PhD Program in Computer Science and Systems Engineering
DIBRIS, Università degli Studi di Genova

Ph.D. Thesis in Computer Science

Innovative Teaching Methodologies for University Courses in Computer Science

Candidate: Daniele Traversaro

Submitted: March 2024

Advisors:
Prof. Giorgio Delzanno (Università degli Studi di Genova)
Prof. Giovanna Guerrini (Università degli Studi di Genova)

External Reviewers:
Prof. Michael Lodi (Università di Bologna)
Prof. Claudio Mirolo (Università degli Studi di Udine)

Abstract

This thesis explores the under-researched area of introductory concurrent programming and data systems education. The lack of pedagogy poses a significant challenge to educators. While these topics are often perceived as complex and require a strong foundation in programming, this thesis presents innovative teaching units based on constructionist learning theory, specifically designed for students with no prior knowledge. It proposes a series of interventions based on domain-specific, real-world and collaborative learning approaches aimed at undergraduate computer science students. In particular, an introduction to concurrency is presented using Sonic Pi, a musical domain-specific language, to intuitively introduce concurrency concepts by exploiting the natural connection between multi-threading and live music coding, through code comprehension and code composition tasks designed to address and explore common concurrency misconceptions. A collaborative learning approach, namely "think-pair-share", is advocated to promote engagement and deeper understanding of logical database design. In addition, an educational Python package based on the Pandas library is presented to simplify data-centric computing pedagogy by providing a less complex notional machine while retaining familiar Pandas-like syntax. Finally, a variety of teaching experiences and proposals are provided to inspire and guide educators. Through these units, the thesis provides suggestions and recommendations for teachers and researchers, offering experiences and formats to effectively equip students with essential skills in introductory concurrent programming and introductory data systems. By demonstrating innovative pedagogical approaches, this thesis contributes to bridging the gap in this critical area of computer science education.

Keywords: Concurrency, concurrent programming education, data education, pedagogy, computer science education, Sonic Pi, collaborative learning, data science, Python, data-centric programming.

Table of Contents

Part I

Introduction and Background

Chapter 1

Motivation and Research Goals

Concurrent programming and data systems are two fundamental areas of computer science (CS). They are essential to the development of modern software systems, which often need to be able to handle multiple concurrent tasks and access and manage shared data.

Typically, concurrency is perceived as a complex topic that requires a strong programming foundation [137]. This often leads to its exclusion from introductory courses at both university and school level [154], even though many computational thinking tools implicitly use and implement concurrent behaviour [167].

Data education is fundamental to a wide range of programmes such as CS, Data Science (DS), Information Systems and beyound. Since the 1970s, the database research community has been actively debating the appropriate placement and best practices for teaching data systems concepts in curricula. This crucial discussion has gained particular momentum in recent years due to the emergence of DS, a relatively young discipline potentially facing a shortage of qualified teachers at school level.

Within the ACM CS2023 curriculum [96], the field of data systems is referred to as "Data Management" (DM) and is recognised as a fundamental area of knowledge within CS[1]. It encompasses not only relational database systems, but also the broader role of data within an organisation. This includes, but is not limited to, the data lifecycle, the social and legal considerations surrounding data collection, and the use of advanced technologies for big data analytics such as NoSQL databases, cloud-based databases, MapReduce, and similar frameworks. This thesis deliberately uses the term "Data Science education" to reflect its broader scope compared to traditional data systems education. While data systems education emphasises the infrastructure and management of data, DS education encompasses a broader range of topics, including programming languages for DS, data visualisation and data storytelling.

[1] ACM CS2023 curriculum

Despite the critical importance of data in the modern digital world, the field of introductory data education remains relatively under-researched. The same applies to concurrency, in particular related to multi-core programming. This pedagogical deficiency poses a significant challenge to educators, hampering their ability to effectively teach these concepts to students.

A recent project by the Italian national working group developing the new 60-credit qualification course for CS teachers in secondary schools (teacher class A-41, "Percorso Formativo Abilitante da 60 CFU") illustrates this need. They produced a document entitled "Syllabus Proposal for Italian Teacher Training for Class A041"[2], which outlines the planned curriculum for the Class A41 teacher training programme in Italy. This initiative highlighted a significant gap in pedagogical resources and tools specifically designed for teaching data systems and concurrency. This is in stark contrast to the abundance of readily available materials and extensive pedagogical research dedicated to introductory programming and computational thinking.

Motivated by the identified gap, this dissertation addresses under-researched areas of CS education at the introductory level, with a particular focus on concurrent programming and data systems. We propose novel innovative approaches to teaching these topics.

Our research has utilised the resources available to us, including both financial and human capital. This included the Innovative Teaching Project (detailed in section Teaching Innovation Project) and the participation of first year students enrolled in our undergraduate CS programme. By focusing primarily on first-year students (with the exception of the experiment within the Introduction to Databases course, which involved second-year students with no prior experience of data modelling), we have attempted to address the needs of teaching at the high school level as well. It is important to recognise that further research is needed to develop fully validated materials specifically tailored for high school students, and possibly integrated into a dedicated programming and data systems education course for school teachers. While a comprehensive solution remains elusive, this research serves as a stepping stone to addressing the identified educational gap. In particular, we raised the following questions:

- How can we introduce the concept of concurrency to students with minimal programming experience, in a way that is both engaging and promotes a deep understanding of its benefits and challenges?
- How can active and collaborative teaching methods be used to enhance data education to improve learning outcomes and student engagement?
- How can we foster data science thinking to students?

Recognising the breadth of data education, which encompasses various data-related fields, this dissertation focuses specifically on database education and introductory DS education in the

[2] Syllabus Proposal for Italian Teacher Training for Class A041

areas of data modelling, programming, data visualisation and data analysis. Additionally, it explores introductory concurrent programming, with a particular emphasis on addressing common students misconceptions. The thesis is expected to make a significant contribution to the field of CS and data education by providing valuable suggestions, experiences and innovative teaching formats to help students acquire these essential skills effectively. To achieve this goal, the following research question guide the course of this dissertation:

RQ *Can innovative teaching methods significantly improve student learning outcomes in introductory concurrent programming and data management courses?*

In order to explore this question in more detail, each chapter examines a number of subquestions. These explore specific aspects of the main research question in greater depth. Specifically:

- Chapter 4: Does the use of Sonic Pi help the students face common misconceptions and mistakes in concurrent programming? Do misconceptions and mistakes in Sonic Pi correspond to misconceptions and mistakes in traditional concurrent programming languages? Can we apply knowledge transfer from Sonic Pi to more traditional programming languages such as C and C++?
- Chapter 5: Is collaborative learning effective, in terms of learning outcomes, for the different skills to be acquired in an introductory university database course?

While the core research experiments in this thesis are aimed at undergraduate CS students, we have also conducted outreach activities aimed at secondary school students, which have yielded promising initial results. These outreach efforts have the potential to inform the design and inspiration of future initiatives for this younger demographic. Ultimately, promoting a broader understanding of computing and the diverse career paths it offers in secondary schools could guide and inspire future generations of computer scientists and data enthusiasts. This, in turn, could lead to a more informed student body entering universities, potentially reducing dropout rates - a challenge faced by many institutions, including our own.

In addition, these outreach efforts may also help to break down negative stereotypes that hinder perceptions of CS and discourage students from pursuing careers in the field. Common stereotypes include the portrayal of computer scientists as socially awkward, overly competitive, and predominantly male [102]. In addition, there are misconceptions that programming is an exceptionally difficult endeavour, accessible only to those with a perceived "geek gene" [132].

It is important to acknowledge that while dropout rates and stereotypes are not directly addressed by the research questions of this thesis, they were a key motivation for the establishment of the Innovative Teaching Project at our university, which provided financial support for our experiments. This project, established with the aim of strengthening teaching and improving student learning and engagement through innovative pedagogy, serves as the broader context from which

the experiments presented in this thesis emerged. In essence, these experiments were conceived within the framework of the project, but were specifically designed to address the identified gap in the CS educational landscape.

Contributions

It should be clear that it is not possible to provide definitive answers to such broad research questions, especially within the narrow confines of a PhD thesis. Moreover, behind these questions lie a number of variables that influence educational experiments, many of which defy categorisation or quantification.

Ethical considerations also constrained our research approach. We deliberately chose not to use randomised controlled trials, a common research methodology, due to the potential for ethical dilemmas and the desire to maintain a natural learning environment for all participants. While these limitations may limit the generalisability of our findings, they do not diminish the value of our research. Instead, they serve as a stark reminder of the intricate complexities of education and the need for continued research and refinement of teaching practices. In particular, we believe that our findings can serve as a beacon for other educators and researchers, providing support and inspiration in the field of CS and data education.

In what follows, we briefly present the key suggestions derived from our research:

- Several programming practices and educational tools have an inherent constructionist flavour (e.g., visualisation, sharing, creative construction of an artefact, etc.) that can be used to introduce concepts that facilitate and promote a learning-by-doing approach.
- Constructivist learning environments should be cultivated, where students can collaborate, share ideas, reflect, and receive formative feedback, thereby strengthening their metacognitive and soft skills.
- Integrating domain-specific and real-world applications into the learning process can tailor the curriculum to students' interests and learning styles. This can increase student motivation, engagement and learning.
- Integrating domain-specific tools like Sonic Pi into the classroom can not only increase motivation, but also make learning more natural and intuitive, leading to deeper understanding.
- Using the notional machine as a central learning objective. This approach emphasizes understanding the underlying mechanisms rather than just syntax. It influences every aspect of instruction, from language choice and abstraction level to addressing misconceptions.

- Recognising DS as a discipline distinct from CS, with its own methodologies and mindset ("data thinking").

Within this framework, the thesis specifically contributes an innovative Python library specifically designed to introduce data science concepts to beginners using a simplified notional machine, an introduction to the data science lifecycle using real-world data, a collaborative learning approach to database logical design, and a teaching format to introduce concurrency using the music domain of Sonic Pi, a digital music creation software.

Thesis Overview

This section provides a detailed outline of each chapter of the thesis. The latter is divided into three main parts:

- Part I: Introduction and Background. This part provides a comprehensive overview of the current landscape for teaching both concurrency and data-related topics in CS. It begins by laying a solid pedagogical foundation in Chapter 2, providing the reader with the necessary knowledge to understand the subsequent chapters. Chapter 3 looks at the state of the art in data education, with a particular focus on databases and introductory data science. It introduces data science as a new discipline and a distinct way of thinking, known as "data thinking", and contrasts it with computational thinking.

- Part II: Innovating Concurrency and Data Education. The second part explores the core contribution of this thesis. Chapter 4 then presents a proposal for teaching concurrent programming through music using Sonic Pi and a collaborative methodology that specifically addresses common misconceptions about concurrency. Chapter 5 then presents an innovative experiment using a collaborative methodology for database logical design, while Chapter 6 presents a Python library that we have specifically designed to facilitate the introduction of programming with data-centric pedagogy.

- Part III: Teaching experiences and Educational Proposals. The third part of the thesis builds on the insights and findings of the previous chapters. It presents a series of carefully designed experiments and concrete proposals aimed at advancing the field of CS and data education. Chapter 7 presents three different approaches (including the data-centricity) to introduce Python programming; Chapter 8 presents an experience with high school students to introduce data literacy through data visualisation using Tableau software. Finally, Chapter 9 proposes a formal 6-credit course to introduce DS fundamentals to students with no CS background.

Teaching Innovation Project

From the 2019/2020 academic year, our University has launched a Teaching Innovation Project, in line with the initiatives promoted by the University Committee for Teaching Innovation (CIDA)[3]. The aim is to support courses that wish to experiment with new teaching and evaluation strategies and innovative technologies in order to strengthen students' learning, engagement and intrinsic motivation to learn (see e.g. [30]). The project consists of a number of phases:

- *Course redesign:* it involves the redefinition of the objectives, contents, and teaching methodologies of one or more courses.
- *Teacher training:* it provides teachers with the knowledge and skills necessary to implement the chosen teaching and evaluation strategies.
- *Support for the creation of teaching materials:* it helps teachers in creating innovative teaching and evaluation materials.
- *Support for the delivery of teaching:* it assists teachers in using innovative teaching methodologies in the classroom.

Our CS department has been involved in the project since the first year of experimentation. Initially, three courses were involved: Introduction to Programming (CS1), Algorithms and Data Structures, and Calculus. In subsequent years, the experimentation was extended to several other courses, such as Algebra and Logic, Computer Architecture, Concurrent and Distributed Programming, and Databases.

In particular, within the CS1 course, we initially experimented with the team-based learning teaching methodology and Sonic Pi to introduce students to more advanced topics and to increase motivation, engagement and curiosity, as well as soft skills such as teamwork. This experiment formed the basis of the experiments presented in this dissertation.

Note on Human Participants

Participants in the experiments of this thesis gave their consent to the use of the data presented (e.g., solutions to the proposed exercises) for scientific research purposes only. Concerning ethical issues, the requirement for approval was waived by the research ethics committee of our University for activities proposed within modules dedicated to innovative teaching methods designed with the support of the University teaching and learning center, such as those presented in this chapter. The reason is that the sole goal of these modules is to improve the learning

[3] https://unige.it/ateneo/comitato-linnovazione-didattica-ateneo-cida

of students through specific training activities strictly related to the topics of the courses, and the corresponding research aims at measuring their effectiveness. Data privacy and retention is ensured by collecting and maintaining anonymised data.

Author's Publications

[1] Daniele Traversaro, Giorgio Delzanno, and Giovanna Guerrini. "Hear" and "Play" Students Misconceptions on Concurrent Programming using Sonic Pi. *Journal of Informatics in Education (2024)*,DOI 10.15388/infedu.2024.22.

[2] Daniele Traversaro. Online Think–Pair–Share in a Third-Age ICT Course. *International Journal of Educational and Pedagogical Sciences*, 16(7):371 – 376, 2022.

[3] Giorgio Delzanno, Giovanna Guerrini, and Daniele Traversaro. Exploring Student Misconceptions about Concurrency Using Sonic Pi. In *Proceedings of the Technical Symposium on Computer Science Education (SIGCSE TS 2024)*, March 2024.

[4] Giorgio Delzanno, Luca Gelati, Giovanna Guerrini, Angela Sugliano, and Daniele Traversaro. Experience-based training in computer science education via online multiplayer games on computational thinking. In *International Workshop on Higher Education Learning Methodologies and Technologies Online (HELMETO '22)*, pages 459–470. Springer, 2022.

[5] Giovanna Guerrini and Daniele Traversaro. Two Introductory Data-driven Activities for Secondary Schools. In Proceedings of *31st Symposium of Advanced Database Systems (SEBD 2023)*, pages 641–649, 2023.

[6] Barbara Catania, Giovanna Guerrini, and Daniele Traversaro. Collaborative Learning in an Introductory Database Course: A Study with Think-Pair-Share and Team Peer Review. In *1st International Workshop on Data Systems Education*, pages 60–66, 2022.

[7] Daniele Traversaro, Giovanna Guerrini, and Giorgio Delzanno. Sonic pi for TBL teaching units in an introductory programming course. In *Adjunct Publication of the 28th ACM Conference on User Modeling, Adaptation and Personalization*, pages 143–150, 2020.

[8] Giorgio Delzanno, Giovanna Guerrini, and Daniele Traversaro. Sonic TBL: Un percorso sonico da creatività a didattica dell'informatica. In Proceedings of *ITADINFO 2023: Convegno italiano sulla didattica dell'informatica*, pag. 109-116, Bari, 2023.

[9] Daniele Traversaro, Giorgio Delzanno, Giovanna Guerrini, and Davide Ponzini. Python per tutti i gusti: tre diversi approcci per introdurre neofiti alla programmazione. In Proceedings of *ITADINFO 2023: Convegno italiano sulla didattica dell'informatica*, pag. 101-108, Bari, 2023.

[10] Giorgio Delzanno, Giovanna Guerrini, Daniele D'agostino and Daniele Traversaro. Un Macchina Nozionale per Architetture dei Calcolatori come possibile collegamento tra corsi del primo anno di informatica. In Proceedings of *ITADINFO 2023: Convegno italiano sulla didattica dell'informatica*, pag. 124-131, Bari, 2023.

[11] Giovanna Guerrini and Daniele Traversaro. Introduction to data literacy with Tableau in high school. In Proceedings of the Italian Conference *Informatica per la Didattica*, *Didamatica 2022*, pages 239–245, 2022.

[12] Manuela Chessa, Giorgio Delzanno, Giovanna Guerrini, and Daniele Traversaro. Virtual learning environments for remote lecturing: Comparing Mozilla Hubs and Gather. In Proceedings of the Italian Conference *Informatica per la Didattica*, *Didamatica 2022*, pages 523–532, 2022.

[13] Marilena Carnasciali, Sara Garbarino, Giovanna Guerrini, Daniele Traversaro, Luca Gelati, and Vincenzo Petito. Team Metrics: dall'aula al team – Una app di supporto alle metodologie didattiche basate sui gruppi. Faculty Development e Innovazione Didattica Uiversitaria, pages 387–404, Genoa University Press, 2021.

[14] Barbara Catania, Sara Garbarino, Giovanna Guerrini, and Daniele Traversaro. Un'esperienza di cooperative learning a distanza nell'insegnamento di basi di dati. In Antonella Lotti, Gloria Crea, Sara Garbarino, Federica Picasso, and Erika Scellato, editors, *Faculty Development e innovazione didattica universitaria*, pages 365–376. Genoa University Press, in Italian, 2021.

Chapter-Specific References This thesis draws material from several sources. In particular, Chapters 3 and 5 draw on references [1] and [6] respectively. In addition, elements from reference [5] have been incorporated into Chapters 6 and 8, and [11] into Chapter 8.

Chapter 2

A Constructionist Approach to Programming

This chapter focuses on the pedagogical and methodological foundations of computer programming education, which forms the theoretical basis of our teaching experiments presented in the following chapters. In particular, the chapter consists of a brief introduction to constructionism and creative thinking, computational thinking, and mental model theory in programming, introducing the notion of notional machine and misconceptions. The chapter aims to give the reader the basis for understanding the following experiments, which focus on pedagogy in the context of concurrent programming.

2.1 Computational Thinking: The Contributions of Papert and Wing

Computational thinking (CT) has become an essential skill for all, to the point that it can be considered the fourth fundamental skill alongside "reading, writing, and arithmetic" [194]. Various definitions of CT have emerged over time [70, 44], some definitions emphasize its ties to CS [126], while others highlight its broader applicability. Others have given a definition very closely related to coding [24]. This diversity poses challenges for educators aiming to integrate CT effectively into teaching practices. In this section, we focus on two definitions given by Seymour Papert, the father of constructionism, and Jeanette Wing, the pioneer of modern computational thinking. It is based on the results published by Lodi and Martini in [106].

The origins of the term can be traced back to Papert's book "Mindstorms" in 1980, which emphasised the role of computers in understanding abstract mathematical concepts [130], even those that Piaget constructivists considered rigidly confined to the formal operational stage.

> *"Their* [of people using computers for providing mathematically rich activities] *visions of how to integrate computational thinking into everyday life was insufficiently developed"*. [130]

The focus of Mindstorms is not on "thinking" per se, but rather on *constructing* concrete versions of abstract ideas using computers as powerful *meta-tools* for understanding the world.

In 2006, Jeanette Wing's seminal essay, "Computational Thinking" [194], brought the term CT to prominence. She advocated for the importance of *"thinking like a computer scientist"* in various fields, emphasizing the benefits of CT beyond the realm of programming. Following her, numerous efforts have been made to refine and elaborate on this concept, especially within K-12 education [44]. In 2008, Wing herself provided a more precise definition, often referred to as the "Cuny, Snyder, Wing" definition:

> *"The thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent."* [195]

This definition extends beyond well-structured problems to encompass complex real-life issues with undefined or immeasurable solutions [164]. Wing emphasized that CT is a fundamental component of CS, where its abstractions can be executed by computational agents. This shift from merely *"solving problems"* to *"effectively solving problems and making something perform computations for us"* distinguishes CT from other disciplines.

According to Wing, the key elements of CT are abstraction and automation, as *"computing is the automation of our abstractions"* [195]. She acknowledges overlaps between CT and other forms of thinking, such as logical and algorithmic thinking [196]. CS abstractions, as per Wing, can be "animated" without physical manifestation, with executable models being discrete, efficient, and scalable across abstraction levels. These characteristics set CS models apart from other scientific models. Additionally, CS offers linguistic support through programming languages for its abstractions, enabling systematic definition and facilitating the transition between abstraction levels leading to physical execution [105].

Thinking Concurrently

Although "concurrency" is not explicitly mentioned in any of the CT definitions, "parallelism" is present as early as in Wing's definitions and in many others [104]. In [196], Wing recognised the significant overlapping or inclusions between CT and other types of thinking, including parallel thinking. Furthermore, the computational concept "parallelization" (or, sometimes, "parallelism") is explicitly included in several definitions of CT, such as those proposed by ISTE &

CSTA [83], Google[1], and Brennan and Resnick [24]. Parallelism is considered to be one of the operational approaches to CT widely used by computer scientists. These definitions emphasise the ability to simultaneously process smaller tasks of a larger task to achieve a common goal more efficiently.

In essence, the role of concurrent thinking in CT is to efficiently manage multiple tasks that occur simultaneously, allowing us to break down complex problems into smaller, manageable tasks that can be performed simultaneously (decomposition). By thinking concurrently, we can design algorithms and systems that make full use of these resources (abstraction), leading to more efficient and faster solutions (evaluation). Moreover, thinking concurrently is not just about making code run faster. It's also about making systems more responsive (pattern recognition: identifying opportunities for parallelization in different problem types).

Many CT tools, such as Scratch[2] and Alice[3], which express their designers' view of how a selected set of CS concepts should be introduced, go beyond simply teaching concepts. In fact, they use a model of concurrent behaviour as a hidden message that shapes how users learn and interpret concurrency-related ideas [167]. These meta-messages act as a guiding principle within the tool and may facilitate future learning of complex concurrency concepts.

2.2 Learning Theories and Paradigms

This section introduces constructionist learning theory and the creative learning method, which are the pedagogical foundations of the teaching experiments presented in this dissertation. It also introduces two educational programming languages that embody these approaches, namely Logo and Scratch.

2.2.1 Constructionism

Constructionism is a learning theory developed by South African mathematician and computer scientist Seymour Papert, co-inventor of the LOGO programming language, which builds on the foundations of constructivism. While constructivism emphasises the active construction of knowledge through individual experience and interaction with the environment, constructionism places a particular emphasis on learning through the concrete act of creating artefacts. At the heart of constructionism is the belief that learning is most effective when learners are actively involved in constructing meaningful objects (*"whether it's a sandcastle on the beach or a theory*

[1] Google Computational Thinking Concepts Guide
[2] https://scratch.mit.edu
[3] http://www.alice.org

of the universe" [130]). This process of creation involves the use of *"building materials"*, both concrete and abstract, as learners transform their ideas into tangible expressions.

For example, Papert notes that as a child he was obsessed with gears and used mental models of how gears worked as a tool for understanding the world, even complex abstract mathematical concepts [130]. In this context, Papert emphasises the importance of *"objects to think with"* and believes that learners need these objects in order to assimilate new knowledge into their existing mental models. Computers play a central role in this by providing a platform for the creation of virtual worlds (called *microworlds*) in which learners can engage with different topics, such as *"Mathland"* for mathematics, allowing learners to manipulate and visualise abstract ideas in a concrete form, even those that Piaget's constructivist saw as rigidly confined to the formal operational stage. Papert believed that computers and programming would revolutionise education by creating constructivist environments, similar to the traditional Brazilian "samba schools" that were fundamental to the preparation of the Rio Carnival, rendering traditional school systems obsolete and empowering learners in areas such as mathematics, physics and probability from a computational perspective [106]. All of this is present in Papert's reference to *"samba schools of computation"* [130].

In summary, constructionism emphasises the value of learning by making, promoting personal relevance and meaningful projects aligned with learners' interests. Collaboration is encouraged to construct knowledge collectively, fostering problem-solving skills and social interaction. Reflective thinking and iterative processes are central, guiding learners to analyze and improve their work. Technology serves as a facilitative tool, enabling exploration and enhancing learning outcomes.

2.2.1.1 LOGO and Syntonic Learning

LOGO is an educational programming language designed by Seymour Papert, Wally Feurzeig and Cynthia Solomon in 1967 for primary school children. It is a multi-paradigm adaptation of Lisp and has a graphical environment. In particular, LOGO's emphasis on the creation and manipulation of graphical objects was perfectly in line with the principles of constructionism, enabling learners to construct their own understanding through the process of creation. Indeed, the programming instructions in LOGO were directed to a virtual "turtle" (or a physical turtle robot), a small triangle that could move around the screen and leave a trail. Specifically, each instruction corresponded to an action to be given to the turtle. By drawing graphical patterns with the turtle, students could develop an understanding of 2D geometry. Figure 2.1 shows some introductory examples of turtle programs from the book "Turtle Geometry" [3]. In addition, they might even discover profound mathematical truths, such as that a circle can be approximated by several straight segments [3].

For Papert, students could identify with the turtle itself and promote *body syntonic learning*. Papert identifies three types of syntonicity: body syntonic, ego syntonic and culturally syntonic.

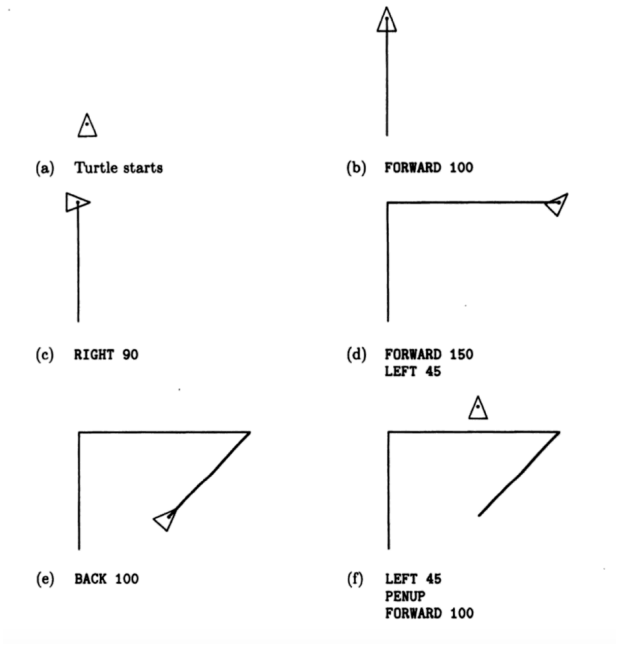

Figure 2.1: Some examples of Turtle programs.

The first refers to the awareness of body movement; ego refers to the idea that children engage in tasks or games that have personal meaning for them; cultural refers to engaging students in tasks that are consistent with the child's cultural context and environment. In his book Mindstorms he stated:

> *"The Turtle circle incident illustrates syntonic learning. This term is borrowed from clinical psychology and can be contrasted to the dissociated learning already discussed. Sometimes the term is used with qualifiers that refer to kinds of syntonicity. For example, the Turtle circle is body syntonic in that the circle is firmly related to children's sense and knowledge about their own bodies. Or it is ego syntonic in that it is coherent with children's sense of themselves as people with intentions, goals, desires, likes, and dislikes. A child who draws a Turtle circle wants to draw the circle; doing it produces pride and excitement."* [130].

What Papert describes is the harmony between the child's understanding of how to draw a circle, what a circle is, and how to tell the turtle to do the same. In the case of body syntonicity, the physical movements of the child and the machine are in harmony. The turtle acts as a bridge between the student and geometry, acting as what Papert called a "transitional object" [131]. While the child has spatial awareness, they may struggle to articulate their knowledge of geometry. The turtle represents a mathematical entity that is not overly abstract, allowing students to relate to

it. As the turtle moves in a similar way to the child, the child can use familiar descriptions and communicate with it through the computer.

Researchers have identified a link between syntonic learning and constructivist learning theory [188]. According to Papert, if the learning experience is too challenging or difficult to understand, learners are more likely to become frustrated and give up. Conversely, if the task is too easy or uninteresting, learners tend to lose interest and give up. This is in line with the principles of the zone of proximal development. Consideration of syntonicity is an approach to instructional design that encourages learners to actively engage with content in a more meaningful way [188].

It should be clear that LOGO was originally conceived to teach mathematics and not directly computer programming, which was a tool for learning, a means of empowering students with mathematical ideas. However, Papert recognised an epistemic potential in the programming approach. Indeed, "powerful ideas" enable new ways of thinking [106] According to Papert:

> *"In teaching the computer how to think, children embark on an exploration about how they themselves think. The experience can be heady: Thinking about thinking turns every child into an epistemologist, an experience not even shared by most adults".* [130]

In summary, LOGO can be seen as the forerunner of all coding initiatives, embodying the core principles of constructionist theory. Numerous contemporary programming languages still implement the "turtle" function, from its heir Scratch to Python. The latter provides a turtle package in its standard library, as shown in Fig 2.2. Finally, many of Papert's ideas have been used as inspiration for the Lego© Mindstorm kits.

2.2.2 Creative Learning

Creative learning, as defined by Mitchel Resnick [159], is an educational methodology that promotes creative thinking, problem solving, and self-expression through the use of technology and other expressive materials. The MIT Media Lab, particularly through the Lifelong Kindergarten group[4], has developed Scratch, a visual programming language based on creative learning and designed specifically for young learners. Its intuitive drag-and-drop interface and engaging storytelling environment make it an accessible and powerful tool for creative learning.

Resnick's approach of creative learning is based on constructionis and on the belief that everyone has creative potential. By providing learners with opportunities to explore, experiment and create, we foster their ability to think creatively, solve problems and express themselves authentically by creating personally meaningful projects using digital tools and technologies.

[4] https://www.media.mit.edu/groups/lifelong-kindergarten/overview/

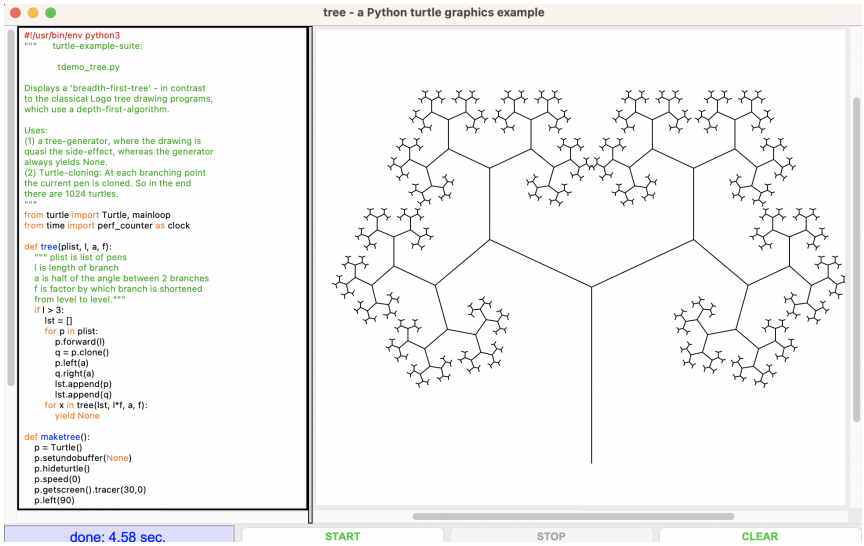

Figure 2.2: A Python turtle graphics example.

In particular, Resnick emphasises the concepts of *low floors* and *high ceilings* [146], originally introduced by Seymour Papert. Low floors mean that learners can get started quickly and engage in meaningful exploration without being overwhelmed. High ceilings, on the other hand, represent opportunities for learners to push their boundaries, delve deeper, and engage in more complex and challenging creative projects as their skills and confidence grow. Resnick expands upon the concepts of low floors and high ceilings by introducing the notion of *wide walls*. It refers to providing a range of possibilities and multiple pathways for learners to explore their interests and express their ideas. Rather than prescribing a single correct solution or outcome, wide walls allow learners to take ownership of their learning journey, make choices, and find their own creative solutions.

In summary, the creative learning model can be described by the "4 Ps" framework [159], which stands for Projects, Passion, Peers and Play:

- **Projects.** Projects refer to hands-on, personally meaningful activities that learners undertake to explore and create something of their own. Creative learning encourages learners to engage in project-based activities where they can pursue their interests, solve problems, and express their ideas through the process of creating tangible outcomes.
- **Passion.** Passion represents the learner's intrinsic motivation, interests, and personal connections to the learning content. When learners are engaged in activities they are passionate about, they are more likely to invest time and effort, take ownership of their learning, and unleash their creativity.
- **Peers.** Peers refer to the social aspect of learning and the value of collaboration and

sharing. Creative learning recognizes the power of peer interaction, collaboration, and feedback in the learning process. By collaborating with others, learners can exchange ideas, learn from different perspectives, and inspire each other's creativity. Peer interactions also foster a sense of community and provide a supportive environment for learners to take risks and explore new ideas.

- **Play.** Play encourages an open and exploratory mindset, where learners can experiment, take risks, and embrace the joy of discovery. It stimulates imagination, curiosity, and creative thinking.

Following these principles, the construction of a creative artifact follows an iterative process called the **Creative Learning Spiral** or *"the kindergarten approach to learning"* [145]. In this spiraling model, depicted in Figure 2.3, children **imagine** what they want to do, **create** a project based on their ideas, **play** with their creations, **share** their ideas and creations with others, **reflect** on their experiences – all of which leads them to **imagine** new ideas and new projects. The process is iterated many times.

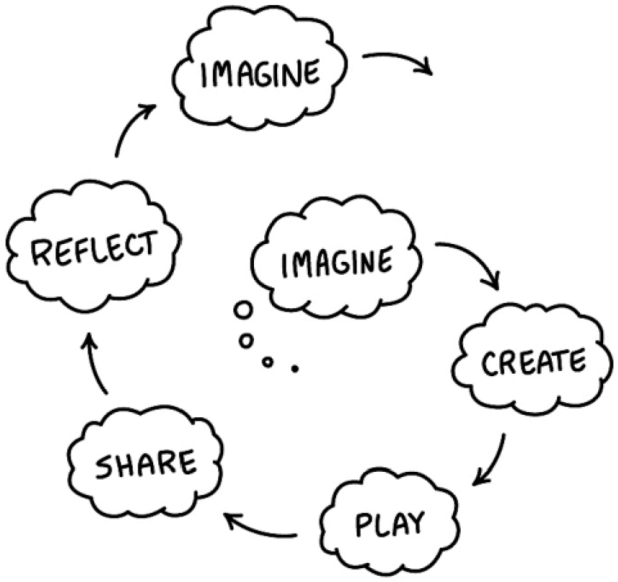

Figure 2.3: Creative Learning Spiral [145].

2.3 Mental Model of Program Behavior

In this section, we move from learning theory to mental model theory by introducing the concept of the notional machine. This part is essential for understanding the main constructivist strategy

used in introductory programming.

2.3.1 Notional Machine

A notional machine (NM) refers to an abstract concept used in programming education to help learners understand how computer programs are executed by a computer. It serves as a conceptual model that simplifies the understanding of complex computer operations and allows programmers to reason about the behavior of their code without needing to understand the specific details of a particular computer architecture. It is often used in introductory programming courses to teach basic programming concepts.

The term NM originated in the 1970s during the rise of the psychology of programming and the development of educational programming languages such as Logo. Benedict du Boulay introduced the concept, referring to it as *"the general properties of the machine that one learns to control"* [57]. One of the first NMs was presented in a Logo manual for primary education, using hand-drawn representations and analogies to aid understanding (see Figure 2.4).

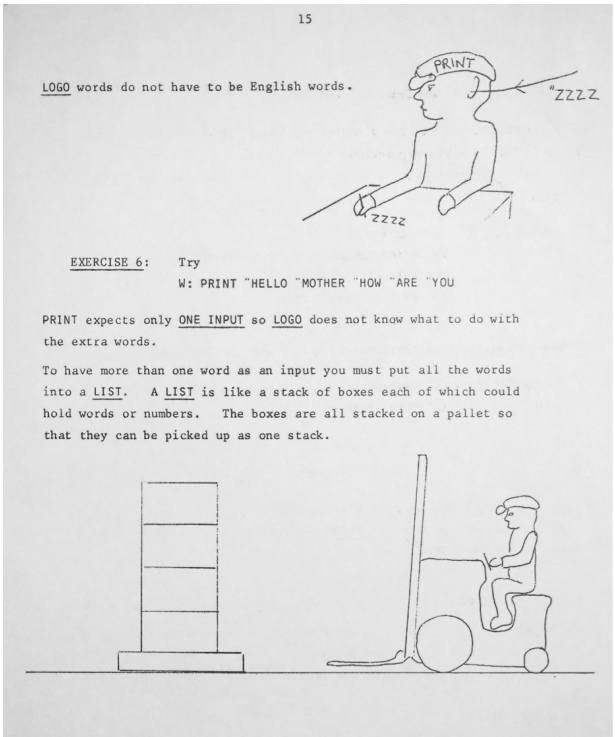

Figure 2.4: A page from the 1976 LOGO handbook (reproduced in [59]).

In the mid-2000s, the term NM gained currency in education and NM was informally defined as

"a model of the computer as it relates to executing programs" [148]. Later, a seminal review of NMs was published [172], focusing on program execution. The author characterised NM as an abstract representation of a computer's behaviour when executing programs within a particular programming language context. He goes on to show that different programming languages can have similar or different NMs, and that a single language can have multiple NMs describing program execution at different levels of abstraction. In addition, NMs can help to compare programming languages based on their programming paradigms [93].

Finally, according to [172], a NM can focus on three different areas of learning: the programming language itself, the "machine" – as in du Boulay's original definition – and the interaction between the language and the machine. A NM is thus an idealized model that guides students toward aspects related to some or all three areas. For example, if the focus is on a particular feature of Python, the domain will be that of the programming language; if the focus is on aspects related to a machine (e.g., the number of pins in a BBC micro:bit), the focus will be on the hardware on which the program will run. Depending on the context (i.e., level of abstraction), the machine may be a physical machine, a virtual machine, an operating system, or even a programming environment (e.g., Scratch).

2.3.2 Mental Model Theory

A mental model is a representation of a system or environment in the mind. In particular, mental models reflect people's beliefs about the system they are interacting with, often based on naive assumptions [129]. Indeed, people often rely on analogies of superficial features when forming mental models of new systems. Mental models then evolve over time through interactions with the system. They are simplified explanations of phenomena and can be mentally simulated to predict system behaviour, albeit with limitations.

Constructivism emphasizes that learners actively construct knowledge based on their prior experience, rather than passively receiving it from teachers. However, learners often develop mental models that are not viable, i.e., inaccurate, incomplete, and based on superficial understanding. Students tend to stick to familiar models, making it more difficult to change existing mental models than to create new ones. As a result, learning a second or subsequent programming language is generally still a difficult transfer task [160].

The researchers suggest that teachers explicitly introduce the NM, an idealized computer that represents the properties implied by the constructs of the programming language. The NM helps learners develop valid mental models and improve their understanding of programming concepts [57]. Following [55], the relation between an NM of a given hardware/programming language and the student's mental model is depicted in Fig. 2.5.

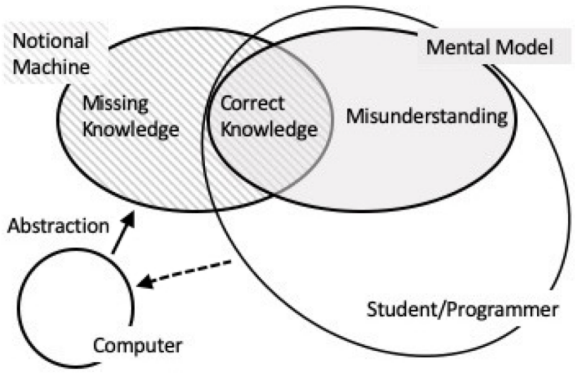

Figure 2.5: Notional machine and students' mental model [55].

An example of a weak mental model is discussed in [189], with the following code snippet:

```
for (i = 0; i < 10; i++) {
    System.out.println(i);
}
```

Specifically, the study highlights that novices tend to confuse the interpretation of the semantics of the `for` statement (a program component) with its specific application in a particular context. In particular, students tend to think that whatever variable is used in the body of the for loop is always set to zero at the beginning of the loop. This understanding is clearly incorrect. In addition, [189] argues that the problem is common in CS1 courses, where teachers often associate a program statement with a single problem/context and vice versa, leading to incorrect mental models.

With experience, mental models become more stable and accurate, leading to mental schemas of computer behaviour that resemble the thinking of expert programmers.

2.3.2.1 Program Tracing

Mental models, as described by [129], are *runnable* and can be simulated in mental working memory. However, this cognitive capacity is limited, so it is necessary to simulate problems at an appropriate level of abstraction, focusing only on essential system factors. In computer programming, this simulation of program execution is known as *program tracing*, a fundamental skill for design and comprehension tasks.

Unfortunately, novice programmers often struggle with program tracing because it requires tracking the state of program execution [172]. The information involved can overwhelm the memory

capacity of even professionals, leading to the use of external debugging tools. Novice programmers face additional challenges because they lack experience in focusing on the right elements, leading to failures due to excessive cognitive load.

Teachers should be aware of this, design tasks with the right level of difficulty, and use learning aids and visual tools to overcome working memory limitations, such as PythonTutor[5].

2.4 Constructionist Programming

Programming is generally regarded as a fundamental component of constructionist approaches. In programming, individuals are tasked with creating a solution to a given problem that can be mechanically executed by an executor. To program effectively, students need to know the basic actions that the executor can perform, as well as to understand the rules and syntax of the programming language. They also need to understand the relationship between syntax and semantics. Learning to program therefore involves understanding the dynamic nature of program execution and the process of creating functional and meaningful artefacts.

In order to assess a learner's mental model, we need a conceptual framework that represents the types of knowledge required to complete a particular task [97]. Cognitive scientists generally agree that there are three types of knowledge involved in computer programming: syntactic, conceptual and strategic [16].

Syntactic knowledge refers to an understanding of the syntax of a programming language, such as the grammar and format of statements. Conceptual knowledge refers to the dynamics of the program, such as knowing how programming constructs work and how they determine the execution of the code. Finally, strategic knowledge relates to problem-solving strategies and refers to the ability to apply the syntactic and conceptual knowledge to develop programs to solve novel problems. Typically, novice programmers *"know the syntax and semantics of individual statements but they do not know how to combine these features into valid programs"* [198]. The following Python example illustrates these concepts:

```
x = 0
while(n != 0):
    n = int(input("Enter a value "))
    x += n
```

In this code snippet, the syntactic knowledge is to understand the syntax of the while loop and the syntax of the addition operator (+=). The conceptual knowledge is to understand the behaviour of the while loop, knowing that it will continue to execute as long as the value of n is not 0, and

[5] https://pythontutor.com

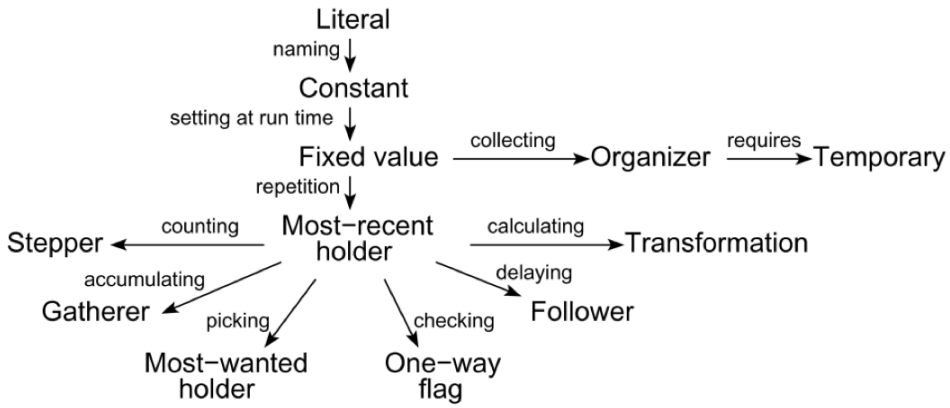

Figure 2.6: Roles of variable hierarchy [151].

that it will stop as soon as the condition is false. The strategic knowledge lies in recognising the variable `n` as an accumulator variable used to calculate the sum of the inputs, accumulating each input value in the variable `x`.

2.4.1 Micro Patterns, Goals and Plans

Mastering strategic knowledge means being able to apply and recognise abstractions in the program, namely micro-patterns. The concept of the role of variables [151, 106] has been proposed to guide novice programmers from initial operational knowledge of a variable to the ability to identify abstract use cases that help to identify the solution to a specific problem. During the process of learning to program, students may realise that the problem is a special case of a more general family of problems, leading them to reason in terms of abstractions.

Patterns should be introduced incrementally to construct and refine the existing mental model grounded in prior knowledge. Literature suggests adhering to the roles of variable hierarchy depicted in Figure 2.6, as elaborated by [151]. The hierarchy should be interpreted as follows: the predecessor of an arrow is a learning prerequisite for the corresponding successor. For example, in the case of variables, teachers should start with the concept of literals and then introduce a role on top of the roles already understood, until more patterns are combined in the same program for complex problems.

These considerations should make it clear that there are differences between novice and expert programmers in the way they analyse and understand programs. Novices tend to read programs sequentially, line by line. Experts take a different approach, looking for “beacons” - lines of code that serve as typical indicators of certain structures or operations [193, 26]. These beacons help experts to identify significant sections of code known as plans or “chunks” [169]. In particular,

experts use abstractions such as micro-patterns to identify the high-level goals and plans within a program.

The following example illustrates the process of designing a solution to a problem by identifying the goals and corresponding plans:

```
Write a program that calculates the average.

Goals: sum N values; count the values; divide the sum by N

Plans: Sum loop (with variable gatherer/accumulator);
       Count loop (with stepper variable);
       Guarded division (division by zero is checked)
```

The example provides a clear understanding of strategic knowledge, which involves problem-solving strategies based on goals and plans identified by micro-patterns such as loop patterns and variable roles. At a higher cognitive level, strategic knowledge involves selecting, combining and tailoring constructs and plans to solve the task [191]. Plans are used as basic building blocks and can be integrated, as presented in [169], by **abutment** (i.e. concatenation, i.e. plans are applied in full one after the other), **nesting** (i.e. plans are applied nested around each other, e.g. double for loop to scan cells of a matrix), **interleaving** (i.e. merging several plans) and **tailoring** to create novel solutions (i.e. adapting the plan to a particular scenario, e.g. a different initialisation of an accumulator variable).

2.4.2 Threshold Concepts

This section provides an overview of threshold concepts and their importance in relation to NMs and, more generally, in programming education. Threshold concepts are key ideas in a field of study that bring about a profound change in learners' understanding and thinking [113, 172]. Once grasped, they become deeply embedded in learners' knowledge frameworks and have a lasting impact on their cognitive structures.

Threshold concepts often challenge learners' preconceptions and require a paradigm shift in thinking. However, they facilitate the integration of different aspects of the subject, promote holistic understanding and enable knowledge transfer across different domains.

There are a number of threshold concepts in programming. Many of them, such as parameters, control flow, recursion and variables, form the basis of effective programming skills [86]. Among these, program dynamics is particularly critical, involving an understanding of program behaviour and the thought processes of experienced programmers [172]. It serves as a boundary between programmers and non-programmers, and its learning is typically irreversible [171].

Other common threshold concepts are program-memory interaction, such as pointers, and object-oriented programming [21].

Recognising the importance of threshold concepts is essential for effective instructional design. Educators should use strategies such as active learning and scaffolding to guide students through these challenges and promote deep understanding and mastery of programming.

2.4.3 Misconceptions

Learning to program, as a process of creating a mental model of the underlying NM, is a difficult task where students can fall into several programming misconceptions [57]. The latter, as defined by Sorva, are *"understandings that are deficient or inadequate for many practical programming contexts"* and can significantly impede progress in learning to program. He further emphasizes the detrimental impact of misconceptions, stating that *"Incorrect and incomplete understandings of programming concepts result in unproductive programming behavior and dysfunctional programs"*.

The root cause of many programming misconceptions lies in prior knowledge from domains outside of programming, such as mathematics and natural languages [170]. High-level programming languages, while providing a convenient level of abstraction, can paradoxically contribute to misconceptions. By hiding the underlying machine-level details, these languages can obscure the concrete execution of programs on the computer. This abstraction, intended to simplify programming, can make it difficult for students to understand the connection between constructs and their actual implementation, leading to bugs and errors.

The literature on programming misconceptions is extensive and common examples include English keywords, mathematical notation and similarities [105]. For example, students may misinterpret the meaning of keywords such as while or if, believing that the loop will continue until the condition is false, or that the then branch will be executed immediately upon condition fulfilment. The assignment operator = can be misinterpreted as an equation or value exchange, and the increment operator ($a = a + 1$) can be considered an invalid expression. Finally, misconceptions can arise from similarities (e.g. metaphors and analogies), such as equating a variable with a box, or treating programming as a conversation with a sentient computer (the so-called "superbug" [134]). This can lead to attributing intentionality to the computer and overestimating its capabilities.

2.4.4 Program Comprehension

In learning to program, a fundamental aspect is the comprehension of code. However, it is only in recent years that the importance of understanding code has been recognised as an integral part

of programming education. Conventionally, educators have prioritized program composition, which encompasses writing code, while considering reading skills as an incidental byproduct of writing code. However, reading and writing code are related skills, but not directly interdependent.

In a study by [84], this competence is conceptualised by the term *"program comprehension"* and defined as *"a process in which an individual constructs his or her mental model of a program"*[84]. Furthermore, it introduced the concept of *"program comprehension tasks"* as a cognitive approach aimed at guiding students in their learning process to construct viable mental models. In particular, a program comprehension task is an exercise that exposes the learner to an artifact that represents the program. The exercise necessitates the learner's active engagement with the artifact in some form. This interaction stimulates the learner to construct and refine their mental model. The artifact can take various shapes, such as source code, Parsons problems[6], or collections of blocks in a visual programming language. Its level of abstraction can vary widely. To overcome the constraints of working memory, learners can utilize external representations like notes, traces, or diagrams during their interactions with the artifact [45, 84].

In [84], researchers propose a framework to assist teachers in considering the cognitive aspects of program comprehension. This framework is organised as a matrix that classifies program comprehension tasks from two perspectives. The first perspective considers the program at different block levels, ranging from individual instructions to individual blocks, relationships between blocks, and the entire program. The second perspective, orthogonal to the first, considers the program in three dimensions: textual surface (the program text as a static entity related to syntactic knowledge), program execution (a dynamic entity representing functional knowledge), and function/purpose (viewing the program as an artifact with an extrinsic purpose reflecting strategic knowledge). Figure 2.7 depicts the block model matrix.

(M) **Macrostructure**	Understanding the overall structure of the program text.	Understanding the *algorithm* underlying a program.	Understanding the goal/purpose of the program (in the context at hand).
(R) **Relationships**	Relations & references between blocks (e.g. method calls, object creation, data access...).	Sequence of method calls, *object sequence diagrams*.	Understanding how subgoals are related to goals, how function is achieved by subfunctions.
(B) **Blocks (Chunks)**	*Regions of Interest* (ROI) that syntactically or semantically build a unit.	Operations of a block, a method, or a ROI (chunk from a set of statements).	Understanding the function of a block, seen as a subgoal.
(A) **Atoms**	Language elements.	Operation of a statement.	Function of a statement: its purpose can only be understood in a context.
	(T) **Text Surface**	(P) **Program Execution**	(F) **Function/Purpose**
Duality	Architecture/Structure Dimensions		Relevance/Intention Dimension

Figure 2.7: The Block Model Matrix [84].

In this matrix, the rows represent a hierarchy of increasingly complex programming structures,

[6]A Parson's problem is a type of program comprehension task where learners are given all the blocks or lines of code needed to solve a problem, but the lines have been mixed up so that they are no longer in the correct order. The learner is asked to put the code back in the correct order to complete a specific task [84].

while the columns represent different dimensions of comprehension corresponding to different levels of abstraction.

2.5 Some Pedagogical Implications

In recent years, a variety of tools and methods for teaching programming have emerged, from project-based learning to peer education and pair programming [138] to visualization tools and block-based coding. However, designing a successful introductory programming course remains a complex endeavor. It requires careful consideration of the diverse characteristics of students, including factors such as school environment, familiarity with technology, individual interests, and learning needs. Sometimes the materials used to teach programming overload students' cognitive resources [112]. In addition, challenges arise from the ineffective use of teaching strategies for problem-solving and coding techniques [36], and from students' lack of mastery of programming syntax [149, 142]. Incorporating visualisations into explanations can be a powerful tool to help students develop clear mental representation of programs [149]. These challenges are compounded by time constraints and limitations imposed by curriculum structures. These difficulties are particularly pronounced in short introductory activities, where the aim is to engage students quickly, capture their attention and interest, and provide a clear understanding of programming concepts within a limited timeframe.

To address these challenges and improve the effectiveness of introductory programming education, this chapter has explored various pedagogical techniques with a constructionist approach. Some of the key pedagogical implications that can be drawn from this chapter include:

- **Shifting Learner Perspectives**: Shift students' perspectives on key concepts for deeper understanding and reflection (metacognition). Prioritize selected concepts to create more enriching learning experiences.
- **Understand Program Dynamics:** Grasping threshold concepts like recursion and reference parameters is crucial. Teachers should help students develop a dynamic view of program execution. Consider using visualization tools like PythonTutor to aid understanding.
- **Make NM explicit:** Clearly establish NM as a learning objective in introductory courses.
- **Enhance Code Comprehension:** Teach program tracing for improved program understanding. Reinforce code comprehension skills to help students build strong mental models.

Chapter 3

Data Science Education

Data Science (DS) education is the foundation for a wide range of programmes, including CS, DS itself, and many other related fields. It equips students with the essential skills to manage data throughout its lifecycle, from collection and storage to processing and analysis. Core topics explored in this area of education include database management, programming, and the ever-evolving field of machine learning (ML). Fundamental to all of these areas is a strong foundation in coding, as it enables students to effectively manipulate, analyse and visualise data, enabling them to tackle real-world DS challenges.

It is difficult to pinpoint a specific date of birth for data education, as the field has evolved over time in response to the growing importance of data in our increasingly digital data-driven world. We can say that early efforts in data education began in the 1970s, primarily within the database research community. These efforts focused on curriculum placement and best practices for teaching data systems concepts. However, it was not until the recent rise of big data and the high demand for DS professionals that universities began to establish dedicated DS programmes or curricula and to recognise DS as an emerging discipline [72]. This emphasis on data education has led to a growing focus on data pedagogy, a critical aspect of faculty development and curriculum design.

This chapter is intended to prepare the reader for the following chapters, which present our pedagogical experiments in Data Systems education. In particular, it aims to briefly present the history and epistemology of DS. It then discusses the state of the art in pedagogical and methodological approaches to data education, as well as the challenges and opportunities of its interdisciplinary nature, with a focus on introductory data science (IDS) courses at both school and university level. Finally, it focuses on database research education.

3.1 Data Science as Discipline

Over time, the term *"data science"* has emerged independently in CS, statistics, and other fields [82]. In CS, the term *"datalogy"* was proposed by Peter Naur in 1966 to the editor of the Communications of the ACM [127] as a new name to CS, *"the science of the nature and use of data"*. In addition, he also proposed the term *"datamaton"* to indicate the automatic device for processing data, and *"datum&tics"* to indicate the part of datalogy which deals with the processing of data by automatic means.

The term "data scientist" emerged in the late 2000s, initially in the tech industry. It was popularized by an essay in 2008 by data analytics leaders at LinkedIn and Facebook, which highlighted the importance of DS in business analytics [47]. The essay also noted the lack of formal educational programs in DS, suggesting that it was an emerging field with significant potential.

The rise of big data has presented challenges for industries, as they grapple with managing and processing large amounts of data that cannot be effectively handled using traditional methods. The term "big data" is often associated with its 5 V's: volume, variety, velocity, value, and veracity. It is important to note that the early research and development efforts in big data were primarily driven by industry, with companies like Google and Meta innovating and developing specialised tools and frameworks to address the challenges of big data management.

While some may perceive DS as merely a collection of existing methods and tools from various fields, its true innovation lies in its holistic approach [73]. DS integrates knowledge and skills from these different disciplines to extract knowledge and value from data. This integration enables DS to tackle complex problems that traditional, siloed approaches may not be able to tackle effectively.

In 2010, Conway [43] proposed a Venn diagram to illustrate the interaction and intersection between these fields, as depicted in Figure 3.1. Many other Venn diagrams have been proposed over the years [182].

Universities are actively exploring how to incorporate DS activities into their curricula and courses [35, 10] and integrate it with existing disciplines. However, despite these efforts, DS education has not yet achieved full recognition as a discipline in its own right. It is more often discussed in the context of related fields such as CS and statistics education. This lack of independent recognition is further evidenced by the absence of a dedicated academic journal focusing solely on DS education [71].

Several researchers see DS as an emerging discipline, moving from a multidisciplinary to an interdisciplinary state [7]. According to [73], the ultimate goal is to achieve transdisciplinarity, where fields are seamlessly integrated with a unified and holistic approach centered on data. However, there is still a long way to go to achieve transdisciplinarity, as the DS community faces many challenges and open questions. In a 2020 paper [197], Jeannette M. Wing, the pioneer

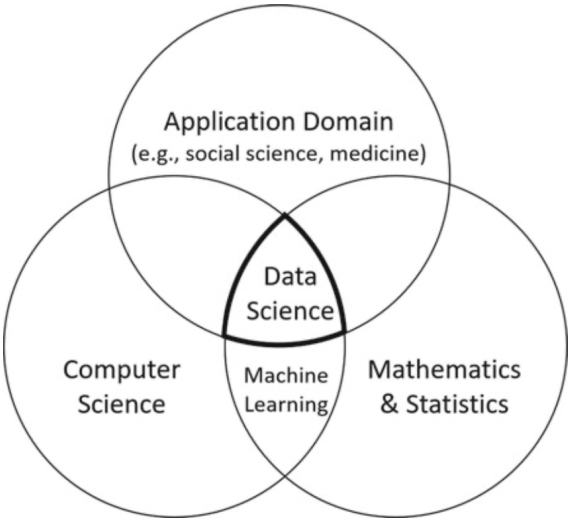

Figure 3.1: Data science Venn diagrams.

of modern CT, identified ten critical research areas that the DS community should prioritise to advance the field and address emerging societal needs, including scientific, technological and ethical challenges. These areas are: scientific understanding of learning, especially deep learning algorithms; causal reasoning; valuable data; multiple, heterogeneous data sources; reasoning from noisy and/or incomplete data; trustworthy AI; computing systems for data-intensive applications; automation of front-end stages of the data lifecycle; privacy; ethics. Beyond these specific research areas, Wing also presents three fundamental meta-questions that challenge the very nature of DS as a discipline: what are the deep questions of DS, what is the role of the domain in DS, and what makes DS unique.

3.1.1 Data Thinking

DS has introduced a new data-driven paradigm. As an emerging field, researchers are exploring the distinctive thinking patterns of data scientists as opposed to statisticians or computer scientists. Thus, the term *"data thinking"* (DT) has been proposed [120]. According to [29], DT refers to the perspective on the methods, processes, structures, characteristics and habits of the mind when dealing with data problems and systems. In particular, DT combines CT, statistical thinking, different mathematical conceptions and domain thinking [120], as depicted in Fig. 3.2.

Domain thinking is the ability to apply knowledge and expertise from a specific field or industry to analyze and interpret data. This domain knowledge allows data scientists to ask meaningful

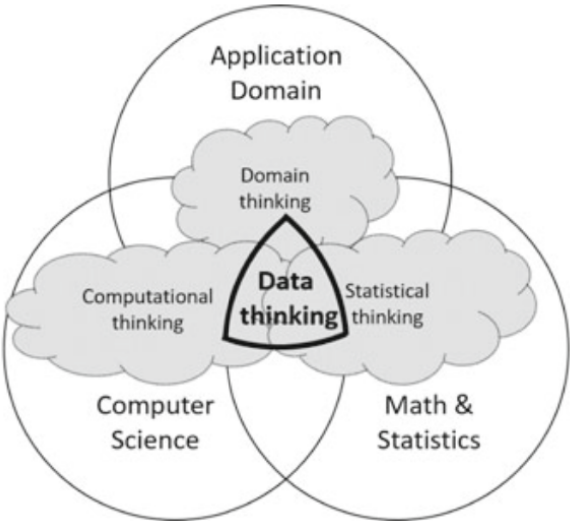

Figure 3.2: Data thinking Venn diagram.

questions, identify relevant patterns, and interpret results within the context of the problem [72].

Statistical thinking, introduced by W. Edwards Deming [54] and further developed by David Moore [125], is a mindset that emphasizes the importance of understanding the nature of data and its variability in solving real-world problems. It involves a process of inquiry that starts with formulating questions, collecting and analyzing data, interpreting results, and evaluating the validity of conclusions.

In summary, DT skills can be described as the cognitive processes required to understand whether data is a good representation of the real situation, to reason about data in terms of meaning within the relevant application domain and not just as a data structure, to analyze data both logically and statistically, and to recognize that problem abstraction is application domain dependent and that generalization is subject to bias and variance in the data [121].

3.1.2 Data Science Pedagogy

DS pedagogy is an interdisciplinary approach that integrates teaching methods from CS, statistics, mathematics, and various application domains. This interdisciplinary perspective provides learners with opportunities to engage with STEM subjects and to develop the cognitive, organizational, and research skills necessary for interdisciplinary work. It emphasizes active learning, contextualized learning, and white-box understanding.

Active learning promotes student engagement and problem-solving skills through hands-on activities, collaborative projects, and case studies. **Context-bases learning** uses real-life examples

and real data to connect theoretical concepts to practical applications. This approach increases learner motivation, engagement, and diversity.

In data education, machine learning (ML) and deep learning algorithms play a crucial role, as they are essential elements of DS, but require an extensive background in mathematics. Typically, a **white-box understanding** approach is preferred. The term "white-box understanding" refers to understanding the details of the algorithm, i.e. how it works, while **black-box understanding** refers to the ability to call a library procedure that executes the algorithm without understanding the internal process of how the algorithm's output is generated.

Practical constraints and prior knowledge may require a black-box approach where students focus on high-level intuition and input-output behavior. For example, to introduce deep learning algorithms, tools such as Google's Teachable Machine[1] can be a valuable resource to keep students engaged without getting bogged down in complex mathematical details.

On the other hand, novices may find it difficult to improve the performance of an algorithm without a white-box understanding. In fact, a superficial understanding is not sufficient for tasks such as hyperparameter tuning, which require familiarity with how the algorithm works. This can lead to oversimplified or inaccurate mental models of AI/ML systems, and also create unrealistic expectations in students, leading to the so-called "Eliza effect", where a system appears much smarter than it is [13]. To this end, several pedagogical strategies have been experimented to support white-box understanding, such as an approach based on hands-on pen-and-paper tasks experimented with in an introductory DS course for psychology graduate students [119]. In addition, there is a significant body of research in the field of AI education, particularly focused on the teaching of ML in K-12 education [183]. This research highlights the need for a paradigm shift in computing education to effectively integrate ML models. [183] identifies key elements for successful integration, including fostering data-driven thinking, emphasising real-world applications, acknowledging the black-box nature of certain algorithms, taking an interdisciplinary approach with STEAM subjects, and introducing students to new NMs. Notably, these findings are also relevant for DS education.

From a learning theory perspective, constructivism and creative learning principles can be integrated into data pedagogy to promote active engagement, project-based learning, and critical thinking. For example, the study presented in [152] highlights a team-based, project-focused introductory course centered around big data. The outcomes of the study reveal positive results in terms of both learning outcomes and practical engagement with tasks mirroring those encountered in the industry. In the study by [123], a team-based learning strategy for database courses is introduced. The study showcases how this approach can yield enhancements in performance while effectively addressing the conceptual and collaborative needs of database education.

Finally, a limited number of studies have explored the integration of gamification into data education. While a significant portion of these studies primarily focus on introductory statistical

[1] https://teachablemachine.withgoogle.com

concepts [100], particularly in the context of online courses, a few experiments have extended their scope to domains beyond statistics. For example, a recent work [99] presents a data literacy experiment based on a data visualization learning application that offers different types of gamification (e.g., badges and challenges, use of avatars, social competition). The results of the study are encouraging, showing positive outcomes in terms of learning and engagement.

3.1.3 The Challenge of Interdisciplinarity

While interdisciplinarity is an intrinsic aspect of DS, it also poses challenges for the design of DS programs and courses. There is an ongoing debate about how to incorporate application domains into the learning content and whether it should be done. Furthermore, the optimal ratio between disciplines, as suggested by [43], remains an open question. For example, many DS programs have their roots in either CS or statistics departments, resulting in a curriculum that is heavily biased toward CS or statistics [4]. To address this, various approaches to integrating application domains have been proposed. For example, [10] developed a multidisciplinary DS curriculum involving faculty from different disciplines. In addition, students are required to earn credits in one of the proposed application domains as part of their degree.

Other researchers have explored how CS education can benefit from DS [117]. Indeed, incorporating real-world data can help CS students better understand real-world problems, while statistical reasoning can improve their problem-solving skills [73]. In addition, some researchers have experimented with integrating application domains into traditional courses. For example, [181] proposed incorporating biological data analysis into a CS1 course in which students learn programming skills using real laboratory data. In [11], an interdisciplinary CS1 course among business and CS students focusing on business domain problems is presented.

Finally, [77] proposes a comprehensive framework of five guiding principles[2] for effective teaching in introductory DS courses. This framework emphasises structuring courses around diverse case studies, incorporating computational elements, teaching abstraction, mimicking real-world scenarios, and promoting critical thinking and scepticism. This framework, with its emphasis on practical application and critical thinking skills, aims to produce confident, competent and analytical data scientists who are well prepared to navigate the complexities of the field.

3.2 Teaching Introductory Data Science

Data is now everywhere, and our lives and decisions depend on vast amounts of data. The latter has generated a lot of excitement in recent years, creating new business models, services and jobs. However, it has also raised a number of concerns, particularly about the ethical and political

[2] https://datasciencelabs.github.io

issues behind the algorithms used to mine data [143]. Therefore, digital skills and awareness to deal with data in personal and professional activities are now considered essential for every citizen [65]. These competences are commonly referred to as *data literacy*. The latter can be defined as the ability to understand and critically evaluate information derived from authentic data [88].

In this section, we explore the intricacies of teaching IDS in schools and universities, particularly to non-CS majors, by examining the motivations, challenges, and pedagogical approaches documented in the literature.

The findings from this research form the basis of the thesis, which presents a set of pathways and tools designed to facilitate IDS education for non-CS majors. These pathways and tools are tailored to both university students and high school students involved in careers guidance activities, ensuring that the benefits of IDS education reach a wide audience.

3.2.1 Teaching IDS in Schools: Motivations & Challenges

As mentioned above, as we live in the data age, it is fundamental that people have some basic data skills in order to be active digital citizens. Students should be aware of the challenges of data ethics and learn how to share, handle, communicate, and interpret data appropriately, regardless of their future careers. In addition, we believe that early exposure to DS topics in school can help to embed data culture in students and make them aware that DS has become ubiquitous due to its domain-specific nature. This is also reflected at university, where many non-computing academic programs (e.g. biology, business, medicine, etc.) have started to offer courses that cover data analytics topics, and students need to be aware of this. Furthermore, it is widely recognized that data thinking, like computational thinking, should be introduced to students as early as possible to spark their interest and create a potential future generation of data scientists, or at least data literate people.

These are the main motivations for offering data dissemination activities to students. If we consider the fact that it is difficult to find experienced DS teachers or support materials for DS education in (Italian) schools, the issue becomes even more important. Furthermore, DS can be a means to promote and improve competency-based education with an interdisciplinary approach. The use of real data can be exploited to design learning units -“Unità di apprendimento” (UdA) in Italian-, creating learning pathways that mix humanities, sciences, coding, and digital skills. In particular, authentic data can be relevant to students’ interests and attitudes, which can have a positive impact on their creativity and motivation to learn. In addition, real data sets have been shown to promote data thinking in learning environments designed to develop CT [120]. Therefore, CS education can also benefit from the potential of data by paving the way for new interdisciplinary learning units where students can integrate data literacy with other soft and hard skills.

On the other hand, early DS activities lead to a number of **challenges** that need to be considered by educators and researchers. Firstly, achieving the desired learning outcomes, as DS education often requires some prior knowledge of computing and statistics/mathematics. Coding and statistical models are important components of DS, and students typically need to learn a programming language and/or some statistical packages (e.g., R, Python, Pandas, NumPy, matplotlib, etc.). The domain-specific nature of DS is certainly an opportunity for learning, but could also be a challenge if not addressed appropriately: domain knowledge should be carefully assessed by choosing a dataset that requires a "right" level of knowledge, and with a view to an UdA, teachers from non-computing disciplines should contribute their domain knowledge. Time and curriculum constraints could be another important barrier to the dissemination of DS in schools. In our experience, activities have to be embedded in a very short time frame of 10/20 hours maximum, which leads to several challenges in terms of providing the right balance between learning content and hands-on practical activities. The latter is a major component of DS education, but often requires a lot of time, as non-computing students often struggle with the technical and coding aspects.

Finally, there are several other aspects that can affect the learning objectives of the activities. One example is the choice of the right level of data complexity, which depends on several characteristics, such as data quality: a fair amount of messiness in the data could create learning opportunities in exploring the data; on the other hand, too much messiness may require skills that are too advanced for students and thus lead to frustration and demotivation (e.g., missing values that may require different interpretations or treatments by students) [88]. Another example is the size of the dataset: a small amount of data can be handled with pen and paper, whereas a large amount of data requires a software tool.

3.2.2 Tailoring IDS for Diverse Majors

In recent years, IDS courses have proliferated in many disciplines. However, most are designed for CS majors and require prerequisites in programming and mathematics. For example, the ACM Data Science Task Force's Draft 2 of Data Science Curriculum Competencies[3] is aimed primarily at CS majors and focuses on knowledge domains such as artificial intelligence, big data systems, data acquisition, data management, and more [46]. This poses a challenge for students from non-CS backgrounds who may not meet these specific requirements [103]. The same situation applies to academic education, where IDS courses are typically advanced elective that require prior knowledge of programming, mathematics, and statistics. Our university is no exception. Nevertheless, for the upcoming academic year (2024/2025), there are plans to introduce an IDS course aimed at students enrolled in the Digital Humanities program.

Designing an IDS course for non programming literate students and without a solid mathematical

[3]The ACM Data Science Task Force Report: https://dstf.acm.org/DSTF_Final_Report.pdf

background faces challenges. The learning objectives of the course should be formulated so as not to assume prior knowledge of CS or mathematics. This factor plays a key role in the course design phase, which includes aspects ranging from defining the desired learning outcomes to curating appropriate learning materials, pedagogical methods, and supportive learning tools.

For example, choosing a white-box approach to teaching ML algorithms may not be the wisest choice, as students lack the basic mathematical skills to understand them. However, the course should equip students with skills that are relevant to the job market. Therefore, it is crucial that students not only understand the concepts, but are also able to perform simple tasks in groups, thus promoting a sense of autonomy in their learning journey.

3.2.3 Data-Centricity Pedagogy

A significant barrier to entry for many IDS courses is the requirement for programming skills. This requirement often excludes students from diverse academic backgrounds, limiting the accessibility and inclusivity of IDS education. However, the recent emergence of data-centricity pedagogy [95] offers a promising approach to addressing this challenge.

This approach emphasises rethinking the content of CS1 courses by incorporating a data-centered and interdisciplinary perspective. It can be defined as a convergence of DS and data structures, prioritising fundamental DS concepts as a springboard for the introduction of data structures [95]. This shift aims to broaden engagement and support for a wider range of students, beyond those traditionally interested in CS. By adopting this lens, students can see programming not just as a technical skill, but as a powerful tool for addressing relevant questions in areas of personal interest.

The data-centric approach has several distinct advantages. A key advantage is the use of real-world data presented in tabular form. Familiarity with tables is widespread, even among younger students, who encounter them in a variety of contexts, such as middle school mathematics and spreadsheet applications. In addition, real-world data promotes alignment between course content and students' personal interests. However, implementing the data-centricity approach is not without its challenges. These challenges include strategically sequencing topics to ensure logical progression, effectively integrating elements of statistics and programming into the curriculum, and selecting appropriate programming methods that align with learning objectives.

Despite these challenges, data-centricity has been successfully applied in several courses and programmes [60]. For example, Brown University offers the course "CSCI 0111"[4], a data-centric course using the Pyret programming language[5] [60]. Similarly, the University of California, Berkeley, offers "Data 8"[6], a course that uses the Python programming language along with the

[4] https://cs.brown.edu/courses/csci0111/
[5] https://pyret.org
[6] https://data.berkeley.edu/education/courses/data-8

"datascience"[7] educational package.

3.2.4 Related Work

Different approaches to IDS education have been explored to cater for the diverse backgrounds and interests of students. One example is the "Data 8: Foundations of Data Science" course at the University of California, Berkeley[8]. This eight-week course is based on Python and uses the "datascience" educational library developed at Berkeley[9]. The "datascience" Python package provides basic data structures for working with tables and maps. The former allows easy manipulation of tabular data, while the latter introduces geographic data. In addition, the library provides complementary utility functions and predicates that facilitate intuitive data retrieval (e.g., `table.where(condition)`). The course begins with an introduction to DS and programming basics, covering variables, type casting, and data types such as integers, floats, strings, and np.arange. This basic programming content is taught over the course of approximately two weeks. The course then moves on to data transformation operations, data visualisation techniques, and statistical analysis topics, including linear regression, classification, confidence intervals, and statistical tests.

Another example is an IDS course for non-CS majors presented in [28]. This course uses the interactive web platform DSLP, designed by the course instructors and inspired by visual programming languages, to teach the high-level workflow of data processing and analysis without requiring coding. It covers topics such as data ethics and feature engineering, and allows students to work on real data through in-class demos and hands-on case studies.

Several researchers advocate introducing students, especially non-CS majors, to programming through JavaScript in conjunction with the D3.js library [85]. This approach seems to encourage interdisciplinarity, as students hone their data thinking skills by programming interactive and data-driven visualisations.

Many researchers suggest using the statistical software R to introduce statistics, DS or AI at both secondary and university level [133, 25, 166]. Some examples include an IDS course for non-majors in a flipped classroom format [32], an undergraduate DS course focusing on the concept of the DS life cycle [200], and a team-based, project-oriented IDS course aimed at students from diverse backgrounds [152]. Both courses cover various programming concepts such as data structures (e.g., lists, vectors, arrays and matrices, data frames), structured programming constructs (e.g., loops, conditional statements, and functions), data and text manipulation (including regular expressions), and file I/O (including Excel spreadsheets). Two other relevant

[7] https://github.com/data-8/datascience
[8] Course "Data 8" https://data.berkeley.edu/education/courses/data-8
[9] Datascience Library, Berkeley: https://github.com/data-8/datascience

examples come from the statistics education community: an IDS course[10] [77] with a focus on computation using the R programming language and R Markdown for report generation, and "Passion-driven statistics"[11] [174], a course-based undergraduate research experience (CURE) based on data-driven projects with a focus on statistical concepts, used in several US colleges and universities. The course provides material for implementing programs in various programming languages such as Python, R, SAS, STATA, and SPSS. The approach has shown positive results in terms of motivation and interest in DS topics, with high participation of female and minority students.

There are also examples coming from CS education, in particular CS1 courses with a data-centric pedagogy. For example, the "CSCI 0111" course offered at Brown University[12], based on the Pyret programming language[13] [60]. The latter is a programming language specifically designed to serve as an introductory language for CS education with strong static typing, Python-inspired syntax, a focus on functional programming concepts, strong support for testing, and native support for class definition, image, map and tabular data. Pyret is the introductory language used in the textbook "A Data-Centric Introduction to Computing" [61], aimed at both secondary and tertiary education. Other examples include an introductory data-centric computing course at Boston University for non-CS majors, covering relational databases, Python programming, and data mining [178]; an interdisciplinary CS1 course for social science majors focusing on the domain knowledge of psychology [118]; and an introductory programming course for CS and non-CS students [28], using the Python programming language and the Grock[14] online learning platform, which allows students to work with textual and numerical data and receive automatic formative feedback. This programme starts with an introduction to the standard Python library, file I/O, iteration, and conditionals to handle tabular data. It then moves on to Python libraries such as Pandas, Matplotlib, and Scikit-learn, focusing on the Dataframe data structure. The course serves as a springboard to a traditional CS1 course.

These approaches aim to introduce students with different backgrounds and interests to the world of data. All courses have a strong practical component, with a focus on real-world projects and teamwork, placing students at the interface between academic and industrial contexts [152].

Other examples come from schools. Although data education is still in its infancy and rarely integrated into school curricula [190], recent years have seen a surge in proposed teaching units using programming languages and statistical software tools. Bootstrap: Data Science [155] is a curriculum designed for use in integrated contexts at the secondary-school level, based on Pyret programming language. Other examples include IDS[15], a year-long course for high-school students using the R programming language for statistical computing; a unit based on survey data

[10] https://datasciencelabs.github.io
[11] https://passiondrivenstatistics.wescreates.wesleyan.edu
[12] CSCI 0111 course: https://cs.brown.edu/courses/csci0111/
[13] Pyret Programming Language: https://pyret.org
[14] Grock Learning: https://groklearning.com
[15] https://www.idsucla.org

using the CODAP[16] online tool [63]; School 21[17] using Tableau; the Tableau's official data literacy courses for schools and academies[18]; and finally, an innovative curriculum where secondary school students learn about big data and its social impact using real multivariate data [64].

3.3 Database Education

Database education is a specific area of data systems education that focuses on databases. It includes topics such as data modelling and query languages. Databases are a fundamental part of CS and DS education. Within this area, researchers dedicated to database education have placed particular emphasis on identifying common SQL misconceptions [115]. Although this dissertation focuses primarily on the area of logical database design, as opposed to the specific query language, this section provides a brief overview of the current state of database education, including the identification of SQL misconceptions, the use of visual query tools, and the introduction of NoSQL in database courses.

3.3.1 SQL Misconceptions

One of the main sources of misconceptions seems to be students' prior knowledge [115, 168]. In SQL education, prior understanding of mathematics, set theory, relational algebra and programming can influence learning through knowledge transfer [115]. For example, the presence of keywords borrowed from natural language in SQL makes it susceptible to linguistic transfer [57], and prior experience with programming languages can leads to misconceptions due to differences in notation and usage [156].

In the research paper [115], numerous misconceptions and errors related to SQL are documented. The study underscores that students often confuse the == and = operators in SQL queries, despite the fact that the == comparison operator is not part of the SQL syntax. This misconception arises from different notations of other programming languages. Other common misconceptions are related to the scope of different query elements. For example, students may define a view using a query with a `WITH` clause, but fail to invoke the view in the main query for proper usage. Similarly, students may mistakenly use a self-join query to retrieve pairs of records from the same table, not realising that the table should be referenced twice in the query. These misconceptions reflect an incorrect or incomplete mental model in which students may view the database management system as a black-box. Indeed, the literature predominantly emphasizes subqueries, `GROUP BY` clauses, and `JOIN` clauses as the main areas where misconceptions are prominently observed [23, 115, 180].

[16] https://codap.concord.org

[17] https://www.tableau.com/community/blog/2021/8/tableau-brings-data-literacy-classroom-school-21

[18] https://www.tableau.com/it-it/learn/data-literacy

Other common misconceptions include the belief that the `DISTINCT` clause selects the first element in a list, inappropriate uses of primary keys, and various syntactic errors, such as remembering only the main part of a keyword or using synonyms interchangeably [139, 115].

Finally, several visualisation tools have been developed to illustrate the underlying notional machine of SQL. For example, SQL Tutor[19], which offers a step-by-step visualization of how a physical query plan is executed, presented in a hierarchical structure. An example of the SQL query visualisation is shown in figure 3.3. Other notable examples include QueryVis [101] and

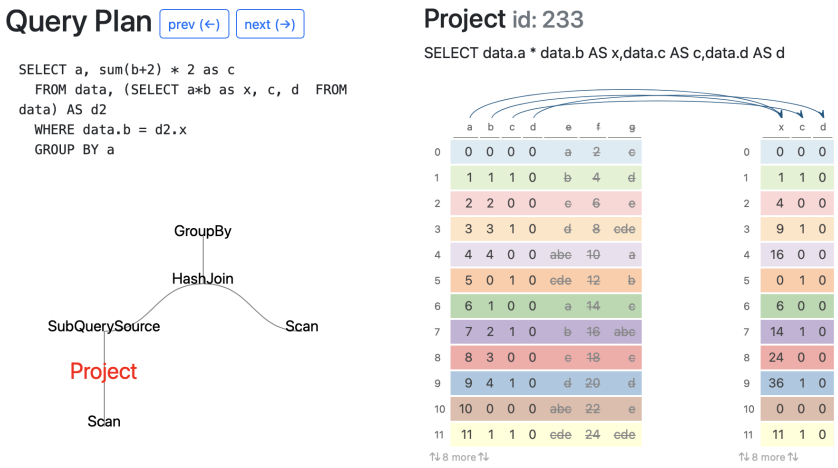

Figure 3.3: SQL Tutor

SQLVis [20] [116]. The latter visualises the SQL query plan and schema at a higher level of abstraction, creating a graph-based visualisation of SQL queries in Jupyter notebooks. A glimpse of SQLVis in action can be seen in figure 3.4.

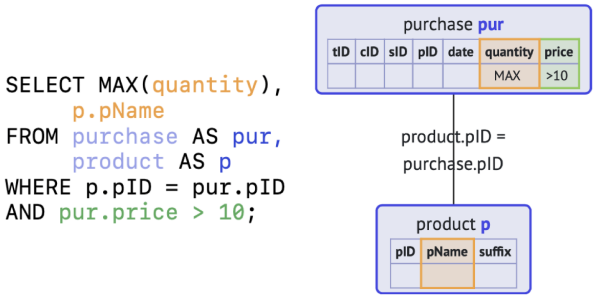

Figure 3.4: SQLVis

[19] https://cudbg.github.io/sqltutor/
[20] https://github.com/Giraphne/sqlvis

3.3.2 NoSQL Education

The term NoSQL has evolved to encompass many different types of database. However, its current usage often emphasises the concept of "Not Only SQL", referring to databases that depart from the traditional relational model based on normalised tables. Several key characteristics distinguish NoSQL databases, such as different data models (e.g. graph, tree, key-value, document), schema flexibility, implicit support for horizontal scaling, and, typically, eventual consistency in contrast to the strong consistency of relational databases [124].

Research has investigated specific mental models and misconceptions that students hold about NoSQL databases, with a particular focus on those related to document-oriented and graph-based models. Although the literature on NoSQL misconceptions is not as extensive as that for SQL, recent studies have shed light on the challenges students face when performing aggregation and union operations in NoSQL database environments.

These findings are consistent with observations in SQL education, suggesting that certain concepts and operations pose similar hurdles across different data management paradigms. For example, students working with MongoDB often face challenges when dealing with concepts such as `GROUP BY` and related subqueries [5]. Similarly, in Cypher, the query language for Neo4j, students struggle with the use of the `WITH` clause, which is used to concatenate the graph results of previous queries into subsequent queries [39]. Furthermore, studies have observed a higher frequency of result set errors in MongoDB compared to graph databases. This discrepancy may be attributed to the greater cognitive effort required to construct an effective mental model for document-based data structures, which can lead to errors in formulating queries that retrieve the desired results [6].

With respect to data modelling, there have been numerous proposals for design methodologies for NoSQL systems: [38] proposes a general workload-driven method for designing key-value, wide-column and document NoSQL database schemas; [12] presents the "NoSQL Abstract Model" - a system-independent representation of application data for NoSQL design. Other examples include: a model-driven approach that generates NoSQL database implementations from a high-level conceptual model [48], a design methodology for transforming entity-relationship models into document-oriented logical models suitable for MongoDB [49], and a similar approach is proposed for Cassandra in [37].

Finally, the inclusion of NoSQL databases in traditional database courses has become a topic of discussion among academics worldwide, reflecting the growing importance of these technologies in industry [199]. This trend is illustrated by several case studies: [153] presents a semester-long web-based project using both SQL (Oracle) and NoSQL (MongoDB) for an undergraduate IDB course; another learning model specifically designed to introduce NoSQL paradigms to undergraduate students is presented in [124]; other examples include the integration of the CouchDB[21]

[21] https://couchdb.apache.org

database management system as an introduction to non-relational databases [62] and the integration of basic NoSQL database commands using Neo4j in vocational training programmes [92].

Part II

Innovating Concurrency and Data Education

Chapter 4

Concurrency Education with Sonic Pi: "Hear" and "Play" Mistakes and Misconceptions in Multithreaded Programs[1]

Concurrency is a complex to learn topic that is becoming more and more relevant, such that many undergraduate CS curricula are introducing it in introductory programming courses. This chapter investigates the combined use of Sonic Pi and Team-Based Learning to mitigate the difficulties in early exposure to concurrency [175]. Sonic Pi, a domain-specific music language, provides great support for "playing" with concurrency, and "hearing" common problems such as data races and lack of synchronization among different concurrent threads. More specifically, the chapter focuses on students' misconceptions regarding concurrency in Sonic Pi and compares them to those arising in traditional concurrent programming languages. In addition, it preliminary explores knowledge transfer from Sonic Pi to C/C++.

4.1 Introduction

Background and Motivations Concurrent programming is a technique in which two or more executions threads start, run in parallel or in an interleaved fashion through context switching, and complete in an overlapping time period by managing access to shared resources. Multithreading can be compared to a pianist playing a piano with both hands. Instructions run in parallel on a single processing unit. The threads may work on different instructions (such as the

[1]This chapter is based on the material published in [185, 51, 52].

pianist playing two separate musical pieces), but work with the same resources and processing power. Concurrent programs can execute multiple tasks at once, e.g., writing a file on disk while appending new items. Concurrency, however, introduces nondeterminism: the exact order in which tasks are scheduled is not known in advance. Furthermore, the interleaving of instructions of different execution threads may yield an exponential number of possible schedules. It is often the case that some schedules will lead to correct outcomes and others will not. Thus, it becomes necessary for programmers to express constraints to prevent the system from allowing schedules that yield incorrect outcomes.

In the past, due to its complexity, concurrency was typically introduced at advanced levels of computer science (CS) curricula. However, due to its growing importance, e.g., event-driven programming and multi-core computing, several universities are now rethinking their approach and starting to introduce concurrency in introductory courses, in some cases adopting simplified concurrency models [175, 58]. Despite this, students often struggle to understand concurrency, which can be particularly challenging when introduced early in the curriculum. Consequently, there is a need for research aimed at effectively introducing concurrency and supporting students in building and refining their viable mental models.

In this chapter we investigate the use of Sonic Pi to mitigate the difficulties in early exposure to concurrency. Sonic Pi [1] is a domain-specific programming language and code-based music tool where musical concepts, and specifically those related to multi-threading, are aligned with programming ideas. The Sonic Pi programming tasks have been used within a collaborative framework based on the Team-Based Learning (TBL) pedagogy [186]. They focused on code comprehension and code composition and targeted common misconceptions about concurrent programming.

The approach, based on an interdisciplinary pedagogy for early exposure to concurrency by combining music and programming activities, has been applied to two teaching experiments for undergraduate CS students, more precisely 130 first-year students in the context of an annual first-year course that focuses on teaching students the fundamentals of computer architecture design for modern microprocessors and 54 third-year students in the context of a third-year, second-semester course that focuses specifically on the design and analysis of algorithms for concurrent and distributed systems.

Research Questions Based on previous research on concurrency education in traditional programming languages, our research questions are:

- Does the use of Sonic Pi help the students face common misconceptions and mistakes in concurrent programming? Do misconceptions and mistakes in Sonic Pi correspond to misconceptions and mistakes in traditional concurrent programming languages?
- Can we apply knowledge transfer from Sonic Pi to more traditional programming lan-

guages such as C and C++? What is the effect in terms of learning outcomes?

Contributions The contribution of this chapter is twofold. Firstly, it presents an introductory approach to concurrency based on Sonic Pi, aimed at undergraduate students, and shares all the detailed information and materials so that other instructors can replicate the activities. This teaching format is based on our extensive research and classroom experimentation over the past three years, which has provided a solid foundation. Secondly, the chapter explores students' misconceptions about concurrency that emerged during the experiments. By collecting and analyzing team responses and deliverables, we delve into students' problem-solving strategies, comparing them to experiences with those based on traditional programming languages.

Design The choice of the Sonic Pi programming language is due to several reasons. As mentioned in [2], *"Sonic Pi was designed to teach a large number of computing concepts covered in the new UK computing curriculum introduced in September 2014. Examples of these concepts are conditionals, iteration, variables, functions, algorithms, and data structures. We also extend beyond these to provide educators with an opportunity to introduce concepts that we believe will play an increasingly important role in future programming contexts such as multi-threading and hot-swapping of code."* Sonic Pi offers distinct advantages over traditional computing methods to facilitate the early introduction of multi-threading. The inherent concurrency of music creation (e.g., several instruments that synchronize on a rhythm and melody) and the simplicity of the tool can provide a natural and immersive learning experience for concurrency. In particular, live coding and auditory output, two key features of Sonic Pi, provide great support for "playing" with concurrency, and "hearing" data races and other common mistakes. To our knowledge, this study represents the first attempt to use Sonic Pi for these learning goals at the university level.

The proposed teaching format is based on our previous research and experiments started in the academic year 2019/2020. We started by experimenting with the use of Sonic Pi with TBL to increase learning motivation and curiosity among freshmen by introducing them to advanced topics [186].

Building on this experience, we further refined and adapted the teaching format in the academic year 2021/2022 for third year students enrolled in the Concurrent Programming and Distributed Algorithms (CP[2]) and Informatics for Creativity, Didactics and Dissemination (ICDD) courses. These courses provided an ideal platform to explore concurrency and its application in creative contexts. In these experiments, the primary learning objective was to introduce students to the concept of multi-threading in Python. However, we approached this goal by starting with concurrency in Sonic Pi and then moving to Python using the `python-sonic` library. This interface allowed students to combine the multi-threading capabilities of Python with the functionality of Sonic Pi within a Python environment. For the students enrolled in the ICDD course, the ex-

[2]Programmazione Concorrente e Algoritmi Distribuiti in Italian.

periment served an additional purpose. It provided an opportunity to examine the pedagogical objectives and didactic concepts of the experiment itself.

The two units presented in this thesis, aimed at third year CPand first year Computer Architecture (CA) students, and experimented during the academic year 2022/23, are based on our previous experience. They represent the culmination of our efforts to introduce concurrency in a creative way and to address misconceptions in concurrency education. More specifically, we focused on the goal of introducing students to the concept of concurrency through a series of short practical tasks, each designed to address specific misconceptions about concurrency and basic programming concepts, allowing students to gradually build their understanding while addressing specific misconceptions, including variable scope, function calls, parameter passing, race conditions, correctness and synchronisation goals. Some tasks focused on program comprehension, while others involved code writing. Program comprehension tasks [84] are a cognitive approach to teaching programming, where the learner interacts with an artifact representing the program, for example a piece of code. See 2.4.4 for more details.

The program comprehension and composition tasks in Sonic Pi have been designed taking into account common misconceptions observed both in our previous teaching experience and in the literature on concurrency education, in particular, those arising from the mixing of concepts from concurrency and the underlying computational models [175], which seem to be particularly challenging for students.

Experimental Evaluation To assess the effectiveness of the activity in terms of engagement and appreciation we considered students' overall perceptions through a post-questionnaire; to evaluate the effectiveness in terms of learning outcomes, we considered TBL data, in particular team responses to the TBL tasks.

In addition, we wanted to gather preliminary empirical evidence on the transferability of the concurrent programming knowledge, skills, and abilities that students acquire in our Sonic Pi-based learning environment to other traditional programming languages, such as C. Knowledge transfer is crucial for effective learning, but it can pose challenges and lead to misconceptions for several reasons, such as differences in the underlying notional machines (NM) of the programming languages considered. To investigate knowledge transfer, we designed a written test consisting of three exercises in either C or pseudocode, focusing on concurrency aspects covered in the music code-based domain. The test was administered one week after the experiment.

Plan of the Chapter Section 4.2 discusses related work while Section 4.3 introduces some preliminary notions about Sonic Pi and TBL. Section 4.4, 4.6 and 4.7 provide all the details of the design of the TBL units (RAT Quiz and Team app). In Section 4.8 we introduce the teaching experiments and we present an aggregate analysis of data collected via quizzes, tests, and questionnaires. Section 4.9 discusses findings and implications. In Section 4.10 we address

some conclusions and future work.

4.2 Related Work

This section presents related work, by surveying approaches combining music and coding, approaches for teaching concurrency, focusing on the most common misconceptions related to concurrency, and finally introducing knowledge transfer.

4.2.1 Music and Coding in CS Education

Live coding involves the real-time manipulation of running programs to generate live auditory or visual effects, with the coding process becoming an integral part of the performance [20]. From a pedagogical perspective, live coding provides immediate feedback and enhanced interactivity, making it a valuable tool for teaching computer programming. In particular, live music coding allows abstract programming concepts to be presented in a more natural way through music.

In recent years, live music coding has gained prominence in interdisciplinary contexts, particularly in the field of STEAM education [27]. This approach has proven successful in teaching programming in both introductory and advanced programming courses [69]. Furthermore, the integration of music and CS has been found to enhance students' appreciation, motivation, and engagement in the learning process [74]. The author of [76] highlights the effectiveness of domain-specific programming platforms in promoting intrinsic motivation and positive attitudes towards learning CS. In [69], the author presents an approach to teaching design patterns and other programming topics (data structures, grammars, parsing, and formal proofs) using a music composition project. A summer camp is presented in [109], designed with a "Code Beats" approach, where students learn fundamental programming concepts to make music using a domain-specific tool that provides immediate feedback and hints for learning. The experiment seemed to increase student interest, motivation and engagement. Finally, in [150] Scratch music coding capabilities are used to teach basic programming concepts. There are a number of educational code-based music creation tools, such as JythonMusic[3], EarSketch[4], TunePad[5] and Sonic Pi[6]. Sonic Pi, developed in 2012 with the UK's new primary and secondary computing curriculum in mind, has proved effective in introducing live music coding to primary schools [1]. It also appears to be effective in promoting positive attitudes towards programming among middle school students [136]. Furthermore, the use of multimedia computer contexts in teaching introductory

[3]https://jythonmusic.me/
[4]https://earsketch.gatech.edu/landing/#/
[5]https://tunepad.live/
[6]https://sonic-pi.net

programming across educational levels has shown positive effects on pass and retention rates [165].

4.2.2 Concurrency Education

In the field of concurrency education, several studies have focused on analyzing programming errors made by students. One study examined errors in specific concurrent assignments, while another conducted a phenomenographic study to understand how students develop concurrent programs [107]. Additionally, the analysis of final written exams from concurrent and operating systems courses aimed to identify common misconceptions among students [176]. Finally, empirical research at the secondary school level has investigated students' perspectives on the correctness of concurrent programs [89, 18].

Concurrency poses challenges as it involves non-determinism, making it difficult to test program correctness by running them. Unlike sequential programming, concurrency requires a more formal and reasoned approach, demanding students to develop viable mental models of concurrent program execution. In addition, students are expected to possess a solid understanding of fundamental concepts like variable scope, parameter passing, aliasing, references, and pointers, which are considered *threshold concepts*—challenging concepts that require significant development or revision of specific areas of the mental model. Mastery of these concepts enables students to recognize connections with other programming concepts [22, 175].

Given the complexity involved, concurrent programming has traditionally been covered in advanced courses that require strong programming skills, such as operating systems courses or other advanced electives. However, the increasing importance of concurrency, as discussed in the previous section, has prompted the CS education community to address concurrency in introductory courses. This has led to a reevaluation of curricula and the emergence of new pedagogical approaches [175].

Over the past few decades, several tools have been developed to aid students in learning about concurrency. Eludicate [81] and Atropos [108] are examples of tools that allow students to capture and visualize concurrent object-oriented execution. However, these tools lack active interaction with the visualization, which diminishes engagement and learning impact, as argued by Sorva [173]. To address this limitation, other tools have been introduced, such as The Deadlock Empire[7], an online educational game specifically designed for learning concurrency. Progvis [177] is another tool that enables visualization of the interaction between concurrency and fundamental programming concepts like scope, parameter passing, and references.

In addition to these tools, older examples include Linda [31] and Multi-Logo [144]. Linda is a coordination language that supports concurrency, and Multi-Logo is a concurrent extension of

[7] https://github.com/deadlockempire/deadlockempire.github.io

Logo, which was experimented in primary schools where students had to control simple robotic devices constructed from LEGO® bricks.

In principle, formal concurrency models may be considered as possible NMs to be used to reason on concurrent executions abstracting away from the syntax of a particular programming language. Process calculi such as CSP [79] and CCS [122] and graphical concurrency models such as Petri nets [135], are well-known abstract models used to represent concurrent computations as mathematical objects and to reason about their properties [98].

For instance, in [17], Mordechai Ben-Ari adopts a logic-based approach based on automata, temporal properties and model checking to reason about properties such as mutual exclusion, starvation, and deadlock interleaved with practical examples of concurrent programming libraries in Java and Ada.

4.2.3 Common Misconceptions in Concurrent Programming

Misconceptions in concurrent programming can often lead to subtle and hard-to-debug issues. In this section, we consider misconceptions related to basic and advanced concepts in concurrent programming.

Many students struggle with synchronization problems, and part of this may lie in their understanding of the prerequisites that are crucial to concurrent programming. In fact, students may struggle not only with concurrency concepts but also with threshold concepts about the underlying computational model.

As discussed, e.g., in [175, 176], students often struggle with the scope of variables, mistakenly believing that local variables are shared between threads or concurrent function calls. This misconception stems from a failure to distinguish between different instances of function calls and to recognize that each instance has its own set of local variables.

Other misconceptions include the belief in global locks as protectors of code rather than protectors of specific resources, which stems from a non-viable mental model that focuses on code rather than data. Additional common misconceptions concern correctness [90], believing that a concurrent program only needs to work most of the time rather than all of the time. They may rely on trial and error for debugging, ignoring the concept of interleaving, which is difficult for many students to grasp. It has also been found that many students experience the program in terms of the program text rather than its dynamic execution [176].

In addition, students find it difficult to identify concurrency problems and formulate synchronization goals [19]. However, once synchronization goals are identified, they generally succeed in implementing solutions that meet those goals. However, in one study [176], students struggled to associate shared data with appropriate synchronization primitives, such as using multiple locks to protect the same instance in different situations.

Students often believe that using synchronization primitives, such as locks or semaphores, ensures thread safety [75]. Students need to understand the importance of correct synchronization placement, avoiding race conditions, and understanding the semantics of different synchronization mechanisms.

Another misconception relates to memory models, where students often assume sequential consistency [75]. This misconception may stem from concurrency courses that focus on context switching on a single CPU and neglect other scenarios involving weaker memory models.

A further common misconception about concurrency, unrelated to the basic concepts, is that adding concurrency to a program will automatically improve performance [179]. In fact, concurrency introduces additional complexity, so it is important for students to understand that performance improvement depends on several factors, such as the nature of the problem, the hardware, and the efficiency of the concurrency design.

Lastly, although it is not necessary to know object-oriented programming (OOP) to work with concurrency, it was observed that a non-viable mental model of OOP can lead to misconceptions in identifying shared data and synchronization mechanisms since some OOP concepts are also relevant to concurrency (e.g. confusing class and instance, which is similar to confusing instances of a function call, values, and references, etc.).

Finally, it is important to note that these misconceptions may vary according to programming language, individual experience, and educational context. Nevertheless, investigating these misconceptions can help to better know student understanding and improve future concurrency education.

4.2.4 Knowledge Transfer

Knowledge transfer refers to *the application of skills (or knowledge, strategies, approaches, or habits) learned in one context to a novel context* [8]. In particular, we refer to the knowledge domain transfer [15], which is the process of applying previously acquired knowledge and skills from one context to another (in the same or a different discipline). In our case, concurrent programming concepts and concurrent problem-solving strategies facilitate the learning and adoption of concurrency in another programming language. However, transferring knowledge from a language such as Sonic Pi to more traditional textual languages such as C and C++ can be challenging due to differences in their NMs. We are not currently aware of any existing research that explores this specific scenario, and further research is needed to understand the effectiveness and challenges of knowledge transfer in this learning context.

Extensive research into the transition from block-based to text-based languages has highlighted differences in syntax, mental models, misconceptions, program comprehension, and learning outcomes [94]. However, current research seems to point in the direction of a positive transfer

between block-based and text-based languages [192].

Learning a second programming language can be challenging [68], even for experienced developers, because of the need to adapt mental models to new language features [163]. However, in some cases, programmers moving from one language to another, such as from C# to Ruby, chose to start from scratch, ignoring prior knowledge, which mitigated the effects of cross-language interference [163].

4.3 Preliminaries

Our proposal relies on a combination of Sonic Pi and Team-Based Learning (TBL) as the teaching format. The resulting learning experience combines hands-on programming activities with collaborative learning. In this section, we focus on the preliminary notions at the basis of the proposed approach, namely, Sonic Pi and Team-Based Learning.

4.3.1 Sonic Pi and Concurrency

Sonic Pi is a domain-specific language for manipulating synthesizers through time. It can also be viewed as a code-based musical creation and performance tool, where each musical concept corresponds to a programming idea. The music domain allows for a pedagogy that focuses on the problem (concurrency) rather than the programming language, proposing real-world examples that are inherently multithreaded. From this perspective, Sonic Pi seems to be naturally constructionist[8] (the *"building material"* [130] to learn concurrency), as Papert's computational thinking stresses the importance of the computer as a powerful meta-tool for *"making the abstract concrete"* [106].

From a technical point of view, Sonic Pi is based on the Ruby programming language, inheriting its simple syntax. It offers an intuitive and user-friendly integrated development environment (IDE) for creating music. Furthermore, Sonic Pi provides auditory output feedback, allowing users to hear the sound generated by their programs in real time. This audio feedback, combined with textual program output and compiler error/warning messages, offers immediate and tangible results, enhancing the learning experience. These characteristics make Sonic Pi a language with a steep learning curve, extremely effective in adapting to different learning goals and environments, and in tailoring topics to students' interests and attitudes [1].

Sonic Pi is officially distributed for Microsoft Windows, Mac OS X and Raspberry Pi OS (but there are also unofficial distributions for Linux). Compared to other live coding languages, it is very easy to install, as it provides an all-in-one installer and does not require a separate in-

[8] See 2.2.1 for a detailed explanation of constructionism.

terpreter. Once inside Sonic Pi, there is no need to route audio to channels: just the command "play" followed by a note number is enough to create the sound. Sonic Pi does not use the usual pitch classifications found in live coding, such as the frequency swing in SuperCollider, but assigns a number to each key of a standard piano [70]. Sonic Pi takes advantage of the speed of

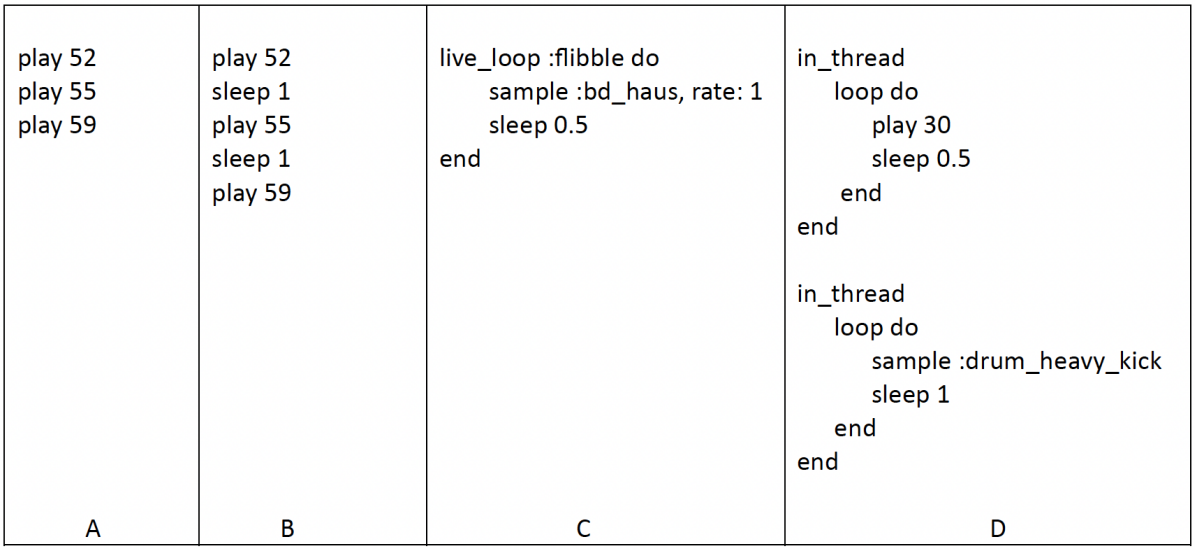

Figure 4.1: Some examples of Sonic Pi programs and instructions.

modern processors in assuming that a sequence of instructions, as those depicted in Fig. 4.1 (A), are likely to be executed so quickly in succession that they will be perceived as a chord, and not as an arpeggio. An arpeggio form can be achieved by "sleeping" the current thread for a number of seconds as in Fig. 4.1 (B). The notion of sleep is similar to that of the standard POSIX sleep operation that suspends execution for the specified time. Sonic Pi 2.0 has introduced special semantics that avoid drifting due to delays (thread scheduling and invocation of the POSIX sleep operation) [2].

Finally, Sonic Pi supports various APIs, allowing users to interact with other platforms and expand its capabilities, such as the `python-sonic` interface.

Due to these factors, Sonic Pi is an ideal tool for teaching both the fundamentals of computer programming, such as iterations, selections, functions, and data structures, as well as advanced concepts like concurrency and scope rules. One of the unique advantages of Sonic Pi is its ability to synchronize and audibly perceive different musical instruments, allowing learners to "feel" local and global objects in action.

The language offers several interesting concurrent features, such as the `in_thread` and `live_loops` control constructs, which are interpreted across multiple concurrent threads. The

`live_loop` construct is the key to mastering live coding with Sonic Pi. For instance, consider the program in Fig. 4.1 (C). Here we create a bass drum beating by repeating the sample `:bd_haus` forever. Thanks to hot-swapping code live loops can be redefined on-the-fly while still running. The `in_thread` construct resembles traditional thread-spawning operations in languages such as C. For instance, consider the program in Fig. 4.1 (D). Here the MIDI note 30 is played at the same time as the sample `:drum_heavy_kick` with half a second between each onset.

These programming abstractions provide an intuitive introduction to concurrency, even for novice programmers who are guided by sound and perception. The resulting programs can therefore be validated with a sense of rhythm and melody.

Sonic Pi provides the `cue` and `synch` functions to create synchronized music patterns between threads and/or live loops and avoid drifting effects. The cue sync mechanism is very similar to the `notifyAll()` and `wait()` methods of the Condition objects of other concurrent programming languages, such as Python and Java. Finally, Sonic Pi uses a lock-based synchronization mechanism, providing the `get` and `set` functions to prevent race conditions. It also utilizes a global memory store called *Time State*, where threads and live loops can share data.

Overall, Sonic Pi's combination of auditory feedback, intuitive concurrency features, and synchronization mechanisms makes it an effective tool for introducing concurrency and other programming concepts.

4.3.2 Team-Based Learning (TBL)

To support a collaborative learning environment, we used the TBL teaching methodology. This approach encourages collaboration between team members, and healthy competition between teams, and enriches the learning environment with gamification elements. All these characteristics made TBL consistent with Papert's constructionist idea of the importance of students' social and effective involvement in the construction of a computational artifact ([106]). Collaborative learning has been widely explored in CS education and there is extensive literature on its benefits. In particular, TBL has been shown to be effective in preventing student dropout and improving exam pass rates in CS1 courses [165]. In addition, a qualitative study has shown that TBL is highly rewarding and engaging for students enrolled in CS1 courses [87].

TBL is a strategy that enables students to follow a structured process to improve their engagement and the quality of their learning. It consists of modules that can be taught in a three-step cycle: pre-class preparation, in-class Readiness Assurance Process (RAP), and application-focused exercise (Team APP). More specifically, TBL has five essential components plus an optional peer evaluation phase. It begins with individual study outside the classroom, followed by a multiple-choice test (RAT), first individually and then in teams who must agree on answers. The RAP phase ends with immediate feedback (usually through scratch cards, which add a playful com-

ponent to the learning), a possible team appeal and a class discussion with the instructors. This is followed by the team application (APP), which is an open-book task where each team works on the same exercise and has to give an answer at the same time. Finally, the teams discuss and compare their solutions in plenary. This phase can be done with different discussion techniques, such as the *gallery walk*, where each team presents its solution in a kind of poster session.

Technically, we designed RAT quizzes on the course Moodle page. Immediate feedback was given at the end of the team quiz. In addition, we proposed a digital gallery walk using Padlet[9], an online bulletin board tool that allows team solutions to be shared with the whole class.

4.4 Learning Content & Material

The design focused on the goal of introducing students to the concept of concurrency through a series of short practical tasks, each designed to address specific misconceptions about concurrency and basic programming concepts, allowing students to gradually build their understanding while addressing specific misconceptions, including variable scope, function calls, parameter passing, race conditions, correctness, and synchronization goals. Some tasks focused on program comprehension, while others involved code writing. Program comprehension tasks [84], as discussed in 2.4.4, are a cognitive approach to teaching programming, where the learner interacts with an artifact representing the program, for example, a piece of code. Through this interaction, the learner is stimulated to build and refine their viable mental model of the underlying NM.

In what follows, we outline the learning content we developed for the TBL units. Specifically, we first present the preparation material in Section 4.5, then the RAT quizzes in Section 4.6 and the team application tasks in Section 4.7.

4.5 Pre-Class Preparation

To ensure adequate preparation and familiarisation with the TBL methodology and the Sonic Pi language, we introduced students to these topics one week prior to the in-class activity. We assigned the following pre-class materials:

- Sonic Pi Tutorial[10] (a fully integrated tutorial that provides a comprehensive introduction to Sonic Pi, assuming no prior knowledge of coding or music).
- Multimedia footage of live coding performances, available on the Sonic Pi website.

[9] https://it.padlet.com
[10] https://sonic-pi.net/tutorial.html

- Slides explaining the TBL methodology.

All preparatory materials were uploaded to the course Moodle page for easy student access. We estimated that students would need about 2-3 hours of individual study to complete the preparation before the class sessions.

4.6 RAT Quiz

We designed an identical RAT quiz for both activities, consisting of five multiple-choice questions that assessed participants' understanding of Sonic Pi language syntax and concurrency concepts. Following the principles of TBL, the quiz was first completed individually and then retaken in teams. The results of these individual and team RATs were collected for information purposes only. As they served as a preparatory tool for the subsequent team application phase and were completely independent of the research question, they were not statistically validated or included in the data analysis. Nevertheless, the results suggest a general trend towards improved performance in teams compared to individual attempts.

The five questions of the Rat quiz are discussed in the rest of the section.

4.6.1 Quiz 1: Hear thread creation

Consider the following program:
 	
`live_loop :foo do`
`  play 60`
`  sleep 1`
`end`
	When the program runs, you hear a basic beep every beat. Change the note 60 to 65 in the editor without stopping the program. What happens?
	
	• It raises a RunTime error.
	• It forks the main thread and creates two new ones. One will play 60 and the other 65. This is live coding.
	• No effect on execution. This is live coding.
	• It changed automatically without missing a beat. This is live coding.

In Quiz 1, we highlight a distinguishing feature of Sonic Pi, i.e., the dynamic management of code updates, a key point for live coding sessions. Among the multiple choices, only the last answer is correct. Indeed hot code-swapping ensures that the behavior of the thread is automatically updated. This feature is at the basis of live coding in Sonic Pi.

4.6.2 Quiz 2: Hear interleavings

Consider the following code:
 	
`live_loop :foo do`
`  play 50`
`  sleep 1`
`end`
 	
`sample :drum_cymbal_open`
	What happens if you run it?
	• All the code after the loop is not executed.
	• The loop and the sample are executed simultaneously.
	• The drum cymbal open sample is played while the loop is "asleep".
	• The loop will repeat 50 times and then play the sample.

In Quiz 2, the attention is focused on the difference between sequential and concurrent execution flow. Here the second answer is the correct one. Indeed, the `live loop` construct starts a concurrent thread which repeatedly plays note 50 while the main program continues to the `sample`.

4.6.3 Quiz 3: Hear the differences between threads and loops

Consider the following code:
 	
`live_loop :foo do`
`  sample :ambi_choir`
`  sleep 0.5`
`end`
 	
`in_thread do`
`  sample :ambi_drone`
`end`
	What happens if you run it?
	
	• The sample in the `live loop` command is played repeatedly, while the sample in the `in_thread` command is played only once.
	
	• It is not possible to execute both a thread and a live loop concurrently (runtime error).
	
	• Both the live loop and the thread sample are played infinitely.
	
	• Only the live loop will be executed.

In Quiz 3, the attention is focused on the difference between threads and iterative task. The `live loop` command starts a concurrent thread which repeatedly executes its body. The `in_thread` command requires an explicit loop inside its body to repeat a command more than once. Therefore, the first answer is the correct one.

4.6.4 Quiz 4: Listen to data races

Consider the following code:

```
live_loop :setter do
   set :foo, rrand(70, 130)
   sleep 1
end

live_loop :getter do
   puts get[:foo]
   sleep 0.5
end
```

What are the Set and Get functions of the Sonic Pi meant for?

- They allow threads to access a shared resource in a thread-safe way, but mutual exclusion is not guaranteed.
- They allow threads to access a shared resource in a thread-safe way, with guaranteed mutual exclusion.
- They are used to produce non-deterministic program behaviour.
- They are used to manipulate objects whose scope is restricted to a single thread or function.

In Quiz 4, we stress the importance of using thread-safe read/write operations in program with multiple threads. The commands `set` and `get` have been introduced to atomically modify data structures in Sonic pi. Therefore, the second answer is the correct one.

4.6.5 Quiz 5: Avoid Drifting

Given the following code:

```
live_loop :foo do
   use_synth :prophet
   play 20
   sleep 8
   cue :f
end

sleep 0.3

live_loop :bar do
   sync :f
   sample :bd_haus
   sleep 0.5
end
```

Are the two live loops synchronized?

- The two live loops are out of phase due to the 0.3 sleep between the two live loops.
- The two live loops are synchronised because the i-th iteration of the live loop `:bar` is synchronised with the i-th iteration of the loop `:foo` via sync, which waits for the `cue :f` event.
- The code is incorrect and will not be executed.
- The two live loops are not synchronised because the `get` and `set` methods are not used.

In Quiz 5, we stress the importance of time synchronization when reproducing audio signals via multiple threads. Due to possible delays in the scheduling of different threads, multiple threads executing different music samples repeatedly may get out of synch after few iterations, i.e. the resulting program can be affected by the drifting problem. To avoid this problem, Sonic Pi provides synchronization operations between thread groups. More specifically, the commands `cue` and `sync` enforce a rendez-vous synchronization in between the `:foo` and `:bar` threads. Therefore, the second answer is the correct one.

4.7 Team Application

In this Section we provide an overview of the team application tasks we developed for the TBL units. Tasks 1, 2 and 3 were designed for the CA and CP units, while tasks 4, 5, 6 and 7 were specifically designed for CP students.

CP stands for "Computer Architecture Course" and is an annual first-year course, while CP stands for "Concurrent Programming and Distributed Algorithms" and is a third-year course aimed at introducing multithreading/multicore, event-driven and distributed programming concepts. For a more detailed explanation, please refer to Section 4.8.1.

The team app tasks included a variety of question formats, including multiple-choice questions and open-ended tasks that required students to implement Sonic Pi code. Each exercise targeted specific misconceptions related to concurrent programming. We focused on a range of concurrent concepts, such as data races and synchronization mechanisms, as well as fundamental programming concepts like variable scope and function calls. The evaluation criteria are based on the

Table 4.1: Learning Objectives for Concurrency Management

Objective	**Level 1**	**Level 2**	**Level 3**	**Level 4**
Recognizing a shared resource	Does not recognize the presence of shared resources in the exercise	Recognizes the presence of shared resources but does not handle concurrent access correctly	Recognizes the presence of shared resources and implements synchronization mechanisms to ensure concurrent access	Recognizes the presence of shared resources, effectively implements synchronization mechanisms, and optimizes performance
Recognizing synchronization goals	Does not recognize the need for synchronization between processes or threads	Recognizes the need for synchronization but implements inadequate or complex solutions	Recognizes the need for synchronization and implements suitable and understandable solutions	Recognizes the need for synchronization, implements innovative and optimized solutions
Recognizing the correct visibility scope of variables	Does not understand the concept of visibility scope of variables in concurrent context	Partially understands the concept of visibility scope but makes errors in managing shared variables	Understands the concept of visibility scope and properly manages shared variables among threads	Fully understands the concept of visibility scope and implements advanced solutions to manage shared variables
Recognizing race condition	Does not encounter race conditions and fails to resolve them	Identifies race conditions but only partially resolves the problem	Effectively manages race conditions in most cases	Successfully avoids race conditions and implements preventive solutions
Recognizing possible deadlock situations	Does not recognize deadlocks and does not resolve them	Identifies deadlock situations, but with some difficulty	Avoids deadlock situations and manages them effectively most of the time	Successfully avoids deadlock situations and implements effective preventive solutions

rubric presented in Table 4.1. It is important to remark that, in presence of multiple execution threads, the traditional concepts of variable scope become more complex. In addition to global and block/function local scope, it is also necessary to consider thread local variables and the interplay among all of them.

By embedding concurrent problems within these fundamental programming concepts, we aimed at highlighting the challenges and complications that arise when students do not have a clear understanding and a viable mental model of sequential execution and concurrency.

4.7.1 Task 1: Data Races and Variable Scope, 10 minutes

A

```
in_thread do
  use_synth :piano
  x = 40
  10.times do
    x += 4
    sleep 0.5
    play x
  end
end

in_thread do
 use_synth :kalimba
  x = 40
  10.times do
    x -= 4
    sleep 0.5
    play x
  end
end
```

B

```
x = 40
in_thread do
  use_synth :piano
  10.times do
    x += 4
    sleep 0.5
    play x
  end
end

in_thread do
  use_synth :kalimba
  10.times do
    x -= 4
    sleep 0.5
    play x
  end
end
```

C

```
x = 40
in_thread do
  use_synth :piano
  10.times do
    x += 4
    sleep 0.5
    play x
  end
end

x = 60
in_thread do
  use_synth :kalimba
  10.times do
    x -= 4
    sleep 0.5
    play x
  end
end
```

Voting cards: Which answer is true?

- In B and C, different threads operate on a common resource and the result depends on the order in which the different threads execute their instructions;
- In A and C, different threads operate on a common resource and the result depends on the order in which the different threads execute their instructions;
- In all three programs, there are no resources shared between threads;
- In A and C, there are no resources shared between threads. The result depends on the order in which the instructions of the different threads are executed.

Task 1 is based on of three Sonic Pi codes, namely A, B, and C, and on some ”voting cards” questions. Each code comprises two threads that access a global variable `x` without any synchronization mechanism. Students were required to select the correct answer from the following options. In script A, variable `x` is thread local, whereas in script B and C `x` is global. Therefore,

only A does not present data races as specified in the first answer. Furthermore, in B both threads simultaneously update `x` starting from note 40. The data race can be heard as perturbations of note 40. In C the second assignment eventually overwrites the first one and thus, after a sequence of perturbations of note 40, the program first jumps to the higher note 60 and then the sequence of perturbations continues from that note. The task specifically targets misconceptions related to data races and the management of global shared resources in a concurrent programming model.

4.7.2 Task 2: Data Races and Function Calls, 10 minutes.

```
x = 40

in_thread do
  use_synth :piano
  10.times do
    x += 4
    play x
    sleep 0.5
  end
end

in_thread do
  use_synth :kalimba
  10.times do
    x -= 4
    play x
    sleep 0.5
  end
end
```

```
x = 40

define :foo do |x|
  x -= 4
  play x
end

in_thread do
  use_synth :piano
  10.times do
    x += 4
    play x
    sleep 0.5
  end
end

in_thread do
  use_synth :kalimba
  10.times do
    foo x
   sleep 0.5
  end
end
```

```
x = 40

define :foo do
  x -= 4
  play x
end

in_thread do
  use_synth :piano
  10.times do
    x += 4
    play x
    sleep 0.5
  end
end

in_thread do
  use_synth :kalimba
  10.times do
    foo
    sleep 0.5
  end
end
```

Voting cards: Which answer is true?

1. Mutual exclusion mechanisms are in place in all programs.
2. In a program there is only one possible race condition in read/write but not related to multiple writes.
3. In programs where the “foo” function or procedure is present, these cannot be done check race conditions.
4. None of the previous answers.

Task 2 consists of three Sonic Pi programs and some “voting cards” questions. Students were required to identify potential data races. The task aimed at addressing both race conditions and fundamental programming concepts such as variable scope, function calls, and parameter passing. To identify a data race, students needed to comprehend the scope of variables and function calls within a concurrent scenario. In all three programs, there exists a shared resource referred

to as x. The first program features two threads that access x without any form of mutual exclusion or synchronization mechanism. The second program involves the definition of a function with a single formal parameter. This function is called 10 times by the second thread, with the variable x passed as an argument by value. Consequently, each function frame possesses its own local variable named x. However, as the function is called, the variable x is accessed without any synchronization mechanism in place, which can potentially lead to data races. In the third program, a function without parameters is defined and subsequently called 10 times by the thread. Each time the function is invoked, it accesses the global variable x without any synchronization mechanism in place.

4.7.3 Task 3: Sleep and Data race, 10 minutes

```
use_synth :piano

note=[52,55,59,40]
i=0

define :foo do |x|
  in_thread do
     play note[x]
  end
end

3.times do
  foo i
  i+=1
end
```

Voting cards: Which answer is true?

(1) The program has no race conditions.
(2) The program creates only one thread.
(3) The program plays the notes in the "`note`" list in sequence.
(4) The program plays the E minor chord.

Gallery walk:
Describe in detail the behavior of the script with particular pay attention to the changes to the value of the variable "`i`" and to the possible sequences of notes play.

Task 3 presents a multiple-choice question that requires students to reason about pausing execution and data races. The table shows the Sonic Pi code for this task, the "voting cards questions", and the "gallery walk". The code consists of two global variables: an array of notes and a counter variable initialized to zero. There is also a function with a parameter `x`, which defines a thread to play the x-th note from the array. A loop calls the function three times and increments the counter variable by one. The array is global and shared between the threads. However each thread uses the same array, i.e., potential race condition on a shared data structure, but with a different index, i.e., different cells of the array. Therefore, without any `sleep` invocation to introduce delays in thread execution, the audible output of the program is the E minor chord, i.e., the latter answer is correct.

4.7.4 Task 4: Thread-safeness, 10 minutes

```
A

a = (ring 60, 57, 65)

in_thread do
  10.times do
    a = a.sort
    print a
    sleep 1
    play a
  end
end

B

in_thread do
  10.times do
    a = a.shuffle
    sleep 1
  end
end
```

Gallery walk:
Implement a thread-safe solution for codes A and B using inter-thread synchronization (hint: use Sonic Pi thread-safe variables and methods).

Task 4 is an open-ended task requiring students to identify the synchronization goal and implement a thread-safe solution. The task involves a ring variable, which can be seen as a type of linked list, shared by two threads in a non-thread-safe manner. One thread sorts the ring, then sleeps for 1 second, and finally plays the notes from the ring. The other thread shuffles the ring and then sleeps. Each thread repeats this process 10 times in a loop.

The current implementation suffers from a race condition, resulting in non-deterministic behavior. Sometimes the thread plays the sorted ring, while other times it does not. The misconception lies in the identification of the synchronization goals, specifically the shared resource accessed by both threads. Students were tasked with identifying the synchronization goals and implementing a thread-safe solution to address the race condition. Three possible solutions of task 4 are shown in Figure 4.2.

4.7.5 Task 5: Locking and Synchronization, 10 minutes

```
set :add, 40

live_loop :producer do
  note = set :add, get[:add]+20
  play note
  sync :wait
  sleep 0.5
end

live_loop :consumer do
  sync :add
  note = set :add, get[:add]-20
  play note
  cue :wait
  sleep 0.5
end
```

Voting cards: Which answer is true?

(1) There exists an execution in which a thread fails and never obtains a resource (starvation).
(2) There exists an execution in which a thread eventually obtains a resource.
(3) There exists an execution in which the threads fail, and block each other, e.g., waiting for a resource or message (deadlock).
(4) The program does not presents the above mentioned behaviours.

Task 5 consists in a multiple-choice question, requiring students to analyze the lock mechanism and identify the presence of a deadlock. The task involves a shared resource, represented by the variable `:add`, which is accessed by multiple threads in a safe manner using the set-get operators of the Sonic Pi time state. The task is designed to address misconceptions regarding synchronization mechanisms based on message passing, specifically the set-get and cue-sync operations in Sonic Pi.

Despite the thread-safe use of the shared variable, a deadlock situation arises as both threads synchronize using `cue` and `sync` on messages `:add` and `:wait`, but in a wrong order, i.e., both threads remain blocked on the corresponding `sync` invocation.

4.7.6 Task 6: Thread-local scope, 20 minutes

```
x = ???__1__???
use_synth :beep

define :f do |x|
  define :g1 do |y|
    x = y + ???__2__???
    play x
  end
  play x
  sleep 2
  g1 x
end

define :g do |n|
  n = n + ???__3__???
  play n
end

define :h do
  x = x + ???__4__???
  play x
end

define :j do |x|
  x = x + ???__5__???
end

define :p do |x|
  use_synth :chipbass
end

define :q do ???__V__???
  x = x + ???__6__???
end

define :z do
  x = x + ???__7__???
  play x
end

in_thread do
  2.times do
    play x
    sleep 2
    x = x + ???__8__???
  end

  f x
  sleep 2
  g x
  sleep 2
  h
  sleep 2
  j x

  cue :tick
  sync :tick3
  ???__A__???
end

in_thread do
  sync :tick
  2.times do
    ???__B__???
    x = x + ???__9__???
    play x
    sleep 2
  end
  cue :tick2
end

in_thread do
  sync :tick2
  p x
  q ???__W__???
  z
  print x
  sleep 2
  cue :tick3
end
```

Determine the sequence of notes 50 54 58 62 66 70 74 78 82 86 by replacing ???__i__???? with the appropriate values/instructions.
Also determine the instructions (or none) to write instead of ???__A__?? and ???__B__?? to play notes 74 and 78 with the dark_ambience synth and note 86 with the beep synth.

Task 6 involves completing the Sonic Pi code using three threads and parameter passing functions to generate an ordered sequence of audible notes. Understanding thread synchronization and

scope rules for variables and synthesizers is crucial to solving the task. In particular, synthesizers are global but thread-local, whereas variables can be shared between threads and be local within a function. This task requires a higher level of understanding from students. Success in these tasks requires mastery of skills and the ability to apply theoretical knowledge creatively.

4.7.7 Task 7: Master/slaves Synchronization, 20 minutes

Given the following melody (theme from Super Mario Bros.):

```
play_pattern_timed([nil,nil,:e5,:ds5,:d5,:b4,nil,:c5,
                    nil,:e4,:f4,:g4,nil,:c4,:e4,:f4,
                    nil,nil,:e5,:ds5,:d5,:b4,nil,:c5,
                    nil,:f5,nil,:f5,:f5,nil,nil,nil,
                    nil,nil,:e5,:ds5,:d5,:b4,nil,:c5,
                    nil,:e4,:f4,:g4,nil,:c4,:e4,:f4,
                    nil,nil,:gs4,nil,nil,:f4,nil,nil,
                    :e4,nil,nil,nil,nil,nil,nil,nil], [0.2])
```

Credits: https://gist.github.com/xavriley/87ef7548039d1ee301bb

The musicians are synchronized with each other, who are in turn synchronized by the conductor, who sets the tempo (locks and unlocks everyone).
Hint: consider a pattern with controllers (sync/cue)

1° thread/live loop

```
use_bpm 100
  use_synth :pulse
  use_synth_defaults release: 0.2, mod_rate: 5, amp: 0.6
play_pattern_timed([nil,nil,:e5,:ds5,:d5,:b4,nil,:c5, nil,:e4,:f4,:g4,nil,:c4,:e4,:f4,
nil,nil,:e5,:ds5,:d5,:b4,nil,:c5, nil], [0.25])
```

2° thread/live loop

```
use_synth :tri
  use_synth_defaults attack: 0, sustain: 0.1, decay: 0.1, release: 0.1, amp: 0.4
 play_pattern_timed([nil,:f5,nil,:f5,:f5,nil,nil,nil], [0.25])
```

3° thread/live loop

```
use_synth :tri
  use_synth_defaults attack: 0, sustain: 0.1, decay: 0.1, release: 0.1, amp: 0.4
play_pattern_timed([nil,nil,:gs4,nil,nil,:f4,nil,nil, :e4,nil,nil,nil,nil,nil,nil,nil], [0.25])
```

4° thread/live loop - Conductor

```
# Do stuffs
```

Task 7 simulates a memory barrier, where four threads, including three musicians and a conductor, interact. The musicians synchronize to play a melody, while the conductor pauses the other threads when a counter variable reaches a certain value. After a set time, the conductor sends a message for the threads to resume playing. The task requires students to synchronize the musicians and implement the behavior of the conductor thread, responsible for setting the tempo for the other threads.

```
a = (ring 60, 57, 65)
in_thread do
  10.times do
    b = get[:asaah]
    ciao = b.sort
    print ciao
    sleep 1
    play ciao
  end
end

in_thread do
  10.times do
    b = a.shuffle
    set : asaah, b
    sleep 1
  end
end
```

```
a = (ring 60, 57, 65)
in_thread do
  10.times do
    set :a, a.sort
    print a
    sleep 1
    play a
  end
end

in_thread do
  10.times do
    sync :a, a.shuffle
    sleep 1
  end
end
```

```
a = (ring 60, 57, 65)
set :foo, a
in_thread do
  10.times do
   aux = get[:foo]
   set :foo, aux.sort
   print get[:foo]
   sleep 1
   play get[:foo]
  end
end

in_thread do
  10.times do
   aux = get[:foo]
   set :foo, aux.shuffle
   sleep 1
 end
end
```

Figure 4.2: Task 4 Solutions

4.8 Experimental Evaluation

In this section we describe the two teaching experiments realized to evaluate the approach, first presenting the setting and then the obtained results.

4.8.1 Experimental Set-Up

The first experiment involved third-year students enrolled in the CP course, while the second experiment involved first-year students enrolled in the CA course on computer architectures.

The CP course is a third-year, second-semester course that focuses specifically on the design and analysis of algorithms for concurrent and distributed systems. It has a unique position in our B.Sc. program in CS as the only course dedicated entirely to concurrent programming. Students enrolled in this course have advanced knowledge in several areas, including programming (non-concurrent), databases, networks, and architectures. This foundational knowledge serves as a basis for exploring the intricacies of concurrent programming and gaining a deeper understanding of its principles and applications.

The CA course is an annual first-year course that focuses on teaching students the fundamentals of computer architecture design for modern microprocessors. Throughout the course, students gain knowledge and skills in several areas, including assembler languages, number representation and arithmetics, combinatorial and sequential circuits, and processor and memory hierarchies. From the 2022/2023 academic year, students have been introduced to concurrency through our experimental approach and then with a very brief introduction to multi-threading in C/C++ using the `pthread` and `std:thread` libraries. It is important to note that students enrolled in the CA course already have one semester of experience in imperative programming using C++.

The CP students are referred to as the P1 population, while the CA students are referred to as the P2 population. Specifically, P1 consisted of 54 third-year students (divided into 10 teams), while P2 consisted of 130 third-year students (divided into 34 teams).

Both experiments described in this chapter were conducted during the second semester of the 2022/2023 academic year. The first experiment took place at the beginning of the CP course, where students were introduced to concurrency concepts. The second experiment was conducted in April, just before students in the CA course were introduced to multi-threading in C. The timing of these experiments allowed students to gain a basic understanding of concurrency through the Sonic Pi-based approach, before delving into multi-threading in other languages.

The activities did not contribute to the student's final summative assessment. Instead, they had formative value, serving as learning experiences and opportunities for students to develop their understanding of concurrent programming concepts.

During the activities, we collected team APP tasks to analyze students' understanding of concurrency and any misconceptions that might arise, according to the rubric described in Table 4.1. We also collected data through an anonymous individual post-questionnaire on students' appreciation/perception of the activity. Finally, we collected the results of a post-individual test to investigate knowledge transfer. The test consisted of three exercises formulated either in pseudocode or in the C programming language. These exercises proposed a similar concurrency scenario, but in other languages, and provided practical opportunities for students to apply their understanding of concurrency in a different learning context.

Technically, the final questionnaire and the "knowledge transfer" test were developed using Google Forms, which provides a user-friendly interface for data collection. The questionnaire and the test can be found in the appendix A.2 and A.3. The TBL quizzes were developed using the Moodle platform, which provides a comprehensive set of features for online learning and assessment. In addition, we used platforms such as Wooclap[11] and Padlet to collect the team task solutions and responses. These platforms provided a collaborative environment where teams could share and submit their solutions, encouraging student engagement and teamwork. Overall, the combination of these tools facilitated efficient data collection and supported the interactive and collaborative aspects of the learning activities.

[11] https://www.wooclap.com

4.8.2 Experimental Results

The questions included in the RAT quiz have been carefully selected based on a refinement process of similar past activities conducted with Sonic Pi. The test aims to assess the students' understanding of some basic concepts related to concurrency and live coding, which are essential for completing the Team APP tasks in the TBL module. Due to time constraints and the structure of the TBL module, only 5 questions were chosen, each focusing on independent concepts. The data collected from the quiz were not used to answer the research questions, but are reported for completeness to present the full material of the TBL activity, as the quiz served as a preparation tool for the subsequent Team APP phase. Notably, the repeated quiz within the team generally yielded better results. However, it was not our intention to statistically validate this result using psychometric or other techniques to assess validity and reliability, as these data were not used in the research study. The results are reported in Appendix A.1 as additional material for completeness only. In the following section we focus our attention on the results obtained for the Team APP tasks.

4.8.2.1 Team APP Results

We conducted both item and code analyses of the team tasks to identify potential errors and misconceptions. Specifically, for multiple-choice questions, we tried to identify patterns in incorrect answers to determine which concepts were common misconceptions. For tasks that required students to write code, we analyzed their solutions to identify specific errors. The results are presented per task, each time distinguishing between the CA and CP scenarios.

Task 1 60% of the CA teams answered the first question correctly. However, 13% incorrectly voted that the threads in the first and third programs use a shared resource, resulting in a nondeterministic program. In addition, 7% incorrectly voted that there are no shared resources in any of the programs, making them deterministic. Finally, 20% voted that the first and third programs had no shared resources and were therefore not deterministic. These responses highlight misconceptions around the concept of shared resources, especially in the first exercise where the misconception is "reinforced" by a variable scope error (confusing local scope with global scope). In the second exercise, it is clear that there is a lack of clarity among these teams on the definition of a shared resource. There is also a misunderstanding about the meaning of a deterministic program, since a program can be non-deterministic even without shared resources between threads. Finally, the last answer reveals a misconception of both shared resources between threads and variable scope, where local variables with the same name are mistakenly considered to be shared between threads.

The CP scenario is significantly different, with 90% of the teams answering correctly. Only 10% voted for the response that all three programs do not have shared resources.

Task 2 In task 2, 48% of CA teams answered correctly. 19% voted that in the programs where the function "foo" is defined, there cannot be any race conditions. This highlights a poor understanding of the program, as there are still data races present in the second program. There is a misconception that, because it is pass-by-value, a race condition can never occur regardless of the context. Additionally, 5% of the teams voted that all programs have a mutual exclusion mechanism for accessing the shared resource. In this case, it is evident that there is a lack of understanding of concurrent mental models, where it is not clear what a synchronization mechanism is for protecting access to a resource. Finally, 29% voted that no answer was correct, failing to recognize the presence of a data race in the second program as well.

In this case, the results for CP are different. The percentage of correct answers is lower, standing at 30%. 70% of the teams voted that none of the answers were correct, thus failing to recognize the existence of a data race in the second program.

Task 3 In task 3, 67% of the CA teams and 78% of the CP teams answered correctly, stating that the output was the chord of A minor. However, 27% of the CA teams (and 22% of the CP teams) failed to recognize the presence of a race condition, and during the plenary discussion, the misconception emerged that a data race is necessary for a race condition to occur. Finally, 6% of the CA teams answered that the notes are always played in the same order, indicating that their mental model associates an implicit sequential order to the creation and execution of the three threads, thus not recognizing the race condition.

Task 4 Tasks from 4 onwards are exclusive to the CP experiment. Figure 4.2 displays three solutions to task 4, which encompass all the other cases. All teams recognized the problem (the thread orders the list but it is not always played in order) and applied a strategy to solve it. However, in the solution on the left, the correct synchronization objective was not identified. The solution was devised based on basic knowledge, which involved creating local copies of the shared variable but failed to eliminate the underlying race condition. In the center solution, there is a partially correct approach, but it exhibits misconceptions regarding synchronization mechanisms, as both a lock and a wait signal are utilized, albeit the wait signal is used incorrectly. Finally, the right solution demonstrates a correct approach (apart from some syntax errors) where locking mechanisms are applied to the shared resource.

Task 5 In Task 5, 50% of the teams identified the deadlock situation. 10% described it as starvation, while 30% believed that one of the two threads would eventually get all the resources. Finally, 10% answered that it was something else. In this case, the students displayed an inaccurate mental model of program execution, failing to recognize that the two threads were stuck waiting for resources held by each other, preventing any progress in the program.

Task 6 & 7 All teams successfully completed Task 6 with the correct solution. During the activity, there were some challenges related to the specific scope rules of Sonic Pi. However, once the dual scope of synthesizers and variables was clarified by us, no further issues arose. The teams effectively utilized the knowledge acquired from previous tasks. Task 7, on the other hand, proved to be more challenging as it was a less guided exercise compared to the previous ones. It required a deep understanding of thread synchronization mechanisms and the management of a shared resource. Not all teams submitted a solution as they were unable to translate the problem into a practical solution. Others partially implemented a solution where the conductor coordinates the various threads but fails to pause/resume execution in a "memory barrier" style. Only one group managed to successfully implement the required solution. Some of the team results can be found in the appendix A.4.

4.8.2.2 Final Questionnaire

CA A total of 23 people responded to the final CA questionnaire. None had any previous experience of concurrent programming and Sonic Pi or Ruby. For the question "I think the musical approach with Sonic Pi is useful for understanding concurrency" [A] the median[12] is 4. For the question "I think the activity was effective in introducing and/or deepening some concepts of concurrent programming" [B] the median is 3. In particular, the students who gave a negative or neutral answer to question A were the same students who gave a neutral or negative answer to question B. Figure 4.3 shows the distribution of responses on a Likert scale from 1 to 5.

Participants commented that the activity was useful for understanding the importance and benefits of concurrent programming, learning a new programming language, understanding the concept of threads and synchronization, and improving their knowledge of concurrent programming.

74% of participants found the activity to be at an appropriate level of challenge, 22% found it difficult and the remaining 4% found it easy. Regarding the time allotted for the activity, 65% of participants felt it was sufficient, while 35% expressed the wish for more time to complete the tasks. In terms of individual preparation, 6 people consider that they have prepared adequately for the activity, 14 do not know, and 3 insufficiently. Finally, 22% of students found the teamwork experience very useful, 30% useful, 26% neutral, 13% not very much, and 9% not at all.

CP The final CP questionnaire was completed by 16 students. Of these, 44% had previous work experience in concurrent programming and 30% had used Sonic Pi in the ICDD course. For the question "I think the musical approach with Sonic Pi is useful for understanding concurrency" [A] the median is 4. For the question "I think the activity was effective in introducing and/or

[12]The median is used in the analysis of Likert scale data because of the ordinal nature of the scale. Ordinal data implies that the response options have a rank or order, but the distance between each option is not necessarily equal.

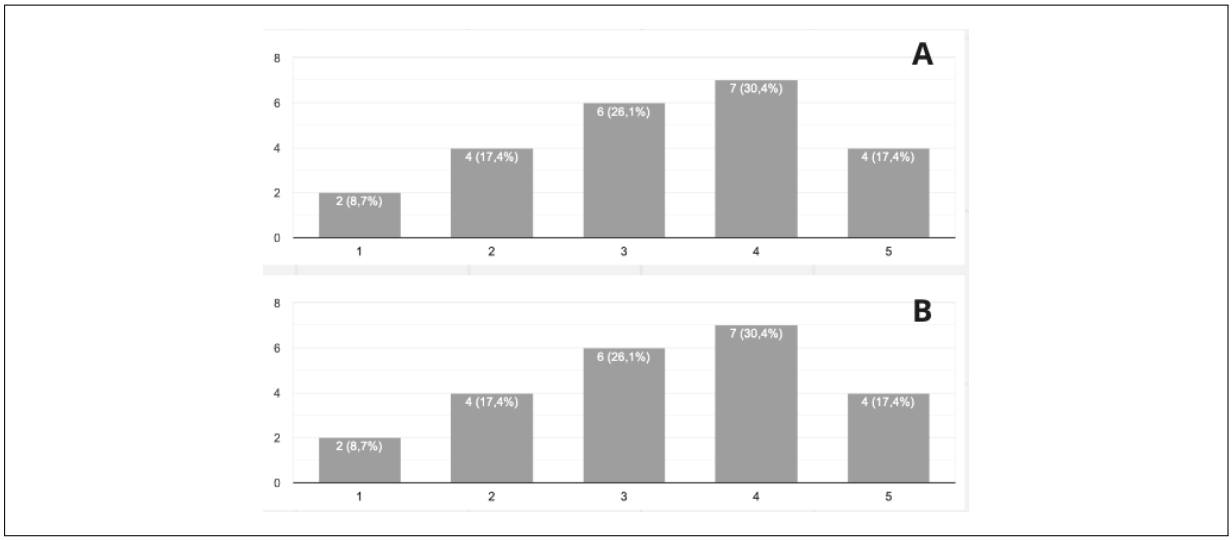

Figure 4.3: CA final questionnaire

deepening some concepts of concurrent programming" [B] the median is 4. In particular, students who gave a negative or neutral response to one question were the same as those who gave a neutral or negative response to the other question. Figure 4.4 shows the distribution of responses on a Likert scale from 1 to 5. In addition, several students commented that the approach provided a useful introduction to concurrency and offered a clear practical example. In addition, 80% of the students reported that they found the level of difficulty of the activity to be fair, while 18% found it to be difficult. Regarding the time allocated, 70% of the students felt that it was fair, while the remaining students expressed a preference for more time to complete the tasks. In terms of individual preparation, 44% people consider that they have prepared adequately for the activity, 31% do not know, and 25% insufficiently. Finally, 25% of students found the teamwork experience very useful, and 75% useful.

4.8.2.3 Assessing Knowledge Transfer

CA Only 6 students answered the first exercise correctly. In particular, most of the incorrect answers revealed a mental model still based on sequential execution, where note 60 would be played only after the other thread had played notes 45 and 50, thus describing deterministic behavior. One student confused the behavior of the `cue` with that of `synth`, while two students confused `in\_thread` with `live\_loops`, stating that the sound would be repeated an infinite number of times. Finally, one student replied that only notes 45 and 60 would be played.

In the second exercise, in pseudo-code, all but one of the CA students correctly identified the possible scenarios. Only one student said that the program always generates an error.

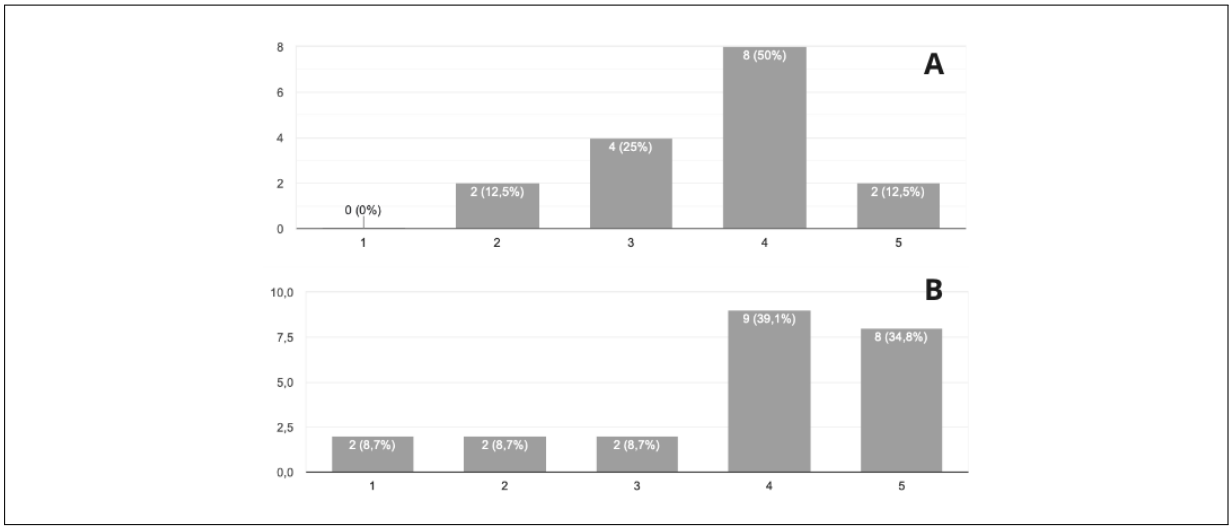

Figure 4.4: CP final questionnaire

In the third exercise, using C with the pthread library, 80% of the CA students answered that the program was not deterministic. However, only 15% of them correctly identified the possible outputs of the program. The remaining students recognised the correct outputs but also added the combination 0 0 or 0 1 10, which are impossible as they can never occur. These answers once again highlight the problem of misconceptions about basic concepts, particularly references and pointers.

CP In the CP experiment, 7 out of 16 students solved the first exercise correctly, while the others either did not answer or considered a sequential model. The second exercise was solved correctly by all students. Finally, 80% answered the third exercise correctly (non-deterministic and correct output), while the remaining students judged the behavior of the program to be non-deterministic.

4.9 Discussion

TBL Methodology An innovative aspect of our approach is the adoption of a TBL pedagogy. This method combines elements typical of flipped-class with individual and team activities in a series of focused tasks that perfectly fit our goal, i.e., exploring concurrency misconceptions and common mistakes. The strict integration of the activity with the considered courses, e.g., knowledge transfer in order to formulate problems seen in Sonic Pi in the languages adopted in the courses, feedback on the proposed quiz and task to the student involved in the course,

combined with a generally positive evaluation of the activity represents a first step towards the stable application in a future edition of our bachelor degree.

TBL results and post-questionnaires allowed us to perform quantitative and qualitative analysis of the outcome of our experiments. It is also important to remark that the population considered in our experiments consisted of 184 university students. Furthermore, the proposed method is the result of a refinement along three different academic years involving around 400 students in total. The high number of students, involved voluntarily, seems to indicate the widespread perception of the usefulness of innovative approaches to programming education.

Sonic Pi for Introducing Concurrency In our opinion, there are several reasons for adopting Sonic Pi as an introductory language for concurrency. First of all, music provides a stimulating and creative domain in which to start "thinking concurrently". Furthermore, Sonic Pi is based on Ruby with advanced programming aspects such as thread-local variables and hot-swapping code, thus representing a stimulating language for university students. The sound manipulation library is interesting by itself since it provides operations to create new sounds, to modify, combine, and reproduce music samples. As in nested variable scopes, filter declarations can be nested to create a sort of effect scope that can be confined within a single thread. Concerning programming constructs, another distinguishing feature is that Sonic Pi provides constructs at different abstraction levels for creating asynchronous threads: the `in_thread` command is closer to the C `pthread_create` invocation but with a much simpler syntax; the `live_loop` embeds a built-in infinite loop and, thus, it resembles the typical `demon` pattern used in the most common system programs. Hot-swapping code makes the editor a very effective tool for experimenting with all features of the Sonic Pi programming language and, in particular, with those related to concurrency and synchronization.

In our concurrency misconception exploration, the possibility of using sound and music creation not only as the target of a given exercise but also as a concrete means to hear anomalies (e.g., data races, missing synchronization, out-of-order executions) turned out to be a key ingredient of our experiments. For instance, we exploited this feature in different tasks included in the Team App, e.g., to hear the difference between a correct program (which expected result is to reproduce a chord) and an incorrect one (which result is a random sequence of the notes in the chord).

The fact that Sonic Pi is a domain-specific language can be seen as a possible disadvantage in using it to introduce concurrency in the first year of undergraduate courses. Introductory courses are typically based on widespread languages such as C and C++ and are mainly focused on programming methodologies and data structures. To overcome this problem, in our experiments, Sonic Pi has been introduced in the context of a mini-course together with the TBL method. Together they were also meant as innovative methods to stimulate the students' attention and learning process.

In conclusion, the simplified concurrency abstractions of Sonic Pi combined with a modern pro-

gramming style and a domain-specific flavor is a perfect fit for first-year university students.

Data Analysis The results obtained from the tasks highlight the presence of misconceptions among the participants, both about concurrency and about fundamental programming concepts within a concurrent scenario. For example, participants showed a lack of clarity in distinguishing between local and global scope and in understanding how shared resources affect program behavior, and some others mistakenly believed that certain programming mechanisms, such as pass-by-value, could eliminate the possibility of race conditions.

The results seem to highlight the importance of addressing misconceptions with targeted, guided interventions aimed at making concepts and scenarios about concurrent programming explicit. Furthermore, the plenary discussion generated during the activity could help participants to consolidate their understanding and develop a viable mental model. A notable finding is that addressing misconceptions is also useful for third-year students, where one might expect them to have acquired advanced programming skills.

In terms of knowledge transfer, it appears that students found it easier to align their existing mental model of programs in Sonic Pi with the pseudo-code representation. On the other hand, the exercises using C syntax required more effort in terms of adapting a mental model from one programming language to another with a different NM.

Generalizability of Results First, the tasks may not cover all aspects of concurrent programming, and the results may not fully capture students' overall understanding of concurrency. Our tasks were designed with only a few misconceptions in mind. As a result, the assessment of students' understanding is based on their responses to specific tasks and may not reflect their overall understanding of concurrent programming concepts. In addition, the sample size in both experiments may not be representative of the entire student population, which may limit the generalisability of the results, especially for the knowledge transfer part. In fact, only a small fraction of the students responded to the final questionnaire, which limits the significance of the results. Therefore, the generalisability of the knowledge transfer results should be interpreted with caution and further research is needed.

4.10 Conclusions

In this chapter, we presented an introductory approach to concurrency aimed at exploring undergraduate students' misconceptions using the Sonic Pi language so as to exploit the natural connection between multi-threading and live music coding. In our view, this chapter serves as a pilot study and provides preliminary evidence for the CS education community.

More specifically, we have discussed the methodology adopted in our experiments and the population settings, the material developed for the RAT quiz and Team APP activities, and commented on the data collected before, during, and after the activities. The experiment comprises an additional transfer learning experiment designed via a series of exercises proposed both in Sonic Pi and more traditional multi-threaded programming languages such as C and C++.

Promising initial results from questionnaires and tasks provide strong motivation for us to delve deeper into live music coding and targeted exercises, particularly those related to program comprehension. Our results are consistent with the existing literature on misconceptions, suggesting that students may face challenges not only in grasping concepts related to concurrency but also in understanding fundamental principles underlying the computational model. These considerations also apply to third-year students, leading us to reflect on the pedagogy of programming education in our undergraduate program.

In particular, the results of the Team APPs and the plenary discussion seem to suggest that a focus on mental models is beneficial in improving concurrency education, both to teach viable mental models and to present specific cases where common non-viable mental models are inappropriate and lead to misconceptions.

The results of the knowledge transfer test are quite encouraging, at least for the pseudo-code scenario, but future research is needed.

Based on the first year's results, we believe that the proposed approach could also be valuable in secondary education, especially in high schools with a focus on STEM education or in vocational schools specializing in technology-related fields.

Our study also has some limitations, such as musical background. The use of Sonic Pi may be a potential barrier for students with no prior musical experience or limited musical interest. It may require an initial investment in learning basic musical concepts before fully grasping the concurrency concepts introduced. This could potentially hinder their interest and understanding of the concurrency concepts. Future research is needed to understand this.

Finally, future experiments will provide fertile ground to further investigate the effectiveness of our approach and to analyze and address other misconceptions that are not included in the current research.

Data availability

The data that support the findings of this study are openly available on Github https://github.com/researchDataset/SonicPi_dataset/tree/main.

Chapter 5

Collaborative Learning in an Introductory Database Course[1]

The learning outcomes of introductory database courses are quite broad, encompassing very diverse abilities and skills. For increasing the effectiveness of learning w.r.t. the different skills and ensuring continuous engagement of students, we experimented cooperative methodologies throughout the course. Specifically, Think-Pair-Share (with adapted and tailored cycle times) was applied for both application exercises and hands-on laboratory activities, whereas we adopted team peer-reviews for the more complex final design activities. The chapter describes and evaluates the intervention, in terms of student engagement, appreciation, and learning outcomes.

5.1 Introduction

Database education plays a significant role in the teaching of CS. Specifically, in our university, the "Introduction to Databases" is a second year course, with around 200 students enrolled in two different degrees, namely the CS and the Statistics Bachelors.[2] The learning outcomes of the course, as those of typical introductory database courses, are quite broad, encompassing very diverse abilities and skills. The aim is for students to acquire both data modelling (M) skills, mainly related to domain understanding and abstraction capabilities, and querying (Q) capabilities, mainly related to precise logical specification of information needs and programming.

The students are thus expected to learn how to (M1) design a database, from a requirements spec-

[1] This chapter is based on the material published in [34].

[2] The course is offered for 12 ECTS to CS students, including also (basic) administration topics, and for 8 ECTS to Statistics, only including database design and querying skills. In the chapter, we focus on the 8 ECTS subset of the course, which is taken by all the students.

ification document to SQL DDL statements, establishing data integrity constraints; (M2) check the quality of the proposed schemas and possibly improve them by eliminating redundancies and anomalies; (Q1) express queries in relational languages; (Q2) express queries and manipulation operations in the SQL language.

In the last decades, active methodologies and learner-centered approaches has proven effective in the teaching/learning process of CS. One of the most common approaches to database education is project-based learning [41, 56]. It was hybridized with collaborative techniques from [162], resulting in a customized approach relying on case studies and teamwork with student roles. Many other approaches rely on gamification, i.e., the use of gamified elements, such as challenges, achievements, points, and story mode, into learning environments. Gamification had been applied for database design in [42] and MonstER Park [158], aimed at learning graphical conceptual modelling (ER diagrams). Game-based approaches to learn SQL include SQL Island [157], an online game for the creation of SQL queries, QueryCompetition [187], a competitive gamified web environment, and SQL Scrolls [141], a game framework with Parson's puzzles.

Collaborative techniques mainly rely on comparison of views and discussion to maximize participation, focus attention, and engage students. Specifically, Think-Pair-Share (TPS) [14] is a collaborative learning strategy where students work together to solve a problem. This strategy requires students to (i) **T**hink individually about the solution to the problem; (ii) **P**air and discuss the devised (partial) solution; (iii) **S**hare ideas with classmates. It has been successfully applied to introductory CS topics, mostly related to programming [91], and in database application exercises [111] but its potential in laboratory and in a more comprehensive design task has not been exploited, nor its effectiveness for modelling skills investigated.

Peer review [128] is a reciprocal process whereby students produce feedback reviews on the work of peers and receive feedback reviews from peers on their own work. Learning benefits are derived both from receiving feedback reviews and from the learning mechanisms activated by producing feedback reviews. Peer review has been used in the form of code review for project work in CS1 courses [80] and also in introductory database courses [78]. When using peer review in early bachelor years, one of the problems is that students struggle to provide meaningful peer feedback, resulting in missing reviews (i.e. students not submitting the review on time) and poor quality feedback. This issue was confronted in the collaborative team peer review [110], in which reviews were carried out by teams. It was found that teams produced higher quality feedback and found the activity more engaging and enjoyable than individual reviews.

The purpose of this study is to investigate collaborative learning techniques (namely, TPS[3] and team peer reviews) for different learning goals in the introductory database course. By applying the TPS cooperative methodology (with adapted and tailored cycle times) we aim at extending the benefit of the methodology to diverse learning goals, working with the same methodology

[3]The methodology we actually employ is referred to as Think/Write-Pair-Share since the solutions are shared in the teams and across teams as written artifacts. Moreover, in our case, we opted for 4 person teams rather than pairs.

for both application exercises and hands-on laboratory activities. For more complex final design activities, the collaborative approach extends to team peer-reviews, where each team collaboratively reviews the design outcome of other teams.

This chapter thus specifically addresses the following research question: *is collaborative learning effective, in terms of learning outcomes, for the different skills to be acquired in an introductory database course?* The degree of achievement of a learning goal will be measured in terms of the partial evaluations on different written exam artefacts. We thus compare the partial evaluations obtained in individual partial assessment for specific skills to be achieved by the students and consider the variance among partial evaluations. Students engagement and appreciation are measured as well, in terms of the number of students leaving the teams/the activities and of answers given in final course feedback questionnaires.

The collaborative activities were designed in the context of a teaching innovation project, with the support of an instructional designer, and were planned for a traditional, on-site, edition of the course. A first edition of the intervention took place in academic year 2019/20, and was repeated in exactly the same way in academic year 2020/21. Both the editions were taught entirely online. In this chapter, we evaluate the second edition of the intervention (2020/21), characterized by on-site standard exam modalities[4]. A preliminary report on the first edition is available in [33] with no specific reference to the research question we address here.

The remainder of the chapter is organized as follows. Section 5.2 provides an overview of the participants and a detailed description of the activities and then discusses data collection and data analysis. Section 5.3 reports the achieved results. The results are discussed in Section 5.4 while some conclusions are presented in Section 5.5.

5.2 Methods

5.2.1 Population

The second-year database course was attended in academic year 2020/21 by 180 undergraduate students enrolled in the Bachelor in CS (around 80%) and in Statistics (around 20%) at the University of Genoa, Italy. The class was heterogeneous as students previously attended different math and programming courses. Prerequisite knowledge consisted of basic programming skills, set theory, and propositional logic. There was a gender gap, with around 20% females (and a much higher rate in Statistics than in CS). Participation in the collaborative activities was optional, and of the 180 students in the enrolled cohort, the set C of students participating in the collaborative activities included 70 students (39%) while the set S of the not participating ones

[4]The evaluation modalities for the 2019/20 academic year were exceptional (oral exam only) and no assessment of individual artifacts is available.

included 110 students (61%) (3 of which initially opted for participation but withdrew or were excluded due to inactivity later in the semester). All the collaborative activities had formative assessment only. The exam modalities (that will be detailed in Section 5.2.3) were identical for C and S students, the only exception being that C students delivered the project as a team, while the project is individual for S students.

Our study sample was a subset of the total cohort. Specifically, we only analyzed the set P of students who passed the exam for the course. The latter consists in 95 students (72 CS students and 23 Statistics students, 20 females), out of which 50 (denoted as PC) participated in the collaborative activities and 45 (denoted as PS) did not.

At the beginning of the semester (second week of classes) we formed the teams by maximizing intra-group heterogeneity (since diversity fosters discussion) and inter-group homogeneity (so that teams are comparable and balanced). To generate the groups we rely on the *GRumbler algorithm*[5], supported by an ad-hoc app. The GRumbler is designed to maximize the mixing across the class and provide the greatest possible degree of diversity within each group. Heterogeneity criteria include gender, degree program, indicators of academic performance (number of credits acquired and average grade), previous knowledge of databases (professional or from high school), and personality traits (e.g., being introvert rather than extrovert). This information was collected during the first week through an online questionnaire. We started with 19 teams, all but 2 of 4 students (2 teams only included 3 students). The groups were stable for all the activities of the semester.

5.2.2 Setting/Approach

The intervention consists in: (i) TPS (with weekly cycles) for laboratory exercises (SQL); (ii) TPS (with short cycles) for application exercises on data modelling; and (iii) collaborative team peer review of the conceptual design for the final project. Collaborative team peer review means that both the reviewed artifact is designed by teams and the review activity is performed collaboratively (first individually drafting the reviews and then refining and consolidating them after group discussion). Moreover, for monitoring the behavior of the teams, we propose (iv) two peer evaluation activities (one intermediate and one final). All the activities provided a formative assessment only and do not contribute to the final grade.

As discussed in Section 5.1, our results refer to the second edition of the intervention, in academic year 2020/21 (March-May 2021). Lessons and collaborative activities were held online, relying on the Microsoft Teams video conference platform, Moodle as e-learning platform, Wooclap as a polling system, and GitHub classroom for SQL assignments. As this was the second year of online teaching, students (and teachers) were already familiar with the systems and platforms. Written exams were in presence.

[5] GRumbler: https://scholar.harvard.edu/msparrow/grumbler

Lab	Topics
1	DDL: creating tables and key/foreing key constraints
2	QL: select project join queries
3	QL: outer join and group by
4	QL: subqueries
5	QL: updates and constraints
6	QL: views

Table 5.1: Weekly cycle for lab exercises - Topics

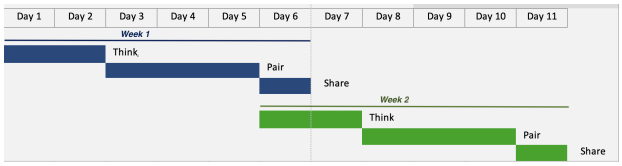

Figure 5.1: Weekly Cycle - Gantt Chart

In what follows, we provide some additional details on the various collaborative activities proposed to C students. The students that decided not to participate in such activities (i.e., S students) were free not to attend classes devoted to collaborative activities (roughly amounting to 40% of the classes[6]) or to attend as spectators. They got access to all the materials, namely assignments, query expected results, and a summary of the discussion/draft solution for database design tasks, but were not involved in teams and had no assignments during the semester.

Weekly cycle TPS for laboratory exercises (SQL) Groups are invited to work for a week, including a 2 hours lab session, on a task (typically, formulating some queries on a given database).

The objective of the activities is to learn the SQL query language. The reference system is PostgreSQL. The organization of the activities (summarized by the Gantt chart in Figure 5.1, corresponding to two weekly cycles) was the following:

- *Think [Days 1 & 2]*: assignment of the task to the students via Moodle, and small confrontation with respect to their doubts. Students are asked to start working individually on the assignment, or at least to get acquainted with it.
- *Pair [Days 3–5]*: work in the small group. This starts in the 2 hours lab session and is carried out within the Microsoft Teams channels. The instructors and the tutors play

[6]The remaining 60% was equally split among (pre-)recorded asynchronous frontal lectures and synchronous discussion sessions, mainly relying on Wooclap pollings.

Ex Session	Task	Description
ER Conceptual	1.1	core entity identification
Modelling	1.2	attribute vs relationship
Case 1	1.3	identifiers
(cinema hall	1.4	relationship vs (weak) entity
programming)	1.5	hierarchies
ER Conceptual	2.1	core entity identification
Modelling	2.2	identifiers
Case 2	2.3	hierarchies
(training company)	2.4	relationship vs (weak) entity

Table 5.2: Short cycle for design exercises - Tasks

the role of learning facilitators by managing the activity by intervening in the channels if requested by the learners and by joining the discussions on the channels at least twice in each session to monitor the progress. (One instructor and two tutors for 19 teams mean 6/7 teams per supervisor.) The step ends with the submission on Moodle of the team artifact (SQL commented file).

- *Share [Day 6]*: in synchronous hours, in the general Team channel, by discussing the main issues emerged by the teams' artifacts and problems encountered. Discussions are facilitated by Wooclap pollings.

The *Share* activity of week i overlapped with the start of the cycle for week $i + 1$. This general structure was applied for 6 weeks.

Table 5.1 reports the details of the specific notions on which each individual weekly task refers to. Note that, for assignments involving query specification (i.e., labs 2–6 in Table 5.1), the result returned by each query on the reference database is provided to the students for checking the correctness of the specified query by result comparison[7]. The availability of such results, together with discussions within the team and with the tutor were quite effective in avoiding, in the queries submitted at the end of the *Pair* phase, the syntax issues, incorrect or missing tables and columns, and query returning incorrect results discussed among the misconceptions identified in [114]. During the sharing phase, the work submitted by the teams (and the main issues emerged during the discussion) are exposed, thus the instructors can monitor the results / improvements obtained by the students. Individual feedback are provided to each team asynchronously.

5.2.2.0.1 Short cycle TPS for data modelling application exercises Teams are invited to carry out all three phases of the think-pair-share technique within the single two-hour exercise

[7]This will be referred to as "query results" in the feedbacks in Table 5.5.

session. The goal of these activities is to deepen aspects concerning the practical application of database design. Specifically, after an introductory section on relational data model understanding, the first two sessions were devoted to conceptual design while the third one was devoted to logical design. The activities are organized according to the following schema:

- *Think*: sharing of the task with the students, short confrontation with respect to doubts by the students, individual work by the student resulting in a written artifact (a small Entity Relationship diagram for the conceptual design) shared with the team (typically posting it on the chat of the Microsoft Teams channel of the team). 10 minutes are dedicated to this phase.
- *Pair*: sharing in the small group takes 12 minutes and is carried out in a meeting on the Teams channels, the instructor and the tutors manage the activity by intervening in the channels if requested by the learners. The team artifact is shared with the instructor.
- *Share*: the sharing phase takes place in 12 minutes, within the general meeting of the Teams channel. A team is chosen to present their solution, another team is chosen as an opponent and, using the Wooclap tool, learners in each team are asked to share the result.

This general structure allows us to complete an exercise divided into 4-5 sub-tasks within a session. Table 5.2 reports the details of the specific notions on which each individual sub-task refers for the ER Conceptual Modelling sessions. During these activities, the role of the instructor and of the tutors is of facilitators. After having explained the practical organization of the activity (phases, roles, timing, tools, and deliverables) and the assigned tasks, during the synchronous lab activity they monitor the interaction within teams and join the team meeting (or interact with them through the chat) if the team asks for help or clarification. During the sharing phase, the work submitted by the teams (and the main issues emerged during the discussion) are exposed, thus the instructor gets insight about the results achieved by the students.

Collaborative Team Peer Review of the conceptual design In this activity, the teams are assigned a case study (specifically, the database supporting an online game platform[8]) and submit a conceptual design for it (i.e., Entity Relationship diagram with data dictionary and constraints). This artifact is submitted via a Moodle Workshop activity and collaboratively peer-reviewed by 3 other teams. The criteria for peer review (presented in Table 5.3) are illustrated to the students and they can do some practice on an example submission (another schema with an example evaluation by the instructor). For each criterion, teams are required to provide a numerical evaluation and to comment / highlight any devised problems in the pdf.

[8]The case study assigned in the 2019/20 edition of the course was the database supporting a solidarity time bank.

Criterion	Question
Syntactic correctness	is the schema syntactically correct?
Syntactic completeness	does the schema include cardinality constraints, identifiers, types of hierarchies?
Semantic correctness: entities and relationships	does the schema correctly identify the main entities and relationships?
Semantic correctness: cardinality and identifiers	are the specified cardinalities and identifiers correct?
Completeness	does the schema omit any relevant information?
Additional constraints (correctness & completeness)	does the project identify and formulate in a (clear and) correct way all the constraints that cannot be expressed in the diagram?
Readability	is the scheme readable and well documented?

Table 5.3: Criteria for conceptual model peer review

The time frame for review is one week. Again, a sort of Think-Pair is adopted for collaborative reviews, since each student is assigned a schema for reviewing (and she is in charge of inserting the final review in Moodle) but the review needs to be discussed and finalized within the team. The most valuable part of the review is not the numerical scores, but the comments, as each team is asked to resubmit the revised design (along with a rebuttal to the reviewers). In this way, a better conceptual design will result in a better final project. The final project submission will also include the logical design and SQL part of the online game support platform. The role of the teacher in this activity is to devise and explain the review criteria, to emphasise that alternative modelling of the domain is perfectly possible, so that the review should not be biased by the previous (possibly different) modelling already done by the team, and to provide the example review.

5.2.3 Data Collection

During the activities we collected data on student engagement (participation in activities, submissions, attendance, etc.) collected by Moodle and Teams, which led to the exclusion of 3 students out of the original 73 from the collaborative activities (as mentioned in section 5.2.1). At the end of the course, we collected data on student appreciation in the form of a final feedback on

Group	N	Mean	SD	t
Experimental Group (PC)	50	0.90	0.84	$t = -2.1907$
Control Group (PS)	45	1.36	1.16	

$p < 0.05$

Table 5.4: The t-test result of the students' deviations between exam results for the two groups.

the course (anonymous), including specific questions about the collaborative activities (appreciation/criticism for different activities). This is collected through a Moodle feedback form.

During the exam sessions (for the whole academic year, 5 exam dates) we collected data on exam results, in particular in addition to the number of students who completed the exam (already mentioned in section 5.2.1), we collected the individual grades assigned to each exercise in the written exam. The written exam consisted of a closed part and a part with exercises on:

M1 reverse engineering (from a relational schema to the corresponding ER diagram)

M2 database normalization (specify functional dependencies and recognize normalization issues in a schema)

Q1 relation algebra queries specification

Q2 SQL queries specification

Each part is assigned a score of 0-5. The passing threshold is 50% on both parts (modelling and querying). The final grade of the student is then obtained as a weighted average with the projects grade. In our analysis, we consider the variance among the individual grades for exercises M1, M2, Q1, and Q2.

5.3 Results

Homogeneity in partial assessments (main hypothesis) We performed a statistical analysis based on some statistical test in order to analyze and compare the variance of individual exercise grades in the two groups: students who passed the exam and joined the collaborative activities (PC, cfr. Section 5.2.1) and students who passed the exam and did not participate (PS, cfr. Section 5.2.1). The aim of the analysis is to assess whether there is a significant difference between the variances in individual exam results (grades) on the four exercises for the PC and

Activity	Usefulness of	Answer (Median)
TPS long	query results	4
	interaction inside the team	4
	interaction with the tutor	3
	feedbacks on delivered solutions	3
TPS short	Think	3
	Pair	3
	Share	3
Team	interaction inside the team[1]	4
Peer	reviewing activity	3
Review	obtained reviews	3

Table 5.5: Usefulness of the collaborative activities, answer on Likert scale from 1, definitely useless, to 4, definetely useful, ([1] in both modeling and reviewing).

PS groups of students. To calculate the variance of each student's exam results on the different parts, we used its unbiased estimator (dividing by $n-1$). As a summary index of the two groups we have chosen the sample mean of the gained variances. To check whether the deviations between exam results on the different exercises in the two student groups are significantly different we performed the unpaired two-sample t-test. All analyses were performed using the statistical software R[9].

To perform the unpaired two-sample t-test, we first verified the conditions for validity of the test. In particular, we assumed the two sample groups to be normally distributed according to the central limit theorem (each sample size is greater than 40).

By F-test we rejected the hypothesis for the two sample group variances to be equal ($p-value = 0.02761$) and consequently we set `"false"` in the `var.equal` argument in the `t.test` function of R. Finally, we set the alternative hypothesis to `"less"`.

The t-test performed on the deviations between exam results showed significant difference between the average variances of the two groups, with $t = -2.1907$ and $p < 0.05$; that is, the average variance of the students who participated in collaborative activities is significantly different than that of the other student group. In particular, the average variance of the students who participated in collaborative activities is significantly less than that of the other student group. Table 5.4 shows the t-test result.

[9] https://www.r-project.org

Engagement and appreciation. Our experience confirmed the usefulness of team peer reviews in mitigating the impact of missing reviews and increasing the quality of feedback [110]. Both are relevant issues when applying peer-review in early bachelor years, but we experienced no issues of late or missing reviews in this activity and we observed an overall good quality of feedback.

For what concerns student appreciation, we summarize in Table 5.5 the median answers on a Likert scale from 1 to 4, where 1 is "not at all" and 4 is "very high" to questions from the final anonymous feedback provided by students in C for specific activities described above. The overall perception of usefulness for the three activities is high (greater than 3). For collaborative peer reviews, the percentages of "useful" and "definitely useful" answers were 50% and 44% for the reviewing activity, while they both were 39% for the obtained reviews. In the final feedback questionnaire we also asked whether the student would have reconfirmed the choice to participate/non participate in the collaborative activities. The choice would have been reconfirmed by 95% of the participating students, and by 70% of the non participating ones.

5.4 Discussion

5.4.1 Performance and Engagement

Achievement of learning outcomes. Our main hypothesis is the effectiveness of collaborative methodologies for the different learning outcomes of the course, for assessing which we compare the partial evaluations obtained in individual partial assessment for specific skills to be achieved by the students and consider the variance among partial evaluations. The results shown in Section 5.3 confirmed that the average variance of the students who participated in collaborative activities is significantly less than that of the other student group, thus the partial results are significantly more homogeneous. Thus, collaborative learning seemed to be effective, in our study, in terms of learning outcomes, for the different skills to be acquired in an introductory database course. The discussions activated within the team and the continuous involvement in assignments etc. likely ensured a more balanced learning effort on the different topics, while students preparing the exam independently might devote more effort on a subject and less on another.

We also observed that the passing rate is much higher in C (71%) than in S (41%). This is likely due to the fact that students joining the collaborative modality are those that are in the most favourable conditions (non working students, no lacking prerequisites, etc.).

Student appreciation From the results of the final course feedback questionnaires the following findings emerged:

- The level of satisfaction with the collaborative activities is high, as the "regret rate" with

respect to the initial choice is much higher for students who do not participate in them, and all activities were perceived as useful (median of 3 or more),

- the longer the time for discussion and confrontation among the team, the higher the perception of usefulness (the lowest median values are those for TPS short),
- in all the collaborative activities, the discussion inside the team is considered as the most useful source of learning (together with the availability of query results for lab assignment, that enabled a first level of correctness check), with higher usefulness than support and feedback from tutors and plenary discussions with the instructor,
- in collaborative peer review, though collaborative reviewing ensured quite high quality reviews, providing review was deemed more effective than receiving them.

5.4.2 Threats to Validity

Internal validity Since the participation was on a voluntary basis, there is a potential bias in students choosing the collaborative modality (i.e., C rather then S). Though the participation did not affect the exam modalities (except for the fact that the final project is developed in a team rather than individually), as previously observed in Section 5.4, students joining the collaborative modality are likely those that are in the most favourable learning conditions. Moreover, more ambitious or communicative students may have more likely chosen C. The design of the teams could have affected the effectiveness of the collaborative activities. The attributes chosen for group assignment could also have introduced some bias, and, even with the same attributes, different algorithms and even different runs of the same algorithm (e.g., the GRumbler algorithm we employed) may produce different team, impacting team behavior and thus the results.

External validity Both editions of the course have been held online and this could have had an impact on collaboration (online collaboration is somehow different from in-person collaboration) preventing to generalize the approach to an in presence edition of the course. Furthermore, the results cannot be generalized to a course level due to the fact that the evaluation was carried out in a particular course at a specific university. However, they could be relevant for courses teaching introductory database modelling and querying notions in other CS-related degrees. Indeed, our study sample represents the target population (second or third year Bachelor's students). In addition, both the modelling and the SQL assignment were prepared by the lecturers with reference to notions typically taught in an introductory database course at tertiary level.

5.4.3 Limitations and Improvements

As mentioned in Section 5.4.2, the obtained results do not generalised to an on-site setting, since the entire course and collaborative activities were held online. Moreover, further factors could have influenced the results obtained by students in the graded artifacts, that we plan to consider in future analyses. The final grade was assigned to students as a weighted average of the written exam mark and the project evaluation. For students in C, the final project was developed in teams and the same grade assigned to all team members. Some peer evaluation activities were performed, in which students were asked to assess on a Likert scale both their own individual contribution to the team (self-evaluation) and that of each team member (peer evaluation), according to a number of dimensions. A finer analysis of the quality of teamwork and learning outcomes is a possible direction for future work, as well as an investigation of relationships between written exam and project marks. Further analyses could consider whether there are significant differences according to the gender, to the degree, or to the exam session.

5.5 Conclusions

The learning goals of introductory database courses are quite diverse, ranging from data modelling to query specification skills. Different active learning techniques are being applied to data education. In this chapter, we focus on collaborative methodologies and, specifically, on TPS and team peer evaluation; our evaluation demonstrates that the use of collaborative learning techniques was effective in getting a more homogeneous achievement of the diverse learning goals, measured in terms of exam results. The partial evaluation on different artefacts was indeed found to be significantly more homogeneous for students taking part in the collaborative activities. The techniques also demonstrated quite engaging and appreciated by the students.

Chapter 6

ToyPandas: A Python Package for Data-Centric Computing

This chapter introduces ToyPandas, an educational Python package that we have developed specifically to promote data-centric pedagogy for beginners with no programming background. The main features of the library are presented, as well as its implementation, pedagogical implications and comparison with related work.

6.1 Introduction

Over the years, we have developed several introductory programming units for freshmen and high school students involved in outreach activities. In the 2022/2023 academic year, we experimented with data-centric computing pedagogy and developed a data-oriented course in the Python programming language.

This course emphasised the importance of grounding programming instruction in real-world data analysis scenarios. Students were exposed to authentic data sets, taught how to extract meaningful insights from that data, and equipped with the programming skills necessary to perform data manipulation, analysis, and visualisation tasks. Further details of the course can be found in section 7.4, and the syllabus in the Appendix B.3.

A crucial aspect of the course design was the choice of programming language and data set. In particular, to effectively implement a data-centric computing pedagogy, it is crucial to have native support for tabular data structures. While professional programming languages, such as Python and R, provide such capabilities through dedicated libraries, they may not be the most appropriate choice for introducing beginners to programming. This is mainly because using these libraries often requires a certain level of familiarity with the core programming language concepts. In

response to this challenge, researchers have developed specialised educational tools. Two prominent examples include the Pyret[1] programming language and the "datascience"[2] Python library, as discussed earlier in section 3.2.4.

Motivations & Context Our instructional design process began with a thorough review of the existing literature to assess the suitability of different programming languages for our educational goals. We recognised the advantages of data-oriented languages such as Python and R, with libraries such as Pandas for tabular data manipulation. However, these solutions presented challenges due to their reliance on fundamental language constructs such as lists and dictionaries, which may not be fully understood by beginners.

While alternatives such as Pyret and the Datascience package were promising, they presented additional barriers. Their unconventional syntax and notional machines deviated significantly from the widely used Python Pandas library, potentially hindering students' transition to the standard Python environment.

Our primary goal was to provide an authentic Python Pandas programming experience, while prioritising accessibility and learning for beginners. To achieve this, we aimed to simplify the notional machine used by the original Pandas library to fit within the confines of a 15-hour introductory course. We wanted a programming language with a short learning curve and seamless integration with Python's Pandas library to minimise the potential for misconceptions that come with learning a new language.

6.2 ToyPandas Package

To address these challenges, we have developed ToyPandas, an educational Python package built on top of Pandas. ToyPandas provides a simplified yet authentic Pandas Python programming experience, with a concise syntax and a notional machine that closely matches that of Pandas. This allows beginners to grasp the core concepts of data analysis without getting bogged down in complex syntax or unnecessary details. Indeed, its primary *pedagogical aid is to make programming more accessible to beginners thanks to a less complex notional machine and an easier-to-learn syntax than original Pandas*.

The transition from ToyPandas to Pandas should be quite intuitive and natural, as students are not asked to construct a new mental model from scratch, but rather to build on their existing one. For instance, creating a DataFrame from two Series using `ToyPandas` involves the following

[1] https://pyret.org
[2] https://github.com/data-8/datascience

code snippet:

```
s1 = series(2, 0, 5, 9)
s2 = series("Bob", "Alice", "Ted", "Carol")
df = concat(s1, s2)
df.show()
```

In particular, the commands are very similar to those in the original Pandas library, but reflect a simplified NM. In particular, in the original Pandas, series can be defined starting from iterables, dictionaries, or scalar values (e.g. with `np.arange()`), so the original command might be something like:

```
s1 = pd.Series([2, 0, 5, 9]|)
```

This approach assumes prior knowledge of basic Python data types such as lists. However, this can be confusing for beginners who are just starting to learn about series, as they may perceive series as similar to lists. In addition, the syntax is more tricky, requiring students to enclose square brackets (to define the list) within parentheses (to call the method). This can lead to a higher cognitive load, as beginners need to grasp multiple concepts at once in a short period of time.

To make series simpler and more like lists, we have hidden an important feature of Pandas, namely hashable indexes and multi-level/hierarchical indexes. In ToyPandas, series can only have numeric indexes, just like traditional Python lists. In addition, when an element is removed from the series, the resulting series still maintains a sequential order of numeric indexes. In contrast, the original library requires manual adjustments to fix the "holes" in the indexes, since they are treated as hashable keys in a map.

Another fundamental difference lies in the names of the data types. In our library, no prior knowledge of Python and NumPy iterable data types is required. Text and heterogeneous series are referred to by different names - "string" and "object" - instead of using a single identifier called "object" as in Pandas. Integer and floating-point columns are simply called "int" and "float" respectively, without specifying the bit size of the type in the column name (e.g., int32, int64, float64, etc.).

Indeed, from a syntactic and standard output perspective, we wanted to make ToyPandas simpler, more expressive and provide clearer semantics. For instance, we simplified the access to a dataframe element by not distinguishing between `.loc[]` and `.iloc[]` and naming it `.indexlocate[]` to emphasize the semantic of the statement. In addition, we have simplified the signature of several methods to use only basic data type of arguments and return value.

On the other hand, we keep the original Pandas syntax for Boolean conditions and data transfor-

mation pipeline idiom. For example:

```
df[df[(Age > 18) & (Name == "Alice")]].groupby('type').median()
```

This design choice stems from the need to maintain compatibility with existing learning tools, such as PandasTutor, to facilitate a white-box understanding of data manipulation concepts.

Iteration over rows is still implicit (like SQL queries)[3], and user-defined functions can still be applied to dataframe columns using our redefined `.apply()` method. Finally, we have overridden several Pandas methods for working with series and dataframes, including those for handling missing values, renaming columns, and removing duplicates.

From a technical standpoint, ToyPandas comprises a collection of subclasses derived from Pandas classes, encompassing Series, DataFrame, and pd.core.indexing._iLocIndexer, among others. These subclassed objects retain compatibility with Matplotlib methods, allowing for seamless data visualization tasks.

Finally, ToyPandas is still compatible with the visualisation tool PandasTutor (see Figure 6.1). This was a key objective, as it allows the NM to be visualised at the data manipulation level.

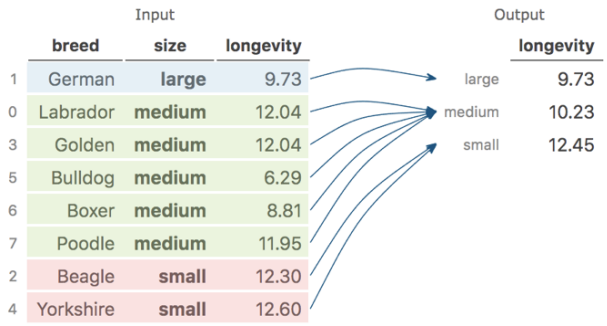

Figure 6.1: PandasTutor: a visualisation tool for Pandas.

6.3 PyPI Release and Future Work

We initially used a local version of ToyPandas during our teaching experiment with school students. Students had to import this local version into their Google Colaboratory repository as an external Python file script. However, to make ToyPandas accessible to a broader

[3]In particular, many Pandas operations are vectorised to improve performance, but this optimisation detail is beyond the scope of our project.

audience, we officially released it on the Python Package Index (PyPI) in July 2023. The released package, currently in version $0.2.0$[4], can be easily installed using the command `pip install ToyPandas`.

This release marks a significant step forward in our efforts to promote ToyPandas as a learning package for data-centric programming. By making ToyPandas widely available, we aim to empower educators and students worldwide to experience the benefits of data-centric computing.

Our future plans for ToyPandas include continued maintenance and updates to extend the library's capabilities and expand its documentation. We are also exploring the possibility of reimplementing ToyPandas using a wrapper class approach.

The feedback from our teaching experiment was generally positive. Students found ToyPandas to be easy to learn and use, and they were able to grasp the core concepts of programming and data analysis. Teachers also found ToyPandas to be a valuable tool for teaching data-centric computing. However, it is important to note that these results are preliminary and do not have statistical validity. Our experiment involved a small group of students, and further research is needed to confirm our findings with a larger and more diverse sample.

Our project has some limitations. For example, it does not support all the features of Pandas, and it is not as powerful as Pandas. Also, ToyPandas is still under development and there may be some bugs or missing features.

In conclusion, our preliminary evaluation suggests that ToyPandas is a promising tool for introducing beginners to data-centric computing. While ToyPandas shows promise, further research is needed to conclusively validate its effectiveness. The limited sample size of students in our experiment limits the generalisability of our findings, which currently positions this work as a valuable experience and technical report, paving the way for future, larger scale studies.

[4] https://pypi.org/project/ToyPandas/

Part III

Teaching Experiences and Educational Proposals

This part presents a curated collection of teaching experiments and proposals conceived and implemented during my PhD journey, based on the research findings presented in the previous part of this dissertation. Specifically, chapter 7 compares the data-centric approach to Python programming with other pedagogical approaches and presents three different methods for introducing students to the language; chapter 8 outlines a lesson designed for high school students that focuses on data visualisation using Tableau software; and chapter 9 presents a proposed course design for an introductory DS course that emphasises both coding and visualisation skills through the use of Tableau and Python. Although chapter 8 presents some qualitative results, it is considered an experience rather than an experiment, due to its highly experiential nature and a population very different (both in age and size) from those involved in the previous experiments presented in Part II.

Before presenting them, a brief overview is given of the research journey that led to their design and implementation. This overview includes other experiments that are not reported in this thesis, but which have contributed to a broader understanding of the research topic.

Prior research activities

Our first experiments were aimed at improving the educational offering of our CS degree programme and the orientation/outreach activities we provide to secondary school students, such as the "Stage of Informatica" programme. These experiments focused primarily on increasing students' motivation and engagement in learning to program, while also strengthening soft skills such as teamwork. We explored different strategies, including gamification and collaborative learning. The latter involved students from different backgrounds working together, mirroring real-world team settings. In particular, we investigated the TBL methodology with Sonic Pi, targeting first-year students enrolled in the CS1 course, and obtained preliminary results in terms of engagement and competence in programming. These results can be found in [186] and formed the basis for the more extensive experiments presented in chapter 4, which involved other courses and academic years.

Over time, the interventions have moved beyond teaching methods to focus on pedagogical approaches to CS and data education. Some experiments have been replicated several times over the years, others adapted to other age groups or courses, allowing us to refine and evaluate activities based on student feedback and learning outcomes. By actively soliciting students' feedback and suggestions (e.g. through anonymous questionnaires), we have been able to refine and improve the interventions according to their needs and preferences, and to systematically integrate these refined interventions into our undergraduate CS curricula or outreach programme activities.

To elaborate on our experiments, we integrated real-world problem-solving scenarios into the activities, providing students with practical challenges to apply theoretical knowledge in a meaningful way, and encouraging critical thinking and creativity.

Finally, in response to the Covid-19 outbreak, we also experimented with distance and blended learning. Many experiments were adapted for distance learning, while others were specifically designed for a distance learning context. For example, collaborative learning experiments were conducted to enhance social interaction, virtual reality (VR) was used to enhance social presence within virtual learning environments, and online gamification platforms were used to promote both engagement and collaboration, creating a playful and rich learning environment where students could learn, collaborate and experiment with social interaction with their peers and instructors. Our research into using VR technology to create more immersive and social learning environments has produced some preliminary results, which can be found in [40].

In parallel, we also experimented with other age groups. Although our target groups were undergraduate and high school students, we also experimented with primary and middle school students and senior citizens. For example, we designed several teaching units that were part of a PON project[5], aimed at children, to introduce CT and data thinking through coding and unplugged activities. We also designed an online challenge based on coding and CT concepts using the online edutainment platform Smart O.C.A.[6] This challenge was a unique opportunity to engage primary and middle school students across Italy in a fun and interactive learning experience while improving their understanding of computing concepts. The platform offered a series of engaging tasks that gradually increased in complexity, allowing students to hone their coding skills. A full overview of the activity and preliminary results can be found at [50].

At the other end of the age spectrum, we experimented with third-age students enrolled in an ICT course organised by UniGeSenior[7], a programme that empowers senior citizens to improve their digital skills. This collaboration provided valuable insights into the learning preferences and challenges of third age ICT students, enabling us to develop effective collaborative teaching strategies tailored to their specific learning needs. For a more detailed discussion, refer to [184].

[5] In Italian, PON stands for "Piano Operativo Nazionale".

[6] Smart O.C.A. (Online/Offline Challenge Activity): https://www.edutainmentformula.com/web-app/smart-oca/

[7] UniGeSenior: https://senior.unige.it

Chapter 7

Python for All: Three Alternative Approaches for Introducing Novices to Programming[1]

Designing an effective introductory programming course is difficult, requiring consideration of student characteristics as well as time and curriculum constraints. This challenge is even greater for introductory orientation and outreach activities designed to quickly engage students with no prior programming experience, to spark and foster their interest.

In this chapter, we present three Python programming modules for secondary school students, each consisting of about 15 hours of theory and practical teamwork. The units share core learning objectives, but their pedagogical approaches vary to suit different educational contexts. In particular, we have experimented with block-based coding, tinkering and data-centric pedagogy.

In the context of this thesis, it is interesting to analyse the data-centric approach in comparison to other pedagogical methods. This analysis serves two purposes: to gather initial student feedback on data-centric programming, and to propose a design for a short teaching unit based on this pedagogy. Individual post-questionnaire feedback shows positive results in terms of interest and engagement across all units. The Python micro:bit module appears to be particularly effective in promoting hands-on learning and engagement. In addition, the data-centric approach, while initially challenging, progressively engages students through real-world data manipulation.

[1]This chapter is based on [53] and experiment with university and high school student groups.

7.1 Introduction

This chapter introduces three compact Python programming modules tailored for educational guidance and outreach in secondary schools, targeting students with no prior programming experience. These modules, each approximately 15 hours in length over two days, include both theoretical and practical activities with the overall goal of achieving the following learning outcomes: at the end of the activity, students will be able to understand basic programming concepts and basic Python syntax and structure, including data types, operators, basic input/output, control structures, iteration, and functions.

While the learning objectives are consistent across the modules, the pedagogical methods vary, allowing for the exploration of different learning scenarios. The first unit takes a "traditional" approach based on imperative programming, but using a visual, block-based environment called EduBlocks [9]. A distance learning version has been developed, that also incorporates elements of gamification to make the unit more engaging when offered in remote learning environments. The second unit takes a physical computing and tinkering approach using the BBC micro:bit platform [140, 161]. Finally, the third unit ventures into the realm of data-centric pedagogy [95], using our own Python package (described in chapter 6) to streamline DS programming tasks.

Context & Contribution The presented teaching units were offered as part of CS orientation activites in different academic years, from 2019/2020 to 2022/2023. They were aimed at secondary school students without CS background. In total, across all editions, our experimentation involved more than 250 students.

These units can be used as inspiration for future introductory and orientation activities. In particular, the third unit also serves to present an experience of using data-centricity pedagogy with our ToyPandas educational library.

Structure of the chapter The remainder of the chapter is organized as follows. Section 7.2 provides a description of the first unit using the block-based imperative approach; Subsection 7.2.1 provides details of the gamified version of the first unit; Section 7.3 provides details of the second unit using the BBC micro:bit. Section 7.4 provides details of the third unit using the data-centric computing approach. Finally, Section 7.5 discusses the outcomes of the experiences.

7.2 A Block-Based Introduction to Python

The first teaching unit was an integral part of a broader CS outreach program in the 2022 and 2023 editions. In this module, students were introduced to Python programming using a traditional approach that emphasizes control structures and imperative programming. However, the

students used EduBlocks, a block-based educational platform designed to simplify programming by providing a visual interface that allows students to build Python programs by dragging and dropping blocks of code. It also facilitates a gradual transition to text-based programming by displaying the relevant Python syntax over each block. In addition, it streamlines understanding and code design by reducing the complexity of the instruction set. Together, these features allow students to focus on solving problems rather than getting bogged down in syntactical intricacies.

The unit consisted of eight short programming modules, each consisting of a brief instructional session and a hands-on group activity. The topics and final assignments for each module are listed in the appendix B.1.

7.2.1 Gamification during Covid-19

During the Covid-19 health emergency, in the 2019/2020 and 2020/2021 academic years, we adapted the teaching module to meet the needs of distance learning. Our approach involved the use of Smart O.C.A., an online edutainment platform, as the foundation of our virtual learning environment. Specifically, Smart O.C.A. is a video quiz game that can be played individually or in groups, and each game instance can be customized based on the target audience.

The introduction to programming was ingeniously structured akin to a tabletop game, where each programming concept was represented by four distinct game tiles. Refer to Figure 7.1 for a visual depiction of the game board in action.

More specifically, the initial cell for each topic encompassed a concise video lecture (derived from the presentation of the topic in the face-to-face unit) accompanied by quizzes designed to provide instant and automated feedback. Upon successful quiz completion, students unlocked access to the fourth section, which encompassed a practical task in EduBlocks. To encourage collaboration during the labs, we grouped students into small teams of 4-5. Students were instructed to use external video conferencing software for screen sharing and collaborative code development.

In cases where quiz responses were incorrect, students progressed to the second and third game boxes, housing supplementary quiz material. The final tasks, however, necessitated synchronous interaction and entailed manual assessment by the instructor. Consequently, students awaited their assignments to undergo evaluation. To facilitate this process, we engaged undergraduate students to partake in assessing assignments, fostering a culture of peer tutoring and support.

We incorporated the following gamification components:

- Real-time leaderboards: Students can see their progress and rankings in real time, fostering a sense of competition and motivation to perform better.
- Colorful gameboard: The game board is designed with vibrant colors and visually appeal-

ing graphics, creating an engaging and immersive learning environment.

- Multimedia Elements: We have incorporated video lessons to enhance the learning content and make it more interactive and engaging.
- Interactive, Collaborative, Timed Quizzes: Timed quizzes challenge students to think on their feet and reinforce their understanding of the material.
- Player Icons: Students can personalize their icons to make the learning experience more enjoyable.
- Levels: The use of levels creates a sense of progression and achievement, motivating students to progress through the learning journey. Each level corresponds to a programming topic.
- Points: Students earn points for completing assignments or quizzes, providing immediate feedback and encouraging active participation and learning.

Figure 7.1: Smart O.C.A. Python.

7.3 A Physical Computing Approach to Introductory Python Programming

The second unit takes a physical computing perspective on programming using the BBC micro:bit - a compact programmable device equipped with a microcontroller, an array of sensors

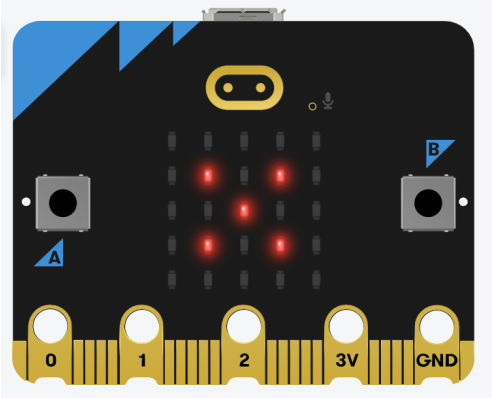

Figure 7.2: BBC micro:bit.

including buttons, an accelerometer and compass, a LED display, and wireless connectivity. The micro:bit supports a range of programming languages, from visual to textual languages, such as Scratch and Python. This approach allows students to create engaging interactive projects and see the immediate impact of their programs in the real world, adding a tangible and practical dimension to programming.

This unit follows the same sequence of programming topics as the previous one, but within an event-driven paradigm. In fact, students had to write simple programs to control the behavior of the micro:bit in response to certain events, such as key presses or motion detection.

Each programming concept was introduced through a concise lecture and then reinforced through hands-on work in a browser-based text programming environment[2], which is tightly integrated with the microbit library and provides code suggestions and device simulation capabilities. We formulated exercises using the idea of the micro:bit as a dice, as shown in the figure 7.2, which represents the five points on the dice. The tasks are listed in appendix B.2.

This teaching unit was experimented with secondary school students from a mechanical vocational school who had no prior programming experience. These students visited our university labs for the experience.

7.4 A *Data-Centric* approach to Programming

Building on the foundation of data-centric computing pedagogy outlined in chapter 3.2.3, the third unit served as a practical application of these concepts. This unit used short, interactive lectures designed to facilitate active learning. These lectures were complemented by hands-on programming activities using our educational library Toypandas, introduced earlier in chapter 6.

[2]Python Editor for micro:bit: https://python.microbit.org/v/3

The emphasis on data manipulation and analysis inherent in the data-centric approach necessitated some adjustments to the learning materials and the sequencing of topics. However, these changes were relatively minor and mainly involved the substitution of similar elements. For example, it was decided to exclude the while loop from the curriculum. This exclusion reflects the fact that dataframes, a core construct in data manipulation, rely heavily on implicit iteration, similar to the behaviour observed in declarative languages such as SQL [93]. A comprehensive list of topics and tasks covered in Unit Three can be found in the Appendix B.3.

Each topic was introduced with a combination of slide presentations and hands-on exercises using the CSV IMDb dataset[3] of movie information and reviews. Students began the process by importing, visualizing, and extracting insights from the data using various Pandas techniques such as head, shape, describe, and info.

They then delved into data transformation, which included eliminating duplicate columns, performing column reorganization operations such as renaming, and addressing missing values either by removing rows with null values or using imputation methods such as replacing them with averages.

Students then gained proficiency in data frame manipulation techniques, including slicing, selection, and extraction operations. They were also tasked with formulating a function capable of translating movies with a rating of 8.0 or above into the text label "good" and classifying the rest as "bad". These transformed labels were then used to create a new column. Applying this function to the column was accomplished using the apply() method.

To conclude their learning journey, students were introduced to data visualization and were able to apply several techniques, including scatter plots, histograms, and word clouds.

This teaching unit was experimented with secondary school students from a tourism vocational school who had no previous programming experience. These students visited our university labs for the experience.

7.5 Feedback and Discussion

After each activity, individual post-questionnaires were administered to the students. These questionnaires included multiple Likert scale questions that assessed students' perceptions of their level of interest, difficulty, usefulness, and support while navigating the field of computer science and the university environment. Additionally, three questions related to Python programming were included, covering topics such as Boolean conditions, iteration with for loops, and indexing with lists/series. Refer to Appendix B.4 for further details.

All of the different approaches to introducing coding seemed to yield positive results in terms

[3]IMDb dataset: https://datasets.imdbws.com

of student engagement. However, the second unit seemed to be the most engaging. From the evaluation questionnaires, we obtained the most positive results in terms of engagement. In addition, these results matched well with our perceptions during classroom observations. In fact, the micro:bit actively engaged students with a "learning by doing" approach that is well aligned with the principles of constructionist learning theory.

Conversely, the data-centric pedagogy initially presented a higher level of cognitive load that could potentially deter students. Even though our library simplified the syntax, many students initially struggled with series. However, data-centric programming quickly became an engaging and accessible experience through hands-on manipulation of real-world data. As a result, the final questionnaire revealed that students found it more challenging, but generally in line with their interests.

Finally, the imperative, block-based approach seemed generally less effective in terms of engagement, likely due to the lack of a tangible real-world artifact. In addition, the gamified version showed comparable results with respect to the imperative version, suggesting that the gamification elements effectively offset the somewhat less engaging remote environment.

When we analyzed the three programming questions on the final questionnaire, we found comparable results across units. This finding was somewhat unexpected, as we had anticipated that the different approaches, offering different perspectives on the topics, might introduce a slight bias in the learning outcomes. However, this was not the case, suggesting that these units can be considered equivalent in terms of learning outcomes and can therefore be used in different learning contexts. For example, the results of the data-centric pedagogy did not outperform the others on Boolean and indexing questions, despite the extensive use of these concepts in manipulating data frames. Similarly, students performed consistently on the for-loop question even though they used more implicit iteration.

Overall, the anonymous, open-ended feedback indicated that the activities served as a valuable orientation tool. Some students found out through our educational guidance activity that CS was not their calling, while others were eager to explore CS further, possibly extending their studies or pursuing a career in the field.

Chapter 8

Data Literacy in High School: An Experience in Introducing Data Science using Tableau[1]

This chapter builds upon the motivations and challenges for introducing DS in schools, as discussed in Chapter 3.2.1, by presenting two teaching units designed to enhance data literacy among high school students.These units use Tableau software[2] and emphasise data evaluation, particularly through data visualisation. Authentic data sets are used, including those related to the Olympic Games and global economic freedom. The first unit includes a two-hour module dedicated to data exploration and preparation, while the second unit focuses exclusively on data visualisation. To provide context, a comprehensive literature review has been presented in Chapter 3.2.4. Our main goal was to give students an authentic experience with DS, foster data thinking, and to make them aware of the importance of data in developing a better society.

The instructional design follows the latest data literacy standards, in particular the Data Literacy Competence Matrix [147] and the European DigComp framework[3]. The latter provides a common understanding of digital literacy.

8.1 Population and Setting

We proposed two teaching units in two different school years (2021/22 and 2022/23). The total duration of the first and second units was roughly 19 and 21 hours respectively, spread over one

[1]This chapter is based on the material published in [66, 67]

[2]Tableau: https://www.tableau.com

[3]DigComp: https://joint-research-centre.ec.europa.eu/digcomp_en

month. Both were aimed at grade-11 students of a high school in humanities (*"Liceo Classico"* in Italian)[4] and were part of a school *Pathways for Soft Skills and Educational Guidance* (known in Italy as P.C.T.O.) activity. In particular, students in the second unit were enrolled in the *European Classical Lyceum* curriculum, which, compared to the traditional one, includes the study of economics, law and two foreign European languages, as well as the strengthening of scientific disciplines.

The population of the first unit consisted of 28 students divided into 10 groups (2-3 students per group). Because of Covid-19, the lesson was conducted remotely via a video meeting platform. Students did not receive a Tableau licence, but were able to use the 12-day free trial of Tableau Cloud.

The population of the second unit consisted of 42 students from two different classes. Each class was divided into 6 groups (4-5 students per group). The activity was carried out face-to-face in a computer laboratory at the Department of Computer Science of the University of Genoa and focused on data visualization, using the free platform Tableau Public.

For both activities, we did not consider programming-based solutions such as Python or R, as the students had no previous experience of computer programming and this would have required a disproportionate cognitive effort in such a short period of time.

At the end of each edition, we collected all the group projects and multimedia presentations. In addition, a satisfaction questionnaire was sent to the students. All analyses of the Likert data were done using Tableau software.

8.2 Unit Overview

The teaching units consisted of a frontal lesson introducing data, a lab lesson using Tableau, a team project, to which students are assumed to devote 15 hours over a month, and a final meeting where student teams could share and discuss their findings. Our unit led students with no previous experience in data literacy to perform basic analysis and visualisation on a real dataset.

The units had the following learning objectives according to the Data Literacy Competence Matrix: perform data exploration; identify anomalies and outliers and perform basic data cleaning operations (Teaching Unit 1 only); develop an analysis plan; create meaningful graphical representations of data; read and understand graphs, maps, and charts; draw information from data; recognise the importance of data and be aware of ethical issues related to data.

[4]The Italian *"Liceo Classico"* (literally *"Classical Lyceum"*) is a high school with a focus on classical studies, where students study both Latin and Ancient Greek, with comprehensive curricula in philosophy, literature and history.

8.2.1 Teaching Unit 1

The first unit (school year 2021/22) consisted of a total of 6 hours of theory and practical activities, plus a group project and a final discussion. At the end of the activity, the students were also given a satisfaction questionnaire about the topic of the activity and the group work. In what follows, we describe in more detail of the different stages of the course.

First Lesson (4 hours) The first lesson started with a brief introduction to data using practical examples from everyday life. We defined basic data types (textual, numerical, location), variables (quantitative, ordinal, and categorical) and tabular data, before introducing students to big data and data science. We proposed some interactive quizzes in Wooclap[5] followed by a plenary discussion in order to actively engage students. For example, we asked them to describe big data in one or two words, and then we used the students' answers to create a word cloud. This simple exercise allowed us to actively introduce students to data collection (their responses), visualisation (word cloud as a visual representation of textual data) and analysis (word cloud interpretation). We then explored the concept of data analysis and data-driven decision making, including a reflection on data ethics. Finally, we introduced students to the topic of data exploration and transformation.

The second half of the lecture was devoted to the hands-on activity in Tableau using a toy dataset provided in Tableau's online tutorials. We performed some basic data preparation operations (e.g. identifying useful data, removing duplicate columns, identifying/correcting typos or inadmissible values, combining tables together, etc.). We adopted a learning-by-doing approach: the students, guided by the teacher, were able to work in groups and replicate the operations on their own machine.

Second Lesson (2 hours) The second lecture focused on data evaluation, in particular data analysis, visualisation, and interpretation. Starting from the output of the data preparation flow, we performed a simple data analysis to generate some visual representations of the data (e.g. interactive geographical maps, bar charts, pie charts, histograms, line graphs, etc.). A lot of time was spent on data interpretation and data storytelling to present the results. We discussed that it is possible to visualise data in different ways and that some visualisations are more appropriate than others for different use cases. Finally, we presented the requirements of the group project.

From a technical perspective, we created Tableau sheets (one plot per sheet) and combined them into a Tableau dashboard where interactivity could be added to enhance the user's insight into the data. For example, students could add a filter to analyse a specific subset of data.

[5] https://www.wooclap.com

Team Project and Final Presentation (expected effort 15 hours, deadline 1 month) Students had to prepare a project on a real dataset about the modern Olympic Games. The dataset was an unclean version of the original Olympics dataset[6]. With our modifications, it could also be used for basic data cleaning practices (we duplicated some columns, created dirty data, and inserted null tuples). The dataset included all the Games from Athens 1896 to Rio 2016 and contained more than 271 thousand rows and 15 columns. Each row corresponded to an athlete competing in an individual Olympic event and contains basic bio data on athletes and medal results. In particular, the NOC column (National Olympic Committee, 3-letter code) provided the opportunity to join another table. We suggested a number of websites where students could download additional datasets (e.g. kaggle.com), but this was optional for the project. The Olympics dataset offered students the opportunity to ask questions about how the Olympics Games evolved over time, including questions about the participation and performance of women, different nations, sports, events, etc.

Students had to identify a research question and, on the basis of such question, clean and prepare the data, create appropriate charts and graphs, and create an interactive dashboard to interpret and present the phenomenon. Finally, they had to prepare a multimedia presentation illustrating the research activity and the results obtained, the encountered difficulties and possible future directions, as well as the dynamics of team work and project management (tasks, deadlines, roles, etc.). Finally, they had to reflect on some aspects of data application for civic responsibility. Due to the trial period of Tableau Cloud, the students had 12 days to work on the data platform and the remaining days to work on the multimedia presentation. Students were provided with guidelines and teaching materials (class slides, dataset, and lecture recordings). In the final session, the teams presented their findings to the whole class.

8.2.2 Teaching Unit 2

The second teaching unit (school year 2022/23) consisted of 4 hours of theory and practical activity, the team project and the final discussion. The activity was structured similarly to the previous one, but with a different dataset and without the data preparation part, as the duration of the module was reduced to 4 hours (instead of 6).

Lesson (4 hours) As before, we introduced students to data literacy, (big) data, and data science. We focused on data evaluation and on the ethical and privacy aspects of data. We spent more time on data storytelling - how to present and communicate the results of data. Again, we offered interactive quizzes on Wooclap. The hands-on activity with Tableau Public Edition took place in our university labs and focused on aspects of data visualisation. For example, we discussed the choice of graph depending on the type of data, as sometimes there is more than one

[6] https://www.kaggle.com/datasets/heesoo37/120-years-of-olympic-history-athletes-and-results

possible graphical representation, and we looked for graphically effective communication strategies. Finally, we put it all together in a dashboard and discussed the result. We practised directly on the dataset they would later use in the group project phase. The data exploration phase was then guided by the teacher.

Team Project and Final Presentation (expected effort 15 hours, deadline 1 month) The students had to work in teams and prepare a data science project with a focus on data visualisation. As the school explicitly requested a legal/economic domain (as the classes were from the *European Classical Lyceum* curriculum), we chose the Economic Freedom Index dataset freely distributed by the Heritage Foundation[7]. It measures the economic freedom of 184 countries based on trade freedom, business freedom, investment freedom, and property rights. Specifically, we combined five years of observations from 2018 to 2022 into a single dataset to allow students to analyse the global economic prosperity over the past five years. In this edition, the school also asked us to assign a research question to each group, and we suggested the following questions:

1) Over the five-year period, how has the Italian economic freedom evolved? 2) Has the global scenario of economic freedom changed between 2018 and today? 3) Has the Covid-19 pandemic had an impact on economic freedom (globally and in 4 other countries of your choice)? 4) How does economic freedom vary in different areas (Europe, America, etc.)? 5) How will countries' economic freedom change in 2022? Compare max and min; 6) How does Europe's economic freedom compare with Russia's? Questions were the same for both high school classes.

Based on the assigned research question, students had to generate appropriate charts and graphs and create an interactive dashboard to interpret the phenomenon. Finally, they had to prepare a multimedia group presentation with the same design requirements as in Teaching Unit 1. In the final session, the teams had to present their results in plenary.

8.3 Preliminary Results

At the end of each edition, we collected team projects and responses to a final satisfaction questionnaire. The questionnaire was individual and consisted of the following questions: (DS) *How interested were you in the unit?*; (H1) *How challenging was it?*; (H2) *How much fun was it?*; (H3) *How useful was it?*. All items were rated on a Likert scale from 0 to 3, with options ranging from "not at all" to 'very much" (note that H1 is a reverse item).

In particular, the first edition had a population of 28 students, while the second edition had 42 students. The first session was held remotely due to the Covid-19 health emergency, while the

[7] https://www.heritage.org/index/

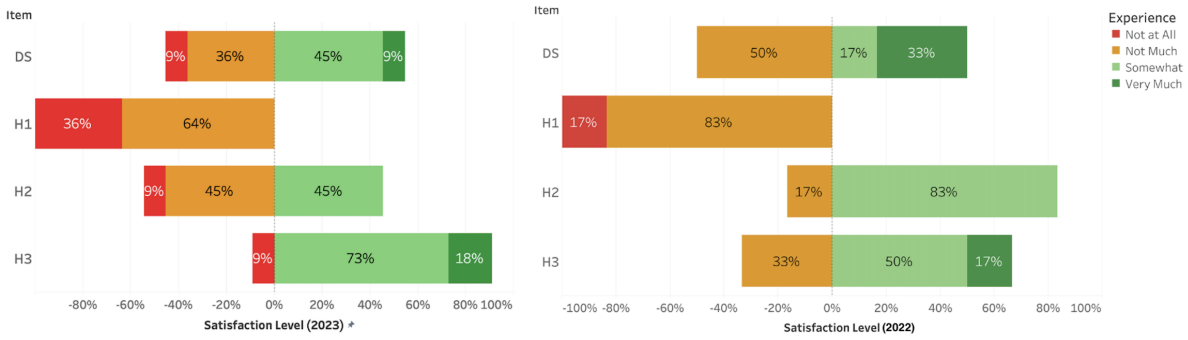

Figure 8.1: Gantt bar chart of Likert scale survey data (2023 vs. 2022 edition).

second session was held in person in a computer laboratory at the Department of Computer Science of the University of Genoa. The team projects were assessed according to the following evaluation criteria: presentation quality and time management (multimedia presentation/slides, adherence with assignment deadline); data analysis (number of variables used; types of data operations such as drill-down, filtering, etc.; the depth and structure of the analysis in relation to the original research question); communication with data (effective use of maps/charts, presence of labels/legends, interactivity in dashboards, etc.), and domain-knowledge (quality of in-depth study, cited sources, etc.). The scores for each dimension were given in decimals, and the final score was calculated as a weighted sum of the different criteria. Presentation, data analysis, and communication had the same weight (0.3), while domain knowledge had a weight of 0.1.

In the first edition, all team projects were of good quality and students seemed to appreciate and grasp the potential of a data-driven approach as well as its application in their personal and professional daily lives.

In the second edition, the quality of projects has been more erratic. In total, we assessed 11 team projects. The mean was 6.9 and the median was 7. More specifically, we gave three groups a mark of 6, six groups a mark of 7, and two groups a mark of 8.

Figure 8.1 shows the results of the final questionnaires. Only 6 and 11 students in 2022 and 2023 respectively completed the questionnaire. The median of the item DS is 1.5 [2 in 2023], while the medians of the three items related to the homework (H1, H2, H3) are 1, 2, and 2 respectively [1, 1 and 2 in 2023]. In general, the activity was perceived as difficult in both editions (all H1 responses were negative), but more so in the second, although it was perceived as more useful. The level of interest in the activity is quite similar, although the first edition had a higher percentage of very positive responses. We did not conduct a statistical control between the two sessions as several variables were changed (remote vs. face-to-face, data cleaning phase, dataset change, pre-set research question), some of which were beyond our control.

In an informal way, we can measure the variability between the two editions based on the anonymous free comments from the 2023 questionnaire. Specifically, about half of the participants expressed a preference for working with a different dataset (one that is not economic/legal) and/or having the opportunity to propose their own dataset. Two respondents would also have liked a lesson on multimedia presentations. Finally, one person suggested splitting the theory and practical sessions into separate days. We believe that a data-driven activity will spark curiosity and motivation if the topic is of interest to the student, so in the future, we will propose a number of datasets to choose from and leave the students free to devise their research questions.

8.4 Limitations and Future Directions

From the first experiment, we realized that the data preparation part required more technical skills than the data visualization part and therefore could be more challenging in those schools where the IT discipline is not present (e.g. in Italy, the high school specializing in classical studies). For this reason, and due to time constraints, we have preferred to focus on data visualization in the second edition of the course.

Another aspect concerns the deadline for the final project. In the first edition, all groups met it, but in the second edition, almost all were late. This could be because Tableau Cloud has a non-renewable trial period, while Tableau Public has an unlimited period. This difference may have affected the students psychologically. In addition, their school's pedagogical approach may not have accustomed students to team projects with deadlines, contributing to the delays.

We believe it's essential to establish a dialogue with the school's math and computer science teachers to align with the class's learning outcomes. Unfortunately, this was not possible in our case, and it had repercussions in the second edition. Students lacked sufficient statistical knowledge, such as the concept of median, and had difficulty identifying when to use the mean. This affected the data analysis process.

In addition, we strongly disagree with the 2023 school's requirement that the research question be stated. We believe that allowing students to pursue inquiries prompted by their curiosity can have a positive impact on motivation and creativity. In the second iteration, students seemed to view the team project as a mere obligation, like an extra homework assignment. This is reflected in the project results, which were noticeably less elaborate and engaging than the first edition. In the first edition, the projects demonstrated higher quality data analysis and produced more valuable data insights.

Furthermore, we recognize that the domain of the dataset could influence students' interests and curiosity. Therefore, we advocate that the dataset be chosen in collaboration with students to increase engagement and excitement.

Regarding the questionnaire in Teaching Unit 1, a limitation is the sample size, which is too

small and therefore not statistically significant (only 6 students responded, probably because the survey was optional).

From a tool perspective, Tableau, with its friendly user interface and real-time suggestions, appeared to be a very powerful and effective data tool with a short learning curve for introducing data literacy to novices.

Furthermore, we believe that the proposed approach could be experimented/extended in other types of secondary schools. As a future direction, we would like to experiment with data ingestion and data preparation using Tableau Cloud in a technical institute specialising in computer technology[8] and experiment with Tableau Public in some science-oriented high schools (*"Liceo Scientifico"* in Italian)[9] to present predictive models (e.g. linear or logistic regression) and analyse data distribution (e.g. with box plots or scatter plots). Finally, we are working on a 2-hour module on building data views for accessibility to raise students' awareness of web content accessibility for people with a wide range of disabilities. In this respect, Tableau seems to be an effective platform for this, as its elements are accessible by design and comply with WCAG 2.0 AA.

[8]In Italy, the technical institute is a type of secondary vocational school.

[9]The *"Liceo scientifico"* (literally *"scientific lyceum"*) is a type of Italian secondary school that emphasises scientific culture, particularly mathematics, physics, chemistry, biology and sometimes, depending on the chosen track, computer science.

Chapter 9

A Data Science Course for Non-Computing Majors

This chapter proposes a formal 6-credit course introducing DS fundamentals to students with no CS background.

The proposal draws on experiences and experiments conducted during the Ph.D., which are detailed in previous chapters. In particular, our experience in data education has informed the exploration of visual tools such as Tableau for data analysis and visualisation activities. This approach facilitates an intuitive and practical introduction to the DS lifecycle for beginners.

In addition, experimentation with innovative teaching methods highlighted the value of fostering active and collaborative learning environments. These strategies not only increase student engagement, but also have been shown to improve learning outcomes.

Finally, experiments in programming highlighted the importance of tailoring activities to students' individual needs and interests, which ultimately supports the development of robust mental models of the underlying concepts.

This course proposal aims to address the challenges outlined in section 3.2.2 by providing students with an introduction to DS that prepares them to both "think with" and "work with" data, promotes data literacy, and emphasises practical applications through the use of data visualisation tools and easy-to-learn programming environments. To provide relevant background information, a comprehensive literature review has been presented in section 3.2.4.

The remainder of this chapter is structured as follows: Section 3.2.2 presents the motivations and context for the course, followed by a review of related work in Section 3.2.4. Section 9.1 details the educational setting, while Section 9.2 outlines the instructional design choices employed. Finally, Section 9.3 discusses limitations and potential avenues for future work.

Appendix C.1 presents the detailed lesson plan.

9.1 Educational Setting

The IDS course proposal is designed as a 6 ECTS semester course aimed at third year undergraduate students majoring in Digital Humanities. These students have no background in programming or statistics. The main goal of the course is to introduce students to different tools and methods to perform simple data analysis and visualisation tasks. Specifically, students will explore the main aspects of the DS lifecycle, covering data exploration, cleaning, manipulation, analysis and visualisation, through in-class demos and hands-on collaborative activities based on data visualisation tools and Python programming. In addition, the course will focus on effective communication with data and encourage reflection on the ethical aspects of DS. The following are the objectives and learning outcomes of the course.

Learning objectives The course aims to foster a data-driven culture among students, including data ethics, data visualization and analysis skills, as well as technical skills to perform basic DS tasks. In particular:

- Introduce students to fundamental DS concepts and techniques.
- Provide a practical understanding of the DS lifecycle.
- Develop essential programming skills for data manipulation and analysis.
- Encourage effective communication of data insights.
- Promote critical reflection on ethical issues in DS.

Learning outcomes By the end of the course:

1. Students will be familiar with a variety of DS tools and methodologies.
2. Students will be able to carry out basic data tasks, including data exploration, analysis and visualization.
3. Students will be able to effectively present data insights using data visualization and storytelling techniques.
4. Students will demonstrate awareness of ethical issues related to DS.

5. Students will develop expertise in data visualization and effective communication using Tableau software.

6. Students will develop proficiency in Python programming using the Jupyter notebook environment with a focus on data-centric tasks using both the standard library and popular DS libraries such as Pandas and Matplotlib.

Exam Modality

- Attendance Mode:
  - Tableau Assignment (maximum 2 points impact on final grade).
  - Final project: Group project on a real-world dataset using the Python programming language.
- Non-attendance mode:
  - A final individual project on a real-world dataset that requires proficiency in both Python and Tableau.
  - An oral exam consisting of 2 open-ended questions on key concepts (e.g., discuss the DS life cycle and provide a detailed description of each stage, discuss data ethics and its consequences in cultural domains, etc.).

Evaluation For evaluation criteria, please refer to the evaluation grids provided in the Appendix.

9.2 Instructional Design Choices

This section outlines pedagogical strategies and tools. The main goal is to create an inclusive and effective learning environment that accommodates students with diverse backgrounds and learning preferences.

9.2.1 Tools and Learning Environment

Tableau provides an easy-to-use environment where students can focus on data thinking without getting bogged down in syntax and coding challenges.

In essence, Tableau serves a similar purpose as Scratch or other visual tools in introductory programming, allowing for a smooth transition to more advanced concepts. In addition, Tableau's industry ubiquity provides students with foundational skills in a widely used data visualisation platform. Building on an introduction to data visualisation with Tableau, the course then equips students with CT skills through Python programming. This approach encourages active learning, creativity and a deeper understanding of data analysis.

Python, chosen for its beginner-friendly syntax, extensive libraries for data manipulation, machine learning and visualisation, and vast online resources, serves as the foundation for this course. The use of Jupyter notebooks within Google Colaboratory, a cloud-based environment, eliminates local setup and provides easy access to libraries, collaboration features, and powerful computing resources. This interactive platform allows students to write and run Python code directly in their web browser, providing a seamless learning experience.

In addition, classes integrate an interdisciplinary perspective and include a significant hands-on component through in-class demonstrations and team-based, project-focused labs that tackle real-world problems. This approach equips students with both theoretical knowledge and practical skills.

9.2.2 Notional Machine

A key challenge in designing this course for non-CS students is determining the optimal depth of programming language coverage and the minimum skill level required for success. It's important to emphasise that the course emphasises essential Python skills for basic data tasks, rather than a comprehensive programming expertise. It should not be seen as a replacement for a traditional CS1 course.

Following a data-centric approach inspired by [28], the course prioritizes data tasks over language constructs. For instance, instead of focusing solely on loops, students learn how to iterate through data elements to achieve specific data manipulation goals. Each operation is accompanied by sample code that serves as a practical guide for solving similar problems.

The course begins with a brief, hands-on introduction to the DS lifecycle in Tableau, and then moves on to script programming in Python. First, students learn standard Python operations and constructs to perform basic data operations, such as importing CSV files, using iteration (for loops), and conditional statements. Students are then introduced to the Pandas and Matplotlib libraries, focusing on features of the language that are fundamental to working with data, particularly tabular data structures. By introducing basic Python first, and then the libraries, students can appreciate the benefits of using Pandas for data tasks, while building a solid foundation in general Python programming.

Students will also be introduced to machine learning through the use of the `scikit-learn`

library. Note, however, that our approach is to present machine learning algorithms using only the call methods provided by the library, without delving into the details of the algorithms. While black-box learning is generally discouraged, we believe that it fits perfectly with the goals of this course, where students do not have a solid background in mathematics.

While ToyPandas can be a valuable tool for engaging students and introducing basic concepts of data manipulation, it's important to recognise its limitations. Designed for short introductory activities, ToyPandas may not be suitable for a full university course. We recommend using the Pandas library directly, or using ToyPandas as a stepping stone from Tableau to Pandas before moving on to the full capabilities of the latter.

9.2.2.1 Dataframes

The core of the Pandas library is the dataframe abstraction, which represents tabular data and is directly related to the commonly used CSV file. Libraries built around this abstraction provide immensely powerful functionality, enabling complex transformations with simple code. Using Pandas can greatly simplify any DS project, so it is essential that students learn to use it effectively by the end of the course.

DataFrames are introduced in Week 5, allowing students to revisit problems and see how much easier solutions become with them.

It's worth noting that certain individual code statements in Python Pandas can be difficult to understand, especially with long data operation pipelines that encourage the use of programming idioms capable of performing multiple operations within a single condensed statement. While this approach can be powerful, it can obscure intermediate steps and turn the underlying process into a black box.

To this end, we include PandasTutor[1] throughout the course lectures to visualize the annotation table transformations. In fact, students can observe and understand how Pandas manipulates their data step by step, facilitating a clearer understanding of the transformations applied at each stage.

9.2.3 Team Formation

For face-to-face students, teamwork will be a compulsory part of the designated laboratory activities throughout the semester and will also be integrated into the final project. We propose the formation of homogeneously (i.e. well-balanced) heterogeneous groups [30]. These groups will be heterogeneous in the sense that they will be composed of members of different genders, nationalities and backgrounds, such as previous experience and digital technical expertise. In addition, course tutors may identify other relevant criteria to ensure an optimal mix within each

[1] PandasTutor: https://Pandastutor.com

team. This approach aims to capitalise on the unique strengths of each individual, while still allowing for meaningful comparisons between teams in the final project.

We recommend utilizing the GRumbler algorithm, developed by Harvard University[2], to form groups. This algorithm effectively handles ordinal categorical variables; therefore, any quantitative variables should be transformed appropriately.

9.3 Limitations & Future Work

In this chapter, a proposal for an introductory DS course for non-CS students has been presented. Although the proposal derives from my own PhD research activity, the whole framework has not been empirically validated, so it is not possible to present insights. Without empirical data, it is difficult to ascertain how effectively the proposed curriculum and pedagogical approach supports the learning objectives of the target students.

In fact, the learning framework could potentially present challenges for both students and educators. In particular, while the integration of Tableau and Python appears promising, the extent of knowledge transfer between them and the subsequent applicability of skills in real-world scenarios remains untested. It becomes mandatory to guide students as they transition from Tableau's visual framework to Python's coding paradigm. While the literature related to transfer scenarios similar to this one (e.g., moving from visual and block-based programming languages to text-based languages) demonstrates generally positive transfer results [192], there is no dedicated study specifically addressing the transition from Tableau to Python. As a future direction, we aim to investigate knowledge transfer between Tableau and Python.

In summary, these limitations underscore the need for empirical validation, adaptability, and continuous refinement. Future directions should include empirical evaluations, assessments of skill transfer, and the long-term sustainability of the course in the changing landscape of DS and digital humanities.

[2]GRumbler algorithm: https://scholar.harvard.edu/files/msparrow/files/grumbler-instructions.pdf

Chapter 10

Conclusions

This thesis presents a series of pedagogical experiments and proposals aimed at improving the learning experience in concurrent programming and data science education. It investigated the potential of innovative teaching methodologies, tools, and programming practices that incorporate a constructionist learning approach to improve the learning experience.

The experiments with Sonic Pi, detailed in Chapter 4, demonstrated significant promise in addressing the RQ outlined in Chapter 1. These experiments effectively engaged students, and facilitated the introduction of concurrency concepts through a playful and familiar domain, with a simplified NM that also presented the challenges of the topic, such as data races and deadlocks. After successful pilot runs in various undergraduate courses, we intend to systematically introduce this activity to first year students in the Computer Architecture course. This approach will introduce and preview an advanced topic, stimulate students' interest, and prepare them for the complexities of concurrent programming that they will encounter in later coursework.

We also plan to present this activity to ICDD students as evidence of our ongoing educational research and as an example of how to design effective teaching modules. This activity covers several key aspects of successful pedagogy, including the selection of an appropriate programming language, the identification and analysis of common student misconceptions, and the development of engaging and interactive learning experiences.

Finally, we aim to present this activity to secondary school teachers as a potential tool for introducing concurrent programming to younger students. We believe that the constructionist approach of the Sonic Pi platform can effectively introduce basic concepts of concurrent programming in an engaging and accessible way. We acknowledge that this study represents a first step in developing effective pedagogical approaches to concurrency education. Further research is warranted to explore the long-term impact of this approach on student learning outcomes and to refine the Sonic Pi-based activities for wider applicability.

Similarly, the experiment described in Chapter 5 produced encouraging results in terms of learning objectives and student engagement, demonstrating its practical application in database education. As a result, the TPS teaching methodology has been successfully incorporated into our database course at the University of Genoa, particularly in the data modelling laboratory sessions and applied exercises. We also intend to retain the team peer review activity for the conceptual design phase of the final project, as it promotes collaborative learning and the development of critical thinking skills.

Experiments with Python have provided valuable insights into different pedagogical approaches to introducing programming. Each of these approaches has its own strengths and limitations, so the choice of methodology depends on the specific characteristics of the target audience. In terms of data-centric pedagogy, our ToyPandas package seems to support this, and we intend to experiment with this in other future activities.

This research serves as a pilot study investigating different strategies for introducing programming to beginners. The results provide a valuable basis for the development of future iterations or activities within introductory programming courses, such as those offered by the University of Genoa's "Stage di Informatica" programme. In addition, these experiments can serve as a reflective tool for the CS1 course, providing valuable insights into approaches that can foster students' curiosity and motivation.

Similarly, the data literacy units designed for high school students can serve as a basis for our data-related courses offered at undergraduate and graduate levels, and their results played a key role in the construction of the course proposal presented in Chapter 9. Furthermore, the author of the thesis, who is now taking on a new role as a CS teacher at a commercial vocational school, has partially adopted this approach to design a learning unit ("Unità di apprendimento" in Italian) for his students. This unit focuses on the analysis of website data using the online data visualisation tool Google Analytics, using the Google Merch dataset provided by Google[1]. The interdisciplinary unit also included mathematical-statistical and marketing concepts. While there is currently no data available to assess the effectiveness of this approach in terms of learning outcomes as the unit has not been completed, initial observations seem to indicate a promising trajectory as students have shown increased levels of engagement and interest. Finally, we express our hope that the proposed IDS course will be implemented in the curriculum of the University of Genoa in the forthcoming academic years.

Although the reported experiences are primarily based on preliminary qualitative findings, they have significant potential to address the research question. The use of visualisation tools such as Tableau, rather than our ToyPandas library with its simplified NM, alongside group projects using real-world datasets, is a promising way to promote DS thinking. Integrating these elements into even short unit modules has the potential to create authentic DS experiences that cultivate

[1]Google Analytics - Demo Account: https://support.google.com/analytics/answer/6367342?hl=en

data-driven thinking in students.

As part of our future research, we would like to experiment with our data activities with high school students. This will involve adapting our existing materials and methods to make them accessible and engaging for a younger audience. We believe that this will help to stimulate interest in data systems and related fields among high school students and encourage them to pursue further studies in these areas.

Finally, although our focus has been on data modelling, we aim to explore and identify the main misconceptions surrounding data query languages (e.g., SQL) at both university and school level. We believe that it will help us to improve the teaching provision and develop more effective teaching methods and resources for database education. This will involve conducting surveys and interviews with students to understand the challenges they face when learning and using SQL. As part of the work of designing the CS curriculum of the school[2] where the author currently works, and based on the general and professional competences indicated by Indire[3], the author of this thesis has decided to introduce databases and SQL into his teaching. He would like to start some preliminary research into students' misconceptions about basic SQL concepts.

Lastly, we aim to validate an active learning methodology specifically designed for application-oriented logical database design. This methodology was introduced to undergraduate students enrolled in our database course. The experiment investigated the feasibility of using our methodology to teach advanced topics such as data redundancy and aggregates, which are fundamental concepts in many NoSQL databases. While the initial results are encouraging, further refinement and empirical validation are needed to solidify the effectiveness of this approach.

Limitations

Our research has several limitations that should be considered. First, our experiments were conducted on a small scale with limited student demographics. This makes it difficult to generalize our findings to broader populations. Second, our research did not always involve a formal evaluation of the effectiveness of our teaching methods. This means that we cannot definitely say whether our methods were more effective than traditional teaching methods. Third, our research did not focus on long-term student outcomes. We do not know whether the positive effects of our teaching methods persisted after students completed our courses. Finally, the decision not to use randomised controlled trials as a research method because of ethical and moral concerns may introduce another layer of bias and affect the generalisability of our findings.

Despite these acknowledged limitations, our research provides valuable insights into the potential of innovative teaching methods in computer science (CS) and data education. These findings

[2] Istituto Scolastico Casaregis

[3] Professional Area Skills for Commercial Vocational Schools, General Area Skills for Vocational Schools

provide a strong foundation for further research, and we believe they hold great promise for improving CS education practices.

Bibliography

[1] Samuel Aaron, Alan F. Blackwell, and Pamela Burnard. The development of Sonic Pi and its use in educational partnerships: Co-creating pedagogies for learning computer programming. *Journal of Music Technology and Education*, 9(1):75–94, 3 2016.

[2] Samuel Aaron, Dominic A. Orchard, and Alan F. Blackwell. Temporal semantics for a live coding language. In Alex McLean, Michael Sperber, and Henrik Nilsson, editors, *Proceedings of the 2nd ACM SIGPLAN international workshop on Functional art, music, modeling & design, FARM@ICFP 2014, September 1-3, 2014*, pages 37–47, Gothenburg, Sweden, 2014. ACM.

[3] Harold Abelson and Andrea A. diSessa. *Turtle geometry: The computer as a medium for exploring mathematics*. MIT press, 1986.

[4] Joel C. Adams. Creating a balanced data science program. In *Proceedings of the 51st ACM technical symposium on computer science education*, pages 185–191, 2020.

[5] Ridha Alkhabaz, Zepei Li, Sophia Yang, and Abdussalam Alawini. Student's learning challenges with relational, document, and graph query languages. In *Proceedings of the 2nd International Workshop on Data Systems Education: Bridging education practice with education research*, pages 30–36, 2023.

[6] Ridha Alkhabaz, Seth Poulsen, Mei Chen, and Abdussalam Alawini. Insights from student solutions to mongodb homework problems. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*, pages 276–282, 2021.

[7] David Alvargonzález. Multidisciplinarity, interdisciplinarity, transdisciplinarity, and the sciences. *International studies in the philosophy of science*, 25(4):387–403, 2011.

[8] Susan A. Ambrose, Michael W. Bridges, Michele DiPietro, Marsha C. Lovett, and Marie K. Norman. *How learning works: Seven research-based principles for smart teaching*. John Wiley & Sons, 2010.

[9] Anaconda. Edublocks by Anaconda: Blocks to Text made easy. Accessed on: August 18, 2023.

[10] Paul Anderson, James Bowring, Renée McCauley, George Pothering, and Christopher Starr. An undergraduate degree in data science: curriculum and a decade of implementation experience. In *Proceedings of the 45th ACM technical symposium on Computer science education*, pages 145–150, 2014.

[11] Daniel Asamoah, Derek Doran, and Shu Schiller. Teaching the foundations of data science: An interdisciplinary approach. *arXiv preprint arXiv:1512.04456*, 2015.

[12] Paolo Atzeni, Francesca Bugiotti, Luca Cabibbo, and Riccardo Torlone. Data modeling in the NoSQL world. *Computer Standards & Interfaces*, 67:103149, 2020.

[13] Eleonora Barelli, Michael Lodi, Laura Branchetti, and Olivia Levrini. Epistemic insights as design principles for a teaching-learning module on artificial intelligence. *Science & Education*, pages 1–36, 2024.

[14] Elizabeth F. Barkley, K. Patricia Cross, and Claire H. Major. *Collaborative learning techniques: A handbook for college faculty*. John Wiley & Sons, 2014.

[15] Susan M. Barnett and Stephen J. Ceci. When and where do we apply what we learn?: A taxonomy for far transfer. *Psychological bulletin*, 128(4):612, 2002.

[16] Piraye Bayman and Richard E. Mayer. Using conceptual models to teach basic computer programming. *Journal of Educational Psychology*, 80(3):291, 1988.

[17] Mordechai Ben-Ari. *Principles of concurrent and distributed programming*. PHI Series in computer science. Prentice Hall, US, 1990.

[18] Mordechai Ben-Ari and Yifat Ben-David Kolikant. Thinking parallel: The process of learning concurrency. *SIGCSE Bull.*, 31(3):13–16, jun 1999.

[19] Yifat Ben-David Kolikant. Learning concurrency: evolution of students' understanding of synchronization. *International Journal of Human-Computer Studies*, 60(2):243–268, 2004.

[20] Alan F. Blackwell, Emma Cocker, Geoff Cox, Alex McLean, and Thor Magnusson. *Live Coding: A User's Manual*. The MIT Press, Boston, 11 2022.

[21] Jonas Boustedt, Anna Eckerdal, Robert McCartney, Jan Erik Moström, Mark Ratcliffe, Kate Sanders, and Carol Zander. Threshold concepts in computer science: do they exist and are they useful? *ACM Sigcse Bulletin*, 39(1):504–508, 2007.

[22] Jonas Boustedt, Anna Eckerdal, Robert McCartney, Jan Erik Moström, Mark Ratcliffe, Kate Sanders, and Carol Zander. Threshold concepts in computer science: Do they exist and are they useful? In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '07, page 504–508, New York, NY, USA, 2007. Association for Computing Machinery.

[23] Stefan Brass and Christian Goldberg. Semantic errors in SQL queries: A quite complete list. *Journal of Systems and Software*, 79(5):630–644, 2006.

[24] Karen Brennan and Mitchel Resnick. New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American educational research association, Vancouver, Canada*, volume 1, page 25, 2012.

[25] Álvaro Briz Redón and Ángel Serrano Aroca. Learning mathematics through the r programming language in secondary education. *Educación matemática*, 30(1):133–162, 2018.

[26] Ruven Brooks. Towards a theory of the comprehension of computer programs. *International Journal of Man-Machine Studies*, 18(6):543–554, 1983.

[27] Pam Burnard, Zsolt Lavicza, and Carrie Anne Philbin. Strictly coding: Connecting mathematics and music through digital making. In Eve Torrence, Bruce Torrence, Carlo Séquin, Douglas McKenna, Kristóf Fenyvesi, and Reza Sarhangi, editors, *Proceedings of Bridges 2016: Mathematics, Music, Art, Architecture, Education, Culture*, pages 345–350, Phoenix, Arizona, 2016. Tessellations Publishing. Available online at http://archive.bridgesmathart.org/2016/bridges2016-345.html.

[28] Joshua Burridge and Alan Fekete. Teaching programming for first-year data science. In *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1*, pages 297–303, 2022.

[29] Longbing Cao. *Data Science Thinking: The Next Scientific, Technological and Economic Revolution*. Data Analytics. Springer International Publishing, 2018.

[30] Marilena Carnasciali, Sara Garbarino, Giovanna Guerrini, Daniele Traversaro, Luca Gelati, and Vincenzo Petito. Team metrics: dall'aula al team – una app di supporto alle metodologie didattiche basate sui gruppi. In *Faculty Development e innovazione didattica universitaria*, pages 387–404, Genoa, Italy, 2021. Genova University Press.

[31] Nicholas Carriero and David Gelernter. Linda in context. *Commun. ACM*, 32(4):444–458, apr 1989.

[32] Lillian N. Cassel, Darina Dicheva, Christo Dichev, Don Goelman, and Michael Posner. Data science for all: An introductory course for non-majors; in flipped format (abstract only). In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, page 691, New York, NY, USA, 2016. Association for Computing Machinery.

[33] Barbara Catania, Sara Garbarino, Giovanna Guerrini, and Daniele Traversaro. Un'esperienza di cooperative learning a distanza nell'insegnamento di basi di dati. In Antonella Lotti, Gloria Crea, Sara Garbarino, Federica Picasso, and Erika Scellato, editors, *Faculty Development e innovazione didattica universitaria*, pages 365–376. Genoa University Press, in Italian, 2021.

[34] Barbara Catania, Giovanna Guerrini, and Daniele Traversaro. Collaborative learning in an introductory database course: A study with think-pair-share and team peer review. In *1st International Workshop on Data Systems Education*, pages 60–66, 2022.

[35] Jennifer Chayes. Data science and computing at uc berkeley. *PubPub*, 2021.

[36] Chin Soon Cheah. Factors contributing to the difficulties in teaching and learning of computer programming: A literature review. *Contemporary Educational Technology*, 12(2):ep272, 2020.

[37] Artem Chebotko, Andrey Kashlev, and Shiyong Lu. A big data modeling methodology for Apache Cassandra. In *2015 IEEE International Congress on Big Data*, pages 238–245. IEEE, 2015.

[38] Liu Chen, Ali Davoudian, and Mengchi Liu. A workload-driven method for designing aggregate-oriented NoSQL databases. *Data & Knowledge Engineering*, 142:102089, 2022.

[39] Mei Chen, Seth Poulsen, Ridha Alkhabaz, and Abdussalam Alawini. A quantitative analysis of student solutions to graph database problems. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*, pages 283–289, 2021.

[40] Manuela Chessa, Giorgio Delzanno, Giovanna Guerrini, and Daniele Traversaro. Virtual Learning Environments for Remote Lecturing: Comparing Mozilla Hubs and Gather. In *Convegno DIDAMATICA 2022*, pages 523–532, 11 2022.

[41] Thomas Connolly and Carolyn Begg. A constructivist-based approach to teaching database analysis and design. *Journal of Information Systems Education*, 17, 01 2006.

[42] Thomas Connolly, Mark Stansfield, and Evelyn Mclellan. Using an online games-based learning approach to teach database design concepts. *The Electronic Journal of E-Learning*, 4, 01 2006.

[43] Drew Conway. The data science Venn diagram. *Retrieved from: http://www. dataists. com/2010/09/the-data-science-venn-diagram*, 2010.

[44] Isabella Corradini, Michael Lodi, and Enrico Nardelli. Conceptions and misconceptions about computational thinking among italian primary school teachers. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*, ICER '17, page 136–144, New York, NY, USA, 2017. Association for Computing Machinery.

[45] Kathryn Cunningham, Sarah Blanchard, Barbara Ericson, and Mark Guzdial. Using tracing and sketching to solve programming problems: Replicating and extending an analysis of what students draw. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*, ICER '17, page 164–172, New York, NY, USA, 2017. Association for Computing Machinery.

[46] Andrea Danyluk, Paul Leidig, Andrew McGettrick, Lillian Cassel, Maureen Doyle, Christian Servin, Karl Schmitt, and Andreas Stefik. Computing competencies for undergraduate data science programs: An ACM task force final report. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, pages 1119–1120, 2021.

[47] Thomas H. Davenport and D.J. Patil. Data scientist. *Harvard business review*, 90(5):70–76, 2012.

[48] Alfonso de la Vega, Diego García-Saiz, Carlos Blanco, Marta Zorrilla, and Pablo Sánchez. Mortadelo: Automatic generation of NoSQL stores from platform-independent data models. *Future Generation Computer Systems*, 105:455–474, 2020.

[49] Claudio de Lima and Ronaldo dos Santos Mello. A workload-driven logical design approach for NoSQL document databases. In *Proceedings of the 17th international conference on information integration and web-based applications & services*, pages 1–10, 2015.

[50] Giorgio Delzanno, Luca Gelati, Giovanna Guerrini, Angela Sugliano, and Daniele Traversaro. Experience-based training in computer science education via online multiplayer games on computational thinking. In *International Workshop on Higher Education Learning Methodologies and Technologies Online*, pages 459–470. Springer, 2022.

[51] Giorgio Delzanno, Giovanna Guerrini, and Daniele Traversaro. Sonic TBL: Un percorso sonico da creatività a didattica dell'informatica. In *Proceedings of ITADINFO 2023, Convegno italiano sulla didattica dell'informatica*, pages 109–116, 2023.

[52] Giorgio Delzanno, Giovanna Guerrini, and Daniele Traversaro. Exploring student misconceptions about concurrency using sonic pi. In *Proceedings of the Technical Symposium on Computer Science Education (SIGCSE TS 2024)*, 2024.

[53] Giorgio Delzanno, Giovanna Guerrini, Daniele Traversaro, and Davide Ponzini. Python per tutti i gusti: tre diversi approcci per introdurre neofiti alla programmazione. In *Proceedings of ITADINFO 2023, Convegno italiano sulla didattica dell'informatica*, pages 101–108, 2023.

[54] W. Edwards Deming. Out of the crisis. *Center for Advanced Engineering Study, Massachusetts Institute of Technology*, 6, 1986.

[55] Paul E. Dickson, Tim D. Richards, and Brett A. Becker. Experiences implementing and utilizing a notional machine in the classroom. In Larry Merkle, Maureen Doyle, Judithe Sheard, Leen-Kiat Soh, and Brian Dorn, editors, *SIGCSE 2022: The 53rd ACM Technical Symposium on Computer Science Education, Providence, RI, USA, March 3-5, 2022, Volume 1*, pages 850–856. ACM, 2022.

[56] César Domínguez and Arturo Jaime. Database design learning: A project-based approach organized through a course management system. *Computers & Education*, 55(3):1312–1320, 2010.

[57] Benedict du Boulay. Some difficulties of learning to program. *Journal of Educational Computing Research*, 2(1):57–73, 1986.

[58] Daniel Ernst and Daniel Stevenson. Concurrent CS: Preparing students for a multicore world. *ACM SIGCSE Bulletin*, 40:230–234, 06 2008.

[59] Sally Fincher, Johan Jeuring, Craig S. Miller, Peter Donaldson, Benedict du Boulay, Matthias Hauswirth, Arto Hellas, Felienne Hermans, Colleen Lewis, Andreas Mühling, Janice L. Pearce, and Andrew Petersen. Notional machines in computing education: The education of attention. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education*, ITiCSE-WGR '20, page 21–50, New York, NY, USA, 2020. Association for Computing Machinery.

[60] Kathi Fisler. Data-centricity: Rethinking introductory computing to support data science. In *1st International Workshop on Data Systems Education*, pages 1–3, 2022.

[61] Kathi Fisler, Shriram Krishnamurthi, Benjamin S Lerner, and Joe Gibbs Politz. A data-centric introduction to computing. 2021 Accessed September, https://dcic-world.org.

[62] Brad Fowler, Joy Godin, and Margaret Geddy. Teaching case: introduction to NoSQL in a traditional database course. *Journal of Information Systems Education*, 27(2):99, 2016.

[63] Daniel Frischemeier, Rolf Biehler, Susanne Podworny, and Lea Budde. A first introduction to data science education in secondary schools: Teaching and learning about data exploration with codap using survey data. *Teaching Statistics*, 43(S1):S182–S189, 2021.

[64] Einat Gil and Alison L Gibbs. Introducing secondary school students to big data and its social impact: A study within an innovative learning environment. In *Promoting understanding of statistics about society. Proceedings of the Roundtable Conference of the International Association for Statistics Education (IASE)*, 7 2016.

[65] Andreas Grillenberger and Ralf Romeike. Developing a theoretically founded data literacy competency model. In *Proceedings of the 13th Workshop in Primary and Secondary Computing Education*, WiPSCE '18, New York, NY, USA, 2018. Association for Computing Machinery.

[66] Giovanna Guerrini and Daniele Traversaro. Introduction to data literacy with Tableau in high school. In *Didamatica 2022*, pages 239–245, 2022.

[67] Giovanna Guerrini and Daniele Traversaro. Two introductory data-driven activities for secondary schools. In *31st Symposium of Advanced Database Systems (SEBD 2023)*, pages 641–649, 2023.

[68] Mark Guzdial. Learner-centered design of computing education: Research on computing for everyone. *Synthesis Lectures on Human-Centered Informatics*, 8(6):1–165, 2015.

[69] John Hamer. An approach to teaching design patterns using musical composition. In *Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, ITiCSE '04, page 156–160, New York, NY, USA, 2004. Association for Computing Machinery.

[70] Matthew Harazim. *//This is a Comment: Music, Computers and Culture in Live Coding*. PhD thesis, The University of Edinburgh, 04 2017.

[71] Orit Hazzan and Koby Mike. A journal for interdisciplinary data science education. *Commun. ACM*, 64(8):10–11, jul 2021.

[72] Orit Hazzan and Koby Mike. The birth of a new discipline: Data science education. In *Guide to Teaching Data Science: An Interdisciplinary Approach*, pages 59–72. Springer, 2023.

[73] Orit Hazzan and Koby Mike. *Guide to Teaching Data Science: An Interdisciplinary Approach*. Springer Nature, 2023.

[74] Jesse M. Heines, Gena R. Greher, and Sarah Kuhn. Music performamatics: interdisciplinary interaction. In Sue Fitzgerald, Mark Guzdial, Gary Lewandowski, and Steven A. Wolfman, editors, *Proceedings of the 40th SIGCSE Technical Symposium on Computer Science Education, SIGCSE 2009, March 4-7, 2009*, pages 478–482, Chattanooga, TN, USA, 2009. ACM.

[75] Maurice Herlihy and Nir Shavit. *The Art of Multiprocessor Programming, Revised Reprint*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2012.

[76] Geoffrey L. Herman. Designing contributing student pedagogies to promote students' intrinsic motivation to learn. *Computer Science Education*, 22(4):369–388, 2012.

[77] Stephanie C Hicks and Rafael A Irizarry. A guide to teaching data science. *The American Statistician*, 72(4):382–391, 2018.

[78] Rune Hjelsvold and Nipuna Hiranya Weeratunge. Peer code review as formative assessment: A case study from a database course project. In *Norsk IKT-konferanse for forskning og utdanning*, 2021.

[79] Charles A. R. Hoare. Communicating Sequential Processes. *Commun. ACM*, 21(8):666–677, 1978.

[80] Christopher Hundhausen, Anukrati Agrawal, Dana Fairbrother, and Michael Trevisan. Integrating pedagogical code reviews into a CS 1 course: an empirical study. *ACM SIGCSE Bulletin*, 41(1):291–295, 2009.

[81] Andrew Hunter and Christopher Exton. Elucidate: employing information visualisation to aid pedagogy for students. In Maria De Marsico, Stefano Levialdi, and Emanuele Panizzi, editors, *Proceedings of the Working Conference on Advanced Visual Interfaces, AVI 2002, Trento, Italy, May 22-24, 2002*, pages 343–344, Italy, 2002. ACM.

[82] Rafael A. Irizarry. The role of academia in data science education. *Harvard Data Science Review*, 2(1):10–1162, 2020.

[83] ISfTiE ISTE and CSTA CSTA. Operational definition of computational thinking for k-12 education. *National Science Foundation*, 2011.

[84] Cruz Izu, Carsten Schulte, Ashish Aggarwal, Quintin Cutts, Rodrigo Duran, Mirela Gutica, Birte Heinemann, Eileen Kraemer, Violetta Lonati, Claudio Mirolo, et al. Fostering program comprehension in novice programmers-learning activities and learning trajectories. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education, ITiCSE-WGR '19*, pages 27–52. ACM, Aberdeen, Scotland, UK, 2019.

[85] Abhijit Jain. Data visualization with the D3. JS Javascript library. *Journal of Computing Sciences in Colleges*, 30(2):139–141, 2014.

[86] Maria Kallia and Sue Sentance. Computing teachers' perspectives on threshold concepts: Functions and procedural abstraction. In *Proceedings of the 12th Workshop on Primary and Secondary Computing Education*, WiPSCE '17, page 15–24, New York, NY, USA, 2017. Association for Computing Machinery.

[87] Michael S. Kirkpatrick. Student perspectives of team-based learning in a CS Course: Summary of qualitative findings. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '17, page 327–332, New York, NY, USA, 2017. Association for Computing Machinery.

[88] Melissa K. Kjelvik and Elizabeth H. Schultheis. Getting messy with authentic data: Exploring the potential of using data from scientific research to support student data literacy. *CBE—Life Sciences Education*, 18(2):es2, 2019. PMID: 31074698.

[89] Yifat Ben-David Kolikant. Students' alternative standards for correctness. In *Proceedings of the First International Workshop on Computing Education Research*, ICER '05, page 37–43, New York, NY, USA, 2005. Association for Computing Machinery.

[90] Yifat Ben-David Kolikant. Students' alternative standards for correctness. In Richard J. Anderson, Sally Fincher, and Mark Guzdial, editors, *International Computing Education Research Workshop 2005, ICER '05, Seattle, WA, USA, October 1-2, 2005*, pages 37–43, Seattle, 2005. ACM.

[91] Aditi Kothiyal, Sahana Murthy, and Sridhar Iyer. Think-pair-share in a large CS1 class: does learning really happen? In Åsa Cajander, Mats Daniels, Tony Clear, and Arnold Pears, editors, *Innovation and Technology in Computer Science Education Conference 2014, ITiCSE '14, Uppsala, Sweden, June 23-25, 2014*, pages 51–56. ACM, 2014.

[92] Dimitrios Kotsifakos, Dimitrios Magetos, Alexandros Veletsos, and Christos Douligeris. Teaching the basic commands of NoSQL databases using neo4j in vocational education and training (vet). *European Journal of Engineering and Technology Research*, pages 13–18, 2019.

[93] Shriram Krishnamurthi and Kathi Fisler. 13 programming paradigms and beyond. *The Cambridge handbook of computing education research*, page 377, 2019.

[94] Shriram Krishnamurthi and Kathi Fisler. *Programming Paradigms and Beyond*, page 377–413. Cambridge Handbooks in Psychology. Cambridge University Press, Cambridge, 2019.

[95] Shriram Krishnamurthi and Kathi Fisler. Data-centricity: a challenge and opportunity for computing education. *Communications of the ACM*, 63(8):24–26, 2020.

[96] Amruth N. Kumar and Rajendra K. Raj. Computer science curricula 2023 (cs2023): The final report. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2*, SIGCSE 2024, page 1867–1868, New York, NY, USA, 2024. Association for Computing Machinery.

[97] Kyungbin Kwon. Novice programmer's misconception of programming reflected on problem-solving plans. *International Journal of Computer Science Education in Schools*, 1(4):n4, 2017.

[98] Leslie Lamport. Teaching concurrency. *SIGACT News*, 40(1):58–62, 2009.

[99] Zampeta Legaki, Daniel Fernandez Galeote, and Juho Hamari. The impact of different gamification types in the context of data literacy: An online experiment. In Mila Bujić, Jonna Koivisto, and Juho Hamari, editors, *Proceedings of the 6th International GamiFIN Conference 2022 (GamiFIN 2022)*, CEUR Workship Proceedings, pages 22–32, 2022. International GamiFIN Conference ; Conference date: 26-04-2022 Through 29-04-2022.

[100] Zampeta Legaki and Juho Hamari. Gamification in statistics education: A literature review. In Jonna Koivisto, Mila Bujić, and Juho Hamari, editors, *GamiFIN Conference 2020*, CEUR workshop proceedings, pages 41–51. CEUR-WS, 2020. JUFOID=53269; International GamiFIN Conference ; Conference date: 01-01-1900.

[101] Aristotelis Leventidis, Jiahui Zhang, Cody Dunne, Wolfgang Gatterbauer, HV Jagadish, and Mirek Riedewald. QueryVis: Logic-based diagrams help users understand complicated SQL queries faster. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 2303–2318, 2020.

[102] Colleen M. Lewis, Ruth E. Anderson, and Ken Yasuhara. I don't code all day fitting in computer science when the stereotypes don't fit. In *Proceedings of the 2016 ACM conference on international computing education research*, pages 23–32, 2016.

[103] Xumin Liu, Erik Golen, Rajendra K. Raj, and Kimberly Fluet. Offering data science coursework to non-computing majors. In *Proceedings of the 2nd International Workshop on Data Systems Education: Bridging education practice with education research*, pages 44–49, 2023.

[104] Michael Lodi. Informatical thinking. *Olympiads in Informatics: An International Journal*, 14:113–132, 2020.

[105] Michael Lodi. *Introducing Computational Thinking in K-12 Education: Historical, Epistemological, Pedagogical, Cognitive, and Affective Aspects*. PhD thesis, University of Bologna, Aprile 2020.

[106] Michael Lodi and Simone Martini. Computational thinking, between papert and wing. *Science & education*, 30(4):883–908, 2021.

[107] Jan Lönnberg. Student errors in concurrent programming assignments. In *Proceedings of the 6th Baltic Sea Conference on Computing Education Research: Koli Calling 2006*, Baltic Sea '06, page 145–146, New York, NY, USA, 2006. Association for Computing Machinery.

[108] Jan Lönnberg, Mordechai Ben-Ari, and Lauri Malmi. Java replay for dependence-based debugging. In João Lourenço and Eitan Farchi, editors, *Proceedings of the 9th Workshop on Parallel and Distributed Systems: Testing, Analysis, and Debugging, PADTAD 2011*, pages 15–25, Toronto, ON, Canada, July 17-21, 2011, 2011. ACM.

[109] Douglas Lusa Krug, Yifan Zhang, Chrystalla Mouza, Taylor Barnett, Lori Pollock, and David C. Shepherd. Using domain-specific, immediate feedback to support students learning computer programming to make music. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*, pages 368–374, 2023.

[110] Mahender Mandala, Christian Schunn, Steven Dow, Mary Goldberg, Jon Pearlman, William Clark, and Irene Mena. Impact of collaborative team peer review on the quality of feedback in engineering design projects. *International Journal of Engineering Education*, 34(4):1299–1313, 2018.

[111] Carme Martin, Toni Urpí, M. José Casany, Xavier Burgués Illa, Carme Quer, M Elena Rodríguez, and Alberto Abello. Improving learning in a database course using collaborative learning techniques. *The International Journal of Engineering Education*, 29(4):986–997, 2013.

[112] Raina Mason and Graham Cooper. Why the bottom 10% just can't do it: Mental effort measures and implication for introductory programming courses. In *Proceedings of the Fourteenth Australasian Computing Education Conference-Volume 123*, pages 187–196, 2012.

[113] Jan H.F. Meyer and Ray Land. Threshold concepts and troublesome knowledge: Linkages to ways of thinking and practising within the disciplines. In *ISL10 Improving Student Learning*, pages 412–424, United Kingdom, January 2003. Oxford Brookes University.

[114] Daphne Miedema, Efthimia Aivaloglou, and George Fletcher. Identifying SQL misconceptions of novices: Findings from a think-aloud study. In *ACM Conference on International Computing Education Research 2021 (ICER '21)*, pages 355–367. ACM, 2021.

[115] Daphne Miedema, Efthimia Aivaloglou, and George Fletcher. Identifying SQL misconceptions of novices: Findings from a think-aloud study. *ACM Inroads*, 13(1):52–65, 2022.

[116] Daphne Miedema and George Fletcher. SQLVis: Visual query representations for supporting SQL learners. In *2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 1–9. IEEE, 2021.

[117] Koby Mike and Orit Hazzan. How can computer science educators benefit from data science education? In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, SIGCSE '21, page 1363, New York, NY, USA, 2021. Association for Computing Machinery.

[118] Koby Mike and Orit Hazzan. Interdisciplinary CS1 Course for Non-Majors: The Case of Graduate Psychology Students. In *2022 IEEE global engineering education conference (EDUCON)*, pages 86–93. IEEE, 2022.

[119] Koby Mike and Orit Hazzan. Machine learning for non-majors: A white box approach. *Statistics Education Research Journal*, 21(2):10–10, 2022.

[120] Koby Mike, Noa Ragonis, Rinat B. Rosenberg-Kima, and Orit Hazzan. Computational thinking in the era of data science. *Commun. ACM*, 65(8):33–35, jul 2022.

[121] Koby Mike, Noa Ragonis, Rinat B. Rosenberg-Kima, and Orit Hazzan. Computational thinking in the era of data science. *Communications of the ACM*, 65(8):33–35, 2022.

[122] Robin Milner. *Communication and concurrency*. PHI Series in computer science. Prentice Hall, 1989.

[123] Alanah Mitchell and Amy Grace Vaughan. Implementing team-based learning: Findings from a database class. *Journal of Information Technology Education. Innovations in Practice*, 21:1, 2022.

[124] Sriram Mohan. Teaching NoSQL Databases to undergraduate students: A novel approach. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, SIGCSE '18, page 314–319, New York, NY, USA, 2018. Association for Computing Machinery.

[125] Peter G. Moore. The skills challenge of the nineties. *Journal of the Royal Statistical Society Series A: Statistics in Society*, 153(3):265–285, 1990.

[126] Enrico Nardelli. Do we really need computational thinking? *Communications of the ACM*, 62(2):32–35, 2019.

[127] Peter Naur. The science of datalogy. *Communications of the ACM*, 9(7):485, 1966.

[128] David Nicol, Avril Thomson, and Caroline Breslin. Rethinking feedback practices in higher education: a peer review perspective. *Assessment & Evaluation in Higher Education*, 39(1):102–122, 2014.

[129] Donald A. Norman. Some observations on mental models. In *Mental models*, pages 15–22. Psychology Press, 2014.

[130] Seymour Papert. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, Inc., USA, 1980.

[131] Seymour Papert. Personal computing and its impact on education. *The computer in the school: Tutor, tool, tutee*, pages 197–202, 1980.

[132] Elizabeth Patitsas, Jesse Berlin, Michelle Craig, and Steve Easterbrook. Evidence that computer science grades are not bimodal. *Communications of the ACM*, 63(1):91–98, 2019.

[133] Liliia V. Pavlenko, Maksym P. Pavlenko, Vitalii H. Khomenko, and Vitaliy I. Mezhuyev. Application of R Programming Language in Learning Statistics. In *Proceedings of the 1st Symposium on Advances in Educational Technology*, volume 2, pages 62–72, 2022.

[134] Roy D. Pea. Language-independent conceptual "bugs" in novice programming. *Journal of educational computing research*, 2(1):25–36, 1986.

[135] Carl Adam Petri and Wolfgang Reisig. Petri net. *Scholarpedia*, 3(4):6477, 2008.

[136] Christopher Petrie. Programming music with Sonic Pi promotes positive attitudes for beginners. *Computers & Education*, 179:104409, 2022.

[137] Gustavo Pinto, Weslley Torres, and Fernando Castor. A study on the most popular questions about concurrent programming. In *Proceedings of the 6th Workshop on Evaluation and Usability of Programming Languages and Tools*, pages 39–46, 2015.

[138] Leo Porter, Mark Guzdial, Charlie McDowell, and Beth Simon. Success in introductory programming: What works? *Communications of the ACM*, 56(8):34–36, 2013.

[139] Kai Presler-Marshall, Sarah Heckman, and Kathryn Stolee. SQLRepair: Identifying and repairing mistakes in student-authored SQL queries. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, pages 199–210. IEEE, 2021.

[140] Mareen Przybylla and Ralf Romeike. Key competences with physical computing. *KEYCIT 2014-Key Competencies in Informatics and ICT*, pages 351–361, 2015.

[141] Ela Pustulka, Kai Krause, Lucia de Espona, and Andrea Kennel. SQL Scrolls - A Reusable and Extensible DGBL Experiment. In *10th Computer Science Education Research Conference*, CSERC '21, page 39–48, New York, NY, USA, 2021. Association for Computing Machinery.

[142] Yizhou Qian and James Lehman. Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education (TOCE)*, 18(1):1–24, 2017.

[143] Juliana Elisa Raffaghelli. *Educators' Data Literacy: Supporting critical perspectives in the context of a "datafied" education*, pages 91–109. Aracné, 12 2018.

[144] Mitchel Resnick. Multilogo: A study of children and concurrent programming. *Interactive Learning Environments*, 1(3):153–170, 1990.

[145] Mitchel Resnick. All I really need to know (about creative thinking) I learned (by studying how children learn) in kindergarten. In *Proceedings of the 6th ACM SIGCHI conference on Creativity & cognition*, pages 1–6, 2007.

[146] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. Scratch: programming for all. *Communications of the ACM*, 52(11):60–67, 2009.

[147] Chantel Ridsdale, James Rothwell, Mike Smit, Michael Bliemel, Dean Irvine, Dan Kelley, Stan Matwin, Brad Wuetherick, and Hossam Ali-Hassan. Strategies and best practices for data literacy education knowledge synthesis report, 01 2015.

[148] Anthony Robins, Janet Rountree, and Nathan Rountree. Learning and teaching programming: A review and discussion. *Computer science education*, 13(2):137–172, 2003.

[149] Anthony V Robins. 12 novice programmers and introductory programming. *The Cambridge handbook of computing education research*, page 327, 2019.

[150] S. Alex Ruthmann, Jesse M. Heines, Gena R. Greher, Paul Laidler, and Charles Saulters II. Teaching computational thinking through musical live coding in scratch. In Gary Lewandowski, Steven A. Wolfman, Thomas J. Cortina, and Ellen Lowenfeld Walker, editors, *Proceedings of the 41st ACM technical symposium on Computer science education, SIGCSE 2010, March 10-13, 2010*, pages 351–355, Milwaukee, Wisconsin, USA, 2010. ACM.

[151] Jorma Sajaniemi. An empirical analysis of roles of variables in novice-level procedural programs. In *Proceedings IEEE 2002 Symposia on Human Centric Computing Languages and Environments*, pages 37–39. IEEE, 2002.

[152] Jeffrey Saltz and Robert Heckman. Big data science education: A case study of a project-focused introductory course. *Themes in science and technology education*, 8(2):85–94, 2016.

[153] Abdul Sattar, Torben Lorenzen, and Keerthi Nallamaddi. Incorporating NoSQL into a database course. *acm Inroads*, 4(2):50–53, 2013.

[154] Emanuele Scapin, Nicola Dalla Pozza, and Claudio Mirolo. An exploratory investigation on high-school students' understanding of threads. In *International Conference on Informatics in Schools: Situation, Evolution, and Perspectives*, pages 93–110. Springer, 2023.

[155] Emmanuel Schanzer, Nancy Pfenning, Flannery Denny, Sam Dooman, Joe Gibbs Politz, Benjamin S. Lerner, Kathi Fisler, and Shriram Krishnamurthi. Integrated data science for secondary schools: Design and assessment of a curriculum. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1*, SIGCSE 2022, page 22–28, New York, NY, USA, 2022. Association for Computing Machinery.

[156] Zahava Scherz, David Goldberg, and Z. Fund. Cognitive implications of learning prolog—mistakes and misconceptions. *Journal of Educational Computing Research*, 6(1):89–110, 1990.

[157] Johannes Schildgen. SQL island: An adventure game to learn the database language SQL. In *8th European Conference on Games Based Learning (ECGBL 2014)*, pages 137–138, 2014.

[158] Johannes Schildgen. Monster park-the entity-relationship-diagram learning game. In *ER Forum/Posters/Demos*, pages 150–157, 2020.

[159] J. Philipp Schmidt, Mitchel Resnick, and Joi Ito. Creative learning and the future of work. *Disrupting unemployment*, pages 147–155, 2016.

[160] Jean Scholtz and Susan Wiedenbeck. Learning second and subsequent programming languages: A problem of transfer. *International Journal of Human-Computer Interaction*, 2(1):51–72, 1990.

[161] Sue Sentance, Jane Waite, Lucy Yeomans, and Emily MacLeod. Teaching with physical computing devices: the bbc micro: bit initiative. In *Proceedings of the 12th Workshop on Primary and Secondary Computing Education*, pages 87–96, 2017.

[162] Bilal Shebaro. Using active learning strategies in teaching introductory database courses. *J. Comput. Sci. Coll.*, 33(4):28–36, apr 2018.

[163] Nischal Shrestha, Colton Botta, Titus Barik, and Chris Parnin. Here we go again: Why is it difficult for developers to learn another programming language? *Commun. ACM*, 65(3):91–99, feb 2022.

[164] Valerie J. Shute, Chen Sun, and Jodi Asbell-Clarke. Demystifying computational thinking. *Educational research review*, 22:142–158, 2017.

[165] Beth Simon, Päivi Kinnunen, Leo Porter, and Dov Zazkis. Experience report: CS1 for majors with media computation. In Reyyan Ayfer, John Impagliazzo, and Cary Laxer, editors, *Proceedings of the 15th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ITiCSE 2010, June 26-30, 2010*, pages 214–218, Bilkent, Ankara, Turkey, 2010. ACM.

[166] Sharad Sinha and Clint P. George. Artificial intelligence for all using R programming language. *AI Matters*, 5(4):10–13, 2020.

[167] Cleyton Slaviero and Edward Herman Haeusler. Computational thinking tools: Analyzing concurrency and its representations. *Journal on Interactive Systems*, 9(1), 2018.

[168] John P. Smith III, Andrea A. diSessa, and Jeremy Roschelle. Misconceptions reconceived: A constructivist analysis of knowledge in transition. *The journal of the learning sciences*, 3(2):115–163, 1994.

[169] Elliot Soloway. Learning to program= learning to construct mechanisms and explanations. *Communications of the ACM*, 29(9):850–858, 1986.

[170] Juha Sorva. The same but different students' understandings of primitive and object variables. In *Proceedings of the 8th International Conference on Computing Education Research*, pages 5–15, 2008.

[171] Juha Sorva. Reflections on threshold concepts in computer programming and beyond. In *Proceedings of the 10th Koli calling international conference on computing education research*, pages 21–30, 2010.

[172] Juha Sorva. Notional machines and introductory programming education. *ACM Transactions on Computing Education*, 13:8:1–8:31, 06 2013.

[173] Juha Sorva. Notional machines and introductory programming education. *ACM Trans. Comput. Educ.*, 13(2), jul 2013.

[174] Naomi J. Spence, Rachel Anderson, Sherryse Corrow, Susan A Dumais, and Lisa Dierker. Passion-driven statistics: A course-based undergraduate research experience (cure). *The Mathematics Enthusiast*, 19(3):759–770, 2022.

[175] Filip Strömbäck. *Teaching and Learning Concurrent Programming in the Shared Memory Model*. PhD thesis, Linköping University, Sweden, 2023.

[176] Filip Strömbäck, Linda Mannila, Mikael Asplund, and Mariam Kamkar. A student's view of concurrency - a study of common mistakes in introductory courses on concurrency. In *Proceedings of the 2019 ACM Conference on International Computing Education Research*, ICER '19, page 229–237, New York, NY, USA, 2019. Association for Computing Machinery.

[177] Filip Strömbäck, Linda Mannila, and Mariam Kamkar. Pilot study of progvis: A visualization tool for object graphs and concurrency via shared memory. In *Proceedings of the 24th Australasian Computing Education Conference*, ACE '22, page 123–132, New York, NY, USA, 2022. Association for Computing Machinery.

[178] David G. Sullivan. A data-centric introduction to computer science for non-majors. In *Proceeding of the 44th ACM technical symposium on Computer science education*, pages 71–76, 2013.

[179] Herb Sutter. The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's Journal*, 30(3):202–210, 2005.

[180] Toni Taipalus and Piia Perälä. What to expect and what to focus on in SQL query teaching. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 198–203, 2019.

[181] Andrea Tartaro and Renee J Chosed. Computer scientists at the biology lab bench. In *Proceedings of the 46th ACM technical symposium on computer science education*, pages 120–125, 2015.

[182] David Taylor. Battle of the data science venn diagrams. *KDNuggets News*, 2016.

[183] Matti Tedre, Tapani Toivonen, Juho Kahila, Henriikka Vartiainen, Teemu Valtonen, Ilkka Jormanainen, and Arnold Pears. Teaching machine learning in k–12 classroom: Pedagogical and technological trajectories for artificial intelligence education. *IEEE access*, 9:110558–110572, 2021.

[184] Daniele Traversaro. Online think–pair–share in a third-age ict course. *International Journal of Educational and Pedagogical Sciences*, 16(7):371 – 376, 2022.

[185] Daniele Traversaro, Giorgio Delzanno, and Giovanna Guerrini. "hear" and "play" students misconceptions on concurrent programming using sonic pi. *Informatics in Education*, 2024.

[186] Daniele Traversaro, Giovanna Guerrini, and Giorgio Delzanno. Sonic Pi for TBL teaching units in an introductory programming course. In *Adjunct Publication of the 28th ACM Conference on User Modeling, Adaptation and Personalization*, pages 143–150, 2020.

[187] Miguel Ehecatl Trujillo and Gabriel Garcéa-Mireles. Gamification and SQL: An Empirical Study on Student Performance in a Database Course. *ACM Transactions on Computing Education*, 21:1–29, 12 2020.

[188] De Montfort University. Educ2323 computer programming as a tool for learning: Week 5 syntonicity, 2021. Accessed on July 9, 2023, https://www.tech.dmu.ac.uk/ mjdean/notes/-modules/education/EDUC2323/2021/content/.

[189] Vesa Vainio and Jorma Sajaniemi. Factors in novice programmers' poor tracing skills. *ACM SIGCSE Bulletin*, 39(3):236–240, 2007.

[190] Leo Van Audenhove, Wendy Van den Broeck, and Ilse Mariën. Data literacy and education. introduction and the challenges for our field. *Journal of Media Literacy Education*, 12:1–5, 12 2020.

[191] Renske Weeda, Sjaak Smetsers, and Erik Barendsen. Unraveling novices' code composition difficulties. *Computer Science Education*, 0(0):1–28, 2023.

[192] David Weintrop and Uri Wilensky. Transitioning from introductory block-based and text-based environments to professional programming languages in high school computer science classrooms. *Computers & Education*, 142:103646, 2019.

[193] Susan Wiedenbeck. Beacons in computer program comprehension. *International Journal of Man-Machine Studies*, 25(6):697–709, 1986.

[194] Jeannette M. Wing. Computational thinking. *Communications of the ACM*, 49(3):33–35, 2006.

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

[196] Jeannette M. Wing. Computational thinking: What and why. In *Presentation slides from Trippel Helix Conference on Computational Thinking and Digital Competencies in Primary and Secondary Education Stockholm, Sweden. https://pdfs. semanticscholar. org/presentation/d20a/a49744877f2bb98d6ad303742be7bd025fcd. pdf*, pages 1378800312–1580695435, 2017.

[197] Jeannette M. Wing. Ten research challenge areas in data science. *arXiv preprint arXiv:2002.05658*, 2020.

[198] Leon E. Winslow. Programming pedagogy—a psychological overview. *SIGCSE Bull.*, 28(3):17–22, sep 1996.

[199] Wei Xiong, Douglas Hawley, and David Monismith. NoSQL in database education: incorporating non-relational concepts into a relational database course: panel discussion. *J. Comput. Sci. Coll.*, 30(5):151–152, may 2015.

[200] Donghui Yan and Gary E Davis. A first course in data science. *Journal of Statistics Education*, 27(2):99–109, 2019.

Appendix A

Concurrency Education with Sonic Pi

A.1 RAT Results

For the sake of completeness, in this section we discuss the results of the RAT quizzes.

- For CA students, as expected, we found that team scores tended to be higher than individual scores. Specifically, the RAT quiz success rate (i.e., the percentage of correct answers) increases from 5.6 for the individual test to 6.5 for the team test. Table A.1 shows the percentage of total correct answers for each item Q_i. We found that teams outperformed individuals in all but the first question, which was about live coding loops. Specifically, Q_1, 19 teams answered correctly, and 6 students voted for "No effect on execution. This is live coding", 7 students voted for "It forks the main thread and creates two new ones. One will play 60 and the other 65. This is live coding", and 1 student voted for "It raises a runtime error". Looking at the individual responses, we can see that the second question was the most difficult, which was about live coding with live loops and a sample outside the loop.
- For the CP results, table A.2 shows the percentage of total correct answers for each item Q_i for both individuals and teams. It can be seen that the teams achieved significantly higher percentages of correct answers than the CA students, performing better in all five questions. For the CP students, the fourth question was the most challenging, which was about live coding with live loops and the get/set synchronization mechanism.

CA	Q_1	Q_2	Q_3	Q_4	Q_5
Individual	75%	37%	49%	44%	60%
Team	59%	63%	59%	63%	81%

Table A.1: CA individual and Team RAT results.

CP	Q_1	Q_2	Q_3	Q_4	Q_5
Individual	61%	46%	48%	37%	70%
Team	91%	91%	82%	73%	91%

Table A.2: CP Individual and Team RAT results.

A.2 Individual Post-Questionnaire

- I think that the activity was effective in introducing and/or deepening some concepts of concurrent programming. *Likert scale ranging from 1 to 5.*
- I believe that the musical approach with Sonic Pi helps to understand concurrency in a simple, intuitive, and less abstract way. *Likert scale ranging from 1 to 5.*
- I think that this activity has mainly helped me to... *Open answer.*
- I consider the level of difficulty of the suggested exercises to be *Easy, Ok, Difficult.*
- I consider the time needed to complete the activity to be *Very suitable, Suitable, Not suitable.*
- I prepared adequately for the class activity. *Likert scale ranging from 1 to 5.*
- Have you developed concurrent code in school/university or professional projects? *Yes (with open answer); No*
- Have you ever used the Ruby programming language or the Sonic Pi development environment? *Yes; No*
- Teamwork was useful and satisfying. *Likert scale ranging from 1 to 5.*

A.3 Knowledge Transfer Test

Figures A.1, A.2, and A.3 show the three exercises of the knowledge transfer test.

A.4 Gallery Walk - Team APP Results

Figure A.4 illustrates the results of the gallery walk for tasks 6 and 7.

Identify the output of the program (code comprehension task).

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

int x = 0;
int y = 0;
int z = 0;
int r1 = 0;

void *
thread1 (void *arg)
{
  x = 1;
  y = 1;
}

void *
thread2 (void *arg)
{
  z = 1;
  r1 = x - 1;
}

void *
thread3 (void *arg)
{
  while (y != 1) {}
  while (z != 1) {}
  r1 = x + 1;
}

int
main ()
{
  pthread_t t1, t2, t3;
  // ... and code to create and join threads
```

Figure A.1: Exercise 1

Assuming that '++' is atomic and that the scheduler allocates a portion of CPU time to each unblocked thread, consider the following concurrent programme: *

INITIALISATION (sequential, before the threads start)
array A[2]={0,0}; // indices of A start from 0 so A[0]=A[1]=0
i=0;
Which answer is correct?

THREAD P:	THREAD Q:	THREAD R:

while (i<2) {
A[i]=1;
i++;
}
	while (i<2) {
	A[i]=2;
	i++;
	}
		A[i]=3;

- ◯ An execution exists that at the end of the program satisfies A={3,3} (i.e. A[0]=3 and A[1]=3)
- ◯ There is an execution that generates an error
- ◯ An execution exists that at the end of the program satisfies A={3,2} (i.e. A[0]=3 and A[1]=2)
- ◯ The programme always generates an error

Figure A.2: Exercise 2

Consider the following C program with pthread. Does it have deterministic behaviour?

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

void *goo (void *vargp) {
    int myid;
    myid = *((int*) vargp);
    printf("%d\n", myid);
}

int main() {
    pthread_t tid[2];
    int i, *ptr;
    ptr = (int *) malloc(sizeof(int));
    for(i=0; i<2; i++) {
        *ptr = i;
        pthread_create(&tid[i], 0, goo, ptr);
    }
    *ptr = 10;
    pthread_join(tid[0],0);
    pthread_join(tid[1],0);
}
```

- ◯ Yes
- ◯ No

Which of the following outputs can the program produce? *

- ☐ 10 10
- ☐ 0 0
- ☐ 1 10
- ☐ 0 1 10

Figure A.3: Exercise 3

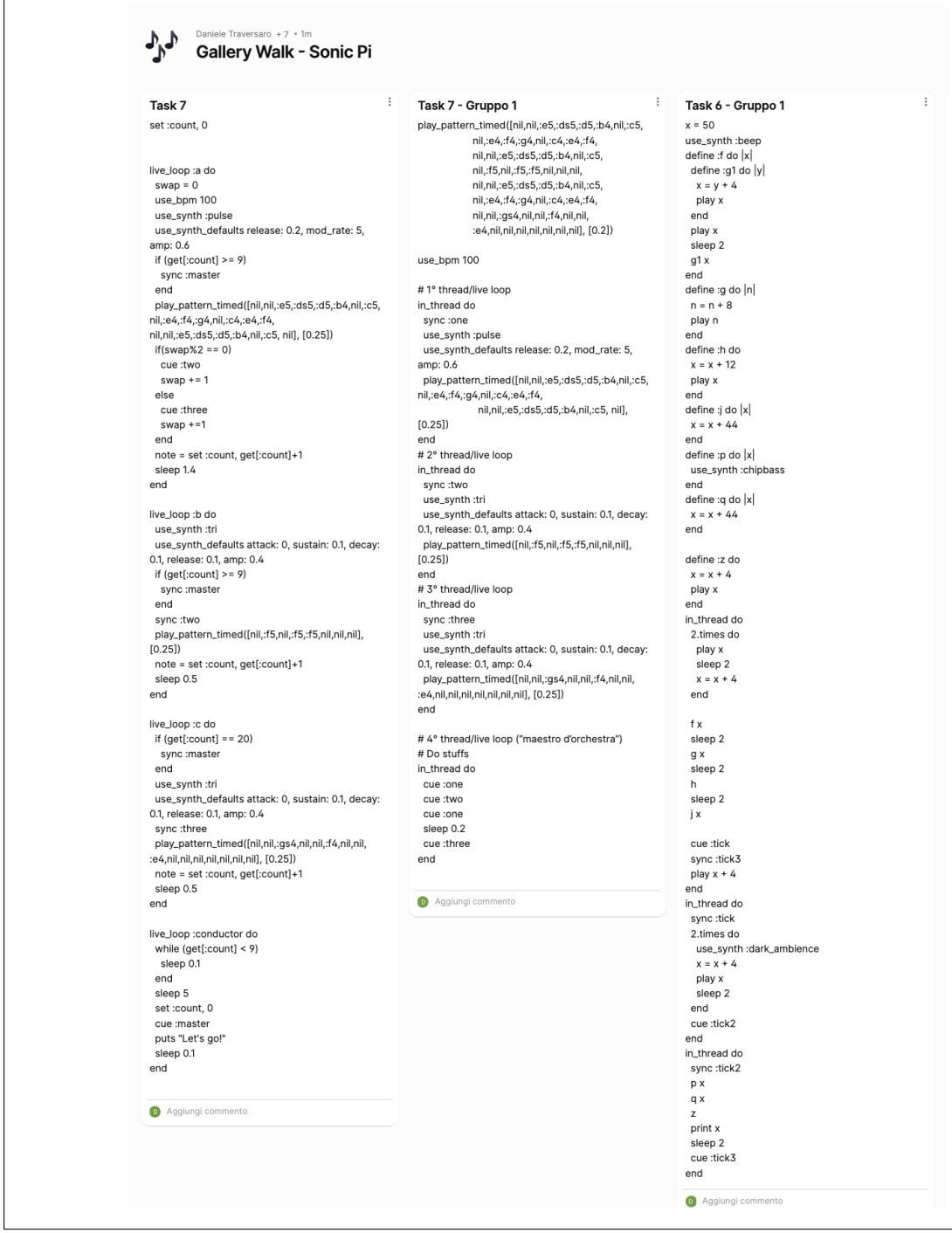

Figure A.4: Team Task 6 and 7 Solutions

Appendix B

Python for All

B.1 A Block-Based Introduction to Python

The teaching unit was structured as follows:

1. Algorithms, Flowcharts, Programs, and Executor
    - Understand algorithm design and representation using flowcharts.
    - Learn how to write and execute Python programs using EduBlocks.
2. Variables
    - Explore the concept of variables in Python.
    - Understand variable naming conventions and rules.
    - Learn how to assign values to variables.
    - Type Casting
3. Basic Data Types and I/O
    - Study the fundamental data types in Python, including integers, floats, strings, and booleans.
    - Learn how to perform operations and manipulations with these data types.
    - Print() and Input() statements
4. Comparison and Boolean Operators
    - Understand comparison operators (e.g., ==, $>$, $<$, etc.) and their usage.

- Explore boolean operators (e.g., and, or, not) for logical evaluations.

5. Conditional Control Structures
    - Learn how to make decisions using if, if-else, and if-elif-else statements in Python.
6. Python Lists
    - Explore Python lists, a versatile data structure for storing collections of elements.
    - Learn list manipulation techniques such as appending, indexing, and slicing.
7. Conditional Loop Structures
    - Iteration: for and while loops.
    - Learn how to control loop execution with break and continue statements (optional).
8. User-defined Functions & Pseudo-random Numbers
    - Study how to create custom functions in Python.
    - Learn how to generate pseudo-random numbers in Python using the random module.

The final assignments for each module require students to:

1. Devise an algorithm for computing the area and perimeter using the provided radius, and generate a corresponding flowchart for the algorithm.
2. Exchange the values of two variables, requiring students to manipulate the roles and assignments of the variables.
3. Receive two input numbers and store them in separate variables, then calculate and display their sum. To work with the numbers as integers, students need to convert the values from strings to integers.
4. Accept an input number and assess if it fulfills a specified condition using boolean and comparison operators.
5. Similar to the previous task, but incorporate an additional condition using an if-else statement.
6. Utilize a Python list to hold values like name, surname, and age. Based on the age value, output an appropriate message using an if-else statement.
7. Validate, for each element within a given list, whether its value matches a specified value. To tackle this, students will need to utilize both conditional and iterative statements.

8. *Final task.* Create a program to implement the "Guess the Number" game using the Random library, lists and user-defined functions, if desidered. In this game, the player provides input (guesses) to the game and receives feedback based on his guesses. The "game master" (the computer) plays the role of generating a random number within a predefined range and provides feedback on the player's guesses. The feedback varies based on the value the player enters; if the guess is less than the hidden number, the program will print a message indicating that it's too low. If the guess is greater than the hidden number, the program will print a message saying it's too high. Finally, if the guess is equal to the hidden number, the program will print a message indicating that the player has guessed correctly. The interaction between the player and the Game Master continues until the player successfully guesses the hidden number. The player's goal is to guess the number in as few attempts as possible.

B.2 A Physical Computing Approach to Introductory Python Programming

We formulated the following tasks:

1. In task 1, students were asked to implement code that would detect button presses and perform the appropriate actions. Pressing key A would increment the variable by 1, while pressing key B would display the updated value of the variable on the micro:bit's screen.

2. In task 2, students have to implement a program that handles button presses and updates the variable value accordingly. Pressing button A would increment the variable by 1, but only if the value was less than 6 (the maximum value a dice can take). Pressing button B would display the current value of the variable on the micro:bit's screen. They then had to implement an additional solution that would reset the count variable to 1 when button A was pressed and the count variable had a value of 6.

   Through this exercise, the students practiced conditional statements, variable manipulation, and screen display on the micro:bit. They also gained an understanding of how to restrict the range of possible variable values and how to respond to specific keystrokes.

3. The third task required changing the behavior of the program when keys A and B were pressed. Specifically, when button A was pressed, the program had to increment the counter variable by one and print its value. When button B was pressed, the program had to decrement the counter variable and display its value.

   By completing this exercise, students practiced handling button events, modifying variables based on user input, and displaying output accordingly. It provided an opportunity

to explore different button interactions and introduced the concept of dynamically manipulating variables in response to user actions.

4. The fourth task was similar to the previous one, but with a special requirement to display the symbol of the die face instead of the decimal number. Students had to use the Image class objects provided by the micro:bit library to represent each cube symbol. They also had to use the if-else construct to select and display the appropriate image based on the variable value.

   By implementing this exercise, students continued to practice conditional statements, manipulating variables, and using micro:bit's built-in features to create visually appealing output.

5. The fifth task, which built on the previous one, asked students to use the accelerometer functionality to detect shaking. When a "shake" was detected, the program had to generate a pseudo-random number using the random library and display the corresponding cube image.

   This exercise allowed the students to further explore other aspects, including sensor integration and random numbers, while increasing the interactivity and unpredictability of the program.

6. The sixth exercise required students to create a list of all the die faces (Image objects) and modify the code to use the elements in the list. By creating a list of all the die faces, students gained a more efficient and organized approach to handling multiple images. They then modified the code to access the images from the list based on the variable value or the randomly generated number. This implementation allowed for more flexibility because changing the cube faces only required changes within the list, rather than modifying the code for each individual image.

   This exercise encouraged students to explore the benefits of using lists to manage collections of data and improve code readability and maintainability.

7. Task 7 required the implementation of a Python function to be called on each event triggered by button presses or the accelerometer. The purpose of the function was to check the parity of a number. If the number was even, the symbol/image of a heart was displayed on the screen, followed by the image of the die face.

   Through this exercise, students gained an understanding of the concept and potential of functions and modular programming. They learned how functions can improve code structure, increase code reusability, and facilitate event-driven behavior.

8. Task 8 required simulating the roll of two dice by combining the values generated by shaking two micro:bits (each of which generates a pseudo-random number). The two microcontrollers exchange messages using the micro:bit's radio protocol (based on Bluetooth).

Through this exercise, students explored the concept of distributed systems and the message passing paradigm. They gained hands-on experience using the micro:bit radio protocol to exchange data between devices. They also gained experience with type casting because the radio protocol only sends string objects, not numbers.

B.3 A Data-Centric approach to Programming

On the first day, we covered the following topics:

1. Introduction to DS and data-driven paradigm.
    - Students grasped the importance of data in driving decisions and solutions across domains, establishing data science as a critical modern discipline.
2. Introduction to programming: algorithms, executors, programs, and programming languages.
    - Students learned the basics of programming, understanding how algorithms solve problems and how programming languages effectively communicate with computers.
3. Variables, data types, and dynamic typing.
    - Students gained insights into variables and data types, specifically focusing on integers, floats, and strings.
4. Tabular data structures: Series and DataFrame.
    - Students explored the value of these structures for efficient data manipulation and analysis, becoming essential tools for handling large data sets.
5. Comparison and logical operators applied to data (e.g., masks and filters).
    - Students discovered how to create masks and filters to extract and manage data based on specific conditions, enabling powerful data analysis tasks.

By the end of the first day, students had a basic understanding of Python programming. On the second day, we presented the following topics:

6. How to import CSV dataset.
    - Students will be guided on reading data from CSV files and storing it in a dataframe.
7. Basic data exploration techniques.

- Students will explore fundamental data exploration techniques, including examining data dimensions, getting data summaries.

8. Handling Missing Values, Duplicates, and Outliers.
    - Students will learn essential data cleaning techniques, ensuring data integrity and accuracy in their analyses.
9. Data Filtering
    - Students will master performing data filtering based on conditions using DataFrame APIs, resembling SQL queries.
    - Example: `movies_df[movies_df['Rating'] >= 8]`
10. Data Transformation
    - Students will learn how to apply custom operations to data and make changes to the dataset according to specific conditions.
    - Students will learn how to define and use functions and conditionals.
11. Data Visualization
    - Students will explore data visualization by creating graphs, charts, and maps using Matplotlib and other libraries.
    - Students will gain insight into how to interpret and communicate effectively with data.

B.4 Individual post-questionnaire

Likert scale quiz (0-4):

- How interesting did you find the proposed activities?
- How difficult did you find the proposed activities?
- How useful did you find the proposed activities?
- How helpful was the PCTO in clarifying your ideas about your future choices?
- Free comments

Programming questions:

1. Given the following expression: `not(not(A) and B)` and assuming that A and B can take the boolean values True and False. For which pairs of values A, B is the expression evaluated as False?

   - A and B True
   - A and B False
   - A = True, B = False
   - A = False, B = True

2. Given the following program:

```
x = 0
for i in range(5):
   print(x)
   x = x + 1
```

   What is the output of the program?

   - 1, 2, 3, 4, 5
   - 2, 3, 4, 5, 6
   - 0, 1, 2, 3, 4
   - 0, 1, 2, 3, 4, 5

3. Given the list: s = [1, 2, 44, 0]. Determine the value of: s[1], s[3], s[4]: (Given the series: s = series(1, 2, 44, 0). Determinare the value of: s.indexlocate[1], s.indexlocate[3], s.indexlocate[4])

   - 1, 44, 0
   - 2, 0, IndexError: list [series] index out of range
   - 1, 0, IndexError: list [series] index out of range
   - 1, 44, IndexError: list [series] index out of range

Appendix C

A Data Science Course for Non-Computing Majors

C.1 Lesson Plan

In this section we outline the lesson plan for the course and the learning content of each lesson. Please note that the above lesson structure is a general guideline and can be adapted according to the teacher's preferences and the students' needs. The time allotted for each section can be adjusted accordingly.

Lesson 1: Introduction to Data Science

The first lesson serves as an introduction to DS and focuses on cultivating a data-centric mindset. Students will be introduced to the concept of thinking with data and making ethical decisions based on data. The lesson is delivered in a frontal format where the teacher combines verbal communication with visual aids such as slides to effectively present examples.

What is Data Science?

- Briefly introduction of the importance of DS in today's digital age.
- Present data literacy as a fundamental 21st century skill for individuals to navigate an increasingly data-driven world, both in everyday life and in professional contexts.
- Highlight the role of data in driving insight, innovation and decision making.

- Define DS as a multidisciplinary field that involves extracting insights and knowledge from data.
- Emphasise the interdisciplinary nature of DS, combining elements of statistics, mathematics, CS and domain-specific knowledge.

Comparison with Other Terms: Big Data and Machine Learning

- Introduce the 5 V's model (Volume, Velocity, Variety, Veracity, and Value) as a framework for understanding the characteristics of Big Data.
- Explain how the 5 V's highlight the challenges and opportunities associated with processing, analyzing, and deriving insights from large datasets.
- Discuss how DS encompasses the methods and techniques used to extract meaning from data, while big data emphasises the challenges of dealing with large and complex datasets, and machine learning focuses on algorithms that enable computers to learn from data.

Ethical Considerations

- Delve into the principles of data ethics, fairness, and privacy.
- Explain the importance of obtaining informed consent, anonymising data where appropriate, and exercising due diligence when handling sensitive information.

By following this lesson structure, students will acquire a robust foundation in grasping the fundamental concepts of DS as an interdisciplinary field. They will also appreciate the importance of data literacy and develop a heightened awareness of the challenges and opportunities associated with data.

Lesson 2: Data Science Lifecycle with Tableau

The second lesson focuses on an introduction to data analysis and visualization using Tableau. Tableau simplifies the process by providing a visual, code-free interface. Each variable is represented as a draggable element and can be a measure or a dimension. To create a visualization, you simply drag and drop variables. The lesson is structured as follows:

Introduction to Data Visualization (10 minutes)

- What is data visualization?

  Practical examples (scatter plots, line plots, bar charts, histograms, etc.).

- The role of data visualization in data analysis and in storytelling.

  In data analysis, visualization serves as a powerful tool for identifying patterns, trends, and anomalies that may not be readily apparent from raw data. By transforming data into visually appealing representations, data visualization enables us to gain insights from large datasets and make informed decisions.

  In storytelling, data visualization serves as a compelling medium for communicating complex information to a wide audience. It conveys data-driven insights in an engaging and understandable way.

Introduction to Tableau (50 minutes)

- A quick overview of Tableau software.
- Import a dataset (e.g., a CSV file) into Tableau.
- Explore the data, in terms of rows (records) and columns.
- Identify a research question that will guide the entire project.
- Introduce the concept of Tableau column data types (e.g., numeric, text, geographic, etc.)
- Identify basic data quality issues (duplicate columns, null records, incorrect values w.r.t. the domain, inconsistent data types etc.)

Visualize the Data (30 minutes)

- Introduce students to the types of statistical variables in Tableau

  Tableau “measures” (quantitative variables) and “dimensions” (qualitative variables).

- Demonstrate how to create different visualizations such as bar charts, line graphs, scatter plots, and geographic maps using Tableau.

Lesson 3: Dashboards and data-driven Stories

In Lesson 3, students will gain hands-on experience with Tableau and develop skills in sharing and communicating with data. They will learn how to create powerful and meaningful visualizations and dashboards for effective data communication.

Choosing the most appropriate type of chart/graph (30 minutes)

- Emphasize the importance of selecting appropriate visualizations based on the type of data and the research question.

  Correct visualizations from a statistical point of view (considering the statistical data type with respect to the chosen graph).

  Effective from a communication point of view, emphasizing data visualization as a means of storytelling.

Dashboards (45 minutes)

- Collect appropriate visualizations in a Tableau dashboard.
- Add interactivity to dashboards.
- Emphasize the importance of creating visually effective dashboards to communicate effectively with data to different audiences.
- Learn the best practices for visualizing data:

  Less can be more. Minimize decorations and other "chart junk";

  Choose colors carefully

  Reduce the number of colors;

  Use only useful data (filter out all irrelevant data);

  ...

Lesson 4: Sharing and Communicating with Data

In this lesson, students will learn how to share accessible dashboards. Students will also reflect on visual best practices to make their visualizations inclusive and accessible. In fact, data visualizations are often inaccessible due to low color contrast, inadequate labeling, or images without alt text. This means that part of the audience may be excluded from the information presented, which should be avoided.

Tableau Stories

- Explain how to publish and share dashboards on the Tableau Public platform.
- Walk students through the process of uploading the dashboard to Tableau Public website.
- Discuss ethical and privacy issues (e.g., personal or sensitive data, etc.) related to data sharing
- Introduce students to Tableau Stories as a way to enrich the dashboard with multimedia elements such as text and images to create a more effective and meaningful visualization, emphasizing the importance of communication and storytelling skills.

Data Vizualization Accessibility

- Improving color contrast, properly labeling elements, adding text to graphs/charts, etc.
- Discuss the impact of inclusive data visualization design on data communication.

Lesson 5: Case-Study with Tableau (2 hours)

Hands-on The fifth lesson presents a hands-on, team-based activity that utilizes Tableau to provide students with the opportunity to apply their acquired data science skills in a real-world context. Working with a dataset of their choice, preferably related to their area of interest, such as cultural data, students will engage in a data-driven exploration to uncover meaningful insights.

A wealth of cultural and digital humanities dataset resources is readily available on the internet. For example, one reputable and valuable source is UNESCO[1], or the University of Pittsburgh[2].

Student groups are tasked with completing the assignment within a ten-day timeframe. The group assessment will be incorporated into the overall individual assessment, ensuring that teamwork and individual contributions are both valued.

Throughout the lesson, the instructor and teaching assistants will act as learning facilitators, providing guidance and timely feedback to students as they collaborate in their teams. This support will foster a collaborative learning environment and ensure that students effectively leverage their data skills to tackle the practical task at hand.

The activity will be structured as follows:

[1] http://uis.unesco.org

[2] https://pitt.libguides.com/findingdata/digitalhumanities

- Introduction to the Assignment: Students will be tasked with creating a data visualization project that showcases their proficiency in using Tableau for effective data communication.
- Data Import and Exploration: Students will import the provided dataset into Tableau, thoroughly exploring and understanding the data to gain insights into its structure and characteristics.
- Research Question Formulation: Students will formulate a specific research question that can be meaningfully addressed by analyzing the dataset, considering the audience they aim to communicate with.
- Data Quality and Cleaning: Students will perform basic data cleaning procedures to ensure the accuracy and integrity of the dataset, addressing any potential inconsistencies or anomalies.
- Visualization Creation: Students will create meaningful visualizations using Tableau to effectively answer the formulated research question. These visualizations should be tailored to the specific needs and preferences of the target audience.
- Dashboard: Students will organize the created graphs and charts into a comprehensive and effective dashboard that effectively communicates the research findings and insights.
- Presentation: Students will present their Tableau dashboard or stories to the wider community, effectively conveying their data-driven findings.
- Technical Report Preparation: Students will prepare a technical report, using word processing software, discussing the design choices made throughout the project, summarizing the key findings and results of the data analysis, and addressing ethical considerations relevant to data handling and presentation.

Deadline. The deadline for submitting the assignment is the end of the next two weeks.

Evaluation Criteria. The evaluation criteria should be provided to students prior to the assignment. The evaluation covers three dimensions: quality of data analysis, quality of data presentation/communication, and quality of the technical report. Instructors should provide constructive feedback to each group to support their learning and improvement, and summarize the take-home message from the assignment. Refer to Table C.1 in Appendix for the details.

Lesson 6: Introduction to Python Programming

This lesson provides an introductory overview of programming concepts and syntax, with a focus on addressing common misconceptions. The instructor will lead in-class demonstrations

and exercises to ensure students grasp the fundamental concepts of programming, such as the distinction between variable names and string values, and the difference between expressions and statements.

Why Programming? (45 minutes)

- From Tableau to Python: Highlight the advantages of using a programming language (e.g., greater customization, rich collection of libraries, etc.).
- Introduction to algorithms, programs, and executors.
- Introduction to Google Colaboratory and Jupyter Notebooks.

Python as a Calculator (45 minutes)

- Using Python to perform basic arithmetic operations.
- Introducing variables (variable name, variable assignment etc.)
- Present three basic data types: integer, float, string.

Lesson 7: File Input/Output, Conditional Statements and For Loop

In Lesson 7, students will explore importing and manipulating text files using conditionals and iteration. Before diving into the new concepts, students will engage in a collaborative activity using the TPS teaching methodology to reinforce their understanding of the topics covered in the previous lesson.

TPS Activity (20 minutes) To effectively review the concepts from the previous lesson, the instructor can employ the TPS teaching methodology, guiding students through a collaborative activity that focuses on swapping variable values using a temporary variable.

5 minutes for individual work, 10 minutes for pair work, 5 minutes for plenary discussion.

File Input / Output (70 minutes)

- Introduction to Python files I/O.
- How to read rows and print lines from a file: Introducing iteration (for loop).

- How to filter data: introduction to conditional statements (if-else) and indexing.

For example, to read a .csv file line by line:

```
import csv

with open('dati.csv', 'r') as file:
    reader = csv.reader(file)

for row in reader:
    print(row)
```

To filter data according to a condition:

```
import csv

reader = csv.reader(file)

for row in reader:
    # Suppose column 2 contains numeric age values and we want
    to print only rows with a value greater than 18.
    if row[2] > 10:
        print(row)
    else:
        print("The value does not satisfy the condition.")
```

The latter examples exposes students to the concept of indexing. In fact, each record of the file is a Python list, where each column is identified by a numerical index. Specifically, the index of the first column is 0, the second column is 1, and so on.
Notes: The course only introduces for loops, but does not cover while loops or recursion. This is beyond the scope of our learning goals, as students will focus on Pandas, which uses a declarative style where iteration is implicit in the dataframe idiom.

Lesson 8: Functions

The objective of this lesson is to introduce students to the concept of functions in Python and enable them to understand the benefits of using functions, how to define and call functions, and how to work with parameters and return values.

Introduction (30 minutes)

- Advantages of using functions: code reusability, organization, and abstraction.
- Explain the syntax of defining a function in Python.
- Demonstrate how to define a simple function without parameters and return values.
- Encourage students to create and call their own simple functions.

Parameters and Return Value (30 minutes)

- Explain the concept of function parameters and their role in passing data to functions.
- Show examples of functions with different types of parameters.
- Guide students in creating functions with parameters and calling them with various arguments.
- Discuss the use of the *return* statement in functions to return value.
- Explain the significance of returning values from functions.
- Show examples and guide students in creating functions with return value.

Scope and Lifetime of Variables (30 minutes)

- Introduce the concepts of variable scope, local and global variables.
- Show examples of variable scope within and outside of functions.

Lesson 9: Python Lab (2 hours)

In this lab, students will work in groups to gain hands-on experience with importing, processing, and manipulating data from CSV files. Students will start by importing a CSV file using the standard Python library. Then, they will store the data in a variable and print the data row by row, filtering the data to print only those rows that meet a given condition on one or more attributes. Finally, students will define a custom function to transform the value of a given attribute and print the updated values.

During the lesson, the instructor and teacher assistants will serve as learning facilitators, guiding students and providing immediate feedback. This lab is considered ungraded and no formal submission is necessary.

Lesson 10: Introduction to Pandas

By the end of this lesson, students should be familiar with the basics of data manipulation and analysis using Pandas.

Introduction to Pandas (20 minutes)

- What is Pandas and why is it popular in data analysis?
- Key features and benefits of using Pandas.
- Instructions to import/install Pandas and its dependencies.
- Introduction to Series: Understanding one-dimensional labeled arrays (similar to a column in a spreadsheet).
- Introduction to DataFrame: Introducing two-dimensional labeled data structures to store tabular data (similar to a spreadsheet).

Data Import and Basic Operations (30 minutes)

- Importing data (e.g., CSV, Excel, etc.)
- Exploring data using head(), tail(), info(), describe(), and shape.
- Data selection, indexing, and slicing.
- Explore built-in functions in Pandas (e.g., max, mean, average) and apply them to a single column to compare them to the file-based approach, emphasizing the simplicity and efficiency of using these functions compared to manual iterations.

Data Cleaning and Preprocessing (30 minutes)

- Handling missing data using dropna() and fillna().
- Removing duplicates with drop_duplicates().
- Renaming columns and handling data types.

Lesson 11: Data Analysis in Pandas

Students will delve into the Pandas library. They will be introduced to aggregation functions, sorting, and complex multi-column boolean conditions to filter and manipulate data.

Data transformation operations (45 minutes)

- Filtering data using boolean indexing.

  Introduction to boolean data types and bitwise operators.

- Grouping data with groupby() and performing aggregate functions such as sum(), mean(), count(), etc.
- Using the pivot() function to transform data between wide and long formats.

Hands-on (45 minutes) Students will work on an guided notebook to practice data transformation operations using Pandas. The dataset should be tailored to students' interest. The instructor will provide assistance and answer questions throughout the lab.

Lesson 12: Advanced data transformations in Pandas

Students will delve deeper into Pandas, exploring merge operations and handling missing values and duplicates. They will also learn how to apply functions to columns.

Data Manipulation

- Summary of Data Transformation in Pandas.
- Merging DataFrames (inner, outer, left, right merge).
- Concatenating two DataFrames, vertically and horizontally.
- Handling of mismatched indexes and columns during concatenation[3].
- Applying user-defined functions to columns using the apply() method to transform data.
- *Optional.* Gain an understanding of anonymous functions (lambda functions) in Python. Learn how to apply anonymous functions inline using the apply() method.

[3]This problem does not occur when using ToyPandas.

Hands-on (45 minutes) Students will work on a Jupyter notebook to practise data transformation operations using Pandas. The instructor will be available throughout the lab to provide assistance and answer questions.

Lesson 13 & 14: Data Visualization with Matplotlib

By the end of this lesson, students should have a basic understanding of data visualization in Python and be familiar with the Matplotlib library.

In particular, students will gain hands-on experience in creating various types of plots, including line plots, scatter plots, bar charts, histograms, and subplots. They will also explore advanced plotting techniques, such as box plots, and gain an introduction to additional plotting libraries like Seaborn and Plotly.

- Installing/importing Matplotlib.
- Creating a basic plot using Matplotlib's interface.
- Customizing plot appearance (colors, labels, titles, legends, etc.).
- Line Plots and Scatter Plots.

  Creating line plots to visualize trends and patterns in data.

  Generating scatter plots to visualize the relationship between two variables.

  Adding markers, colors, and labels to enhance plot readability.
- Bar Charts and Histograms.

  Creating bar charts to compare categorical data.

  Customizing bar charts with different styles and colors.

  Understanding histograms and their use in visualizing data distributions.
- Advanced Plots with Matplotlib.

  Subplots: Arranging multiple plots in a single figure.

  Box-plot: visualize the center and distribution of the data.
- Introduction to other plotting libraries (choose one or more libraries, such as Seaborn, Plotly, or ggplot, wordcloud etc.).

Lesson 16: Pandas Lab

In this lab, students will be given a partially completed notebook to complete. The notebook will take them through the various steps they have learnt in previous lessons, from importing data to creating simple plots using Python libraries. ere are some specific tasks that students will be asked to complete:

- Import data from a CSV file
- Clean and format the data
- Create simple graphs and plots using Matplotlib or other libraries.
- Customize the appearance of the plots.

This lab is designed to help students solidify their understanding of the concepts they have learned and to gain hands-on experience with using Python for data visualization.

The instructor and any teaching assistants will serve as facilitators/tutors, helping students as needed.

If students do not complete the lab in class, they can complete it and submit it on the course website. The lab assignment is not graded, but it is required for students to complete the course.

Lesson 16 &17: Introduction to Machine Learning - NLP

In this lesson, students will be introduced to the fundamental concepts of machine learning using the scikit-learn library in Python.

Introduction to Machine Learning (15 minutes)

- Define machine learning and its role in various fields, including digital humanities.
- Introduce the concept of natural language processing (NLP) and its applications in text analysis.
- Introduce other examples including social media analysis and literary analysis.
- Explain the concept of training data and model prediction.

The Machine Learning Workflow (20 minutes)

- Present the basic steps of the machine learning workflow:
  - Data collection and preparation.
  - Model selection.
  - Model training.
  - Model evaluation.
  - Prediction using the trained model.

Hands-on Activity 1: Text Classification (60 minutes) To introduce text analysis, it is suggested to provide a notebook with Naive Bayes, as it is a very simple algorithm that is already implemented in the scikit-learn library. It is also suggested to choose a simple dataset, such as news articles categorised by topic.

- Explain the concept of Naive Bayes and its underlying principles.
- Discuss the pros and cons of using Naive Bayes for text classification.
- Import the necessary libraries in Python.
- Import a dataset of text documents labeled with various categories.
- Preprocess the text data, such as removing punctuation and converting to lowercase.
- Create a vocabulary of unique words and their frequencies in each category.
- Calculate the conditional probabilities of different words given a class label.
- Use the Naive Bayes algorithm to classify new text documents into the appropriate category.
- Practice building a Naive Bayes classifier using Python on a sample dataset.
- Observe how the classifier performs on new text documents.
- Experiment with different parameters and techniques to improve the classifier's accuracy.

Hands-on Activity 2: Sentiment Analysis (60 minutes) Students will be introduce to sentiment analysis. We suggest to use a simple dataset of social media posts with sentiment labels for initial practice, such as tweets categorized as positive, negative, or neutral.

- Introduction to Sentiment Analysis and Social Media
- Define sentiment analysis and its role in social media analysis.
- Discuss the benefits of sentiment analysis for understanding public opinion, identifying trends, and analyzing social phenomena.
- Introduce social media platforms and their role in digital humanities research.
- Present Naive Bayes algorithm for sentiment analysis, and introduce the concept of sentiment lexicons and their applications in sentiment analysis.
- Provide a dataset of social media posts with sentiment labels.
- Students have to preprocess the text data, such as removing punctuation, converting to lowercase, and handling emojis.
- Create a vocabulary of unique words and their frequencies in each sentiment class.
- Calculate the conditional probabilities of different words given a sentiment label.
- Apply the Naive Bayes algorithm to classify new social media posts into the appropriate sentiment category.
- Evaluate the performance of the sentiment analyzer using accuracy, precision, and recall metrics.
- Create some visualisations, such as a word cloud, histogram and so on.

Lesson 17: Presentation of the Final Project

This lesson focuses on introducing the final team project, where students will analyze a real-world dataset of their choice. The project will require them to collect, prepare, and transform data, analyze it, and create meaningful graphical visualizations. Each team will be required to:

- Analyze data using Python libraries, especially prepare a Jupyter Notebook Python in which all analyses and visualizations are properly annotated and presented.

- Prepare a multimedia presentation showing the overall project, the target audience, research question, results, ethical considerations, as well as discussing limitations, possible future directions, organization of teamwork, and a reflection on the data literacy and soft skills acquired.

The instructor needs to make explicit:

- The purpose and objectives of the final project.
- Provide guidelines for selecting a suitable dataset relevant to their interest, introducing popular data sources like Kaggle, etc.
- Clarify the evaluation criteria (see Section C.2), and the project submission deadline.
- Provide additional resources or tutorial if needed.

Students may submit an abstract/proposal of their project (e.g., selected dataset and research question) to the instructor for preliminary feedback before starting the project.

C.2 Evaluation Grids

This appendix provides the evaluation grid for the Tableau project and the final project. Refer to table C.1 for the details of the tableau assignment, C.2 for the details of the final project grid.

Aspect	Criteria and Description	Rating (1-5)
Data Analysis	Number of Graphs and Correct Use - Number of relevant and insightful graphs for the research question - Correct use of graph types based on variable types (categorical, numerical) - Appropriate use of aggregation functions (mean, median, etc.)	
Data Communication	Effective and Visually Appealing Dashboards - Filter and visualization of only relevant data to the research question - Appropriate choice of visualization types to convey information - Effective use of labels, additional text, and colors to enhance understanding	
Technical Report	Clarity and Completeness of the Technical Report - Clear explanation of design choices - Clear presentation of research question and intended audience - Well-presented results and insights - Thorough consideration of ethical implications - Inclusion of metacognitive elements (difficulties faced, limitations, etc.)	

Table C.1: Tableau Assignment Evaluation Grid: with Likert scale: Excellent (5); Good (4); Satisfactory (3); Needs Improvement (2); Poor (1).

Criterion	**Evaluation (1-5)**
Data Understanding - Chose a meaningful and relevant dataset, explaining the choice - Demonstrated a clear understanding of the data through its description	
Data Exploration - Used Pandas functions to load and examine the data - Identified data types, presence of missing values, and key features of the dataset etc.	
Identifying Research Question - Formulated a clear and relevant research question	
Data Analysis - Applied appropriate techniques to answer the research question - Performed correct calculations/statistics using Pandas	
Visualizations - Created meaningful and appropriate visualizations using Matplotlib or other plotting libraries - Annotated visualizations clearly and accurately (show only relevant data, legend, alternative text, color contrast etc.)	
Presentation - Prepared a well-structured and documented Jupyter notebook - Presented results logically and coherently - Discussed ethical and privacy considerations	
Code Proficiency - Wrote clean, readable, and efficient Python code - Commented code clearly and understandably	

Table C.2: Final Project Evaluation Grid with Likert scale: Excellent (5); Good (4); Satisfactory (4); Needs Improvement (3); Insufficient (2).