5-2023

Immersive Learning Environments for Computer Science Education

Dillon Buchanan
East Tennessee State University

Immersive Learning Environments for Computer Science Education

---

A thesis

presented to

the faculty of the Department of Computing

East Tennessee State University

In partial fulfillment

of the requirements for the degree

Master of Science in Computer Science

---

by

Dillon Buchanan

May 2023

Keywords: computer-aided instruction, human-computer interaction, computer science education, integrated development environments, data structures

ABSTRACT

Immersive Learning Environments for Computer Science Education

by

Dillon Buchanan

This master's thesis explores the effectiveness of an educational intervention using an interactive notebook to support and supplement instruction in a foundational-level programming course. A quantitative, quasi-experimental group comparison method was employed, where students were placed into either a control or a treatment group. Data was collected from assignment and final grades, as well as self-reported time spent using the notebook. Independent t-tests and correlation were used for data analysis. Results were inconclusive but did indicate that the intervention had a possible effect. Further studies may explore better efficacy, implementation, and satisfaction of interactive notebooks across a larger population and multiple class topics.

DEDICATION

To Chloé Clark, who I met as Chloé Gamroth, and Stephanie Alu – two women with beautiful hearts who taught me my worth.

ACKNOWLEDGEMENTS

Thank you to Phil for spending the summer proofreading the notebook used in this study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

Computer science has seen rapid growth in recent years due to the rise of technology and automation, resulting in an increased demand for individuals with skills and knowledge in this field. According to the Bureau of Labor Statistics, there are over 1.4 million computer science jobs but only around 400,000 computer science graduates [1].

A study conducted by the National Center for Education Statistics (NCES) found that the average retention rate for students enrolled in introductory computer science courses was only 67% in 2018. This is lower than introductory retention rates in any other STEM field (e.g., physics and mathematics, where the retention rate was 75%). The NCES study also found that student satisfaction in introductory computer science courses was low, with only 40% of students reporting that they were "very satisfied" with their courses. Students in computer science programs reported the lowest levels of engagement and sense of belonging compared to other STEM fields [2, 3].

These findings suggest that there is a problem of practice in computer science education, particularly in lower-level programming courses, that requires attention. One of the major challenges facing computer science education is the lack of opportunities for students to practice and receive feedback. Traditional approaches to teaching computer science often rely on lectures and rote memorization. Lower division programming courses require students to understand complex concepts, and students can struggle when applying lecture topics to practical exercises. Some dissonance exists between concepts taught in the lecture hall and applying the concepts through writing code. This issue is especially true of students enrolled in lower-level computing courses. The high cognitive load of these topics and a lack of practice opportunities can lead to disengagement and difficulty in understanding the material.

Also, people who teach computer science in post-secondary settings are not, as a rule, conversant in learning science. Understanding these theories can help computer science educators design effective learning experiences tailored to their learners' needs and learning styles. To address these challenges, this thesis study aims to create an immersive learning environment to provide personalized learning opportunities for students to explore beyond the curriculum and to remediate their understanding where necessary.

The Department of Computing at ETSU has observed a significant number of students struggling with data structures, one particular introductory course [4] which has historically lead to a pass rate of 65% – a prominent example of the challenges described above. Department leadership postulates that high instructor turnover for this course and a traditionally lecture-heavy format are root causes for this. To alleviate these concerns, department leadership suggested the data structures course as the environment for this study.

To address these challenges, this thesis study designed, implemented, and assessed the use of emerging technologies that provide personalized learning opportunities for exploratory and dynamic practice. Emerging virtual learning technologies can be used to create texts for computer science courses. This study explored both the creation and efficacy of an immersive learning environment as an instructional asset in a lower-level programming course. To do so, this study required the creation and implementation of an interactive notebook to serve as an immersive learning environment.

Interactive notebooks are development environments that contain both text and encapsulated, executable code sections. This allows for lecture information to be close to code examples and practice exercises. One example, Jupyter Notebook, was developed for Python in 2014. Recently (2021), Microsoft has released a new notebook format called Polyglot Notebooks (formerly .NET Interactive Notebooks), which supports multiple languages. These emerging technologies can be used to create instructional assets for computer science courses. The notebook was following the instructional design patterns presented by Xie et al. Relevant examples and exercises were placed close to the content. Following experiential learning theory, this manner of composition provides opportunities for concrete learning, reflective observation, and active experimentation.

Research Questions and Hypothesis

This study sought to determine the following questions:

- **Q1**: Is there a significant difference in learning outcomes between students provided with an interactive learning environment and those that are not?
- **Q2**: Will interaction time positively correlate with learning outcomes among students who are provided with the interactive learning environment?

Hypothesis: The use of an interactive learning environment will result in improved learning outcomes for students compared to traditional, lecture-based instruction. Furthermore, student outcomes will correlate positively with usage among those provided with the interactive learning environment.

Definition of Terms

This study will leverage the following terms:

- Data Structures: Programming constructs for organizing and storing data [5].
- Contiguous Memory: A block of memory that a data structure occupies where all elements are physically side-by-side [5].
- Memory Reference: A variable that stores an address to a memory location that contains pertinent data [5].

- Algorithm: A sequence of programmatic steps prescribed to solve a problem. [6]
- Immersive Learning Environment: A learning mechanism that requires exploration and increasing mastery, maintains user engagement, allows users to alter the environment, and provides opportunity for accomplishment [7].
- Experiential Learning: "Experiential learning is the process of learning through experience, or more specifically, learning through reflection on doing" [8].
- Interactive Notebook: An application that allows for multi-modal content blocks (such as code, text, and other static media) to be placed adjacent to one another.

Significance of Study

This study aimed to provide insights into effective pedagogical approaches that can be used to teach introductory concepts such as data structures in computer science education. By integrating immersive learning contexts and reflection in teaching data structures concepts, the study aimed to improve student learning outcomes and engagement, which can ultimately contribute to the employability of computer science graduates. Additionally, this study can help bridge the gap between theory and practice in computer science education and inform computer science educators on effective pedagogical approaches for teaching foundational programming skills.

CHAPTER 2. BACKGROUND

Learning Science

Learning science is the study of how people learn and/or retain knowledge. Learning science is often informed by two schools of thought: cognitive neuroscience and cognitive psychology [9]. Scholarship in this field describes how people process and recall information, solve problems and make decisions, and apply what they learn. The resulting learning theories are frameworks that describe the application and efficacy of instructional methodologies. Some of the most influential learning theories include behaviorism, cognitivism, and constructivism. These theories differ in their assumptions about how learning takes place and what role the learner assumes in the learning process [10].

Experiential Learning

First introduced by Dewey [11], experiential learning reexamines traditional education by incorporating opportunities to acquire and apply new knowledge through experience. His theory emphasizes learning experiences, internships, and capstone projects that develop critical thinking skills, problem-solving abilities, and technical competence in transferable environments.

Informed by Dewey's work, Kolb [8] extended experiential learning to include observation and reflection. Kolb defines learning as "the process whereby knowledge is created through the transformation of experience" [8]. Two dialectic processes facilitate this transformation: action and reflection paired with experience and abstraction.

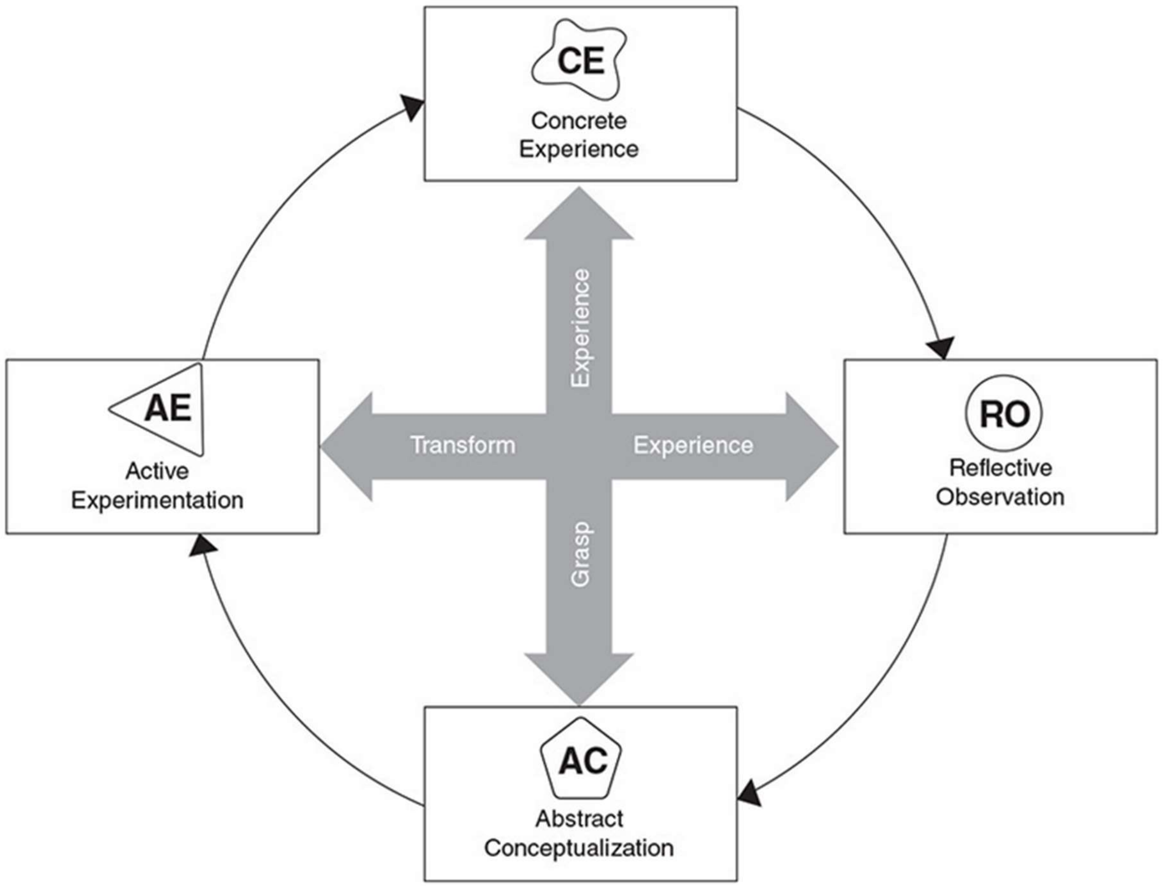


Fig. 2.1 – Kolb's Model of Experiential Learning [8]

Kolb organizes these processes into a recurrent cycle (viz. Figure 2.1). Concrete experiences provide a basis for making observations and reflecting on the event(s). Reflections provide a basis for conceptualization. Finally, the understanding gained from conceptualization can be tested by experimentation. Experiments lead to experiences, and thus the cycle continues. According to [8], individuals must engage in all four stages of the learning cycle to truly learn from their experiences.

Individualized Instructional Design

Dirksen [12] describes strategies for overcoming deficiencies, or "gaps", in students' backgrounds that inhibit their ability to learn. Apart from a lack of knowledge, these gaps include a lack of appropriate skills, poor learning environments, ineffective attempts to communicate course content, and insufficient motivation.

- Skill gaps involve a lack of facility with technique. Dirksen attributes these to insufficient practice. Instructors can accommodate for gaps by providing students with avenues for practice [13].
- Environmental gaps involve external, systemic barriers to learning. They include workplace discrimination, lack of funding, lack of equipment, and bureaucratic overhead. Instructors can inadvertently introduce these into a course – for example, by requiring an expensive textbook when a less expensive one might do.
- Communication gaps often arise due to "the curse of knowledge": i.e., a mismatch in fluency between an instructor's expertise and a student's lack of experience. One strategy for bridging this gap is to present content that a first-time learner may need to know and to clarify key terms and concepts – particularly esoteric computing terms like "class" that have different meanings depending on how they are used.
- Motivation gaps stem from causes that include a reluctance to change, anxiety, a lack of understanding or interest, and distraction. While instructors cannot change how students think and feel, even subtle changes in class structure and presentation can make content easier to access, giving students more opportunities to self-motivate.

A specific type of motivation gap stems from the need to unlearn old methods, processes, and habits to acquire new techniques. The reward for unlearning a process or habit, particularly one that works, can seem insignificant compared to the effort to do so – especially at first. For example, an introductory student will often name variables "x", "y", or "z", as is taught in algebra and calculus. In programming, a variable’s name should indicate its purpose, but transitioning to more meaningful, longer names can be difficult for some novices.

A fifth issue raised by Dirksen is the difficulty of addressing the needs of individual students in classroom settings. To manage this challenge, [12] recommends implementing strategies for transforming extrinsic into intrinsic motivation. Outside factors, such as grades and promotions, drive extrinsic motivation; self-esteem and pride drive intrinsic motivation. This transformation can be aided by engaging students in things they like – for example, by connecting assignments to personal interests. To do this effectively, instructors need to sense students' motivations, sentiments, and backgrounds, while being mindful that these may differ from their own. For example, an instructor could lead a discussion by soliciting examples from the audience. Also, the level of the material and the amount of guidance must be appropriate for each student. Personalized learning opportunities can provide students of different skill levels with an appropriate level of scaffolding.

[12] models learning as a process that transfers content to a person's long-term memory via their sensory and short-term memories. Due to the volume of sensory information that people process and the limitations of human short-term memory, little of what people perceive finds its way into long-term memory [14, p. 59]. Dirksen recommends a two-part strategy for presenting content that addresses these limitations. Part one uses surprising, significant, and/or unexpected stimuli to ensure that content reaches a person's short-term memory. Part two relates this content to what students have already learned and, if possible, experienced. Ideally, these relationships should be strong to promote retention. While repetition alone can force content into long-term memory, memories that are disconnected from other experiences will, over time, become cloudy

or fade altogether. Experiential learning helps promote memories from short-term to long-term memory by employing significant and interesting recurrent experiences and repeated exposure.

Additionally, [12] recommends appealing to people's intuition to ensure retention. Dirksen, like Kahneman, posits two modes of human decision-making: a fast, intuitive mode and a slower, rational mode. Kahneman refers to the former as an elephant and the latter as a rider—a metaphor that expresses the relative strength of intuitive cognition [15]. Dirksen recommends storytelling and social engagement as two ways of engaging this faster, "impulsive" mode of thinking. In support of social engagement, Dirksen cites evidence that Asian students perform better than other minority groups because they study together. Other recommendations for stimulating impulsive thought include the use of pertinent props, visual aids, humor, and rewards—though Dirksen cautions about the unpredictable nature of rewards.

According to [12], "Change is a process, not an event." This is a core tenant for teaching good habits and helping students unlearn undesirable ones. One crucial requirement for building good habits, self-efficacy, is maintaining confidence in one's ability to succeed. Instructors can build self-efficacy by providing students with early successes and allowing them to solve problems independently.

Habits can also be developed using trigger models: formulas of the form "If X, then Y." One type of trigger, an event trigger, instills a new habit by associating it with an existing habit: for example, by adding a new, helpful habit to an existing routine, or chaining a mitigating action to an old, undesirable habit. A second, an environmental trigger, uses a person's surroundings to cue action: e.g., posting signs in bathrooms to remind employees to wash their hands before returning to work.

Friction and cognitive load are opposing principles that must be balanced when designing instruction. Friction is the use of challenging content to promote retention. Cognitive load is the effort needed to meet a course's challenges. Cognitive load takes three common forms. Intrinsic cognitive load is inherent to the topic at hand. Extraneous cognitive load is created by factors that require effort and attention without adding to the knowledge gained. Germane cognitive load is introduced by the learning mechanism. Load is a limiting factor: a student can only put in so much effort before a lesson is lost on them. However, a lack of cognitive load can cause a student to lose interest.

One way to create friction is to challenge what a student already knows. Doing so can create relationships with other memories while challenging students' misconceptions about the material being taught. Another is to present ideas using examples and counterexamples rather than simply stating facts. A third is to introduce a social element into lessons.

The first step in balancing friction and cognitive load is to reduce extraneous cognitive load. According to [12], the learning environment is the biggest source of extraneous load: "Changing the design of the environment can make knowledge or skill gaps disappear." Providing too much embedded information too quickly can overwhelm students, introducing extraneous load; knowledge, rather, should be progressively embedded in the environment as needed. Similarly, removing distractions from the environment will reduce extraneous load. For example, Figure 2.2 illustrates two stovetop designs. In the left design, it is unclear which dial affects which heating element. The design on the right is much clearer.

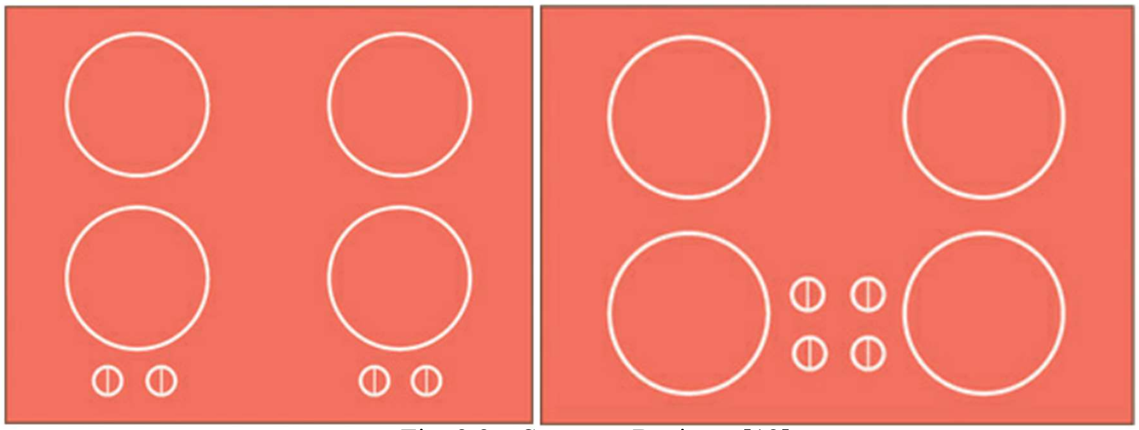


Fig. 2.2 – Stovetop Designs. [12]

The second step is to adjust the load based on the difficulty of a lesson's content. For complex content, learning mechanisms should introduce as little germane load as possible. [12] says to "give learners early wins," validating the cognitive load they endure when first encountering content. Conversely, instructors can improve retention for more straightforward or familiar content by challenging students to make their own decisions. Dirksen provides the example of recipes saying "season to taste" to illustrate directions that allow for self-driven learning.

Knowledge is reinforced and misconceptions dispelled through feedback: the evaluation of a student's work, including the method to communicate that evaluation. One popular feedback device in academia is the multiple-choice test. [12] says, "Multiple-choice tests aren't really about making the learner better at what they need to do. They are about making test administration efficient and consistent." Multiple-choice tests frequently fail to measure knowledge or skill. Other feedback and evaluation methods are more helpful and constructive for students. Providing coaching after a learning experience can help students evaluate their abilities and dispel misconceptions about the experience's content. Interactive notebooks, such as Jupyter, can provide this feedback automatically. Self-evaluation is also helpful: it allows students to reflect on what they need to improve and instructors to use these reflections to coach students.

Mechanisms for Enhancing the Instruction of Programming

Scaffolding for Coding

In [16], Xie et al. introduce an iterative approach for teaching introductory computer science. The authors argue that introductory courses need better methods of teaching the authors' four core competencies. One, Tracing Code, is the ability to understand what code does line-by-line. A second, Writing Syntax, is the ability to write code that is syntactically correct. A third, Recognizing Templates, is identifying a block of code that accomplishes a given task and understanding its operation and purpose. A fourth, Implementing Templates, is using learned code blocks and structures to accomplish a task independently. These competencies can be represented across two dimensions: skills (reading and writing code) and knowledge (language semantics and templates.) [Table 2.1]

Table 2.1 Competencies for Novice Programmer Success

	Coding Semantics	*Objective Templates*
Read	1. Trace Code	3. Recognize Templates
Write	2. Write Correct Syntax	4. Implement Templates

[12] used the Structure of Observed Learning Outcomes (SOLO) Taxonomy to develop an iterative approach to teaching these competencies. This taxonomy uses a four-level model to characterize a student's mastery of a given process: prestructural, or "no relevant knowledge"; unistructural, or an elementary understanding of a portion of that process; multistructural, or a detailed understanding of most of a process; and relational, or an ability to summarize a process and explain how it achieves its intended purpose. This taxonomy is derived from Bloom's six-level taxonomy for measuring conceptual understanding [17].

The authors argue that the ability to trace code is a prerequisite for writing code, and that an understanding of semantics is a prerequisite for understanding a code's purpose. They hypothesize that students can fail to develop a relational knowledge of content when these competencies are taught concurrently rather than in their proper sequence.

The authors suggest using an iterative approach to teaching each of the standard topics in introductory programming. For example, to teach variables the instructor should first have students read lines of code declaring, initializing, and changing the value of variables, and explain what each line does. Then, they should have students write their own code to declare, initialize, and change the value of variables. Next, they should show students templates related to variables, like a code block that uses a third, temporary variable to swap the value of two variables. Students should then be asked to describe what the code block is meant to do and identify that pattern in code examples. Finally, instructors should have students employ the templates that they have learned to complete a coding exercise. [16]

To test their theory, the authors conducted a one-day, 3-hour introduction to Python for volunteers with no programming background. Volunteers were divided into a control group that practiced only writing skills and an experimental group that practiced reading and writing. Volunteers were asked to work through one of two pamphlets: one, for the control group, with only writing exercises, and one for the experimental group, with exercises for each of the 4 core competencies. After the instruction, participants were given a 15-minute break and a "distractor task … to mitigate short-term, temporary learning gains related to taking the post-test shortly after learning the material" [16]. The participants then took a 1-hour post-test and a survey.

The study's results supported three of the authors' hypotheses: "H1: Novices who practice each skill [independently] will be able to complete more programming tasks", "H2:

Novices who practice each skill independently will make fewer errors”, and "H3: Novices who practice each skill [independently] will have a greater depth of understanding of the skills." The experimental group completed more practice problems than the control group and correctly answered more of the post-test's questions – especially questions regarding Reading and Writing Templates. The authors applied the SOLO Taxonomy to written responses and code comments from the participants' work. Their analysis (ibid., pp. 36-37) supports H3 by showing that the experimental group demonstrated a relational knowledge of the material in more cases than the control group.

Results for a fourth hypothesis, "H4: Novices who practice each skill [independently] will demonstrate more engagement in the learning process," were inconclusive largely due to the study's cognitive load. To test H4, per the authors, the study's instructional design would need to be altered.

Automated Feedback for CS Students

In [18], Fangohr et al. describe their experiences with their automatic grading system at the for novice software engineering students learning Python. The University of Southampton's novice software engineering class provides students with lab time – 90 minutes once each week – to complete the week's assigned lab work. During lab time the professor and a selection of assistants are available to help students as they work on the labs. The authors’ system was developed to reduce the time that instructors and teaching assistants spent correcting code so that they could spend more time explaining technique and new content.

Fangohr et al.’s system receives student assignments via email. When it receives an assignment, the system queues that assignment with others waiting to be graded. Once an assignment reaches the head of the queue, checks are run to ensure that the code is secure and can be processed. If an error is encountered at this step, an email describing the error is queued. If no error is encountered, the system runs a series of assignment-related tests and records a grade. A response email is constructed containing an itemized grade card for the assignment. Test cases for any incorrect problems are appended to the message, indicating which ones passed and failed, with comments on each case's purpose. This response email is queued to be returned to the student. In addition to alerting students to their errors, the system informed them of the challenges to come in the next problems and test cases.

The system places value on three quality attributes: availability, accuracy, and security. The system needs to maintain consistent uptime without error due to its mission-critical nature – it must allow students to work on and submit assignments at any time, up to an assignment's deadline. Responses need to be accurate to assure that students receive correct information and that their grades are correct. Security is maintained by limiting submissions to authorized student accounts; institutional security is maintained by the checks that are run prior to the code being processed by the system.

Before the automatic grading system was implemented, faculty estimated that they spent 90% of time checking code for correctness and 10% helping students understand concepts and build skills. After the system's implementation, students began to finish work before lab time much more frequently, implying that the system’s responses were helping students to learn without instructor intervention. As a result, lab time was spent on solidifying concepts and building skills for students.

The system assessed two types of assignments: ungraded training exercises and graded lab exercises. While each assessment could be submitted any number of times to receive additional feedback, a lab exercise's first submission was the one recorded for a grade. When they reviewed how students submited work, Fangohr et al. found that the training exercises were being repeatedly submitted and improved, in order to build a deeper understanding of content. The data also indicates that some students resubmitted lab exercises, even though doing so did not increase their grade. The data implies that the automatic grading tool can serve an auxiliary role as a teaching mechanism for improving student comprehension and skills.

Interactive Notebooks

Jupyter

Jupyter Notebooks are a tool for developing texts with an embedded environment for writing and running code. They were developed as a means for data scientists to present the stages in a dataset's acquisition, transformation, and analysis. Notebook content consists of two types of blocks. One, a Markdown block, is a plain text field that can be edited similarly to text in a word processor (e.g., Microsoft Word, Google Docs). The other, a code block, contains code; this code can be executed within the notebook, and its output will display underneath the block [19].

.NET Interactive

The default Jupyter framework limits a notebook’s examples to just one language: e.g., Python or C#, but not both. .NET Interactive, a Microsoft product, removes this restriction, allowing designers to create examples from codes in multiple programming languages that share data. For example, this could involve simulating a C# backend while a Javascript/HTML frontend displays the results [20].

Using .NET Interactive, notebooks that build multilingual examples can be configured in one of two ways. The first is to add the .NET Interactive kernel to a Jupyter environment. The second is to use Microsoft's Polyglot Notebooks extension for Visual Studio Code.

Notebooks for Teaching

In [21], Barbara and Barker discuss Jupyter as a tool for education. A survey of case studies shows that Jupyter is being used to teach a wide range of topics, including writing, music, engineering, geoscience, astronomy – and computer science. Barbara and Barker say that Jupyter helps to create a learning environment that encourages class participation and content engagement. It allows students to interact with a notebook's elements, such as code or dynamic multimedia. This allows for active experimentation – one of Kolb's 4 steps in experiential learning.

One reason for Jupyter's popularity is its designation as an open source framework: anyone can extend the source code to fit their needs. Three of the case studies that the authors cite used such extensions to adapt a notebook to the classroom. In 2013, Willing used the Music21 extension to assist in teaching music theory. In 2017, Heagley used an extension called GeoSci.xyz to assist in lectures covering applied geophysics and geophysical methods. Finally, Columbia's School of Engineering and Applied Sciences uses the matplotlib and cartopy extensions to facilitate presentations to students visiting the labs on field trips.

The authors discuss seven different modes of notebook use. The first is a textbook that provides linear material to read and interact with. A second is a workbook that students change, add to, or otherwise manipulate. A third is a drill sheet: a text that asks students to solve problems. Similar to math homework, these problems are typically repetitive and increase in difficulty. A fourth is a guided note sheet: linear material that students follow during a lecture. A fifth is an application: an interactive medium with (e.g.) buttons and sliders for manipulating displayed content. A sixth is a lab report: a student-created notebook that synthesizes written responses and code. The last is a presentation replacement: material used to augment a lecture.

Notebooks can function as static media or augmented with live coding demonstrations. Instructors can mix and match these modes of use – for example, an instructor may use a "guided note sheet" notebook as a "presentation replacement", having students follow along as they solve problems that they have laid out.

In [22], Pfeiffer et al. present a textbook in the format of a Jupyter Notebook for an upper division elective class aimed to teach Python 3 to programmers with prior experience in object-oriented programming. The textbook contains two sections to cover the scope of the language. The first (chapters 2-19) covers Python's core features; the second (chapters 19-29) covers a few of the Python's more popular extensions and libraries.

Each chapter is divided into sections that cover the core concepts of the topic at hand. Within those sections, the text begins with an explanation of the concept, followed by an example in code relevant to that concept. Students can execute the code example to examine the result; more than that, they can edit the provided example to see how their changes affect the result. In some cases, the sections provide exercises for students to attempt. The proximity between these examples and exercises to the textbook content means that it is easier to reference the text and understand what is being demonstrated or required for an exercise.

CHAPTER 3. METHODOLOGY

This study aims to evaluate the impact of an interactive notebook on student performance in a lower-division computer science course. This chapter's first section describes construction of an interactive notebook for supplementing a course on data structures, a standard lower-division computer science course. It explains and states the rationale for the selection of materials, topic sequencing, and the design and development of the notebook itself. The following section describes the educational intervention study. It explains and states the rationale for the study's research design, sampling procedure, and data collection and analysis methods.

Notebook Construction

A Tour of Data Structures in C# is a Polyglot Notebook whose use of experiential learning theory principles was intended to enhance students' learning experience. ETSU's Data Structures class (CSCI 2210) was selected as the study's setting due to the convenience of its location, a recent unexpected change of instructor, and historical low student performance and satisfaction. The notebook follows the instructional design patterns presented by Xie et al. with relevant examples and exercises placed close to the content. This manner of composition provides opportunities for concrete learning, reflective observation, and active experimentation [8].

Topic Sequencing

The notebook contains 5 chapters, each building on concepts presented in the previous chapter. Each of the notebook's final four chapters reinforces concepts through the use of questions that relate new concepts to concepts introduced in previous chapters – e.g., Q3 from 5.3. Additionally, each of the notebook's first four chapters asks questions that foreshadow material presented in future chapters: e.g., Q1 from 4.1 (see Appendix for questions). Sequencing

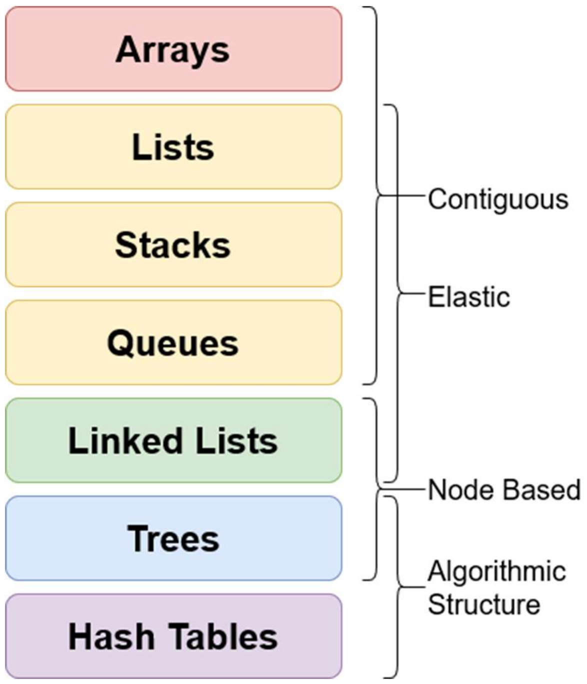


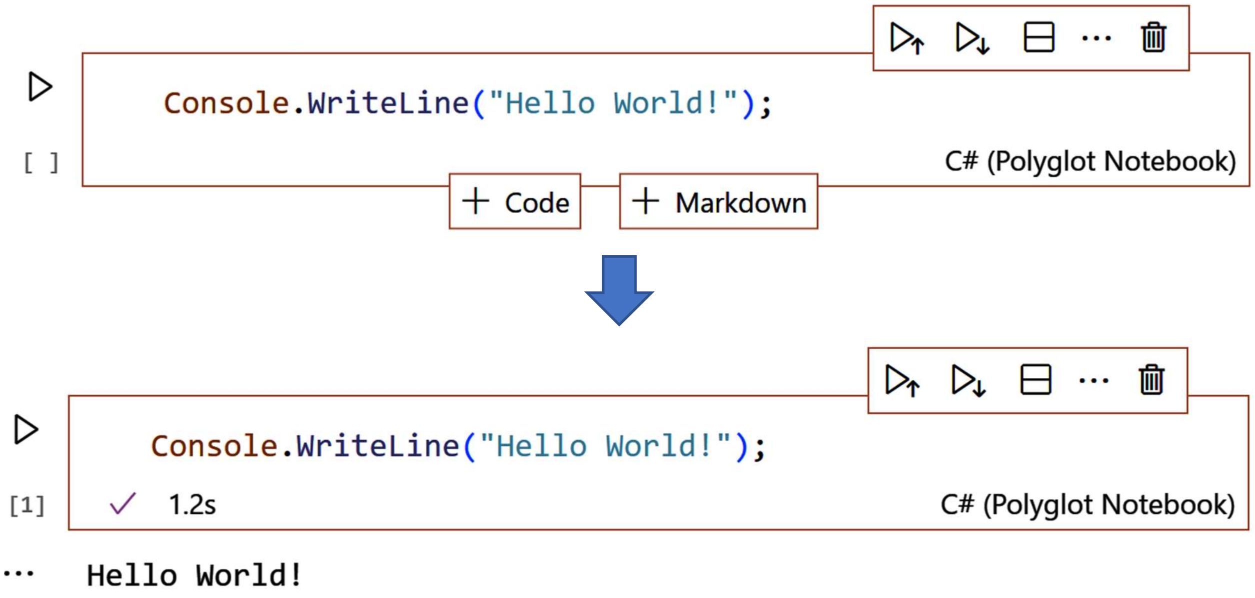
Fig. 3.1 – List of topics in sequence as they appear in the notebook. Chapters are separated by color.

in this manner promotes long-term retention by relating content to students' prior knowledge while avoiding jumping from topic to topic – a potential source of extraneous cognitive load [12].

The chapters are arranged based on four attributes that neighboring chapters share: *contiguous* memory location, *elastic* size, *node-based* construction, and *algorithmic* construction (Figure 3.1). *Contiguous* data structures store elements in a linear sequence in memory. Elements are accessed based on their offset from the structure's first element. *Elastic* structures can change in size as elements are added and removed. *Node-based* structures store data in isolated memory locations. Each node can store references to other nodes, thereby linking them into a structure. Finally, the placement of elements in an *algorithmic structure* is determined by an underlying algorithm (e.g., a hash function or sorted order).

Content Creation

The notebook features two types of content. One, traditional textbook-style content, is composed with Markdown Language and HTML, allowing for rich text content and static multimedia (such as images and video). The textbook content describes a concept and is followed by a code example that demonstrates it.



```
Console.WriteLine("Hello World!");
```

```
Console.WriteLine("Hello World!");
```

```
Hello World!
```

Fig. 3.2 – Polyglot Notebooks for Visual Studio Code – before and after running a code cell.

The other, executable code, consists of examples that are placed near textbook content. This content allows for immediate active experimentation with the concepts presented in the adjacent text, encouraging learners to engage in the Experiential Learning cycle modeled by Kolb. Learners can then reflect and observe with reference material close at hand [8].

Exercise Scaffolding

Each chapter of the notebook contains four sections. Each section ends with exercises that—as suggested by Xie et al.—are presented in increasing order of difficulty. The first section in each chapter asks learners to read a line of code and explain it. This is followed by second that asks learners to write lines of code to do tasks, a third that asks learners to explain blocks of code, and a fourth that asks learners to apply the chapter's concepts to a new problem, incorporating new elements.

Study Methodology

This research was undertaken to determine whether the use of an interactive notebook could improve student performance in a computing class and to identify those aspects of the notebook that contribute to any observed changes. The study's results were intended to provide insight into the notebook's effectiveness and to inform future research and practice in educational technology. The following section outlines the design and methodology of the educational intervention used to measure the interactive notebook's efficacy.

Study Design

This study implemented a quasi-experimental design with non-equivalent groups. Student participants were split into a treatment and control group. The treatment group could access the interactive notebook, and the control could not. This design was chosen to provide a means for inferring the intervention's effects on student performance while controlling for other factors.

The initial sample population was gathered from all students enrolled in CSCI 2210 – Data Structures. Each student received an invitation to participate in the study via email, a Desire2Learn (D2L, ETSU's online LMS) news post, and an in-class announcement. This initial sampling was based on the recommendation and support of department leadership. This is called purposeful sampling, where participants are selected based on ease of access rather than using a random or stratified sampling technique. Despite its limitations, this method can provide meaningful results if the sample is representative of the target population's demographic and background characteristics.

Once a student opted in for the study, either by responding to the solicitation email or by emailing the primary investigator (PI) directly, the PI shared the informed consent (IC) document electronically. The IC form was facilitated via Microsoft Forms and required authentication via ETSU authentication protocols. Access to form responses was limited to the PI and the advisor.

After the 1.5-week opt-in period, the PI used Python's *random* library to load the list of participant names and randomly distribute them into 2 groups of 7 students each. The PI then shared the group enrollment list with the course instructor, who created three groups using the D2L groups tool: "control", "treatment", and "unassigned". The "control" and "treatment" groups were assigned according to the Python script's distribution; students on the class list who did not elect to participate in the study were placed in the "unassigned" group. The PI was granted access to all three groups in order to disseminate the components and information related to the interactive notebook. The PI did not have access to any student grades or performance indicators during the academic semester. Group-level student demographic and biographical data was not collected per IRB restrictions.

Distribution and Support of ILE

Using the D2L content tool and built-in system release conditions, the PI released chapters of the interactive notebook to the study's participants in the "treatment" group incrementally and notified them via a D2L news post. Students in the "treatment" group were encouraged, but not required, to use or leverage the interactive notebook. The interactions between the PI and the treatment group were limited to answering questions related to notebook functionality and/or compatibility. The PI did not communicate with the "control" group or the "unassigned" group.

Data Collection

At the conclusion of the semester, the PI distributed a survey to the “treatment” group via the Brightspace LMS. The survey contained a single prompt: “*How many hours did you spend using the [interactive notebook] to study this semester? Round to the nearest hour.”* Also, the course instructor provided anonymized assessment results to the PI via an encrypted spreadsheet.

Data Analysis

The analysis of the data collected for this study involved the use of an independent t-test and a Pearson correlation coefficient. The independent t-test was used to determine if access and use of an interactive learning environment had a significant impact on the grades of students enrolled in the study. The Pearson correlation coefficient was used to examine the relationship between notebook usage and final grade scores among students provided with an interactive learning environment.

CHAPTER 4. ANALYSIS

Independent T-Test

Q1 examines whether there was a significant difference in learning outcomes between students who were provided with the notebook and those who were not. To determine if access to and use of the notebook had a significant impact on students' grades, the researcher conducted an independent t-test. The fourteen students were divided into a 7-person control group, which did not receive the notebook, and a 7-person treatment group, which did. The results showed no statistically significant difference between the mean project and assignment grades of the control group (n=7, M=86.82, SD=8.61) and the treatment group (n=7, M=74.86, SD=19.97), t(11)=1.455, p=.171. However, the effect size was medium (.778), indicating a potential practical significance. The results failed to reject the null hypothesis, suggesting that the availability of an interactive notebook did not have a significant impact on students' grades in this particular course.

Table 4.1 Group Statistics

	grp	N	Mean	Std. Deviation	Std. Error Mean
Final Grade	c	7	86.816	8.607	3.253
	t	7	74.857	19.969	7.547
proj1	c	7	88.290	7.566	2.860
	t	7	84.000	13.916	5.260
proj2	c	7	93.860	6.256	2.365
	t	7	77.140	38.499	14.551
proj3	c	7	90.710	8.864	3.350
	t	7	90.710	12.724	4.809
proj4	c	7	85.000	10.408	3.934
	t	7	59.290	40.252	15.214
proj5	c	7	94.140	5.178	1.957
	t	7	77.140	34.017	12.857
proj6	c	7	58.570	42.692	16.136
	t	7	65.000	35.237	13.318
proj7	c	7	97.140	5.669	2.143
	t	7	70.710	36.105	13.646
Midterm Exam	c	7	93.860	5.843	2.209
	t	7	94.570	10.518	3.975
Final Exam	c	7	82.710	8.845	3.343
	t	7	79.570	12.205	4.613

Table 4.2 Independent Samples Test

	Equal variances assumed?	Levene's Test		t-test for Equality of Means			t-test for Equality of Means			
		F	Sig.	t	df	Sig. 2-Sided p	Mean Dif.	Std. Err. Dif.	95% Confidence Lower	Upper
Final Grade	Yes	2.333	0.153	1.455	12.000	0.171	11.959	8.219	-5.948	29.866
	No			1.455	8.155	0.183	11.959	8.219	-6.931	30.849
proj1	Yes	6.178	0.029	0.716	12.000	0.488	4.286	5.987	-8.759	17.330
	No			0.716	9.262	0.492	4.286	5.987	-9.200	17.771
proj2	Yes	9.703	0.009	1.134	12.000	0.279	16.714	14.742	-15.406	48.834
	No			1.134	6.317	0.298	16.714	14.742	-18.924	52.353
proj3	Yes	1.122	0.310	0.000	12.000	1.000	0.000	5.861	-12.770	12.770
	No			0.000	10.713	1.000	0.000	5.861	-12.943	12.943
proj4	Yes	9.127	0.011	1.636	12.000	0.128	25.714	15.714	-8.524	59.953
	No			1.636	6.799	0.147	25.714	15.714	-11.668	63.097
proj5	Yes	3.547	0.084	1.307	12.000	0.216	17.000	13.005	-11.336	45.336
	No			1.307	6.278	0.237	17.000	13.005	-14.484	48.484
proj6	Yes	0.771	0.397	-0.307	12.000	0.764	-6.429	20.923	-52.015	39.158
	No			-0.307	11.584	0.764	-6.429	20.923	-52.197	39.340
proj7	Yes	6.823	0.023	1.913	12.000	0.080	26.429	13.814	-3.669	56.526
	No			1.913	6.296	0.102	26.429	13.814	-6.991	59.848
Midterm Exam	Yes	1.661	0.222	-0.157	12.000	0.878	-0.714	4.548	-10.623	9.194
	No			-0.157	9.382	0.879	-0.714	4.548	-10.938	9.510
Final Exam	Yes	1.745	0.211	0.552	12.000	0.591	3.143	5.697	-9.270	15.556
	No			0.552	10.940	0.592	3.143	5.697	-9.405	15.690

Correlation Test

Q2 investigates whether interaction time positively correlated with learning outcomes among students provided with the notebook. To determine whether this correlation held, the researcher computed a Pearson correlation coefficient to examine the relationship between interactive notebook usage and the final grade of the treatment group. The analysis revealed a strong positive relationship between notebook usage (M = 2.34, SD = 2.99) and final grade (M = 74.85, SD = 19.97) scores, but the correlation was not statistically significant (N = 7, r = .625, p = .134). Thus, the results failed to reject the null hypothesis: there is no statistically significant difference in students' grades based on interactive notebook usage. These findings suggest that students with high notebook usage could also have higher final grades, but the correlation requires a larger sample size for an increase in significance.

Table 4.3 Treatment Group Statistics

	Mean	Std. Deviation	N
Final Grade	74.857	19.969	7
Study Time	2.430	2.992	7

Table 4.4 Correlations

		Final Grade	Study Time
Final Grade	Pearson Correlation	1	.625
	Sig. (2-tailed)		.134
	N	7	7
Study Time	Pearson Correlation	.625	1
	Sig. (2-tailed)	.134	
	N	7	7

CHAPTER 5. CONCLUSION

The results of the independent t-test and Pearson correlation coefficient tests failed to show any statistically significant difference in learning outcomes between students provided with the notebook and those who were not. While the effect size for the t-test was medium, the lack of significance suggests that the notebook's availability did not have a significant impact on students' grades in this particular data structures course.

This study was limited by several factors. Firstly, the study was conducted in a single section of the class, which may affect its external validity. Secondly, the control group by chance consisted of high-performing students, which may have influenced the results – particularly those of the t-test. Finally, the small sample size limits the study's power to detect differences between the groups.

One interesting finding was the positive relationship between notebook usage and final grade scores, as shown by the Pearson correlation coefficient. While the correlation was not statistically significant, it suggests that students with high notebook usage could also have higher final grades. However, this relationship needs to be further investigated with a larger sample size to increase the statistical power and determine the significance of the correlation.

Future Work

Future work related to the use of notebooks in classroom settings should include improvements to the educational intervention and changes to the study. By enhancing the intervention and the notebook, it may be possible to build upon the foundation laid by this study and further support student learning outcomes. Alternatively, changing the study design could address some of the current study's limitations and shortcomings.

One feature of Polyglot Notebooks is the ability to simultaneously use an arbitrary number of languages that share state. This feature can allow for more interactive and engaging examples, such as visualizations that are easier to create or data structures that are simpler to understand in one language versus another. It would also provide an environment for side-by-side comparison between languages. Overall, incorporating multiple languages into the notebook could provide a richer and more diverse learning experience.

In a future study, it would be beneficial to more closely integrate the interactive notebook into the course curriculum for the treatment group. During the current study, the notebook was an optional tool for students to use; some students may not have taken full advantage of its features. Incorporating the notebook more thoroughly into the course may increase students' exposure to the intervention. Additionally, a more integrated approach could allow for better tracking of student progress and engagement with the tool, which could provide more comprehensive data on the notebook's impact on student learning outcomes. Further investigation is needed to determine the most effective way to integrate the notebook into the course and to explore any potential barriers to adoption by both students and instructors.

Finally, future work could address the study's limitations. Surveying two sections instead of one would likely result in a larger sample size and a control group with a more controlled population. A larger sample size would allow for more sophisticated statistical analysis and could potentially reveal significant effects that were not detected in this study.

REFERENCES

[1] Simon *et al.*, “Pass Rates in Introductory Programming and in other STEM Disciplines,” *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education*, Dec. 2019, doi: https://doi.org/10.1145/3344429.3372502.

[2] J. McFarland *et al.*, “The Condition of Education 2018,” May 2018 [Online]. Available: https://nces.ed.gov/pubs2018/2018144.pdf.

[3] C. Stephenson *et al.*, “Retention in Computer Science Undergraduate Programs in the U.S. Data Challenges and Promising Interventions,” 2018 [Online]. Available: https://www.acm.org/binaries/content/assets/education/retention-in-cs-undergrad-programs-in-the-us.pdf.

[4] ACM/IEEE-CS Joint Task Force on Computing Curricula. "Computing Curricula 2020: Paradigms for Global Computing Education." Association for Computing Machinery and IEEE Computer Society, 2020.

[5] J. Kubica, *Data Structures the Fun Way*. No Starch Press, 2022.

[6] D. Zingaro, *Algorithmic Thinking*. No Starch Press, 2021.

[7] P. LeBlanc, “Reading Signals from the Future: EDUCAUSE in 2038,” Jul. 02, 2018. [Online]. Available: https://er.educause.edu/articles/2018/7/reading-signals-from-the-future-educause-in-2038. [Accessed: Mar. 29, 2023]

[8] D. A. Kolb, *Experiential Learning: Experience as the Source of Learning and Development*, 2nd ed. Upper Saddle River, New Jersey: Pearson Education, Inc, 2014.

[9] N. Wexler, “What Teachers Need To Know About The Science Of Learning--And What They Don’t,” *Forbes*, Feb. 19, 2019. [Online]. Available: https://www.forbes.com/sites/nataliewexler/2019/02/19/what-teachers-need-to-know-about-the-science-of-learning-and-what-they-dont/?sh=151467ba6582#372154456582. [Accessed: Mar. 29, 2023]

[10] J. Wong *et al.*, “Educational Theories and Learning Analytics: From Data to Knowledge,” in *Utilizing Learning Analytics to Support Study Success*, D. Ifenthaler, D. Mah, and J. Yau, Eds. Cham: Springer, 2019.

[11] J. Dewey, *Experience and Education*. New York: Free Press, 1938.

[12] J. Dirksen, *Design for how people learn*, 2nd ed. Berkeley, Calif.: New Riders, 2016.

[13] J. McManus and P. Costello, “Project based learning in computer science: a student and research advisor’s perspective,” *Journal of Computing Sciences in Colleges*, vol. 34, no. 3, Jan. 2019.

[14] S. Weinschenk, 100 Things Every Designer Needs to Know About People, 2nd ed. Peachpit Press, 2020.

[15] Kahneman, D. (2011). Thinking, fast and slow. New York, NY: Farrar, Straus and Giroux.

[16] B. Xie *et al.*, “A theory of instruction for introductory programming skills,” *Computer Science Education*, vol. 29, no. 2–3, pp. 205–253, Jan. 2019, doi: https://doi.org/10.1080/08993408.2019.1565235. [Online]. Available: https://faculty.washington.edu/ajko/papers/Xie2019IntroCSTheoryOfInstruction.pdf. [Accessed: Nov. 14, 2021]

[17] B. S. Bloom, M. D. Engelhart, E. J. Furst, W. H. Hill, and D. R. Krathwohl, Taxonomy of Educational Objectives: The Classification of Educational Goals. Handbook I: Cognitive Domain. Longman, 1956.

[18] H. Fangohr, N. O'Brien, A. Prabhakar and A. Kashyap, "Teaching Python Programming with Automatic Assessment and Feedback Provision," University of Southampton, 11 September 2015. [Online]. Available: https://arxiv.org/pdf/1509.03556.pdf. [Accessed Nov. 14, 2021].

[19] Project Jupyter. "Project Jupyter." [Online]. Available: https://jupyter.org/. [Accessed: Apr. 2, 2023].

[20] Microsoft. “.NET Interactive.” [Online]. Available: https://github.com/dotnet/interactive [Accessed: Apr. 2, 2023].

[21] L. A. Barba and L. J. Barker, "Teaching and Learning with Jupyter," 06 12 2019. Available: https://jupyter4edu.github.io/jupyter-edu-book/index.html. [Accessed Dec. 19, 2022].

[22] P. Pfeiffer, Z. Bunch and F. Oyeniyi, "A Tour of Python 3," East Tennessee State University, 2021.

APPENDIX: QUESTION LIBRARY

Chapter 1.1

1. What does the following code do?

   ```
   int[] monthlySales = new int[];
   ```

2. Given the array `inchesOfSnow = [7, 2, 0, 0, 1, 0, 5]`, what does the following code do?

   ```
   Console.WriteLine(inchesOfSnow[5]);
   ```

3. Given the above array, what does the following code do?

   ```
   inchesOfSnow[4] = 2;
   ```

4. What does the following code do?

   ```
   bool[] questionsCorrect = [true, false, false, true, true];
   ```

5. What does the following code do?

   ```
   bool[] questionsCorrect = {true, false, false, true, true};
   ```

6. Give some examples of builtin `Array` methods in C#. What do they do?

Chapter 1.2

1. Create an array and fill it with the first 7 numbers of the Fibonacci Sequence, starting with `0`.
2. Using the array created in exercise 1, print the value `3` to the screen.
3. Reverse the above array, and then subtract 5 from each value.
4. Create a multidimensional array to represent a Tic-Tac-Toe board.
5. Initialize the values in the array to spaces. If you wish, you can combine 4 and 5 into one step.
6. Place a few 'X's and 'O's on the board, and then print the array.
7. Create a jagged array with 6 rows.

8. Using a fresh copy of the array from exercise 1, set the sizes of the jagged array's rows to the Fibonacci Sequence: for example, this array's first and last rows should have 1 and 8 elements, respectively. Fill each row to capacity with the initial values of the Fibonacci sequence: for example, this array's third row should have {1, 1, 2} as its elements, while its fifth should have {1, 1, 2, 3, 5}. Finally, print the array.

Chapter 1.3

1. Explain what the following code accomplishes.

```
foreach (int x in array) {
    Console.Write($"{x} ");
   }
```

2. Explain what the following code accomplishes.

```
for(int i = 0; i < array.Length; i++) {
    array[i] *= 2;
   }
```

3. Explain what the following code accomplishes.

```
int[][] arr = new int[4][];
int[] lengths = {4, 2, 5, 7};

for(int i = 0; i < arr.Length; i++) {
    arr[i] = new int[lengths[i]];
}
```

4. Explain what the following code accomplishes.

```
using(var reader = new StreamReader(@"./file.txt")) {
    for (int i = 0; i < array.Length; i++) {
        var line = reader.ReadLine();
        array[i] = int.Parse(line);
    }
}
```

5. Explain what the following code accomplishes.

```
int index = -1;

for(int i = 0; i < array.Length; i++) {
    if(array[i] == 3) {
        index = i;
        break;
    }
}
```

6. Explain what the following code accomplishes.

```
for (int i = 0; i < array.Length - 1; i++) {
    for (int j = 0; j < array.Length - i - 1; j++) {
        if (array[j] > array[j + 1]) {
            int temp = array[j];
            array[j] = array[j + 1];
            array[j + 1] = temp;
        }
    }
 }
```

Chapter 2.1

1. What does the following code do?

   `List<char> letters = new List<char>();`

2. Use the following list for the exercises below: `list = [9, 3, 5, 1, 2, 4, 7, 6, 8]`

   What does the list look like after the following code executes?

   `list.Add(27);`

3. What does the list look like after the following code executes?

   `list.Insert(3, 99);`

4. What does the list look like after the following code executes?

   `list.Remove(3);`

5. What does the list look like after the following code executes?

   `list.Remove(10);`

6. What does the list look like after the following code executes?

   `list.RemoveAt(2);`

7. What does the list look like after the following code executes?

   `list.RemoveAt(14);`

8. Give some examples of builtin `List` methods in C#. What does each of them do?

Chapter 2.2

1. Create a queue and fill it with the first 5 digits of pi, excluding the decimal point.

2. Print the value `4` to the screen using the queue from exercise 1.
   💡 **Hint:** There are multiple ways to solve this problem. Try a few different methods.

3. Create a stack from the given array `string[] fruits = {"apple",`

   `"banana", "cherry", "durian", "elderberry", "fig"}.`

4. Use a loop to print all of a stack's contents.

5. What happens when you `Peek` the stack? What about when you use `TryPeek`?

   Demonstrate this in the code blocks, and then record your answer in a markdown cell.

Chapter 2.3

1. List some other potential uses for stacks and queues.

2. Explain what the following code accomplishes.

```
foreach (int x in list) {
Console.Write($"{x} ");
}
```

3. Explain what the following code accomplishes.

```
string s = "";
while (stack.TryPop(out s)) {
Console.WriteLine(s);
}
```

4. Explain what the following code accomplishes.

```
string s = "";
char nextLetter;
while (stack.TryPop(out nextLetter)) {
```

```
s += nextLetter;
}
```

5. Explain what the following code accomplishes.

```
string s = "";
if (queue.TryDequeue(out s)) {
Console.WriteLine(s);
}
else {
Console.WriteLine("Queue is empty.");
}
```

6. What is (are) the difference(s) between using `foreach` with a list to process data and using `Pop` in a stack or `Dequeue` in a queue?

Chapter 3.1

1. What is a reference type variable?
2. What is held in `Node.data`?
3. What is held in `Node.next`?
4. What does the following loop accomplish?

```
while (currentNode != null) {
    //...
}
```

Chapter 3.2

1. Print the contents of the list in the cell above.
2. Print the value "101" from the list.
3. Swap the head node with the 3rd node in the list (`data = 12`).
4. Add a new node with `data = 88` ***after*** the second node (`data = 71`)

Chapter 3.3

1. In a *doubly-linked list*, why is it possible to remove the last element in `O(1)` time?
2. Explain the purpose of the following loop.

```
do {
    //...
     } while (currentNode != circularLinkedList.head);
```

3. Explain what the following function accomplishes.

```
function() {
    if(this.head == null) {
        return;
    }
    else if(this.head.data == data) {
        this.head = this.head.next;
    }
    else {
        Node current = this.head;
        while(current.next != null) {
            if(current.next.data == data) {
                current.next = current.next.next;
                return;
            }
            current = current.next;
        }
    }
}
```

4. What is the time complexity of the function from exercise 3? Why?
5. Designers of data structures routinely tinker with the details of their implementations in order to optimize the behavior of key operations.

   One such variant of a linked list places a dummy element at the head of every linked list. This dummy element contains no data - rather, it anchors the rest of the list. One drawback of this approach is that it "wastes" space: the dummy element will never contain meaningful content. What, then, are this implementation's advantages?

   💡 **Hint**: how do `add`, `remove`, and `find` handle empty lists?

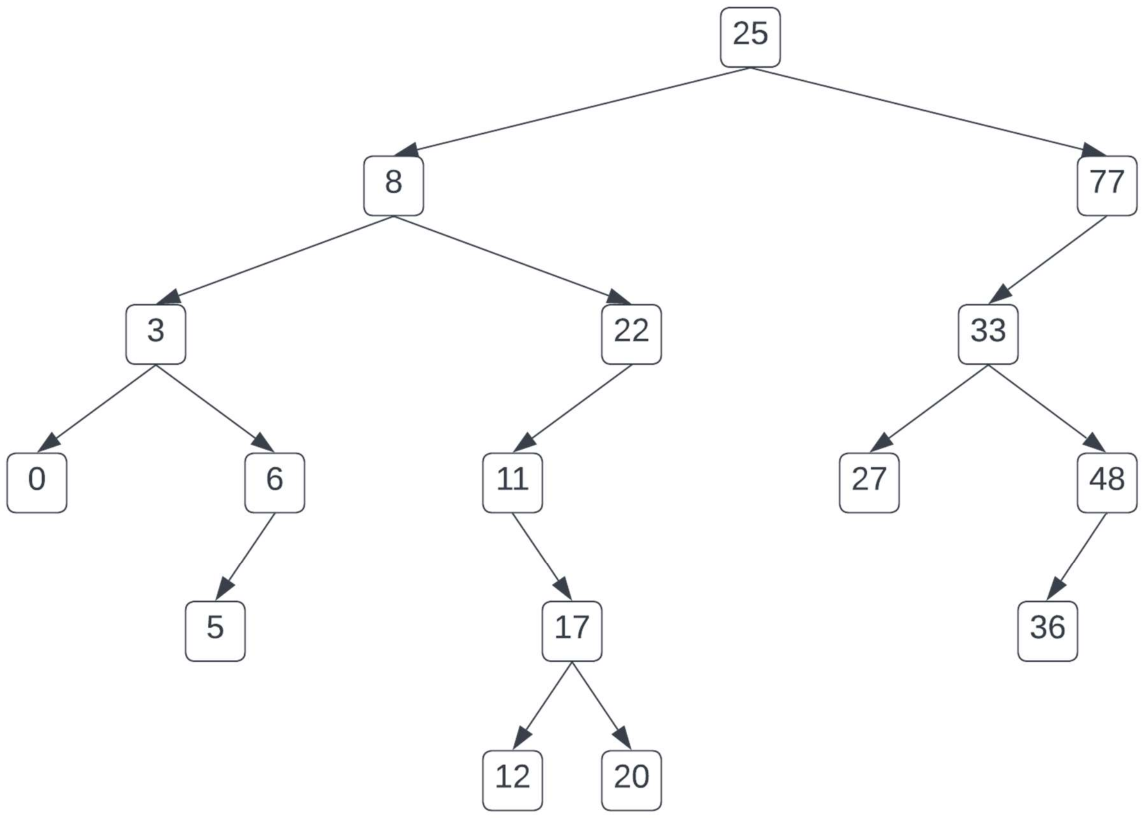
Chapter 4.1

1. What qualities of a tree make them similar to liked lists? What qualities make them different?

2. What is the `root` node?
3. What are parents and children?
4. In the diagram above, what is the depth of 9? of 0? of 4?
5. In the diagram above, what is the height of the tree?
6. What is traversal? How does *breadth-first* traversal differ from *depth-first* traversal?

Chapter 4.2

1. Using the `BinarySearchTree` class, create the tree in the following diagram:



2. In what order can the nodes be added to create a balanced tree? Write the code below.

3. How does order affect accessing nodes in a tree? What about balance? Write your answer in a new markdown cell.

💡 **Challenge:** In the following `code` cells, create methods to perform reverse traversals for pre-order, in-order, post-order, and level order traversals.

Chapter 4.3

1. What does the following loop accomplish? Why does the first if statement return null?

```
while (currentNode.data != data) {
    if (currentNode.left == null && currentNode.right == null) {
        return null;
    }

    if (data < currentNode.data) {
        currentNode = currentNode.left;
    }
    else {
        currentNode = currentNode.right;
    }
}
return curentNode;
```

2. What does the following function accomplish? What process does it aid in?

```
public Node function(Node x) {
    Node current = x.right;
    while (current.left != null) {
        current = current.left;
    }
    return current;
}
```

3. Describe a left rotation. When would a left rotation result in a balanced tree?

4. Describe a right-left rotation. In what case would you this instead of a left rotation?

Chapter 5.1

1. Given `array = {17.5, 16.7, 0.0, 8.4, 99.3, 66.7, 59.8}`, what value does `array[2]` access?

2. Given the same array, what value does `array["Tuesday"]` access?

3. Given:

   ```
   dictionary = {
     Sunday: 17.5,
     Monday: 16.7,
     Tuesday: 0.0,
     Wednesday: 8.4,
     Thursday: 99.3,
     Friday: 66.7,
     Saturday: 59.8
   }
   ```

   what value does `dictionary[2]` access?

4. Given the same dictionary, what value does `dictionary["Tuesday"]` access?
5. Given the same dictionary, what value does `dictionary["February"]` access?
6. What can you infer about the data being stored in these structures? How does that inference change from one to the other, if at all?

Chapter 5.2

1. Write the code to access the grade of Rachael, given the following dictionary:

```
Dictionary<string, int> dict = new Dictionary<string, int>() {
{"Andrea", 97},
{"Samuel", 26},
{"Gerald", 85},
{"Tomas", 85},
{"William", 92},
{"Xander", 91},
{"Rachael", 89},
{"Collin", 85},
{"Brutus", 74},
{"Caesar", 90}
};
```

2. Write the code to add a grade of 96 to a student called Luis.
3. Write the code to increase Luis's grade to a 100.
4. What is the worst-case lookup time for a hash table implemented with chaining? What about probing? Give your reason for both.
5. Consider a hash table that implements linear probing where n=3. What problems might occur when the capacity (total number of buckets) is 30, if any?

Chapter 5.3

1. What is the purpose of the following code?

```
public int this[string key] {
    set {
        //...
    }
    get {
        //...
    }
}
```

2. Explain how function caching would be helpful for a function that returns the nth item of the Fibonacci sequence given n.
3. The AssociativeArray class in 5.1 uses a list. Could its performance be improved by using a tree instead? And, if so, what requirements would need to be imposed on the key-value pairs?
4. Chaining also uses lists. Given that tree accesses are O(log n), which underlying structure will perform better on average? Why? In what cases does this vary, if any?

VITA

DILLON BUCHANAN

Education:	M.S. Computer Science, East Tennessee State University, Johnson City, Tennessee, 2023
	B.S. Computer Science, East Tennessee State University, Johnson City, Tennessee, 2021
Professional Experience:	Graduate Teaching Associate, East Tennessee State University, College of Business and Technology, 2023
	Graduate Research Assistant, East Tennessee State University, Clemmer College of Education, 2022
	Information Systems and Technology Specialist, Trailblazer Coffee, 2020-2021
Publications:	Buchanan, Dillon and Pottinger, Benjamin (2023). “Character Analysis from Text Stories Using Sentiment Analysis and Unsupervised Clustering” Appalachian Regional Business Symposium. Under Review.
Honors and Awards:	2023 Outstanding Graduate Student, Department of Computing