

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Towards a Live Programming Platform for K-12

Filipa Manita Santos Durão



Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Prof. Ademar Aguiar

July 8, 2021

Towards a Live Programming Platform for K-12

Filipa Manita Santos Durão

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Prof. António Coelho

External Examiner: Prof. Jorge Simões

Supervisor: Prof. Ademar Aguiar

July 8, 2021

Abstract

As the relevance of technology rises in the world, it is getting increasingly important that people learn its fundamentals from a young age. The early learning of Computer Science would enable people to understand better how computers work and provide them with critical thinking in the computational area. Computation is not just about being able to use a computer, but also the structuring of thought and the capability of dividing problems to easily solve them.

Currently, in many countries, including Portugal, Computer Science is not part of the schools' curriculum. In the effort to change this situation, multiple worldwide organizations have worked towards a proper Computer Science curriculum. However, some skills learned through this kind of subject need to be practiced and, as such, proper environments must exist for students to develop their skills.

The review of the state of the art done during this work showed the multiple advantages of learning Computer Science from a young age and which countries have already implemented this subject into their curricula. Further research also demonstrated that there are many programming platforms for children, such as Scratch or Tynker, but they all lack requirements that this work deemed worthy of exploring. This gap was the main focus of this thesis.

To address these shortcomings, we developed Loki, a customized Eclipse distribution that features Live Programming in Python, Automated Testing and Collaborative Development. Loki was developed from a regular Eclipse for Java, to which were added the necessary plugins for all the features required and customized the UI to make it simpler. Loki is shareable and executable in Windows, macOS and Linux environments.

We validated Loki during development, performing small use-case tests to ensure all features were working properly. Afterwards, an experiment with college students was performed, where they used Loki to solve a Python exercise sheet, to study the influence of Live Programming on programming efficiency.

Additionally, we surveyed and interviewed ENSICO Master Teachers, experts in teaching Computer Science in schools, about their opinion on this platform. This was important, since this platform is aimed at 10 to 14-year-old students, but experimenting with them was not possible. Overall, they believed Loki to be useful and appropriate for their students. The survey and interview results also showed that Live Programming is a valuable asset in the students' programming learning process.

Keywords: K-12, Live Programming, Computer Science, Computational Thinking, Python

Resumo

À medida que a importância da tecnologia aumenta no mundo actual, torna-se cada vez mais importante que todos aprendam os seus fundamentos desde a mais tenra idade. A aprendizagem de Ciências da Computação desde cedo permitirá que a população tenha uma melhor compreensão de como os computadores funcionam e dar-lhe-á um pensamento crítico na área computacional. A Computação não é apenas saber utilizar um computador, é também a estruturação do pensamento, a capacidade de dividir os problemas para os resolver facilmente.

Actualmente, em muitos países, incluindo Portugal, a disciplina de Ciências da Computação não faz parte do currículo das escolas. Num esforço para alterar esta situação, várias organizações mundiais têm trabalhado no sentido de criar um currículo adequado para esta disciplina. No entanto, as competências adquiridas através deste tipo de disciplinas precisam de ser praticadas e, como tal, devem existir ambientes adequados que os estudantes possam utilizar.

A revisão do estado da arte deste trabalho mostrou as múltiplas vantagens da aprendizagem de Ciências da Computação desde tenra idade e quais os países que já implementaram esta disciplina nos seus currículos. Pesquisas posteriores mostram também que existem diversas plataformas de programação para crianças, tais como Scratch ou Tynker, mas todas elas carecem de requisitos que este trabalho considera pertinentes para exploração. Esta lacuna será o foco principal da presente tese.

Para abordar esta lacuna, desenvolvemos o Loki, uma distribuição personalizada de Eclipse, que apresenta as funcionalidades de Live Programming em Python, Testes Automatizados e Desenvolvimento Colaborativo. O Loki foi desenvolvido a partir do Eclipse para Java, ao qual foram adicionados os plugins necessários para todas as funcionalidades pretendidas, e a sua interface foi simplificada. O Loki é partilhável e executável em Windows, macOS e Linux.

Validámos o Loki durante o seu desenvolvimento, realizando testes com pequenos casos de utilização para garantir que todas as funcionalidades estavam ativas. Mais tarde, foi realizada uma experiência com estudantes universitários, onde usaram o Loki para resolver uma ficha de exercícios de Python, em que pretendíamos estudar a influência do Live Programming na eficiência a programar.

Adicionalmente, realizámos um inquérito online e uma entrevista com os Master Teachers da ENSICO, especialistas no ensino de Ciências da Computação em escolas, para saber a sua opinião sobre o Loki. Este passo da validação foi importante, pois esta plataforma tinha como público alvo alunos dos 10 aos 14 anos de idade, mas realizar experiências com eles não foi possível. Globalmente, acharam que o Loki seria útil e apropriado para as idades dos alunos em questão. Os resultados dos seus inquéritos e entrevistas também demonstraram que Live Programming é um bem valioso no processo educativo da programação.

Keywords: K-12, Live Programming, Computer Science, Computational Thinking, Python

Acknowledgements

Ao meu orientador, o Professor Ademar Aguiar, que, para além de sempre disponível para qualquer dúvida, batalhou para que, num ano complicado como este, eu tivesse as melhores oportunidades possíveis. Foi incansável, e, sem ele, este trabalho não existiria.

Aos meus pais, Egídio e Manuela, que foram um apoio incondicional, não apenas durante esta tese, mas durante toda a minha vida. Desde nova me incentivaram a seguir o que me fazia feliz, e sempre alimentaram a minha confiança que, desde que trabalhasse o suficiente, seria capaz de tudo a que me propusesse. Sem eles, não teria sido capaz de chegar onde cheguei.

À minha irmã Leonor e à sua confiança cega na minha pessoa, e que foi a minha escapatória da seriedade deste trabalho sempre que precisei. A sua boa disposição alegrou os meus dias.

Ao meu namorado Rui, que me apoiou durante todo este percurso, que esteve sempre disponível para mim e que foi, por muitas vezes, a primeira cobaia para testes. Nunca me deixou desesperar, e sempre me incentivou a olhar em frente e continuar.

A todo o resto da minha família e amigos que estiveram sempre aqui para mim, especialmente a Avó Manita, o Avô Egídio, a Avó Branca e o Avô Porfírio.

A todos os participantes da experiência de validação, não imaginam o impacto que tiveram no meu trabalho e o quanto me ajudaram.

Obrigada a todos,

Filipa

*“Science is not only a disciple of reason
but, also, one of romance and passion.”*

Stephen Hawking

Contents

1	Introduction	1
1.1	Context	1
1.2	Motivation	2
1.3	Research Goal	2
1.4	The Problem	2
1.5	Proposed Contribution	3
1.6	Validation	4
1.7	Document Structure	4
2	State of the Art	7
2.1	Methodology	7
2.1.1	Databases	7
2.1.2	Process	8
2.2	Digital Literacy and Digital Skills	8
2.3	The Importance of Computer Science in Schools	8
2.3.1	Fundamental Technology Concepts	8
2.3.2	Computational Thinking and Problem Solving Capabilities	9
2.3.3	College Enrollments	9
2.4	Computer Science Curriculum in Other Countries	10
2.4.1	England	10
2.4.2	United States of America	11
2.4.3	New Zealand	11
2.4.4	Israel	11
2.5	Live Programming	12
2.6	Best Development Practices: A Subset for K-12	14
2.6.1	Collaborative Development	14
2.6.2	Version Control	14
2.6.3	Automated Testing	14
2.7	Programming Environments	15
2.7.1	Coding Platforms for Children and Teenagers	15
2.7.2	Programming Environments Supporting Liveness	19
2.7.3	Customizable Programming Environments	21
2.7.4	Python Live Programming Environments	23
2.8	Summary	24

3	Problem	25
3.1	Research Problem	25
3.2	Scope	26
3.3	Hypothesis	26
3.4	Envisioned Solution	26
3.5	Research Strategy	27
3.6	Validation	28
3.7	Summary	28
4	The Loki Environment	29
4.1	Architecture	29
4.2	Features	30
4.2.1	Python Support	30
4.2.2	Live Programming	31
4.2.3	Collaborative Development	32
4.2.4	Others	33
4.3	User Interface Development	34
4.4	Eclipse Customized Distribution Sharing	36
4.4.1	Oomph	36
4.4.2	Yatta Profiles	36
4.4.3	Distribution Package	37
4.5	System Requirements And Running Instructions	38
4.6	Using Loki: An Example	39
4.7	Summary	42
5	Validation	43
5.1	Live <i>versus</i> Not Live: Experiment With College Students	43
5.1.1	Designing the Experiment	43
5.1.2	Selecting the Subjects	44
5.1.3	Running the Experiment	45
5.1.4	Processing the Results	46
5.2	Live <i>versus</i> Not Live: Experiment With 10-14 Year Olds	49
5.3	Collaborative <i>versus</i> Individual Development: Experiment With 10-14 Year Olds	50
5.4	Expert Evaluation	50
5.4.1	Description	51
5.4.2	Results	51
5.5	Summary	54
6	Conclusions and Future Work	57
6.1	Conclusions	57
6.2	Contributions	58
6.3	Challenges	58
6.4	Future Work	59
	References	61

A	Platform Installation and Usage	67
A.1	Python installation Script	67
A.2	Guides Sent to Experiment Participants	67
A.2.1	Installation Guide	67
A.2.2	Usage Guides	72
A.3	Usage Guides For Children	74
B	Moodle Python Test	75
B.1	Moodle Python Test for Experiment Participants	75
C	ENSICO Master Teachers' Survey	79
C.1	Python Live Programming Platform Review	79
C.1.1	The Platform's Influence on the teaching of programming at schools . . .	79
C.1.2	Development of Computational Thinking and its Advantages	80
C.1.3	Final Remarks	81
D	Experimental Results	83
D.1	First Experiment Results	83
D.2	Expert Evaluation Answers	92
D.2.1	Master Teacher Luís Neves - Questionnaire Answers	92
D.2.2	Master Teacher Rui Grandão - Questionnaire Answers	93
D.2.3	Master Teacher Inês Guimarães - Questionnaire Answers	94
D.2.4	Master Teacher Liliana Monteiro - Questionnaire Answers	95

List of Figures

2.1	Tanimoto's Liveness Level Hierarchy.	13
2.2	Scratch Programming Environment.	16
2.3	Tynker Programming Environment.	16
2.4	Code.org Programming Environment.	17
2.5	Swift Playgrounds Programming Environment.	18
2.6	Alice Programming Environment.	19
2.7	Eclipse Integrated Development Environment.	22
2.8	Visual Studio Code Editor.	22
2.9	Online Python Tutor.	23
3.1	Design Science Research Process.	27
4.1	Component Diagram.	30
4.2	Eclipse Liveness Panel.	31
4.3	Eclipse Liveness Panel - Turtle Graphics.	31
4.4	Saros Collaborative Session - User 1.	32
4.5	Saros Collaborative Session - User 2.	33
4.6	Code Time Dashboard.	34
4.7	Code Time Online Dashboard.	35
4.8	Original Eclipse Menus and Tools.	35
4.9	Eclipse Menus and Tools on Final Distribution.	35
4.10	Yatta Profiles Platform.	37
4.11	Eclipse Folder Structure.	38
4.12	Python Interpreter Choice Menu.	39
4.13	Initial Loki Screen.	40
4.14	Loki- Python Live Programming.	40
4.15	Loki - Turtle Live Programming.	41
4.16	Loki - Automated Tests.	41
5.1	Distribution of the Volunteers by College Year.	45
5.2	Average Number of Exercises Solved by Group.	47
5.3	Relative Frequency of Number of Students to (Fully or Partially) Solve Each Exercise in Each Group.	47
5.4	Relative Frequency of Number of Exercises Solved in Each Group.	48
A.1	Code Time Setup Step 1.	68
A.2	Code Time Setup Step 2.	69
A.3	Code Time Setup Step 3.	69
A.4	Code Time Setup Step 4.	70

A.5	Code Time Setup Step 5.	70
A.6	Group A Instructional Image.	72
A.7	Group B Instructional Image.	72
A.8	Group B Console.	74
D.1	Distribution of the Participants by College Year.	84
D.2	Distribution of Participants by College Year in Both Experimental Groups.	84
D.3	Distribution of Participants by College Year in the Liveness Group.	85
D.4	Distribution of Participants by College Year in the Control Group.	85
D.5	Absolute Frequency of Number of Exercises Solved in Each Group.	88
D.6	Absolute Frequency of Number of Students to (Fully or Partially) Solve Each Exercise in Each Group.	90

List of Tables

2.1	Platforms' features comparison.	20
5.1	Experiment Results Summary.	48
5.2	Experiment Results Summary - Removing Outliers.	49
D.1	Distribution of the Volunteers by College Year.	83
D.2	Distribution of the Participants by College Year.	83
D.3	Liveness Group Experiment Results on Each of the 13 Test Exercises.	86
D.4	Control Group Experiment Results on Each of the 13 Test Exercises.	86
D.5	Absolute Frequency of Number of Exercises Solved in Each Group.	87
D.6	Relative Frequency of Number of Exercises Solved in Each Group.	89
D.7	Absolute Frequency of Number of Students to Solve Each Exercise in Each Group.	90
D.8	Relative Frequency of Number of Students to Solve Each Exercise in Each Group.	91

Abbreviations

ACM	Association for Computing Machinery
AT	Automated Testing
BDP	Best Development Practices
CAS	Computing at School
CD	Collaborative Development
CS	Computer Science
CSAPC	Computer Science Advanced Placement Course
CSTA	Computer Science Teachers Association
CT	Computational Thinking
ICT	Information and Communication Technology
IDE	Integrated Development Environment
JRE	Java Runtime Environment
K-12	From Kindergarten to 12th Grade
LP	Live Programming
OO	Object-Oriented
UI	User Interface
USB	Universal Serial Bus
VC	Version Control
VS Code	Visual Studio Code

Chapter 1

Introduction

1.1	Context	1
1.2	Motivation	2
1.3	Research Goal	2
1.4	The Problem	2
1.5	Proposed Contribution	3
1.6	Validation	4
1.7	Document Structure	4

In a society where technology’s role is increasingly more relevant, it is crucial that everyone knows how to use it. Learning Computer Science (CS) from a young age has shown evidence of improving children’s cognitive skills and problem-solving abilities [AVR20].

Furthermore, we believe that understanding the deeper functioning of the technology used daily can also improve the way we use it.

1.1 Context

In Portugal, Computer Science is not yet a widespread subject in schools. However, some projects and organizations have strove to implement this area of knowledge in public schools. ENSICO (Associação para o Ensino da Computação / Association for Computation Education) [Ens21], for example, is an organization devoted to create a CS curriculum and pursuing its implementation. One of the most important goals for this subject is to help the students develop Computational Thinking skills.

Computer Science is a scientific discipline that comprises a multitude of concepts, such as programming, problem-solving, algorithms, among others. As a scientific discipline, it features rigorous concepts, methods and techniques, one of which is Computational Thinking [WDB⁺17]. On the other hand, Computational Thinking is the action of recognizing and solving real-world problems and situations by applying Computer Science techniques and tools. Shortly, it consists

of collecting data and analyzing it to understand a problem and decompose it into smaller and more manageable problems [WDB⁺17].

1.2 Motivation

There is evidence that learning CS from a young age improves students' cognitive skills and planning abilities [AVR20], and some state that developing Computational Thinking (CT) is essential to all subjects *"through the processes of problem solving and algorithmic thinking"* [VFG⁺15]. Studies have also shown that it can improve students' likelihood to enroll in college up to 38% [BBP20].

Furthermore, we believe that when people know how something works internally, its usage is facilitated. They can use it intuitively and infer new ways to operate it. We believe the same goes for technological areas, such as CS. If learning these areas of knowledge starts from an early age, we are confident that the children who study this will grow up to be adults who know how to handle technology and use it to their own benefit [Lod20].

1.3 Research Goal

This work aims to provide a better programming learning experience for students from 10 to 14 years of age by providing a platform where they can code inside and outside the classroom.

Deriving from this primary goal, it is expected that, the more children and teenagers learn how to properly use technology from an early age, the more can be achieved in terms of a more technologically literate population that not only knows how to use existing technology but also how it operates internally, and, more importantly, how to use it for their purposes and needs.

1.4 The Problem

The teaching and learning of CS is comprised of multiple different activities, one of which is programming. We believe programming to be essential in learning CS, as it creates a tangible mean with which the students can interact and understand the concepts.

Currently, CS is not part of most of the schools' curriculum. In order to be integrated, there are two crucial points to be provided:

- A CS curriculum suitable for the age-group concerned.
- An adequate set of tools where the students can apply and practice the knowledge gathered.

The first item is being worked on for a while by some entities worldwide, and a summary of those efforts can be found in Section 2.4.

We believe that, in order for a CS curriculum to be efficiently implemented, a proper set of tools for the students to practice their work is crucial sooner or later in the learning process.

Several platforms exist, but they either follow their own curriculum, through puzzles and pre-established activities, or use specific programming languages out of the focus of this thesis. A summary of these platforms can be found in Section 2.7.1.

Live Programming (LP) is also an important concept studied in this work, so research on coding platforms that support LP is presented in Section 2.7.2.

To guide this work, the problem was decomposed into a list of questions that framed the research to be done:

- **Research Question 1:** *Does using customized development environments benefit students learning CS at school?*
- **Research Question 2:** *Which programming environments exist for CS at school and which requirements do they fulfill?*
- **Research Question 3:** *How much can having a fun and playful programming environment help children focus on their work and keep their interest on CS for longer?*

We believe these questions covered the entire subject being studied and could provide a wide range of useful results when adapted to research queries.

1.5 Proposed Contribution

The contribution of this work, to help better learning of CS in the context of K-12, specifically 10 to 14 years old, was to conceptualize and implement a platform that students can use inside and outside of the classroom to learn and practice programming in a more engaging and playful way.

This development environment must contain important features of a classic development environment but adapted to the target age. It must be intuitive to use and appealing to young users and be flexible to adapt to curricula as needed. It should also be as interactive as possible to engage the students' attention without losing their interest [CZW14].

To guide the development, a set of requirements we deemed essential was defined:

- Supporting programming in Python, which is an easy-to-learn and popular programming language that has useful features and easy to read code [Bri13]. It's also suitable for a wide range of ages and expertise, from 6 years old to professionals.
- Being simple to use, so it wouldn't demotivate the students using it.
- Featuring Live Programming, so it becomes more interactive and engaging for students, to help them develop good solutions faster.
- Featuring Collaborative Development, so students can learn to cooperate and solve problems together, to share the knowledge and develop critical thinking.

- Featuring Automated Testing support, so the students can keep track of their solutions themselves and so the teachers can easily evaluate student's work.
- Featuring Version Control, for the students to feel more confident to experiment new solutions without the fear of losing their previous work.

The final result of this work is Loki, a programming environment supporting Live Programming in Python, Collaborative Development and Automated Testing capabilities, developed from the Eclipse IDE.

1.6 Validation

The platform described above, Loki, was validated in three different ways:

- Small test cases, performed by ourselves, to ensure that all the requirements are being met.
- An expert evaluation, where opinions and suggestions were collected and used to improve the work.
- An experiment with college students, where they used Loki to solve a programming exercise sheet. The goal of this experiment was to evaluate if Loki brought any change in efficiency facing those who did not use a LP environment.

We had also planned experiments featuring students within the age range that this platform was developed for, however, due to the COVID-19 pandemic, those were not possible.

1.7 Document Structure

This document is organized into the following six chapters:

- First, the **Introduction** (Chapter 1), describes the context in which this work is inserted in, as well as its main goals.
- Further ahead, the **State of the Art** is introduced (Chapter 2), where research on the importance of teaching CS to school-aged children is described, along with research on existing coding platforms.
- Then, the **Problem Statement** (Chapter 3) presents a detailed description of the problem. Along with this, a comprehensive description of the solution can be found.
- Moreover, one can find the **Development** process followed to create the platform's Proof of Concept (Chapter 4).
- In (Chapter 5) are presented the **Validation** activities that were used in order to test the platform's efficiency and importance within the teaching and learning process.

- Finally, there is the **Conclusions** chapter (Chapter 6), which presents the full results of this work and how we plan to continue this research in the future.

Chapter 2

State of the Art

2.1	Methodology	7
2.2	Digital Literacy and Digital Skills	8
2.3	The Importance of Computer Science in Schools	8
2.4	Computer Science Curriculum in Other Countries	10
2.5	Live Programming	12
2.6	Best Development Practices: A Subset for K-12	14
2.7	Programming Environments	15
2.8	Summary	24

In this chapter, the State of the Art for this thesis' topic is presented. This research is fundamental to understand what has previously been done to implement Computer Science in other countries, as well as what platforms are already available for students of all ages to practise programming. This chapter also addresses the importance of learning CS and of several development practises. A summary of the methodology followed to perform all the research presented is also available.

2.1 Methodology

To gather information for this chapter, a research based on existing literature was performed. First, the databases where the information was researched are listed (section 2.1.1). Then, the process with which the research was performed is detailed (section 2.1.2).

2.1.1 Databases

The information was gathered from several sources, including ACM Digital Library, Elsevier, Scopus and IEEE Xplore. The information gathered from these databases ranges from books to articles and conference proceedings.

In addition, it is important to notice that some information collected for this chapter was not extracted from scientific literature but rather from websites. This is mostly the case in Section 2.7.1, where much of the information was gathered from the platforms' official pages.

2.1.2 Process

From the set of Research Questions (Section 1.4), queries were designed to be able to search the databases' information. These queries were improved over time as concepts and synonyms emerged from the investigation.

The first selection criterion for a given element was the abstract. From there, it was discarded or approved for further reading. All the materials that proceeded to the next stage were uploaded to the Mendeley tool for easier management. Following this step, a full read of each paper was performed, highlighting and annotating relevant information. Additionally, more results were drawn from each of these research results by analyzing their references in a snowball-like methodology.

2.2 Digital Literacy and Digital Skills

Throughout the research performed for this work, the concepts of **Digital Literacy** and **Digital Skills** were found on multiple occasions. However, they were rarely defined, and, even when they were, their definitions were inconsistent [SSLA18]. Because of this, here we present the definitions we chose to use throughout this work.

Digital Literacy: "*[T]he functional access, skills and practices necessary to become a confident, agile adopter of a range of technologies for personal, academic and professional use*" [SSLA18].

Digital Skills: "*Digital skills are defined as a range of abilities to use digital devices, communication applications, and networks to access and manage information.*" [UNE18].

2.3 The Importance of Computer Science in Schools

The importance of learning Computer Science from a young age has been highlighted for its multiple benefits to student's lives. In this section, those benefits are detailed and why they are important to the children's development and future.

We present why CS is essential to create a base of fundamental technology knowledge (section 2.3.1), how learning CS can teach Computational Thinking to students and improve their Problem-solving skills (section 2.3.2) and finally how learning CS in school has shown evidence of increasing one's chance to enroll in college (section 2.3.3).

2.3.1 Fundamental Technology Concepts

Technology is increasingly a fundamental part of everyone's lives. It is everywhere, from our phones and laptops to hospital information systems and electronic voting software.

As technology surrounds us, we believe it is more important than ever that people know not only how to use it but also how it works and how to create it. Like other fundamental areas of knowledge, such as mathematics, physics and linguistics, we believe Computer Science should be taught in schools. This does not mean that every child needs to follow this scientific area, it means that they can perform an informed choice about it. Even if they choose not to study this subject further, they have a fundamental basis of knowledge to apply throughout their lives [Lod20].

We believe this knowledge is transverse to all other areas, and, as such, its learning benefits every individual, as their technology usage can be more efficient and with more effective results [Lod20].

2.3.2 Computational Thinking and Problem Solving Capabilities

As stated in Section 1.1, Computer Science and Computational Thinking are not the same thing. However, they are intrinsically connected, as CT derives from studying CS [Lod20].

However, CT is an essential skill, not only for its application in this area but for all areas of knowledge, as it changes the way people think and solve their problems [Bun07]. As such, one cannot teach CS without focusing on CT, as this skill is a crucial part of these teachings [SC17].

Furthermore, teaching CS in schools has been demonstrated to improve children's cognitive development [MGB15] as well as their grades in multiple kinds of assessments, like Mathematics, Literacy, Sciences and English Language Arts [CFZ19].

Developing CT, although not limited to it, is also very often closely linked with programming [VFG⁺15, BCD⁺16a]. It allows for a more concrete display of CT knowledge, thus facilitating its acquisition and development [BCD⁺16b, BCD⁺16a]. Experiments show that, from an early age, children who are taught to program understand its core concepts, such as functions, recursion, loops, sequences and conditionals and could apply them while solving programming problems [MGB15]. It has also been demonstrated that after practicing coding, children's abilities to solve other programming problems improved, as well as their planning abilities [AVR20]. The use of these new technologies in classrooms while teaching CS is essential and it also further helps to develop CT [WDB⁺17].

So, one can understand that learning CS from a young age can positively impact, not only in technological areas, but in all areas, as it develops new ways to think and solve problems.

2.3.3 College Enrollments

As mentioned in the previous subsections, learning CS has shown evidence of positively impacting students' cognitive development and problem-solving skills. However, this alone does not prove that these students have a positive advantage in an academic environment over their peers who have not studied this subject.

To analyse this hypothesis, West Coast Analytics [BBP20] studied the different college acceptance rates between students who took a CS Advanced Placement Course (CSAPC) and those who did not.

Results from this study show that taking the CSAPC did have a significant impact on student's college enrollments, increasing their likelihood to enroll by 34%. This value increases up to 38% for minority groups [BBP20].

This study's conclusions show that Advanced Placement courses increase the students' chances to enroll in college but that the CSAPC had a more meaningful impact than any other.

In addition to this study's findings, learning CS is also important once one is already enrolled in college, as it creates a base of knowledge where colleges can build their curriculum on and also equips the student's with the fundamental technological knowledge for their academic life [HAB⁺11].

These findings demonstrate that learning CS impacts one's cognitive skills and actively prepares students for their academic life while also increasing their chances to enroll in a superior degree.

2.4 Computer Science Curriculum in Other Countries

Despite CS still not being part of the curriculum in most countries, a few countries have already implemented this subject in their schools. In this section, some of the countries that already have CS being taught in the schools will be presented, along with some of those curricula's characteristics and the organizations who helped to draft it, if applicable.

2.4.1 England

In September of 2013, England introduced a Computing course in its curriculum. This new subject was created to substitute an older version of a technological course, Information and Communication Technology (ICT), when the country reformed its entire curriculum for all subjects [BMD⁺15].

The new Computing curriculum was based on four important pillars [BMD⁺15]:

- Understand and apply principles and concepts of CS.
- Develop CT and apply it to solve problems.
- Develop the ability to use technology, even if unfamiliar, to solve problems.
- Develop the ability to responsibly and efficiently use ICT.

This improved curriculum enables students to go beyond the basic usage and learning of technology, it teaches them the tools and methods of CS, such as CT [WDB⁺17], which can help them in many other aspects of their lives.

The new curriculum was heavily influenced by the Computing at School (CAS) organization. In 2012, CAS designed a CS curriculum proposal, which was used as the base of the new curriculum once the subject was implemented [BMD⁺15].

CAS aims to provide teachers and educators with the resources and support needed in order for them to be motivated and fully equipped to teach CS at schools. One of their core principles is that CS is for everyone, so they strive to fight issues such as racism, homophobia and sexism. They also established a connection to The Chartered Institute for IT, a professional society in the subject of CS, which provided CAS with better resources and support in defending a quality teaching of CS [aS21].

2.4.2 United States of America

The U.S.A. does not have a national CS curriculum, partly because it is subject to each state's approval. However, multiple organizations have worked to create a curriculum that could be used in American schools [BMD⁺15].

The **Computer Science Teachers Association** (CSTA) is an organization of CS teachers from the U.S.A. and Canada whose goal is to enable CS teachers with better conditions to introduce and teach this subject in their schools [Ass16]. This organization collaborated with the Association for Computing Machinery (ACM) to create CS teaching standards for K-12. CSTA and ACM also collaborated with other organizations such as Code.org and National Math and Science Initiative to create frameworks to implement this curriculum [Lod20].

Code.org has contributed largely to the introduction of CS in schools around the world. Many American school districts have partnered with it, and many students are already enrolled in this platform.

2.4.3 New Zealand

New Zealand introduced its new CS curriculum in 2011 [WDB⁺17]. There had previously been a computing subject from 1974 to 1985, which was associated with applied maths [BAL10]. After this, Computing started being taught in a joint course for all technological areas, from food technology and meal planning to software development. This caused it to become less accessible to students who wished to study CS specifically [BAL10].

CS was later re-introduced in New Zealand's schools only for high school students. This was considered late in the student's school career and students showed difficulty learning the required skills. Teachers also showed their concerns, both because this subject was being introduced in the same years the students started having external evaluations [WDB⁺17] as well as the lack of guidance provided to them [BNDJ14]. In response to the former, a national committee was organized to decide what changes need to be made to the curriculum. Several resources were created to solve the latter, including CS guides and programming resources [BNDJ14].

2.4.4 Israel

For a long time, Israel had a curriculum featuring the Computing subject, but primarily either as an elective subject or as part of other subjects [BHC⁺12].

More recently, CS has been a subject on its own available in some high schools. However, these high schools are mostly elite institutions, and students who take this course have a heavier workload since they also are obligated to study advanced mathematics and physics [WDB⁺17].

Students that do not belong to these elite institutions are still offered technological courses, mostly digital literacy oriented, as Israel believes that every student should be educated on the basic usage of computers [WDB⁺17].

However, since CS is only offered in some high schools, this creates a severe disparity between the country's students, since some students will finish their school career much more technologically educated and prepared.

2.5 Live Programming

Liveness in programming (LP, also known as Interactive Programming or Exploratory Programming) concerns programs that can be edited while running and that its results will change in real-time [Tan13].

LP makes programming more straightforward and fluid by allowing the developer to edit and debug the work in real time, while running the code [McD13]. This new approach differs from the traditional development cycle (edit, compile, link, run), since it only has, theoretically, one development stage, in which the developer has the program running while modifying it and monitors the changes of the outcomes [Tan13, Mat18].

In 1990, Steven Tanimoto proposed the VIVA programming language, whose goal was to provide as much feedback as possible to the developers, which allowed them to understand the system's behavior better. This work was the first to define the term liveness, and Tanimoto's following works have deepened this concept [Ama18].

Liveness has six hierarchical levels that describe different types of liveness, as described by Tanimoto [Tan13], which can be seen in Figure 2.1. Tanimoto proposed this level hierarchy in 1990 [Tan90] and it originally had only the first four levels. This was later subjected to change in 2013, also by Tanimoto [Tan13], where levels five and six were added.

- **Level 1: Informative** is the level of the hierarchy that reflects a simple flowchart-like situation, where there is not an actual program written but rather a visual representation that aids the programmer [Tan13, CNB10, Tan90].
- **Level 2: Informative and Significant** is the level that reflects an executable program that can be run and whose results can be seen. This program is not changeable while it is executing [Tan13, CNB10, Tan90].
- **Level 3: Informative, Significant and Responsive** is the level that reflects a programming environment that automatically executes changes made by the programmer after a short period of time or a triggering event, like saving the work [Tan13, CNB10, Tan90].

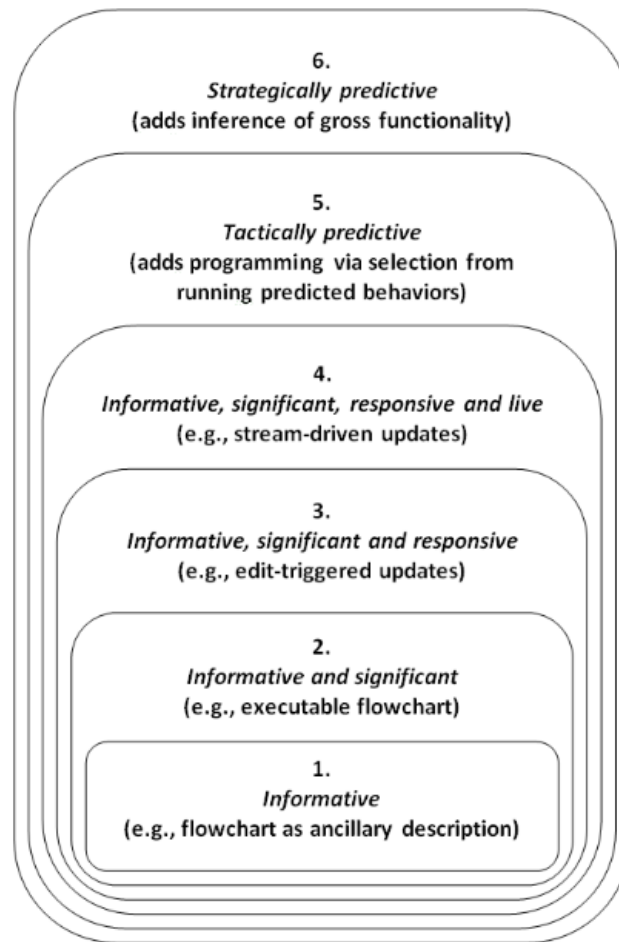


Figure 2.1: Tanimoto's Liveness Level Hierarchy [Tan13].

- **Level 4: Informative, Significant, Responsive and Live** is the level that reflects a programming environment that never stops the program's execution during the entire process and changes are visible to the programmer immediately [Tan13, CNB10, Tan90].
- **Level 5: Tactically Predictive** is the level that reflects a programming environment that, besides featuring all of Level 4's capabilities, predicts the programmer's next actions, presenting multiple different alternatives, thus staying ahead of the programmer in the development [Tan13].
- **Level 6: Strategically Predictive** is the level that reflects a programming environment that, still staying ahead of the programmer's development like Level 5, would not just predict the programmer's next actions but rather create a strategic prediction of the overall software's goal, creating a program based on this prediction and other available data [Tan13].

We believe this kind of real time editing process can be valuable in creating an engaging programming environment, like the one being created throughout this work.

2.6 Best Development Practices: A Subset for K-12

This section describes some of the considered Best Development Practices in software development. Although there are many more, the ones selected to be described in this section are the practices considered more relevant for the context of this work and directly connected to the topic being studied.

2.6.1 Collaborative Development

Collaborative Development (CD) is the process where two or more developers are working simultaneously on the same problem or code.

In the professional world, CD has improved development speed in certain projects [AMS13]. Furthermore, since development teams are becoming increasingly geographically scattered, multiple collaborative development environments have appeared, allowing remote working teams to work together in the same projects and discuss their ideas [BB03].

At schools, this practice has shown evidence of improving students' engagement in activities and increase their learning and discussion opportunities, both with their instructors and peers [IA15].

2.6.2 Version Control

Version Control (VC) is the process through which one can keep track of the multiple past versions of files [Tic85]. This process is usually performed by a Version Control System, a system that records the file's changes and allows backtracking to a certain point in time in the file's history. This history can be recorded locally on one's computer or remotely [Som13].

Having a file's complete edit history is extremely useful and allows to easily return to an older version if needed. However, VC is not used only with single-file projects. It can be used to keep track of the complete chronological history of a whole project. Some VC systems can also establish multiple parallel chronological lines, allowing multiple developers of the same project to work in it simultaneously while keeping their individual changes' history [Spi12].

These tools can help school-aged children and teenagers keep track of their progress and return to previous states of their projects, should they need to, providing a feeling of safety and freedom to explore.

2.6.3 Automated Testing

Testing is an important activity of software development. Testing allows to discover errors and to evaluate if the product meets the requirements. While tests can be performed manually [WA06], this costs developers time and is prone to mistakes.

This is where Automated Testing (AT) becomes relevant. AT consists of creating a layer in the development process where the tests are executed without needing any human intervention [DRP99]. Even though this kind of testing cannot fully substitute manual testing, since some

testers are specialized in finding software flaws [BWK05], it can speed up the development process by automatically executing a test suite deemed essential for the project to be considered successful.

AT can be particularly beneficial for schools teaching CS. Teachers can create a test suite translating the requirements that their students' projects need to fulfill without students needing to run manually the tests themselves.

2.7 Programming Environments

As previously mentioned, the benefits of learning CS from an early age are being studied and praised. To accompany this development, platforms need to exist where the students can exercise such activities.

For the next sections, we researched and present multiple development platforms with different purposes. We list platforms available for students (section 2.7.1), along with a brief description of each of them. Additionally, we present a selection of platforms that support Live Programming, both in Python and other programming languages (sections 2.7.2 and 2.7.4). We also describe a selection of platforms whose programming environment is customizable by the user (section 2.7.3).

2.7.1 Coding Platforms for Children and Teenagers

The platforms presented in this section were chosen using multiple criteria: the existence of the requirements we defined for our own work, such as Live Programming and Collaborative Development; its popularity within the sector, which can influence the reach and impact it has on the society; and if they have any scientific backing to their development.

One important note before starting detailing is that one of the comparison terms was the existence (or lack of it) of LP. In fact, only Scratch supports LP, so the absence of this parameter in the text implies this feature's non-existence in the given platform.

Another comparison feature was CD, but none of the platforms selected supported this feature.

In Table 2.1 there is a comparison of the platforms described below on multiple different parameters.

Scratch

The **Scratch** platform provides a visual programming environment that allows its users to learn programming while working on interesting projects such as animated stories and games [MRR⁺10].

Scratch was created by Lifelong Kindergarten Group at the MIT Media Lab [Scr07] and is entirely free.

This platform enables kids to program with a simple and intuitive block-based language. To create their programs, the students need only to drag and drop certain blocks, snapping them together like puzzle pieces, to create a sequence flow for their characters [MRB⁺04]. The result of the code that they create immediately appears on the side in the form of creatures of their choice.

The main goal when creating **Scratch** was to provide a platform where children could "*develop technological fluency, mathematical and problem-solving skills, and a justifiable self-confidence*" [MRB⁺04]. A screenshot of Scratch's web programming environment can be found in Figure 2.2.

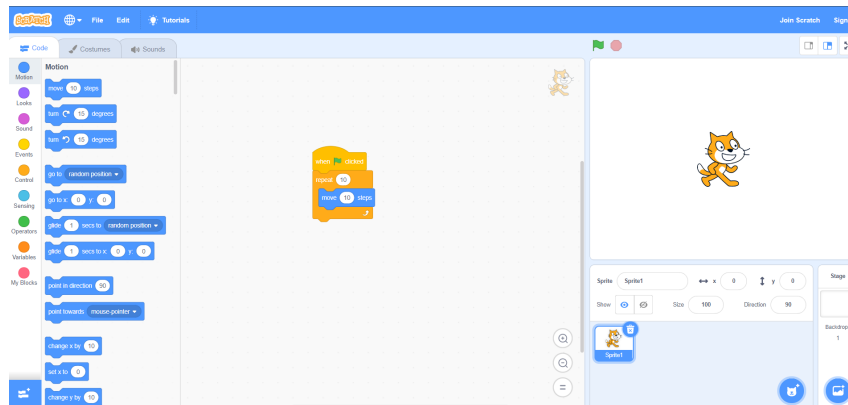


Figure 2.2: Scratch Programming Environment [Scr07].

Tynker

Tynker is a coding platform for children and teenagers from 5 to 17 years of age. This platform offers courses designed for each specific age and has partnerships with multiple companies, such as LEGO and Barbie, to encourage students to try the multiple challenges. The courses are designed to look like games, thus their advertising of the platform is "*The fun way to learn programming and develop problem-solving & critical thinking skills!*" [Tyn13].

This platform also allows teachers to create virtual classrooms with their students to track their progress on the different courses. In addition, **Tynker** also has integration with Clever Sync, Google Classroom and Microsoft Azure, which further helps educators to organize pupil's work.

The platform is paid, but it offers users the opportunity to try 20 coding games for free. A screenshot of Tynker's web programming environment can be seen in Figure 2.3.

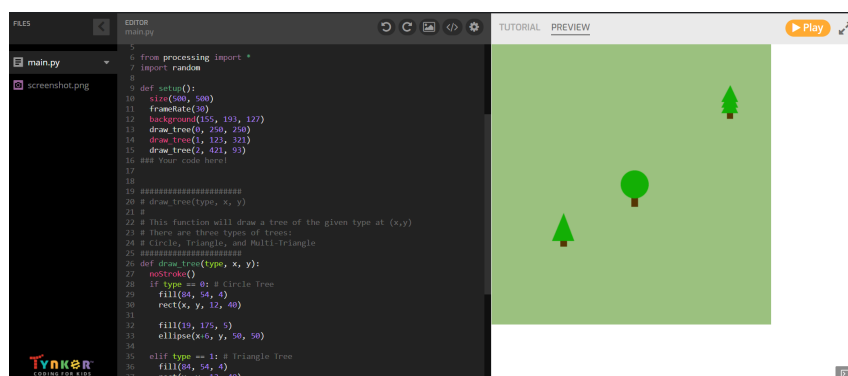


Figure 2.3: Tynker Programming Environment [Tyn13].

Code.org

Code.org is a worldwide non-profit initiative that is "*dedicated to expanding access to computer science in schools*" [Cod13a] while also striving to achieve more involvement by women and minority groups. This platform is entirely free.

One of their goals is to achieve a broader diversity in CS, and thus, they have already noted that 45% of their students are women and 50% are from marginalized racial and ethnic groups [Cod13a]. They also strive to create a more comprehensive CS curriculum for schools. So far, more than 180 American school districts have introduced this scientific area in their student's classes, totaling 30% of American students enrolled in Code.org to learn CS. Worldwide, they have over 50 million students, which accounts for 15% of all students, and 1 million teachers on Code.org [Cod13b].

This organization created the *Hour of Code* initiative, whose goals were to be a one-hour introduction to CS, encourage participation in the CS field, and show that anyone can learn how to code [oC13]. This initiative reached more than 15% of all worldwide students [Cod13a]. A screenshot of Code.org's web page can be seen in Figure 2.4.

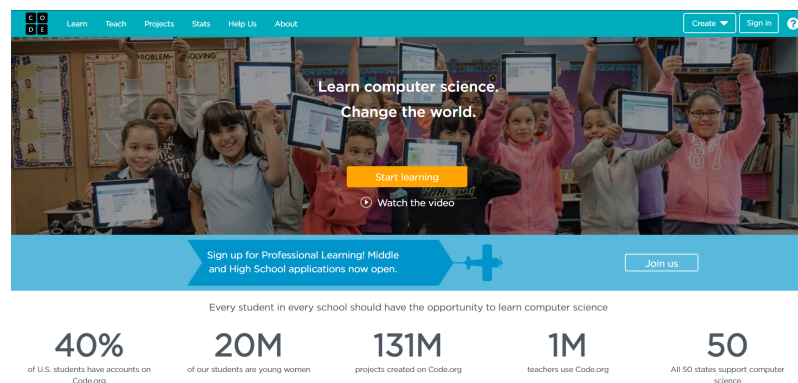


Figure 2.4: Code.org Web page [Cod13a].

Swift Playgrounds

Swift Playgrounds is a platform created by Apple for children to learn Swift, an open source language developed by Apple for creating applications [App18].

This platform has a series of puzzles to be completed with Swift. One of the most significant advantages of this platform is that "*It requires no coding knowledge, so it's perfect for students just starting out*" [App18], but the student is actually writing real code. It has a split-screen approach: on the left side of the screen, there is the code and on the right, there is the world to be manipulated, which allows the student to see the results of their work immediately. The difficulty of the puzzles is incremental, and while the children are effectively programming in Swift, they do not need to know the syntax, since in each puzzle there are hints on what should be used [mac16, wea18].

Besides the puzzles, children can also experiment freely with Swift. This app can be linked with other devices, such as drones and robots, and those same devices can be controlled via code produced on the platform [App18].

Swift Playgrounds only works on Apple devices but is entirely free to use [App18]. A screenshot of its programming environment can be found in Figure 2.5.

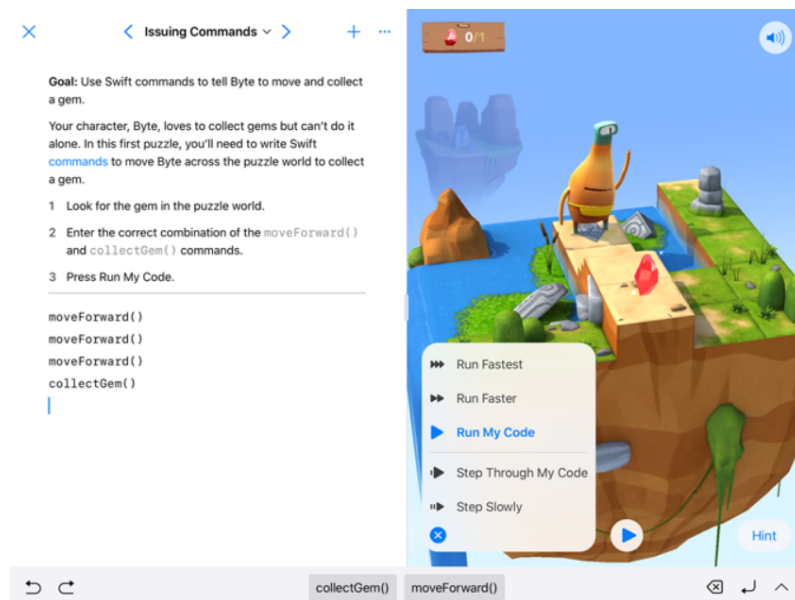


Figure 2.5: Swift Playgrounds Programming Environment [App18].

Alice

Alice is a platform created in 1995 at Carnegie Mellon University as a VR prototyping tool. It later evolved to the learning tool whose goal is to teach Object-Oriented (OO) with Java to beginners, using 3D worlds and characters with which students can create scenes and stories. This platform features a drag-and-drop approach, so it is easy for the students to create each element in the scene [Nea16, Ali15].

Multiple versions of Alice have since been released, like Alice 2, Storytelling Alice and Looking Glass. These versions each brought something new to the original platform, improving it and making it more appealing to multiple audiences. Storytelling Alice, for example, was created with the intention of being more appealing to school-aged girls by presenting itself like a tool to create a story rather than being just a programming environment [KP06].

Although this platform does not support Collaborative Development, there is a framework developed based on it that does, named "AliCe-ViLlagE", whose goal was to create an environment where students could work together on Alice projects [AJP15]. A screenshot of Alice's programming environment can be found in Figure 2.6.

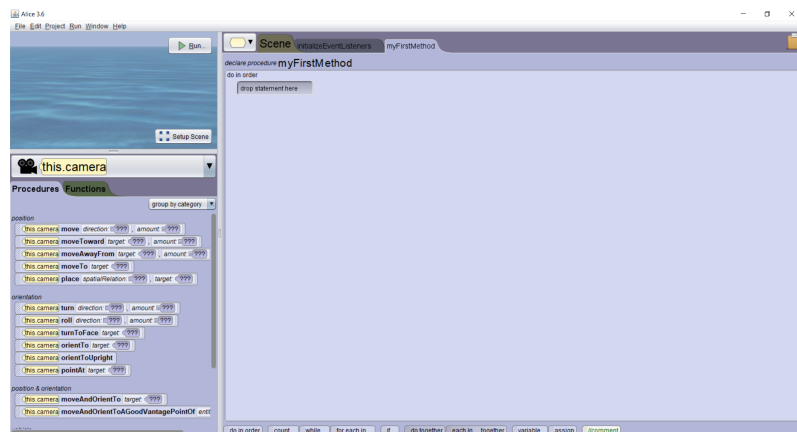


Figure 2.6: Alice Programming Environment [Ali15].

2.7.2 Programming Environments Supporting Liveness

This subsection features a selection of Programming Environments that support LP. It is important to notice that Scratch also features liveness, but was included in the "Coding Platforms for children and teenagers" section (2.7.1).

Sketchpad

Sketchpad is a system created by Ivan Sutherland in 1963 at MIT as part of his doctoral thesis [Kas14]. Sketchpad allowed for computer-aided drawings to be made, eliminating all text apart from captions and was the base framework for modern computer graphics [Kas14, Sut63].

Even though Sketchpad is not a programming environment, it was one of the first systems to feature liveness, as the user specifies the geometric shapes desired and they appeared on the screen immediately [Tan13].

Sketchpad had multiple improvements over the years, like Sketchpad III, created by Sutherland and other colleagues from MIT, that expanded the original Sketchpad 2D environment to 3D [Kas14].

Smalltalk-80 System

Smalltalk-80 is a graphical programming environment [GR83] created for users to program and interact with the Smalltalk coding language. The environment was designed to improve the language's visualization and ease its usage [GR83]. This system was released to the public in 1980 and allows the creation of text and graphics as well as their manipulation and program development within an OO approach [Kay96, Gol84].

In Smalltalk-80, the programmer first compiles the first version of its code, which is then executed. After this, new code can be written and tested without any need to stop the program's execution. Instead, this new code is compiled and linked to the original program and executed

Table 2.1: Platforms' features comparison.

	Live Programming	Collaborative Development	Python Development	Free
Scratch	Yes	No	No	Yes
Tynker	No	No	Yes	No*
Code.org	No	No	No	Yes
Swift Playgrounds	No	No	No	Yes
Alice	No	No	No	Yes

Note: The presence of the "*" symbol for the Tynker platform means that, although the platform is paid, there is a free trial that any student can access to.

[Gol84]. This features a Level 3 of liveness (see Section 2.5), as the user has to trigger the update, but the execution is not stopped.

Squeak

Squeak is a free and open-source programming environment that supports live programming with the OO programming language Smalltalk [Squ96a].

Squeak's virtual machine was built in Smalltalk, and it is platform-independent. It was initially developed for Macintosh but later ported to most platforms, like Windows and Linux distributions. This platform, which was created in 1996, reconditioned much of the original Smalltalk-80 programming environment, to which was added a Smalltalk to C translator and some other new features, like new support for color, musical synthesis and a portable file system [Squ96b, IKM⁺97].

JupyterLab

JupyterLab is a tool created by Project Jupyter. Project Jupyter is a non-profit and open-source organization whose goal is to provide a set of interactive tools for exploratory computing [Jup15a, PG15].

JupyterLab is a web-based interactive environment for Jupyter Notebook, a tool that allows users to interactively code, write equations and text, as well as complement these elements with other types of media [PG15].

This tool allows users to arrange the interface as they wish, supporting multiple types of windows and plugins, thus creating new user-customized workflows [Jup15b]. Furthermore, this tool features level 3 liveness (see Section 2.5), as one can see the results of what is being typed after a short waiting time, without the need to refresh or compile any of the work.

LiveCode

LiveCode is both a framework and a language that focuses on cutting development time and its associated costs. This is achieved by having a language similar to spoken English and by featuring drag-and-drop pre-coded tools that accelerate development [Liv21].

This language is highly portable, working in most currently used environments, such as macOS, Windows, iOS, Android and Linux [Liv21].

Even though this platform features its own language, functionalities created using this tool can be used with other common development languages, like C or Java, in a black-box development style, where the feature's internal functioning is unknown or hidden [Liv21].

LiveCode features level 4 liveness (see Section 2.5), as the program execution never stops and changes are displayed immediately [Liv21].

2.7.3 Customizable Programming Environments

This section presents customizable programming environments available in the market. By customizable, we mean extensible in features, via plugins and extensions, and customizable in appearance, such as the themes, workflows and available menus.

It is important to notice that JupyterLab is also customizable and extensible but was included in the "Programming Environments supporting Liveness" section (2.7.2) instead.

Eclipse

Eclipse is a free, open-source Integrated Development Environment (IDE) designed by Eclipse Foundation, a community that works towards open-source software and innovation [Fou04b, Fou04a].

This IDE, which runs on Windows, macOS and Linux, has a marketplace available, featuring various plugins and extensions to extend the original platform, allowing the support of multiple programming languages, workflows and new functionalities [Fou04a].

Eclipse features plugins for Live Programming in Python, Version Control, Automated Testing and Collaborative Development, making it a good candidate for developing the platform for this work. A screenshot of Eclipse IDE can be found in Figure 2.7.

Visual Studio Code

Visual Studio Code (also known as VS Code) is a free, open-source code editor created by Microsoft. This platform is lightweight but highly customizable, having multiple plugins and extensions to adapt it to support different programming languages [Cod15].

This tool runs on most of the mainstream operating systems, such as Windows, macOS and Linux. Furthermore, VS Code also has built-in version control (by supporting Git commands), debugging mechanisms and syntax highlighting. The user can also customize the appearance of the interface. VS Code features support for Python, Collaborative Development, Automated Testing (and Version Control as stated above), but no feature for Live Programming was found.

A screenshot of VS Code's interface can be found in Figure 2.8.

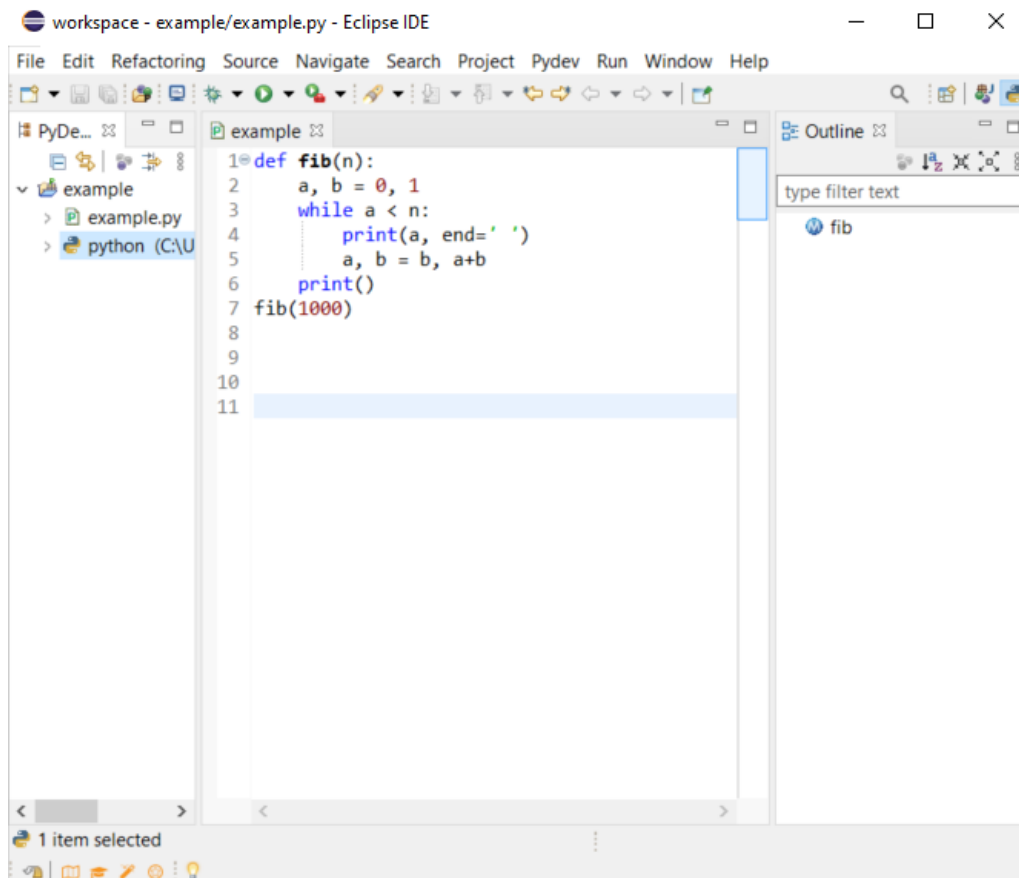


Figure 2.7: Eclipse Integrated Development Environment [Fou04a].

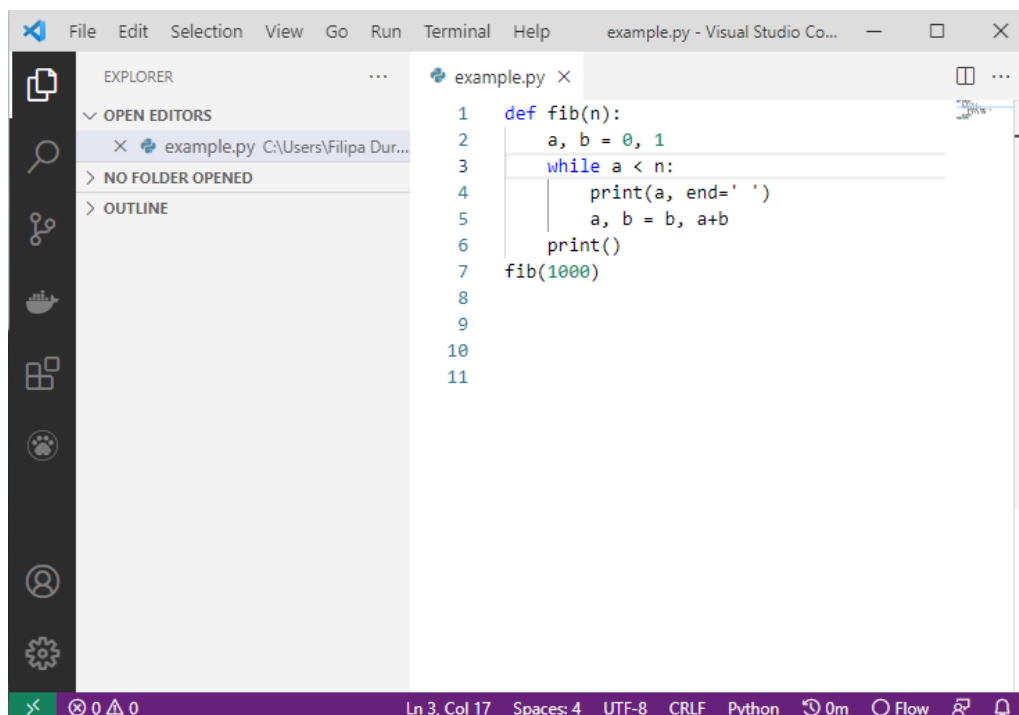


Figure 2.8: Visual Studio Code Editor [Cod15].

2.7.4 Python Live Programming Environments

This section presents environments that were explicitly designed to allow one to program live in Python.

Online Python Tutor

Online Python Tutor is an open-source web application [hci21b] designed to allow one to program live in Python. It has a simple interface, with only a textbox where one writes the code and a blank space where the values of the variables and program results appear.

This tool only runs on a browser, and, besides LP, it also allows the backtracking and the forwarding of steps of the program, so one can understand the entire functioning of what was written [hci21a].

There is, however, no customization available for this tool. One can integrate it into personal web projects, as the code is open-source, but there are no customization options for the base platform. A screenshot of Online Python Tutor's interface can be found in Figure 2.9.

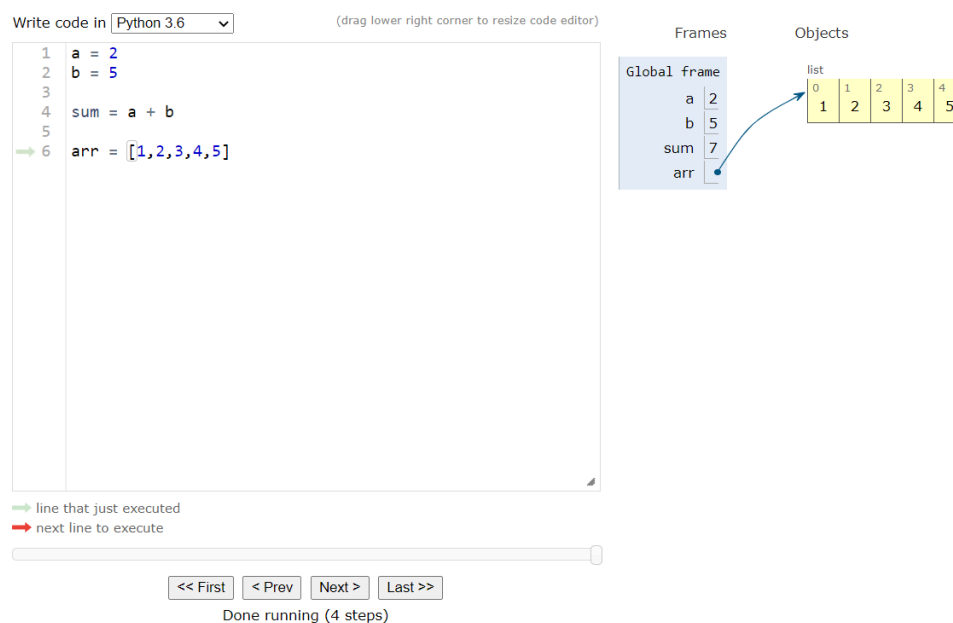


Figure 2.9: Online Python Tutor [hci21a].

PyLive Coding

PyLive Coding is a Python library that automatically reloads the code results as they are being written, allowing for LP [kil21].

For this library to be used, one has to import it into the code and initialize several variables according to the official documentation. This allows LP, but that needs to inherently be present in

the script itself. This can be a disadvantage if the LP feature is being used only for development because afterwards, the code needs complete refactoring to remove this logic from it.

2.8 Summary

Learning CS from a young age has multiple benefits, such as improving one's problem-solving skills, enhance one's grades in other subjects, and even boosting the student's chances to enroll in college (section 2.3).

There are multiple countries that have already implemented Computer Science into their curricula, or are working towards that goal. Each country has their unique approach to the subject, implementing it in different ways (section 2.4).

Live Programming has also been proven to have benefits on the learning of programming, by creating a more engaging environment (section 2.5).

We also researched what programming platforms were available featuring the requirements we designed. We found multiple platforms for students; Live Programming-oriented platforms, with some of them being exclusively for Python programming; and customizable programming platforms, where one can customize the working environment to suit their needs (section 2.7).

Chapter 3

Problem

3.1	Research Problem	25
3.2	Scope	26
3.3	Hypothesis	26
3.4	Envisioned Solution	26
3.5	Research Strategy	27
3.6	Validation	28
3.7	Summary	28

A good definition of a problem is a strong first step in the resolution of said problem. Having all the facets of the problem well structured and defined helps one's structuring of thought and planning.

In this chapter, we fully defined the problem being studied in this work, as well as its scope (sections 3.1 and 3.2). The hypothesis we aim to support is also described (section 3.3). We also present the envisioned solution to solve the problem, as well as the Research Strategy followed to solve it (sections 3.4 and 3.5). Finally, we present an overview of the validation processes used to validate our solution (section 3.6).

3.1 Research Problem

Currently, CS is not part of most of the schools' curriculum. As it was described in Section 2.4, there has been an effort in changing this situation, and multiple countries have been taking steps in order to implement this subject in their schools.

In Portugal, ENSICO [Ens21] is a non-profit organization devoted to creating the CS curriculum and pursuing its implementation. This organization collaborates with several experts to create an adequate curriculum for all K-12 students and plan the best way to implement it.

In addition, we believe that, in order for a CS curriculum to be efficiently implemented, and programming being a part of that curriculum, a proper platform for the students to practice their

work will be crucial one day. Several platforms of this kind exist, as described in Section 2.7.1, but none of these fit all requirements we believe to be ideal. So this work's goal is to explore how to fill this gap and work towards a solution, providing a Proof of Concept.

3.2 Scope

This thesis focuses on understanding the needed requirements for a platform where teenagers from 10 to 14 years of age can program with liveness and collaboratively, following some of the best practices of software development, in an engaged and playful way, and implement the first version of said platform. This Proof of Concept should have the main features available and be extensible in the future.

Although this platform is designed to help implement CS in schools, the curricula or its class-room implementation is not within this thesis's scope.

3.3 Hypothesis

The hypothesis this work aims to support is that children who study Computer Science at school, and thus learn programming as part of that subject, will learn how to program more easily if they have available a suitable platform, designed specifically for them and featuring Live Programming, Version Control, Collaborative Development and Automated Testing, to apply and practice their skills in an exploratory, interactive, safe and playful way.

This work also aims to understand if children will learn better if they feel that the environment is interactive and fun by diminishing their distractions while working.

3.4 Envisioned Solution

The envisioned solution for the stated problem is a platform that students can use inside and outside the classroom to learn and practice programming.

This platform contains important features of a classic development environment but adapted to the target age. One of the main goals was to make it intuitive to use, fun and engaging to young students in order to keep them motivated and focused [CZW14].

The requirements of the platform, which were defined to provide easier learning of programming by CS students, are the following:

- Flexibility. By not imposing a specific set of lessons or curriculum, teachers can use their own or those created by others, so the platform fits more use cases and can even be used in other countries with different curricula.
- Expansibility, so new practises and tools can be integrated in the future, improving it.

- Interactivity and exploratory capability, by having this development environment support LP. LP creates an interactive development environment, so students can better visualize their projects' evolution and efficiently fix their mistakes [Lod20, Tan90].
- Being free, which allows for all students to use it without any cost-related concerns.

The language supported by this platform is Python, an easy-to-learn programming language that has useful features while still having an easily readable code [Bri13].

The development process to create this platform is detailed in (Chapter 4), describing how the platform was developed and how its distribution was prepared.

Finally, the name for this platform is **Loki**. This name was built from the words "Live prOgramming for KIDs". Furthermore, Loki is the name of the Norse god of Mischief, which can be defined as a playful or trickster behaviour, usually by part of a child, that is not serious. As this tool is targeted at children and teens, we believe this name fits the demographics. Furthermore, Loki is now also the name of a well-known pop-culture character, which can be a first incentive for the students to use the platform.

3.5 Research Strategy

After completing the review of the State of the Art and defining the Problem, Loki's requirements and mockups were drafted. Following this step, the development of the platform began. This was performed iteratively, alternating building Loki and comparing it to the requirements and testing it to assure its quality, following a Design Science Research approach [DLA15]. A diagram detailing this process can be found in Figure 3.1.

When we believed Loki to be ready enough, we proceeded to the external validation phase. As described (Chapter 5), the whole validation process with students within the target age group was prepared, but we were later informed that it was deemed impossible, so, we used only expert reviews and an experiment with college students as volunteers.

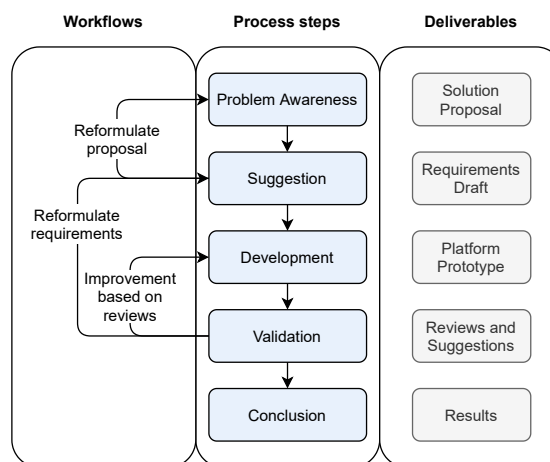


Figure 3.1: Design Science Research Process Diagram (Adapted from [HGTK10]).

3.6 Validation

Loki was validated in three different ways:

- Short experiments, performed by ourselves, to understand if the requirements drafted were met. In this first step, we aimed to understand if all the requirements were being met correctly. This was performed all throughout the development process, using simple Python scripts.
- Thorough empirical evaluation by experts in Computer Science education, who registered their opinions and suggestions for improvements on the work. In this second way, we aimed to understand if there are any underlying problems with the platform, such as usability issues. These insights were valuable to understand if Loki would be intuitive to use by the target audience.
- Controlled classroom experiments with Informatics students from FEUP. This process was created preventively to substitute the school experiments, should they not be possible. In this last step, we wished to gather data on how students used this platform. As it was impossible to use target-age students in this experiment, we asked for Informatics students volunteers to solve some exercises using the platform. This experiment gave us essential feedback if the platform was indeed effective, intuitive and if it improved the programming experience.

We believed these steps were sufficient to provide a comprehensive validation of the project.

Two more experiments were prepared, which would be performed with 10 to 14 year old students studying Computer Science. However, both of these experiments were eventually cancelled due to the COVID-19 pandemic.

3.7 Summary

In this chapter, we presented the problem being studied in this work, the lack of Computer Science subjects in most schools' curriculum (section 3.1). With this work, we aim to support the hypothesis that children will learn to program (which is an important facet of CS) more easily if they have an engaging platform where to practise their work. The solution envisioned to help solve this problem is Loki, a platform in which students can practise their programming (section 3.4).

The development of this work was performed following a Design Science Research approach, with Loki being developed and validated iteratively (section 3.5).

Loki was validated by ourselves along the process of its development, and when it was finished, it was then validated in an experiment with college students and reviewed by experts in teaching CS (section 3.6).

Chapter 4

The Loki Environment

4.1	Architecture	29
4.2	Features	30
4.3	User Interface Development	34
4.4	Eclipse Customized Distribution Sharing	36
4.5	System Requirements And Running Instructions	38
4.6	Using Loki: An Example	39
4.7	Summary	42

Loki was developed using the Eclipse IDE as the base of all development. As there is no native Eclipse for Python available, we used Eclipse for Java. Eclipse features several customization options that can later be shared among users, which made it an ideal candidate for development. In order to add the needed features to Eclipse, plugins were researched, selected and installed (section 4.2). As Loki is to be used by children, one of the main goals was to make it as straightforward as possible, so the interface was simplified as much as possible to make it intuitive and straightforward (section 4.3). We also performed research on how to share the final platform created and the essential requirements to execute it (sections 4.4 and 4.5). Finally, we present a small demonstration of Loki’s functionalities and how they can be used (section 4.6).

4.1 Architecture

For Eclipse to be adapted to fit the requirements defined, several features had to be added to it. In order to add these features, plugins were tested and installed. A component diagram detailing Loki’s architecture is presented in Figure 4.1. Thus, Loki was built by layers, being the base layer the Eclipse IDE, receiving on top of that the plugins needed.

For Loki to be completed, the UI of Eclipse was also adapted and simplified.

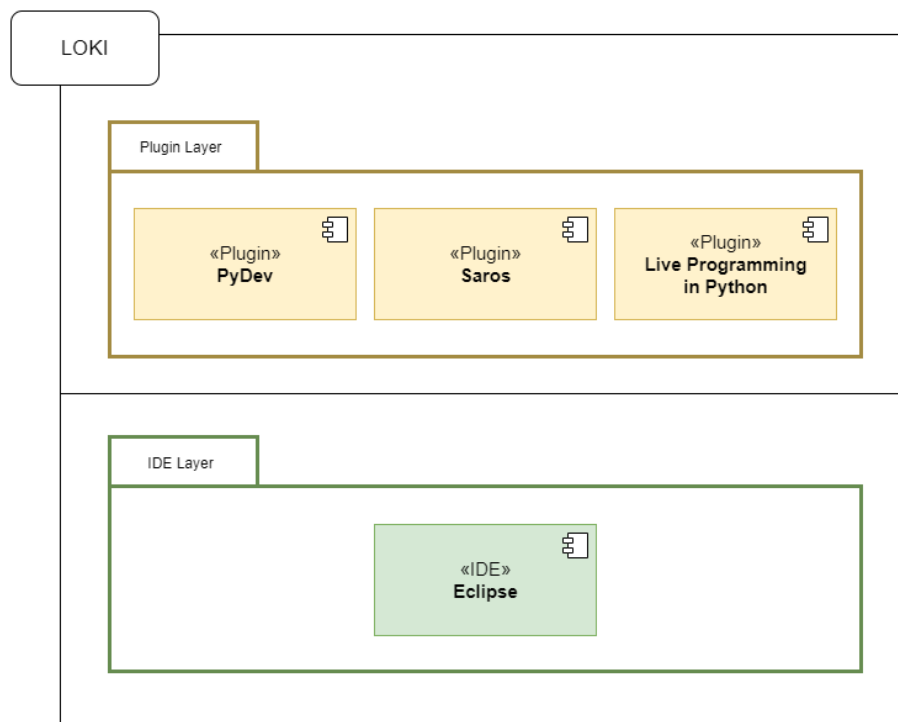


Figure 4.1: Component Diagram Detailing Loki's Architecture.

4.2 Features

For each feature we wished to add, several searches with keywords were performed in the Eclipse Marketplace, where all the plugins available for Eclipse are listed. Then, for the first page of results of each research, the official plugin page of each plugin was opened. From the description on the page, the plugin advanced to being tested on Eclipse or discarded. The plugins described in this chapter belong to the last step, having all been installed and tested on Eclipse, and later selected or discarded based on their performance.

4.2.1 Python Support

PyDev - Python IDE for Eclipse 8.2.0

PyDev is an open-source plugin that enables Eclipse to be used as a Python IDE. This plugin adds the needed functionalities to develop and interpret Python code, run and debug it. It also allows performing unitary tests with PyUnit (adding the AT feature as well).

This plugin, however, does not feature a Python installation, only the support for integration with Python. The user is responsible for having a functioning Python version available in its machine so PyDev can adequately configure the Python interpreter.

4.2.2 Live Programming

Live Programming in Python 2.25.0

Live Programming in Python is an Eclipse plugin that allows programming live (level 4 liveness) with Python. This plugin can only be used when Python support is already assured by some other plugin (such as the PyDev plugin) as it does not enable Eclipse to program in Python by itself, it only enables the liveness feature.

This plugin shows in real-time the value of every variable declared and how its value changes throughout the execution of the script. It also supports LP with the Turtle library [Lib21], allowing the live editing and instant feedback of graphical outputs. The live feedback, in both cases, appears in a panel beside the code. Screenshots of this plugin's usage can be found in Figures 4.2 and 4.3.

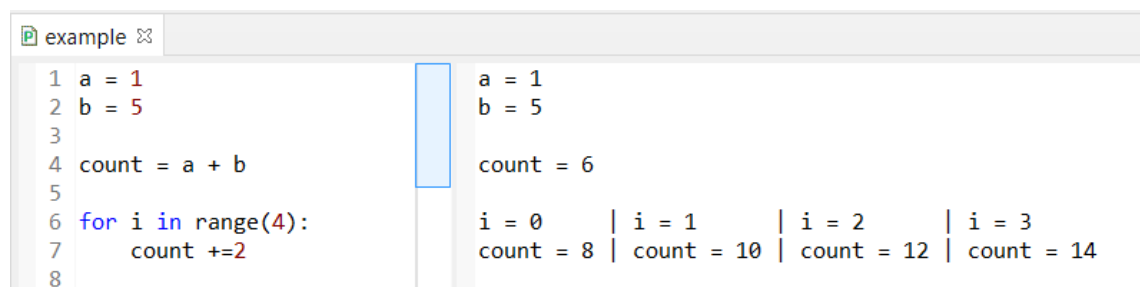


Figure 4.2: Eclipse Liveness panel created by the Python LP plugin [Fou04a].

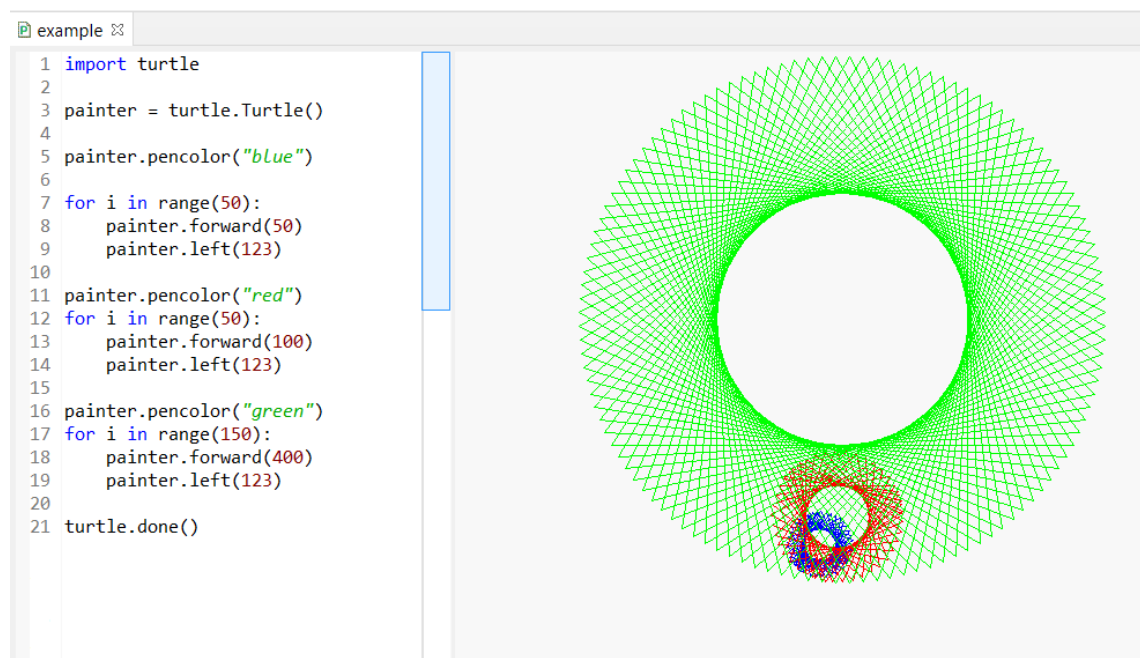


Figure 4.3: Eclipse Liveness panel created by the Python LP plugin while programming with the Turtle graphics library [Fou04a].

4.2.3 Collaborative Development

Saros - Distributed Collaborative Editing and Pair Programming 15.0.0

Saros is an Eclipse plugin that allows two or more users to collaborate and work at the same time in the same file remotely. To do so, one only needs to create an account in the Saros software company, Jabber, and log into it in the Eclipse plugin.

While coding collaboratively, each user is assigned a color, and that helps to clearly see where other users are editing since the cursors take that same color. Moreover, blocks of code created or edited by a specific user will be highlighted with that user's color.

Furthermore, the file being edited by all users is the file in the host user of the collaborative session. However, all users can choose to save a copy of the file on their own computers.

One can also save a contact list of recurrent collaborators to easily reach them when needed. A user is notified when another user is attempting to reach them.

Finally, it is important to notice that this plugin is compatible with the LP plugin, so, while in a CD session, if both users are using this platform, both users can have a LP session running simultaneously with the CD session.

Screenshots featuring a session with two users collaborating in one script can be found in Figure 4.4 and Figure 4.5. Each user sees highlighted code where the other user has collaborated.

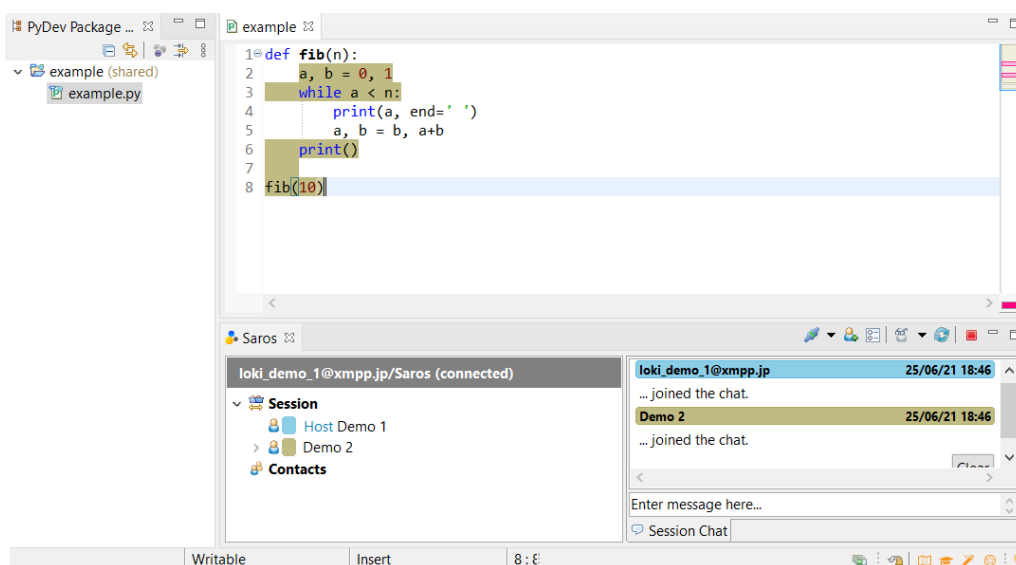


Figure 4.4: Saros Collaborative Session With Two Users - User 1 View.

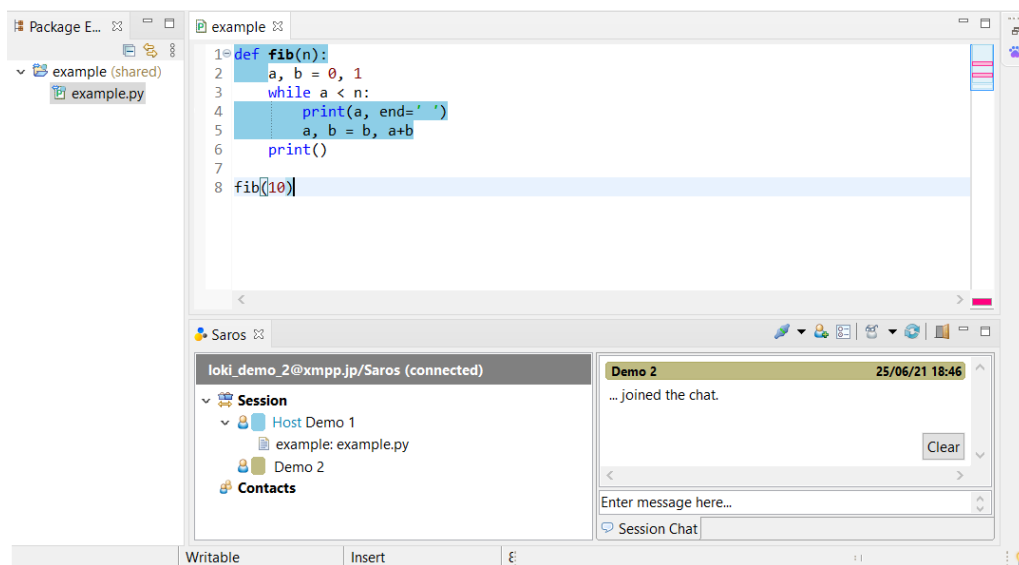


Figure 4.5: Saros Collaborative Session With Two Users - User 2 View.

Code Together 4.0.1a

Code Together is a plugin that enables remote CD in Eclipse. This plugin allowed two people to edit the same document remotely, with real-time changes appearing on both sides. However, this plugin presented two severe issues:

- It did not allow for both users to edit at the same time. Both users shared a cursor, so only one could be editing at a certain point in time.
- This plugin is not fully compatible with the LP plugin, so, while having both plugins installed was not an issue, one could not use the LP feature while in a CD session.

For these aforementioned reasons, this plugin was discarded, and the Saros plugin was chosen instead.

4.2.4 Others

While the plugins previously described were the ones that added the required features to the platform in order to fit all the requirements, those alone did not allow to collect data about the coding sessions taking place in Eclipse. As such, and in an attempt to gather as much information possible from each experiment taking place with students, we also used **Code Time** for its data collection and analysis capabilities.

Code Time is a platform for tracking development information. It has plugins available for several mainstream IDEs, such as Eclipse and VS Code.

After installing, signing up and logging into the plugin, it monitors the user's activity, such as the number of lines of code added and removed, percentage of active coding time and keystrokes. A screenshot of this dashboard can be found in Figure 4.6.

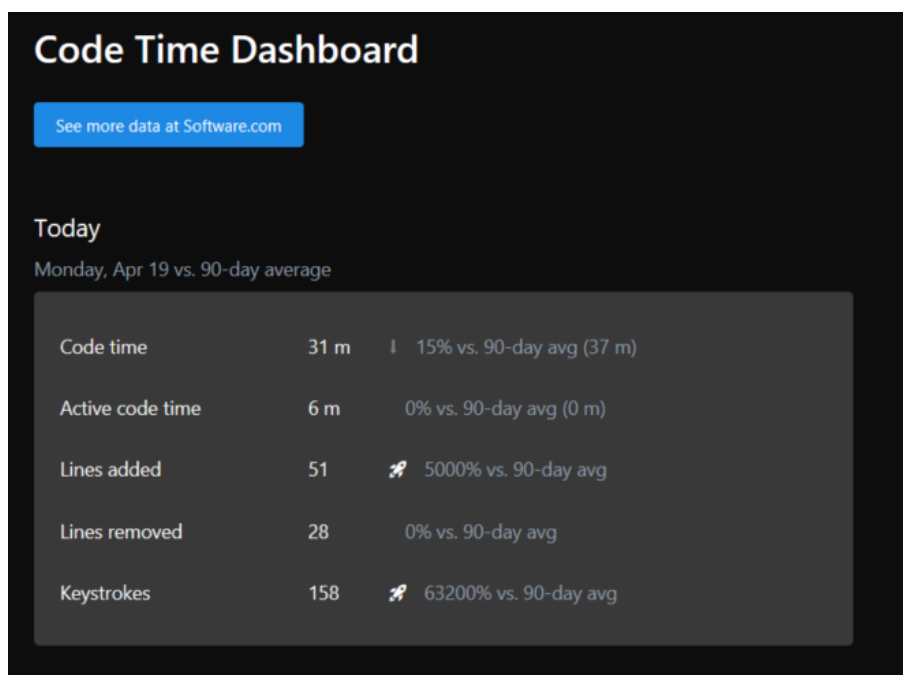


Figure 4.6: Code Time Dashboard [Sof21].

In addition, more information can be seen in an online dashboard, summarizing the current week's activity and comparing it to previous work weeks. A screenshot of this online dashboard can be found in Figure 4.7.

This plugin is not part of the final platform, but it was used for the classroom experiments to gather the maximum amount of data possible from each experiment.

Furthermore, the plugin to integrate the usage of the Yatta Profiles tool with Eclipse was also tested, but it was discarded too. The reasons for this decision are detailed further ahead in Section 4.4.2.

4.3 User Interface Development

As previously mentioned at the beginning of this chapter, the base development environment was Eclipse for Java, to which were added the plugins described in Section 4.2. However, Eclipse is a powerful but complex IDE, with multiple menus, tools and panels in its interface. Moreover, for each installed plugin, several menus and tool icons were added. As this platform is to be used by students, it is essential that the environment is as simple as possible to make its usage more intuitive and easy.

To that end, the "Perspectives" functionality was used, where a user can customize the menus and tools available in the User Interface (UI) and save those preferences. With this functionality, the number of menus available (after installing all the needed extensions) was reduced from 12 to 4, and the number of tool icons was reduced from 23 to 6 (Figures 4.8 and 4.9).

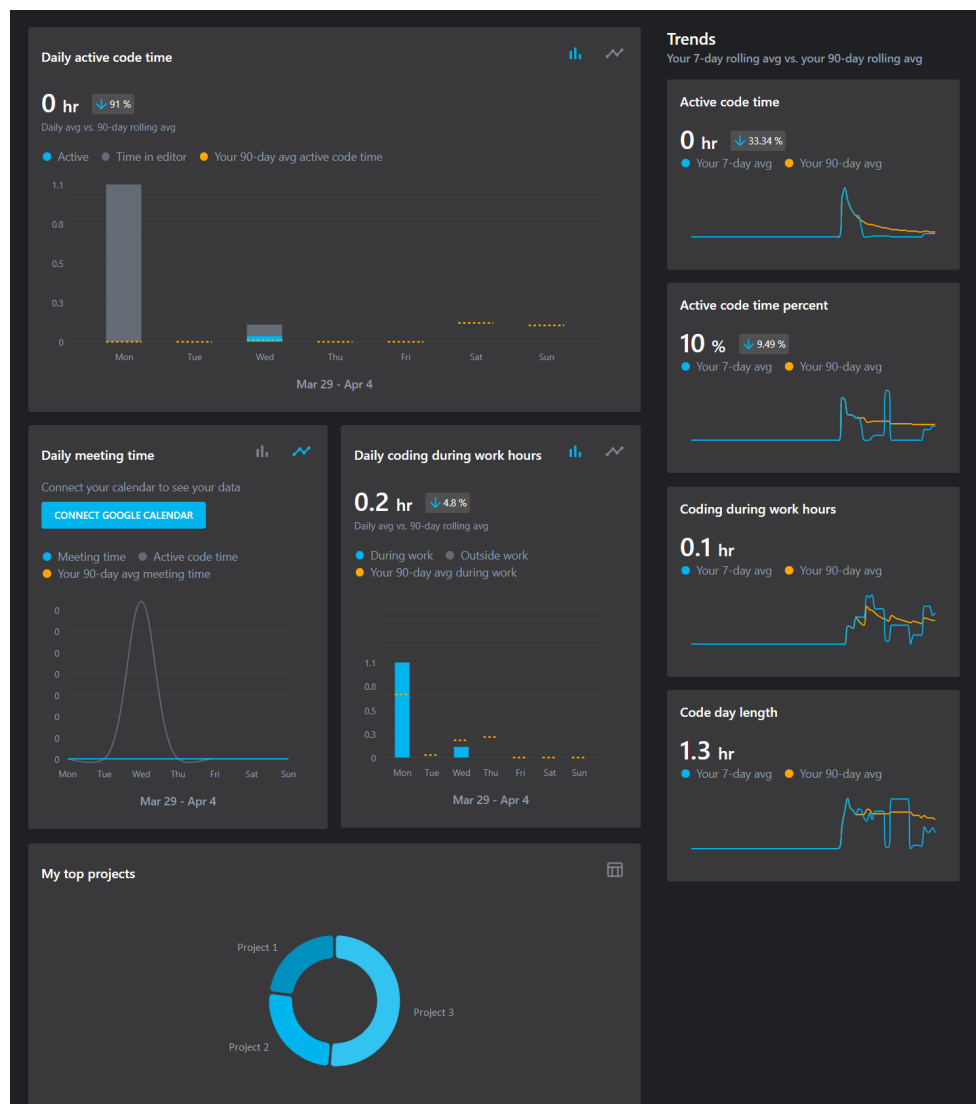


Figure 4.7: Code Time Online Dashboard [Sof21].



Figure 4.8: Original Eclipse Menus and Tools After Installing All the Needed Plugins [Fou04a].

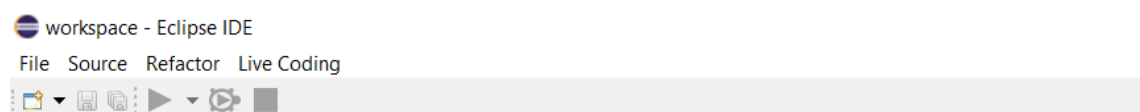


Figure 4.9: Eclipse Menus and Tools After Installing All the Needed Plugins and Creating Customized Perspective [Fou04a].

4.4 Eclipse Customized Distribution Sharing

Eclipse customizations can be saved and used in further sessions and shared among devices as a customized Eclipse distribution.

4.4.1 Oomph

Oomph is a tool that allows to create and share Eclipse installations. It allows to create and share packages that contain Eclipse configurations for specific projects, such as the plugins installed and repositories for the work to be pushed to.

An attempt was made to use this specific tool to create the customized Eclipse distribution. After installing Oomph, the steps to create an Eclipse distribution were followed, but multiple issues emerged:

- The tool's usage is complex. Given that children would need to use this tool to install the distribution in their own computers, this was a serious problem.
- In no step of the profile-creation process that we have completed, one was asked to edit the UI of the IDE. Clearly, this does not present a problem in a professional environment, where the setup needed relies mostly on the plugins installed and not in the edition of the UI menus and tools. This was, however, an important aspect to this project.

Given the problems mentioned above, this tool's usage was discarded, and other options were researched.

4.4.2 Yatta Profiles

Yatta Profiles is a tool that allows to save and share a given Eclipse customization among working environments, allowing the creation of distributions to be used by entire teams. This tool is based on Oomph (Section 4.4.1). However, unlike Oomph, the usage of Yatta was straightforward, but it did present some problems as well.

The Yatta launcher is simple and intuitive. All one has to do is download an Eclipse distribution from the Yatta online cloud. To launch a profile is as simple as clicking a button. Creating the profiles, however, demanded a plugin to be installed into Eclipse when developing the profile, and the plugin conflicted with the launcher. To solve this, the launcher had to be uninstalled every time a new profile had to be created. Furthermore, the plugin had multiple bugs that forced us to restart Eclipse several times during this process.

After creating a profile consisting of the plugins needed and the UI configuration, that profile was uploaded to the Yatta cloud. It was then downloaded to several machines to test it. On the machine it was created, the profile worked as it was expected, with Eclipse opening up with the correct plugins and UI configuration. However, the same did not happen in other machines, where only the plugins were present, while the UI setup was the original Eclipse UI layout, not the edited layout. We believe this difference between the original machine and other machines can be

explained by cached information on the original machine that saved the UI information and that that information was never really stored in the profile created by Yatta.

Given that the UI configuration is an essential part of the platform developed, Yatta's usage was discarded and other options were researched. An image of Yatta Profile's interface can be found in Figure 4.10.

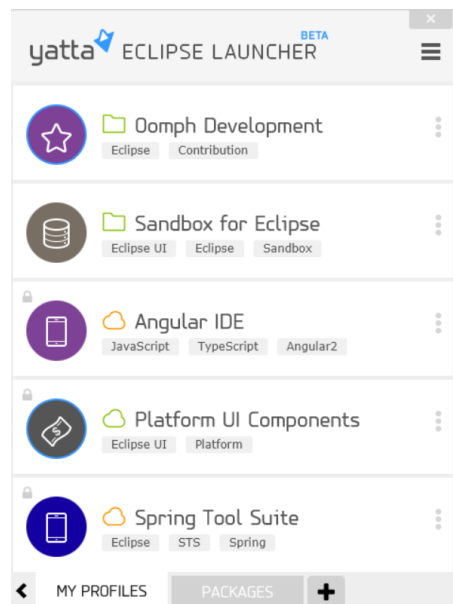


Figure 4.10: Yatta Profiles Platform [Yat21].

4.4.3 Distribution Package

The final method tested for the saving and sharing of the Eclipse distribution, which is the one that was chosen, consisted of a compressed folder containing all the Eclipse configuration files and executables.

Eclipse distributions can be downloaded from the Eclipse Foundation website in the form of a zip folder containing all the needed information to run Eclipse and the respective executable file. When running Eclipse, a prompt asks to choose a folder in which to save the workspace information. To try this method, a folder named "workspace" was created, and Eclipse was executed using that folder as the directory in which to save the workspace information. The structure of directories is shown in Figure 4.11 (for the Windows version, where a Python folder is also available in the directory, which will be further detailed in Section 4.5).

After finishing installing all plugins and editing the UI, the entire directory containing the original Eclipse files downloaded from the Eclipse foundation, plus the "workspace" folder, was compressed. This newly compressed folder was then decompressed in another machine and Eclipse was ran choosing "workspace" as the workspace information source. Eclipse started and the UI state and the plugins were as desired, equal to the originally created distribution.

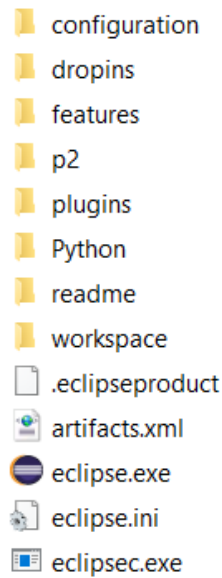


Figure 4.11: Eclipse Folder Structure in Windows.

Furthermore, this is the simplest of the methods studied, as the process only consists of decompressing a folder and choosing the correct directory when prompted with the choice. As such, this was the method chosen to proceed with this work.

4.5 System Requirements And Running Instructions

Three Eclipse distributions (Loki versions) were created, being compatible with the Windows, Linux and macOS operating systems, the most widely used (according to [Liu21], by January 2021, Windows covered about 71% of the OS' market, MacOS 16% and Linux almost 2%).

As the base Eclipse environment is Eclipse for Java, in order for any of the functionalities to work, it needs a valid Java Runtime Environment (JRE) available and its directory defined in the Eclipse. A JRE is available in the Eclipse compressed folder (in the "plugins" directory that can be seen in Figure 4.11).

For the Python interpretation to work on Eclipse, a Python interpreter must be available in the machine that is running the platform.

This can be done in three different ways.

- In the Windows version, a "Python" directory is available in the compressed folder (Figure 4.11). When prompted with the choice, one can choose the "python.exe" (see Figure 4.12) executable file that can be found in that directory.

- For macOS and Linux, one needs to previously install Python in the machine. Then, when prompted with the choice, one must use the "Choose from list" (Figure 4.12) option that finds all Python installations available.
- Having Python installed in the machine can also be a solution in Windows and, in this case, it is needed to add it to the PATH system variable. In order for this process to be automated, a script was created to perform all the tasks related to the Python installation and setup. This script can be found in Appendix A.1. Then, one chooses the "Config first in PATH" option (see Figure 4.12) and Eclipse manages the rest of the process of finding and setting up the Python interpreter.

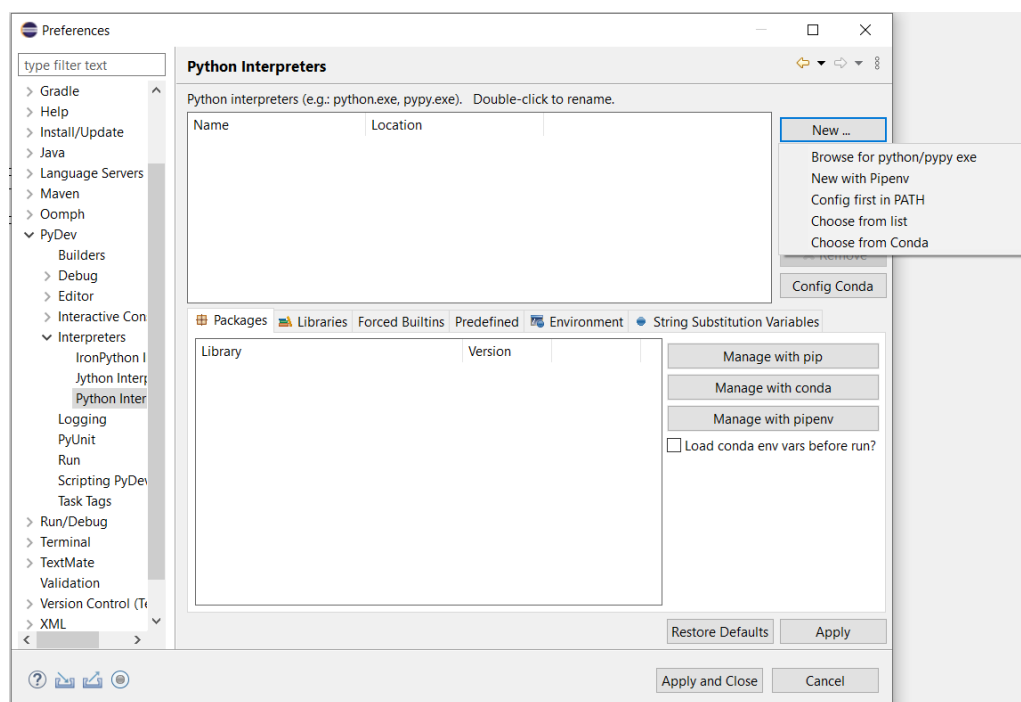


Figure 4.12: Python Interpreter Choice Menu.

Loki can be run from either an internal computer disk or from an external USB drive, following the same steps.

Complete and detailed instructions for installing, running and using Loki are available in Section A.2.

4.6 Using Loki: An Example

In this section, we wish to demonstrate the key functionalities of Loki.

The environment was designed to be as simple as possible, featuring only the essential menus and tools. The projects created appear on the left panel. The Live Programming buttons are

available on the top icon bar. On the bottom panel, there is the Collaborative Development plugin, Saros. There is also a tool icon for Automated testing on the right vertical bar (Figure 4.13).



Figure 4.13: Initial Loki Screen, Featuring the Projects Panel, The Live Programming Action Buttons, The Saros Panel and the Automated Testing Icon.

On any Python program, one can start the LP feature at any time. This will run the program as it's being written on the left panel, allowing one to track the changes in real time on the right panel (Figure 4.14).

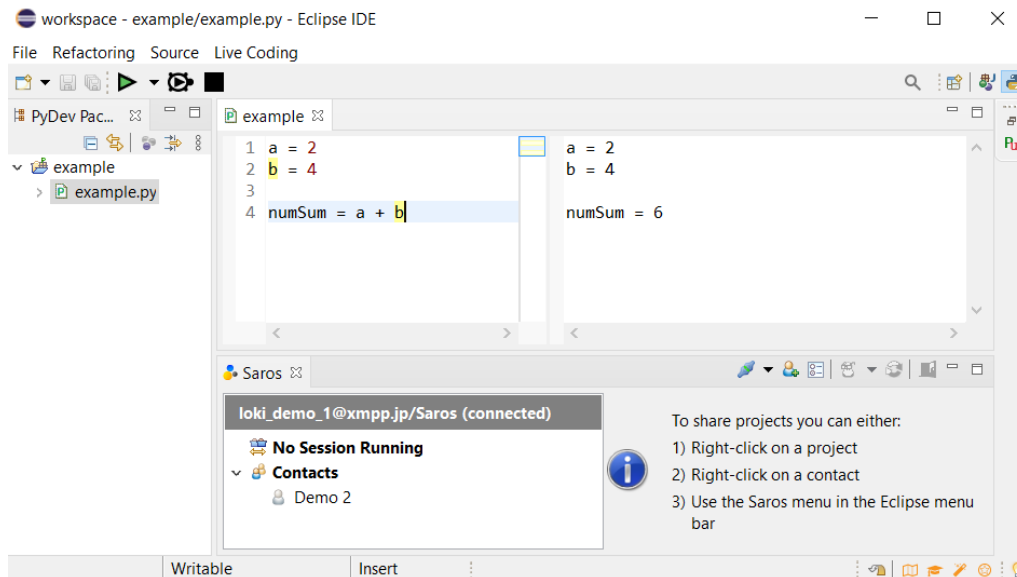


Figure 4.14: Loki - Python Live Programming Environment Running.

The same strategy is followed when one is programming with the turtle library, one only needs to change the execution action button (Figure 4.15).

Finally, the menu available to manage the unitary tests with PyUnit is easily accessible on the right tool bar, and, when chosen, a panel appears with a synthesis of the tests' results (Figure 4.16).

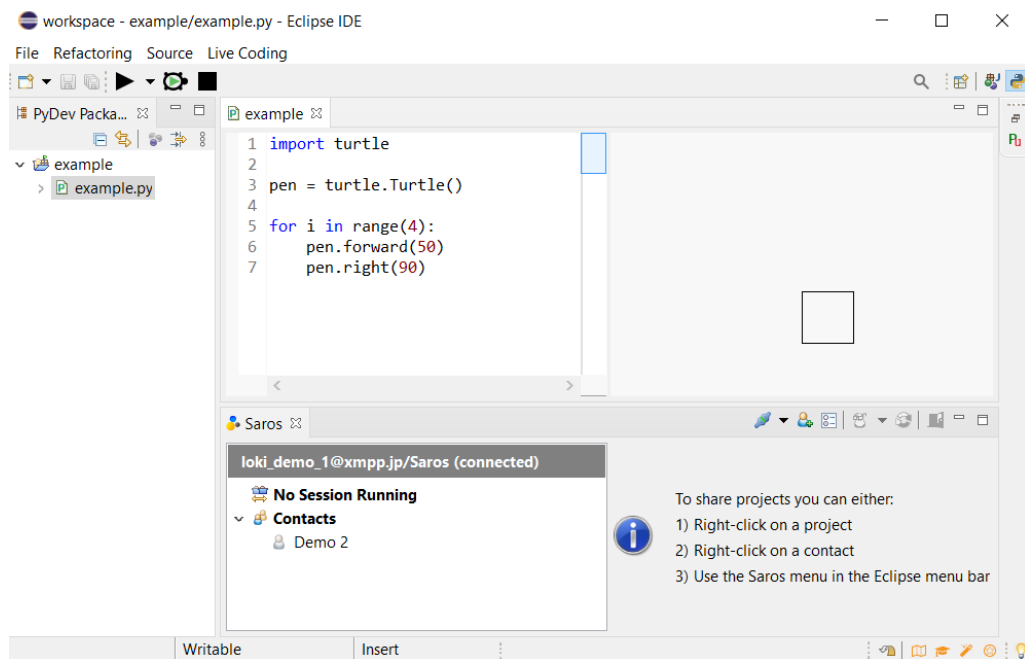


Figure 4.15: Loki - Turtle Live Programming Environment Running.

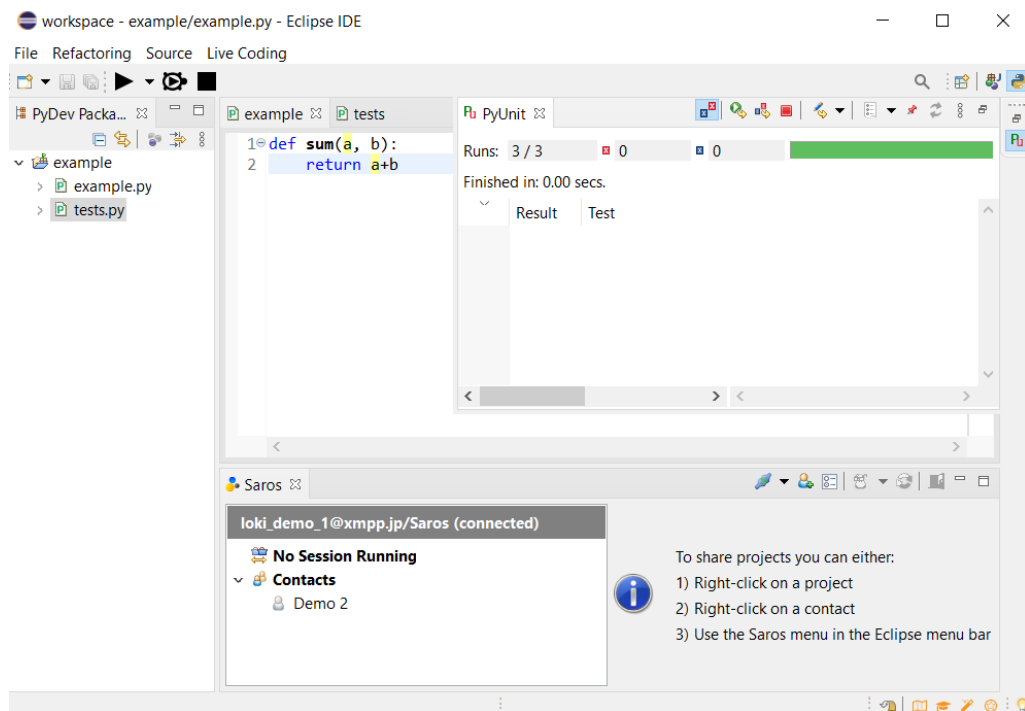


Figure 4.16: Loki - Automated Tests' Results Panel.

4.7 Summary

In this chapter, the development process of Loki is described.

Loki was developed using the Eclipse IDE as a base, to which several plugins were added (section 4.2). Some of these plugins were installed to add the needed features, such as PyDev, to add Python support; Live Programming in Python, to add LP support for Python, and Saros, to add the CD feature. There was also one plugin installed for analytics purposes during the validation experiment, Code Time, which monitors coding activity and generates statistics.

The UI of Eclipse is very complex and features several menus and tool icons. Additionally, the more plugins there are installed, the more menus and tools are added to the UI. So, the UI was edited to make it more straightforward and intuitive for young students (section 4.3).

After completing Loki's creation, several ways of sharing it were studied. The tools Oomph and Yatta were tested but discarded. The chosen method for sharing Loki was creating a compressed folder with all the needed components to run the platform and share it along several machines (section 4.4).

The platform created is compatible with Windows, Linux and macOS. To execute it, a JRE and Python installation are needed. OS-specific installation instructions and system requirements can be found in Section 4.5.

Chapter 5

Validation

5.1	Live <i>versus</i> Not Live: Experiment With College Students	43
5.2	Live <i>versus</i> Not Live: Experiment With 10-14 Year Olds	49
5.3	Collaborative <i>versus</i> Individual Development: Experiment With 10-14 Year Olds	50
5.4	Expert Evaluation	50
5.5	Summary	54

In this chapter, we present and describe the multiple ways through which Loki was validated. We detail the planning, execution and results of the experiments with college students (section 5.1) as well as the planning made for the experiments in schools with students from 10 to 14 years old, which ended up not being performed due to pandemic related constraints (sections 5.2 and 5.3). Finally, we describe the expert evaluation process, as well as detail the results of the interviews and surveys performed (section 5.4).

5.1 Live *versus* Not Live: Experiment With College Students

5.1.1 Designing the Experiment

The goal of this experiment was to understand the impact of LP on college students' performance at solving programming exercises in Python using Loki.

Rationale

There were two experimental groups, one with Loki and one with a very similar Eclipse distribution but without LP (the control group). The goal was to evaluate the number of exercises solved by each student in 15 minutes.

Subjects

The experiment subjects consisted of students from MIEIC at FEUP. All students were invited to participate in the experiment. To this invitation answered 29 students. From these, 27 completed the experiment within the allowed time frame.

Tasks

All of the subjects were asked to complete a set of tasks to prepare for the experiment, and then the tasks of the experiment itself:

- **Sign up** for the experiment in the form sent to their emails. This form requested basic demographic information such as their college year, as well as their name, student number and email for contact and identification if needed.
- **Download** the experimental package that was sent to them. Loki was available for Windows, Linux and macOS. The package also contained installation and usage instructions.
- **Solve** the Moodle test once it became available to them, using the Loki version sent to them.

Factors

The experiment consisted of observing the effect of LP on problem-solving efficiency with Python.

The control variable of the experiment was the Eclipse distribution version sent to the participants. The Control Group had available an Eclipse distribution supporting Python, but nothing else. The Liveness Group had Loki, supporting Live Programming in Python.

Attributes of Interest

The experiment's goal was to evaluate if the presence of the LP feature in any way affected the capacity of the students to solve the exercises presented to them in the test.

Measurements

To measure the students' efficiency in solving the test, all students were given 15 minutes. Then we evaluated the number of exercises solved by each of them, considering the experimental group they were assigned to.

Threats to Validity

This experiment was performed remotely, and, as such, the students had to perform the Loki setup on their own. Because of this, mistakes might have happened during this process.

5.1.2 Selecting the Subjects

All the participants are MIEIC students from FEUP who volunteered for this experiment. Their distribution by college year is shown in Figure 5.1. As we can see, more than half of the students

belong to the last two years of the degree (totaling over 65% of the volunteers). The rest of the participants are distributed more evenly throughout years 1, 2 and 3 of the degree.

Distribution of the Volunteers by College Year

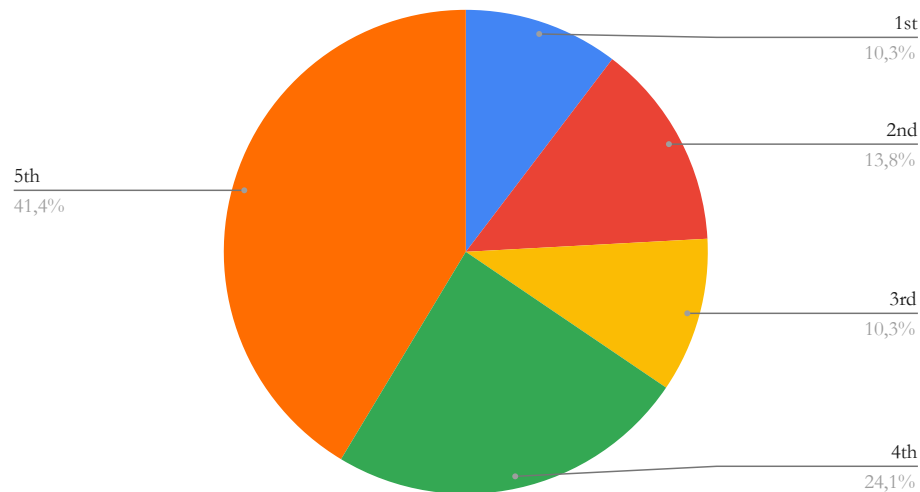


Figure 5.1: Distribution of the Volunteers by College Year.

The students were distributed by each experimental group (Live and Control), following the order they signed up for the experiment. So, for every two signed-up students, the first went to the Liveness Group and the second to the Control Group. This ended up totaling 15 students in the Liveness Group and 14 students in the Control Group. There are a few reasons why we decided to assign the students this way, namely:

- To allow for an unbiased distribution of students. Since all the participants are Informatics students from FEUP, attending the same course that this thesis is being prepared for, it is natural that some of them are of acquaintance. Distributing them randomly avoided unconscious bias that could temper with the experiment's results.
- One of the questions pondered while distributing students was if it made sense to redistribute them after the sign-up period ended, evening the number of students of each college year by each experimental group. However, this option was discarded since college year does not equal better or worse Python programming capability. For example, students from years 1 through 3 had Python lectures in the first year, while students from years 4 and 5 did not, having learned it independently.

5.1.3 Running the Experiment

The experiment was not presential nor synchronous, and each of the participants could choose, within a specific time frame (from the 20th of May to the 2nd of June), when to participate and could do so at home.

Each one of the participants was sent the platform installation package and the Installation and Usage Guides (available in Section A.2) and performed the setup autonomously. These files were all available in a Google Drive folder to which they were sent a viewing link.

The questions to be solved were available in a Moodle test (available in Section B), and the answers were submitted there. This allowed for the automatization of the correction process, as well as to collect more data since Moodle registers the number of submissions per question and the timestamps of each answer submission.

As for the test itself, it was designed to have two distinct parts: algorithmic logic (programming regular Python scripts) and graphical drawings (with the Turtle library). Within each part of the test, the questions have an increasing level of difficulty.

As for why having the test consisting of two different kinds of questions, we wanted to evaluate the impact of the LP feature in different types of questions and if it differed significantly or not. However, since most students have never used the Turtle library, the algorithmic questions were given priority, being the first ones to be presented to the participants.

Instructions on how to structure Python scripts using the Turtle library were also provided to the participants during the test, at the beginning of the graphical section.

Finally, help was available to the participants at all times, since it was given to them both an email address and a Discord server to contact us if any doubt should arise.

5.1.4 Processing the Results

Unfortunately, due to an error with the plugin itself, Code Time did not register any participants' data. Moreover, we have also had information of students who did not complete the login step due to forgetfulness, so the data would most likely have been inconsistent even if the plugin had worked.

The complete data of the participants' college year, test results and solving times can be found in Section D.1. In this section, we will highlight what we believe to be the most relevant findings.

For each student completing the assessment, the data about how many exercises were solved and the solving time were registered. The solving time would be a useful metric should any participant finish the whole test before the 15-minute time frame ended, but as no one did, this metric was not used.

The first metric to be evaluated was the average of exercises solved by each group, which can be found in Figure 5.2.

Average Number of Exercises Solved by Each Group

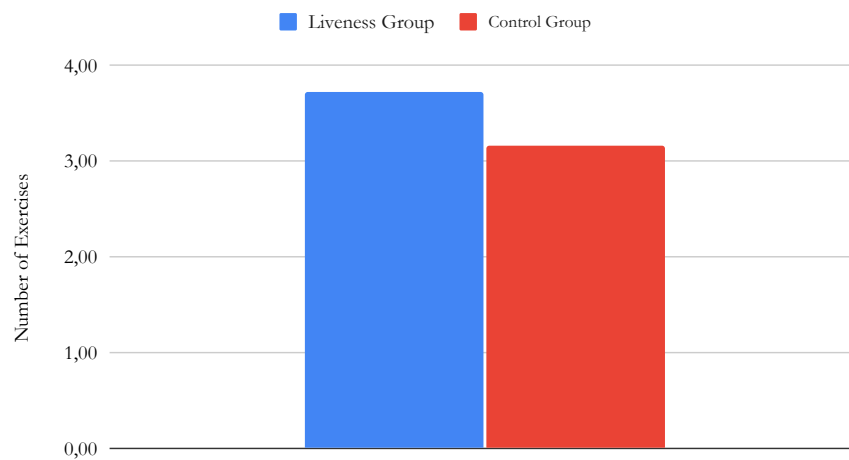


Figure 5.2: Average Number of Exercises Solved by Group.

As we can verify, the average of exercises solved by the participants in the Liveness group is higher than the average of the Control group. This was expected since we believed the LP feature would make development faster and less prone to errors.

Then, we proceeded to analyse the percentage of students in each group to solve each particular exercise (Figure 5.3) and the percentage of students in each group to solve a certain number of exercises (Figure 5.4).

Relative Frequency of Number of Students to (Fully or Partially) Solve Each Exercise in Each Group

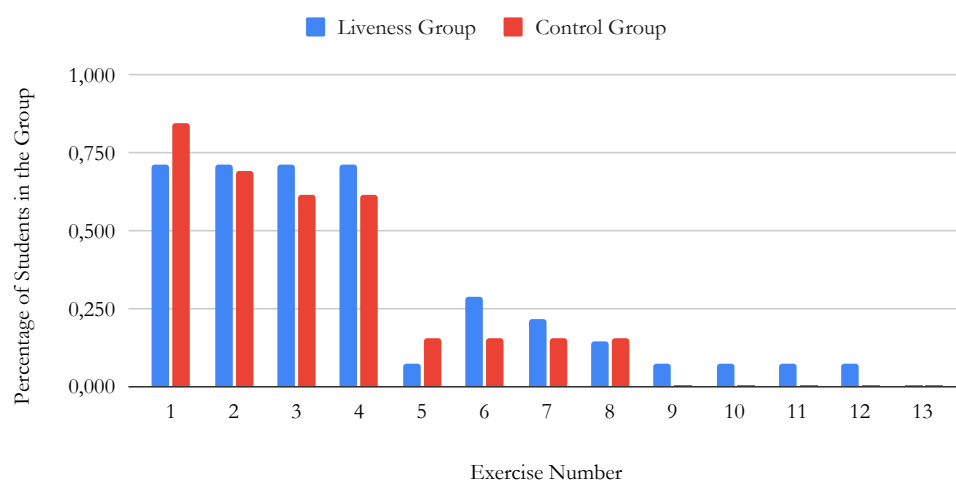


Figure 5.3: Relative Frequency of Number of Students to (Fully or Partially) Solve Each Exercise in Each Experimental Group.

Relative Frequency of Number of Exercises Solved in Each Group

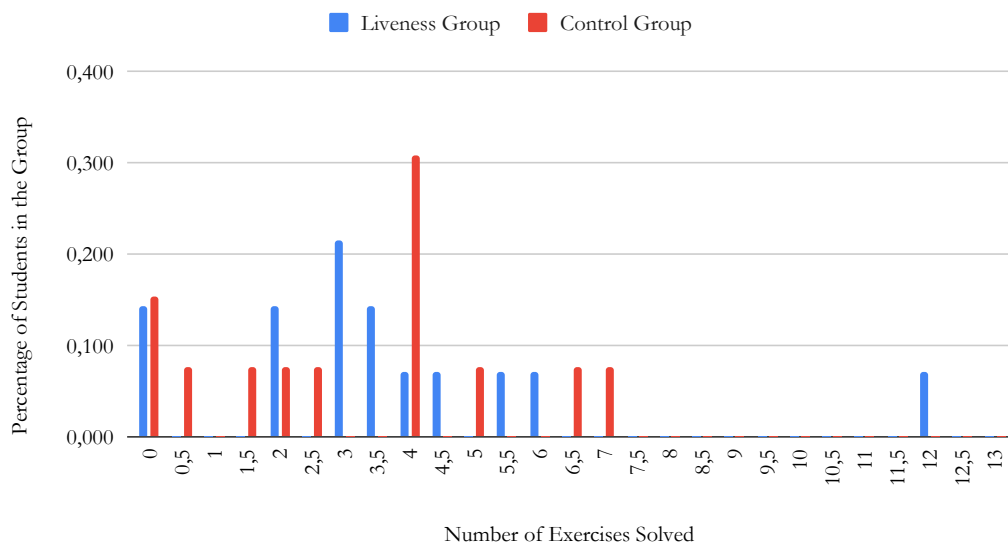


Figure 5.4: Relative Frequency of Number of Exercises Solved in Each Group.

As we can verify, out of the 13 questions in the test, only 12 were solved by at least one participant. Furthermore, out of those 12, only 3 had a more significant percentage of participants from the Control group to solve it, making all the other 9 questions having more correct solves from Liveness group participants.

Furthermore, considering that the graphical questions start from question 6, it is clear that the Liveness group performed better than the Control group in these questions overall, which might mean that the influence of the LP feature has a more meaningful impact on this type of question than in regular algorithmic challenges.

From the two charts above, we can see that there is a considerable variation between the average of a group and the solves of each student. Because of this, we also studied the deviation from the average for each group. This can be found in Table 5.1.

Table 5.1: Experiment Results Summary.

	Liveness Group	Control Group	Overall
Total Students	14	13	27
Average	3,71	3,15	3,44
Highest Positive Deviation	8,29	3,85	8,56
Highest Negative Deviation	3,71	3,15	3,44

From the table 5.1, we can see that the Highest Negative Deviation is the same as the average in both experimental groups. This is because in both groups there were participants that solved no questions, and, as such, their score was 0. It is also clear that the Highest Positive Deviation

is higher in the Liveness group than in the Control group. This is due to the one student in the liveness group that correctly solved every question up to question 12. The maximum number of correct answers in the control group was 7.

We then removed the outlier element that creates the highest deviation (the participant who solved 12 exercises). The new average for the Liveness Group is 3,08. This is below the Control Group's average.

We then proceeded to additionally remove the participants who scored 0 in both groups (2 in each group) and the highest scoring participant of the Control Group, who scored 7. The results are shown in Table 5.2.

Table 5.2: Experiment Results Summary - Removing Outliers.

	Liveness Group	Control Group
Average	3,64	3,40

In this scenario, the Liveness Group has a higher average again, but the Control Group's average is much closer than in the original scenario evaluated.

For this experiment, the expected results were the Liveness Group having a clear advantage over the Control Group, demonstrating a significantly higher average of solved exercises. However, analysing the results obtained, one can not state with certainty that the Liveness Group is better than the Control Group. Overall, we do find the results of this experiment to be mostly inconclusive due to the small size of the experimental sample (27 students) and the unfavourable conditions in which this experiment had to be performed (remotely with autonomous setup).

Because of the aforementioned reasons, a template package to replicate this experiment was created, so it can hopefully be repeated under more favorable conditions in the future. The package can be found [here](#).

5.2 Live versus Not Live: Experiment With 10-14 Year Olds

This experiment was designed to test the effect of LP on Python development. There would be two experimental groups solving the same exercise sheet: one with Live Programming and one without.

All the details of this experiment are equal to those of the experiment described in Section 5.1, except for the **subjects**, who were supposed to be 10 to 14-year-old students; and the **Tasks**, since the sign-up and download tasks would not exist in this case and all the participants had to do was to solve the test. Because of this, the **threats to validity** would also have been different, even though we cannot state which would have been due to the experiment not have been performed.

This experiment was fully prepared and ready to be performed in schools:

- Contacts were made with the ENSICO Master Teachers responsible for teaching Python in schools to set up the experiment.

- It was soon made clear that no installations could be performed in schools' computers, so several alternatives were arranged to mitigate this issue, such as creating a platform version executable from a USB drive and later a version that did not need a complete Python installation on the machine, being all available from the USB.
- Usage guides for children were created in order for them to understand the tasks and the platform's functioning. These guides, one for the LP group and one for the control group, can be found in Section A.3. These guides are similar to those sent to the college students, but with a simpler language adapted to the children's age, and considering a presential experiment, where help would be available should they need it.

However, in early May, we were informed that the lessons were severely delayed due to the pandemic and that the students had not yet started to program. Because of that, they would not be able to participate in any experiment.

5.3 Collaborative *versus* Individual Development: Experiment With 10-14 Year Olds

This experiment was designed to test the effect of Collaborative Development on Python development. There would be two experimental groups solving the same exercise sheet: one group where students would be working in pairs and other where the students would be working alone. The experimental group where the students were supposed to collaborate should perform the experiment separately to test the efficacy of the remote collaboration feature.

All of the details of this experiment are the same as the details of the experiment described in Section 5.2, apart from the **Factors** and **Attributes of Interest**, in which the variable to be studied and analysed would be the presence of CD, and not LP.

However, due to COVID-19, the teaching plans were delayed, and all plans for the realization of this experiment were discarded.

Prior to this cancellation, the Loki version supporting CD was developed to accommodate this experiment, and demonstrations were prepared and performed for the ENSICO Master Teachers.

5.4 Expert Evaluation

Besides having the empirical experiment to test Loki, we considered it essential to have expert reviews, where experts in informatics and in teaching informatics to 10 to 14 year-olds could express their opinions and evaluate the platform.

To this end, the ENSICO Master Teachers were invited to answer a short online survey (Annex C) in which they could weight in both the importance of the platform in the classroom and its importance to the development of Computational Thinking.

5.4.1 Description

We organized a series of face-to-face (remote) interviews to discuss and complement the data collected in the survey. It is important to notice that all of the ENSICO Master Teachers had access to Loki and could experiment with it, but not all of them completed the setup, receiving it already installed and ready. This fact impacts some of the answers to the interview that focused on the setup process.

5.4.2 Results

The results from the Expert Evaluation were derived from the answers provided in the survey sent to the ENSICO Master Teachers and the interviews. In total, 4 Master Teachers completed both of these steps: Luís Neves, Rui Grandão, Inês Guimarães and Liliana Monteiro.

The interviews did not follow a strict script, but an effort was made to get an answer to the following topics:

- **Background:** What was their area of studies (sciences, engineering, other).
- **Programming experience:** If they know how to program and, if the answer was yes, for how long they have been programming and with what languages.
- **Familiarity with IDEs:** If throughout their academic and professional lives they used an IDE and how often (especially Eclipse).
- **Setup:** What their opinions were on the Loki setup (when provided with the installation guide).
- **Classroom demographics:** What is the age of the students they are teaching CS to, and if they believe Loki to be adapted to those ages.
- **Improvement suggestions:** If they did not believe Loki to be intuitive for their students, if they think it would be suited for older students and / or what changes would they suggest to improve it.
- **LP importance:** Finally, we also wanted to have some detail into their opinion about the importance of the integration of LP in this kind of learning platform.

Next, we will summarize the findings, combining information from the survey answers and the interview. In Section D.2 are the full survey answers from each of the participants.

All of the Master Teachers teach students ranging from 10 to 14 years old.

Master Teacher Luís Neves

Master Teacher Luís Neves studied CS at the University of Minho, a 5-year degree with half of the contents focusing on mathematics and half focusing on Computation and Programming. As a result, he has a high experience programming, both in his academic and professional life.

He has used IDEs frequently throughout his life, mainly Visual Studio and VS Code, both from Microsoft. He had heard of Eclipse but had never used it.

As for the Loki setup guide and process, he stated that he found it very simple and straightforward. He said, however, that he thought that this guide was not simple enough to be followed just by anyone, only by people in the IT area.

When asked if he found Loki to be suited for the ages he teaches to, he mentioned that ENSICO's focus for children that young is not an installed full IDE, but more of a step by step process, where they start by just seeing someone programming, proceed to program on paper, advance to some simple assisted programming tool, such as Jupyter and only then should they proceed to an IDE. From his experience, he also believes that having a web environment is preferable over an installed environment because of the lack of setup. However, he did say that looking past the installation process and the learning process that they have created for ENSICO, Loki's environment is simple enough for children.

When asked what would be important points to improve, he said that the LP visualisation could be better and more appealing to the children. He mentioned the tool Python Tutor, which supports LP in Python, as a good example of it.

Finally, regarding the integration of LP in this kind of learning platform, he believes it to be very useful. He believes that:

"[...] they [the students] will be experiencing a permanent "dialogue" with the machine (computer). And such a permanent "dialogue" will allow them to better understand how programs and algorithms really work."

Master Teacher Rui Grandão

Master Teacher Rui Grandão studied Informatics Engineering. He has extensive experience with programming, both in his academic and professional life.

He has had experience with multiple IDEs, including Eclipse.

As for the Loki setup guide and process, he stated that he found it easy.

When asked if he found Loki to be suited for the ages he teaches to, he mentioned that the students had not programmed much yet, so their knowledge of operating a platform like Loki would be limited. However, he also mentioned that if they were programming for a bit longer, he believed Loki to be intuitive for them, probably being used after a more accessible beginner platform such as Scratch.

When asked what would be important points to improve, he said that at the time, none came to mind.

Finally, when asked about the importance of LP in teaching CS to children, he said that:

"I believe Live Programming is a feature that will make teaching programming to young children more interactive and entertaining."

He also stated that instant gratification is currently an important factor in children and, as such, LP is important, as it gives programming an almost gaming component.

Master Teacher Inês Guimarães

Master Teacher Inês Guimarães is currently studying Mathematics at the University of Porto. She has little experience with programming, being limited to a few curricular units during her degree, focusing on Python and Haskell.

She has virtually no experience with IDEs, including Eclipse.

As for the Loki setup, no insight could be given since she already received it fully operational and did not need to complete the installation process.

When asked if she found Loki suited for the ages she teaches to, she said that, even though their students are not programming yet, yes, she believes it to be simple enough for children that young to use.

When asked what would be important points to improve, she said that maybe something *"more visual, appealing (...) and with better graphical support"* could help the students to engage even more.

Finally, regarding the integration of LP in this kind of learning process, she believes that the feedback of seeing the steps of the code that is being developed is helpful and that it helps to understand what is going on in the intermediate steps. She finds it particularly helpful when errors are involved since it is easier to understand where it is failing. Overall she believes that:

"(...) it helps them visualize the code they have written, showing them what they are doing right or wrong, and even hinting at where the mistakes might be. This clearly promotes a better understanding of specific computer programs and programming paradigms themselves."

Master Teacher Liliana Monteiro

Master Teacher Liliana Monteiro completed 2 years of an Informatics degree and then changed to a Mathematics degree, which she is currently pursuing. She has the 2 years of the Informatics degree in programming experience, having programmed in C, Haskell and Java, but has not programmed much since.

She has used IDEs during her time in the Informatics degree, but has never used Eclipse before.

As for the Loki setup, no insight could be given, since she already received it fully operational and did not need to complete the installation process.

When asked if she found Loki suited for the ages she teaches to, she said that she believed that the older ones in the group definitely would be capable of using it, but was not entirely sure about the younger ones, even though she was inclined to say that they, too, would be able to use it.

When asked what would be important points to improve, she said that she thought that Loki was very intuitive, but perhaps the only change that she would suggest was to make it more visually appealing, more beautiful.

Finally, regarding the integration of LP in this kind of learning process, she entirely agrees with it. She believes that since the children are starting their programming learning, they need something simple and intuitive to write their code and see the results. She also believes that, since the newer generations are more prone to demand immediate results, the LP feature will help them to keep their focus on their work.

Overall Findings

From the summaries presented above, it is clear that the Master Teachers believe Loki to be adequate for the children within the ages they teach to, even though they are not using tools like it just yet, mainly due to pandemic re-planning needs.

The reviews of the LP feature are also overall positive, but there were multiple suggestions of improvement of the visualization, being that something that could be improved in the future. The installation process was also focused as something to be simplified, culminating, ideally, in requiring no installation process at all, and transforming Loki in a web-based programming environment.

As for the importance of LP in the teaching and learning of programming, all of the Master Teachers agreed that it is a powerful and important tool that can help the students stay focused and engaged for longer, and even helping them understand better what their code does and how it behaves. Furthermore, and from the data of the survey they answered to, all of them agree that developing CT, which learning to program does, can help students achieve better results at all of their school subjects, not only STEM, and that it can equip them with better mental skills to face real-life problems in their future.

5.5 Summary

In this chapter, Loki's validation processes are described.

Even though the initial plan was to perform two experiments with 10 to 14-year-old students, one to evaluate the Live Programming feature and the other to evaluate the Collaborative Development feature, these were not possible due to the COVID-19 pandemic. Furthermore, while the CD experiment was only loosely drafted, the LP experiment was fully prepared and was ready to be applied. Unfortunately, it was only in May that we were informed that it would not be possible.

To make up for these challenges, we designed a similar experiment to what was drafted for the younger students, but applied it to college students. These students were from the MIEIC course from FEUP and volunteered for this experiment. The experiment was performed remotely, and each participant made Loki's setup independently and solved a Python programming test on Moodle. However, this experiment's results turned out to be inconclusive, as the values for the Liveness group and Control group were very similar and the experimental sample was small.

Finally, a survey and a set of interviews were made with the ENSICO Master Teachers, from which we gathered information, not only about Loki but also about the importance of LP in the teaching and learning of programming. Loki obtained overall good reviews on its usability, even

though it was mentioned that its installation could be easier and the interface could be more appealing. It was, though, consensual among the Master Teachers that the LP is essential in the programming learning process, as it makes the processes more transparent and clear.

Chapter 6

Conclusions and Future Work

6.1	Conclusions	57
6.2	Contributions	58
6.3	Challenges	58
6.4	Future Work	59

6.1 Conclusions

Computational Thinking is becoming an essential skill, impacting the students' critical thinking and problem-solving capabilities.

Throughout the State of the Art research, it became clear that multiple countries have taken steps to integrate Computer Science classes into the curricula, thus creating room to develop Computational Thinking. It also became clear that programming helps to develop Computational Thinking, as it creates a tangible mean to study it.

We also found many programming platforms within the segments we were aiming to create our own: programming for young students, programming in Python, Live programming and customizable environments. However, throughout our research, we found none that gathered all the requirements we deemed necessary. So, we developed Loki.

Loki was developed by creating a customized Eclipse distribution and manipulating it in order to simplify it as much as possible while gathering the features we deemed essential: Live Programming in Python, Collaborative Development and Automated Testing.

Loki was then validated in an experiment with college students from FEUP. This experiment was performed remotely, and each student had to setup their own environment, although help was available online whenever needed. This experiment's results ended up being inconclusive, as the number of participants was low and the results between the control group and the group using Loki were too close to be considered meaningful. A replication package for this experiment was

created, hoping that one day it can be repeated in more suitable conditions to evaluate Loki's effectiveness better.

Expert reviews also supported that Loki can be a valuable asset in teaching Computer Science in schools, but it might still need some work on the interface component. These reviews also support that Live Programming is an important feature to have when teaching programming to students.

However, we do hope that this work has contributed to better understanding the current state of Computer Science education worldwide and providing a tool to apply in the classroom and at home, where students can explore programming in an engaging way.

6.2 Contributions

Throughout this work, several contributions were made, such as:

- **State of the Art research** on:
 - The benefits of Computer Science in schools.
 - CS school programs around the world.
 - Different types of programming environments, such as programming environments for children and teenagers, programming environments supporting Live Programming, customizable programming environments and Python programming environments supporting LP.
- **Loki**: a programming platform supporting Live Programming, Collaborative Development and Automated Testing that was developed and tested during the course of this work.

6.3 Challenges

There were many challenges to be faced throughout the development of this work.

Firstly, concerning Loki's development itself, it was a very complex process to understand how one could share the configurations precisely as they were created without losing information in between. This problem was later exacerbated by the issue of not being able to install anything on school computers, where we initially intended to perform the experiments. As described in Section 4.4.3, these problems were solved by creating a shareable package with the platform that could be executed from a USB drive. This process kept all configurations intact and avoided any installation on school machines.

However, the main challenges that came up during this work were due to the COVID-19 pandemic. Due to both the mandatory quarantine and the advised social distancing once the quarantine

ended, this entire work was developed remotely. This not only added difficulty to the communication between student-supervisor, requiring rigorously scheduled appointments, but also significantly reduced discussion of multiple topics between peers, which would have been simpler if we had been working in person at college.

Finally, the most complex challenge faced, also due to the pandemic, was related to the experiments planned. As described in Chapter 5, two experiments were planned with 10 to 14-year-old students to test Loki's adequacy and efficiency. However, both experiments were cancelled due to the extreme delay in the programming curriculum caused by the pandemic. The most severe issue was the cancellation of the most critical experiment only one and a half months from the final delivery, which left us with little room to prepare and launch a whole new experience. This, however, was done, and a similar experiment to what we had planned to do with the school students was performed instead of with college students.

6.4 Future Work

The final result of this work is Loki, a Proof of Concept platform supporting Live Programming, Collaborative Development and Automated Testing in Python in a simplified IDE environment.

Loki was validated through an experiment and expert reviews (Chapter 5). However, there are still improvements that could be made in the future.

Firstly, one add-on that was discussed during the development of this work was Version Control. This was not finished due to the lack of time. However, it would be an important feature to simplify and have in the future.

Moreover, it was concluded, particularly after the interviews with the ENSICO Master Teachers (Section 5.4), that Loki would greatly benefit from becoming a web environment instead of an installable IDE. Considering that many IDEs are progressively offering online versions, it is possible that Eclipse might follow those steps and that this tool can one day be transferred to a web environment. Another improvement suggestion that derived from the interviews concerned the LP data visualisation. Some Master Teachers suggested that the visualisation could be improved, featuring more appealing graphics and simplified logic.

Finally, this work would greatly benefit from a new validation experiment, taking place post-COVID-19, when the target audience (school-aged children) is prepared for it and featuring, preferably, more participants for a more accurate statistical evaluation.

References

- [AJP15] Ahmad Al-Jarrah and Enrico Pontelli. AliCe-ViLlagE Alice as a Collaborative Virtual Learning Environment. *Proceedings - Frontiers in Education Conference, FIE*, 2015-Febru(February), 2015.
- [Ali15] Alice. Our history, 2015. Available at <https://www.alice.org/about/>, Accessed last time in January 2021.
- [Ama18] Diogo da Silva Amaral. Towards a Live Software Development Environment. 2018.
- [AMS13] Soheli Ahmad, Debasish N. Mallick, and Roger G. Schroeder. New product development: Impact of project characteristics and development practices on performance. *Journal of Product Innovation Management*, 30(2):331–348, 2013.
- [App18] Apple. Swift playgrounds, 2018. Available at <https://www.apple.com/swift/playgrounds/>, Accessed last time in January 2021.
- [aS21] Computing at School. Computing at school: About us, 2021. Available at <https://www.computingatschool.org.uk/about>, Accessed last time in January 2021.
- [Ass16] Computer Science Teachers Association. About csta, 2016. Available at <https://www.csteachers.org/Page/about-csta>, Accessed last time in January 2021.
- [AVR20] Barbara Arfé, Tullio Vardanega, and Lucia Ronconi. The effects of coding on children’s planning and inhibition skills. *Computers and Education*, 148(December 2019), 2020.
- [BAL10] Tim Bell, Peter Andreae, and Lynn Lambert. Computer Science in New Zealand High Schools. *Conferences in Research and Practice in Information Technology Series*, 103(May 2014):15–22, 2010.
- [BB03] Grady Booch and Alan W. Brown. Collaborative Development Environments. *Advances in Computers*, 59(C):1–27, 2003.
- [BBP20] Emily Anne Brown, Richard S Brown, and D Ph. The Effect of Advanced Placement Computer Science Course Taking on College Enrollment. 2020.
- [BCD⁺16a] Stefania Bocconi, Augusto Chiocciariello, Giuliana Dettori, Anusca Ferrari, Katja Engelhardt, Panagiotis Kampylis, and Yves Punie. *Developing Computational Thinking in Compulsory Education - Implications for policy and practice*. Number June. 2016.

- [BCD⁺16b] Stefania Bocconi, Augusto Chioccariello, Giuliana Dettori, Anusca Ferrari, Katja Engelhardt, Panagiotis Kampylis, and Yves Punie. Exploring the field of computational thinking as a 21st century skill. 07 2016.
- [BHC⁺12] Iris Zur Bargury, Bruria Haberman, Avi Cohen, Orna Muller, Doron Zohar, Dalit Levy, and Reuven Hotoveli. Implementing a new Computer Science Curriculum for middle school in Israel. *Proceedings - Frontiers in Education Conference, FIE*, 2012.
- [BMD⁺15] Erik Barendsen, Linda Mannila, Barbara Demo, Nataša Grgurina, Cruz Izu, Claudio Mirolo, Sue Sentance, Amber Settle, and Gabriele Stupuriene. *Concepts in K-9 computer science education*. 2015.
- [BNDJ14] Tim Bell, Heidi Newton, Caitlin Duncan, and Sam Jarman. Adoption of Computer Science in NZ schools. *Proceedings of ITx - New Zealand's Conference of IT*, pages 203–209, 2014.
- [Bri13] Jason R. Briggs. *Introduction: Why Python?* No Starch Press, 4 edition, 2013.
- [Bun07] Alan Bundy. Computational Thinking is Pervasive. *Journal of Scientific and Practical Computing*, 1(2):67–69, 2007.
- [BWK05] Stefan Berner, Roland Weber, and Rudolf K. Keller. Observations and lessons learned from automated testing. *Proceedings - International Conference on Software Engineering*, 2005:571–579, 2005.
- [CFZ19] Jeanne Century, Kaitlyn Ferris, and Huifang Zuo. Preliminary Findings of an Exploratory Study Finding Time for Computer Science in the Elementary Day. page 3, 2019.
- [CNB10] Luke Church, Chris Nash, and A. F. Blackwell. Liveness in Notation Use: From Music to Programming. In *Proceedings of the 22nd Annual Workshop of the Psychology of Programming Interest Group (PPIG 2010)*, pages 2–11, 2010.
- [Cod13a] Code.org. About code.org, 2013. Available at <https://code.org/about>, Accessed last time in January 2021.
- [Cod13b] Code.org. Promote computer science, 2013. Available at <https://code.org/promote>, Accessed last time in January 2021.
- [Cod15] Visual Studio Code. Documentation for visual studio code, 2015. Available at <https://code.visualstudio.com/docs>, Accessed last time in February 2021.
- [CZW14] M. Chandramouli, M. Zahraee, and C. Winer. A fun-learning approach to programming: An adaptive virtual reality (vr) platform to teach programming to engineering students. In *IEEE International Conference on Electro/Information Technology*, pages 581–586, 2014.
- [DLA15] Aline Dresch, Daniel Pacheco Lacerda, and José Antônio Valle Antunes. *Design science research: A method for science and technology advancement*. 2015.
- [DRP99] E. Dustin, J. Rashka, and J. Paul. *Automated Software Testing: Introduction, Management, and Performance*. Addison-Wesley, 1999.

- [Ens21] Ensico. Ensico, 2021. Available at <https://ensico.pt/>, Accessed last time in January 2021.
- [Fou04a] Eclipse Foundation. Eclipse ide 2020-12, 2004. Available at <https://www.eclipse.org/eclipseide/>, Accessed last time in February 2021.
- [Fou04b] Eclipse Foundation. Enabling open innovation & collaboration, 2004. Available at <https://www.eclipse.org/>, Accessed last time in February 2021.
- [Gol84] Adele Goldberg. *Smalltalk-80 : the interactive programming environment*. Xerox Corporation, Palo Alto, 1 edition, 1984.
- [GR83] Adele Goldberg and David Robson. *Smalltalk-80: The Language and its Implementation*. Xerox Corporation, Palo Alto, 1 edition, 1983.
- [HAB⁺11] Peter Hubwieser, Michal Armoni, Torsten Brinda, Valentina Dagiene, Ira Diethelm, Michail N. Giannakos, Maria Knobelsdorf, Johannes Magenheimer, Roland Mittermeir, and Sigrid Schubert. Computer Science/informatics in secondary education. *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, pages 19–38, 2011.
- [hci21a] hcientist. Online python tutor, 2021. Available at <http://pythontutor.com/live.html>, Accessed last time in May 2021.
- [hci21b] hcientist. Online python tutor git repository, 2021. Available at <https://github.com/hcientist/OnlinePythonTutor>, Accessed last time in May 2021.
- [HGTK10] Remko Helms, Elia Giovacchini, Robin Teigland, and Thomas Kohler. A design research approach to developing user innovation workshops in second life. *Journal For Virtual Worlds Research*, 3, 04 2010.
- [IA15] Fawzi Fayeze Ishtaiwa and Ibtehal Mahmoud Aburezeq. The impact of Google Docs on student collaboration: A UAE case study. *Learning, Culture and Social Interaction*, 7:85–96, 2015.
- [IKM⁺97] Dan Ingalls, Ted Kaehler, John Maloney, Scott Wallace, and Alan Kay. Back to the Future: The Story of Squeak, A Practical Smalltalk Written in Itself. *SIGPLAN Notices (ACM Special Interest Group on Programming Languages)*, 32(10):318–326, 1997.
- [Jup15a] Project Jupyter. Project jupyter: About us, 2015. Available at <https://jupyter.org/about>, Accessed last time in January 2021.
- [Jup15b] Project Jupyter. Project jupyter home, 2015. Available at <https://jupyter.org/index.html>, Accessed last time in January 2021.
- [Kas14] Dalal Kassem. The Sketchpad Window. *ProQuest Dissertations and Theses*, page 199, 2014.
- [Kay96] Alan C. Kay. The early history of Smalltalk. *History of programming languages—II*, pages 511–598, 1996.
- [kil21] kilon. Pylive coding git repository, 2021. Available at <https://github.com/kilon/pylivecoding>, Accessed last time in May 2021.

- [KP06] Caitlin Kelleher and Randy Pausch. Motivating programming: Using storytelling to make computer programming attractive to middle school girls. *Dai*, 68(01B):369, 2006.
- [Lib21] Python Standard Library. Python standard library - turtle graphics, 2021. Available at <https://docs.python.org/3/library/turtle.html>, Accessed last time in June 2021.
- [Liu21] Shanhong Liu. Global market share held by computer operating systems 2012-2021, by month, 2021. Available at <https://www.statista.com/statistics/268237/global-market-share-held-by-operating-systems-since-2009/>, Accessed last time in May 2021.
- [Liv21] LiveCode. Core benefits of livecode, 2021. Available at <https://livecode.com/core-benefits-of-livecode/>, Accessed last time in January 2021.
- [Lod20] Michael Lodi. Introducing Computational Thinking in K-12 Education: Historical, Epistemological, Pedagogical, Cognitive, And Affective Aspects. 2020.
- [mac16] Getting started with swift playgrounds, Aug 2016. Available at https://www.youtube.com/watch?v=jC_p84oThrs, Accessed last time in January 2021.
- [Mat18] Nuno Guilherme Matos. Towards Live Development of IoT Systems. 2018.
- [McD13] Sean McDirmid. Usable live programming. In *SPLASH Indianapolis 2013: Onward! 2013 - Proceedings of the 2013 International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, pages 53–61, 2013.
- [MGB15] Cecilia Martínez, Marcos J. Gómez, and Luciana Benotti. A comparison of preschool and elementary school children learning computer science concepts through a multi-language robot programming platform. *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, 2015-June:159–164, 2015.
- [MRB⁺04] John Maloney, Natalie Rusk, Leo Burd, Brian Silverman, Yasmin Kafai, and Mitchel Resnick. Scratch: A sneak preview. *Proceedings - Second International Conference on Creating, Connecting and Collaborating Through Computing*, (January 2014):104–109, 2004.
- [MRR⁺10] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. The scratch programming language and environment. *ACM Transactions on Computing Education*, 10(4):1–15, 2010.
- [Nea16] Simona Nicoleta Neagu. The 12 th International Scientific Conference eLearning and Software for Education FACTORS INVOLVED IN ADULT LEARNING. (April):12753, 2016.
- [oC13] Hour of Code. Hour of code faqs - what is the hour of code?, 2013. Available at <https://hourofcode.com/pt/gb>, Accessed last time in January 2021.
- [PG15] Fernando Pérez and Brian E Granger. Project Jupyter : Computational Narratives as the Engine of Collaborative Data Science. *Retrieved September*, (April):1–24, 2015.

- [SC17] Sue Sentance and Andrew Csizmadia. Computing in the curriculum: Challenges and strategies from a teacher's perspective. *Education and Information Technologies*, 22(2):469–495, 2017.
- [Scr07] Scratch. About scratch, 2007. Available at <https://scratch.mit.edu/about>, Accessed last time in January 2021.
- [Sof21] Software. Automate your time tracking, 2021. Available at <https://www.software.com/code-time>, Accessed last time in April 2021.
- [Som13] R. Somasundaram. *Git: Version Control for Everyone*. Beginner's guide. Packt Publishing, 2013.
- [Spi12] D. Spinellis. Git. *IEEE Software*, 29(3):100–101, 2012.
- [Squ96a] Squeak. Squeak wiki: About squeak, 1996. Available at <http://wiki.squeak.org/squeak>, Accessed last time in January 2021.
- [Squ96b] Squeak. Squeak wiki: The birth of squeak, 1996. Available at <http://wiki.squeak.org/squeak/1985>, Accessed last time in January 2021.
- [SSLA18] Maria Spante, Sylvana Sofkova Hashemi, Mona Lundin, and Anne Algers. Digital competence and digital literacy in higher education research: Systematic review of concept use. *Cogent Education*, 5(1), 2018.
- [Sut63] Ivan E. Sutherland. Sketchpad a man-machine graphical communication system. *AFIPS Conference Proceedings - 1963 Spring Joint Computer Conference, AFIPS 1963*, pages 329–346, 1963.
- [Tan90] Steven L. Tanimoto. VIVA: A visual language for image processing. *Journal of Visual Languages and Computing*, 1(2):127–139, 1990.
- [Tan13] S. L. Tanimoto. A perspective on the evolution of live programming. In *2013 1st International Workshop on Live Programming (LIVE)*, pages 31–34, 2013.
- [Tic85] Walter F. Tichy. Rcs — a system for version control. *Software: Practice and Experience*, 15(7):637–654, 1985.
- [Tyn13] Tynker. Coding for kids made easy, 2013. Available at <https://www.tynker.com/>, Accessed last time in January 2021.
- [UNE18] UNESCO. Digital skills critical for jobs and social inclusion, 2018. Available at <https://en.unesco.org/news/digital-skills-critical-jobs-and-social-inclusion>, Accessed last time in June 2021.
- [VFG⁺15] Joke Voogt, Petra Fisser, Jon Good, Punya Mishra, and Aman Yadav. Computational thinking in compulsory education: Towards an agenda for research and practice. *Education and Information Technologies*, 20(4):715–728, 2015.
- [WA06] Tom Wissink and Carlos Amaro. Successful test automation for software maintenance. *IEEE International Conference on Software Maintenance, ICSM*, pages 265–266, 2006.

- [WDB⁺17] Mary Webb, Niki Davis, Tim Bell, Y. Katz, Nicholas Reynolds, Dianne P. Chambers, and Maciej M. Sysło. Computer science in K-12 school curricula of the 21st century: Why, what and when? *Education and Information Technologies*, 22(2):445–468, 2017.
- [wea18] Apple swift playgrounds: Learning to code on ipad, Jan 2018. Available at <https://www.youtube.com/watch?v=syX5cFfV3RQ>, Accessed last time in January 2021.
- [Yat21] Yatta. Profiles for eclipse, 2021. Available at <https://www.yatta.de/profiles/>, Accessed last time in February 2021.

Appendix A

Platform Installation and Usage

A.1 Python installation Script

```
$ python-3.8.0.exe /quiet InstallAllUsers=1 PrependPath=1
```

A.2 Guides Sent to Experiment Participants

A.2.1 Installation Guide

The platform runs in Windows, Linux and MacOS, however, if possible, use the Windows version.

How to install the Eclipse Distribution

Windows

After unzipping the Eclipse folder, run eclipse.exe. When prompted to choose a workspace, choose the “workspace” folder (available in the same folder as the eclipse.exe).

If prompted with choices **after** starting Eclipse (i.e. “Change Perspective?”), click “Cancel” in all of them.

- JRE setup: Open the menu Window > Preferences > Java > Installed JREs. Verify if a path to a JRE is set. If not, set to the folder “plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.1.v20201027-0507\jre”.
- Python setup: Open the menu Window > Preferences > Pydev > Interpreters > Python Interpreter and click New > “Browse for Python/pypy exe”. Browse to the Eclipse > Python folder and choose the “python.exe” file. Check all items in the next step, and click “Apply and Close”.

Linux

After unzipping the Eclipse folder, run the eclipse executable. When prompted to choose a workspace, choose the “workspace” folder (available in the same folder as the eclipse executable). You need a working Python3 installation on your computer.

If prompted with choices **after** starting Eclipse (i.e. “Change Perspective?”), click “Cancel” in all of them.

- JRE setup: Open the menu Window > Preferences > Java > Installed JREs. Verify if a path to a JRE is set. If not, set to the folder “plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64_15.0.2.v20210201-0955/jre”.
- Python setup: Open the menu Window > Preferences > Pydev > Interpreters > Python Interpreter and click New > “Choose From List”. Choose the most recent Python version, check all items in the next step, and click “Apply and Close”.

MacOS

After unzipping the Eclipse folder, run the eclipse executable package. When prompted to choose a workspace, choose the “workspace” folder (available in the same folder as the eclipse executable). You need a working Python3 installation on your computer.

If prompted with choices **after** starting Eclipse (i.e. “Change Perspective?”), click “Cancel” in all of them.

- JRE setup: Open the menu Eclipse > Preferences > Java > Installed JREs. Verify if a path to a JRE is set. If not, set to the folder “plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_15.0.2.v20210201-0955/jre”.
- Python setup: Open the menu Eclipse > Preferences > Pydev > Interpreters > Python Interpreter and click New > “Choose From List”. Choose the most recent Python version, check all items in the next step, and click “Apply and Close”.

Please also log in to the Code time plugin. To do so, follow the steps in the images below:

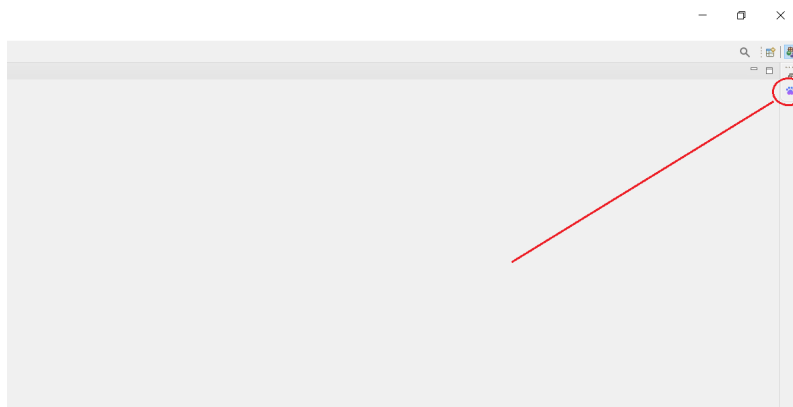


Figure A.1: Code Time Setup Step 1.

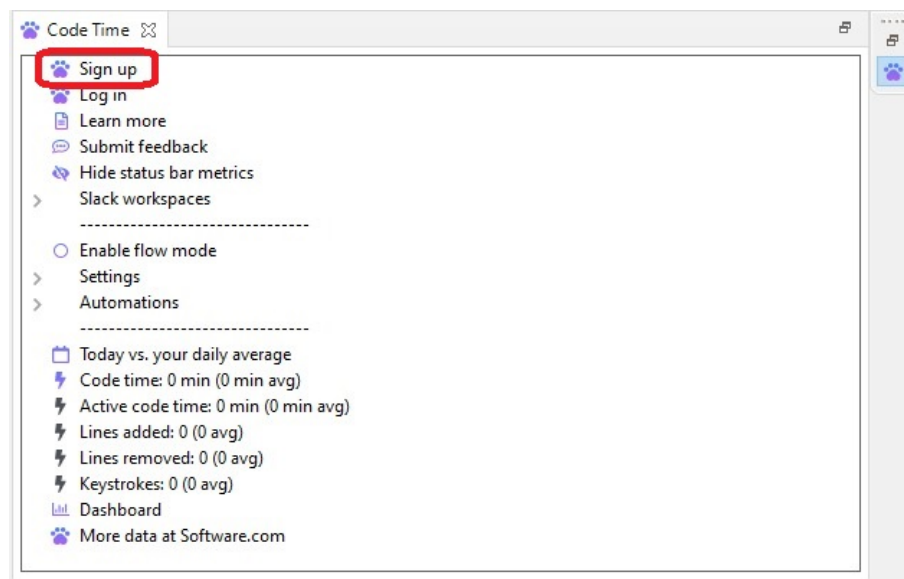


Figure A.2: Code Time Setup Step 2.

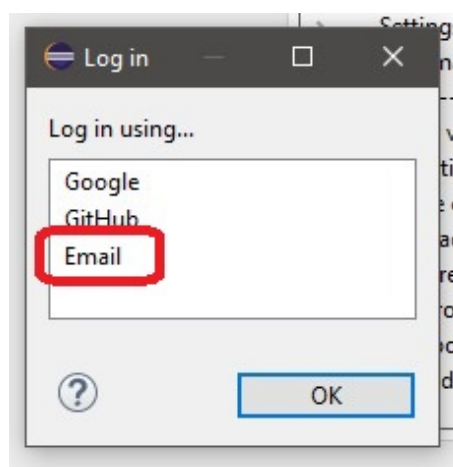


Figure A.3: Code Time Setup Step 3.

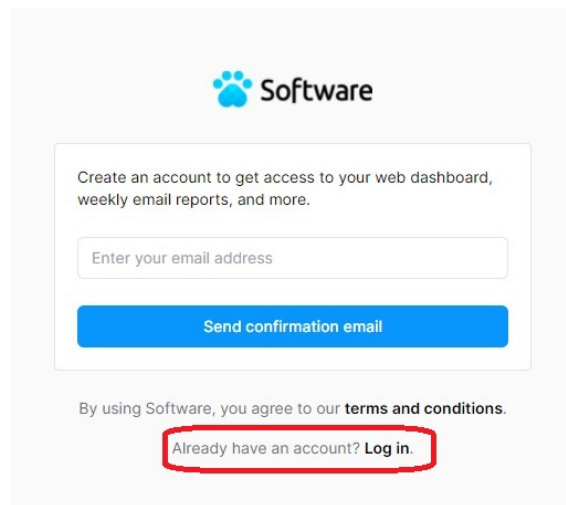


Figure A.4: Code Time Setup Step 4.

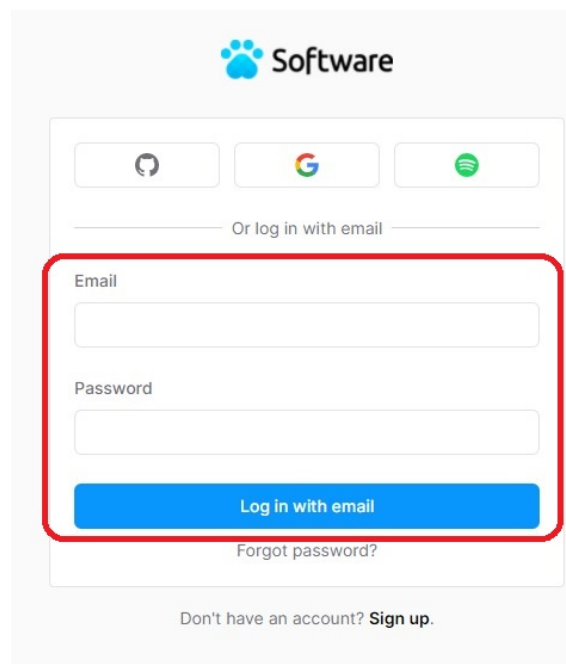


Figure A.5: Code Time Setup Step 5.

Log in with the following credentials, using, in the email, the experimental number you received in your email.

Email: `live.programming.experiment+ExperimentalNumber@gmail.com`

Password: 123456789

For example, if your number is 5:

`live.programming.experiment+5@gmail.com`

The platform is ready to use!

Thank you for your patience!

A.2.2 Usage Guides

A.2.2.1 Usage Guide for Experimental Group A - Liveness Group

Creating and running files

To create a project, click “Create a Project” > “PyDev” > “PyDev Project”. If prompted with the choice to change to the PyDev perspective, choose “No”. Then, for each file one needs to create, right click the project and choose “New > Other... > PyDev > PyDev Module”.

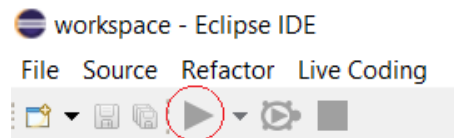


Figure A.6: Group A Instructional Image - Tool Icons to be Used.

Please click the run button (circled above) before starting to program and let the program run continuously while programming. The values of all the variables in each step will appear at the side of the code panel. Click the Turtle button (at the right side of the run button) before starting to program the turtle exercises and let the program run continuously while programming. The result of the turtle drawings will appear at the side of the code panel.

On the non-Windows version of the platform, a known error that can happen, depending on the Python version installed, is the lack of existence of the “tk” library. This can be solved by opening a terminal and running

```
pip install tk
```

A.2.2.2 Usage Guide for Experimental Group B - Control Group

Creating and running files

To create a project, click “Create a Project” > “PyDev” > “PyDev Project”. If prompted with the choice to change to the PyDev perspective, choose “No”. Then, for each file one needs to create, right click the project and choose “New > Other... > PyDev > PyDev Module”. Click “Cancel” in all prompts that show up.

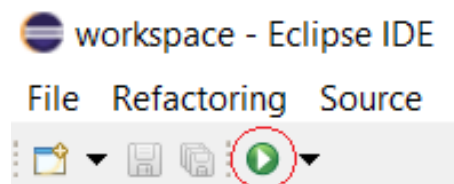


Figure A.7: Group B Instructional Image - Tool Icons to be Used.

To run each file, click the run button circled above and run as “Python Run”.

Turtle files will run in a detached window. For that window to persist after the program finishes running, add

```
turtle.done()
```

to the end of the program.

On the non-Windows version of the platform, a known error that can happen, depending on the Python version installed, is the lack of existence of the “tk” library. This can be solved by opening a terminal and running

```
pip install tk
```

A.3 Usage Guides For Children

A.3.0.1 Usage Guide for Experimental Group A - Liveness Group

During this experiment, you will solve some exercises. You must solve as many as you can, but don't worry if you can't solve them all.

What should you do? On the top left corner of your screen, you have a menu that looks like this:

(see Figure A.6)

When you start writing your programs, you must click the button with the arrow that is circled red in the image above (Figure A.6). On the right side of the document where you are writing a panel will appear. In that panel, the results of the programs that you are making should appear as you work. You can use that information to guide your work.

A.3.0.2 Usage Guide for Experimental Group B - Control Group

Creating and running files

During this experiment, you will solve some exercises. You must solve as many as you can, but don't worry if you can't solve them all.

What should you do? On the top left corner of your screen, you have a menu that looks like this:

(see Figure A.7)

To run your programs and see your results, you must click the green button with the arrow, that is circled red in the image. Under the document where you are writing, there is a panel called "Console", that looks like the image below (Figure A.8) where your results will show up. You can run your programs as many times as you want.

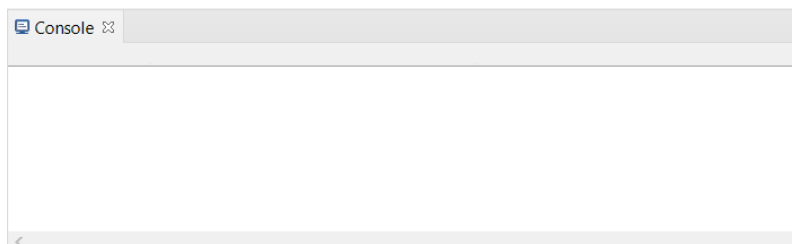


Figure A.8: Group B Console.

Appendix B

Moodle Python Test

B.1 Moodle Python Test for Experiment Participants

1. Implement a function **med(list)** that calculates the average of the values in a list. If the list is empty, the function **med** should return 0.
2. Implement a function **reps(list)** that checks if there are repeated values in a list. It should return True if there are repeated values, False otherwise.
3. Implement a function **fib(n)** that returns the n-th element of the Fibonacci sequence.

Definition: The Fibonacci sequence is a sequence in which the two first terms are 0 and 1 and all subsequent terms are the sum of the two previous terms.

Example:

$\text{fib}(0) = 0$

$\text{fib}(1) = 1$

$\text{fib}(2) = 0 + 1 = 1$

$\text{fib}(3) = 1 + 1 = 2$

4. Given a list of numbers, implement a function **even(list)** that returns a new list with all the even elements belonging to the list passed as argument, in the same order in which they appeared originally. If there are none, return an empty list.

Example:

$\text{even}([1,2,3,4])$ returns $[2,4]$

5. Given a string, implement a function **anagram(string)** that returns True if the string is the anagram of a palindrome, False otherwise (for alphabet a-z).

Definitions:

A **palindrome** is a word that it's read the same way from right to left and from left to right.

An **anagram** is a word that can originate another word when its letters are rearranged.

Examples:

`anagram(aabbccd)` = returns true because `abcdcba` is an anagram of `aabbccd` and it is also a palindrome.

INFORMATION

Turtle is a Python library that allows one to create drawings and geometrical shapes.

To start each exercise, in the beginning of each script there must be the set of instructions presented below, in which the first instruction imports the library's functionalities and the second instruction creates the "pen" that allows one to draw (here named *pen*):

```
import turtle
```

```
pen = turtle.Turtle()
```

After initializing the pen, the drawing is made as it moves.

The essential commands to move it are listed below:

- **Move X units forward:**

```
pen.forward(X)
```

- **Move X units backwards:**

```
pen.backwards(X)
```

- **Rotate X degrees to the left:**

```
pen.left(X)
```

- **Rotate X degrees to the right:**

```
pen.right(X)
```

There are three additional commands that allow for more complex designs and to add functionalities to the pen:

- **Changing the color** with which you draw to color X. The color name must be written between quotes (for example: "Green", "Blue", "Yellow", "Pink", "Red"). When a color isn't chosen, the drawings will be made in black.

```
pen.color("X")
```

- **"Lift"** the pen from the "paper", so when it moves it does not draw, and **"put down"** the pen again, so one can continue drawing. This allows for gaps in the drawing.

```
pen.penup()
```

```
pen.pendown()
```

- **Transport** the pen to the X and Y coordinates.
`pen.goto(X, Y)`
6. Draw a **line segment** 50 units long.
 7. Draw a **square** with 40 units of side.
 8. Draw a **rectangle**, 20 units tall and 40 units wide.
 9. Draw a **hexagon** (internal angles are 120°) with 20 units of side.
 10. Draw a **pentagon** (internal angles are 108°) with 20 units of side.
 11. Draw **two horizontal line segments**, separated by 20 units. The left segment must be 10 units long, and the right segment must be 40 units long.
 12. Draw **two squares**, side by side, separated by 20 units. The left should have a 40 units' side and the right one should have a 20 units' side.
 13. Draw the **word "HELLO"**, all in capital letters. Each letter should be 70 units tall and 30 units wide. All letters must be separated by 20 units.

Appendix C

ENSICO Master Teachers' Survey

C.1 Python Live Programming Platform Review

This form is to be filled after experimenting the Eclipse Distribution with Live Programming in Python.

C.1.1 The Platform's Influence on the teaching of programming at schools

This part of the questionnaire is to be answered based on the experimentation of the platform.

1. The Platform's usage is intuitive and simple.

	1	2	3	4	5	
Strongly Disagree	☐	☐	☐	☐	☐	Strongly Agree

2. The Live Programming feature makes programming in Python simpler for students.

	1	2	3	4	5	
Strongly Disagree	☐	☐	☐	☐	☐	Strongly Agree

3. The Live Programming feature makes programming in Python more entertaining for students.

	1	2	3	4	5	
Strongly Disagree	☐	☐	☐	☐	☐	Strongly Agree

4. This platform will help teachers in teaching programming in class.

	1	2	3	4	5	
Strongly Disagree	☐	☐	☐	☐	☐	Strongly Agree

5. This platform will help in better evaluating the students' progress and development in this subject.

	1	2	3	4	5	
Strongly Disagree	☐	☐	☐	☐	☐	Strongly Agree

6. Do you feel like the Live Programming feature is an asset in teaching programming to young children? Why?

C.1.2 Development of Computational Thinking and its Advantages

This part of the questionnaire aims to understand the importance of Computational Thinking in the students' lives.

1. Do you believe that Computational Thinking is a skill that can help students achieve better results in STEM school subjects?
2. Do you believe that Computational Thinking is a skill that can help students achieve better results in other school subjects?
3. Do you believe that Computational Thinking is a skill that can help students to face real-life problems in their future?
4. Do you feel like the developed platform with Live Programming features might be better or worse than other available platforms (e.g.: VSCode, Jupyter Notebooks, Scratch, Tynker, Eclipse) for the development of Computational Thinking in school children? In what way?

C.1.3 Final Remarks

1. If there are any suggestions you would like to make about the Platform, its usage and features, please describe them below.

Appendix D

Experimental Results

D.1 First Experiment Results

Volunteer Data

Table D.1: Distribution of the Volunteers by College Year.

College Year	Liveness Group	Control Group	Total
1st	1	2	3
2nd	4	0	4
3rd	1	2	3
4th	2	5	7
5th	7	5	12

Participants Data

Table D.2: Distribution of the Participants by College Year.

College Year	Liveness Group	Control Group	Total
1st	1	2	3
2nd	4	0	4
3rd	1	2	3
4th	2	4	6
5th	6	5	11

Distribution of the Participants by College Year

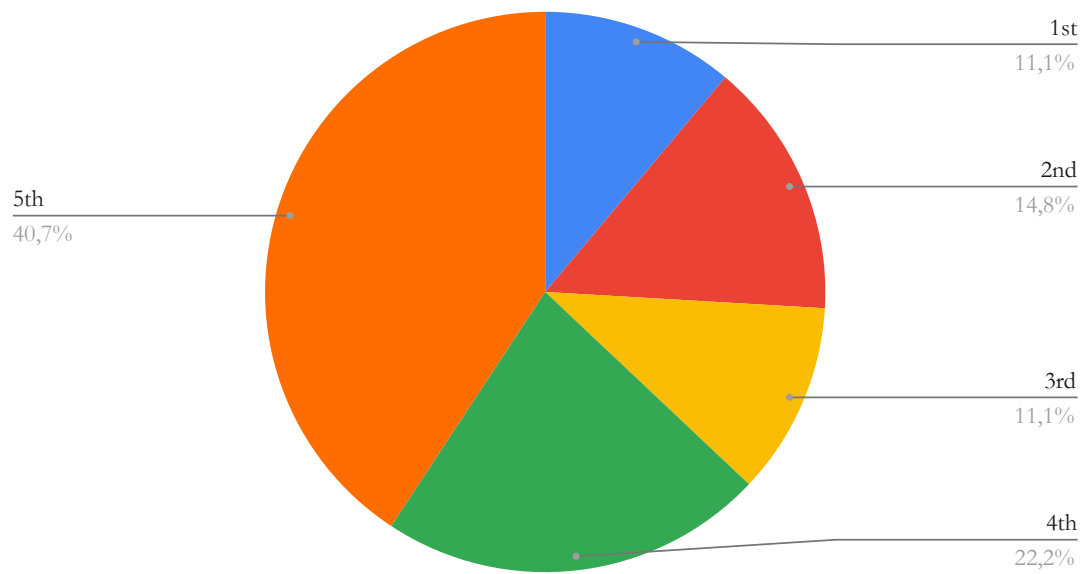


Figure D.1: Distribution of the Participants by College Year.

Distribution of Participants by College Year in Both Experimental Groups

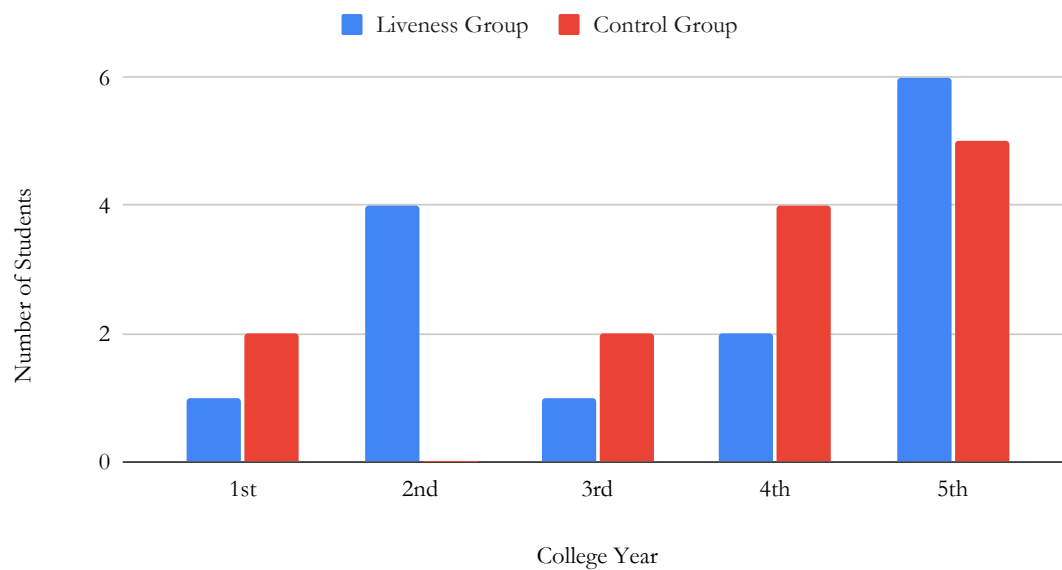


Figure D.2: Distribution of Participants by College Year in Both Experimental Groups.

Distribution of Participants by College Year in Liveness Group

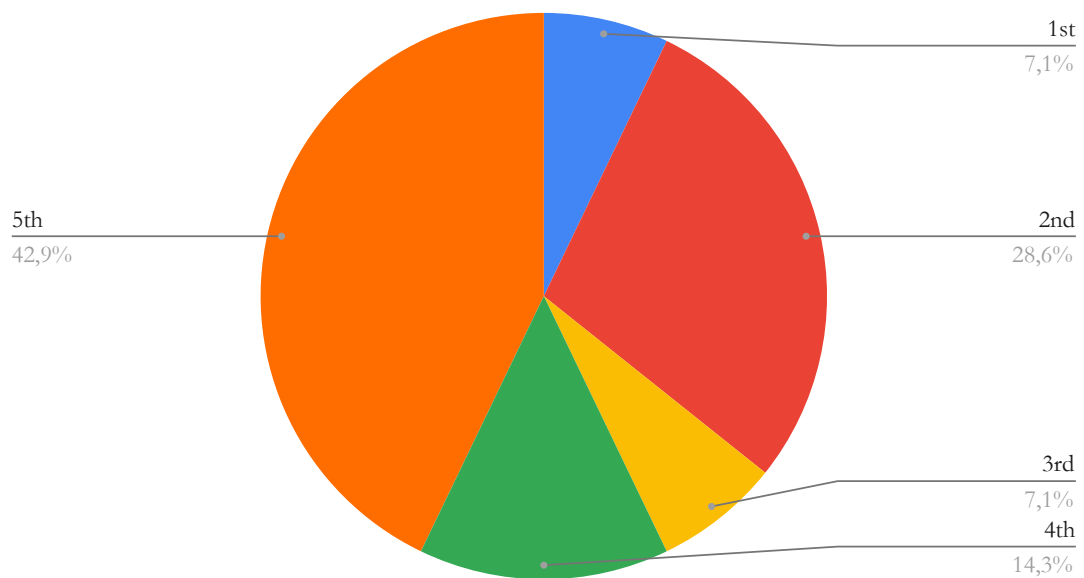


Figure D.3: Distribution of Participants by College Year in the Liveness Group.

Distribution of Participants by College Year in Control Group

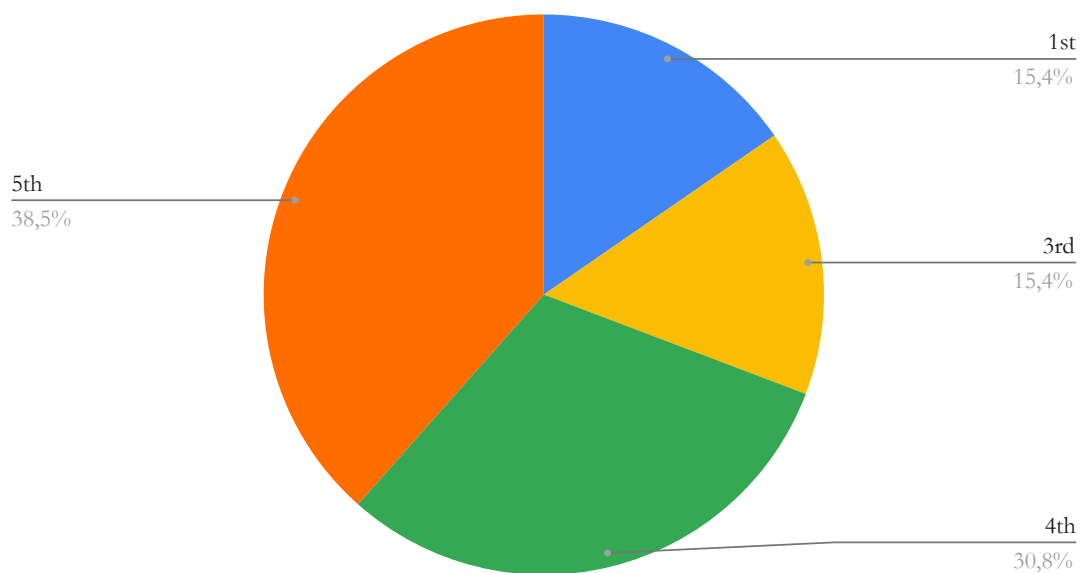


Figure D.4: Distribution of Participants by College Year in the Control Group.

Table D.5: Absolute Frequency of Number of Exercises Solved in Each Group.

Number of Exercises Solved	Liveness Group	Control Group
0	2	2
0,5	0	1
1	0	0
1,5	0	1
2	2	1
2,5	0	1
3	3	0
3,5	2	0
4	1	4
4,5	1	0
5	0	1
5,5	1	0
6	1	0
6,5	0	1
7	0	1
7,5	0	0
8	0	0
8,5	0	0
9	0	0
9,5	0	0
10	0	0
10,5	0	0
11	0	0
11,5	0	0
12	1	0
12,5	0	0
13	0	0

Absolute Frequency of Number of Exercises Solved in Each Group

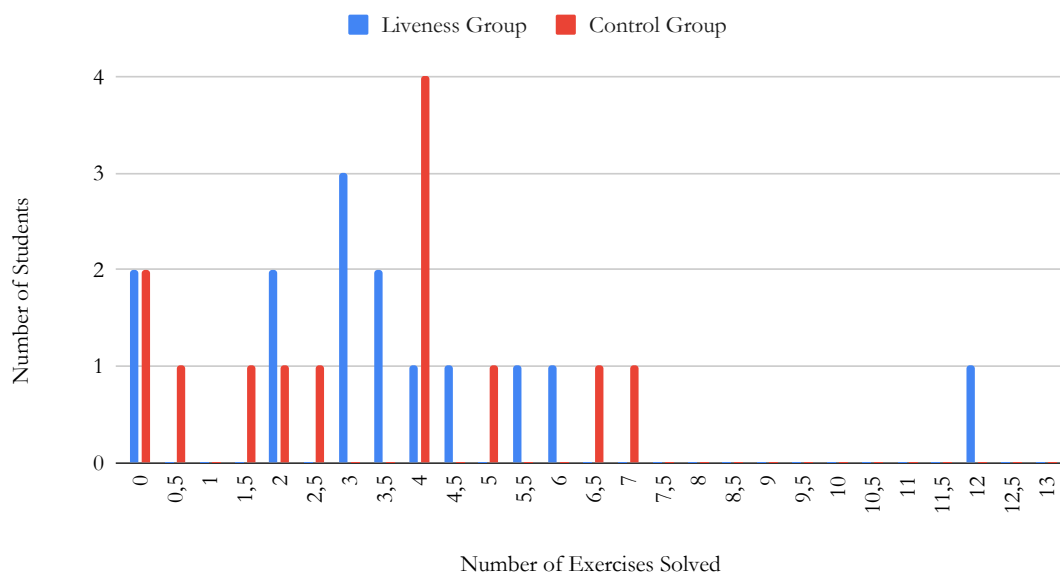


Figure D.5: Absolute Frequency of Number of Exercises Solved in Each Group.

Table D.6: Relative Frequency of Number of Exercises Solved in Each Group.

Number of Exercises Solved	Liveness Group	Control Group
0	0,143	0,154
0,5	0,000	0,077
1	0,000	0,000
1,5	0,000	0,077
2	0,143	0,077
2,5	0,000	0,077
3	0,214	0,000
3,5	0,143	0,000
4	0,071	0,308
4,5	0,071	0,000
5	0,000	0,077
5,5	0,071	0,000
6	0,071	0,000
6,5	0,000	0,077
7	0,000	0,077
7,5	0,000	0,000
8	0,000	0,000
8,5	0,000	0,000
9	0,000	0,000
9,5	0,000	0,000
10	0,000	0,000
10,5	0,000	0,000
11	0,000	0,000
11,5	0,000	0,000
12	0,071	0,000
12,5	0,000	0,000
13	0,000	0,000

Table D.7: Absolute Frequency of Number of Students to Solve Each Exercise in Each Group.

Exercise Number	Liveness Group	Control Group
1	10	11
2	10	9
3	10	8
4	10	8
5	1	2
6	4	2
7	3	2
8	2	2
9	1	0
10	1	0
11	1	0
12	1	0
13	0	0

Absolute Frequency of Number of Students to (Fully or Partially) Solve Each Exercise in Each Group

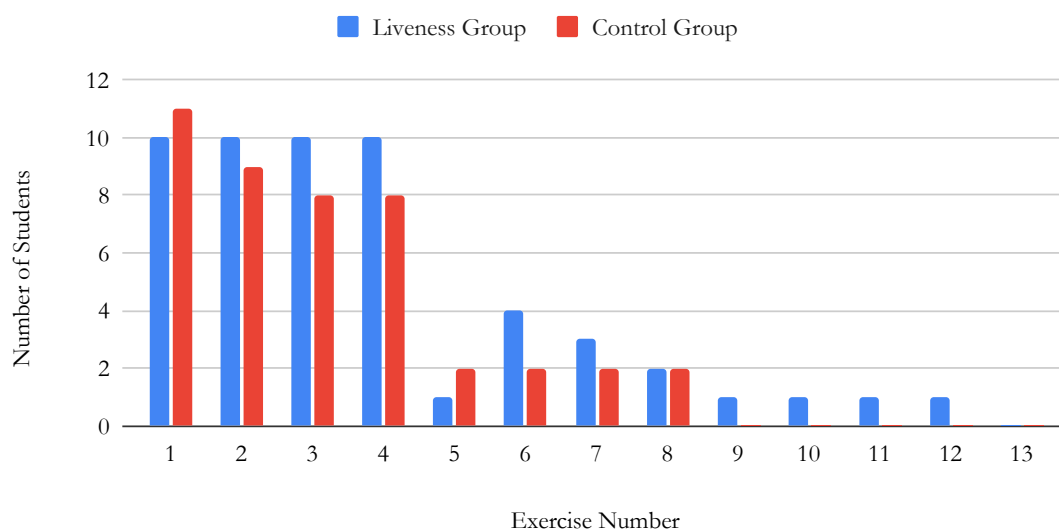


Figure D.6: Absolute Frequency of Number of Students to (Fully or Partially) Solve Each Exercise in Each Group.

Table D.8: Relative Frequency of Number of Students to Solve Each Exercise in Each Group.

Exercise Number	Liveness Group	Control Group
1	0,714	0,846
2	0,714	0,692
3	0,714	0,615
4	0,714	0,615
5	0,071	0,154
6	0,286	0,154
7	0,214	0,154
8	0,143	0,154
9	0,071	0,000
10	0,071	0,000
11	0,071	0,000
12	0,071	0,000
13	0,000	0,000

D.2 Expert Evaluation Answers

D.2.1 Master Teacher Luís Neves - Questionnaire Answers

1. **The Platform's usage is intuitive and simple.**

3

2. **The Live Programming feature makes programming in Python simpler for students.**

4

3. **The Live Programming feature makes programming in Python more entertaining for students.**

5

4. **This platform will help teachers in teaching programming in class.**

4

5. **This platform will help in better evaluating the students' progress and development in this subject.**

4

6. **Do you feel like the Live Programming feature is an asset in teaching programming to young children? Why?**

Yes, I do. Because they will be experiencing a permanent "dialogue" with the machine (computer). And such a permanent "dialogue" will allow them to better understand how programs and algorithms really work.

7. **Do you believe that Computational Thinking is a skill that can help students achieve better results in STEM school subjects?**

Absolutely.

8. **Do you believe that Computational Thinking is a skill that can help students achieve better results in other school subjects?**

Absolutely.

9. **Do you believe that Computational Thinking is a skill that can help students to face real-life problems in their future?**

Absolutely.

10. **Do you feel like the developed platform with Live Programming features might be better or worse than other available platforms (e.g.: VSCode, Jupyter Notebooks, Scratch, Tynker, Eclipse) for the development of Computational Thinking in school children? In what way?**

I feel that the developed platform is not better or worse than others. It is an important platform that should be used at specific stages of the k12 CT learning process.

- 11. If there are any suggestions you would like to make about the Platform, its usage and features, please describe them below.**

The visualisation of the execution steps must be improved in order to be more appealing to young students. The platform should also be available as a web application. See, for example, the pythontutor <http://pythontutor.com/visualize.html#mode=edit>

D.2.2 Master Teacher Rui Grandão - Questionnaire Answers

- 1. The Platform's usage is intuitive and simple.**

4

- 2. The Live Programming feature makes programming in Python simpler for students.**

5

- 3. The Live Programming feature makes programming in Python more entertaining for students.**

5

- 4. This platform will help teachers in teaching programming in class.**

5

- 5. This platform will help in better evaluating the students' progress and development in this subject.**

4

- 6. Do you feel like the Live Programming feature is an asset in teaching programming to young children? Why?**

I believe Live Programming is a feature that will make teaching programming to young children more interactive and entertaining.

- 7. Do you believe that Computational Thinking is a skill that can help students achieve better results in STEM school subjects?**

Yes, computational thinking is a great skill to teach students because it perfects logical thinking. Having a better logical thinking makes the student perform better in other school subjects as well.

- 8. Do you believe that Computational Thinking is a skill that can help students achieve better results in other school subjects?**

I completely believe it does. Computational thinking perfects logical thinking and that is very important and can be used not only in maths, but in portuguese and other subjects. Having a better computational thinking makes the student better overall.

9. Do you believe that Computational Thinking is a skill that can help students to face real-life problems in their future?

Yes, with computational thinking the student can better understand how the world works and can easily face real life problems.

10. Do you feel like the developed platform with Live Programming features might be better or worse than other available platforms (e.g.: VSCode, Jupyter Notebooks, Scratch, Tynker, Eclipse) for the development of Computational Thinking in school children? In what way?

The live programming features are more intuitive for students, so i believe is better.

11. If there are any suggestions you would like to make about the Platform, its usage and features, please describe them below.

I would suggest make it more more attrative and intuitive.

D.2.3 Master Teacher Inês Guimarães - Questionnaire Answers

1. The Platform's usage is intuitive and simple.

5

2. The Live Programming feature makes programming in Python simpler for students.

5

3. The Live Programming feature makes programming in Python more entertaining for students.

4

4. This platform will help teachers in teaching programming in class.

5

5. This platform will help in better evaluating the students' progress and development in this subject.

4

6. Do you feel like the Live Programming feature is an asset in teaching programming to young children? Why?

Yes, because it helps them visualize the code they have written, showing them what they are doing right or wrong, and even hinting at where the mistakes might be. This clearly promotes a better understanding of specific computer programs and programming paradigms themselves.

7. Do you believe that Computational Thinking is a skill that can help students achieve better results in STEM school subjects?

Yes.

8. Do you believe that Computational Thinking is a skill that can help students achieve better results in other school subjects?

Yes.

9. Do you believe that Computational Thinking is a skill that can help students to face real-life problems in their future?

Yes.

10. Do you feel like the developed platform with Live Programming features might be better or worse than other available platforms (e.g.: VSCode, Jupyter Notebooks, Scratch, Tynker, Eclipse) for the development of Computational Thinking in school children? In what way?

In my opinion, the developed platform is better than the ones mentioned above, since it definitely feels more "live" and engaging. By being able to visualize each intermediate step of a program in a very clear and intuitive way, it promotes a sharper understanding of what is going on.

11. If there are any suggestions you would like to make about the Platform, its usage and features, please describe them below.

-

D.2.4 Master Teacher Liliana Monteiro - Questionnaire Answers

1. The Platform's usage is intuitive and simple.

5

2. The Live Programming feature makes programming in Python simpler for students.

5

3. The Live Programming feature makes programming in Python more entertaining for students.

5

4. **This platform will help teachers in teaching programming in class.**

4

5. **This platform will help in better evaluating the students' progress and development in this subject.**

3

6. **Do you feel like the Live Programming feature is an asset in teaching programming to young children? Why?**

I do, since it help them what would otherwise be hard to visualize

7. **Do you believe that Computational Thinking is a skill that can help students achieve better results in STEM school subjects?**

Yes

8. **Do you believe that Computational Thinking is a skill that can help students achieve better results in other school subjects?**

Yes

9. **Do you believe that Computational Thinking is a skill that can help students to face real-life problems in their future?**

Yes

10. **Do you feel like the developed platform with Live Programming features might be better or worse than other available platforms (e.g.: VSCode, Jupyter Notebooks, Scratch, Tynker, Eclipse) for the development of Computational Thinking in school children? In what way?**

It's different. Jupyter isn't exactly live programming for example and Tynker is a good intro tool but not really helping with actual code per se.

11. **If there are any suggestions you would like to make about the Platform, its usage and features, please describe them below.**

-

