in real-world situations ... so you get to know how things work” (P15, Focus Group Interview).

“They (robotic element) are very applicable in real-life problems” (P9, Focus Group Interview).

“I enjoyed the problem solving; I enjoy combining things and then putting them together to make up one thing” (P26, Focus Group Interview).

Table 20

Response to item 7. From my own understanding of programming, it is boring.

	N	%
Strongly disagree	37	49.33%
Disagree	32	42.67%
Undecided	6	8.00%

A minority of students (8.00%) were unsure if programming is boring or not. On the other hand, most students reported that Computer Programming is not mundane (92.00%, *Disagree* + *Strongly disagree*). It was noted earlier (*Chapter two: Literature review*), programming can be complex and difficult (Rubio et al., 2013), which leads to boredom (Khaleel et al., 2017). However, through manipulatives, such as robots (Kurebayashi et al., 2006) and game design (Govender & Govender, 2020), negative attitudes towards Computer Programming can be changed.

Table 21

Response to item 8. My perception of programming is that it is difficult to learn.

	N	%
Strongly disagree	6	8.00%
Disagree	23	30.67%
Undecided	26	34.67%
Agree	16	21.33%
Strongly agree	4	5.33%

Some students were unsure if programming is difficult (34.67%, *Undecided*) as this was their first encounter with coding. A minority viewed programming as difficult (26.66%, *Agree* + *Strongly agree*). The majority of students responded that programming is not difficult to learn (38.67%, *Disagree* + *Strongly disagree*).

Table 22

Response to item 9. I think programming is hard.

	N	%
Strongly disagree	6	8.00%
Disagree	23	30.67%
Undecided	26	34.67%
Agree	13	17.33%
Strongly agree	7	9.33%

Strikingly, the results in Table 22 are similar to Table 21 regarding programming as challenging to learn for *Strongly disagree*, *Disagree* and *Undecided*. The minority of students thought programming is complex (26.66%, *Agree* + *Strongly agree*), similar to the results in Table 21. The majority believed that programming is not hard (38.67%, *Disagree* + *Strongly disagree*), again similar to the results in Table 21. The uncertainty (34.67%) of whether programming is hard or not is, once again, similar to the results in Table 21.

Table 23

Response to item 10. I have an interest in microcontrollers (robotic element).

	N	%
Undecided	5	6.67%
Agree	28	37.33%
Strongly agree	42	56.00%

A large number of students purported to have an interest in microcontrollers (93.33%, *Agree* + *Strongly agree*), while few are undecided (6.67%). Thereby suggesting and confirming the results in Table 14 that the robotic element had definitely sparked curiosity and interest towards the learning of programming. Supported by the findings by Merkouris et al. (2018) that manipulatives such as the microcontroller used in this study stimulated students' interest and kept students engaged while learning how to code.

“I struggled with some programming; it can get quite abstract, but the microcontroller makes it most fun and I feel like it solidifies understanding in some way” (P62, Focus Group Interview).

“With this workshop having to hold and work the microcontroller made it (coding) interesting” (P42, Focus Group Interview).

6.3.2 Questionnaire test one~ Problem solving and logic

6.3.2.1 Part A: Pre-test based on Computational Thinking

Part A (Appendix G) of the problem solving and logic pre-test based multiple-choice questions (MCQ), tested students' knowledge of concepts they would come across during the workshop sessions. In addition, the questionnaire sort to obtain a general understanding of students' ability to think, reason and exercise logic; hence a Computational Thinking test. As Romero et al. (2017) state “programming is not only about writing code, but also about the capacity to analyse a situation, identify its key components, model the data and processes, and create or refine” (p. 2), which is part of the Computational Thinking process. Prior to answering the MCQs, student bibliographic information (Table 24) was gathered. As predicted in *Chapter four: Research design and methodology, section 4.8*, the majority of the students were first-year students (30.67%). All students had some form of pre-university mathematical experience, with the majority having done pure Mathematics at school (56%).

Table 24

Bibliographic information

Item	Options	N	%
Year of Study	1 st	23	30.67%
	2 nd	21	28.00%
	3 rd	11	14.67%
	4 th	20	26.67%
	N	75	100%
Age	16-17	0	0%
	18 -19	18	24.00%
	20-21	26	34.67%
	>21	31	41.33%
	N	75	100%
Phase	FET	20	26.67%
	Senior FET	43	57.33%
	Intermediate Senior	11	14.67%
	ECD	1	1.33%

	N	75	100%
Math background	Mathematical Literacy	29	38.67%
	Pure Mathematics	42	56.00%
	Mathematics SG	0	0%
	Mathematics HG	3	4.00%
	Vocational Mathematics	1	1.33%
	No Mathematics	0	0%
	Other	0	0%
	N	75	100%

Table 25 shows the percentage responses per option for each question and the common response for each question. (Refer to Appendix G for the questionnaire.)

Table 25

Overall summary response from Part A (pre-test)

Question	A	B	C	D	Mode
1.	·29.30%	0.00%	2.70%	68.00%	D
2.	8.00%	·58.70%	13.30%	20.00%	B
3.	20.00%	9.30%	·44.00%	26.70%	C
4.	21.30%	·50.70%	18.70%	9.30%	B
5.	30.70%	10.70%	·46.70%	12.00%	C
6.	·46.70%	4.00%	10.70%	38.70%	A
7.	1.30%	·49.30%	25.30%	24.00%	B
8.	2.70%	1.30%	·54.70%	41.30%	C
9.	13.30%	4.00%	5.30%	·77.30%	D
10.	·17.30%	5.30%	18.70%	58.70%	D

Note. The dot above symbol (·) is used to indicate the correct answer.

Drawing from the modal shown in Table 25, most students were correct for 8 out of the 10 MCQs. Table 26 indicates that the majority of students obtained a mark of 5 out of 10 (20.00%). In addition, more than fifty percent of the students (56.00%) achieved a mark of fifty percent (5 out of 10) and above in the Computational Thinking test. The findings show that 8% of students scored a mark of eighty percent and above (8 out of 10), and only 1.33% (n=1)

achieved 9 out of 10, which was the highest mark obtained for the Computational Thinking test.

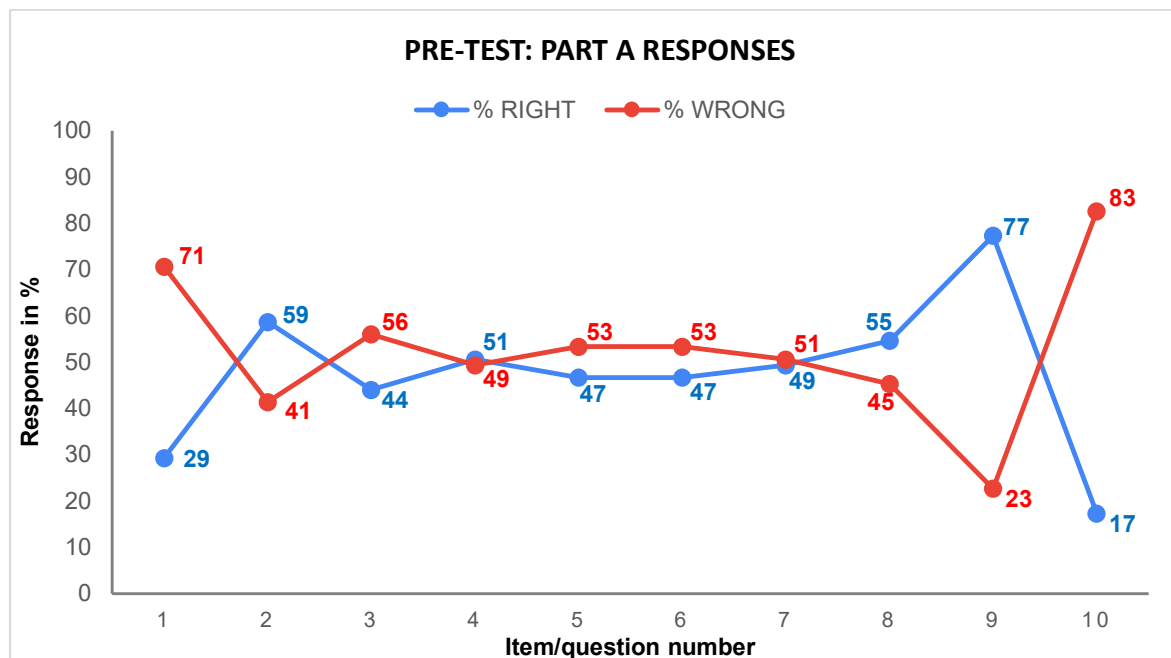
Table 26

Distribution of Part A (pre-test) marks obtained (out of 10)

Mark out of 10	N	%
1	4	5.33%
2	4	5.33%
3	12	16.00%
4	13	17.33%
5	15	20.00%
6	13	17.33%
7	8	10.67%
8	5	6.67%
9	1	1.33%
10	0	0.00%
N	75	100%

Figure 44

Right vs Wrong responses for Part A



Note. A visual representation of correct and incorrect responses based on Table 25; percentages rounded.

From Figure 44, we note that there are extreme differences in the range for correct and incorrect answers to questions one, nine and ten (see Appendix G). On closer inspection of question one, the majority of students selected option *D - All of the above* (Table 25) as opposed to the correct answer *A - Is a series of instructions* (Table 27, next page), which was the second-highest option selected (Table 25). The possibility of the high selection of option A (29.33%) and option D (68.00%) could be a result of the question text containing the words *best described* in response to what a computer program is. Hence, the likelihood that most selected option *D - All of the above*.

Table 27*Summary of responses to question one*

	Options	N	%
A.	<i>Is a series of instructions</i>	22	29.33%
B.	<i>Can be short or long</i>	0	0%
C.	<i>Is written with code</i>	2	2.67%
D.	<i>All of the above</i>	51	68.00%

It was encouraging that the majority of students selected option D (77.33%, Table 28) for question nine, which was the correct answer (Table 25). Question nine was based on speed and movement; which participants will be exposed to when they cover motor movement during the online workshop sessions. (Refer to Appendix G for the diagrammatic options A, B, C and D for question nine).

Table 28*Summary of responses to question nine*

	Options	N	%
A.		10	13.33%
B.		3	4.00%
C.		4	5.33%
D.		58	77.33%

For question ten, the most selected option was D (58.67%, Table 25), which is incorrect. Question A (17.33%) being the correct answer (Table 29).

Table 29*Summary of responses to question ten*

	Options	N	%
A.		13	17.33%
B.		4	5.33%
C.		14	18.67%
D.		44	58.67%

The majority selected option *D - Line 4 contains Z=95*, which is not displayed (output). Although the variable Z is not directly displayed, the value is used to calculate another variable

that is displayed. (Refer to Appendix G for the algorithm and line number options for *A*, *B*, *C* and *D* for question ten. Given that students lack exposure to programming at this stage, and are still to develop their coding skills through the workshop, most did not realise the indirect use of the variable. As an individual’s coding knowledge and skills develop, such use of variables as in question ten, are instantly recognised as a result of the prior knowledge of trace tables, flowcharts and other debugging techniques that are introduced in the early stages of programming (Saeli et al., 2011; White & Sivitanides, 2005).

Table 30

Five-point number summary with mean and standard deviation (SD) for Part A (pre-test)

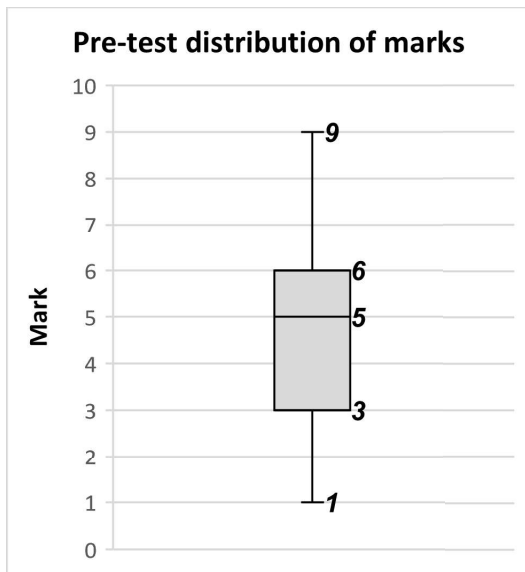
Item	Value
Minimum	1
Quartile 1	3
Quartile 2 (median)	5
Quartile 3	6
Maximum	9
Mean	4.747
Standard deviation	1.904

Note. Total for test: 10 marks.

It is noticed that the majority of students achieved a mark of five out ten (20%, Table 26). This mark, although very close, is greater than the sample mean $\bar{x} = 4.747$ (Table 30). The box plot (Figure 45) offers a virtual representation of the 5-point summary from Table 30. The highest mark obtained was 9 out of 10, while the lowest was 1 out of 10.

Figure 45

Summary of marks for Part A

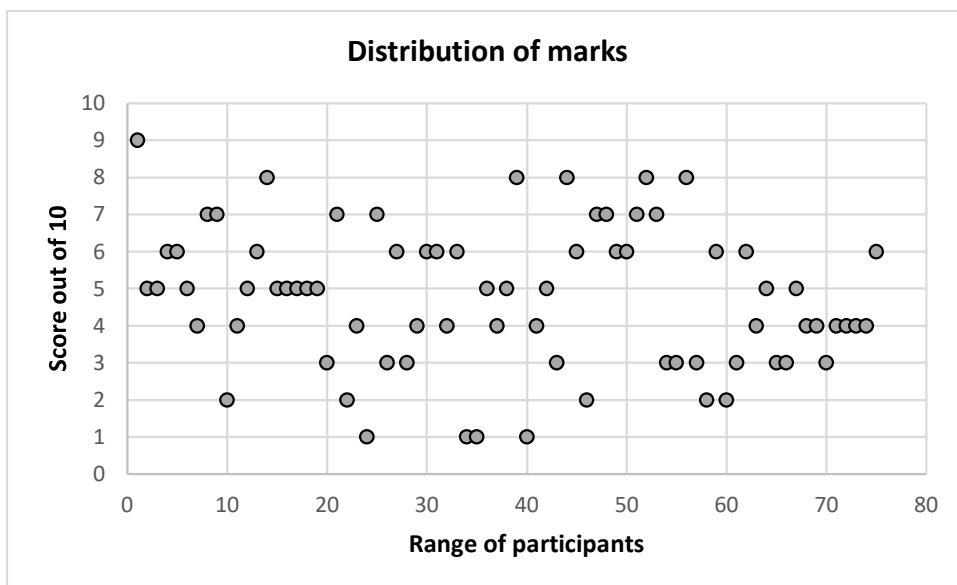


Note. Box plot base on marks obtained from the ten MCQs (Part A).

There were no outliers reported as is confirmed by the scatter plot (Figure 46), which shows a wide spread of the marks obtained by the students.

Figure 46

Distribution of marks for Part A



Note. Distribution of marks obtained by students in Part A.

The data (marks for Part A) is normally distributed since the mean (\bar{x}) is equal to the median (quartile 2) (Table 30). The marks (raw values out of 10) are standardised to a normal distribution by converting them into z-scores (Table 31); allowing the proportion of the values that fall within a specified number of standard deviations (σ) from the mean (\bar{x}) population. Therefore, the data being normally distributed adheres to the empirical rule known as the three-sigma rule or the 68-95-99.73 rule (Pukelsheim, 1994); indicating that approximately 68% of the data falls within one standard deviation (SD) of the mean; while approximately 95% of the data falls within two standard deviations of the mean; and approximately 99.73% of the data falls within three standard deviations of the mean. The distribution data is further supported by creating the bell curve (Figure 47, page after next), displaying three standard deviations based on Table 31.

Table 31

Standardisation of values (raw marks out of 10)

SD increments	Mean*SD*SD increments	Normalize/Z-distribution
-3.00	-0.711	0.002
-2.90	-0.521	0.003
-2.80	-0.330	0.004
-2.70	-0.140	0.005
-2.60	0.050	0.007
-2.50	0.241	0.009
-2.40	0.431	0.012
-2.30	0.621	0.015
-2.20	0.812	0.019
-2.10	1.002	0.023
-2.00	1.193	0.028
-1.90	1.383	0.034
-1.80	1.573	0.041
-1.70	1.764	0.049
-1.60	1.954	0.058
-1.50	2.144	0.068
-1.40	2.335	0.079
-1.30	2.525	0.090
-1.20	2.716	0.102
-1.10	2.906	0.114

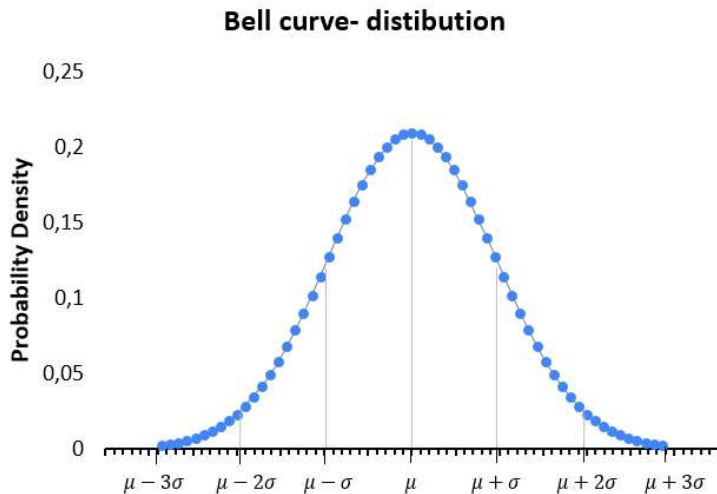
-1.00	3.096	0.127
-0.90	3.287	0.140
-0.80	3.477	0.152
-0.70	3.667	0.164
-0.60	3.858	0.175
-0.50	4.048	0.185
-0.40	4.239	0.193
-0.30	4.429	0.200
-0.20	4.619	0.205
-0.10	4.810	0.209
0.00	5.000	0.210
0.10	5.190	0.209
0.20	5.381	0.205
0.30	5.571	0.200
0.40	5.761	0.193
0.50	5.952	0.185
0.60	6.142	0.175
0.70	6.333	0.164
0.80	6.523	0.152
0.90	6.713	0.140
1.00	6.904	0.127
1.10	7.094	0.114
1.20	7.284	0.102
1.30	7.475	0.090
1.40	7.665	0.079
1.50	7.856	0.068
1.60	8.046	0.058
1.70	8.236	0.049
1.80	8.427	0.041
1.90	8.617	0.034
2.00	8.807	0.028
2.10	8.998	0.023
2.20	9.188	0.019
2.30	9.379	0.015
2.40	9.569	0.012
2.50	9.759	0.009
2.60	9.950	0.007

2.70	10.140	0.005
2.80	10.330	0.004
2.90	10.521	0.003
3.00	10.711	0.002

Note. Mean=5 and standard deviation = 1,904.

Figure 47

Visualisation depicting normal distribution; based on data from Table



Note. Normal Bell shape obtained based on Part A.

Aligned to the research aim exploring external factor/s that contribute towards the learning of Computer Programming, the marks achieved (dependent variable) were cross-tabulated against the type of Mathematics (independent variable) experience that students had (Table 32). As mentioned in the *Chapter two: Literature review*, Computer Programming is a subgroup of Computational Thinking. Thus, there is the possibility that an individual's experience of Mathematics will influence their HOT logical thinking or problem-solving abilities, which are required for Computer Programming (García-Peñalvo, 2018; Jacob et al., 2018). As discussed earlier in this chapter and shown in Table 30 and Figure 45, no student obtained a full mark of 10 out of 10. The highest mark achieved was 9 out of 10 by one student (Figure 46). Hence, cross-tabulation provided further insight into the relationship between *type of Mathematics* and *mark obtained* from Part A (computational think test), as shown in Table 32.

Table 32

Differences depending on the type of Mathematics vs MCQ mark obtained using descriptive statistics

	Mark obtained									N	M	SD
	1	2	3	4	5	6	7	8	9			
Mathematical Literacy	3	2	4	7	2	6	4	1	0	29	4.45	1.99
	4%	3%	5%	9%	3%	8%	5%	1%	0%	38.67%		
Pure Mathematics	1	3	7	7	11	7	1	4	1	42	4.9	1.84
	1%	4%	9%	9%	15%	9%	1%	5%	1%	56.00%		
Mathematics HG	0	0	0	0	0	0	3	0	0	3	7.00	0.00
	0%	0%	0%	0%	0%	0%	4%	0%	0%	4.00%		
Vocational Mathematics	0	0	0	0	1	0	0	0	0	1	5.00	0.00
	0%	0%	0%	0%	1%	0%	0%	0%	0%	1.33%		
N	4	5	11	14	14	13	8	5	1	75		
	5.33%	6.67%	14.67%	18.67%	18.67%	17.33%	10.67%	6.67%	1.33%	100%		

Table 33

Mark distribution based on percentage weighting per mark obtained

	Mark obtained								
	1	2	3	4	5	6	7	8	9
Mathematical Literacy	10%	7%	14%	24%	7%	21%	14%	3%	0%
Pure Mathematics	2%	7%	17%	17%	26%	17%	2%	10%	2%
Mathematics HG	0%	0%	0%	0%	0%	0%	100%	0%	0%
Vocational Mathematics	0%	0%	0%	0%	100%	0%	0%	0%	0%

The number of participants for Mathematics HG and Vocational Mathematics is four and it was, therefore, decided to be omitted in further analysis. The statistical software results support this decision (*see notes below, Table 34*). It is common knowledge that computers were first developed to help solve complex mathematical problems, dating back to the ¹²Abacus. Mathematics ability is, therefore, a good indicator of the cognitive potential to learn to code. Saeli et al. (2011) suggested that it is not unusual to find Mathematics concepts in programming. This is due to the nature and approach of programming whereby one would automatically encounter fundamental concepts in mathematic. Inferring from Tables 32 and

¹² Abacus is tool used to perform mathematical calculation by counting beads along rods or grooves (Chappelow, 2020).

33, it can be noted that the type of Mathematics has a significant impact on the mark achieved. Since 12% Pure Mathematics background (Table 33) participants achieved above 8 out of 10 compared with 3% Mathematical Literacy background (Table 33) participants achieved above 8 out of 10. In addition, 55% of Mathematical Literacy background (Table 33) participants obtained below 5 out of 10 compared with 43% of Pure Mathematics background (Table 33) participants.

While Pure Mathematics seems to be a contributing factor, this does not mean that one needs to possess advanced mathematical skills or estimable mathematical knowledge. Instead, the type of rational exercises required in Mathematics are necessary for Computer Programming. Findings by Bubica and Boljat (2015) state that “assessing programming students have proven to be ineffective for students who have previous experience of programming and students who have poor knowledge of mathematical algebra” (p. 5884). This indicates that Mathematics might not be the required element to understand programming.

Table 34

Descriptive statistics based on the type of mathematics

Choose one that best describes your school-leaving Mathematics		Statistic	Std. error	
Mathematical Literacy	Mean	4.45	.370	
	95% Confidence Interval for mean	Lower Bound	3.69	
		Upper Bound	5.21	
	5% Trimmed mean	4.46		
	Median	4.00		
	Variance	3.970		
	Std. deviation	1.993		
	Minimum	1		
	Maximum	8		
	Range	7		
	Interquartile range	3		
	Skewness	-.151	.434	
	Kurtosis	-.886	.845	
	Pure Mathematics	Mean	4.79	.284
95% Confidence Interval for mean		Lower bound	4.21	
		Upper bound	5.36	
5% Trimmed mean		4.76		
Median		5.00		
Variance		3.392		

	Std. deviation	1.842	
	Minimum	1	
	Maximum	9	
	Range	8	
	Interquartile range	3	
	Skewness	.282	.365
	Kurtosis	-.186	.717
Mathematics HG^a			
Vocational Mathematics^a			

Note. Scale is constant when Choose one that best describes your school-leaving Mathematics = Mathematics HG and Mathematics = Vocational Mathematics. These have been omitted.

According to Hair et al. (2010) and Bryne (2010), data is considered normal if skewness is between -2 to +2 and kurtosis is between -7 to +7. There is no clear indication of the skew and kurtosis values to indicate non-normality; however, the skewness z-value and kurtosis z-value are generally between -1.96 and +1.96 (Blanca et al., 2013). To confirm normality, the calculation of the skewness and kurtosis z-values are obtained by dividing the skewness measure by its standard error (SE) (from Table 34).

Descriptive analysis was carried out to determine if the Mathematical Literacy and Pure Mathematics marks are normally distributed (Table 35). For normality, the skewness and kurtosis measures should be close to as zero as possible, since in reality, data are often skewed and kurtotic (Kim & White, 2004; Brown, 1997). Therefore, a slight departure from zero is consequently no issue.

Table 35

Calculation of the skewness z-value and kurtosis z-value

	Skewness	Kurtosis
Mathematics literacy	$= \frac{-0.151}{0.434} = -0.348$	$= \frac{-0.886}{0.845} = -1.049$
Pure Mathematics	$= \frac{0.282}{0.365} = 0.772$	$= \frac{-0.186}{0.717} = -0.259$

All calculated z-values are between -1.96 and +1.96 (Table 35). Therefore, regarding the skewness and kurtosis, the data are slightly skewed and kurtotic for both Mathematics Literacy

and Pure Mathematics, but it does not differ significantly from normality. It can be assumed that the data are approximately normally distributed in terms of skewness and kurtosis.

Table 36

Shapiro-Wilk test of normality

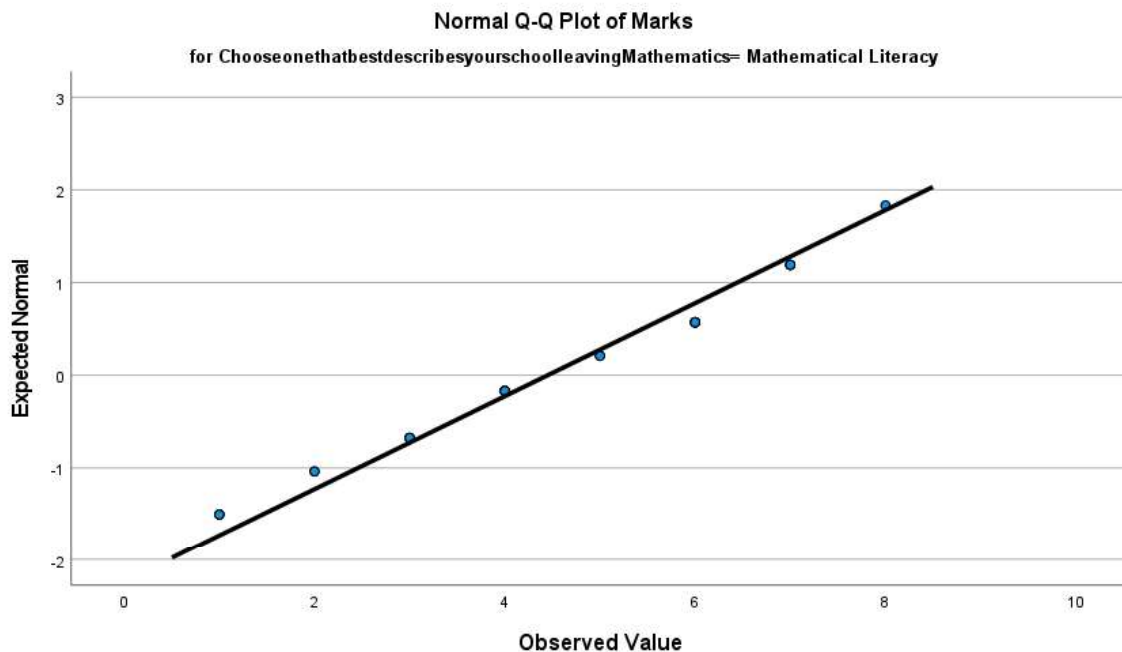
Choose one that best describes your school-leaving Mathematics	Shapiro-Wilk test		
	Statistic	df	Sig.
Mathematical Literacy	.944	29	.128
Pure Mathematics	.959	42	.131
Mathematics HG ^a			
Vocational Mathematics ^a			

Note. Scale is constant when Choose one that best describes your school-leaving Mathematics = Mathematics HG and Mathematics = Vocational Mathematics. These have been omitted.

The null hypothesis (H_0) for the Shapiro-Wilk test of normality is that the data are normally distributed. Hence the alternative hypothesis (H_a) is that the data is not normally distributed. As shown in Table 36, both p values are above 0.05; therefore, failing to reject the H_0 resulting in the data being normally distributed.

Figure 48

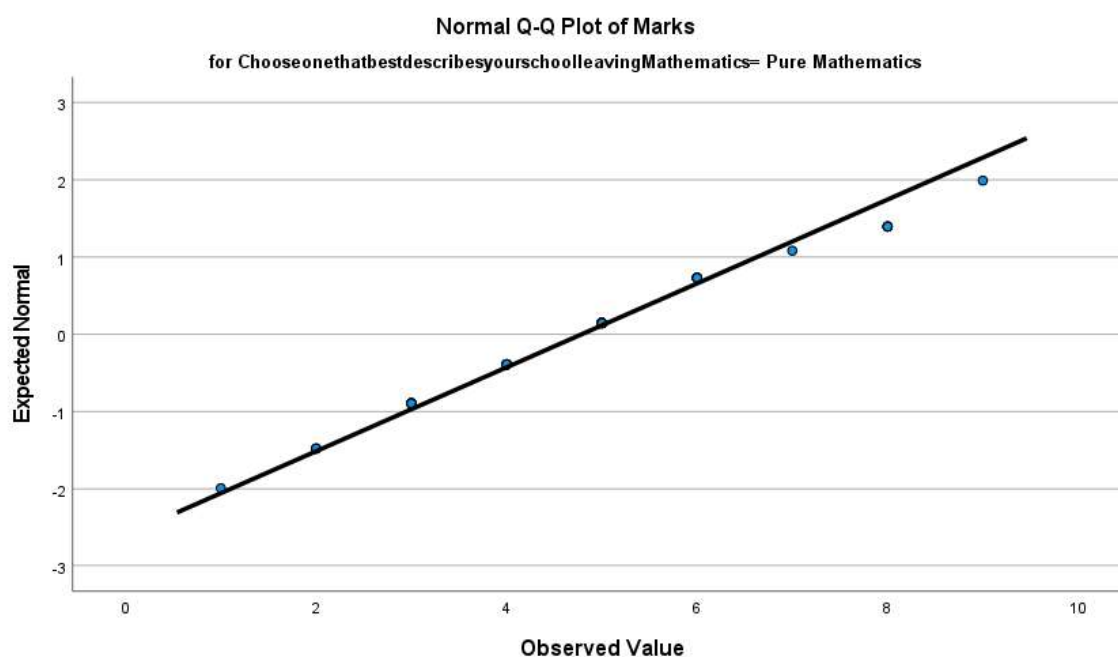
Q-Q Plot Mathematical Literacy



Note. Quantile-Quantile (Q-Q) plot visualising the distribution of Mathematical Literacy.

Figure 49

Q-Q Plot Pure Mathematics



Note. Quantile-Quantile (Q-Q) plot visualising the distribution of Mathematics.

A Shapiro-Wilk's test ($p > 0.05$) (Razali & Wah, 2011; Shapiro & Wilk, 1965) and a visual inspection of the normal Q-Q plots (Figure 48 & Figure 49) show that the marks were approximately normally distributed for both Mathematics Literacy and Pure Mathematics, with skewness of -0.151 (SE¹³=0.434) and kurtosis of -0.886 (SE=0.845) for Mathematic Literacy and skewness of 0.282 (SE=0.365) and kurtosis of -0.186 (SE=0.717) for Pure Mathematics (Doane & Seward, 2011; Cramer & Howitt, 2004; Cramer, 1998). The data is approximately normally distributed allowing for further investigative analysis and exploration through parametric methods later on.

6.3.2.2 Part B: Abstract Reasoning Test

The Abstract Reasoning Test (ART) formed the second part of the problem-solving and logic data collection tools known as Part B. An ART uses symbols and shapes instead of words and numbers, and measures an individual's ability to identify patterns, logical rules and trends in new data (Simanjuntak et al., 2018). The ART (Appendix H) consisted of 25 items with a 20-minutes time limit to answer. This test responds to research question two coupled with other

¹³ Standard error

data captured on students' biographical information, such as, Mathematics background and the questionnaire based on problem-solving skills.

There is no passing score for the ART and other psychometric tests, such as verbal and numerical reasoning. The results obtained from the abstract reasoning test are calculated relative to that of other students in the group in terms of the number of attempted questions and the number of correctly answered questions (Warne et al., 2014). The maximum raw score obtained from the ART based on N=75 was 21 out of 25. Hence all scores were adjusted according to the maximum weighting; with 21 becoming the benchmark representing 100%. Table 37 provides a summary of the scores obtained from the ART.

Table 37

Five-point number summary with mean and SD Part B

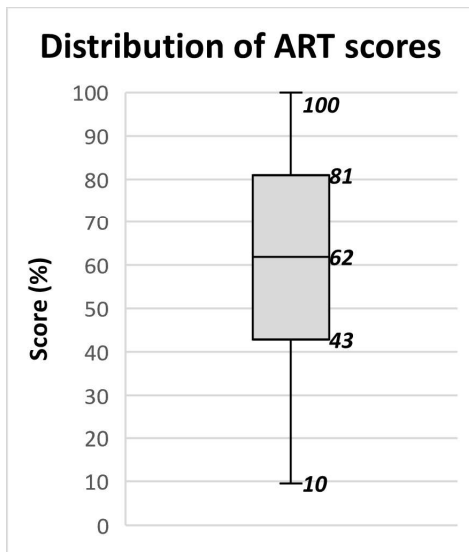
Item	Value
Minimum	10
Quartile 1	43
Quartile 2 (median)	62
Quartile 3	81
Maximum	100
Mean	60.190
Standard deviation	23.455

Note. Summary of ART scores for Part B. Maximum obtained was 21 out 25 which is converted to 100%.

The box plot (Figure 50, next page) offers a virtual representation of the distribution of scores for the ART based on the 5-point summary (Table 37).

Figure 50

Summary of scores for Part B: ART



Note. Box plot base on scores obtained from the ART.

In order to determine if the scores from the ART are normally distributed, a Kolmogorov Smirnov test (Table 38) was carried out, and the Q-Q plot (Figure 51) examined. As Das and Imon (2016) state, “normality can be assessed both visually and through normality tests” (p.11). The null hypothesis (H_0) for the Kolmogorov Smirnov test of normality is that the data are normally distributed. Hence the alternative hypothesis (H_a) is that the data are not normally distributed.

Table 38

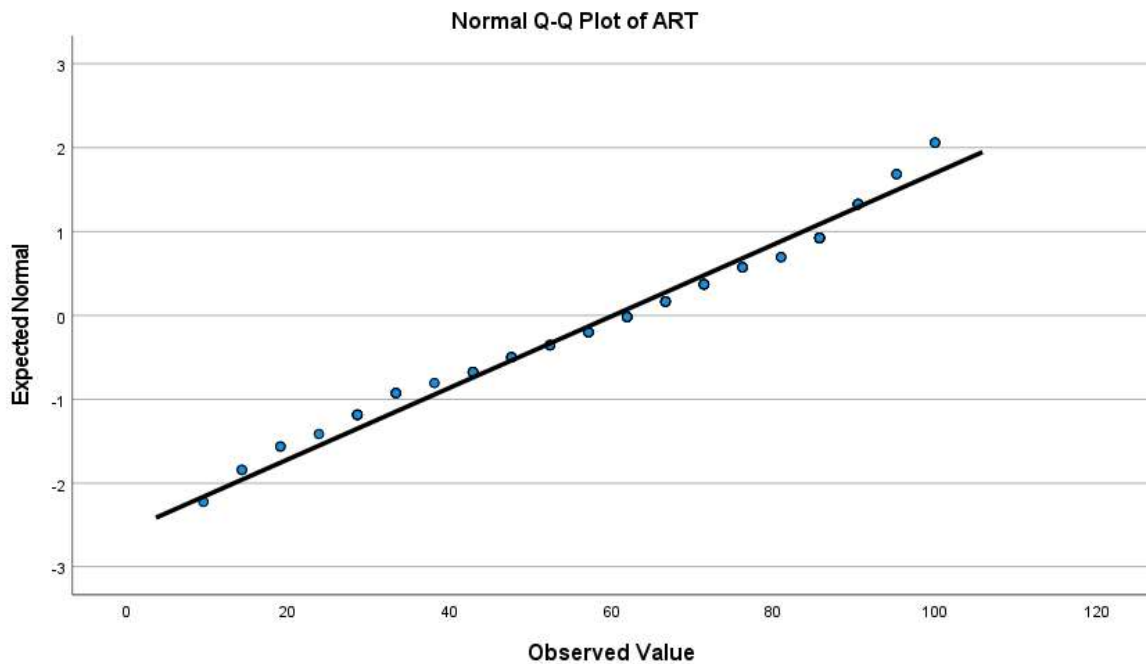
Kolmogorov-Smirnov test of normality

	Statistic	df	Sig.
ART	.088	75	.200

Note. Kolmogorov-Smirnov test of normality on Part B: Abstract Reasoning Test.

Figure 51

Q-Q Plot Part B: ART



Note. Quantile-Quantile (Q-Q) plot visualising distribution ART.

A Kolmogorov Smirnov test ($p > 0.05$) and a visual inspection of the normal Q-Q plot (Figure 51) show that the ART scores are not significantly different from a normal distribution. Hence the H_0 (null hypothesis) is accepted, meaning that the data are approximately normally distributed.

Although a Kolmogorov Smirnov test was used to determine normality instead of the Shapiro-Wilk test used earlier, the Q-Q plot was examined in both normality tests. As supported by Razali et al. (2011), “The normal quantile-quantile plot (Q-Q plot) is the most commonly used and effective diagnostic tool for checking normality of the data” (p.21).

The Admission Points Score (APS) was the converted average of an individual’s ¹⁴Grade 12 marks at the end of the school-leaving year. Each mark was converted to a mark out of 10 and then added together to give an APS. An individual’s APS determines if they are eligible for a specific tertiary education course. The APS is an ideal benchmark indicator for higher education institutions in determining if a prospective student is capable of a particular course.

¹⁴ Grade 12 is final grade in school

The minimum requirement for undergraduate qualifications to enter the Bachelor's Degree is 21 points, although some exceptions may apply between institutions and courses (Together we pass, 2019). The maximum point that an individual can be rewarded is 7 per subject for 7 subjects; therefore, the maximum number of points that can be achieved is 49. The average acceptable APS that prospective students entering into the Bachelor's Degree is $21/49 \times 100 = 42.857\%$. It is worthy to determine if there is any significant difference between the ART mean and the APS mean. The ART involves pictorial representation that measures an individual's ability to identify patterns, logical rules and trends relevant to Computer Programming. These qualities are harnessed in the development of learning to code (Robert, 2017; Roberts, 2009).

A sample t-test compares the mean of a sample to a hypothesized population mean. This comparison will indicate any significant difference between the two means. In this study, a one-sample t-test, otherwise known as a single sample t-test, is ideal for comparing the ART and the APS means. Based on the substantive reasoning highlighted in the previous paragraph, the test variable mean is calculated from the ART scores, which is compared with the test value, which is the average acceptable APS. To carry out the t-test, it is noted that the four main assumptions are met: the dependent variable is continuous data, no potential outliers in the dependent data (Figure 51), observations are independent of one another and the dependant variable is approximately normally distributed (Table 38 and Figure 51).

The sample mean (\bar{x}) in reference to the test variable (Part B: ART) = 60,190%. The known or hypothesized mean value in the population (μ) in reference to the test value (APS) = 42.857%. Statement: To check if national APS to enter a Bachelor degree is equal to ART.

H_0 = There is no significant difference in the ART of students who are part of the coding and robotic workshop and APS; meaning the population mean is equal to the proposed mean ($\mu = \mu_0$).

H_a = There is a significant difference in the ART of students who are part of the coding and robotic workshop and APS; meaning population mean is not equal to the proposed mean ($\mu \neq \mu_0$).

Table 39*One-sample t-test*

	t	df	Sig. (2-tailed)	Mean difference	95% Confidence Interval of the difference	
					Lower	Upper
ART	6.400	74	.000**	17.333	11.94	22.73

Note. One-sample t-test testing significance difference in ART and APS mean.

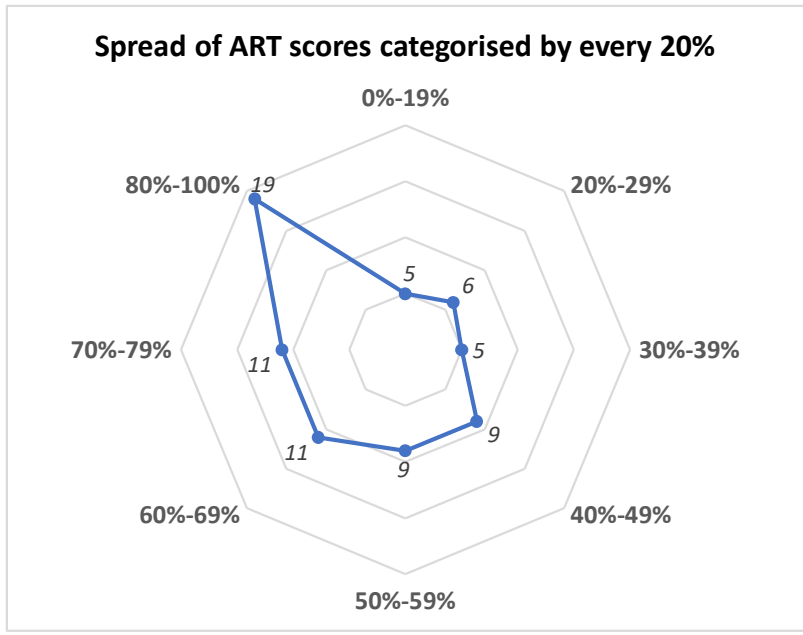
Test value = 42.857

** significance at $p < 0.01$ (2-tailed).

A one-sample t-test (Table 39) was carried out to assess if ART scores (Part B of the problem-solving and logic test) differ significantly compared with the APS. The descriptive statistics showed that 60.190 is the average score obtained from the sample, and 23.455 is the SD (Table 37). The results reveal a significant difference in the ART compared with APS; $t(74) = 6.400$, $p < 0.05$. Hence the average ART is significantly different from the average APS. This shows that the sample does not represent the population; therefore, H_0 is rejected ($\mu \neq \mu_0$). Under the null hypothesis, the mean difference would have equalled to 0; however, it is 17.333. When examining the lower and upper values, it is clear that a mean of 0 falls outside these limits, supporting the acceptance of H_a . The rejection of the null hypothesis is evidence that this sample was likely not drawn from a population such that the mean APS equals 42.857%. Therefore, it substantially indicates that students who chose to be part of the workshop were interested in coding and Robotics and have ART scores not significant to the average APS. Further inspection reveals that the ART mean $>$ APS mean, which indicates the students who were part of the workshop scored above the average APS.

Figure 52

Spread of Part B: ART scores



It is interesting to note that majority of students achieved between 80% and 100% (Figure 52), with some students achieving a maximum of 100%. Furthermore, 56% $(11+11+19/75 * 100)$ (Figure 52) of students achieved above the mean of 60% (Table 37).

Both problem-solving and logic data sets, namely Part A: Pre-test based on Computational Thinking and Part B: ART comprise continuous data, marks and scores, respectively. Hence, Pearson correlation (Table 40) is ideal for determining a relationship between Part A and Part B.

Table 40

Pearson correlation

		Part A	Part B
Part A	Pearson correlation	1	.416**
	Sig. (2-tailed)		.000
	N	75	75
Part B	Pearson correlation	.416**	1
	Sig. (2-tailed)	.000	
	N	75	75

Note. Pearson correlation is used to determine the relationship between problem-solving and logic data sets. Part A: Pre-test based on Computational Thinking and Part B: ART.

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

Pearson correlation based on the Computational Thinking pre-test marks and ART scores shows a correlation ($p < 0.05$). There is a significant positive relationship ($p \leq 0.05$) between the ART and MCQ logical test, $r(73) = 0.416$, $p = 0.000$. The study of Computer Science, particularly concerning programming, is viewed as the study of algorithms (Káta, 2014; Milková, 2012). As remarked by Hromkovič et al. (2016), “Computer Science is a vast field with algorithmic thinking at its core” (p.114). Thus, to be a successful computer programmer, it would seem that one needs to develop and acquire advanced thinking skills, such as problem-solving, logical and mathematical thinking, critical thinking, and creative thinking (Fang, 2012).

*The six workshops were analysed in the prototyping phase of DBR - *Chapter five: Iteration*.

6.4 Post-workshop session

6.4.1 Post-survey

This study applies Partial Least Squares-Structural Equation Modelling (PLS-SEM) to assess the data generated from the post-survey. PLS-SEM is commonly used in business industrial research and analysis; however, the statistical modelling technique has gained popularity in education research (Law & Fong, 2020; Lin et al., 2020). Research by Astrachan et al. (2014) advocate that PLS-SEM offers many advantages over Covariance Based-Structural Equation Modelling (CB-SEM), one significant advantage being sample size. Another critical difference between CB-SEM and PLS-SEM is the approach used when assessing the *structural model*. In CB-SEM, the fit is established on accurately estimating the observed covariance matrix, while PLS-SEM fit is established upon accounting for explained variance in the endogenous constructs (Hair et al., 2014).

Ramli et al. (2018) found that PLS-SEM analysis offers fewer contradictory results than regression analysis, despite PLS-regression models being a subset of PLS-SEM models. Garson (2016) explains that PLS-SEM models differ from regression models as they are “path models in which some variables may be effects of others while still be causes for variables later in the hypothesized causal sequence” (p.13). PLS-SEM analysis consists of two parts, firstly examining the *measurement model* that consists of the indicator reliability, convergent reliability and discriminant validity. Secondly, the *structural model* assesses collinearity issues,

path coefficients, the significance of the relationships, level of R^2 , effect size (f^2) and predictive relevance (Q^2).

Mainly in response to research question one, the following definition was set out before proceeding to feed the data generated from the post-survey into the PLS-SEM software: Explore the factors contributing to one's knowledge of programming through the use of Robotics. The 30-item Likert scale post-survey was designed with this statement in mind and comprised six constructs supported by relevant literature (discussed in *Chapter two: Literature review*), with 5 items per construct¹⁵. To reiterate, the six constructs which represent the factors used in the model are:

- Confidence: student confidence in their ability to learn programming;
- Interest: student interest in programming;
- Motivation: student motivation by the use of robots;
- Belief: student intrinsic belief in solving problems;
- Mathematics: student perception of mathematical influence on programming; and
- Knowledge: student knowledge of programming through the use of Robotics.

In PLS-SEM, constructs are known as latent variables (LV) and items are known as indicators. The model created (Figure 53) follows the principles of a reflective model since, as Garson (2016) explains, in such models “indicators are a representative set of items which all reflect the latent variable they are measuring” (p. 18). This is observed in the model created during the initial stages of the study (Figure 53), where all six LVs offer loadings onto the respective five indicators. Reflective models allow the omitting or dropping of indicators that do not matter while sustaining the meaning of the LV (Garson, 2016). It is important to note that omitting indicators that are not significant would be essential in producing a model that makes meaning and sense.

The minimum sample size estimation method in PLS-SEM follows the 10 times rule (Hair et al., 2016; Hair et al., 2012). The rule applied to the reflective conceptual model in this study would be:

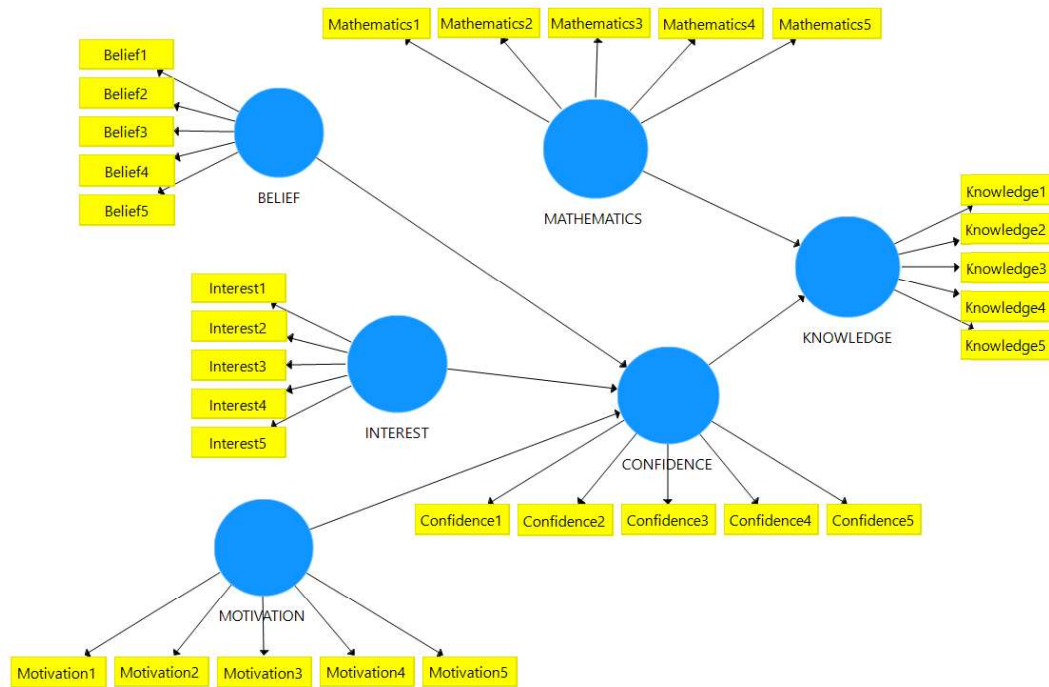
sample size > 10 × *the largest no. of structural paths directed at a particular LV*, in the structural model. Examining the conceptual model (Figure 53 and Figure 54), the largest number of structural paths directed to a particular LV is three (i.e., *Belief*, *Interest* and

¹⁵ The word construct = factors = latent variables

Motivation informing *Confidence*). Therefore, the sample size is in accordance with the ten times rule: 75 (*sample size*) $> 10 \times 3$ (*largest no. of structural paths*) $= 75 > 30$.

Figure 53

The initial form of the model



Note. The initial form of the conceptual model created prior to the analysis stages.

Figure 53 depicts the model in early stages prior to running any analysis, showing all LVs and indicators. *Knowledge* is referred to as an endogenous LV. An endogenous latent variable is an LV informed by at least one other LV (Garson, 2016; Hair et al., 2016). In other words, graphically, the model has at least one incoming arrow from another LV. LVs' *Belief*, *Interest*, *Motivation*, and *Mathematics* are referred to as exogenous variables since any other LV does not inform them. Hence, they do not have any incoming arrow/s from any other LV/s.

As discussed in an earlier in *Chapter two: Literature review, sub chapter 2.5.4 The influence of Mathematics* there are mixed views regarding the effect of *Mathematics* on coding *Knowledge*. An individuals' *Confidence* is informed by their *Belief*, *Interest*, and *Motivation* (Douglas et al., 2019; Blotnick et al., 2018; Firat et al., 2018; LaForce et al., 2017). While the latter factors may not directly affect *Knowledge*, they have the potential to affect it indirectly through *Confidence* (Fischer & Sliwka, 2018; Sadler, 2013). Hence *Confidence* is referred to

as a mediating variable as it is an intervening variable. *Through Robotics, knowledge of coding* is a dependent LV informed by the following possible contributing factors: *Belief, Interest, Motivation, Confidence, and Mathematics*. The study will determine if the following relationships exist through the SEM-PLS model, hence hold true (Table 41).

Table 41

Set of hypotheses based on model

Hypothesis	Relationship
H ₁	BELIEF → CONFIDENCE
H ₂	CONFIDENCE → KNOWLEDGE
H ₃	INTEREST → CONFIDENCE
H ₄	MATHEMATICS → KNOWLEDGE
H ₅	MOTIVATION → CONFIDENCE

The alternative hypothesis (H_a) for each relationship found in Table 41 is that there is no relationship between the LVs in question.

6.4.1.1 Measurement model

The study presents the measurement model into three parts: indicator reliability, convergent reliability and discriminant validity. *Indicator reliability* examines the measure and validity of the reflective indicator loadings, Cronbach’s Alpha (CA) and rho_A (ρ_A). The *convergent reliability* examines the Average Variance Extracted (AVE) and internal consistency. The evaluation of *discriminant validity* considers the cross-loading criteria, Fornell & Larker criteria and Heterotrait-Monotrait ratio of correlations (HTMT).

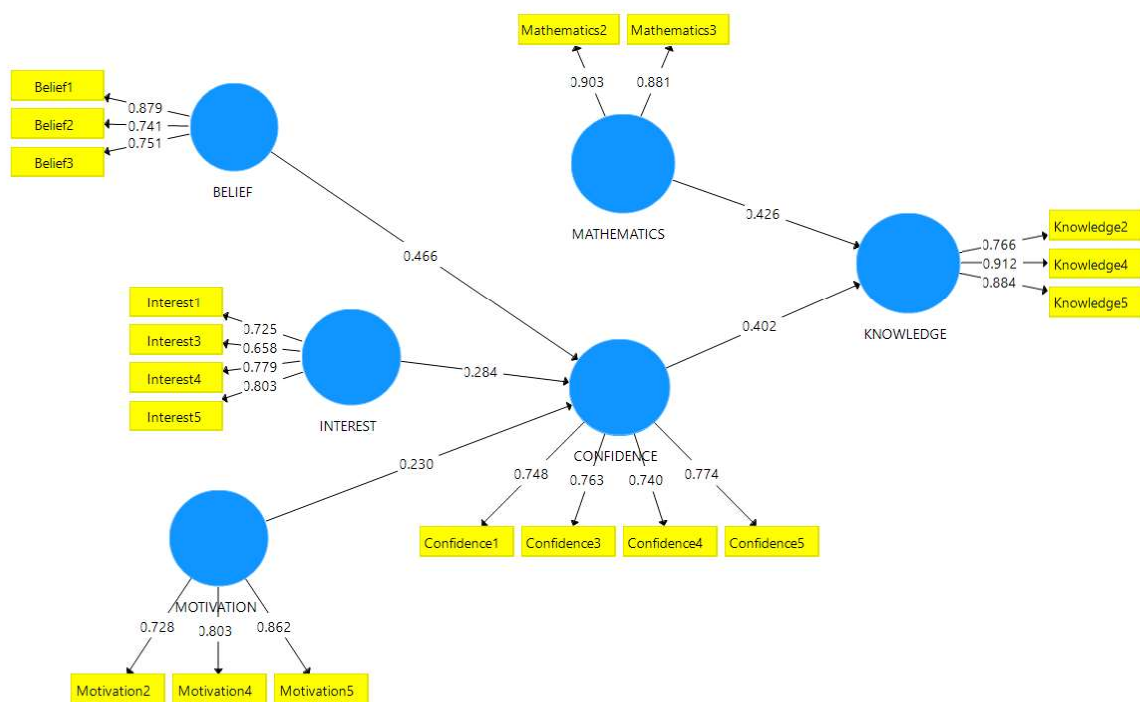
Indicator reliability

The PLS algorithm was executed on the model (Figure 53) with an initial analysis of 300 iterations and later with a maximum of 500 iterations resulting in the outer indicator loadings (Figure 54). The loadings can be considered a form of item reliability in reflective models; as Garson (2016) describes, “the closer the loadings are to 1.0, the more reliable that latent variable” (p. 60). He goes on to express that path loadings for such a model should be > 0.7 (Garson, 2016), while Hulland (1999) recommends that reflective indicator loadings > 0.5 show that the indicator is a good measurement of the LV. A more refined and applied criteria

to this study is by Hair et al. (2014), who propose that an indicator loading in the range of 0.40 to 0.70 may be dropped only if it improves Composite Reliability (CR). Therefore, outer loadings of 0.7 or higher are considered highly approved, while 0.5 is deemed acceptable (Memon & Rahman, 2014). Figure 54 depicts the outer loadings of indicators that meet the criteria of > 0.7 or retained because they do not improve CR when discarded.

Figure 54

Model showing valid path loadings



Note. The weighting scheme was initially set on a maximum of 300 iterations and later set on a maximum of 500 iterations.

The CA value evaluates the reliability of the set of indicator items. Therefore, measures the extent to which all the LVs in the model are positively related to each other. As mentioned earlier, an alpha value greater than 0.7 ($\alpha > 0.7$) is acceptable (Nunnally, 1978).

Using rho_A (ρ_A) is a more consistent measure of reliability than Cronbach's Alpha. As Dijkstra and Henseler (2015) describe, ρ_A is measured the same as Cronbach's Alpha and has a better reliability measure than Cronbach's Alpha in SEM. Since ρ_A is based on the loadings rather than the correlations observed between the variables.

Convergent reliability

One of the measures of convergent reliability is Average Variance Extracted (AVE). AVE is comparable to the proportion of variance explained in factor analysis. AVE values are between 0 and 1, and an AVE > 0.5 is desired (Bagozzi & Yi, 1988; Fornell & Larcker, 1981).

Internal Consistency is assessed by validating the Composite Reliability (CR), which measures the reliability of the indicators where values are between 0 and 1. CR values greater than 0.7 (CR > 0.7) proves to show adequate consistency (Gefen et al., 2000). Chin (1998) proposed that internal consistency is measured through the CR, known as Dillon-Goldstein's rho or Jöreskog's rho. CR is a favoured alternative to CA as a test of convergent reliability in a reflective model Garson, 2016). As Demo et al. (2012) propound the view that CR offers a more reliable measure than CA in SEM. The findings support Dijkstra-Henseler (2015) when examining (Table 42) the cut-off value of ρA (Rho A) > 0.7 ensures Composite Reliability (CR > 0.7).

Table 42*Measurement model including LV (construct) reliability and validity*

	Items	Loadings	AVE	CR	CA	Rho A
BELIEF	Belief1	0.879	0.628	0.834	0.703	0.731
	Belief2	0.741				
	Belief3	0.751				
INTEREST	Interest1	0.725	0.553	0.831	0.729	0.725
	Interest3	0.659 [†]				
	Interest4	0.779				
	Interest5	0.803				
MOTIVATION	Motivation2	0.728	0.639	0.841	0.716	0.730
	Motivation4	0.803				
	Motivation5	0.862				
MATHEMATICS	Mathematics2	0.903	0.795	0.886	0.744	0.751
	Mathematics3	0.881				
CONFIDENCE	Confidence1	0.748	0.572	0.842	0.751	0.750
	Confidence3	0.763				
	Confidence4	0.740				
	Confidence5	0.774				
KNOWLEDGE	Knowledge2	0.766	0.733	0.891	0.821	0.867
	Knowledge4	0.912				
	Knowledge5	0.884				

Note.

- Indicator items removed: Belief 4, Belief 5, Interest 2, Motivation 1, Motivation 3, Mathematics 1, Mathematics 4, Mathematics 5, Confidence 2, Knowledge 1, Knowledge 3.
- All item loadings > 0.5; indicate indicator reliability (Hair et al., 2014; Hulland, 1999).
- All Average Variance Extracted (AVE) > 0.5 indicating convergent reliability (Bagozzi and Yi, 1988).
- All Composite Reliability (CR) > 0.7; indicate internal consistency (Gefen et al., 2000).
- All Cronbach's Alpha (CA) > 0.7 (Nunnally, 1978).
- All rho_A (ρA) > 0.7 Dijkstra-Henseler (2015).
- [†]Retained because it did not change the CR when dropped/removed.

Discriminant validity

Discriminant validity is the extent to which a LV is truly distant from other LVs, which implies that the LV is unique. Discriminant validity is also known as vertical collinearity, which validates the subjective independence of every indicator on the LV. The cross loadings criterion helps reduce the presence of multicollinearity amongst the LVs by denoting that the AVE of a LV should be higher than the square correlations between the LV and all other variables (Chin, 2010; Chin, 1998; Fornell & Larcker, 1981). In other words, the loadings of an indicator on its assigned LV should be a higher value than its loadings on all other associated LV values (refer to Table 43- *values in bold are greater than horizontal associated values*).

Table 43*Discriminant Validity- indicator item cross loading*

	LATENT VARIABLES					
	BELIEF	CONFIDENCE	INTEREST	KNOWLEDGE	MATHEMATICS	MOTIVATION
Belief1	0.881	0.707	0.505	0.589	0.879	0.624
Belief2	0.741	0.574	0.371	0.566	0.513	0.493
Belief3	0.751	0.488	0.370	0.344	0.546	0.249
Confidence1	0.524	0.748	0.684	0.516	0.444	0.603
Confidence3	0.604	0.763	0.412	0.557	0.580	0.402
Confidence4	0.664	0.740	0.416	0.502	0.574	0.548
Confidence5	0.492	0.774	0.581	0.502	0.426	0.590
Interest1	0.342	0.463	0.725	0.466	0.332	0.579
Interest3	0.599	0.594	0.658	0.473	0.570	0.478
Interest4	0.211	0.470	0.779	0.296	0.131	0.440
Interest5	0.361	0.499	0.803	0.408	0.250	0.590
Knowledge2	0.288	0.469	0.472	0.766	0.346	0.591
Knowledge4	0.652	0.588	0.489	0.912	0.653	0.528
Knowledge5	0.630	0.674	0.492	0.884	0.705	0.546
Mathematics2	0.628	0.496	0.314	0.647	0.903	0.330
Mathematics3	0.879	0.707	0.505	0.589	0.881	0.624
Motivation2	0.414	0.547	0.488	0.550	0.530	0.728
Motivation4	0.426	0.492	0.526	0.458	0.255	0.803
Motivation5	0.576	0.646	0.660	0.512	0.459	0.862

Note. All item loadings on its assigned latent variable (bold values) are higher than its loadings on all other latent variables. In other words, all the indicator's outer loading on the associated LV is greater than all its loadings on other LVs; therefore, cross loadings criterion is fulfilled (Chin, 2010; Chin, 1998; Fornell & Larcker, 1981).

The Fornell and Larcker criterion is that the AVE of an LV should be higher than the squared correlations between the LV and all other variables (Chin, 2010; Chin 1998; Fornell & Larcker, 1981). The software processes this calculation in which the diagonals are the square root of the AVE of the LVs (Table 44, next page) and should be the highest in any associated column or row.

Table 44*Discriminant Validity- Fornell and Larcker criterion*

	BELIEF	CONFIDENCE	INTEREST	KNOWLEDGE	MATHEMATICS	MOTIVATION
BELIEF	0.793					
CONFIDENCE	0.755	0.756				
INTEREST	0.531	0.694	0.743			
KNOWLEDGE	0.643	0.686	0.562	0.856		
MATHEMATICS	0.838	0.668	0.454	0.694	0.892	
MOTIVATION	0.598	0.710	0.705	0.635	0.527	0.800

Note. Fornell and Larcker criterion has been met in all instances since all values (in bold) are greater than their associated values (the values in bold forming the diagonal are the highest compared to associated values across and below).

Heterotrait-Monotrait ratio of correlations, bootstrapping and normality

The Heterotrait-Monotrait ratio of correlations (HTMT) was developed to address the insensitivity of the Fornell & Larcker and cross loadings criterion. To determine if discriminant validity is achieved using HTMT and bootstrapping is required. Although PLS-SEM assumes that the data is not normal, unlike CBS-SEM (Covariance Based Structural-Equation Modelling), this study will still confirm that the data is not normal before proceeding to bootstrap. Bootstrapping is a nonparametric procedure that allows testing statistical significance through a method that uses random sampling, mimicking the sampling process-resampling method. It can be applied to regression models giving insight into how variable the model parameters are (Godwin, 2021). Bootstrapping estimates the spread, shape and bias of the sampling distribution of the population. The observed sample is treated as a representation of the population. Bootstrapping creates a large, pre-specified number of samples. Every time sampling happens during bootstrap, the same number of cases as the original sample will be analysed; thus, N bootstrap → n samples Chin (1998). The cut-off values for univariate skewness are ±1 and kurtosis are ±7 (DeCarlo, 1997; Mardia, 1970). The cut-off value for Mardia's multivariate skewness is ±1 and kurtosis are ±20 (Mardia, 1974).

Table 45*Assessing Normality- Univariate and multivariate skewness and kurtosis*

	<i>Skewness</i>	<i>SE_skew</i>	<i>Kurtosis</i>	<i>SE_kurt</i>
BELIEF	1.0581	0.2774	1.5678	0.5482
CONFIDENCE	0.6666	0.2774	0.5753	0.5482
INTEREST	1.4769	0.2774	2.7626	0.5482
KNOWLEDGE	0.3320	0.2774	-0.1976	0.5482
MATHEMATICS	0.7718	0.2774	1.0837	0.5482
MOTIVATION	1.3041	0.2774	2.1185	0.5482

Note.

- a. Sample size: 75
- b. Number of variables: 6
- c. SE_skew= Standard error skewness
- d. SE_kurt= Standard error kurtosis

Table 46*Assessing Normality- Mardia's multivariate skewness and kurtosis*

	<i>b</i>	<i>z</i>	<i>p value</i>
Skewness	11.96683	149.58536	1.84E-10
Kurtosis	52.12194	1.821659	6.85E-02

Note. *b* = Mardia's coefficient for skewness and kurtosis.

The findings from running the normality tests (Table 45) show that the kurtosis values are within bounds; however, the skewness for LVs: Belief, Interest and Motivation are out of bounds. Further examining Mardia's coefficient (Table 46), the values of skewness and kurtosis are out of the respective bounds (Skewness $b > 1$; Kurtosis $b > 20$). The normality assessment deduces that the data does not follow the normal distribution; hence bootstrapping can be applied (Frost, n.d.).

As a result of bootstrapping, the discriminant validity based on the Heterotrait-Monotrait ratio of correlations (HTMT) can be examined. HTMT estimates the correlation between the LV based on the average Heterotrait-Heteromethod correlation (Henseler et al., 2015). HTMT is assessed by examining the Confidence Interval- Upper Limit (CI-UL) and is expected to be less than 0.90 (at the 95% Confidence Interval). Therefore, a CI-UL value higher than 0.9 indicates there is a lack of discriminant validity. Ringle (2015) purports that discriminant

validity has not been established if the CI-UL value is above 1. As a statistical test- testing of the null hypothesis ($H_0: HTMT < 1$) versus the alternative ($H_a: HTMT \geq 1$) (Henseler et al., 2015), $HTMT_{95\% \text{ Confidence Interval}}$ contains the value one or above \rightarrow no discriminant validity. Initial assessment, using a smaller number of complete bootstrapping with subsample sizes of 500, 1000 and 3000 with parallel processing, was carried out. For final result preparation, a large subsample size of 5000 was used during bootstrapping.

Table 47

Discriminant Validity- HTMT

	ORIGINAL SAMPLE (O)	SAMPLE MEAN (M)	CI LL 5.00%	CI UL 95.00%
BELIEF \rightarrow CONFIDENCE	0.466	0.470	0.353	0.592
CONFIDENCE \rightarrow KNOWLEDGE	0.402	0.397	0.216	0.576
INTEREST \rightarrow CONFIDENCE	0.284	0.292	0.129	0.454
MATHEMATICS \rightarrow KNOWLEDGE	0.426	0.432	0.261	0.600
MOTIVATION \rightarrow CONFIDENCE	0.230	0.216	0.067	0.362

Note.

- a. Complete bootstrapping performed.
- b. Set at 5000 subsamples.
- c. Parallel processing.
- d. H_0 holds since all CI-UL < 0.9 (Henseler et al., 2015).

Drawing from the findings (Table 47), the null hypothesis has failed to be rejected; hence discriminant validity has been established, CI-UL < 0.9 .

6.4.1.2 Structural model

The structural model examines horizontal collinearity. Therefore, assessing the structural model results enables determining the model's capability to predict one or more target LV/s (construct/s). The study presents the structural model in six parts: collinearity issues, path coefficients, the significance of the relationships, level of R^2 , effect size (f^2) and predictive relevance (Q^2).

Collinearity issues

Collinearity arises when two indicators are highly correlated. Collinearity among LVs is assessed through the Variance Inflated Factor (VIF). A VIF value of greater than or equal to five ($VIF \geq 5$) indicates a potential collinearity problem (Hair et al., 2011). While a more stringent guideline purported by Diamantopoulos and Siguaw (2006) is that a $VIF \geq 3$ indicates potential collinearity problems. Table 48 shows the evaluation of the VIF values, where each set of predictor LV is assessed separately for each subpart of the structural model.

Table 48

Structural model- Variance Inflated Factor

	BELIEF	CONFIDENCE	INTEREST	KNOWLEDGE	MATHEMATICS	MOTIVATION
BELIEF		1.618				
CONFIDENCE				1.807		
INTEREST		2.066				
KNOWLEDGE						
MATHEMATICS				1.807		
MOTIVATION		2.310				

Note. All VIF values are within the prescribed tolerance ranges ($VIF \geq 5$, Hair et al., 2011; $VIF \geq 3$, Diamantopoulos & Siguaw, 2006).

All VIF is within the prescribed guidelines (Hair et al., 2011; Diamantopoulos & Siguaw, 2006), meaning there is no strong indication of collinearity issues.

Path coefficients

Path coefficients are the coefficient linking LVs in the structural model. The coefficient represents the hypothesised relationship of the relationship strength hence the significance of relevance of the relationships. Accordingly, the primary way to compare the strength of relationships is to examine the path coefficients. Thereby, the path coefficients indicate to which extent an independent variable affects a dependent variable (through bootstrapping, the significance of the relationships are examined- *looked at in the next subheading*). Path coefficients vary between -1 and +1, coefficient values closer to +1 indicate a strong positive relationship (and vice versa for negative values). Higher values denote stronger (predictive) relationships between the LVs. The closer the value is to 0 signifies a weak relationship and is not statistically significant.

There are three types of effects. Firstly, *direct effect*- a relationship linking two LVs with a single arrow between the two. Secondly, an *indirect effect*- a sequence of relationships with at least one intervening LV involved, and thirdly *total effect*- the sum of the direct effect and all indirect effects linking two LVs. It is important to note that the conceptual model created and tested in this study (Figure 53 and Figure 54) is designed based on a direct effect relationship.

Table 49

Path coefficients

		DEPENDENT VARIABLES					
		BELIEF	CONFIDENCE	INTEREST	KNOWLEDGE	MATHEMATICS	MOTIVATION
INDEPENDENT VARIABLES	BELIEF		0.466				
	CONFIDENCE				0.402		
	INTEREST		0.284				
	KNOWLEDGE						
	MATHEMATICS				0.426		
	MOTIVATION		0.230				

The path coefficient values (Table 49) do not exhibit high values but indicate all positive relationships between IVs and DVs. This can be depicted visually in Figure 55, showing all positive relationships exist.

Figure 55

Visual representation of path coefficients



Significance of the relationships- t values and p values

The significance of the relationships is further assessed by means of bootstrapping (i.e., examining whether the effect of a certain IV on a certain DV is significant). Bootstrapping analysis evaluates the direct effects of all the hypothesised relationships represented by statistical testing of the hypothesis (refer to Table 41 earlier). Determining whether a coefficient is significant depends on its standard error (SE) obtained by bootstrapping that computes the empirical *t* values and *p* values for all structural paths. When an empirical *t* value is larger than the critical value, it can be concluded that the coefficient is statistically significant at a certain error probability. Commonly used critical values for a two-tailed test are 1.65 at a significant level of 10%, 1.96 at a significant level of 5% and 2.58 at a significant level of 1%. The confidence level is equivalent to 1; in humanities, it is typical to adapt a significance level of 0.05, which corresponds to a confidence level of 95%.

If $t_{0.05} > 1.96$ (for a 2-tailed test, critical value = 1.96), the hypothesis is supported (Peng & Lai, 2012). Hair et al. (2017) suggest assessing beta values (β) and the corresponding t-values through a bootstrapping procedure with a resample of 5000. The individual path coefficient is interpreted as the standardised coefficient in an ordinary least squares (OLS) regression. A one unit change of exogenous construct changes the endogenous construct by the size of the path coefficient when everything else remains constant (Hair et al., 2011).

Table 50

Direct relationships for Hypothesis testing

Hypothesis	Relationship	Std Beta	Std Error	t value	p value	Decision	95%CI LL	95%CI UL
H1	BELIEF → CONFIDENCE	0.470	0.073	6.352**	0.000	Supported	0.353	0.592
H2	CONFIDENCE → KNOWLEDGE	0.397	0.109	3.689**	0.000	Supported	0.216	0.576
H3	INTEREST → CONFIDENCE	0.292	0.100	2.836**	0.005	Supported	0.129	0.454
H4	MATHEMATICS → KNOWLEDGE	0.432	0.104	4.103**	0.000	Supported	0.261	0.600
H5	MOTIVATION → CONFIDENCE	0.216	0.089	2.578*	0.010	Supported	0.067	0.362

Note.

- a. ** $p < 0.01$, * $p < 0.05$;
- b. All relationships supported.

The *p* values are used to assess the significance levels. When assuming a significance level at 5%, the *p* value must be smaller than 0.05 to conclude that the relationship under consideration is significant. It is noted that most *p* values meet the condition when assuming a significance

level of 1%, where the p value is smaller than 0.01, which concludes that the relationship under consideration is significant not only at a 5% level but also at a 1% level (Table 50).

Level of R²

R square (R^2) is the coefficient of determination, which measures the proportion of variance in a latent endogenous variable explained by the other exogenous expressed as a percentage (Chin, 1998). Hence, R^2 measures the model’s predictive accuracy, representing the amount of variance in the endogenous constructs explained by all exogenous constructs linked to it. R^2 values range between 0 and 1, with higher values indicating higher levels of predictive accuracy.

It is considered that values of $R^2 \approx 0.25$: weak, $R^2 \approx 0.50$: moderate and $R^2 \approx 0.75$: substantial (Hair et al., 2011; Henseler et al., 2009). Chin (1998) articulates that R^2 values of 0.67, 0.33 and 0.19 as substantial, moderate and weak. The adjusted R^2 values are interpreted similarly to the R^2 square values, as the adjusted R^2 controls for model complexity when comparing different model set-ups (Karch & van Ravenzwaaij, 2020; Harel, 2009).

Table 51

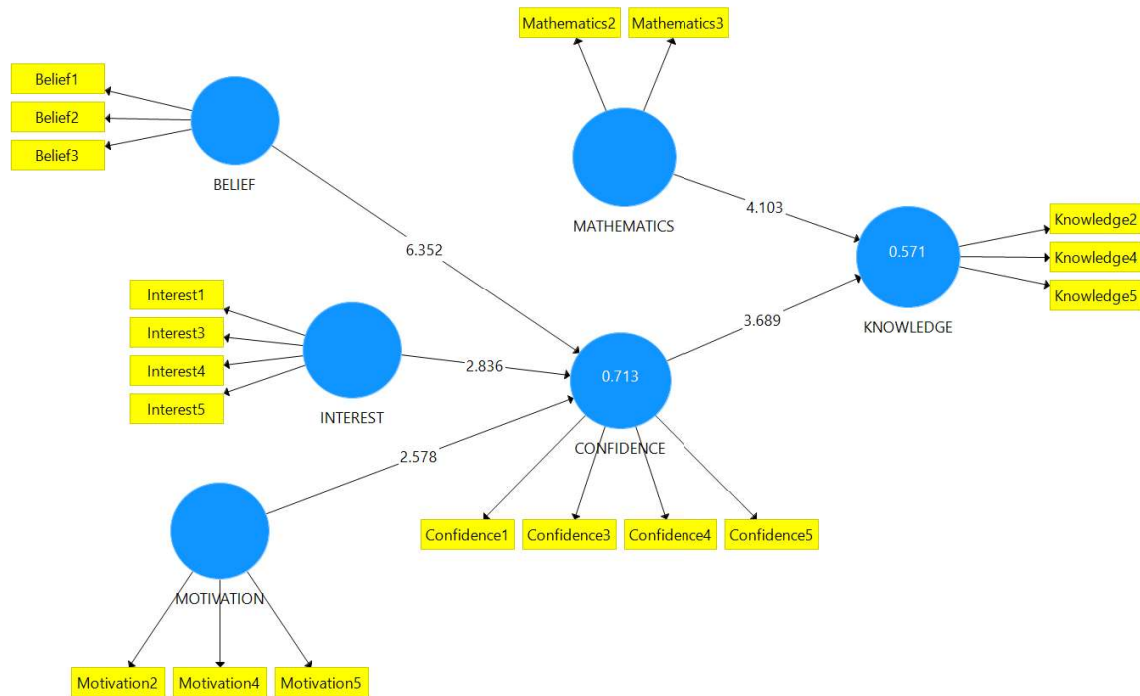
R Square (R^2) values

	R SQUARE	R SQUARE ADJUSTED	OUTCOME
CONFIDENCE	0.713	0.701	Substantial- Moderate
KNOWLEDGE	0.571	0.559	Moderate- Substantial

Note. R^2 and R^2 adjusted values for CONFIDENCE are substantial and KNOWLEDGE is moderate- $R^2 \approx 0.19$ - 0.25: weak, $R^2 \approx 0.33$ - 0.50: moderate and $R^2 \approx 0.67$ - 0.75: substantial (Hair et al., 2011; Henseler et al., 2009; Chin, 1998).

Figure 56

R Square values (R^2) and inner model depicting t values.



Note.

- LVs (constructs) showing R square values and the inner model showing t values.
- Model result of bootstrapping subsample 5000.

The R square values represent the amount of variance by the endogenous LVs (constructs) explained by all exogenous LVs linked to it. As depicted in Table 51 and Figure 56, the endogenous LVs *Confidence* can be found to have a *Substantial to Moderate* outcome while *Knowledge* can be expressed to have a *Moderate to Substantial* outcome (Hair et al., 2011; Henseler et al., 2009; Chin, 1998). Visual inspection of Figure 56 shows that the endogenous LVs *Mathematics* and *Confidence* are dependent variables (DV) informed by the independent variables (IV). Note that while *Confidence* is regarded as endogenous, it is also a mediating variable that acts as a DV that informs the IV *Knowledge*. Further calculation of the effect strength (f^2) for each IV allows the assessment to which extent the IV contributes to the explanation of the DV.

Effect size (f^2)

The assessment of the effect size allows the observance of the effect of each exogenous LV on the endogenous LV. In doing so, the effect size assesses how strongly one exogenous LV contributes to explaining a certain endogenous LV in terms of R^2 . The effect size formula is as follows: $f^2 = \frac{R^2_{included} - R^2_{excluded}}{1 - R^2_{included}}$, where $R^2_{included}$ and $R^2_{excluded}$ where the R^2 values of the endogenous latent variable (LV) when a selected exogenous LV is included or excluded from the model. The change in the R^2 values is calculated (Table 52) by estimating the PLS path model twice, that is, once with the exogenous LV included (yielding $R^2_{included}$) and the second time with the exogenous LV excluded (yielding $R^2_{excluded}$). Therefore, the effect size (f^2) evaluation determines whether the omitted LV has a substantive impact on the endogenous construct, also known as the effect size of the exogenous LV on the model. The assessment of the effect size follows Cohen's (1988) guidelines which are 0.02, 0.15 and 0.35 for small, medium and large effects, respectively. Effect size values less than 0.02 indicate that there is no effect.

Table 52

Effect Size (f square)

PREDICTOR	ENDOGENOUS	R-SQ INCLUDED	R-SQ EXCLUDED	EFFECT SIZE (F^2)	OUTCOME
BELIEF	CONFIDENCE	0.713	0.590	0.429	<i>strong</i>
CONFIDENCE	KNOWLEDGE	0.571	0.494	0.179	<i>moderate</i>
INTEREST	CONFIDENCE	0.713	0.675	0.132	<i>weak</i>
MATHEMATICS	KNOWLEDGE	0.571	0.467	0.242	<i>moderate</i>
MOTIVATION	CONFIDENCE	0.713	0.690	0.080	<i>weak</i>

Note.

- All effect sizes are valid since no $f^2 \leq 0.02$.
- Effect Size impact indicator are according to Cohen (1988); f^2 values: 0.35 (large), 0.15 (medium), and 0.02 (small).

Predictive relevance (Q^2)

In addition to evaluating the magnitude of the R^2 values as a criterion of prediction accuracy, researchers also examine predictive relevance (Q^2), also known as the Stone-Geisser Q^2 value (Geisser, 1974; Stone, 1974). This measure is an indicator of the model's predictive power or predictive relevance. The Q^2 value is obtained through blindfolding procedures for a specified omission distance (D) with values between 5 and 10. Q^2 values larger than zero suggest that

the model has predictive relevance for a certain endogenous LV (Henseler et al., 2009; Tenenhaus et al., 2005; Chin, 1988). As remarked by Garson (2016), “values of Q^2 greater than 0 means that the PLS-SEM model is predictive of the given endogenous variable under scrutiny” (p.117). In contrast, values of 0 and below indicate a lack of predictive relevance. Cohen (1988) prescribes $0.02 \leq Q^2 < 0.15$: *small* effect size, $0.15 \leq Q^2 < 0.35$: *medium* effect size and $Q^2 \geq 0.35$: *high* effect size. Predictive relevance is assessed through the findings of the construct cross-validation redundancy, which addresses model fit (Garson, 2016).

Table 53

Construct Cross validated Redundancy

	SSO	SSE	$Q^2 (=1-SSE/SSO)$
BELIEF	225.000	225.000	
CONFIDENCE	300.000	189.747	0.368
INTEREST	300.000	300.000	
KNOWLEDGE	225.000	137.841	0.387
MATHEMATICS	150.000	150.000	
MOTIVATION	225.000	225.000	

Note.

- a. Blindfolding omission distance $D=7$.
- b. SSO represents the mean value prediction.
- c. SSE is the prediction error when using the model prediction.
- d. All $Q^2 > 0$ showing model has predictive relevance (Henseler et al., 2009; Tenenhaus et al., 2005; Chin, 1998).
- e. Predictive Relevance (Q^2) of predictor exogenous latent variables as according to Henseler et al. (2009), Q^2 values: 0.35 (large), 0.15 (medium), and 0.02 (small)
- f. All $Q^2 \geq 0.35$ high effect.

Findings of the Q^2 values (Table 53) indicate that endogenous LVs *confidence* and *knowledge* possess high predictive relevance due to $Q^2 > 0.35$. Further suggests that the LVs that inform *confidence* and *knowledge* make a meaningful contribution in promoting predictive relevance and that the model is meaningful.

6.4.2 Questionnaire test two~ Post-test based on programming

The post-test sorts out to gather responses to ten MCQs to assess students' knowledge of programming concepts they were exposed to during the workshop sessions.

Table 54 shows the percentage responses per option for each question and the common response for each question. The dot above symbol (·) is used to indicate the correct answer. Refer to Appendix W for the questionnaire.

Table 54

Overall summary response from post- test

Question	A	B	C	D	Mode
1.	14.67%	22.67%	10.67%	·52.00%	D
2.	2.67%	1.33%	·96.00%	0.00%	C
3.	2.67%	·74.67%	8.00%	14.67%	B
4.	9.33%	·68.00%	14.67%	8.00%	B
5.	20.00%	·73.33%	1.33%	5.33%	B
6.	12.00%	30.67%	29.33%	·28.00%	B
7.	25.33%	0.00%	·70.67%	4.00%	C
8.	·8.00%	26.67%	18.67%	46.67%	D
9.	2.67%	18.67%	·32.00%	46.67%	D
10.	9.33%	13.33%	·54.67%	22.67%	C

Drawing from the modes shown in Table 54, most students were correct for seven out of the 10 MCQs. Table 54 indicates that the majority of students obtained a mark of eight out of 10 (22.67%). In addition, more than fifty percent of the students (69.33%) achieved a mark in the ranges of five (5 out of 10) and eight (8 out of 10) in the programming test. The findings show that 24.00% of students scored a mark of eighty percent and above (8 out of 10), which includes 1.33% (n=1) achieved nine (9 out of 10), which is the highest mark obtained.

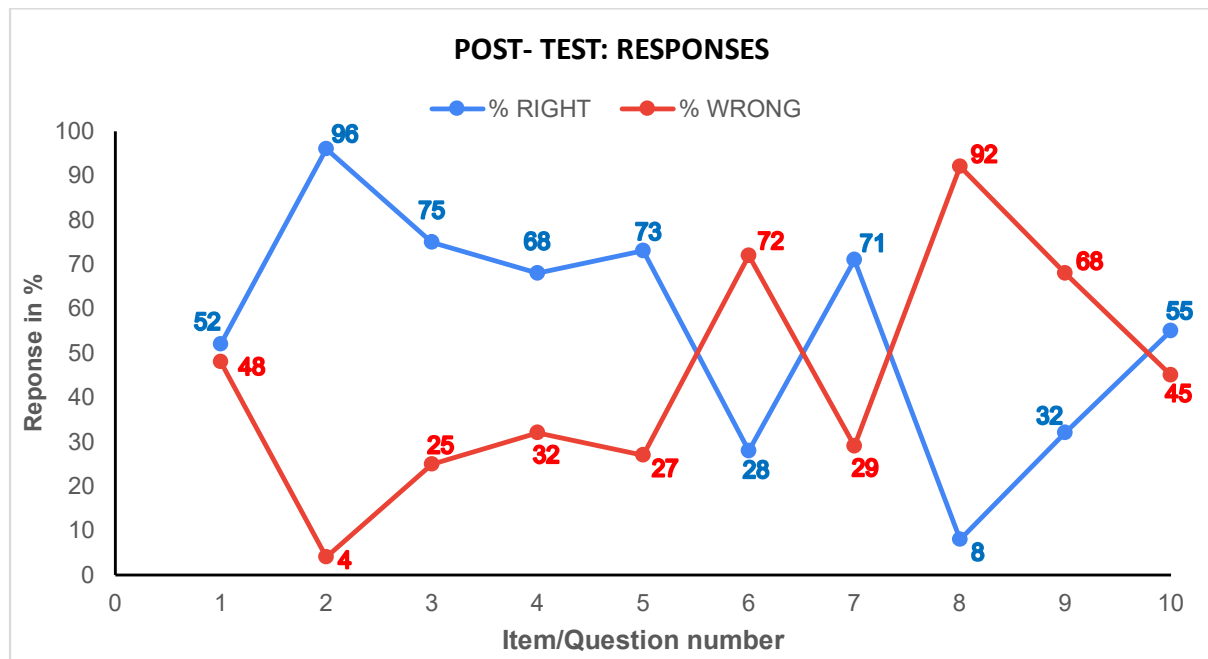
Table 55

Distribution of post-test marks obtained (out of 10)

Mark out of 10	N	%
1	2	2.67%
2	0	0.00%
3	8	10.67%
4	12	16.00%
5	15	20.00%
6	16	21.33%
7	4	5.33%
8	17	22.67%
9	1	1.33%
10	0	0.00%
N	75	100%

Figure 57

Right vs Wrong responses for programming test



Note. A visual representation of correct and incorrect responses based on Table 54; percentages rounded.

Figure 57 depicts extreme differences in the ranges for correct and incorrect answers to questions 2 and 8 (see Appendix X). On closer inspection, most students have provided correct answers to questions 2 and 8.

Table 56

Five-point number summary with mean and SD for programming test (post-test)

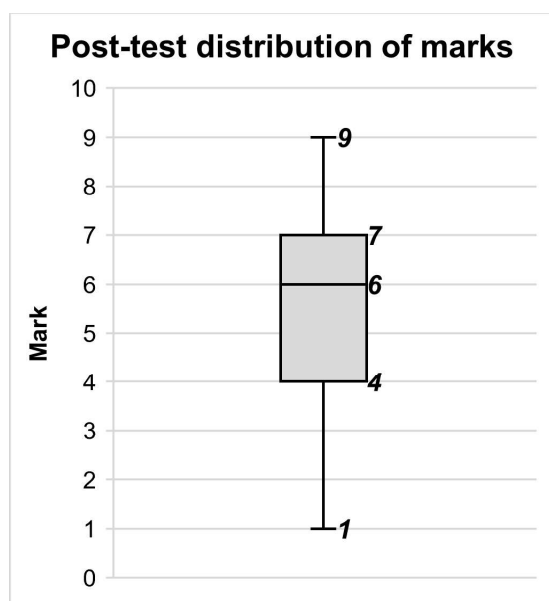
Item	Value
Minimum	1
Quartile 1	4
Quartile 2 (median)	6
Quartile 3	7
Maximum	9
Mean	5.573
Standard deviation	1.847

Note. Total for test: 10 marks.

It is noticed that the majority of students achieved a mark of 8 out of 10 (22.67%, Table 55); this mark is greater than the sample mean $\bar{x} = 5.573$ (Table 56). The box plot (Figure 58) offers a virtual representation of the 5-point summary from Table 56. The highest mark obtained was 9 out of 10, while the lowest was 1 out of 10.

Figure 58

Summary of marks for Programming test



Note. Box plot base on marks obtained from the programming test (post-test).

There were no outliers reported which shows a wide spread of the marks obtained by the students. The mean (\bar{x}) value is very close to the median (quartile 2), which suggests that data is fairly normal (normally distributed).

To assess students' programming abilities through the use of the robotic element, a paired samples t-test analysis was carried out based on the pre-test (Part A) and post-test (programming test).

H_0 : The average difference scored in the post-test will be less than or equal to the pre-test.

H_a : The average difference scored in the post-test will be greater than the pre-test.

Table 57

Paired Samples Statistics

	Mean	N	Std. Deviation	Std. Error Mean
PRE-TEST	47.47	75	19.037	2.198
POST-TEST	55.73	75	18.466	2.132

Table 58

Paired Samples Test (pre-test versus post-test)

Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference		t	df	Sig. (2-tailed)
			Lower	Upper			
8.267	25.961	2.998	14.240	2.294	2.758	74	.007

A paired sample t-Test revealed an increase in students' performance after exposure to robotic and coding workshops $t(74) = 2.758$, $p \leq 0.05$ (Table 58). Further, the post-test mean is greater than the pre-test mean, $55.73 > 47.47$ (Table 57). H_0 is rejected, and H_a is accepted, supporting that the average difference scored in the post-test is greater than the pre-test. The following excerpts support the above:

"I felt very highly motivated to just complete the workshop activity and then jump to the next one. Just for you to see it actually works and what new things I could do further" (P56, Focus Group Interview).

"Each workshop would give you the foundation and get an idea of how this component connects to the breadboard... and then with that, you can then have the freedom to play; that's when the fun comes" (P62, Focus Group Interview).

“The robotic element made it most fun and I feel like it solidifies understanding in some way” (P49, Focus Group Interview).

“I feel like it was much easier for me to understand Python by looking at how and how the microcontroller execute” (P21, Focus Group Interview).

“Microcontroller helped me understand if something is wrong in my code to see if the end product works. I would refer back to the prototype design and then check my code...without the microcontroller, I don't think I would have been able to understand the code and even do the workshop” (P1, Focus Group Interview).

“I would have understood and caught with Python without the microcontroller, but it wouldn't have been so fast, would take me sometime” (P9, Focus Group Interview).

“I always thought it was like overly complicated and I tried watching stuff on YouTube, but now it is understandable. All those diagrams, whatever. But this workshop helped me by giving me a push in the right direction” (P42, Focus Group Interview).

“The micro controller helped me and I enjoyed the workshop so much because the equipment was given... I was able to write programs and see them executed” (P16, Focus Group Interview).

Interestingly beside the influence of robot coding in the process of learning introductory basic Computer Programming concepts participants had a keen interest in the activities involving the flame sensor and traffic light simulation:

“I feel my favourite activity was the LED; it was interesting to see the lights come on like a traffic light” (P17, Focus Group Interview).

“My favourite activity was programming traffic lights so because I always wonder how the traffic lights went so now I have a basic idea of how” (P16, Focus Group Interview).

“I also liked using the fire sensor activities” (P15, Focus Group Interview).

“Like the flame one... also liked when we started to use more than two components” (P42, Focus Group Interview).

Some participants have expressed growing curiosity in robot and further exploration of coding:

“So actually, after the course after the workshop, I actually started to try and modify and build and code on my own. Actually, I am now working on a prototype with the buzzer” (P62, Focus Group Interview).

“After doing the workshop, I saw the need to go back to the drawing board and actually study Python in depth and because I would like to develop/build things. And I would like to program solutions to whatever problems we're facing in the future” (P47, Focus Group Interview).

6.6 Conclusion

This chapter sought to answer the research questions by analysing the data collected through quantitative software tools thus findings. For coherence purposes, key interview excerpts are intertwined within the quantitative analysis in giving rise to the findings.

Data were collected through a series of workshop sessions: a pre-workshop session, six workshop sessions and a post-workshop session. The pre-workshop session consisted of the pre-survey, pre-test and ART. The six workshop sessions represented the iterative process aligned to Kolb's Experiential Learning Cycle informed by the DBR's prototyping phase (discussed in *Chapter five: Iteration*). Lastly, the post-workshop session consisted of the post-survey and pre-test.

Through the pre-workshop session, students' perceptions of coding Robotics were analysed. In addition, one's rational ability versus programming knowledge was also tested by examining mathematical abilities and the ART results. Further, a one sample t-test was carried out by using the APS as the test value. Through the post-workshop session, a well-defined reflective model was conceptualised and tested. The study utilised the bootstrapping algorithm to analyse the significance of the relationships in the model. This ultimately allowed for examining the contribution of Robotics to the understanding of programming. A paired sample test was carried out between the pre-test and post-test, which revealed a significant increase in coding knowledge through Robotics.

Chapter seven: Discussion and conclusion

“First, solve the problem. Then, write the code.” ~John Johnson

7.1 Introduction

Chapter six provided an in-depth analysis of the data collected, the findings and discussion. This chapter¹⁶ provides a holistic summary and discussion of findings from *Chapters five and six*, thereafter presenting conclusions from the research. The Fourth Industrial Revolution (4IR) has spurred an educational revolution. Propelled by COVID-19 and lockdown, it has changed the way teaching and learning is carried out. In response to the national and international initiatives promoting coding through government policies, curriculum changes, online platforms and gamified environments, this study set out to explore Robotics in the learning of programming. All participants in this study met the following quota to form the sample: registered in a computer course/module; and having had no prior exposure to Computer Programming in the duration of the degree. The use of DBR grounded in the pragmatic paradigm supported mixed-methods data collection. The data collection took the form of a pre-workshop session, six workshop sessions and a post-workshop session, all of which were online due to the pandemic. The pre-workshop session comprised a pre-survey and problem solving/logic test (namely Part A: Pre-test based on Computational Thinking; and Part B: Abstract Reasoning Test). The six workshop sessions, each having three activities, formed the iteration/cycles of the prototyping phase in terms of DBR, guided by second-generation Activity Theory. Kolb’s four experiential learning stages influenced the prototyping phase. Each activity in the workshop was created to accomplish a concrete experience, reflective observation, abstract conceptualisation and active experimentation. The workshops required building, coding and testing prototypes. Coding of the robot (*artefact*) was performed in a text-based environment using the Python programming language to introduce basic coding concepts to the *subject*. The coding concepts covered include data types, variables, arguments, iteration, conditions and calculations. The post-workshop session consisted of a post-survey and post-test on programming.

¹⁶ For easy reading reference to Kolb’s stages of Experiential Learning is underlined, the second generation Activity Theory is in *italics* and PLS-SEM is in **bold**.

7.2 Discussion

The research emerged from a growing interest in coding and Robotics in the context of the 4IR industrial revolution. Hence the purpose was to explore the use of Robotics to introduce the basics of programming using a text-based environment rather than a block-based environment (the *rules*). Second-generation Activity Theory was found to be ideal for this exploration as the model contains crucial elements/actors that contribute to the *outcome*. In accordance with second-generation Activity Theory, the *rules* represented a change in the conventional way of introducing coding to the novice programmer. Hence a shift from conventional block-based coding to text-based coding using an *artefact* represented by the robot. The interaction between the *subject* and *artefact* through the code promotes (implication of the *rules*) Human Computer Interaction (HCI) which is supported by Activity Theory.

The *division of labour* represented the way the work is divided; and the *community* took account of the social context of second-generation Activity Theory, leading the *subject* to the *object*, resulting in the *outcome*. While all elements of Activity Theory played a role in attaining the *outcome*, the main contributor in this study was the *artefact* (i.e., the prototype could successfully complete the mission/s set out in each of the activities. Successful completion of each mission indicates achievement of the *outcome*, where the *outcome* represents the inherent knowledge of coding. The *division of labour* is the interaction with the robot kit (the build-code-test of prototypes), which necessitate that the *subject* (student) becomes acquainted with the coding. The qualitative findings from the interview were merged with the quantitative findings, as the quantitative component was dominant over the qualitative component in this research design. The qualitative findings supported the quantitative findings. In addressing the need for coherence, each research question is answered accordingly, as discussed below:

1. What are students' perceptions of Robotics when learning to program?

Answers for this research question are derived from the pre-survey, post-survey and interviews. Findings revealed that the *subject* (students) valued using the mediating *artefact* (robot) when learning to code in a text-based environment in achieving the *outcome* by the *rules* set out in the activity network. The building of prototypes resulted in the use of robotic elements acting as manipulatives. It was highly engaging, which gave meaning to the code, in contrast to viewing the execution of the code on the screen in a 2D environment. The ideology of build-code-test (prototype building) resonates through all activities in the workshops. The *artefact*

provided an excitement factor to the learning of code. The majority of students had a strong interest in the robot (*artefact*) (Figure 43, Q10). However, their perception of coding was neutral in the sense of not knowing what to expect (Figure 43: Q9, Q8, Q3) while having a strong interest in coding (Figure 43: Q1). This strongly indicates that the robotic element acted as a motivation to the learning of code. In accordance with DBR and pragmatism, the researcher finds a solution to a real problem. Gauging the afterthoughts of participants in the post-survey allowed for the creation and testing of a conceptualised model through PLS-SEM analysis. Key constructs were supported by literature and tested during the analysis. The model met all respective criteria within the measurement model (indicator reliability, convergent reliability and discriminant validity) and structural model (collinearity issues, path coefficients, significance of the relationships, level of R^2 , effect size f^2 and predictive relevance Q^2), thereby deeming the model valid and successful in explaining students' responses. The model depicts that **belief** of coding, **interest** of coding and **motivation** to code encourages an individual's **confidence** to learn code. Besides these factors (constructs), other influences were the *community* in which the *subject* was located and other elements in the activity system, most importantly, the *subject's* exposure to the *artefact* (robot) through *rules*. Hence, these factors significantly influence the subject's enthusiasm and self-awareness of their ability to learn how to code – **knowledge/outcome**. Similar research contains evidence that tasks involving robots have been shown to improve students' engagement, interest, attitude, and motivation (Hadad et al., 2021; Karim et al., 2016; Zhang & Wan, 2020).

2. Does a high rational ability contribute to attaining programming knowledge?

Response to this question was informed by Questionnaire A (Part A-pre-test and Part B-ART), Questionnaire B (post-test) and interviews. The evidence presented when examining students' mathematical background versus the mark obtained in the pre-test indicates that students with a Pure Mathematics background obtained higher marks than others. While some studies concur with this, other research findings have shown otherwise, i.e., that students' mathematical background does not influence their ability to code (Bubica & Boljat, 2015). However, the type of rational exercise that one encounters in Mathematics is necessary for Computer Programming (Saeli et al., 2011). As such, Mathematics exercises trigger the development of reasoning, Computational Thinking and higher-order thinking skills. These cognitive processes, which are mentally engaging, might not be explicit to an individual who learns to code. However, these processes are crucial in a proficient computer programmer. In addition, they are likely to be influenced and regulated by the *community* (i.e., an individual's

surroundings, including family, friends, peers, other computer students, university, social context expectations, etc.) and *division of labour* (communication with people and objects around the subject that they can draw upon in making decisions and sharing the work).

A Pearson correlation was used to determine the relationship between problem-solving and logic data sets, namely Part A: Pre-test based on Computational Thinking and Part B: ART scores. The result showed a significant positive relationship, indicating that the rational and logical power required when answering are similar. The *community* strongly influences an individual's APS score and mathematical background as there may be expectations. Since the type of Mathematics and APS score attained are determining factors in South Africa – determining what one can study and the kind of institution one can attend. Hence to further understand whether a high rational ability contributes to developing programming knowledge, the national average APS acceptance at university and the ART scores were examined. A one-sample t-test proved that there is no significant difference between the ART scores and national average APS. Although the ART mean > APS mean, this indicated that the students who were part of the workshops scored above the average APS and most probably had a genuine interest into coding and Robotics. A high rational ability, measured by Mathematics achievement, ART scores, and the national average APS, were found to have no significant impact in attaining programming knowledge. In contrast, the PLS-SEM developed based on the post-survey data showed that **Mathematics** plays a role. It is important to note that two out of the five indicator items were retained, and that the construct with its indicator items was based on the individual's (*subject*) perspective. Being based on a subject's perspective, the *community* and *division of labour* have a significant influence on the *subject's* self-evaluation of their ability. When coupled with the *artefact* and *rules* in reaching the *object*, the exploration and experience created within the activity network had a positive effect on **motivation, confidence, interest and belief**. This gives reason to negate the argument that a novice needs a high rational ability before developing coding knowledge (note that higher-order thinking skills are unconsciously developed through CT during the development of coding knowledge), but perhaps it is the individual's self-awareness, self-conscious, inherent or intrinsic view of Computer Programming that plays a more important role in attaining coding **knowledge** (*outcome*).

3. How does the use of Robotics contribute to the understanding of programming?

The answer to this question was informed by the six workshop sessions involving KELC, interviews, pre-test and post-test. Kolb's Experiential Learning Cycle integrated into DBR's prototyping phase proved successful. The four stages of KELC were key in designing each workshop. These stages provided an ideal transition for the individual from a concrete experience to reflective observation, to an abstract conceptualisation and finally to active experimentation before returning to the concrete experience, and so on. Each *subject* would have transitioned through six cycles as a result of the six workshop sessions.

These six workshop sessions allowed students to develop their knowledge of Python coding in a text-based environment. Each workshop comprised of three activities that were successfully based on Kolb's Experiential Learning Cycles. Activity 1 provided a concrete experience; Activity 2 offered a reflective observation and abstract conceptualisation and lastly, Activity 3 provided active experimentation.

It was evident from excerpts that introducing Robotics into the introduction of text-based coding enhanced and promoted visual learning and kinesthetic learning, making learning code a fun experience. As the *subject* was not only able to execute/run their code on a computer screen, but be actively involved in the touch, design, build of the prototype and autonomous movement of the *artefact* in real life. This was supported by the first activity in each workshop, creating a concrete experience. Bringing code to life through robot coding added meaning to what was coded and what the code was expected to do before initial planning. As the *subject* completed the first activity and moved to the second activity, they reflected on their experience and gained knowledge from their experience (abstract conceptualisation). The radar graphs shown in Chapter five per workshop activity during the experiential learning cycle depict the easiness and simplicity of archiving each *object*. By the time the subject reaches the third and last activity within each workshop, they can actively experiment and test their coding knowledge developed due to engagement, reflection, and gained experiences as purported by KELC.

The positive influence of Robotics resulting in the *outcome*, knowledge progression, can promote hybrid knowledge content that goes beyond coding and Robotics. It was evident that students had to develop skills and knowledge in basic electrical circuitry, gear motion, sensors and assembly. This was in addition to the problem solving needed in every activity during the

workshop. Therefore, Robotics enhances understanding and incorporates multi-disciplinary content, especially STEM-related subjects. Robotics as a learning tool integrated into the learning process offers a more innovative learning paradigm (Gaudiello & Zibetti, 2016).

The progressive build-up to the introduction of Computer Programming using the Python language was successful. The workshop sessions guided by KELC, which formed the prototyping phase, offered students an opportunity to engage with the robotic element (Arduino kit) and code (Python), developing their confidence and coding ability as they progressed through the workshops. As the *subject* became comfortable coding in text-base using the *artefact*, activities for workshops 4, 5 and 6 were rated easier as compared with workshops 1, 2 and 3.

Each experiential cycle, referred to as workshop sessions, offered a progressive development of basic coding concepts. The completion of each cycle using the robot as the *artefact* resulted in the PLS-SEM showing that **confidence** played a crucial role in learning to code. This could be a result of the student reaching the **knowledge** through the aid of the robot by transposing through KELC, thereby building their confidence as they cycled through.

Many teachers/instructors find themselves in a dilemma when selecting the first programming language to use when introducing coding concepts to students (Aleksić & Ivanović, 2016). The From the literature review (*Chapter two: Literature review, sub-chapter 2.7 Programming languages*), Python was favoured as an introductory language. Python is less verbose and less syntax laden, and thus more easily understood by students. For example, Xinogalos et al. (2018) states that introducing programming through Python is ideal because the development of programming skills and the algorithmic style of thinking is natural.

A paired sample t-test based was carried out on the pre-test and post-test results, which focused on coding. This revealed an increase in students' performance after exposure to robotic and coding workshops. Undoubtedly, being able to see in reality what the code does upon execution makes coding meaningful, thus registering in long-term memory, since the coding concept is now associated with a particular event or experience.

7.3 Closing remarks

Robotics was used to introduce basic Computer Programming in a text-based environment rather than being limited to a computer screen with text-based coding or block-based coding. Hence, changing the conventional method of introduction to coding and introducing a new norm (*rule*) – coding Robotics to introduce programming. While block-based programming languages make coding accessible to the novice (Maloney et al., 2004), the ultimate environment for coding is a text-based programming language. The results in this study show that students participating in an authentic learning experience with text-based code, with the aid of the robotic element, have proved to be successful. The robotic element simplified the learning process of how to code in a text-based environment. Hence, introducing code through physical manipulatives such as Robotics eradicates the abstract nature of coding.

The target participants were students registered in a computer course at tertiary level without prior exposure to coding. This ensured that students had an absence of Computer Programming in early years, as there is little to no foundation and awareness of coding in pre-tertiary years (schooling years). Hence, early and appropriate exposure of Computer Programming should be incorporated in early schooling in South Africa. In the South African school curriculum, there is only one subject that offers Computer Programming. The subject is called Information Technology (IT)¹⁷, offered in the FET¹⁸ (Further Education and Training) phase and the prescribed programming language is Delphi (Department of Basic Education, 2011). During this research, the South African National Department of Education had started to pilot a subject that comprises coding and Robotics at the primary school level. It is hoped that this study's findings will contribute to the fine-tuning of this prospective subject.

In response to the 4IR, the South African curriculum should provide the necessary additions to the subject package offerings by introducing coding and Robotics to early grades. There should be a growing interest and use of robots in learning due to its diverse nature and ability to integrate across multiple disciplines. The application of Robotics involves developing technical knowledge from construction to programming in real-world scenarios (Kaloti-Hallak & Armoni, 2015). Therefore, the learning of coding can be regarded as secondary knowledge acquisition along the learning path when integrated with subject matter from other disciplines.

¹⁷ An Information Technology teacher is the equivalent to a Computer Science teacher

¹⁸ Grades 10-12 late high school

This research can have a considerable impact by providing recommendations to policy and curriculum changes that involve Robotics and coding.

Coding in a text-based programming environment using Python was confirmed to be easy for the novice programmer enabling them to grasp the basics. The Python language was designed with education in mind, having simple syntax compared with other languages. The use of Python to introduce coding allows the focus to be placed on algorithmic thinking and programming rather than on learning complex, verbose syntax. At the end of each workshop, students understood the content and were able to apply the skills developed to the next activity and workshop. Such skills include conditions (*if* statements), loops (*for* and *while*), operators, functions, basic data types and the Python programming syntax. Irrespective of the programming language, computational, problem-solving and coding skills can be applied to any language since only the syntactical structure changes.

Multiple contributing factors play a role in developing coding knowledge when Robotics is used in the learning process. Mathematics and Computer Science provide a vital foundation for answering pivotal and complex questions, from quantum computing to advancements in biotechnology. The study showed that the type of mathematic background is not a contributing factor for determining rational and logical ability. This was measured through the ART and CT test. More influential is an individual's **belief** about coding, **interest** in coding and **motivation** to code, which affect the individual's **confidence** in the knowledge attainment of coding through the use of Robotics. These factors collectively harness and exercise an individual's (*subject*) problem-solving skills and higher-order thinking skills that develop while becoming proficient in Computer Programming.

Furthermore, these factors are important for student success since pedagogic techniques that are considered uninteresting tend to lead to higher student dropout rates. Therefore, it is crucial to choose approaches that are appealing and fun. The findings show that robot coding is a more effective method, as compared with traditional approaches, for introducing programming.

The initial plan was to hold face-to-face workshop sessions, but due to the COVID-19 lockdown, online workshops were conducted. Although this was unintentional, the study has shown that developing one's coding skills through Robotics in an online environment is possible.

The use of Robotics in the learning of text-based coding using Python made the introduction of Computer Programming interesting and easier to understand. Robotics simplified the understanding of Computer Programming, making creating a concrete experience, where one could see, touch and witness real-life changes (autonomous) to the prototype that was built and coded. The introduction of coding through the use of Robotics propelled the novice from being consumers of code to creators and producers.

7.4 Limitation and further research

It would be ideal to repeat this research over three–four years at multiple institutions. In addition, the possibility of having sub-sampling with three or four sets of participants, based on a distinct characteristic, would prove to be interesting. The option was not possible given the current study’s dependence on funds, logistics, limited robotic equipment and the unforeseen COVID-19 pandemic.

A more extensive data set would have enhanced the findings. However, the current study yielded comparative data, leading to generalizable findings. A controlled experiment is needed to claim causation, especially with regards to the post-test and pre-test. However, this was not necessary as this research was an explorative study that tested an intervention that used Robotics to learn basic coding using Python.

The Python language avoids the requirement of a variable declaration by supporting dynamic typing, which does not prevent the novice from typing incorrect code (Nasrawt & Lam, 2019). Future research could consider using different programming languages other than Python to allow for the checking of typing prior to execution of compile and run. In some contexts, with limited ICT infrastructure, robotic equipment might be challenging to procure, thus posing a challenge for such an approach to the learning of code. The researcher proposes to use coding and robotics in the first year introduction course to programming and similar studies will be conducted and results published in journal papers.

7.5 Conclusion

The study sought to explore the use of Robotics in the learning of basic coding among a cohort of students without university-level coding experience. Design Base Research was used to direct the research. Kolb's Experiential Learning Cycle integrated into the prototyping phase of DBR. As a framework of theorising, second-generation Activity Theory was used. All elements or actors in the second-generation activity system were found to be essential in obtaining the *outcome*. Mixed-methods research was used for data collection, according to the pragmatic paradigm favoured by DBR.

Findings revealed that **belief**, **interest** and **motivation** played a pivotal role in participant **confidence** in the acquisition of coding knowledge through the use of Robotics. While Mathematics might influence coding ability, many studies has shown otherwise. It is key to point out that, in relation to Activity Theory, participant confidence and perception of Mathematics were strongly influenced by their surroundings that form their *community*. After analysis and interpretation of the results, the findings strongly suggest that the use of Robotics played a pivotal role in the acquisition of coding knowledge and skills. In addition, the students expressed the use of Robotics to be exciting and fun, which promoted enthusiasm in learning to code. The robotic element, as a physical manipulative to the learning of coding, simplified the understanding of text-based coding. That allowed for early prediction and expectancy of what to code and which coding structures to use based on the prototype built. The robotic element 'brought the code to life', offering a three-dimensional, autonomous and live output of the code, which solidified understanding of basic coding structures and syntax (input, output, condition, loops, etc.). It is hoped that the experience students encountered be shared to encourage and motivate others in the learning of Computer Programming and Robotics, which are deemed to be crucial 4IR skills.

References

- Abdallah, M. (2013). Employing a three-phase design-based research methodology for expanding student teachers' language-related literacy practices in an Egyptian pre-service English education programme. In T. Plomp & N. Nieveen (Eds.), *Educational design research – Part B: illustrative cases* (pp. 927-946). SLO.
- ACM/IEEE-CS. (2013). *Joint Task Force on Computing Curricular: computer science curricular technical report*.
https://www.acm.org/binaries/content/assets/education/cs2013_web_final.pdf
- Akcay, N., Avci, H., Gungor, A. & Adiguzel, T. (2018). The relationship between computer programming and English language skills. In V. Dagiene & E. Jasute (Eds.), *Constructionism, computational thinking and educational innovation*. Proceedings of Constructionism 2018 (pp. 782-785).
http://www.constructionism2018.fsf.vu.lt/file/repository/Proceeding_2018_Constructionism.pdf
- Al-Jepoori, M. & Bennett, D. (2018). Understanding of the programming techniques by using a complex case study to teach advanced object-oriented programming. *International Journal of Educational and Pedagogical Sciences*, 12(8), 1060-1064.
- Aleksić, V. & Ivanović, M. (2016). Introductory programming subject in European higher education. *Informatics in Education*, 15(2), 163-182.
<https://doi.org/10.15388/INFEDU.2016.09>
- Alghamdi, A. H. & Li, L. (2013). Adapting design-based research as a research methodology in educational settings. *International Journal of Education and Research*, 1(10), 1-12.
- Arduino UNO R3 (n.d.). *Arduino UNO R3 pin diagram*. Elprocus.
<https://www.elprocus.com/what-is-arduino-uno-r3-pin-diagram-specification-and-applications/>
- Armoni, M., Meerbaum-Salant, O. & Ben-Ari, M. (2015). From scratch to real programming. *ACM Transactions on Computing Education (TOCE)*, 14(4), 1-15.
<https://doi.org/10.1145/2677087>

- Aslam, S. K., Faithful, W. J. & Teahan, W. J. (2018). A middleware to link Lego Mindstorms robots with 4th generation language software NetLogo. In M. Bramer and M. Petridis (Eds.), *International conference on innovative techniques and applications of artificial intelligence*. (pp. 416-430). <https://doi.org/10.1007/978-3-030-04191-5>
- Astrachan, C. B., Patel, V. K. & Wanzenried, G. (2014). A comparative study of CB-SEM and PLS-SEM for theory development in family firm research. *Journal of Family Business Strategy*, 5(1), 116-128. <https://doi.org/10.1016/j.jfbs.2013.12.002>
- Azad, M. S. A. R., Raouf, M. S., Nabi, R. & Hussein, D. (2018). The impact of teaching materials on learning computer programming languages in Kurdistan region universities and institutes. *Kurdistan Journal of Applied Research*, 3(1), 27-33. <https://doi.org/10.24017/SCIENCE.2018.1.7>
- Bagozzi, R. P. & Yi, Y. (1988). On the evaluation of structural equation models. *Journal of the Academy of Marketing Science*, 16(1), 74-94. <https://doi.org/10.1007/BF02723327>
- Bakker, A. & Van Eerde, D. (2015). An introduction to design-based research with an example from statistics education. In A. Bikner-Ahsbabs, C. Knipping & N. Presmeg (Eds.), *Approaches to qualitative research in mathematics education* (pp. 429-466). Springer.
- 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. <https://doi.org/10.1145/1929887.1929905>
- Barrett, S. F. (2020). *Arduino I: getting started*. Morgan & Claypool.
- Bati, T. B., Gelderblom, H. & Van Biljon, J. (2014). A blended learning approach for teaching computer programming: design for large classes in Sub-Saharan Africa. *Computer Science Education*, 24(1), 71- 99. <https://doi.org/10.1080/08993408.2014.897850>
- Bau, D., Sheldon, J., Gray, J., Kelleher, C. & Turbak, F. (2017). Learnable programming: blocks and beyond. *Communications of the Acm*, 60(6), 72-80. <https://doi.org/10.1145/3015455>

- Belski, I. (2009). Teaching thinking and problem solving at university: a course on TRIZ. *Creativity and Innovation Management*, 18(2), 101-108.
<https://doi.org/10.1111/j.1467-8691.2009.00518.x>
- Bertram, C. & Christiansen, I. (2014). *Understanding research: an introduction to reading research*. Van Schaik.
- Biggers, M., Brauer, A. & Yilmaz, T. (2008). Student perceptions of computer science: a retention study comparing graduating seniors with cs leavers. *ACM SIGCSE Bulletin*, 40(1), 402-406. <https://doi.org/10.1145/1352322.1352274>
- Biju, S. M. (2013). Difficulties in understanding object oriented programming concepts. In K. Elleithy & T. Sobh (Eds.), *Innovations and advances in computer, information, systems sciences, and engineering* (pp. 319-326). Springer.
- Blanca, M. J., Arnau, J., López-Montiel, D., Bono, R. & Bendayan, R. (2013). Skewness and kurtosis in real data samples. *Methodology*, 9(1), 78-84. <https://doi.org/10.1027/1614-2241/a000057>
- Blanchard, J. (2017). Hybrid environments: a bridge from blocks to text. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*, (pp. 295-296). ACM. <https://doi.org/10.1145/3105726.3105743>
- Blotnicky, K. A., Franz-Odendaal, T., French, F. & Joy, P. (2018). A study of the correlation between STEM career knowledge, mathematics self-efficacy, career interests, and career activities on the likelihood of pursuing a STEM career among middle school students. *International Journal of STEM Education*, 5(1), 1-15.
<https://doi.org/10.1186/s40594-018-0118-3>
- Bogdan, M. (2018). Multiple solutions in linear programming problem. *Procedia Manufacturing*, 22(1), 1063-1068. <https://doi.org/10.1016/j.promfg.2018.03.151>
- Boldbaatar, N. & Sendurur, E. (2018). Developing educational 3D games with starlogo: the role of backwards fading in the transfer of programming experience. *Journal of Educational Computing Research*, 57(6), 1468-1494.
<https://doi.org/10.1177/0735633118806747>

- Börstler, J. & Schulte, C. (2005). Teaching object oriented modelling with CRC cards and roleplaying games. In J. Pittman (Ed.), *WCCE 2005: 8th IFIP World Conference on Computers in Education: 40 years of computers in education: What works?* (pp. 1-9). Emerald.
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.437.3492&rep=rep1&type=pdf>
- Brown, J. D. (1997). Skewness and kurtosis. *JALT Testing & Evaluation SIG Newsletter*, 1(1), 20-23.
- Bubica, N. & Boljat, I. (2015). Programming novices' mental models. In L. G. Chova, A. L. Martínez, & I. C. Torres (Eds.), *Education and New Learning Technologies*. Proceedings of the 7th International Conference on Education and New Learning Technologies (pp. 5882-5891). IATED. <https://doi.org/10.13140/RG.2.1.3773.2960>
- Bubnó, K. T., Takács, V., Ambrus, A. & Vásárhelyi, É. (2014). Solving word problems by computer programming. In A. Ambrus & É. Vásárhelyi (Eds.), *ProMath*. Proceedings of the Problem Solving in Mathematics Education 2013 Conference (pp. 193-208). Eötvös Loránd University. <http://webdoc.urz.uni-halle.de/dl/287/pub/ProMath2013.pdf>
- Byrne, B. M. (2010). *Structural equation modeling with AMOS: basic concepts, applications, and programming* (2nd ed.). Routledge/Taylor & Francis Group.
- Byrne, P. & Lyons, G. (2001). The effect of student attributes on success in programming. *ACM SIGCSE Bulletin*, 33(3), 49-52. <https://doi.org/10.1145/507758.377467>
- Calao, L. A., Moreno-León, J., Correa, H. E. & Robles G. (2015). Developing mathematical thinking with Scratch. In G. Conole, T. Klobučar, C. Rensing, J. Konert & E. Lavoué (Eds.), *Design for teaching and learning in a networked world* (pp. 17-27). Springer International.
- Calderon, A. C., Crick, T. & Tryfona, C. (2015). Developing computational thinking through pattern recognition in early years education. In S. Lawson & P. Dickson (Eds.), *Proceedings of the 2015 British HCI Conference* (pp. 259-260). Association for Computing Machinery. <https://doi.org/10.1145/2783446.2783600>

- Cansu, S. K. & Cansu, F. K. (2019). An overview of computational thinking. *International Journal of Computer Science Education in Schools*, 3(1), 1-11.
<https://doi.org/10.21585/ijcses.v3i1.53>
- Carbone, A., Hurst, J., Mitchell, I. & Gunstone, D. (2009). An exploration of internal factors influencing student learning of programming. *Proceedings of the Eleventh Australasian Conference on Computing Education - Volume 95* (pp. 25-34). Australian Computer Society.
- Carvalho, M. B., Bellotti, F., Berta, R., De Gloria, A., Sedano, C. I., Hauge, J. B., Hu, J. & Rauterberg, M. (2015). An activity theory-based model for serious games analysis and conceptual design. *Computer Education*, 87(1), 166-181.
<https://doi.org/10.1016/j.compedu.2015.03.023>
- Cass S. (2021). *Top programming languages is one of IEEE Spectrum's most popular interactives*. IEEE. <https://spectrum.ieee.org/top-programming-languages/#toggle-gdpr>
- Catlin, D., Kandlhofer, M., Holmquist, S., Csizmadia, A. P., Angel-Fernandez, J. & Cabibihan, J.-J. (2018). EduRobot taxonomy and Papert's paradigm. In V. Dagienė & E. Jasutė (Eds.), *Constructionism 2018: Constructionism, computational thinking and educational innovation: Conference proceedings*. (pp. 150-159). Vilnius University.
http://www.constructionism2018.fsf.vu.lt/file/repository/Proceeding_2018_Constructionism.pdf
- Cazzola, W. & Olivares, D. M. (2015). Gradually learning programming supported by a growable programming language. *IEEE Transactions on Emerging Topics in Computing*, 4(3), 404-415. <https://doi.org/10.1109/TETC.2015.2446192>
- Chandler, P. D. (2017). To what extent are teachers well prepared to teach multimodal authoring?. *Cogent Education*, 4(1), 1-19.
<https://doi.org/10.1080/2331186X.2016.1266820>
- Chapman, B. E. & Irwin, J. (2015). Python as a first programming language for biomedical scientists. In K. Huff & J. Bergstra (Eds.), *SciPy 2015*. Proceedings of the 14th Python in Science Conference, (pp. 12-17). SciPy. <https://doi.org/10.25080/MAJORA-7B98E3ED-002>

- Check, J. & Schutt, R. (2012). *Research methods in education*. SAGE.
<https://www.doi.org/10.4135/9781544307725>
- Chen, Y., Spagna, A., Wu, T., Kim, T. H., Wu, Q., Chen, C., ... Fan, J. (2019). Testing a cognitive control model of human intelligence. *Scientific Reports*, 9(1), 2898.
<https://doi.org/10.1038/s41598-019-39685-2>
- Chen, Q., Tang, Y., Li, L., Yang, G., Yang, M., Xie, Z., ... Huang, R. (2017). A practice on Lego Mindstorms for computer science freshman experimental education. *Destech Transactions on Social Science, Education and Human Science*, 1(1), 17-21.
- Chesher, C. (2018). Mechanology, Mindstorms, and the genesis of robots. In S. J. Thompson (Ed.), *Androids, cyborgs, and robots in contemporary culture and society* (pp. 120-137). IGI Global.
- Chetty, J. & Barlow-Jones, G. (2012). Bridging the gap: the role of mediated transfer for computer programming. *Proceedings of the 4th International Conference on Education Technology and Computer* (pp. 1-5). IACSIT Press. <http://www.ipcsit.com/vol43/001-ICETC2012-C0019.pdf>
- Chin, W. W. (1998). Commentary: issues and opinion on structural equation modeling. *MIS Quarterly*, 22(1), 7-16. <http://www.jstor.org/stable/249674>
- Chin, W. W. (2010). How to write up and report PLS analyses. In V. E. Vinzi, W. W. Chin, J. Henseler & H. Wang (Eds.), *Handbook of partial least squares* (pp. 655-690). Springer.
- Chin, K.-Y., Hong, Z.-W. & Chen, Y.-L. (2014). Impact of using an educational robot-based learning system on students' motivation in elementary education. *IEEE Transactions on Learning Technologies*, 7(4), 333–345. <https://doi.org/10.1109/TLT.2014.2346756>
- Clark, S. K. (2015). Research by design: design-based research and the higher degree research student. *Journal of Learning Design*, 8(3), 108-122.
- Clemmensena, T., Kaptelininb, V. & Nardic, B. (2016). Making HCI theory work: an analysis of the use of activity theory in HCI research. *Behaviour & Information Technology*, 35(8), 608-627. <https://doi.org/10.1080/0144929X.2016.1175507>

- Cochrane, T., Cook, S., Aiello, S., Christie, D., Sinfield, D., Steagall, M. & Aguayo, C. (2017). A DBR framework for designing mobile virtual reality learning environments. *Australasian Journal of Educational Technology*, 33(6), 54-68.
- Cohen, J. (1988). *Statistical power analysis for the behavioral sciences*. Lawrence Erlbaum.
- Cohen, L., Manion, L. & Morrison, K. (2002). *Research methods in education*. Routledge.
- Collatto, D., Dresch, A., Lacerda, D. & Bentz, I. (2018). Is action design research indeed necessary? Analysis and synergies between action research and design science research. *Systemic Practice and Action Research*, 31(3), 239-267.
<https://doi.org/10.1007/s11213-017-9424-9>
- Combéfis, S., Beresnevičius, G. & Dagienė, V. (2016). Learning programming through games and contests: overview, characterisation and discussion. *Olympiads in Informatics*, 10(1), 39-60. <https://doi.org/10.15388/IOI.2016.03>
- Comer, D. (2017). *Essentials of computer architecture*. Chapman and Hall.
- Cramer, P. (1998). Defensiveness and defense mechanisms. *Journal of Personality*, 66(6), 879-894. <https://doi.org/10.1111/1467-6494.00035>
- Cramer, D., & Howitt, D. L. (2004). *The Sage dictionary of statistics: a practical resource for students in the social sciences*. Sage.
- Creswell, J. W. (2003). *Research design: qualitative, quantitative, and mixed methods approaches*. (2nd ed.). Sage.
- Creswell, J. W. & Clark, V. L. P. (2011). *Designing and conducting mixed methods research*. Sage.
- Creswell, J. W. & Creswell, J. D. (2018). *Research design: qualitative, quantitative and mixed methods approaches* (5th ed.). Sage.
- Das, K. R. & Imon, A. H. M. R. (2016). A brief review of tests for normality. *American Journal of Theoretical and Applied Statistics*, 5(1), 5-12.
<https://doi.org/10.11648/j.ajtas.20160501.12>

- Daly, T. (2011). Minimizing to maximize: an initial attempt at teaching introductory programming using Alice. *Journal of Computing Sciences in Colleges*, 26(5), 23-30.
- Department of Basic Education. (2018). *Annual report*.
<https://www.education.gov.za/Resources/Reports.aspx>.
- DeCarlo L. T. (1997). On the meaning and use of kurtosis. *Psychological Methods*, 2(3), 292-307. <https://doi.org/10.1037/1082-989X.2.3.292>
- Demo, G., Neiva, E. R., Nunes, I. & Rozzett, K. (2012). Human resources management policies and practices scale (HRMPPS): exploratory and confirmatory factor analysis. *BAR-Brazilian Administration Review*, 9(4), 395-420.
<https://doi.org/10.1590/S1807-76922012005000006>
- Denning, P. J. (2009). Beyond computational thinking. *Communications of the ACM*. 52(6), 28-30. <https://doi.org/10.1145/1516046.1516054>
- Denscombe, M. (2014). *The good research guide* (5th ed.). McGraw-Hill.
- Derus, S. R. & Ali, A. Z. M. (2012). Difficulties in learning programming: views of students. In S. I. A. Dwiningrum (Ed.), *Current issues in education*. Proceedings of the International Centre for Innovation in Education (pp. 74-78).
<https://doi.org/10.13140/2.1.1055.7441>
- Department of Basic Education. (2011). *Curriculum assessment policy statement*. Pretoria, South Africa: Department of Education.
- Dewey, J. (1905). The postulate of immediate empiricism. *The Journal of Philosophy, Psychology and Scientific Methods*, 2(15), 393-399. <https://doi.org/10.2307/2011400>
- Dewey, J. (1925). *Experience and nature*. Dover.
- Dewey, J. (1929). *The quest for certainty*. Minton, Balch and Company.
- Diamantopoulos, A. & Sigauw, J. A. (2006). Formative versus reflective indicators in organizational measure development: comparison and empirical illustration. *British Journal of Management*, 17(4), 263-282. <https://doi.org/10.1111/j.1467-8551.2006.00500.x>

- Dijkstra, T. K. & Henseler, J. (2015). Consistent partial least squares path modeling. *MIS Quarterly*, 39(2), 297-316. <https://doi.org/10.25300/MISQ/2015/39.2.02>
- Dikko, M. (2016). Establishing construct validity and reliability: pilot testing of a qualitative interview for research in Takaful (Islamic insurance). *The Qualitative Report*, 21(3), 521-528. <https://doi.org/10.46743/2160-3715/2016.2243>
- Dmitrieva, T. A., Prutzkow, A. V. & Pylkin, A. N. (2019). Two-level study of object-oriented programming by university students. *Modern Information Technology and IT Education*, 15(1), 200-206. <https://doi.org/10.25559/SITITO.15.201901.200-206>
- Doane, D. P. & Seward, L. E. (2011). Measuring skewness: a forgotten statistic?. *Journal of Statistics Education*, 19(2). <https://doi.org/10.1080/10691898.2011.11889611>
- Doshi, V. P. & Patil, V. (2016). Competitor driven development hybrid of extreme programming and feature driven reuse development. *2016 International Conference on Emerging Trends in Engineering, Technology and Science (ICETETS)* (pp. 50-55). IEEE. <https://doi.org/10.1109/ICETETS.2016.7602985>
- Douglas, M. E., Peecksen, S., Rogers, J. & Simmons, M. (2019). College students' motivation and confidence for ePortfolio use. *International Journal of ePortfolio*, 9(1), 1-16.
- Economist (2018, July 26). *Python is becoming the world's most popular coding language* <https://www.economist.com/graphic-detail/2018/07/26/python-is-becoming-the-worlds-most-popular-coding-language>
- Elaine, J. H. (2013). *What is maths? Live science*. McGraw Hill.
- Engeström, Y. (2001). Expansive learning at work: toward an activity theoretical reconceptualization. *Journal of Education and Work*, 14(1), 133-156. <https://doi.org/10.1080/13639080020028747>
- Erdogmus, H., Morisio, M. & Torchiano, M. (2005). On the effectiveness of the test-first approach to programming. *IEEE Transactions on Software Engineering*, 31(3), 226-237. <https://doi.org/10.1109/TSE.2005.37>

- Erol, O. & Kurt, A. A. (2017). The effects of teaching programming with scratch on pre-service information technology teachers' motivation and achievement. *Computers in Human Behavior*, 77(1), 11–18. <https://doi.org/10.1016/j.chb.2017.08.017>
- Fagan, M. (2002). Design and code inspections to reduce errors in program development. In M. Broy & E. Denert (Eds.), *Software Pioneers* (pp. 575- 607). Springer.
- Fang, X. (2012). Application of the participatory method to the computer fundamentals course, Affective Computing and Intelligent Interaction. *Advances in Intelligent and Soft Computing*, 137(1), 185-189.
- Farag, W., Ali, S. & Deb, D. (2013). Does language choice influence the effectiveness of online introductory programming courses? In W. D. Armitage (Ed.), *Information technology education. SIGITE '13: Proceedings of the 14th annual ACM SIGITE Conference on Information Technology Education* (pp. 165-170). Association for Computing Machinery. <https://doi.org/10.1145/2512276.2512293>
- Feilzer, M. (2010). Doing mixed methods research pragmatically: implications for the rediscovery of pragmatism as a research paradigm. *Journal of Mixed Methods Research*, 4(1), 6-16. <https://doi.org/10.1177/1558689809349691>
- Fischer, M. & Sliwka, D. (2018). Confidence in knowledge or confidence in the ability to learn: an experiment on the causal effects of beliefs on motivation. *Games and Economic Behavior*, 111, 122-142. <https://doi.org/10.1016/j.geb.2018.02.005>
- Firat, M., Kılınç, H. & Yüzer, T. V. (2018). Level of intrinsic motivation of distance education students in e-learning environments. *Journal of Computer Assisted Learning*, 34(1), 63-70. <https://doi.org/10.1111/jcal.12214>
- Ford, C., McNally, D. & Ford, K. (2017). Using design-based research in higher education innovation. *Online Learning*, 21(3), 50-67.
- Fornell, C. & Larcker, D. F. (1981). Evaluating structural equation models with unobservable variables and measurement error. *Journal of Marketing Research*, 18(1), 39-50. <https://doi.org/10.2307/3151312>

- Frost, J. (n.d.) *Introduction to bootstrapping in statistics with an example*. Making statistics intuitive. <https://statisticsbyjim.com/hypothesis-testing/bootstrapping/>
- Feng, A., Gardner, M. & Feng, W. C. (2017). Parallel programming with pictures is a Snap!. *Journal of Parallel and Distributed Computing*, 105(1), 150-162. <https://doi.org/10.1016/j.jpdc.2017.01.018>
- Ferrari, A., Poggi, A. & Tomaiuolo, M. (2016). Object oriented puzzle programming. *Mondo Digitale*, 15(2016), 1-10.
- Gal-Ezer, J. & Stephenson, C. (2010). Computer science teacher preparation is critical. *ACM Inroads*, 1(1), 61-66. <https://doi.org/10.1145/1721933.1721953>
- García-Peñalvo, F. J. (2018). Computational thinking. *IEEE Ibero-American Journal of Learning Technologies*, 13(1), 17-19. <https://doi.org/10.1109/RITA.2018.2809939>
- García-Peñalvo, F. J. (2018). Computational thinking and programming education principles. In F. J. García-Peñalvo (Ed.), *TEEM'18: Proceedings of the Sixth International Conference on Technological Ecosystems for Enhancing Multiculturality* (pp. 14-17). Association for Computing Machinery <https://doi.org/10.1145/3284179.3284184>
- Garson, G. D. (2016). *Partial least squares: regression and structural equation models*. Statistical Associates Publishers.
- Gaudiello, I. & Zibetti, E. (2016). *Learning robotics, with robotics, by robotics: educational robotics*. John Wiley & Sons.
- Gasparetto, A. & Scalera, L. (2019). From the unimate to the delta robot: the early decades of industrial robotics. *History of Mechanism and Machine Science*, 37(1), 284-295. https://doi.org/10.1007/978-3-030-03538-9_23
- Gefen, D., Straub, D. & Boudreau, M. C. (2000). Structural equation modeling and regression: guidelines for research practice. *Communications of the Association for Information Systems*, 4(1), 1-78. <https://doi.org/10.17705/1CAIS.00407>
- Geisser, S. (1974). A predictive approach to the random effect model. *Biometrika*, 61(1), 101-107. <https://doi.org/10.1093/BIOMET/61.1.101>

- Godwin, J. A. (2021). *Linear regression with bootstrapping*. Towards data science.
<https://towardsdatascience.com/linear-regression-with-bootstrapping-4924c05d2a9#:~:text=The%20bootstrap%20method%20can%20be,small%20changes%20in%20data%20values>
- Govender, I. (2010). From procedural to object-oriented programming (OOP) - an exploratory study of teachers' performance. *South African Computer Journal*, 46(2010), 14-23. <https://hdl.handle.net/10520/EJC28111>
- Goff, W. M. & Getenet, S. (2017). Design based research in doctoral studies: adding a new dimension to doctoral research. *International Journal of Doctoral Studies*, 12(1), 107-121.
- Govender, R. G. & Govender, D. W. (2020), ROBOPROG: Learning of flowcharts through a gamified experience, *International Journal of Business and Management Studies*, 12(2). 612-624.
- Govender, T. P. & Govender, D. W. (2016). Peer-to-peer programming versus individualised programming: the real world. *The Independent Journal of Teaching and Learning*, 11(1), 56-68. <http://hdl.handle.net/11622/120>
- Graham, R. L., Knuth, D. E., Patashnik, O. & Liu. S. (1989). Concrete mathematics: a foundation for computer science. *Computers in Physics*, 3(106), 106-107.
<https://doi.org/10.1063/1.4822863>
- Grover, S. & Pea, R. (2013). Computational thinking in K–12: a review of the state of the field. *Educational Researcher*, 42(1), 38–43. <https://doi.org/10.3102/0013189X12463051>
- Hadad, S., Shamir-Inbal, T., Blau, I. & Leykin, E. (2021). Professional development of code and robotics teachers through Small Private Online Course (SPOC): teacher centrality and pedagogical strategies for developing computational thinking of students. *Journal of Educational Computing Research*, 59(4), 763-791.
<https://doi.org/10.1177/0735633120973432>
- Hair, J. F., Black, W.C., Babin, B.J. & Anderson, R.E. (2010). *Multivariate data analysis* (7th ed.). Pearson International.

- Hair, J. F., Hult, G. T. M., Ringle, C. & Sarstedt, M. (2016). *A primer on partial least squares structural equation modeling (PLS-SEM)*. Sage.
- Hair, J. F., Ringle, C. M. & Sarstedt, M. (2011). PLS-SEM: indeed a silver bullet. *Journal of Marketing Theory and Practice*, 19(2), 139-152. <https://doi.org/10.2753/MTP1069-6679190202>
- Hair, J. F., Sarstedt, M., Hopkins, L. & Kuppelwieser, V. G. (2014). Partial least squares structural equation modeling (PLS-SEM): an emerging tool in business research. *European Business Review*, 26 (2), 106-121. <https://doi.org/10.1108/EBR-10-2013-0128>
- Hair, J. F., Sarstedt, M., Ringle, C. M. & Gudergan, S. P. (2017). *Advanced issues in partial least squares structural equation modeling*. Sage.
- Hair, J. F., Sarstedt, M., Ringle, C. M. & Mena, J. A. (2012). An assessment of the use of partial least squares structural equation modeling in marketing research. *Journal of the Academy of Marketing Science*, 40(3), 414-433. <https://doi.org/10.1007/s11747-011-0261-6>
- Hamzah, N., Shaari, N. & Rahman, H. (2019). Undergraduate computer science students' perception and motivation: a feasibility study and a proposed technique for multimedia approach in teaching and learning introductory programming. In N. Mohamad Noor, B. Ahmad, M. Ismail, H. Hashim & B. Abdullah Baharum (Eds.), *Proceedings of the Regional Conference on Science, Technology and Social Sciences: Social Sciences* (pp. 187-201). Springer. https://doi.org/10.1007/978-981-13-0203-9_18
- Harel, O. (2009). The estimation of R² and adjusted R² in incomplete data sets using multiple imputation. *Journal of Applied Statistics*, 36(10), 1109-1118. <https://doi.org/10.1080/02664760802553000>
- Hasan, H. (1999). Integrating IS and HCI using activity theory as a philosophical and theoretical basis. *Australasian Journal of Information Systems*, 6(2), 44-55. <https://doi.org/10.3127/ajis.v6i2.305>
- Hendrix, R. & Weeks, M. (2018). First programming language for high school students. In E. Langran & J. Borup (Eds.), *Proceedings of Society for Information Technology & Teacher Education International Conference* (pp. 1896-1903). Association for the

Advancement of Computing in Education.

<https://www.learntechlib.org/primary/p/182788/>

Henseler, J., Ringle, C.M. & Sarstedt, M. (2015). A new criterion for assessing discriminant validity in variance-based structural equation modeling. *Journal of the Academy of Marketing Science*, 43(1), 115-135. <https://doi.org/10.1007/s11747-014-0403-8>

Henseler, J., Ringle, C.M. & Sinkovics, R.R. (2009). The use of partial least squares path modeling in international marketing. In R. R. Sinkovics & P. N. Ghauri (Eds.), *New challenges to international marketing (advances in international marketing)* (pp. 277-319). Emerald Group Publishing.

Homer, M. & Noble, J. (2017). Lessons in combining block-based and textual programming. *Journal of Visual Languages and Sentient Systems*, 3(1), 22-39. <https://doi.org/10.18293/VLSS2017-007>

Hosanee, Y. & Panchoo, S. (2015). An enhanced software tool to aid novices in learning object oriented programming (OOP). In *2015 International Conference on Computing, Communication and Security (ICCCS)* (pp. 1-7). IEEE. <https://doi.org/10.1109/CCCS.2015.7374197>

Hourani, H., Wasmi, H. & Alrawashdeh, T. (2019). A code complexity model of object oriented programming (OOP). In K. M. Jaber (Ed.), *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT): Proceedings*. (pp. 560-564). IEEE. <https://doi.org/10.1109/JEEIT.2019.8717448>

Howe, K. R. (1988). Against the quantitative-qualitative incompatibility thesis or dogmas die hard. *Educational Researcher*, 17(8), 10-16. <https://doi.org/10.3102/0013189X017008010>

Hromkovič, J., Kohn, T., Komm, D. & Serafini, G. (2016). Examples of algorithmic thinking in programming education. *Olympiads in Informatics*, 10(1), 111-124.

Huang, Y., Zhang, T. & Xu, L. (2018). The development of a game with applications of object-oriented programming concepts. *American Journal of Advanced Research*, 2(1), 7-13. <https://doi.org/10.5281/zenodo.1410734>

- Hulin, C., Netemeyer, R. & Cudeck, R. (2001). Can a reliability coefficient be too high?. *Journal of Consumer Psychology, 10*(1), 55-58. <https://doi.org/10.2307/1480474>
- Hulland, J. (1999). Use of partial least squares (PLS) in strategic management research: a review of four recent studies. *Strategic Management Journal, 20*(2), 195-204. [https://doi.org/10.1002/\(SICI\)1097-0266\(199902\)20:2<195::AID-SMJ13>3.0.CO;2-7](https://doi.org/10.1002/(SICI)1097-0266(199902)20:2<195::AID-SMJ13>3.0.CO;2-7)
- Husain, M., Patil, S., Shettar, P., Meti, A.S. & Bidari, I. (2016). The role of programming paradigms in building projects. *Journal of Engineering Education Transformations, 29*(3), 155-160. <https://doi.org/10.16920/jeet/2016/v29i3/85251>
- Inayama, Y. & Hosobe, H. (2018). Toward an efficient user interface for block-based visual programming. In J. Cunha, J. P. Fernandes, C. Kelleher, G. Engels and J. Mendes (Eds.), *Proceedings 2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (pp. 293-294). IEEE. <https://doi.org/10.1109/VLHCC.2018.8506530>
- Ioannou, A. & Makridou, E. (2018). Exploring the potentials of educational robotics in the development of computational thinking: A summary of current research and practical proposal for future work. *Education and Information Technologies, 23*(6), 2531-2544. <https://doi.org/10.1007/s10639-018-9729-z>
- Jackson, D. & Miller, R. (2009). *A new approach to teaching programming*. MIT Press. <http://people.csail.mit.edu/dnj/articles/teaching-6005.pdf>
- Jacob, S., Nguyen, H., Tofel-Grehl, C., Richardson, D. & Warschauer, M. (2018). Teaching computational thinking to English learners. *NYS TESOL Journal, 5*(2), 12-24. <https://doi.org/10.26716/jcsi.2018.01.1.1>
- Javidi, G., & Sheybani, E. (2018). Teaching computer programming through game design: a game-first approach. *GSTF Journal on Computing, 4*(1), 17-22.
- Job Test Prep (2019). *Free Abstract Reasoning Test*. <https://www.jobtestprep.co.uk/free-abstract-reasoning-test>

- Jonassen, D. H. (1999). Designing constructivist-learning environments. In C. Reigeluth (Ed.), *Instructional design theories and models: a new paradigm of instructional theory* (pp. 215-239). Lawrence Erlbaum Associates.
- Jonassen, D. H. & Rohrer-Murphy, L. (1999). Activity theory as a framework for designing constructivist learning environments. *Educational Technology Research and Development*, 47(1), 61-79. <https://doi.org/10.1007/BF02299477>
- Káta, Z. (2014). The challenge of promoting algorithmic thinking of both sciences and humanities oriented learners. *Journal of Computer Assisted Learning*, 31(4), 287-299. <http://dx.doi.org/10.1111/jcal.12070>
- Kalelioğlu, F. (2015). A new way of teaching programming skills to K-12 students: code.org. *Computers in Human Behavior*, 52(1), 200-210. <https://doi.org/10.1016/j.chb.2015.05.047>
- Kalelioğlu, F. & Gülbahar, Y. (2014). The effects of teaching programming via Scratch on problem solving skills: a discussion from learners' perspective. *Informatics in Education*, 13(1), 33-50.
- Kaloti-Hallak, F. & Armoni, M. (2015). The effectiveness of robotics competitions on students' learning of computer science. *Olympiads in Informatics*, 9(1), 89–112. <http://doi.org/10.15388/ioi.2015.08>
- Kalra, H. K. & Chadha, R. (2018, March). A review study on humanoid robot SOPHIA based on artificial intelligence. *International Journal of Technology and Computing (IJTC)*, 4(3), 31-33.
- Karch, J. & Van Ravenzwaaij, D. (2020). Improving on adjusted R-squared. *Collabra: Psychology*, 6(1). <https://doi.org/10.1525/collabra.343>
- Karim, M. E., Lemaignan, S. & Mondada, F. (2016). A review: can robots reshape K-12 STEM education?. *2015 IEEE International Workshop on Advanced Robotics and its Social Impacts (ARSO)* (pp. 1-8). Curran Associates. <https://doi.org/10.1109/ARSO.2015.7428217>

- Karp, T., Gale, R., Lowe, L. A., Medina, V. & Beutlich, E. (2010). Generation NXT: building young engineers with LEGOs. *IEEE Transactions on Education*, 53(1), 80-87. <https://doi.org/10.1109/TE.2009.2024410>
- Khaleel, F. L., Ashaari, N. S., Tengku, T. S. M. & Ismail, A. (2017). Programming learning requirements based on multi perspectives. *International Journal of Electrical and Computer Engineering*, 7(3), 1299-1307. <http://doi.org/10.11591/ijece.v7i3.pp1299-1307>
- Khasawneh, E., Gosling, C. & Williams, B. (2021). What impact does maths anxiety have on university students?. *BMC psychology*, 9(1), 1-9. <https://doi.org/10.1186/s40359-021-00537-2>
- Kim, T. H. & White, H. (2004). On more robust estimation of skewness and kurtosis. *Finance Research Letters*, 1(1), 56-73. [https://doi.org/10.1016/S1544-6123\(03\)00003-5](https://doi.org/10.1016/S1544-6123(03)00003-5)
- Kivunja, C. & Kuyini, A. (2017). Understanding and applying research paradigms in educational contexts. *International Journal of Higher Education*, 6(5), 26-41. <https://doi.org/10.5430/ijhe.v6n5p26>
- Knox, D., Wolz, U., Joyce, D., Koffman, E., Krone, J., Laribi, A., Myers, J. P., Proulx, V. K., Reek, K. (1996). Use of laboratories in computer science education: guidelines for good practice: report of the working group on computing laboratories. *ACM Sigcse Outlook*, 24(3), 167-181. <https://doi.org/10.1145/237466.237644>
- Kölling, M. (1999). The problem of teaching object-oriented programming, part 1: languages. *Journal of Object-oriented Programming*, 11(8), 8-15.
- Kolb, D. (2015). *Experiential learning: experience as the source of learning and development* (2nd ed.). Pearson.
- Kolb, A. & Kolb, D. (2018). Eight important things to know about the experiential learning cycle. *Australian Educational Leader*, 40(3), 8-14.
- Kovács, L. I. (2019). Gesture-driven LEGO robots. *The Journal of Sapientia Hungarian University of Transylvania*, 11(1), 80-94. <https://doi.org/10.2478/ausi-2019-0006>

- Krueger, R. A. & Casey, M. A. (2015). Focus group interviewing. In K. E. Newcomer, H. P. Hatry & J. S. Wholey (Eds.), *Handbook of practical program evaluation* (pp. 506-534). John Wiley and Sons.
- Kühn, T. & Cazzola, W. (2016). Apples and oranges: comparing top-down and bottom-up language product lines. In H. Mei (Ed.), *SPLC '16: Proceedings of the 20th International Systems and Software Product Line Conference* (pp. 50-59). Association for Computing Machinery. <https://doi.org/10.1145/2934466.2934470>
- Kucuk, S. & Sisman, B. (2017). Behavioral patterns of elementary students and teachers in one-to-one robotics instruction. *Computers & Education*, 111(1), 31-43. <https://doi.org/10.1016/j.compedu.2017.04.002>
- Kuncoro, R. D. K., Arifudin, R. & Sugiharti, E. (2018). Implementation of NXT 2.0 Mindstorm robot sensors on mobile education for students. In *Integrating knowledge for future sustainable development*. International Summit on Science Technology and Humanity (ISETH 2018) (pp. 125-130). Universitas Muhammadiyah Surakarta. <https://publikasiilmiah.ums.ac.id/bitstream/handle/11617/11668/15.pdf?sequence=1&isAllowed=y>
- Kurdi, H. A. (2013). Review on aspect oriented programming. *International Journal of Advanced Computer Science and Applications*, 4(9), 22-27. <https://doi.org/10.14569/IJACSA.2013.040904>
- Kurebayashi, S., Kamada, T. & Kanemune, S. (2006). Learning computer programming with autonomous robots. In R. T. Mittermeir (Ed.), *Informatics education – the bridge between using and understanding computers*. Proceedings of the International Conference in Informatics in Secondary Schools – Evolution and Perspectives, ISSEP 2006, Vilnius, Lithuania, November 7-11 (pp. 138-149). Springer. https://doi.org/10.1007/11915355_13
- Kuutti, K. (1996). Activity theory as a potential framework for human-computer interaction research. In B. A. Nardi (Ed.), *Ontext and oniciousness: activity theory and human computer interaction* (pp.17-44). MIT Press.

- Law, L. & Fong, N. (2020). Applying partial least squares structural equation modeling (PLS-SEM) in an investigation of undergraduate students' learning transfer of academic English. *Journal of English for Academic Purposes*, 46(2020), 1-22.
<https://doi.org/10.1016/j.jeap.2020.100884>
- Lahtinen, E., Ala-Mutka, K. & Järvinen, H. M. (2005). A study of the difficulties of novice programmers. *ACM Sigcse Bulletin*, 37(3), 14-18.
<https://doi.org/10.1145/1067445.1067453>
- LaForce, M., Noble, E. & Blackwell, C. (2017). Problem-based learning (PBL) and student interest in STEM careers: the roles of motivation and ability beliefs. *Education Sciences*, 7(4), 92. <https://doi.org/10.3390/educsci7040092>
- Lategan, T. (2020, May 19). *Digital skills South African graduates will need in 2020*. Business Chief. <https://businesschief.eu/leadership-and-strategy/digital-skills-south-african-graduates-will-need-2020>
- Lee, V. C. S., Yu, Y. T., Tang, C. M., Wong, T. L. & Poon, C. K. (2018). A virtual debugging advisor for supporting learning in computer programming courses. *Journal of Computer Assisted Learning*, 34(1), 243-258. <https://doi.org/10.1111/jcal.12238>
- Leyk, T., McInvale, R. & Chen, L. (2017). *Structured peer learning program - an innovative approach to computer science education*. Cornell University.
<https://arxiv.org/pdf/1703.04174.pdf>
- Lie, J., Hauge, I. & Meany, T. (2017). Computer programming in the lower secondary classroom: learning mathematics. *Italian Journal of Educational Technology*, 25(2), 27-35. <https://doi.org/10.17471/2499-4324/911>
- Lin, H. M., Lee, M. H., Liang, J. C., Chang, H. Y., Huang, P. & Tsai, C. C. (2020). A review of using partial least square structural equation modeling in e-learning research. *British Journal of Educational Technology*, 51(4), 1354-1372.
<https://doi.org/10.1111/bjet.12890>
- Lin, H. T. & Kuo, T. H. (2010). Teaching programming technique with edutainment robot construction. In V. Mahadevan & G. S. Tomar (Eds.), *Education technology and*

- computer*. Proceedings of the 2nd International Conference on Education Technology and Computer (pp. 226-229). IEEE. <https://doi.org/10.1109/ICETC.2010.5529557>
- Lincoln, Y. S. & Guba, E. G. (1985). *Naturalistic inquiry*. Sage.
- Lincoln, Y. S., Lynham, S. A. & Guba, E. G. (2011). Paradigmatic controversies, contradictions, and emerging confluences, revisited. In N. K. Denzin & Y. S. Lincoln (Eds.), *The Sage handbook of qualitative research* (4th ed., pp. 97-128). Sage.
- Lions, S. & Peña, M. (2016). Reading comprehension in Latin America: difficulties and possible interventions. *New Directions for Child and Adolescent Development*, 2016(152), 71-84. <https://doi.org/10.1002/cad.20158>
- Liu, Y., Sun, M. & Chen, Y. (2016). Teaching guidance in programming courses from procedure-oriented to object-oriented. In X. Xiao, S.-B. Tsai & R. Feng (Eds.), *Proceedings of the 2nd International Conference on Social Science and Higher Education* (pp. 250-253). Atlantis Press. <https://doi.org/10.2991/icshe-16.2016.80>
- Lo, C. A., Lin, Y. T. & Wu, C. C. (2015). Which programming language should students learn first? A comparison of Java and Python. In J. E. Guerrero (Ed.), *Proceedings 2015 International Conference on Learning and Teaching in Computing and Engineering LaTiCE 2015* (pp. 225-226). IEEE. <https://scholar.lib.ntnu.edu.tw/en/publications/which-programming-language-should-students-learn-first-a-comparis>
- Lye, S. Y. & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12?. *Computers in Human Behavior*, 41(1), 51-61. <https://doi.org/10.1016/j.chb.2014.09.012>
- Maisiri, W. & Van Dyk, L. (2021). Industry 4.0 skills: a perspective of the South African manufacturing industry. *SA Journal of Human Resource Management*, 19(0), a1416. <https://doi.org/10.4102/sajhrm.v19i0.1416>
- Malinga, S. (2021, February 9). *Software developers in high demand in SA*. ITWEB. <https://www.itweb.co.za/content/4r1ly7RblGE7pmda>
- Maloney, J., Resnick, M., Rusk, N., Silverman, B. & Eastmond, E. (2010). The Scratch programming environment. *ACM Transactions on Computing Education*, 10(4), 1-15. <https://doi.org/10.1145/1868358.1868363>

- Mannila, L. & De Raadt, M. (2006). An objective comparison of languages for teaching introductory programming. In A. Berglund (Ed.), *Baltic Sea '06: Proceedings of the 6th Baltic Sea Conference on Computing Education Research: Koli Calling 2006* (pp. 32-37). Association for Computing Machinery <https://doi.org/10.1145/1315803.1315811>
- Mardia, K. V. (1970) Measures of multivariate skewness and kurtosis with applications. *Biometrika*, 57(1), 519-530. <https://doi.org/10.2307/2334770>
- Mardia, K. V. (1974). Applications of some measures of multivariate skewness and kurtosis in testing normality and robustness studies. *Sankhya: The Indian Journal of Statistics*, 36(2), 115-128.
- Margulieux, L. E., Catrambone, R. & Guzdial, M. (2016). Employing subgoals in computer programming education. *Computer Science Education*, 26(1), 44-67. <https://doi.org/10.1080/08993408.2016.1144429>
- Martin, N. L., & Soares, A. L. (2017). The introductory programming course issues in information systems. *International Association for Computer Information Systems*, 16(3), 128-137. <https://doi.org/10.4018/978-1-5225-1034-5.ch005>
- Mead, J., Gray, S., Hamer, J., James, R., Sorva, J., Clair, C. S. & Thomas, L. (2006). A cognitive approach to identifying measurable milestones for programming skill acquisition. *ACM SIGCSE Bulletin*, 38(4), 182-194. <https://doi.org/10.1145/1189215.1189185>
- Memon, A. H. & Rahman, I. A. (2014). SEM-PLS analysis of inhibiting factors of cost performance for large construction projects in Malaysia: perspective of clients and consultants. *The Scientific World Journal*, 2014(1), 1-9. <https://doi.org/10.1155/2014/165158>
- Mendes, A. J., Paquete, L., Cardoso, A. & Gomes, A. (2012). Increasing student commitment in introductory programming learning. In *Soaring to new heights in engineering education: 2012 Frontiers in Education Conference Proceedings* (pp. 82-87). IEEE. <https://doi.org/10.1109/FIE.2012.6462486>
- Merkouris, A., Chorianopoulos, K. & Kameas, A. (2017). Teaching programming in secondary education through embodied computing platforms: robotics and

- wearables. *ACM Transactions on Computing Education (TOCE)*, 17(2), 1-22.
<https://doi.org/10.1145/3025013>
- Merriam, S. (1998). *Qualitative research and case study applications in education* (2nd ed.). Jossey-Bass.
- Miles, B. (2016, January 7). *Making the links between computing and mathematics*. An Open Mind. <http://milesberry.net/2016/01/making-the-links-between-computing-and-mathematics/>
- Milková, E. (2012). Development of algorithmic thinking and imagination: base of programming skills. *Proceedings of the 16th WSEAS International Conference on Communications and Computers* (pp. 68-72).
- Misfeldt, M. & Ejsing-Duun, S. (2015). Learning mathematics through programming: an instrumental approach to potentials and pitfalls. In K. Krainer & N. Vondrová (Eds.), *Research in Mathematics Education. Proceedings of the Ninth Congress of the European Society for Research in Mathematics Education (CERME9)* (pp. 2524-2530). Charles University in Prague, Faculty of Education; ERME <https://hal.archives-ouvertes.fr/hal-01289367/document>
- Moniruzzaman, M., Zishan, M. S. R., Rahman, S., Mahmud, S. & Shaha, A. (2018). Design and implementation of urban search and rescue robot. *International Journal of Engineering and Manufacturing (IJEM)*, 8(2), 12-20.
<https://doi.org/10.5815/ijem.2018.02.02>
- Morgan, D. L. (2014). Pragmatism as a paradigm for social research. *Qualitative Inquiry*, 20(8), 1045-1053. <https://doi.org/10.1177/1077800413513733>
- Nanz, S. & Furia, C. A. (2015). A comparative study of programming languages in rosetta code. In P. Kellenberger (Ed.), *IEEE/ACM 37th IEEE International Conference on Software Engineering* (pp. 778-788). IEEE. doi: 10.1109/ICSE.2015.90
- Nasrawt, Z. O. & Lam, M. O. (2019). Less-Java, more learning: language design for introductory programming. *Journal of Computing Sciences in Colleges*, 34(3), 64-72.

- National Council of Teachers of Mathematics. (2016). *Computer science and mathematics education: a position of the National Council of Teachers of Mathematics*.
https://www.nctm.org/uploadedFiles/Standards_and_Postions/Position_Statements/Computer_science_and_math_ed_022416.pdf
- Ni, L. & Guzdial, M. (2012). Who am I?: understanding high school computer science teachers' professional identity. In *SIGCSE '12: Proceedings of the 43rd ACM technical symposium on Computer Science Education* (pp. 499-504). Association for Computing Machinery. <https://doi.org/10.1145/2157136.2157283>
- Nieveen, N. (2009). A key role for formative evaluation in educational design research. In T. Plomp & N. Nieveen (Eds.), *An introduction to educational design research* (pp. 89-101). SLO.
- Nugent, G., Barker, B., Grandgenett, N. & Adamchuk, V. (2009). The use of digital manipulatives in k-12: robotics, GPS/GIS and programming. In *2009 39th IEEE Frontiers in Education Conference*, (pp. 1-6). IEEE.
<https://doi.org/10.1109/FIE.2009.5350828>
- Nunnally, J.C. (1978). *Psychometric theory* (2nd ed.). McGraw-Hill.
- Olsson, M., Mozelius, P. & Collin, J. (2015). Visualisation and gamification of e-learning and programming education. *Electronic Journal of e-Learning*, 13(6), 441-454.
- Osanloo, A. & Grant, C. (2016). Understanding, selecting, and integrating a theoretical framework in dissertation research: creating the blueprint for your “house”. *Administrative Issues Journal: Connecting Education, Practice, and Research*, 4(2), 12-16. <https://doi.org/10.5929/2014.4.2.9>
- Othman, Z., Abdullah, N. A., Chin, K. Y., Shahrin, F. F. W., Ahmad, S. S. & Kasmin, F. (2018). Comparison on cloud image classification for thrash collecting LEGO Mindstorms EV3 robot. *International Journal of Human and Technology Interaction (IJHaTI)*, 2(1), 29-34.
- Owens, K., Edmonds-Wathen, C. & Bino, V. (2015). Bringing ethnomathematics to elementary school teachers in Papua New Guinea: a design-based research project. *Revista Latinoamericana de Etnomatematica*, 8(2), 32-52.

- Padmanabhuni, V. V. K., Tadiparthi, H. P. & Muralidhar Yanamadala, S. M. (2012). Effective pair programming practice - an experimental study. *Journal of Emerging Trends in Computing and Information Sciences*, 3(4), 471-479.
http://cisjournal.org/journalofcomputing/archive/vol3no4/vol3no4_2.pdf
- Palalas, A. & Wark, N. (2017). Design principles for an adult literacy mobile learning solution. In F. Loizides, G. Papadopoulos & N. Souleles (Eds.), *mLearn 2017: Proceedings of the 16th World Conference on Mobile and Contextual Learning* (pp. 1-8). Association for Computing Machinery. <https://doi.org/10.1145/3136907.3136934>
- Papert, S. (1980). *Mindstorms: children, computers, and powerful ideas*. Harvester Press.
- Pérez, E. S. & López, F. J. (2019). An ultra-low cost line follower robot as educational tool for teaching programming and circuit's foundations. *Computer Applications in Engineering Education*, 27(2), 288-302. <https://doi.org/10.1002/cae.22074>
- Pea, R. & Kurland, M. (1984). On the cognitive effects of learning computer programming. *New Ideas Psychology*, 2(2), 137-168. [https://doi.org/10.1016/0732-118X\(84\)90018-7](https://doi.org/10.1016/0732-118X(84)90018-7)
- Peng, D. X. & Lai, F. (2012). Using partial least squares in operations management research: a practical guideline and summary of past research. *Journal of Operations Management*, 30(6), 467-480. <https://doi.org/10.1016/j.jom.2012.06.002>
- Plomp, T. (2009). Educational design research: an introduction. In T. Plomp & N. Nieveen (Eds.), *An introduction to educational design research* (pp. 9-35). SLO.
- Plomp T. (2013). Educational design research: an introduction. In T. Plomp & N. Nieveen (Eds.), *Educational design research: part A: an introduction* (pp. 10-51). SLO.
- Plomp, T. & Nieveen, N. (2013). References and sources on educational design research. In T. Plomp & N. Nieveen (Eds.), *Educational research design* (pp. 170-199). SLO.
- Plonka, L., Sharp, H., Van der Linden, J. & Dittrich, Y. (2015). Knowledge transfer in pair programming: an in-depth analysis. *International Journal of Human-Computer Studies*, 73(1), 66-78. <https://doi.org/10.1016/j.ijhcs.2014.09.001>
- Porter, L., Bouvier, D., Cutts, Q., Grissom, S., Lee, C., McCartney, R., Zingaro, D. & Simon, B. (2016). A multi-institutional study of peer instruction in introductory computing. In

- SIGCSE '16: *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, (pp. 358-363). Association for Computing Machinery.
<https://doi.org/10.1145/2839509.2844642>
- Pukelsheim, F. (1994). The three sigma rule. *The American Statistician*, 48(2), 88-91.
<https://doi.org/10.1080/00031305.1994.10476030>
- Rajala, T., Laakso, M. J., Kaila, E. & Salakoski, T. (2008). Effectiveness of program visualisation: a case study with the ViLLE tool. *Journal of Information Technology Education*, 7(1), 15-32. <http://jite.org/documents/Vol7/JITEv7p061-080Lee332.pdf>
- Ramli, N. A., Latan, H. & Nartea, G. V. (2018). Why should PLS-SEM be used rather than regression? Evidence from the capital structure perspective. In K. N. Avkiran & C. M. Ringle (Eds.), *Partial least squares structural equation modeling* (pp. 171-209). Springer.
- Razali, N. M. & Wah, Y. B. (2011). Power comparisons of Shapiro-Wilk, Kolmogorov-Smirnov, Lilliefors and Anderson-Darling tests. *Journal of statistical modeling and analytics*, 2(1), 21-33.
- Reddy, V., Visser, M., Winnaar, L., Arends, F., Juan, A., Prinsloo, C. & Isadale, K. (2016). *TIMSS 2015: highlights of mathematics and science achievement of grade 9 South African learners: nurturing green shoots*. Human Sciences Research Council.
<http://www.hsrc.ac.za/en/research-data/view/8456>
- Reeves, T. (2006). Design research from a technology perspective. In J. van den Akker, K. Gravemeijer, S. McKenney & N. Nieveen (Eds.), *Educational design research* (pp. 64-78). Routledge.
- Reeves, T. C. (2000). Enhancing the worth of instructional technology research through “design experiments” and other development research strategies. *International Perspectives on Instructional Technology Research for the 21st Century*, 27(1), 1-15.
- Resnick, M., Ocko, S. & Papert, S. (1988). Lego, logo, and design. *Children's Environments Quarterly*, 5(4), 14-18.

- Revez, J. & Borges, L.C. (2018). Pragmatic paradigm in information science research: a literature review. *Qualitative and Quantitative Methods in Libraries (QQML)*, 7(1), 583-593.
- Ringle, C. M. (2015). *HTMT discriminant validity*. SmartPLS.
<https://forum.smartpls.com/viewtopic.php?t=3616>
- Robert, W. S. (2017). *Concepts of programming languages, global edition* (11th ed.). Pearson.
- Roberts, P. (2009). Abstract thinking: a predictor of modelling ability?. In *Proceedings of the Educators Symposium of the ACM/IEEE 12th International Conference on Model Driven Engineering Languages and Systems* (pp. 753-754).
- Robins, A., Rountree, J. & Rountree, N. (2003). Learning and teaching programming: a review and discussion. *Computer Science Education*, 13(2), 137-172.
<https://doi.org/10.1076/csed.13.2.137.14200>
- Rodríguez, F. J., Price, K. M., Isaac, J., Boyer, K. E. & Gardner-McCune, C. (2017). How block categories affect learner satisfaction with a block-based programming interface. In A.Z. Henley, P. Rogers & A. Sarma (Eds.), *Proceedings IEEE Symposium on Visual Languages and Human-Centric Computing* (pp. 201-205). IEEE.
<https://www.computer.org/csdl/proceedings-article/vlhcc/2017/08103468/17D45XoXP6h>
- Romero, M., Lepage, A. & Lille, B. (2017). Computational thinking development through creative programming in higher education. *International Journal of Educational Technology in Higher Education*, 14(1), 42. <https://doi.org/10.1186/s41239-017-0080-z>
- Rorty, R. (1991). *Objectivity, relativism and truth: philosophical papers*. Cambridge University Press.
- Rosen, K. H. (2018). *Handbook of discrete and combinatorial mathematics* (2nd ed.). Taylor & Francis Group.
- Royal Society Report. (2012). *Shut down or restart? The way forward for computing in UK schools*. <http://www.royal.society.org/education/policy>

- Rubio Escudero, M. A., Mañoso Hierro, C. M. & Pérez de Madrid y Pablo, A. (2013). Using arduino to enhance computer programming courses in science and engineering. In L. Gómez Chova, A. López Martínez & I. Candel Torres (Eds.), *EDULEARN13 proceedings: 5th International Conference on Education and New Learning Technologies* (pp. 1-3). International Association of Technology, Education and Development.
- Russell, I., Rosiene, C. P. & Gold, A. (2020). A CS course for non-majors based on the Arduino platform. In J. Zhang, M. Sherriff, S. Heckman, P. Cutter & A. Monge (Eds.), *SIGCSE '20: The 51st ACM Technical Symposium on Computer Science Education* (pp. 1309-1309). Association for Computing Machinery.
<https://doi.org/10.1145/3328778.3366955>
- Sadler, I. (2013). The role of self-confidence in learning to teach in higher education. *Innovations in Education and Teaching International*, 50(2), 157-166.
<https://doi.org/10.1080/14703297.2012.760777>
- Saeli, M., Perrenet, J., Jochems, W.M.G. & Zwaneveld, B. (2011). Teaching programming in secondary school: a pedagogical content knowledge perspective. *Informatics in Education*, 10(1), 73-88. <https://doi.org/10.15388/infedu.2011.06>
- Samuel, M. S. (2017). An insight into programming paradigms and their programming languages. *Journal of Applied Technology and Innovation*, 1(1), 37-57.
<https://doi.org/10.5120/ijca2020920172>
- Sebesta, R. (2004). *Concepts of programming languages* (6th ed.). Pearson/Addison-Wesley.
- Sentence, S. & Waite, J. (2017). PRIMM: exploring pedagogical approaches for teaching text-based programming in school. In E. Barendsen & P. Hubwieser (Eds.), *WiPSCE '17: Proceedings of the 12th Workshop in Primary and Secondary Computing Education* (pp. 113-114). Association for Computing Machinery.
<https://doi.org/10.1145/3137065.3137084>
- Shapiro, S. S. & Wilk, M. B. (1965). An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4), 591–611. <https://doi.org/10.2307/2333709>

- Sharma, P. (2018). *Programming in Python: learn the powerful object-oriented programming*. BPB Publications.
- Shein, E. (2015). Python for beginners. *Communications of the ACM*, 58(3), 19-21.
<https://doi.org/10.1145/2716560>
- Simanjuntak, M. V., Abdullah, A. G. & Maulana, I. (2018). Promoting middle school students' abstract-thinking ability through cognitive apprenticeship instruction in mathematics learning. *Journal of Physics: Conference Series*, 948(12051), 0-4.
<https://doi.org/10.1088/1742-6596/948/1/012051>
- Sleeman, D. (1986). The challenges of teaching computer programming. *Communications of the ACM*, 29(9), 840-841. <https://doi.org/10.1145/6592.214913>
- Sobral, S. R. (2021). Flipped Classrooms for Introductory Computer Programming Courses. *International Journal of Information and Education Technology*, 11(4), 178-183.
<https://doi.org/10.18178/ijiet.2021.11.4.1508>
- Soloway, E. (1986). Learning to program learning to construct mechanisms and explanations. *Communications of the ACM*, 29(9), 850-858. <https://doi.org/10.1145/6592.6594>
- SONA. (2019). *State of the Nation Address 2019*. <https://www.gov.za/speeches/president-cyril-ramaphosa-2019-state-nation-address-7-feb-2019-0000>
- Soykan, F. & Kanbul, S. (2018). Analysing K12 students' self-efficacy regarding coding education. *TEM Journal*, 7(1), 182-187. <https://doi.org/10.18421/TEM71-22>
- Stellenbosch University handbook (2019). *General Rules and Policies Stellenbosch University*.
<https://www.sun.ac.za/english/Documents/Yearbooks/Current/GeneralPoliciesAndRules.pdf>
- Stone, M. (1974). Cross-validation and multinomial prediction. *Biometrika*, 61(3), 509-515.
<https://doi.org/10.2307/2334733>
- Stueben, M. (2018). *Good habits for great coding improving programming skills with examples in Python*. Apress.

- Summerfield, M. (2010). *Programming in Python 3: a complete introduction to the Python language*. Addison-Wesley Professional.
- Tanrikulu, E. & Schaefer, B. C. (2011). The users who touched the ceiling of scratch. *Procedia-Social and Behavioral Sciences*, 28(1), 764–769.
<https://doi.org/10.1016/j.sbspro.2011.11.140>
- Techapalokul, P. & Tilevich, E. (2017). Understanding recurring quality problems and their impact on code sharing in block-based software. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (pp. 43-51). IEEE.
<https://doi.org/10.1109/VLHCC.2017.8103449>
- Teddle, C. & Tashakkori, A. (2011). Mixed methods research: contemporary issues in an emerging field. In N. K. Denzin & Y. S. Lincoln (Eds.), *The SAGE handbook of qualitative research* (pp. 285-299). Sage.
- Tenenhaus, M., Vinzi, V. E., Chatelin, Y. M. & Lauro, C. (2005). PLS path modeling. *Computational Statistics & Data Analysis*, 48(1), 159-205.
<https://doi.org/10.1016/j.csda.2004.03.005>
- Toh, L. P. E., Causo, A., Tzuo, P. W., Chen, I. M. & Yeo, S. H. (2016). A review on the use of robots in education and young children. *Educational Technology & Society*, 19(2), 148–163. <https://doi.org/10.2307/jeductechsoci.19.2.148>
- Thomas, E. & Georg, G. (1995). The complexity of logic-based abduction. *Journal of the ACM*, 42(1), 3-42. <https://doi.org/10.1145/200836.200838>
- Thota, N. & Whitfield, R. (2010). Holistic approach to learning and teaching introductory object-oriented programming. *Computer Science Education*, 20(2), 103-127.
<https://doi.org/10.1080/08993408.2010.486260>
- Together We Pass (2019). *How to calculate your APS*.
[https://togetherwepass.co.za/how-to-calculate-your-aps/#:~:text=The%20minimum%20requirements%20for%20the,points%20\(some%20exceptions%20may%20apply\)](https://togetherwepass.co.za/how-to-calculate-your-aps/#:~:text=The%20minimum%20requirements%20for%20the,points%20(some%20exceptions%20may%20apply))
- Tollervey, N. (2015). *Python in education*. O'Reilly Media.

- Topalli, D. & Cagiltay, N. E. (2018). Improving programming skills in engineering education through problem-based game projects with Scratch. *Computers & Education*, 120(1), 64-74. <https://doi.org/10.1016/j.compedu.2018.01.011>
- Tsai, M. J., Wang, C. Y. & Hsu, P. F. (2019). Developing the computer programming self-efficacy scale for computer literacy education. *Journal of Educational Computing Research*, 56(8), 1345-1360. <https://doi.org/10.1177/0735633117746747>
- Turchi, T., Fogli, D. & Malizia, A. (2019). Fostering computational thinking through collaborative game-based learning. *Multimedia Tools and Applications*, 78(10), 13649-13673. <https://doi.org/10.1007/s11042-019-7229-9>
- Utting, I., Tew, A. E., McCracken, M., Thomas, L., Bouvier, D., Frye, R., Paterson, J., Caspersen, M. E., Kolikant, Y. B-D., Sorva, J. & Wilusz, T. (2013). A fresh look at novice programmers' performance and their teachers' expectations. In J. Carter (Ed.), *ITiCSE -WGR '13: Proceedings of the ITiCSE Working Group Reports Conference on Innovation and Technology in Computer Science Education-Working Group Reports* (pp. 15-32). Association for Computing Machinery. <https://doi.org/10.1145/2543882.2543884>
- University of Cape Town handbook (2019). *General Rules and Policies*. <http://www.students.uct.ac.za/students/study/handbooks/archive/2019>
- University of Kwa-Zulu Natal (2019). *Handbook 2019*. http://saa.ukzn.ac.za/Forms_proce/Handbooks.aspx
- University of Witwatersrand (2019). *Rules and Syllabuses*. <https://www.wits.ac.za/students/academic-matters/rules-and-syllabuses/>
- Varney, M. W., Janoudi, A., Aslam, D. M. & Graham, D. (2012). Building young engineers: TASEM for third graders in Woodcreek Magnet Elementary School. *IEEE transactions on education*, 55(1), 78-82. <https://doi.org/10.1109/TE.2011.2131143>
- Voogt, J., Fisser, P., Good, J., Mishra, P. & Yadav, A. (2015). Computational thinking in compulsory education: Towards an agenda for research and practice. *Education and Information Technologies*. 20(4), 715-728. <https://doi.org/10.1007/s10639-015-9412-6>

- Vygotsky, L. S. (1978). *Mind and society: the development of higher mental processes*. Harvard University Press.
- Warne, R. T., Yoon, M. & Price, C. J. (2014). Exploring the various interpretations of “test bias”. *Cultural Diversity and Ethnic Minority Psychology*, 20(4), 570. <https://doi.org/10.1037/a0036503>
- Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L. & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology*, 25(1), 127-147. <https://doi.org/10.1007/s10956-015-9581-5>
- Weller, C. (2017, October 10). Meet the first ever robot citizen - a humanoid named Sophia that once said it would destroy humans. *Business Insider*. <https://www.businessinsider.com/meet-the-first-robot-citizen-sophia-animatronic-humanoid-2017-10?IR=T>
- White, G., & Sivitanides, M. (2003). An empirical investigation of the relationship between success in mathematics and visual programming courses. *Journal of Information Systems Education*, 14(4), 409.
- White, G. & Sivitanides, M. (2005). Cognitive differences between procedural programming and object oriented programming. *Information Technology and Management*, 6(4), 333-350. <https://doi.org/10.1007/s10799-005-3899-2>
- Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33-35. <https://doi.org/10.1145/1118178.1118215>
- Xinogalos, S. Ivanović, M., Pitner, T. & Savić, M. (2017). Technology enhanced learning in programming courses - international perspective. *Education and Information Technologies: The Official Journal of the Ifip Technical Committee on Education*, 22(6), 2981-3003. <https://doi.org/10.1007/s10639-016-9565-y>
- Xinogalos, S., Pitner, T., Ivanović, M. & Savić, M. (2018). Students’ perspective on the first programming language: C-like or Pascal-like languages?. *Education and Information Technologies*, 23(1), 287-302. <https://doi.org/10.1007/s10639-017-9601-6>

- Yadin, A. (2011). Reducing the dropout rate in an introductory programming course. *ACM Inroads*, 2(4), 71-76. <https://doi.org/10.1145/2038876.2038894>
- Yadav, A., Gretter, S., Hambrusch, S. & Sands, P. (2016). Expanding computer science education in schools: understanding teacher experiences and challenges. *Computer Science Education*, 26(4), 235-254. <https://doi.org/10.1080/08993408.2016.1257418>
- Yilmaz I. & Koc, M. (2021). The consequences of robotics programming education on computational thinking skills: an intervention of the young engineer's workshop (yew). *Computer Applications in Engineering Education*, 29(1), 191-208. <https://doi.org/10.1002/cae.22321>
- Yin, R. K. (2014). *Case study research: design and methods*. Sage.
- Yoshiaki, M., Takashi, O., Manabu, S. & Sanshiro, S. (2015). Language migration in non-CS introductory programming through mutual language translation environment. In *SIGCSE '15: Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (pp. 185-190). Association for Computing Machinery. <https://doi.org/10.1145/2676723.2677230>
- Zainal, N. F. A., Shahrani, S., Yatim, N. F. M., Abd Rahman, R., Rahmat, M. & Latih, R. (2012). Students' perception and motivation towards programming. *Procedia-Social and Behavioral Sciences*, 59(2021), 277-286. <https://doi.org/10.1016/j.sbspro.2012.09.276>
- Zayour, I. & Hajjdiab, H. (2013). How much integrated development environments improve productivity?. *Journal of Software*, 8(10), 2425-2431. <https://doi.org/10.4304/jsw.8.10.2425-2431>
- Zhang, M. & Wan, Y. (2020). Improving learning experiences using LEGO Mindstorms EV3 robots in control systems course. *The International Journal of Electrical Engineering & Education*, 0(0), 1-23, <https://doi.org/10.1177/0020720920965873>
- Zuhud, D. A. Z. (2013). From programming sequential machines to parallel smart mobile devices: Bringing back the imperative paradigm to today's perspective. In J. Labadin, J. Minoi, D. NurFatimah, A. Iskandar & A. B. Masli (Eds.), *Information Technology in Asia*. Proceedings of the 8th International Conference on Information Technology in Asia (CITA) (pp. 1-7). <https://doi.org/10.1109/CITA.2013.6637580>

Zuhud, D. A. Z., Rahman, N. & Ismail, M. (2013). *A preliminary analysis on the shift of programming paradigms*. Proceedings of the 5th International Conference on Information and Communication Technology for the Muslim World (ICT4M), (pp. 1-5).
<https://doi.org/10.1109/ICT4M.2013.6518917>

Appendices

Intentionally blank

Appendix A - Permission from Registrar



29 October 2019

Mr Reginald Gerald Govender (SN 207501841)
School of Education
College of Humanities
Edgewood Campus
UKZN
Email: 207501841@stu.ukzn.ac.za

Dear Mr Govender

RE: PERMISSION TO CONDUCT RESEARCH

Gatekeeper's permission is hereby granted for you to conduct research at the University of KwaZulu-Natal (UKZN) towards your postgraduate studies, provided Ethical clearance has been obtained. We note the title of your research project is:

"Exploring the Use of Robotics in the Learning of Programming."

It is noted that you will be constituting your sample by handing out questionnaires and/or conducting interviews with students on the Edgewood campus.

Please ensure that the following appears on your notice/questionnaire:

- Ethical clearance number;
- Research title and details of the research, the researcher and the supervisor;
- Consent form is attached to the notice/questionnaire and to be signed by user before he/she fills in questionnaire;
- gatekeepers approval by the Registrar.

You are not authorized to contact staff and students using 'Microsoft Outlook' address book. Identity numbers and email addresses of individuals are not a matter of public record and are protected according to Section 14 of the South African Constitution, as well as the Protection of Public Information Act. For the release of such information over to yourself for research purposes, the University of KwaZulu-Natal will need express consent from the relevant data subjects. Data collected must be treated with due confidentiality and anonymity.

Yours sincerely

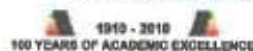
**DR KE CLELAND
REGISTRAR (ACTING)**

Office of the Registrar

Postal Address: Private Bag X54001, Durban, South Africa

Telephone: +27 (0) 31 260 8005/2206 Facsimile: +27 (0) 31 260 7824/2204 Email: registrar@ukzn.ac.za

Website: www.ukzn.ac.za



Founding Campuses: Edgewood Howard College Medical School Pietermaritzburg Westville

Appendix B - Informed consent letter

Dear Student

INFORMED CONSENT LETTER

My name is **Reginald G. Govender** I am a PhD candidate studying at the University of KwaZulu-Natal, Edgewood campus, South Africa. I am interested in exploring the use of robotics in the learning of computer programming. To gather the information, I am interested in asking you some questions.

Please note that:

- Your confidentiality is guaranteed as your inputs will not be attributed to you in person, but reported only as a population member opinion.
- Audio and video recording will take place.
- There will be surveys, questionnaires and an interview.
- Any information given by you cannot be used against you, and the collected data will be used for purposes of this research only.
- Data will be stored in a filing cabinet at UKZN and destroyed after 5 years.
- You have a choice to participate, not participate or stop participating in the research. You will not be prejudiced for taking such an action.
- Your involvement is purely for academic purposes only, and there are no financial benefits involved.
- The project will last for approximately five sessions for a period of five weeks.
- If you are willing to be interviewed, please indicate (by ticking as applicable) whether or not you are willing to allow the interview to be recorded by the following equipment:

EQUIPMENT	ACCEPT	DECLINE
Audio equipment	<input type="checkbox"/>	<input type="checkbox"/>
Video equipment	<input type="checkbox"/>	<input type="checkbox"/>

I can be contacted at:

Email: govenderR4@ukzn.ac.za
Phone number: +2731 260 3136

My supervisor is **Prof. D.W. Govender**
Email: govenderD50@ukzn.ac.za
Phone number: +2731 260 3428

You may also contact the Research Office through:
Ms P Ximba (HSSREC Research Office)
Email: ximbap@ukzn.ac.za
Tel: +2731 260 3587

Thank you for your contribution to this research.

DECLARATION

I
(full name)

hereby confirm that I understand the contents of this document and the nature of the research project, and I consent to participating in the research project.

.....
SIGNATURE OF PARTICIPANT

.....
DATE

.....
SIGNATURE OF PARENT/GUARDIAN
(If under 18 years of age)

.....
DATE

Appendix C - Ethical clearance



31 October 2019

Mr Reginald Gerald Govender (207501841)
School of Education
Edgewood Campus

Dear Mr Govender,

Protocol reference number: HSS/00000574/019M

Project title: Exploring the use of Robotics in the learning of programming

Approval Notification – Expedited Application

This letter serves to notify you that your application received on 16 August 2019 in connection with the above, was reviewed by the Humanities and Social Sciences Research Ethics Committee (HSSREC) and the protocol has been granted **FULL APPROVAL**

Any alteration/s to the approved research protocol i.e. Questionnaire/Interview Schedule, Informed Consent Form, Title of the Project, Location of the Study, Research Approach and Methods must be reviewed and approved through the amendment/modification prior to its implementation. In case you have further queries, please quote the above reference number. **PLEASE NOTE:** Research data should be securely stored in the discipline/department for a period of 5 years.

This approval is valid for one year from 31 October 2019.

To ensure uninterrupted approval of this study beyond the approval expiry date, a progress report must be submitted to the Research Office on the appropriate form 2 - 3 months before the expiry date. A close-out report to be submitted when study is finished.

Yours sincerely,

Professor Urmilla Bob
University Dean of Research

/ms

Humanities & Social Sciences Research Ethics Committee
Dr Rosemary Sibanda (Chair)
UKZN Research Ethics Office Westville Campus, Govan Mbeki Building
Postal Address: Private Bag X54001, Durban 4000
Website: <http://research.ukzn.ac.za/Research-Ethics/>

Founding Campuses: ■ Edgewood ■ Howard College ■ Medical School ■ Pietermaritzburg ■ Westville

INSPIRING GREATNESS

Appendix D - Turn- it- in Report

Exploring the use of Robotics in the learning of programming

ORIGINALITY REPORT

8%	6%	3%	3%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Universiti Teknologi MARA Student Paper	1%
2	www.tandfonline.com Internet Source	<1%
3	researchspace.ukzn.ac.za Internet Source	<1%
4	Submitted to The Scientific & Technological Research Council of Turkey (TUBITAK) Student Paper	<1%
5	my.unisa.ac.za Internet Source	<1%
6	Submitted to University of Economics Ho Chi Minh Student Paper	<1%
7	Submitted to University of Lancaster Student Paper	<1%
8	ijern.com Internet Source	<1%
9	link.springer.com	

Appendix E - Editor's Report

CONFIRMATION OF EDITING


25 November 2021

This letter serves to confirm that the following thesis has been language edited:

Exploring the use of Robotics in the learning of programming

by Reginald Gerald Govender

Kind Regards



XS Kyriacou (PhD)



Academic Editing and Writing Support

Cell 0614252802

Email xeniaxsk7@gmail.com

Appendix F – Pre-survey on computer programming (survey one)

WRITE YOUR
CODE HERE

Computer programming survey one

Please indicate how you feel about computer programming by the extent of agreement using the following words: Strongly disagree, Disagree, Undecided, Agree and Strongly agree. Put a tick only in one box for each statement. If you make a mistake put a cross through the marked box and then tick in the correct box.

No	Statement	Strongly disagree	Disagree	Undecided	agree	Strongly agree
1	I have an interest in programming					
2	I lack a basic mathematical background.					
3	I think programming is too technical.					
4	I can succeed in learning computer programming.					
5	I am good at problem solving.					
6	I think it would be interesting to use programming to solve problems.					
7	From my own understanding of programming, it is boring.					
8	My perception of programming is that it is difficult to learn.					
9	I think programming is hard.					
10	I have an interest in microcontrollers (robotic element).					

THANK YOU

Appendix G - Questionnaire one | Part A: Pre-test based on computational thinking

WRITE YOUR
CODE HERE

Questionnaire A

Instructions: Please write your code in the top right block. Complete the following questionnaire by filling in the blanks or placing a tick in the block.

Personal information:

A Year of current study:

1 st	2 nd	3 rd	4 th
-----------------	-----------------	-----------------	-----------------

B Age:

16-17	18-19	20-21	>21
-------	-------	-------	-----

C Phase specialisation:

ECD	Intermediate Senior	Senior FET	FET
-----	---------------------	------------	-----

D What is/are your subject specialisation/s?

Background:

E Year finished school:

<2008	2008-2010	>=2011
-------	-----------	--------

F Computer subjects enrolled at school (Gr10-12):

Computer Applications Technology	
Information Technology	
Did not do any Computer Subject	

G Do you have any prior computer programming experience?

(Examples: Online courses, workshops, etc.)

Yes	No
-----	----

H If response was YES to G- what programming language did you use:

I Choose **one** that best describes your school leaving Mathematics

Mathematical Literacy	
Pure Mathematics	
Mathematics SG	
Mathematics HG	
Vocational Mathematics	
No Mathematics	
Other	

If Other specify: _____

Multiple choice:

Choose the most appropriate answer by placing a tick or circling your response.

1. A computer program can be best described as:
A Is series of instructions
B Can be short or long
C Is written with code
D All of the above
2. If $\text{num1} \leftarrow 12$
 $\text{num2} \leftarrow \text{num1} \times 2$
 $\text{num3} \leftarrow \text{num2} \times 0$
 $\text{num1} \leftarrow \text{num1} \times 1$
The value of num1 is:
A 2
B 12
C 0
D 1
3. A variable in programming can be described as:
A Logical structure
B Fixed data
C Place holder
D All of the above
4. If the circumference of a circle is calculated by $c = \pi d$, where d represent's the diameter then the radius (r) is equal to
A $r = \pi r^2$
B $r = \frac{c}{2\pi}$
C $r = \frac{c}{2d}$
D $r = c\pi d$

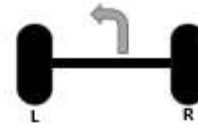
5. In which direction would the following wheels need to move in order to reverse. In the figure alongside the arrows show the reverse direction.
A Right wheel clockwise, Left wheel clockwise
B Right wheel anti-clockwise, Left wheel clockwise
C Right wheel anti-clockwise, Left wheel anti-clockwise
D Right wheel clockwise, Left wheel anti-clockwise



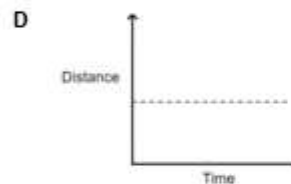
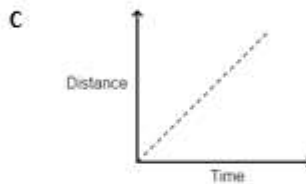
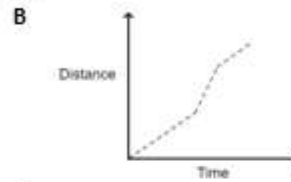
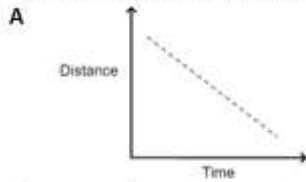
6. Which wheel would be able to cover a greater distance between two points?
A X
B Y
C Z
D It doesn't matter



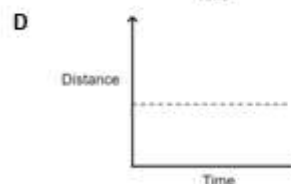
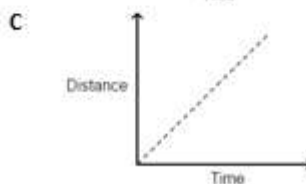
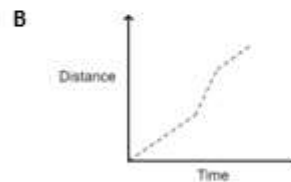
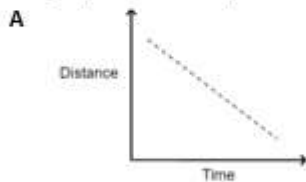
7. Which wheel would need to rotate if in order to make a left turn?
 A Left wheel
 B Right wheel
 C Left wheel and right wheel
 D Right wheel and left wheel



8. The graph that best represents a constant speed



9. The graph that best represents no movement



10. Which of the following lines (table alongside) can be deleted without affecting the result?

- A Line 1
 B Line 2
 C Line 3
 D Line 4

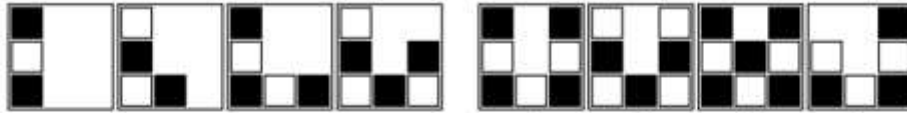
Line number	Statement
1	$W = 60$
2	$X = 70$
3	$Y = 80$
4	$Z = 95$
5	$Y = Y - Z$
6	INPUT W
7	$W = W + X$
8	OUTPUT (W + X)
9	OUTPUT Y

Appendix H - Questionnaire one | Part B: Abstract reasoning test

Matching patterns – 25 Questions

Answer as many questions as you can in 20 minutes. Circle the letter on the right which corresponds to the correct answer.

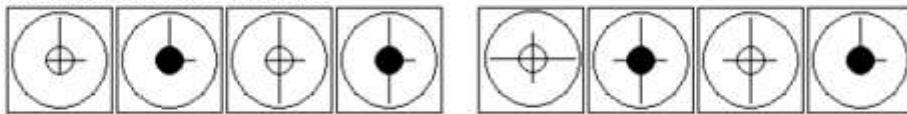
1. Which figure completes the series?



A B C D

A B C D

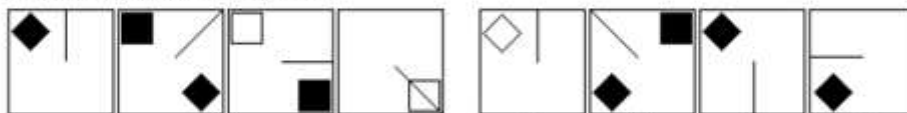
2. Which figure completes the series?



A B C D

A B C D

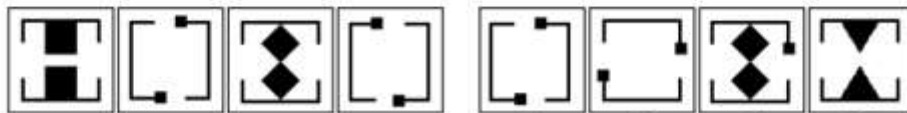
3. Which figure completes the series?



A B C D

A B C D

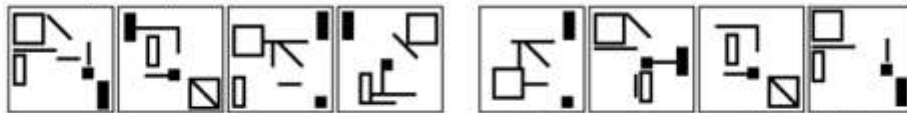
4. Which figure completes the series?



A B C D

A B C D

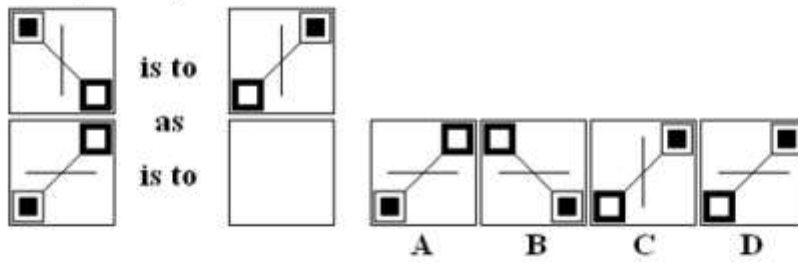
5. Which figure completes the series?



A B C D

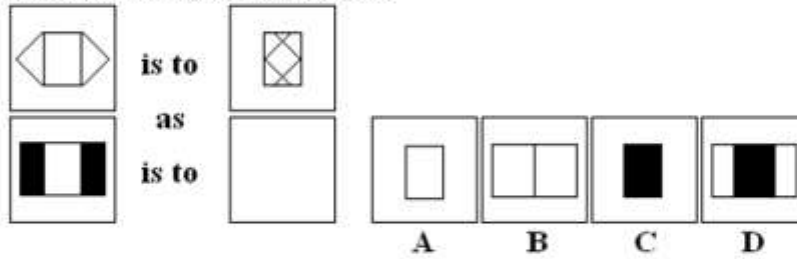
A B C D

6. Which figure completes the statement?



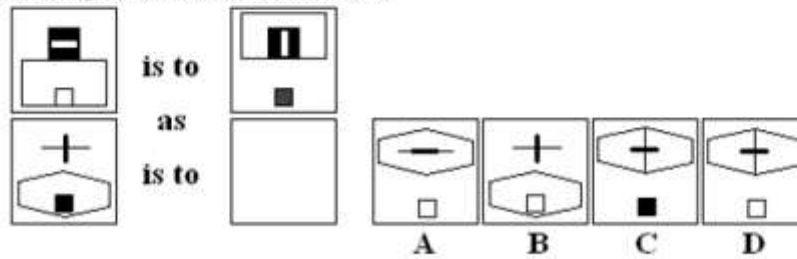
A B C D

7. Which figure completes the statement?



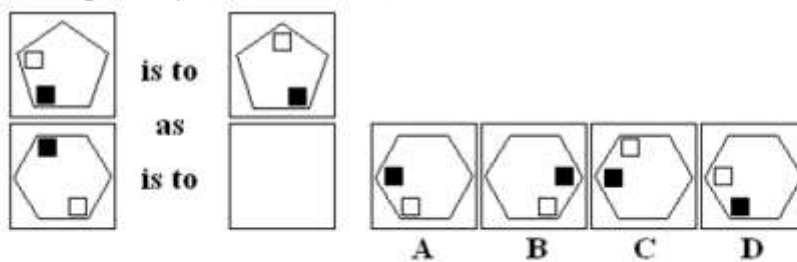
A B C D

8. Which figure completes the statement?



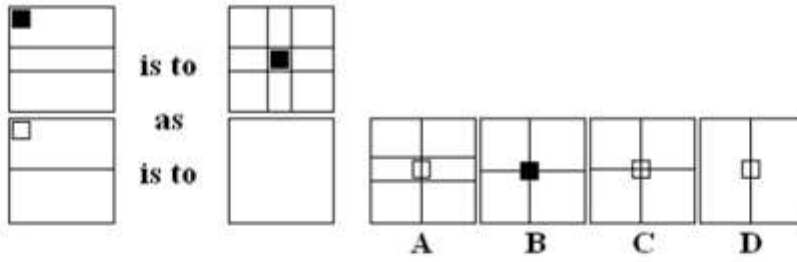
A B C D

9. Which figure completes the statement?



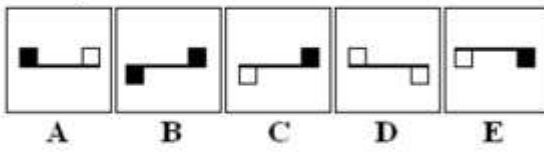
A B C D

10. Which figure completes the statement?



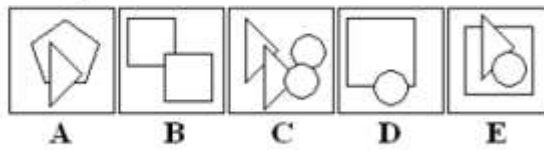
A B C D

11. Which figure is the odd one out?



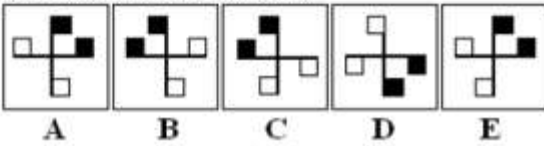
A B C D E

12. Which figure is the odd one out?



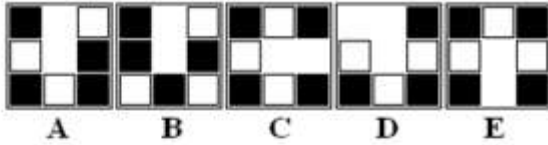
A B C D E

13. Which figure is the odd one out?



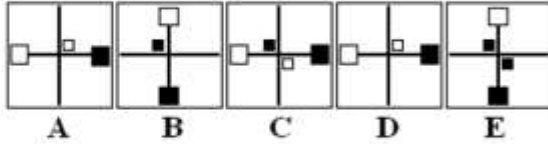
A B C D E

14. Which figure is the odd one out?



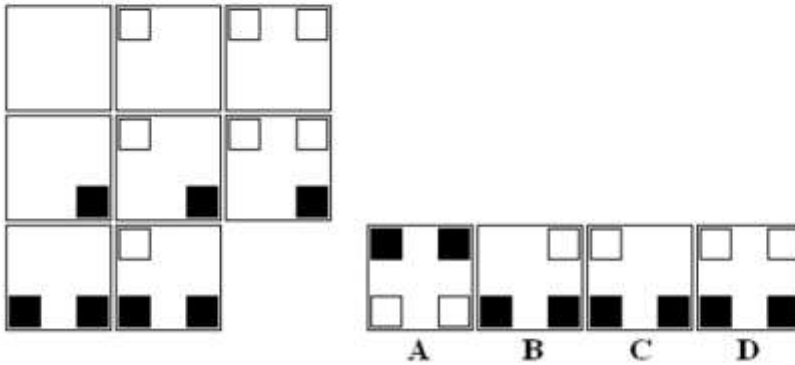
A B C D E

15. Which figure is the odd one out?



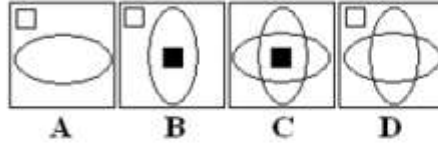
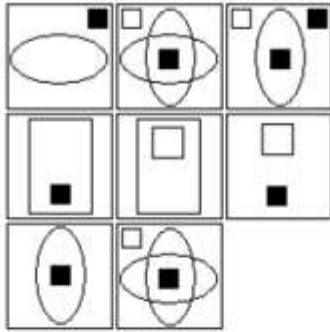
A B C D E

16. Which figure completes the series?



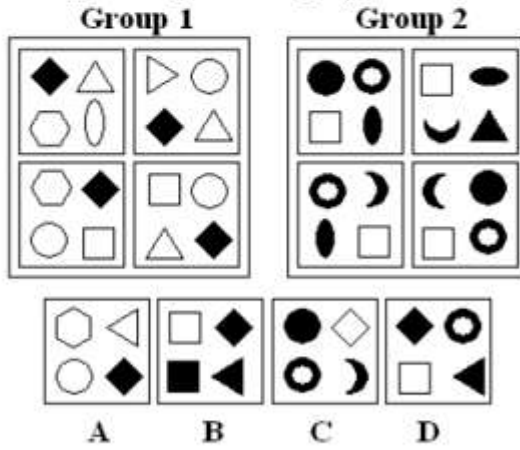
A B C D

17. Which figure completes the series?



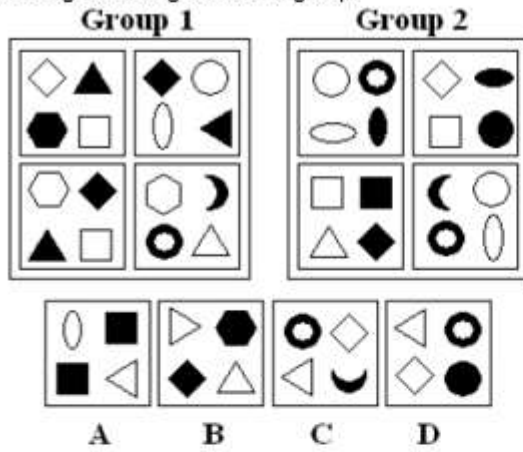
A B C D

18. Which figure belongs in neither group?



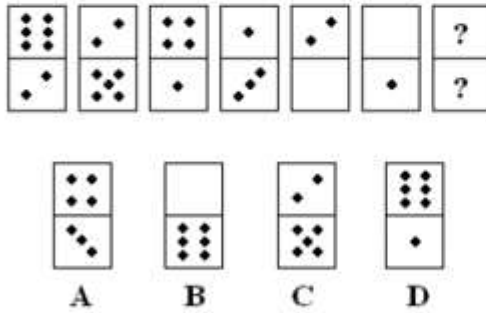
A B C D

19. Which figure belongs in neither group?



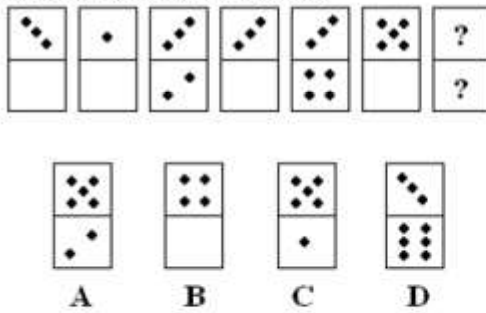
A B C D

20. Which figure is next in the series?



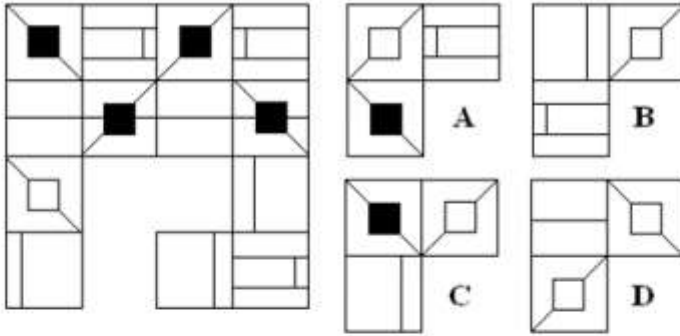
A B C D

21. Which figure is next in the series?



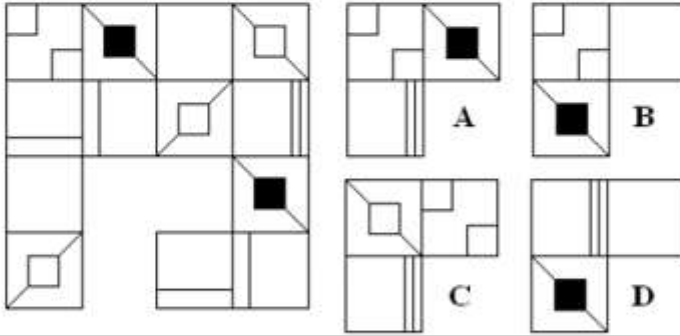
A B C D

22. Which figure completes the grid?



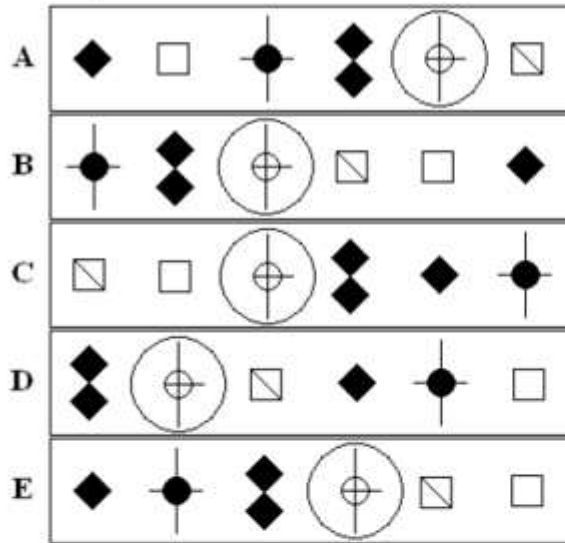
A B C D

23. Which figure completes the grid?



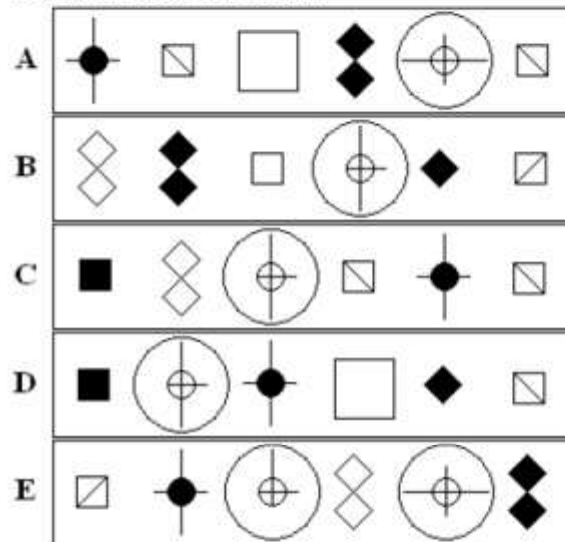
A B C D

24. Which figure is the odd one out?



A B C D E

25. Which figure is the odd one out?



A B C D E

Thank you

Appendix I – Pre workshop session

 YOU CAN ZOOM WITHIN YOUR BROWSER TO ENLARGE

© Copyright 2020 by Reginald G. Govender

Setting up the software: Arduino & Python

Make sure you are connected to the internet during the setup. If you have worked with Python or Arduino before make sure that you have uninstalled any previous versions before proceeding.

If you have a 64bit PC you should still be able to install 32bit versions of the software. If you encounter a problem contact: govenderR4@ukzn.ac.za

STEPS:

1. Installing Python

- i. Make sure to select "Add Python 3.8 to PATH" then click "Install Now". Refer to Figure 1



Figure 1: Start screen of Python installation

- ii. Accept/agree and click install to all windows that pop up. Note: Do not change any settings when installing.

iii. Successful installation screen Figure 2

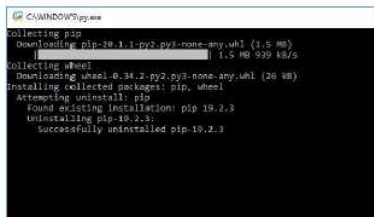


Figure 2: Screen shown on successful completion of Python

2. Activating pip

"pip" is a de facto standard package-management system used to install and manage software packages written in Python.

- i. Double click on the get-pip.py file (A blank might appear for a few seconds and disappear – this is OK)
- ii. Successful installation. Figure 3

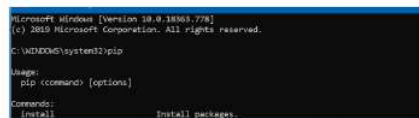


```
C:\WINDOWS\system32
collecting pip
downloading pip-20.1.1.py2.py3-none-any.whl (1.5 MB)
100%
collecting wheel
downloading wheel-0.34.2.py2.py3-none-any.whl (28 kB)
installing collected packages: pip, wheel
Attempting uninstall: pip
Found existing installation: pip 19.2.3
uninstalling pip-19.2.3:
Successfully uninstalled pip-19.2.3
```

Figure 3: Screen shown on successful completion of pip

- iii. To make sure that pip is installed:
 - Open command prompt by: holding down the Windows key + R → type "cmd" → Enter
 - OR
 - click the search → type "cmd" → Enter
 - Once the cmd window is open, type in "pip" → Enter

If you see the pip menu appearing (Figure 4) you good to proceed to the next step! (You can close the cmd window afterwards)



```
Microsoft Windows [Version 10.0.18363.778]
(c) 2019 Microsoft Corporation. All rights reserved.
C:\WINDOWS\system32\pip
Usage:
pip (command) [options]
Commands:
install packages
```

Figure 4: pip menu

3. Installing Arduino

- i. Open Windows installer, Windows 7 and up (recommend). Accept/agree and click install to all windows that pop up. Note: Do not change any settings when installing.
- ii. Accept and install all drivers



- iii. Successful installation. Figure 5

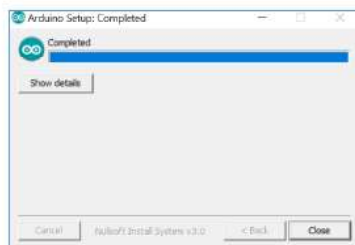
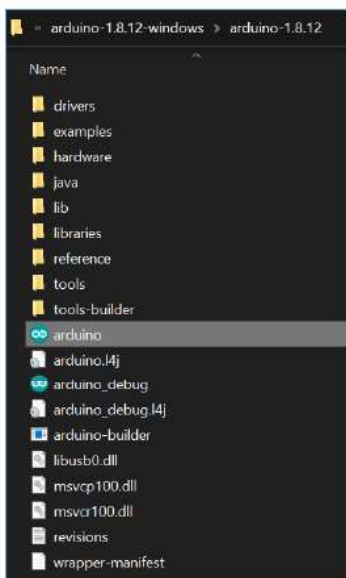


Figure 5: Screen shown on successful completion of Arduino

*You can also run Arduino without installing. Just unzip the compressed folder and open "Arduino" by double clicking. **HOWEVER IT IS BETTER TO INSTALL**



4. Finding your COM port number

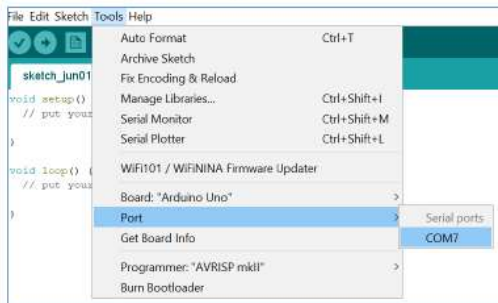
COM port (communication port) is a serial port that allows data to be exchanged from PC to device and vice-versa.

- i. Connect the Arduino microcontroller to the PC via the USB. Depending on your PC give it some time to pick up the board.



- ii. Open Arduino

- iii. Tools → Port → Select the COM port



iv. Remember your COM port number! *Write this port number down you will need it when writing Python code to Arduino.

**When connecting the microcontroller "Allow access" if the firewall alert pops up!*



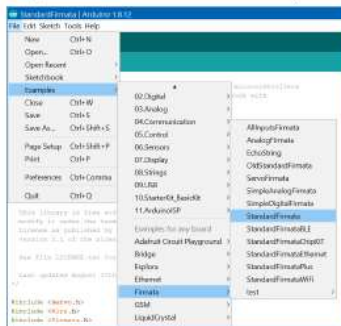
5. Setting up Standard Firmata on the Arduino UNO R3

First connect your Arduino UNO board to your PC. Depending on your PC give it some time to pick up the board.

- i. Open Arduino
- ii. Make sure to select the COM port



- iii. File → Examples → Firmata → StandardFirmata (double click)



- iv. Click on Verify (tick icon)



```
Done compiling.  
Sketch uses 12406 bytes (38%) of program storage space. Maximum is 32256 bytes.  
Global variables use 1085 bytes (52%) of dynamic memory, leaving 963 bytes for local variables. Maximum is 2048 bytes.
```

- v. Click on Upload (arrow icon). Wait for the upload to complete then close the Arduino IDE.



```
Done uploading.  
Sketch uses 12406 bytes (38%) of program storage space. Maximum is 32256 bytes.  
Global variables use 1085 bytes (52%) of dynamic memory, leaving 963 bytes for local variables. Maximum is 2048 bytes.
```

6. Check if you loaded Standard Firmata successfully

First connect your Arduino UNO board to your PC. Depending on your PC give it some time to pick up the board.

- i. Open the program "Firmata test" (make sure to close Arduino and Python before opening Firmata test)
- ii. Click Port → select the port that appears.

If you correctly uploaded StandardFirmata onto the Arduino UNO R3 you will see the COM port number and a list of all available ports.
(You can close the program "Firmata test")

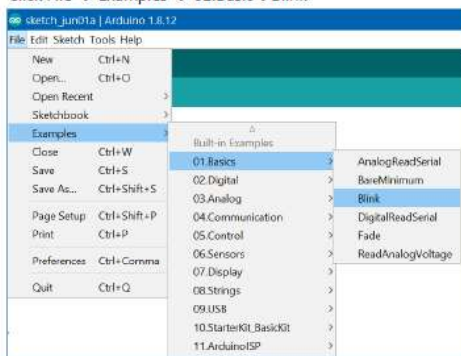
ONCE STEPS 1 TO 6 ARE COMPLETED PROCEED TO "Knowing your way around IDLE Python"

If for some reason your Python program does not run on Arduino UNO R3 (likely to happen if you reset the Arduino UNO R3), just repeat step 5.

HOW TO RESET ARDUINO?

This process will undo the Standard Firmata Sketch (to complete the workshop you need Standard Firmata)

- i. Click File → Examples → 01.Basic→Blink



- ii. Verify and Upload



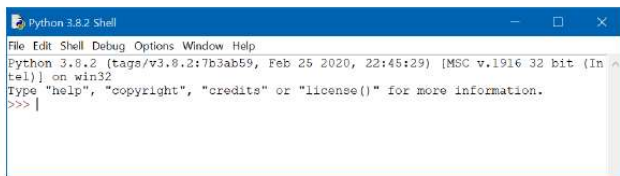
YOU CAN ZOOM WITHIN YOUR BROWSER TO ENLARGE

© Copyright 2020 by Reginald G. Govender

Knowing your way around IDLE Python

When you open IDLE Python you are presented with the "Python shell" window. Python shell will only be used during the running of your program when you enter input or when you output (display on screen).

For each program that you create you must select: "File" → "New File" from the Python shell window.



IDLE stands for Integrated DeveLopment Environment or Integrated Deveopment and Learning Environment

The new file created is known as a script and the remains "untitled" until you save it. This file will automatically be saved with the file extension .py (Python).

When typing your code in the new script you can minimize the Shell window but do not close the Shell window.

To run your program "Run" → "Run Module" or press "F5".



Knowing Python language

About the Python language:

- Text case sensitive
- Indentation of code represents begin and end
- # used to comment a line
- " ... " used to comment a segment
- Empty lines are skipped

You will notice that as you type certain words in Python it will change color denoting variables, input, output, etc.

To run your code in IDLE:

Run → Run Module
OR
F5

If your code does not run:

1. Check the spelling
2. Check the case of the letters
3. Indentation

3 TYPES OF ERRORS

SYNTAX/COMPILER ERROR: error relating to the programming language used. Unsuccessful compilation and program does not run. Examples: missing indentation for begin/end, missing brackets, undeclared variables etc.

LOGICAL ERROR: error in programming logic. Program runs until user exits. Does not give desired output. Examples: Not adding 15% VAT, average- sum not divided by total participants, etc.

RUN TIME/EXECUTION ERROR: error that appears after you compile and run your code. Program runs until certain point. Example: Divided by zero, trying to open a file that does not exist, reading wrong value type, integer too large, etc.

NOW YOU READY TO PROCEED TO WORKSHOP 1!





Appendix J - Workshop session one

Workshop session 1

© Copyright 2020 by Reginald G. Govender

Activity 1: Getting know the hardware

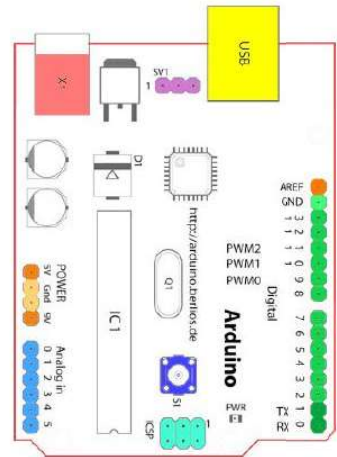
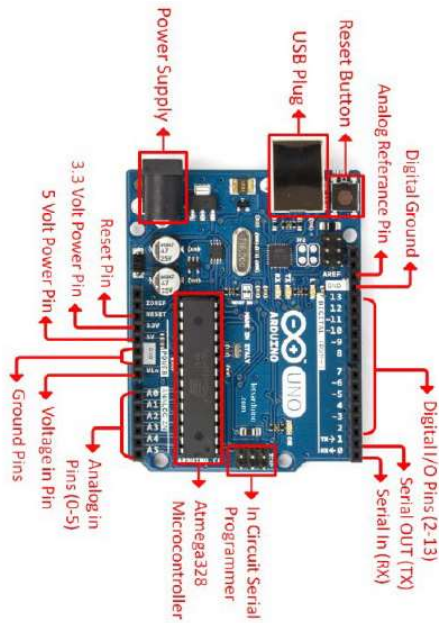
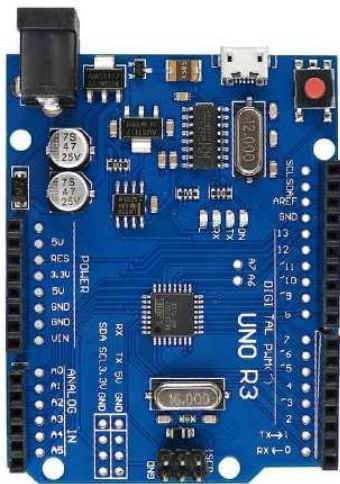
Identifying the basic hardware components (Additional components will be explored in the activity that it is used)

Arduino UNO R3	Breadboard	USB cable	Jumper cables
			
<ul style="list-style-type: none"> ✓ The Arduino UNO R3 is a microcontroller board that is used to build prototypes. ✓ Programs are created and executed to the microcontroller. ✓ There are digital and analog pins on the microcontroller. ✓ Buttons, sensors, and motors are connected to the microcontroller by the use of jumper wires. ✓ The microcontroller connects to the computer via USB. ✓ The USB connection is used to power and transfer data between the microcontroller and computer. 	<ul style="list-style-type: none"> ✓ The breadboard is made up of plastic with a bunch of tiny holes in it. ✓ Under the holes, there are metal strips that allow you to easily insert electronic components to the prototype. ✓ <i>When connecting a pin to the breadboard use little force the first time.</i> 	<ul style="list-style-type: none"> ✓ Allows for the transmission of data between the microcontroller and computer. ✓ It supplies power to the microcontroller. 	<ul style="list-style-type: none"> ✓ Jumper wires allow you to easily connect components to the microcontroller without the need to solder. ✓ You have been given two types of jumper wires: male-to-male and male-to-female. ✓ The male-to-male has a pin protruding that can plug into components, while female ends do not and are used to plug things into.

Further exploration of the microcontroller

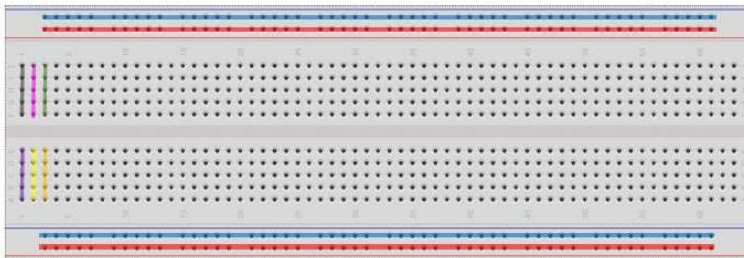
NOTE: The Arduino UNO R3 microcontroller given to you might slightly differ in physical appearance but all the components remain the same from the pictures below.

Familiarize yourself with the microcontroller especially where the digital and analog pins are located.



Further exploration of the breadboard

The colour lines show the electric flow and how the metal strips are positions under the holes.

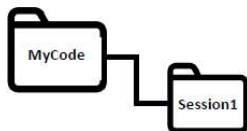


Additional notes and tips appear
alongside each activity



YOU CAN ZOOM WITHIN YOUR BROWSER TO ENLARGE IN ALL WORKSHEETS

Before starting Activity 2: Create the subfolder folder named "Session1". All programs (scripts) created in this workshop session will be saved in the "Session1" folder



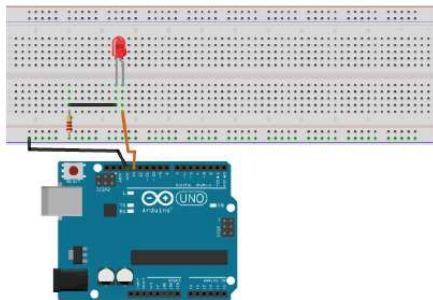
Activity 2: Hello world Disconnect the Arduino UNO R3 from the PC

In this activity, you will program an LED to blink and output "Hello world". This activity will not require user input.

INPUT	PROCESSING	OUTPUT
Keyboard input	What's happening on the prototype?	Screen output (Python shell)
-	Built-in LED blinks	"Hello world!"

Build:

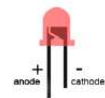
D3 (Digital pin 3) connects to the anode (+) of the LED
 GND(Ground) connects to breadboard
 220Ω resistor is connected between the cathode (-) of the LED and GND



Prototype is reference to the Arduino board and the breadboard.

When building you can use any colour jumper and LED unless specified in the activity.

Light Emitting Diode (.LED)



Always connect a resistor between the GND and the cathode (short leg).

A resistor is needed on the GND connection because, without it, we can cause a short circuit resulting in a burnt LED.

Code: Save the script as *Act2HelloWorld*

```
1 import pyfirmata
2 import time
3
4 board= pyfirmata.Arduino('COM4')
5 time.sleep(2)
6 led = board.get_pin('d:3:o')
7
8 led.write(1)
9 time.sleep(0.5)
10 led.write(0)
11 print ("Hello world!")
```

Explanation of code:

Line 1: Making use of the Pyfirmata package. Remember you already installed Pyfirmata earlier, here we just telling Python that we using this package.

Line 2: Making use of the time package. The time package is built in Python therefore we didn't need to install.

Line 4: Declare a variable called *board* that is assigned to the Arduino attached to your PC. **Remember the COM port number from the setup session.**

Line 5: Delay the code for 2 seconds. This is for Python to establish a connection between the Arduino and the PC.

Line 6: Declare a variable called *led*. The *d* refers to the digital pin, *3* is the pin that the jumper is connected and *o* refers to output.

Line 8: Turn the LED ON.

Line 9: Delay the code for 0.5 seconds. This keeps the LED ON for 0.5 seconds before moving the Line 10.

Line 10: Turn the LED OFF.

Line 11: Display message on the PC screen (will appear in the Python Shell window).

Now connect the Arduino to the computer, give it a few seconds to detect before running code

You notice the LED will blink and "Hello world!" will be displayed on the Python Shell window

To create a new script:

Open IDLE Python, you will be presented with Python Shell. From Python Shell go to File→New

Digital pins like can have only two possible values:

Value	Level	Voltage
0	Low	0V
1	High	5V

This means you can write 1 to turn a digital pin ON and write 0 to turn OFF

My COM port number is 4, your COM port number maybe a different.

Your COM port number will usually stay the same. If in doubt open the "Firmata_test" software to check. Remember when you open this software you must close Arduino and Python.

To run your code in IDLE:

Run→Run Module
OR
F5

Try out before moving to Activity 3

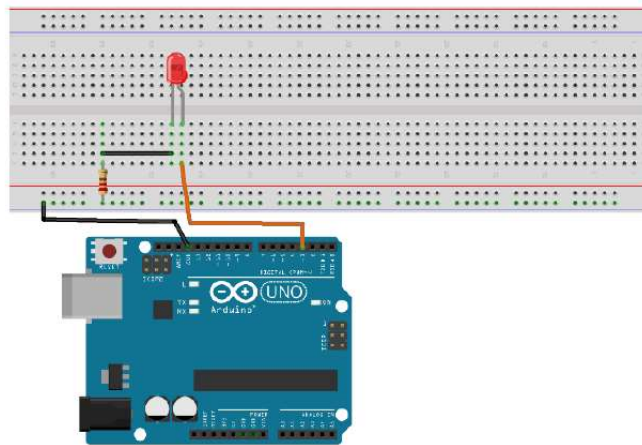
Program the LED to blink and output "Hello world" 3 times.

Activity 3: Blinking LED Disconnect the Arduino UNO R3 from the PC

In this activity, you will program an LED to blink 10 times and then display "Hello World!" This activity will not require user input.

INPUT	PROCESSING	OUTPUT
Keyboard input	What's happening on the prototype?	Screen output (Python shell)
-	LED blinks	"Hello world!"

Build: (Same as Activity 2)



Your code: Save the script as Act3Blink

Now connect the Arduino board to the computer, give it a few seconds to detect before running code.

REDUCE CODE
 Instead of retyping lines 8-10 (Activity 2), ten times for the LED to blink ten times we will use a looping structure (iteration structure) called the *for loop*. The *for loop* only runs for a specified number of times.

Iteration structure: for loop

Example:


```
for i in range(5):
    print('i is now:', i)
print('The end')
```

OUTPUT:
 i is now 0
 i is now 1
 i is now 2
 i is now 3
 i is now 4
 The end

range() generates an iterator to progress integers starting with 0 up to n-1

Alternative: `for i in range (start value, end value)`
 Example: `for i in range (0, 4)`

Appendix K – Self-evaluation workshop session one



Workshop session 1

Rate your accomplishment based on the three activities from this workshop.

- 1- Very difficult
- 2- Difficult
- 3- Neutral
- 4- Easy
- 5- Very Easy

*Required

ID Code *

Your answer _____

Activity 1: Getting to know the hardware *

	1	2	3	4	5	
Very difficult	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very easy

Activity 2: Hello world *

	1	2	3	4	5	
Very difficult	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very easy

Activity 3: Blinking LED *

	1	2	3	4	5	
Very difficult	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very easy

Submit

Never submit passwords through Google Forms.

[Report Abuse](#) · [Terms of Service](#) · [Privacy Policy](#)

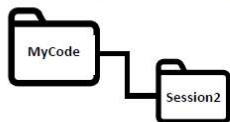
Google Forms

Appendix L - Workshop session two

Workshop session 2

© Copyright 2020 by Reginald G. Govender

Create the following subfolder named "Session2". All programs (scripts) created in this workshop session will be saved in the "Session2" folder.

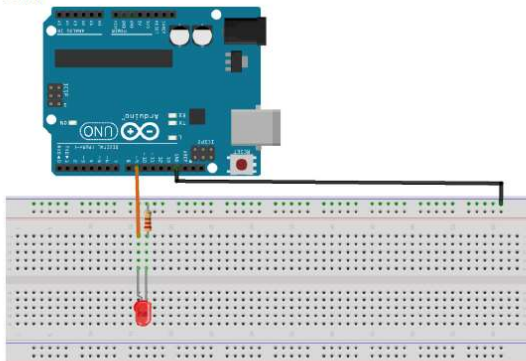


Activity 1: User control **Disconnect the Arduino UNO R3 from the PC**

In this activity, the user will be able to enter "1" to turn on a LED connected to D9 (Digital pin 9).

INPUT	PROCESSING	OUTPUT
Keyboard input	What's happening on the prototype?	Screen output (Python shell)
1	Turns LED ON for 3 seconds only.	"LED turned ON"
Any other input	LED is OFF.	"Invalid input run program again"

Build:



Code: Save the script as *Act1Userinput*

```
1 import pyfirmata
2 import time
3
4 board= pyfirmata.Arduino('COM4')
5 time.sleep(2)
6 led = board.get_pin('d:9:o')
7 print ("Enter 1: switch LED ON")
8 userinput = input()
9 if (userinput == '1'):
10     led.write(1)
11     print ("LED turned ON")
12     time.sleep(3)
13     led.write(0)
14 else:
15     print ("Invalid input run program again")
```

Explanation of code:

Line 1: Making use of the Pyfirmata package.
Line 2: Making use of the time package.
Line 4: Declare a variable called *board* that is assigned to the Arduino attached to your PC.
Line 5: Delay the code for 2 seconds. This is for Python to establish a connection between the Arduino and the PC.
Line 6: Declare a variable called *led*. The *d* refers to the digital pin, *9* is the pin that the jumper is connected and *o* refers to output.
Line 7: Message on the PC screen (Python Shell) prompting user input.
Line 8: Declare a variable called *userinput* to hold/store what is entered through the keyboard.
Line 9: Check if a "1" was entered.
Line 10: Begin of *if statement*. Turn LED ON.
Line 11: Display message on the PC screen.
Line 13: Delay the code for 3 seconds.
Line 14: End of *if statement*. Turn LED OFF.
Line 15: *else* part of *if statement*. This will ensure that the program will not crash when the user enters any other value besides "1".
Line 16: Display message on the PC screen.

Now connect the Arduino board to the computer, give it a few seconds to detect before running code.

Notice that when the user enters "1" the LED is turned ON for 3 seconds. If any other value is entered a message is displayed.

About the if statement:

```
if expression:
    statement(s)
else:
    statement(s)
```

Example code:

```
import pyfirmata
amount=600
if (amount<1000):
    discount = amount*0.05
    print ("Discount", discount)
else:
    discount = amount*0.10
    print ("Discount", discount)
print ("Net payable:", amount-discount)
```

OUTPUT:
Discount 30.0
Net payable: 570.0

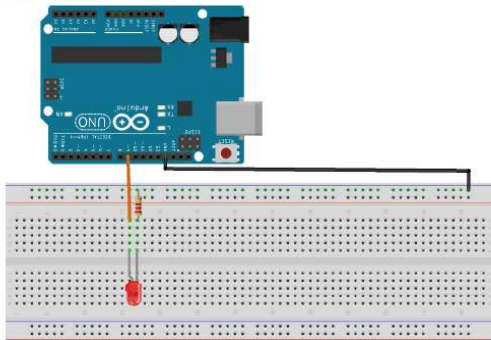
Note: When user input is required it must be entered in the Python Shell window.

Activity 2: Blinking LED (Disconnect the Arduino UNO R3 from the PC)

In this activity, you are going to make some changes to `Act1Userinput.py` so that the user will be able to enter "1" to turn the LED ON or "0" to turn the LED OFF.

INPUT	PROCESSING	OUTPUT
Keyboard input	What's happening on the prototype?	Screen output (Python shell)
1	Turns LED ON	"LED turned ON"
0	LED remains OFF	"LED turned OFF"
Any other input	LED remains OFF	"Invalid input run program again"

Build: Same as Activity 1



Code: Save the script as *Act2Userupdate*

```
1 import pyfirmata
2 import time
3
4 board= pyfirmata.Arduino('COM4')
5 time.sleep(2)
6
7 led = board.get_pin('d:9:o')
8 print ("Enter 1: switch LED ON or 0: switch LED OFF")
9 userinput = input()
10 if (userinput == '1'):
11     led.write(1)
12     print ("LED turned ON")
13 elif (userinput == '0'):
14     led.write(0)
15     print ("LED turned OFF")
16 else:
17     print ("Invalid input run program again")
```

Explanation of code:

Line 1: Making use of the Pyfirmata package.

Line 2: Making use of the time package.

Line 4: Declare a variable called *board* that is assigned to the Arduino attached to your PC.

Line 5: Delay the code for 2 seconds. This is for Python to establish a connection between the Arduino and the PC.

Line 7: Declare a variable called *led*. The *d* refers to the digital pin, *9* is the pin that the jumper is connected and *o* refers to output.

Line 8: Displays message on PC screen prompting user input.

Line 9: Declare a variable called *userinput* to hold/store what is entered through the keyboard.

Line 10: Check if a "1" was entered.

Line 11: Turn the LED ON.

Line 12: Display message on the PC screen.

Line 13: Check if a "0" was entered.

Line 14: Turn the LED OFF.

Line 15: Display message on the PC screen.

Line 16: else part of *if statement*.

Line 17: Displays message on the screen.

Now connect the Arduino board to the computer, give it a few seconds to detect before running code.

You will notice that once the LED is switched ON or OFF you cannot change the state of the LED, except by running the code again.

This is solved by adding an iteration structure called the *while loop*.

This is added before the user is prompted to enter their choice.

Modification to code (Lines 8 and 10):

```
1 import pyfirmata
2 import time
3
4 board= pyfirmata.Arduino('COM4')
5 time.sleep(2)
6
7 led = board.get_pin('d:9:o')
8 flag = True
9
10 while flag:
11     print ("Enter 1: switch LED ON or 0: switch LED OFF")
12     userinput = input()
13     if (userinput == '1'):
14         led.write(1)
15         print ("LED turned ON")
16     elif (userinput == '0'):
17         led.write(0)
18         print ("LED turned OFF")
19     else:
20         print ("Invalid input run program again")
```

Explanation of code:

Line 8: Declare a variable *flag* and set it to true.

Line 10: Start of *while loop*. You will notice that the *flag* is never set to false within the while loop. Therefore lines 11 to 20 will continually run.

Lines 10 to 19 falls within the while statement therefore the indentation. Take note that the *begin and end of the while loop is shown visually by indentation*.

Run the code.

The *while loop* is unlike the *for loop*. Remember the *for loop* only runs for the specified number of times. In the *while loop* as long as the condition is met the code inside will continuously execute.

Because the *if statements* are within the *while loop* we call this nested.

Iteration structure: while loop

while expression:
statement(s)

Example code:

```
count = 0
while (count < 9):
    print ('The count is:', count)
    count = count + 1
print ("Good bye!")
```

OUTPUT:

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
Good bye!
```

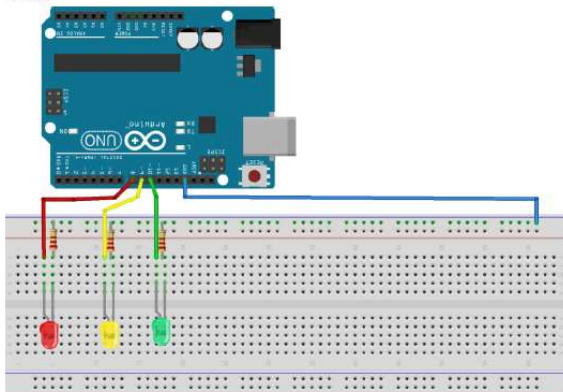
Activity 3: Traffic lights (Disconnect the Arduino UNO R3 from the PC)

The user must be able to control the traffic lights:

INPUT	PROCESSING	OUTPUT
Keyboard input	What's happening on the Arduino board	Screen output (Python shell)
1	Turn green LED ON.	"Go Go Go!"
2	Turn yellow LED ON.	"Slow down"
3	Turn red LED ON.	"Stop Stop Stop!"
"Any other input"	All LEDs blink twice together (ON-OFF-ON-OFF). The interval for each blink is 0.5	"Invalid input"

Remember to use a *while* loop so that the user can change the state of the LED.

Build:




Your code: Save the script as *Act3Traffic*

Test your code:

Now connect the Arduino board to the computer, give it a few seconds to detect before running code.

Appendix M – Self-evaluation workshop session two



Workshop session 2

Rate your accomplishment based on the three activities from this workshop.

- 1- Very difficult
- 2- Difficult
- 3- Neutral
- 4- Easy
- 5- Very Easy

***Required**

ID Code ^{*}

Your answer _____

Activity 1: User control ^{*}

	1	2	3	4	5	
Very difficult	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very easy

Activity 2: Blinking LED ^{*}

	1	2	3	4	5	
Very difficult	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very easy

Activity 3: Traffic lights ^{*}

	1	2	3	4	5	
Very difficult	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very easy

Submit

Never submit passwords through Google Forms.

This content is neither created nor endorsed by Google. [Report Abuse](#) · [Terms of Service](#) · [Privacy Policy](#)

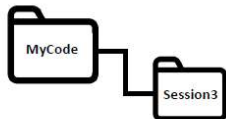
Google Forms

Appendix N - Workshop session three

Workshop session 3

© Copyright 2020 by Reginald G. Govender

Create the following subfolder named "Session3". All programs (scripts) created in this workshop session will be saved in the "Session3" folder.



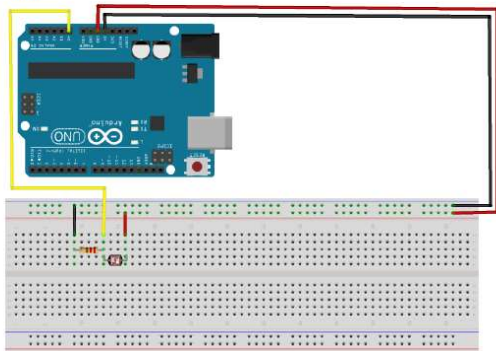
Activity 1: Reading the LDR value **(Disconnect the Arduino UNO R3 from the PC)**

In this activity, you are going to determine the light intensity of your surroundings when the user (you) enter "Y" on the keyboard.

*The value will be different for everyone depending on the amount of light in the room.

INPUT	PROCESSING	OUTPUT
Keyboard	What's happening on the prototype?	Screen output (Python shell)
Y	The LDR will read in value.	Value is displayed on the screen.

Build:



Light Dependent Resistor (LDR)



The Light Dependent Resistor is a special type of resistor that allows higher voltages to pass through it (low resistance) whenever there is a high intensity of light, and passes a low voltage (high resistance) whenever it is dark.

LDR gives out an analog voltage, therefore must be connected to an analog input pin on the Arduino board.

Code: Save the script as *ActLDRread*

```
1 import time
2 import pyfirmata
3
4 board= pyfirmata.Arduino('COM4')
5 time.sleep(2)
6
7 ldrAnalog= board.get_pin('a:0:i')
8 flag= True
9
10 it = pyfirmata.util.Iterator(board)
11 it.start()
12
13 while flag:
14     print ("Read LDR value")
15     userInput= input()
16     if userInput=="Y":
17         time.sleep(0.2)
18         ldrVal=ldrAnalog.read()
19         print(ldrVal)
20     else:
21         print('Invalid')
```

Explanation of code:

Line 1: Making use of the Pyfirmata package.

Line 2: Making use of the time package.

Line 4: Declare a variable called *board* that is assigned to the Arduino attached to your PC.

Line 5: Delay the code for 2 seconds. This is for Python to establish a connection between the Arduino and the PC.

Line 7: Declare a variable called *ldrAnalog*. The *a* refers to the analog pin, *0* is the pin that the jumper is connected and *i* refers to input.

Line 8: Declare a variable *flag* and set it to true.

Line 10: Declare a variable *it* which sets up an iterator thread that will be used to read the status of the inputs of the circuit.

Line 11: Start the iterator otherwise the board will keep sending data to your serial until it overflows.

Line 13: Start of *while loop*. You will notice that *flag* is never set to false within the *while loop*. Therefore lines 14 to 21 will continually run.

Line 14: Screen display prompting user input.

Line 15: Declare a variable called *userinput* to hold/store what is entered through the keyboard.

Line 16: Checks if "Y" is entered.

Line 17: 0.2-second delay is needed so that a reliable LDR value is captured- without this delay, the LDR sensor will lag to read in a value.

Line 18: Declare a variable called *ldrVal* to hold/store LDR value.

Line 19: Displays the value on the PC screen – Python shell

Line 20: Checks if the user enter something other than "Y"

Line 21: Displays invalid message on the PC screen – Python shell

Now connect the Arduino board to the computer, give it a few seconds to detect before running code.

Use your fingers to block out light from entering the LDR. Use three fingers (pointer, middle, and thumb) to prevent light getting to the sensor. Notice the change in value NOW when you enter "Y". When light is blocked from the LDR, what is its value?

The LDR reads the light intensity therefore will be inputting values to the program. The values from the LDR will fluctuate because light intensity changes and the sensor is very sensitive.

The *flag* variable

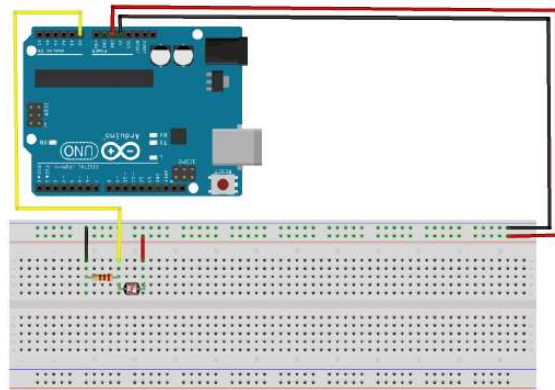
We refer to a variable that can be set to true or false, Boolean. Boolean is a logic data type.

Activity 2: Continuous reading of LDR values Disconnect the Arduino UNO R3 from the PC

This activity will not require user input but will continuously read and display the values from the LDR.

INPUT	PROCESSING	OUTPUT
Keyboard	What's happening on the prototype?	Screen output (Python shell)
-	The LDR will read in the value.	Value is displayed on the screen.

Build: Same as Activity 1



Code: Save the script as *Act2LDRcont*

```
1 import time
2 import pyfirmata
3
4 board= pyfirmata.Arduino('COM4')
5 time.sleep(2)
6 ldrAnalog= board.get_pin('a:0:i')
7 flag= True
8
9 it = pyfirmata.util.Iterator(board)
10 it.start()
11
12 while flag:
13     time.sleep(0.2)
14     ldrVal=ldrAnalog.read()
15     print(ldrVal)
```

Explanation of code:

Line 1: Making use of the Pyfirmata package.

Line 2: Making use of the time package.

Line 4: Declare a variable called *board* that is assigned to the Arduino attached to your PC.

Line 5: Delay the code for 2 seconds. This is for Python to establish a connection between the Arduino and the PC.

Line 6: Declare a variable called *ldrAnalog*. The *a* refers to the analog pin, *0* is the pin that the jumper is connected to and *i* refers to input.

Line 7: Declare a variable *flag* and set it to true.

Line 9: Declare a variable *it* which sets up an iterator thread that will be used to read the status of the inputs of the circuit.

Line 10: Start the iterator otherwise the board will keep sending data to your serial until it overflows.

Line 12: Start of *while* loop. You will notice that *flag* is never set to false within the *while* loop. Therefore lines 13 to 15 will run continually.

Line 13: 0.2-second delay is needed so that a reliable LDR value is captured- without this delay, the LDR sensor will lag to read in a value- "NONE" will be captured.

Line 14: Declare a variable called *ldrVal* to hold/store LDR value.

Line 15: Display the value on the PC screen

***Now connect the Arduino board to the computer, give it a few seconds to detect the board and run code.**

Use your fingers to block out light from the LDR. Use three fingers (pointer, middle, and thumb) to prevent light getting to the sensor.

Notice that the code will continuously read in values from the LDR sensor and display in Python Shell.

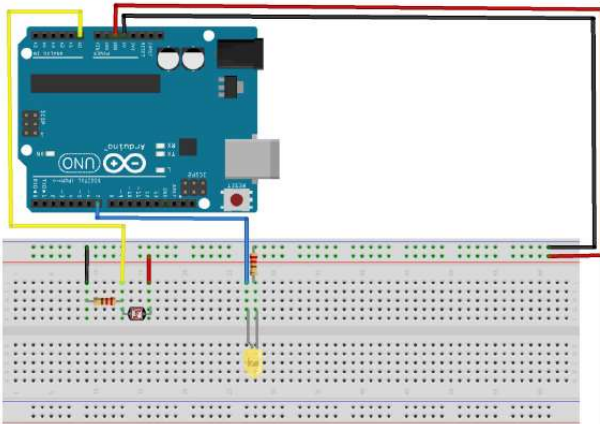
Activity 3: Night sensor (Disconnect the Arduino UNO R3 from the PC)

In this activity, the LED will automatically turn ON when the LDR sensor detects darkness. From Activity 2 you would have noticed that the LDR value is "0" when it is covered and not exposed to any light. This would mean that the LED is ON when the value is "0" otherwise the LED will remain OFF.

INPUT	PROCESSING	OUTPUT
LDR	What's happening on the prototype?	Screen output (Python shell)
Uncovered	LED is OFF.	Display the LDR value.
Covered (LDR value is 0)	LED turns ON.	Display the LDR value.

Remember to use a *while* loop so that the LDR value is always updated.

Build: Same build as Activity 2 with the following additions: LED, resistor and jumper to D7




If you using your fingers to block out light from the LED- use three fingers (pointer, middle and thumb) to prevent and light getting to the sensor.

Your code: Save the script as *Act3Nightlight*

Test your code:

Now connect the Arduino board to the computer, give it a few seconds to detect before running code.

Appendix O – Self-evaluation workshop session three



Workshop session 3

Rate your accomplishment based on the three activities from this workshop.

- 1- Very difficult
- 2- Difficult
- 3- Neutral
- 4- Easy
- 5- Very Easy

***Required**

ID Code ^{*}

Your answer

Activity 1: Reading the LDR value ^{*}

	1	2	3	4	5	
Very difficult	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very easy

Activity 2: Continuous reading of LDR values ^{*}

	1	2	3	4	5	
Very difficult	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very easy

Activity 3: Night sensor ^{*}

	1	2	3	4	5	
Very difficult	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very easy

Submit

Never submit passwords through Google Forms.

This content is neither created nor endorsed by Google. [Report Abuse](#) · [Terms of Service](#) · [Privacy Policy](#)

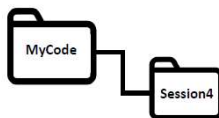
Google Forms

Appendix P - Workshop session four

Workshop session 4

© Copyright 2020 by Reginald G. Govender

Create the following subfolder named "Session4". All programs (scripts) created in this workshop session will be saved in the "Session4" folder.

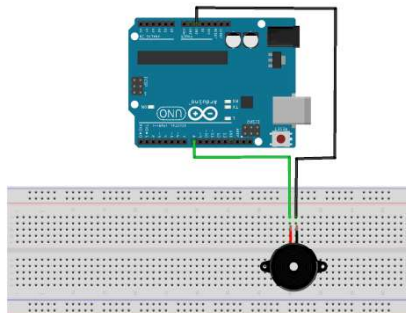


Activity 1: Buzzer tone Disconnect the Arduino UNO R3 from the PC

This activity will not require user input.

INPUT	PROCESSING	OUTPUT
Keyboard	What's happening on the prototype?	Screen output (Python shell)
-	The buzzer will turn ON creating a sound effect.	After the buzzer tone, a message will be displayed on the screen.

Build: For this activity you could connect the buzzer directly to the Arduino UNO R3, therefore, bypassing the breadboard.



Buzzer



The buzzer is an output device that produces a sound. To connect: the + leg is connected to the pin that will provide the power and the other leg is connected to ground.

Code: Save the script as *Act1Buzzer*

```
1 import time
2 import pyfirmata
3
4 board= pyfirmata.Arduino('COM4')
5 time.sleep(2)
6 pinBuzzer= board.get_pin('d:8:o')
7
8 for i in range(1,10):
9     pinBuzzer.write(1)
10    time.sleep(0.5)
11    pinBuzzer.write(0)
12    time.sleep(0.1)
13 print('The end of sound')
```

Try out!

Change the sleep times on lines 10 and 12 this will create different sound effects.

Explanation of code:

Line 1: Making use of the time package.

Line 2: Making use of the Pyfirmata package.

Line 4: Declare a variable called *board* that is assigned to the Arduino attached to your PC.

Line 5: Delay the code for 2 seconds. This is for Python to establish a connection between the Arduino and the PC.

Line 6: Declare a variable *pinBuzzer*. The *d* refers to the digital pin, *8* is the pin that the jumper is connected and *o* refers to output.

Line 8: Start of *for loop*. The range for the loop is set from 1 to 10. Lines 9 to 12 falls within the *for loop* and will repeat 10 times.

Line 9: Turn the buzzer ON.

Line 10: Delay for half a second before going to line 11.

Line 11: Turn the buzzer OFF.

Line 12: Delay for 0.1 seconds.

Line 13: Display message. This line is outside the *for loop*.

***Now connect the Arduino board to the computer, give it a few seconds to detect before running code.**

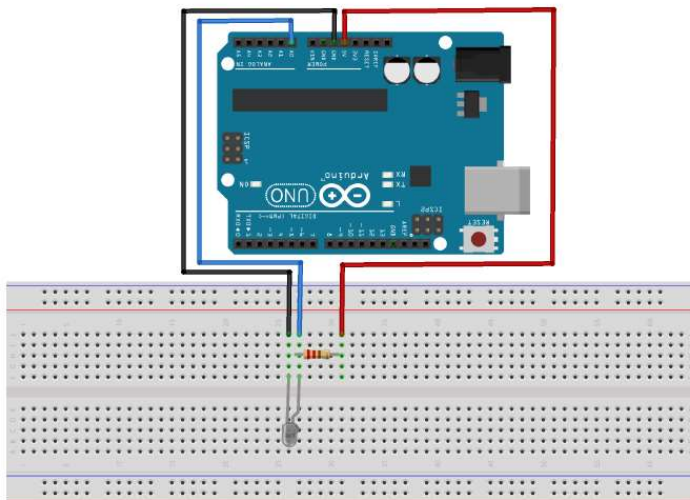
You will hear a sound effect that will play and then a message will be displayed on the screen (Python Shell).

Activity 2: Flame Sensor Disconnect the Arduino UNO R3 from the PC

This activity will not require user input but will continuously read and display the values from the sensor.

INPUT	PROCESSING	OUTPUT
Keyboard	What's happening on the prototype?	Screen output (Python shell)
-	The state of the flame sensor will be captured.	Value is displayed on the screen

Build:



DO NOT USE A FLAME

To imitate a fire flame use your flash on your smartphone camera. Turn ON the flash and hold the camera flash light near the sensor. This will have the same effect as holding a flame.

Flame sensor



The flame sensor looks like a black LED. You will notice that the bulb is "glass like". When fire burns it emits a small amount of infra-red light, this light will be received by the Photodiode (IR receiver) on the sensor module.

Code: Save the script as *Act2FireSensor*

```
1 import time
2 import pyfirmata
3
4 board= pyfirmata.Arduino('COM4')
5 time.sleep(2)
6
7 pinFlame= board.get_pin('a:0:i')
8 flag= True
9
10 it = pyfirmata.util.Iterator(board)
11 it.start()
12
13 while flag:
14     flameVal= pinFlame.read()
15     if flameVal is not None:
16         print(flameVal)
```

"None"

When power is supplied to the sensor it takes some time for it to register and read in values, in the meantime it will read in "NONE". Line 15 ensures that a numerical value displayed and not "NONE".

Explanation of code:

Line 1: Making use of the time package.

Line 2: Making use of the Pyfirmata package.

Line 4: Declare a variable called *board* that is assigned to the Arduino attached to your PC.

Line 5: Delay the code for 2 seconds. This is for Python to establish a connection between the Arduino and the PC.

Line 7: Declare a variable *pinFlame*. The *a* refers to the analog pin, *0* is the pin that the jumper is connected and *i* refers to input.

Line 8: Declare a variable *flag* and set it to true.

Line 10: Declare a variable *it* which sets up an iterator thread that will be used to read the status of the inputs of the circuit.

Line 11: Start the iterator otherwise the board will keep sending data to your serial until it overflows.

Line 13: Start of *while* loop. You will notice that *flag* is never set to false within the *while* loop. Therefore lines 14 to 16 will continually run.

Line 14: Declare a variable called *flameVal* to hold/store reading from the sensor.

Line 15: *if* statement to check for a valid reading. THIS IS USED INSTEAD OF TIME DELAY (compare with Workshop 3, *Act1LDRread*, Line 17)

Line 16: Displays the value.

***Now connect the Arduino board to the computer, give it a few seconds to detect before running code.**

Switch your smartphone camera flash ON. Hold the flash near the sensor and notice what happens to the values.

Activity 3: Fire alarm [Disconnect the Arduino UNO R3 from the PC]

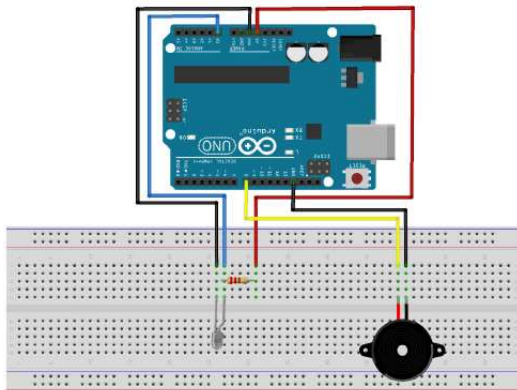
From Activity 2 you would have noticed that the flame sensor will have a value of "1.0" when there is no flame (flashlight) detected and the value decreases when it detects a flame (flashlight). In this activity, you will code such that when a flame is detected the buzzer will be activated. Create a unique tone for the alarm by experimenting with the number of second/s the buzzer is ON and OFF.

INPUT	PROCESSING	OUTPUT
Flame Sensor	What's happening on the prototype?	Screen output (Python shell)
Camera flashlight OFF	No sound from the buzzer.	Display "SAFE"
Camera flashlight ON	Sound from the buzzer.	Display "DANGER"

Remember to use a *while loop* so that the sensor value is always updated.

In Activity 2 you would have noticed that the value of the sensor is 1. Once it picks up the light intensity of a "flame(flash)" the value drops below 1.

Build:



Your code: Save the script as `Act3FireAlert`

Test your code:

*Now connect the Arduino board to the computer, give it a few seconds to detect before running code.

Appendix Q – Self-evaluation workshop session four



Workshop session 4

Rate your accomplishment based on the three activities from this workshop.

- 1- Very difficult
- 2- Difficult
- 3- Neutral
- 4- Easy
- 5- Very Easy

***Required**

ID Code *

Your answer _____

Activity 1: Buzzer tone *

	1	2	3	4	5	
Very difficult	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very easy

Activity 2: Flame Sensor *

	1	2	3	4	5	
Very difficult	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very easy

Activity 3: Fire alarm *

	1	2	3	4	5	
Very difficult	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very easy

Submit

Never submit passwords through Google Forms.

This content is neither created nor endorsed by Google. [Report Abuse](#) - [Terms of Service](#) - [Privacy Policy](#)

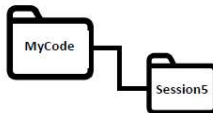
Google Forms

Appendix R - Workshop session five

Workshop session 5

© Copyright 2020 by Reginald G. Govender

Create the following subfolder named "Session5". All programs (scripts) created in this workshop session will be saved in the "Session5" folder.

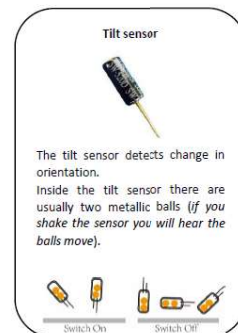
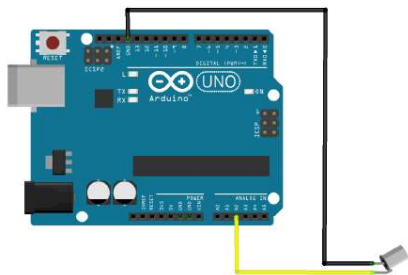


Activity 1: Reading form a tilt sensor **Disconnect the Arduino UNO R3 from the PC**

In this activity, you are going to determine the state of the tilt sensor. No user input needed.

INPUT	PROCESSING	OUTPUT
Keyboard	What's happening on the prototype?	Screen output (Python shell)
-	Value of the tilt sensor will be captured	Value is displayed on the screen

Build: In this build we will not connect via the breadboard. However, if want- you can!



Code: Save the script as *Act1TiltSensor*

```
1 import time
2 import pyfirmata
3
4 board= pyfirmata.Arduino('COM4')
5 time.sleep(2)
6
7 tiltсен= board.get_pin('a:2:i')
8 flag= True
9
10 it = pyfirmata.util.Iterator(board)
11 it.start()
12
13 while flag:
14     time.sleep(0.5)
15     tiltVal=tiltсен.read()
16     print (tiltVal)
```

Explanation of code:

Line 1: Making use of the time package.

Line 2: Making use of the Pyfirmata package.

Line 4: Declare a variable called *board* that is assigned to the Arduino attached to your PC.

Line 5: Delay the code for 2 seconds. This is for Python to establish a connection between the Arduino and the PC.

Line 7: Declare a variable called *tiltсен*. The *a* refers to the analog pin, *2* is the pin that the jumper is connected and *i* refers to input.

Line 8: Declare a variable *flag* and set it to true.

Line 10: Declare a variable *it* which sets up an iterator thread that will be used to read the status of the inputs of the circuit.

Line 11: Start the iterator otherwise the board will keep sending data to your serial until it overflows.

Line 13: Start of *while loop*. You will notice that *flag* is never set to false within the *while loop*. Therefore lines 14 to 16 will continually run.

Line 14: Delay for half a second before going to line 15 to read the value.

Line 15: Declare a variable called *tiltVal* to hold/store reading from the sensor.

Line 16: Display the value on the PC screen (Python shell)

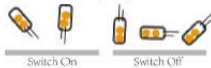
Now connect the Arduino board to the computer, give it a few seconds to detect before running code.

What is the value of the tilt sensor when held upright?



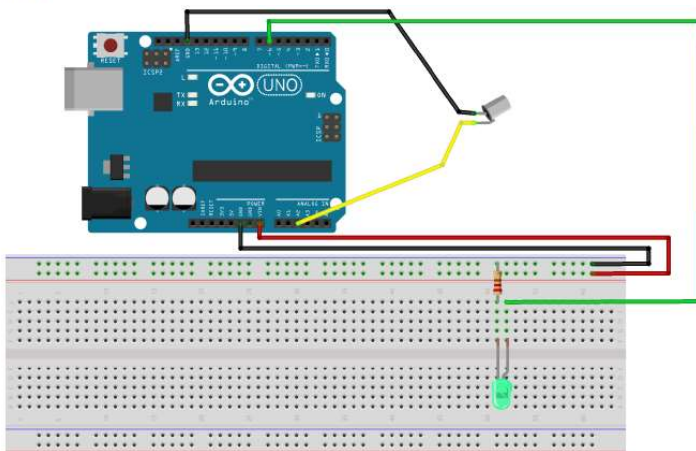
Activity 2: Tilt lights ON Disconnect the Arduino UNO R3 from the PC

In this activity, you will code to switch a LED ON when the tilt sensor is in an OFF state. From Activity 1 you would have noticed that when the tilt sensor is ON, that's in an upright position the value is zero ("0").



INPUT	PROCESSING	OUTPUT
Tilt sensor	What's happening on the prototype?	Screen output (Python shell)
ON state	The LED is OFF	Tilt sensor value is continuously displayed.
OFF state	The LED is ON	Tilt sensor value is continuously displayed.

Build:



Code: Save the script as *Act2TiltLight*

```
1 import time
2 import pyfirmata
3
4 board= pyfirmata.Arduino('COM4')
5 time.sleep(2)
6
7 led= board.get_pin('d:6:o')
8 tiltSen= board.get_pin('a:2:i')
9 flag= True
10
11 it = pyfirmata.util.Iterator(board)
12 it.start()
13
14 while flag:
15     time.sleep(0.5)
16     tiltVal=tiltSen.read()
17     print(tiltVal)
18     if tiltVal != 0:
19         led.write(1)
20     else:
21         led.write(0)
```

Explanation of code:

Line 1: Making use of the time package.

Line 2: Making use of the Pyfirmata package.

Line 4: Declare a variable called *board* that is assigned to the Arduino attached to your PC.

Line 5: Delay the code for 2 seconds. This is for Python to establish a connection between the Arduino and the PC.

Line 7: Declare a variable *led*. The *d* refers to the digital pin, *6* is the pin that the jumper is connected and *o* refers to output.

Line 8: Declare a variable called *tiltSen*. The *a* refers to the analog pin, *2* is the pin that the jumper is connected and *i* refers to input.

Line 9: Declare a variable *flag* and set it to true.

Line 11: Declare a variable *it* which sets up an iterator thread that will be used to read the status of the inputs of the circuit.

Line 12: Start the iterator otherwise the board will keep sending data to your serial until it overflows.

Line 14: Start of *while loop*. You will notice that *flag* is never set to false within the *while loop*. Therefore lines 15 to 21 will continually run.

Line 15: Delay for half a second before going to line 16 to read the value.

Line 16: Declare a variable called *tiltVal* to hold/store reading from the sensor.

Line 17: Display the value on the PC screen (Python shell).

Line 18: *if statement* to check if the value is not "0".

Line 19: Turn LED ON.

Line 20: *else* part of *if statement* (when the value is equal to "0").

Line 21: Turn LED OFF.

***Now connect the Arduino board to the computer, give it a few seconds to detect before running code.**

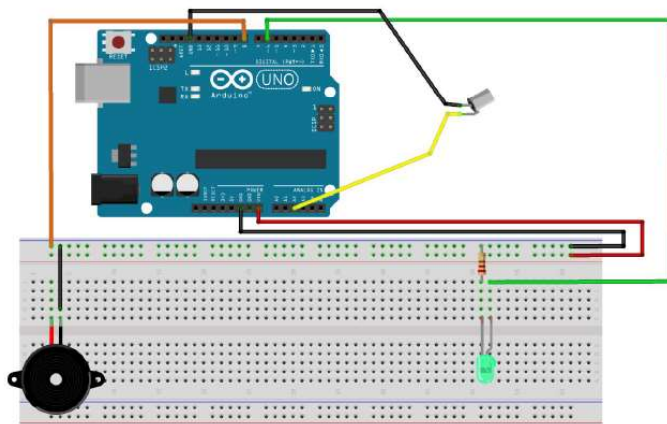
Activity 3: Earthquake (Disconnect the Arduino UNO R3 from the PC)

In this activity, you will code to switch the LED ON and sound the alarm when the tilt sensor is in an OFF state (meaning when its value is not equal to zero "0").

INPUT	PROCESSING	OUTPUT
Tilt sensor	What's happening on the prototype?	Screen output (Python shell)
ON state	LED OFF and no sound from the buzzer.	Displays "SAFE".
OFF state	LED ON and sound from the buzzer.	Displays the "DANGER".

Remember to use a *while loop* so that the value is always being updated.

Build: Same build as Activity 2 with the following addition: buzzer




Your code: Save the script as *Act3Earthquake* (You can modify the code from Activity 2)

Test your code:

Now connect the Arduino board to the computer, give it a few seconds to detect before running code.

Appendix S – Self-evaluation workshop session five



Workshop session 5

Rate your accomplishment based on the three activities from this workshop.

- 1- Very difficult
- 2- Difficult
- 3- Neutral
- 4- Easy
- 5- Very Easy

***Required**

ID Code *

Your answer _____

Activity 1: Reading form a tilt sensor *

	1	2	3	4	5	
Very difficult	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very easy

Activity 2: Tilt lights ON *

	1	2	3	4	5	
Very difficult	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very easy

Activity 3: Earthquake *

	1	2	3	4	5	
Very difficult	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very easy

Submit

Never submit passwords through Google Forms.

This content is neither created nor endorsed by Google. [Report Abuse](#) - [Terms of Service](#) - [Privacy Policy](#)

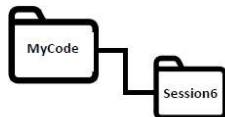
Google Forms

Appendix T - Workshop session six

Workshop session 6

© Copyright 2020 by Reginald G. Govender

Create the following subfolder named "Session6". All programs (scripts) created this workshop session will be saved in the "Session6" folder.

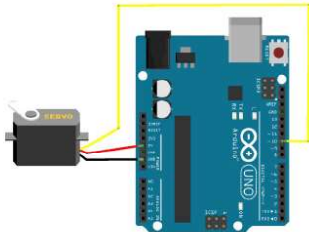


Activity 1: Turn to a degree (Disconnect the Arduino UNO R3 from the PC)

In this activity, you are going to prompt the user to input a degree to turn.
The servo motor will then turn to the specified degree.

INPUT	PROCESSING	OUTPUT
Keyboard	What's happening on the prototype?	Screen output (Python shell)
angle	The motor turns according to the angle entered.	Value is displayed on the screen

Build: Attach the servo horn arm to the servo motor *after* it automatically moves to 0°.



Servo motor



When a servo motor is powered it will automatically move to the starting position. **This position represents 0°** (imagine a protractor). The servo motor only turns to $\pm 180^\circ$ and back in the same direction to 0°. *It's like a swinging gate!*

*****DO NOT FORCE THE MOTOR BY HAND*****

Servo horn arm



Attach the horn arm blade to the servo motor *after* it automatically moves to 0°.

Code: Save the script as *Act1Degree*

```
1 import time
2 import pyfirmata
3
4 board= pyfirmata.Arduino('COM4')
5 time.sleep(2)
6
7 motorpin= board.get_pin('d:10:s')
8 print('Motor automatically positioned at 0 degrees')
9 while True:
10     angle= input("Enter angle:")
11     motorpin.write(angle)
12     print('Motor opened at:',angle,'degrees')
```

Explanation of code:

Line 1: Making use of the time package.

Line 2: Making use of the Pyfirmata package.

Line 4: Declare a variable called *board* that is assigned to the Arduino attached to your PC.

Line 5: Delay the code for 2 seconds. This is for Python to establish a connection between the Arduino and the PC.

Line 7: Declare a variable *motorpin*. The *d* refers to the digital pin, *10* is the pin that the jumper is connected and *s* refers to servo.

Line 8: Display a message on the screen that the motor is positioned at 0°.

Line 9: Start of *while loop*. You will notice that *flag* is never set to false within the *while loop*. Therefore lines 10 to 12 will continually run.

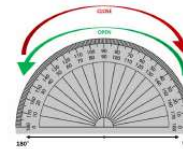
Line 10: Declare a variable called *angle* to hold/store the user input.

Line 11: Writes *angle* to the pin where the motor is connected.

Line 12: Display a message.

Now connect the Arduino board to the computer, give it a few seconds to detect before running the code.

If you enter a degree value greater than 180° the motor will only open to it's maximum which is about 180°.



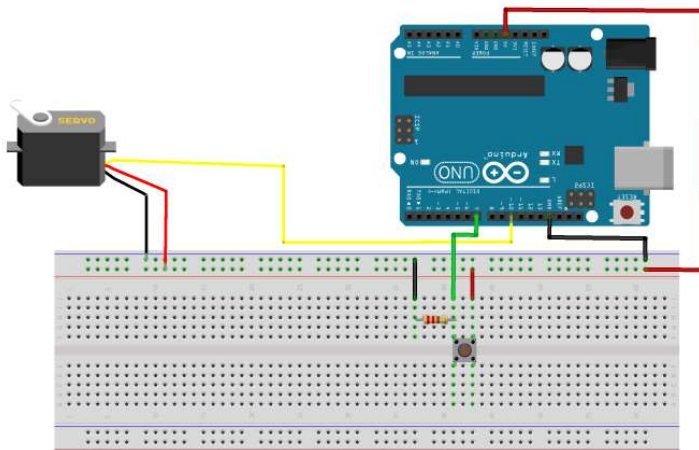
If you enter a degree value less than 0° you will encounter an error. This error is known as a run-time error.

Activity 2: Button open and close [Disconnect the Arduino UNO R3 from the PC]

In this activity, you are going to code such that when the user holds down on a button the servo motor must swing open. When the user releases the button the servo motor must return to its starting position (0°).

INPUT	PROCESSING	OUTPUT
Push button	What's happening on the prototype?	Screen output (Python shell)
Pressed	The motor will swing open.	OPEN
Released	The motor will return to its starting position (0°).	CLOSE

Build:



Push button



The push button comes with four legs which can be placed on the breadboard. The legs that are directly opposite each other are connected on the inside. Therefore you will notice in the build we will only use one side of the legs. You can also attach a cap/cover on the button. The push button can have two states True (button pushed down) or False (button released).

Code: Save the script as *Act2Button*

```
1 import time
2 import pyfirmata
3
4 board= pyfirmata.Arduino('COM4')
5 time.sleep(2)
6
7 it = pyfirmata.util.Iterator(board)
8 it.start()
9
10 motorPin = board.get_pin('d:10:s')
11 buttonPin = board.get_pin('d:7:i')
12 print("Hold down button to open")
13 while True:
14     buttonState= buttonPin.read()
15     if buttonState== True:
16         print('OPEN')
17         motorPin.write(180)
18     else:
19         motorPin.write(0)
20         print('CLOSE')
```

Explanation of code:

Line 1: Making use of the time package.

Line 2: Making use of the Pyfirmata package.

Line 4: Declare a variable called *board* that is assigned to the Arduino attached to your PC.

Line 5: Delay the code for 2 seconds. This is for Python to establish a connection between the Arduino and the PC.

Line 7: Declare a variable *it* which sets up an iterator thread that will be used to read the status of the inputs of the circuit.

Line 8: Start the iterator otherwise the board will keep sending data to your serial until it overflows.

Line 10: Declare a variable *motorPin*. The *d* refers to the digital pin, *10* is the pin that the jumper is connected and *s* refers to servo.

Line 11: Declare a variable *buttonPin*. The *d* refers to the digital pin, *7* is the pin that the jumper is connected and *i* refers to input.

Line 12: Display message.

Line 13: Start of *while loop*. You will notice that *flag* is never set to false within the *while loop*. Therefore lines 14 to 20 will continually run.

Line 14: Declare a variable called *buttonState* to hold/store the state of the button.

Line 15: *if statement* to check the state of the button.

Line 16: Display "OPEN" on the screen.

Line 17: Turns the motor to 180 degrees.

Line 18: *else* part of *if statement*.

Line 19: Return (close) the motor to starting position by writing 0 degrees.

Line 20: Display "CLOSE" on the screen.

Now connect the Arduino board to the computer, give it a few seconds to detect before running code.

Activity 3: Fibonacci in motion Disconnect the Arduino UNO R3 from the PC

In this activity, you will code the servo motor to turn accordingly to the first 12 terms of the Fibonacci sequence. Your program must display the Fibonacci number on the screen and make a beep sound after every motor turn. No user input is needed.

Further explanation: 0 (no move), 1 (one-degree move), 1 (one-degree move), 2 (two-degree move), 3 (three-degree move), 5 (five-degree move), and so on.

Fibonacci:

The Fibonacci sequence (1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144 ...) can be seen in so many places- in nature, art, engineering, music and mathematics! We make each term in the series by adding together the two previous terms: $1+1 = 2$, $1+2 = 3$, $2+3 = 5$, and so on

Fibonacci code for the 6 first terms

```

1  nterms = 6
2  n1 =0
3  n2 = 1
4  count = 0
5  while count < nterms:
6      print(n1)
7      nth = n1 + n2
8      n1 = n2
9      n2 = nth
10     count += 1
    
```

Explanation of lines:

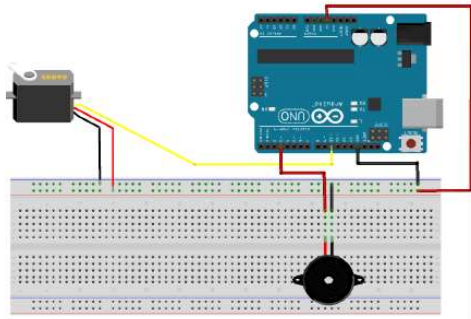
```

1  set the number of terms
2  first term is set
3  second term is set
4  sets the counter for loop
5  while count < nterms
6  screen output
7  next term is calculated
8  n1 updated
9  n2 updated
10 counter incremented
    
```

INPUT	PROCESSING	OUTPUT
Keyboard	What's happening on the prototype?	Screen output (Python shell)
-	Servo motor turns and buzzer sounds	Fibonacci term

*Remember to use a time delay between switching the buzzer ON and OFF.

Build:



Your code: Save the script as *Act3Fibonacci*

Test your code:

Now connect the Arduino board to the computer, give it a few seconds to detect before running code.

What will you see and hear?

You will see on the screen	You will hear	You will see the motor
0	"buzzer beep"	move 0"
1	"buzzer beep"	move 1"
1	"buzzer beep"	remain at same position
2	"buzzer beep"	move 2"
3	"buzzer beep"	move 3"
5	"buzzer beep"	move 5"
8	"buzzer beep"	move 8"
13	"buzzer beep"	move 13"
21	"buzzer beep"	move 21"
34	"buzzer beep"	move 34"
55	"buzzer beep"	move 55"
89	"buzzer beep"	move 89"

Appendix U – Self-evaluation workshop session six



Workshop session 6

Rate your accomplishment based on the three activities from this workshop.

- 1- Very difficult
- 2- Difficult
- 3- Neutral
- 4- Easy
- 5- Very Easy

***Required**

ID Code *

Your answer _____

Activity 1: Turn to a degree *

	1	2	3	4	5	
Very difficult	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very easy

Activity 2: Button open and close *

	1	2	3	4	5	
Very difficult	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very easy

Activity 3: Fibonacci in motion *

	1	2	3	4	5	
Very difficult	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very easy

Submit

Never submit passwords through Google Forms.

This content is neither created nor endorsed by Google. [Report Abuse](#) - [Terms of Service](#) - [Privacy Policy](#)

Google Forms

Appendix V – Post- survey (survey two)

Computer programming survey two

WRITE YOUR
CODE HERE

Please indicate how you feel about computer programming by the extent of agreement using the following words: Strongly agree, Agree, Undecided, Disagree and Strongly disagree. Put a tick only in one box for each statement. If you make a mistake put a cross through the marked box and then tick in the correct box.

No	Statement	Strongly agree	Agree	Undecided	Disagree	Strongly disagree
1	I am comfortable with learning programming concepts.					
2	I have little self-confidence when it comes to programming.					
3	I can learn to understand programming concepts on my own.					
4	I think can achieve good grades in programming.					
5	I am confident that I can solve problems by using programming.					
6	I hope to use programming sometime in my future.					
7	The challenge of solving problems using programming does not appeal to me.					
8	I can use the thinking developed when programming in my daily life.					
9	I think computer science (programming) is interesting.					
10	I would voluntarily take additional Computer Science courses to learn programming.					
11	I was more interested in the robotic element (microcontroller) than the programming.					
12	I found programming attractive through the use of the microcontroller.					
13	The use of the microcontroller held my attention for longer.					
14	Easier for me to comprehend and retain information in coding through the use of a physical device (microcontroller).					
15	Programming the microcontroller was a positive experience for me.					

1 of 2

No	Statement	Strongly agree	Agree	Undecided	Disagree	Strongly disagree
16	I could come up with a suitable strategy for a given programming project in a short time.					
17	I could manage my time efficiently if I had a pressing deadline on a programming project.					
18	I could find ways of motivating myself to program, even if the problem area was of no interest to me.					
19	I could complete a programming project if someone showed me how to solve the problem first.					
20	I could complete a programming project if I could call someone for help if I got stuck.					
21	I think mathematics is not needed to be able to program.					
22	I can translate the mathematical content into a programming code.					
23	I think basic mathematics is needed to be good at programming.					
24	I think you would need more than basic mathematics knowledge to program.					
25	I think the mathematics involved in programming is technical.					
26	The microcontroller allowed for the understanding of programming.					
27	I would had understood the programming concepts without the robot element (microcontroller).					
28	The microcontroller provided visual aid of what my programming was doing.					
29	I could rewrite lengthy and confusing portions of code to be readable and clear.					
30	I could write a program that someone else could comprehend and add features to it later on.					

Thank you

Appendix W - Post survey (survey two) categorised under constructs

POST SURVEY

Based on the review of the literature, the following five constructs were identified:

- A. **Confidence-Student confidence in his/her ability to learn programming.**
- B. **Interest-Student interest in programming.**
- C. **Motivation- Student motivated by the use of physical computing.**
- D. **Belief- Student intrinsic belief to solve problems.**
- E. **Mathematics- Student perception of mathematical influence on programing.**
- F. **Knowledge- Students knowledge of programming through the use of physical computing.**

A. Confidence-Student confidence in his/her ability to learn programming.

1. I am comfortable with learning programming concepts.
2. I have little self-confidence when it comes to programming.
3. I can learn to understand programming concepts on my own.
4. I think can achieve good grades in programming.
5. I am confident that I can solve problems by using programming.

B. Interest-Student interest in programming.

6. I hope to use programming sometime in my future.
7. The challenge of solving problems using programing does not appeal to me.
8. I can use the thinking developed when programming in my daily life.
9. I think computer science (programming) is interesting.
10. I would voluntarily take additional Computer Science courses to learn programming.

C. Motivation- Student motivated by the use of robots.

11. I was more interested in the robotic element (microcontroller) than the programming.
12. I found programming attractive through the use of the microcontroller.
13. The use of the microcontroller held my attention for longer.
14. Easier for me to comprehend and retain information in coding through the use of the physical device (microcontroller).
15. Programming the microcontroller was a positive experience for me.

D. Belief- Student intrinsic belief to solve problems.

16. I could come up with a suitable strategy for a given programming project in a short time.
17. I could manage my time efficiently if I had a pressing deadline on a programming project.
18. I could find ways of motivating myself to program, even if the problem area was of no interest to me.
19. I could complete a programming project if someone showed me how to solve the problem first.
20. I could complete a programming project if I could call someone for help if I got stuck.

E. Mathematics- Student perception of mathematical influence on programming

21. I think mathematics is not needed to be able to program.
22. I can translate the mathematical content into a programming code.
23. I think basic mathematics is needed to be good at programming.
24. I think you would need more than basic mathematics knowledge to program.
25. I think the mathematics involved in programming is technical.

F. Knowledge- Students knowledge of programming through the use of robotics

26. The microcontroller allowed for the understanding of programming.
27. I would have understood the programming concepts without the robotic element (microcontroller).
28. The microcontroller provided visual aid of what my programming was doing.
29. I could rewrite lengthy and confusing portions of code to be readable and clear.
30. I could write a program that someone else could comprehend and add features to it later on.

Appendix X – Post-test (questionnaire on programming language)

WRITE YOUR
CODE HERE

Instructions: Please write your code in the top right block.

Multiple choice:

Choose the most appropriate answer by placing a tick or circling your response.

1. Which of the following could NOT be stored in a variable?
 - A Text
 - B Number with a decimal
 - C Boolean
 - D Conditional

2. Runs the same sequence multiple times
 - A Conditional
 - B Bug
 - C Loop
 - D Function

3. Changes the flow of the program based on a true or false statement
 - A Event
 - B Condition
 - C Operator
 - D Variable

4. $Y = 5 * X - 5$
Given that X is a number between 0 and 1 inclusively. What is the range of Y?
 - A between 0 and 1 inclusively
 - B between -5 and 0 inclusively
 - C between 0 and 5 inclusively
 - D between 5 and 10 inclusively

5. Alongside are different stages in software development:
Which of the following gives the correct order of stages?
 - A 1→5→4→2→3
 - B 5→ 1 →4→2→3
 - C 3→1→4→2→5
 - D 4→1→3→2→5

- | | |
|---|--------------------|
| 1 | problem analysis |
| 2 | program coding |
| 3 | program testing |
| 4 | algorithm design |
| 5 | problem definition |

6. Given the table below representing memory allocation.

X	Y	Z
2	5	

Which of the following program segment(s) interchange(s) the contents of variables X and Y without deleting any contents?

- (1) $X = Y$ (2) $X = X - Y$ (3) $Z = X$
 $Y = X$ $Y = X + Y$ $X = Y$
 $X = Y - X$ $Y = Z$

- A (1) only
B (2) only
C (1) and(3)only
D (3) only
7. In Python, which on the following lines will output "Hello"?

- A `print "Hello"`
B `output (Hello)`
C `print ('Hello')`
D `output "Hello"`

8. Consider the following code segment:

```
x=0  
y=1  
x=3*y
```

After code execution which of the following statement/s is always true?

- I. x is a declared variable.
II. y is a declared variable.
III. The value of x depends on the value of y.

- A II only
B III only
C I and II
D I, II and III
9. Consider the following code segment:

```
A=3  
B=7  
flag= A>B  
while flag:  
    print('.')  
    B=B+1  
print('...')
```

After the above code is executed, the output is:

- A ...
B
C ...
D Continuously outputs:==
|

10. A process needs to be repeated 10 times. Which construct can be used?

- A

```
if count < 10:  
    print ('Programming')
```
- B

```
count = 1  
while (count < 10):  
    print ('Programming')  
    count = count + 1
```
- C

```
for count in range(10):  
    print('Programming')
```
- D Neither A, B or C

Appendix Y - Interview

Semi- structured focus group interview

- 1.** Have you had a workshop experience before? Could you please share it with us?
- 2.** What is the advantage for you in completing this workshop outside the scope/objectives?
- 3.** Do you think you would had understood the programming concepts without the use of the robotic element (microcontroller)?
- 4.** Based upon your current understanding (after the workshops) of programming, what challenges do you foresee? How will you resolve the situation?
- 5.** How did you actually approach the tasks in each workshop (strategies used to solve your design problem)? I would appreciate if you could share them with me in as much detail as possible.
- 6.** What do you like the most and the least about the workshops that you have completed? Why?
- 7.** During course, has your understanding, knowledge about programming ever changed? Please share with us how that has (has not) changed.
- 8.** Having completing this course, what do you perceive the level of difficulty of computer programming to be?

Appendix Z - Interview transcript

Researcher: Good afternoon, everyone. We will start in the next minute

[after a minute]

Researcher: Okay. Good afternoon, everyone. We will go ahead and start and then as people join us, they can then share. First of all, thank you for taking the time to join. So I'm going to share with you the eight questions. I'm going to keep it up on the screen. Please take some time to read the questions on the screen and reflect.

[after ±2 minutes]

Researcher: Let's start; please feel free to unmute and discuss and share your thoughts, etc.

P17: Right, good afternoon, guys. I hope everyone is well. In terms of how they feel about the entire shop. Okay. I completed all workshops, which were very fun activities to participate in. I've had, I've had quite a few workshops before and some workshops was in the form of summer camp workshops. They just go and have a particular workshop and come back. But what I will say is this was my first time having to attend the workshop 100% online. So, which was a new experience for me. But it was an amazing experience. A lot of the things we were doing that was interesting to learn how we can use basic programming to do tasks.

P9: Just like add the robot kit used they're very applicable in real-life problems. Also, it was a very exciting workshop for me, compared to others because I've never been to an online workshop, but I feel it was essential. I would have understood and caught with Python without the microcontroller, but it wouldn't have been so fast, would take me sometime.

P62: I did a computational engineering workshop, but I haven't done a hands-on Arduino board or using a breadboard and advance stuff. I would have not liked a theoretical course on Arduino microcontrollers. It's not the same as this as this was a hands-on course, you know. It is how mechanical computers and electronics come together.

Researcher: Okay.

P17: I would say this workshop is an example of a workshop that helps you think. Guided towards solving problems in this workshop and then given a chance to apply and apply more of our knowledge into solving a bigger problem, like you guys have seen almost every workshop was guiding us towards starting the coding and then giving you a chance to apply your own logic. That's all I can say; this was an interesting workshop compared to most of the workshops I've attended.

P18: Good afternoon, everyone. Hope you guys can all hear me.

Researcher: Yes, yes, we can.

P18: Okay guys, before I share my experience with you guys. I haven't attended a new workshop. This was my first one, and it was so exciting. It was an amazing experience. From the beginning, everything was step by step and everything was explained well. Yeah and we also had Zoom sessions to help us understand or clarify what is it that is expected from us in each and every workshop. So it was a great experience and it was amazing. That's all I can say. Thank you.

Researcher: Thank you. Anybody else wants to, share your experience?

P62: My problem is that it installs Python to a different path. I actually run the software from my flash drive. I just don't recommend that.

P42: Hi everyone! I just like to talk about it. The issue of my one of my main problems is that using the breadboard. Usually, whenever I was stuck in a problem. It wasn't because my code was faulty; it was one of my connections with it. Just because I didn't understand, sometimes I'd be stuck for hours. Sometimes, all I would do is just change the position of the connections and then suddenly would work. I wouldn't change my connector all, but changing positions on the breadboard would make it work.

Researcher: Thank you for that. Sometimes, you just need to force the wire pin into the breadboard. Sometimes your code, there'd be nothing wrong, but just one wire is not connected correctly, which causes your entire prototype not to work. Anybody else wants to share the experience?

P62: I struggled with some programming; it can get quite abstract, but the microcontroller makes it most fun and I feel like it solidifies understanding in some way. And just having a robotic, you can now monitor the behavior and see. I mean, you kind of run code for a compiler, but it's not the same as not making it do something in reality and real off activating something or driving something. That's really cool. It just brings it all to laugh. It makes the most fun and I feel like it solidifies understanding in some way.

P15: Good afternoon, everyone. I found that this workshop was interesting to me. It's something that I think I wanted to learn as coding was interesting for me. It is something that I have wanted to learn for a long time and I think it is a very valuable skill. I really think that we had enough support to do all the tasks that we did for us to do. I think even... I'm sure that if some never did in coding like us, but through the support and the way things were in the workshop it would be doable. There were clear instructions and everything. Even if you have done what you're pointing to, it is allowed you to have that some sort of confidence and apply your mind so that you can do actual, which was a little bit harder. But I think you know it was a great experience. I'm not sure about anything that can be improved. Others can comment and it's but thank you very much.

Researcher: Thank you anyone else?

P42: I would like to add something?

Researcher: Sure, go-ahead

P42: So I always had this fear of electronics before completing this workshop and just doing this workshop is helped me get over that. Because I have tried to fix household appliances before and I've gotten shocked a lot, but then completing this workshop helped me overcome this fear of electronics. I always thought it was like overly complicated and I tried watching stuff on YouTube, but now it is understandable. All those diagrams, whatever. But this workshop helped me by giving me a push in the right direction. Like, I know what to look for and it just taught me a lot in understanding coding and electronics.

P60: Hello everyone. First of all, I like to say I really enjoyed this workshop it was an interesting journey, learning to work with a microcontroller. It was a bit difficult and it did require a bit of thinking, but nothing was impossible. So this was the first time I did a workshop like this, but previously, I've worked as a technician and installed smoke detectors and other types of sensors but did not code them. So whatever we learned in this workshop, like it was really interesting and I know that it could be used in many different ways, like, you know,

Researcher: Thank you for that. You would know that most of these components used in the activities are used in some of the devices we use daily.

P18: The thing that I liked the most was you could read the programming language and easily understand what it does. Using Python, which might be an advantage when we are going to their field. If we encounter any situations, we have previous experience of using Python. Then we can just interact and yeah, that's the advantage.

Researcher: Thank you for that anybody else want to comment?

P47: Yes.

P17: Okay, you go first.

P47: Good afternoon, everyone. Okay. At first, the activities were very challenging for me. But now that I learned a coding language by engaging... I now have started a new project on my own because each and every workshop, we were required to have a different output. Most of the time I didn't want to skip a step without understanding so that's when I needed the help. After doing the workshop, I saw the need to go back to the drawing board and actually study Python in depth and because I would like to develop/build things. And I would like to program solutions to whatever problems we're facing in the future. So it was really a great opportunity to be part of such a wonderful workshop for free, for that matter.

Researcher: Thank you for that.

P17: Yes. Yes. Thank you. It's quite similar to that. I feel like after landing on one program. I feel like Python and doing this stuff (prototyping) is relevant to the real world now

because it's more advanced. Like industrial programming. It was an amazing experience to see that you can actually write some code and apply it. The physical changing of the light using the LDR and coding of the buzzer was very interesting. I feel like I can use that information in developing solutions in the future.

P66: My experience with it was so far... I really enjoyed it. It was something new to me at the beginning I did have a challenge because my board wasn't working reason was I was unable to get my comport number. So that was my challenge and needed to be manually replaced. That was my challenge at the beginning and otherwise, everything was interesting and fun. Some activities were challenging, though, but the forums and tips were really helpful. It was helpful to see what challenges others were facing during the forums and what you could use from this information and improve. The pre-recorded videos in the pre-workshop were really helpful. Otherwise, I enjoyed my experience and I've learned a lot and it was a new experience for me. And I think I take this along; it'll help my future. Thank you.

Researcher: Okay, thank you for that.

P15: In high school, I did Java as a programming language. But now doing Python, it's less complicated and simple...everything is straightforward.

P21: I feel it was easy for me to understand Python by looking at how the microcontroller executes. Execute the commands that you're writing on your code. I feel like it makes it easier for someone to understand that I'm writing this line of code.

P17: It was very useful to see those explanations you put along with the code and then explain line by line, this line of code to do this and this for that. The physical aspects of the workshop were even more helpful because we could easily explain that this line of code is during this but when the prototype performed an actioned ... actually understand well yeah this is what it's supposed to do. And it's actually doing. So I feel like the hard way is doing justice to microcontrollers useful to understand the program.

P5: I hope all are good! What I can say is that I have never done any workshop like this before, but I do appreciate that it had come to find me because like I do love coding. I love using computers and doing unthinkable things like programming. Things that people couldn't think of. So I always wanted to do it. But then I couldn't quite qualify for that... so it was a good experience.

Researcher: Thank you for that. Anybody else wants to share?

P1: The robotic or microcontroller helped me understand if something is wrong in my code to see if the end product works. I would refer back to the prototype design and then check my code. In the end, so without the microcontroller. I don't think I would have been able to understand and even do the workshop but the coding somethings is tedious and then it's not like you can really see your progress when you stuck.

P18: I did have a strategy that I used when going through every activity in every workshop... strategy I used was to build my prototype then; I will code because after building, I would have an idea of what the code must do and what must take place.

Researcher: Thank you for that will; then anybody else wants to share their strategy.

P42: By sorting the things I need for the build, then build and code. Also, create comments in code if the code was the same thing, just to feel more clearer on the way.

Researcher: Thank you for that.

P17: My strategy ... I was building everything onto the same build and I wasn't removing anything unless I had to remove a component. It's was nice but got a little a bit tricky about workshop three. I had a very messy breadboard. I had a lot of things connected, but I knew very well what each and every component was doing. So, one may say it's confusing to do like that. It was interesting to be able to just click around on my code and then I see an execution on the prototype.

P62: Um, I think I read through the first workshop quite a bit, but about a week and I was like, You know what, I have this. I read the question correctly, but I always read the headings carefully, look at what you're trying to do and what you're trying to achieve, what the goal is like a fire alarm and the buzzer goes off. Visual. Visual

person. And for me, just trying to visualize what I wanted the end trying to predict the outcome. I'm just; it just creates like an end goal and insights. This definitely helps a beginner like me because we are in a very technological age. So I think this is valuable to know. I mean, for me.

P42: So prior to the workshop, I had tried to learn the stuff, I mean, there's multitudes of information available internet, especially for Python as well. But like, it was just it on point.

P62: They'll always be the challenges, while you're learning something new, um, I mean, my first go to just be a really good computer or software textbook, but I think the challenges is definitely time you have to then read up and research into it and look at what works, maybe watch some YouTube videos. Like this, um, the robot just makes the learning and understanding quicker. I think

P42: In some workshops, the coding started out to be really motivating at the beginning and then eventually, I will just get bored of it because although it was amazing to code, it sometimes gets a bit!

Researcher: Please explain a bit

P42: And then it's not like you can really see your progress. I mean, it's very hard to explain that. But, but then with this workshop having to hold a microcontroller, it reminds you that your code is relevant. The thing with the bread is every time you ran your code; you got to see it work.

P56: I felt like I was very highly motivated to just complete the workshop activity and then jump to the next one. Just for you to see it actually works and what new things I could do further.

Researcher: Thank you for that.

P62: It was useful in this course having some foundation of coding. Each workshop would give you the foundation and get an idea of how this component connects to the breadboard... and then with that, you can then have the freedom to play; that's when the fun comes. So actually, after the course after the workshop, I actually

started to try and modify and build and code on my own. Actually, I am now working on a prototype with the buzzer.

P16: Greetings, everyone. I think it was late 2013 I started an online learning course but left because I didn't have the equipment. So, this workshop helped me so much. The microcontroller helped me and I enjoyed the workshop so much because the equipment was given. Also, I was able to study by myself get motivated as I was able to write programs and see them execute.

Researcher: Thank you for that.

P18: Hi everyone. Okay, there was this component, the flame sensor it was problematic cause of its sensitivity. I think yeah, that was the most challenging workshop.

Researcher: Thank you for that. Yes, that was because you had to imitate a natural flame by using a torch.

P17: Using the tilt sensor was fun, I cannot remember the entire setup, but I don't think it was just one activity. You're supposed to be like coming up with a specific value to activate it. So I feel like I've got a better understanding of what actually happens inside our smartphones in other in other. Also, I feel my favorite activity was the LED; it was interesting to see the lights come on like a traffic light. The least exciting activity for me was activity one, maybe because it was supposed to be like the activity to get us to begin the workshop and I give us just basics.

P47: My favorite was the ones with the sound. It goes to show that you can code a particular sound to make a melody... actually create different sounds and stuff with that buzzer.

P42: I'd actually like the flame one. I think it was activity three where we used it in conjunction with the buzzer. I also liked when we started to use more than two components. I think it was the first time using three. So it was quite fun to have all the connections and all the coding and remember all that stuff because I just

P26: Yeah, my favorite activities were every activity of the entire session. I enjoyed the problem solving; I enjoy combining things and then putting them together to make up one thing. So I enjoyed it and the challenges.

Researcher: Okay, thank you for sharing.

P62: The questions were really interesting, like the puzzles and solving; we had to try and see to figure things out.

P16: My favorite activity was programming traffic lights so because I always wonder how the traffic lights went so now I have a basic idea of how. My least favorite one was when I had to use the motor because my motor didn't work at first, but at a later stage, it worked.

P15: I think the most exciting workshops for me were the ones that included light coming into your room. It was a light sensor (LDR) but it was really interesting because you can use it. I got an idea of how they (robotic element) are really used in real-world situations ... so you get to know how things work in today's tech world. With the robot combined with programming, you may get everyday solutions. I also liked using the fire sensor activities.

Researcher: Are there any more comments? Or anything else anybody wants to mention or share? (waited for about a minute)

Researcher: Okay, thank everyone for availing yourself to this info-sharing Zoom session and participating in this workshop. Goodbye and enjoy the rest of your afternoon.