Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

Design and Implementation of a Frame-Based Python Editor for Novice Programmers

Anna Enikő Illési

Supervisor: Glenn Strong

May 8, 2022

A dissertation submitted in partial fulfilment
of the requirements for the degree of
MAI (Computer and Electrical Engineering)

Declaration

I hereby declare that this dissertation is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at `http://www.tcd.ie/calendar`.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at `http://tcd-ie.libguides.com/plagiarism/ready-steady-write`.

Signed:  Date: May 8, 2022

Abstract

Programming is difficult to learn, even at the early stages. Beginner programmers need to master abstract algorithmic thinking and problem-solving. The complex syntax, the extensive vocabulary of programming languages and semantic issues make it difficult for novices to acquire the necessary programming skill set. Although block-based tools support the first steps, transitioning from these to text-based programming is challenging. This paper presents an intermediate frame-based tool that can facilitate the transition from block-based editing to frame-based languages, mainly from Scratch to Python. This frame-based Python editor is developed as an addition to the Pytch application, developed at Trinity College Dublin. The editor was designed to combine aspects of block-based and frame-based editing to allow users to discover the programming mindset and improve problem solving while familiarising themselves with correct Python syntax. A functional frame-based editor prototype has been developed and integrated with the Pytch environment. The frame-based Python editor's preliminary evaluation tools, a workshop and a survey have been created. The study finds that the developed frame-based Python editor is likely successful at easing the transition from Scratch to Python, however further evaluation is needed. The prototype developed in this project can be used as a basis for further development, due to its modular and scalable structure.

Acknowledgements

I would like to express my gratitude and appreciation for my supervisor, Glenn Strong whose guidance, unwavering support, and encouragement has been invaluable throughout this study.

I would also like to thank Ben North, for his help, guidance and exceptionally insightful advice.

Finally, thank you to my family and Nas for always supporting and believing in me.

Contents

List of Figures

List of Tables

Nomenclature

FBPE	Frame-Based Python Editor
AST	Abstract Syntax Tree
TS	Type Script
IDE	Integrated Development Environment

1 Introduction

Block-based languages are becoming increasingly popular in early programming education. They provide a beginner-friendly, visual and interactive environment for learning programming concepts and improving abstract thinking and problem-solving skills. Coding tools like Scratch [1], Alice [2], and Snap! are becoming increasingly popular and are widely used in schools and programming courses around the world.

However, text-based languages remain the industry standard, so students eventually have to transition to text-based languages as they continue their education. By transitioning to Python, Java or JavaScript, for example, students gain a very powerful tool to solve a wide range of problems that are way beyond the scope of block-based editors. The transition from block-based to text-based programming environments can present a challenge. Students at this stage are faced with complex syntax, confusing vocabulary and cryptic error messages.

While many make this transition with currently available training materials, it is made difficult by the vast differences between the two languages. The most common errors for beginners are misunderstanding syntax, type errors, and other semantic errors. While these are mostly eliminated by block-based editors, they present a new challenge to those adapting to text-based editors.

Frame-based editors attempt to bridge this gap with a development environment that combines characteristics of block-based and text-based editors. The two most popular examples of frame-based editors are the BlueJ and Greenfoot [3] editors, which use Java-like syntax. However, there is currently no frame-based editor that focuses on the transition to Python specifically.

The goal of the study is the design, implementation and preliminary evaluation of a syntax-driven editor for teaching Python, which will be referred to as FBPE. The editor was proposed as an addition to the Pytch system [4], a web-based educational environment developed at Trinity College Dublin. Through the integration of the editor into the larger Pytch system, the users can enjoy the benefits of both, as Pytch seamlessly compiles the program and provides a visual output, as well as tutorials and training material.

The FBPE serves as an introduction to Python syntax while maintaining the benefits of block-based editors. The FBPE is designed primarily for beginners transitioning from Scratch to Python that have developed a good understanding of programming logic and constructs such as variables, loops, and functions, but are unfamiliar with Python syntax.

Chapter 1 of this paper introduced the context of early programming education, block-based and frame-based programming languages. The research problem has been identified as the difficulty of transitioning from block-based to text-based languages, specifically from Scratch to Python. The objectives and the ways the FBPE could tackle this problem have also been discussed.

In chapter 2, the challenges students face in the early stages of learning programming will be discussed in detail. The concept of frame-based editing will be defined with respect to previous works and the purposes of this paper. The current literature will be analysed to identify current attempts at bridging the gap between block-based and text-based programming languages.

The design of the frame-based python editor is discussed in chapter 3. The design decisions and the alternatives considered are also described, along with the theoretical limitations of the final design.

In chapter 4, the technical implementation is described in detail, including the technologies, framework, architecture, and challenges faced during the implementation.

Chapter 5 covers the evaluation methods proposed for the prototype. This includes a detailed design of a workshop, survey and technical evaluation methods of the quality and robustness of the software.

Finally, chapter 6 concludes the paper, providing an overview of the frame-based editor implementation in the context of the research problem. This chapter also contains suggestions for future work and alternative applications.

2 Background and Motivation

This chapter will discuss the theoretical background considered when developing the FBPE proposed by this dissertation. Firstly, it is crucial to understand the experience of novice programmers, their learning outcomes and their difficulties in achieving them. Furthermore, both the skills gained from an early education with Scratch and the skills needed to progress with more advanced text-based programming tools will be considered. To better understand the transition between the block-based and text-based environments, the characteristics of each will be analysed. The research on currently available transitional tools will be discussed, including the Pytch application which has a critical influence on the FBPE's design, implementation, and perception. Finally, the principles of accessible software design features will be reviewed.

2.1 Early Programming education with Block-Based Editors

Block-based editors are commonly used to teach beginners the fundamentals of programming. Weintrop et al. [5] defines block-based editors as a subset of the structured editors that "make the atomic unit of composition a node in the Abstract Syntax Tree (AST) of the program, as opposed to a smaller element (i.e., a character) or a larger element (like a fully formed functional unit)". The most popular block-based editors include Scratch [1], Alice [2], and other Blockly-based tools [6].

Block-based are designed to be easier for beginners to get started with programming. They employ more natural, language-like constructs. There is also a clear, graphical representation of scopes. Users do not need to memorise complex syntax structures or keep track of variable types. Additionally, they allow the user to "tinker" i.e., explore the tools available and get instant feedback in a highly graphical environment [1] [7]. An example of a simple program written in Scratch can be seen in Fig. 2.1. The high level of interactivity is proven to be beneficial [8] [5], especially for beginners. The target audience of Block-based editors is school-aged children. Scratch for example is intended primarily for ages 8 to 16.

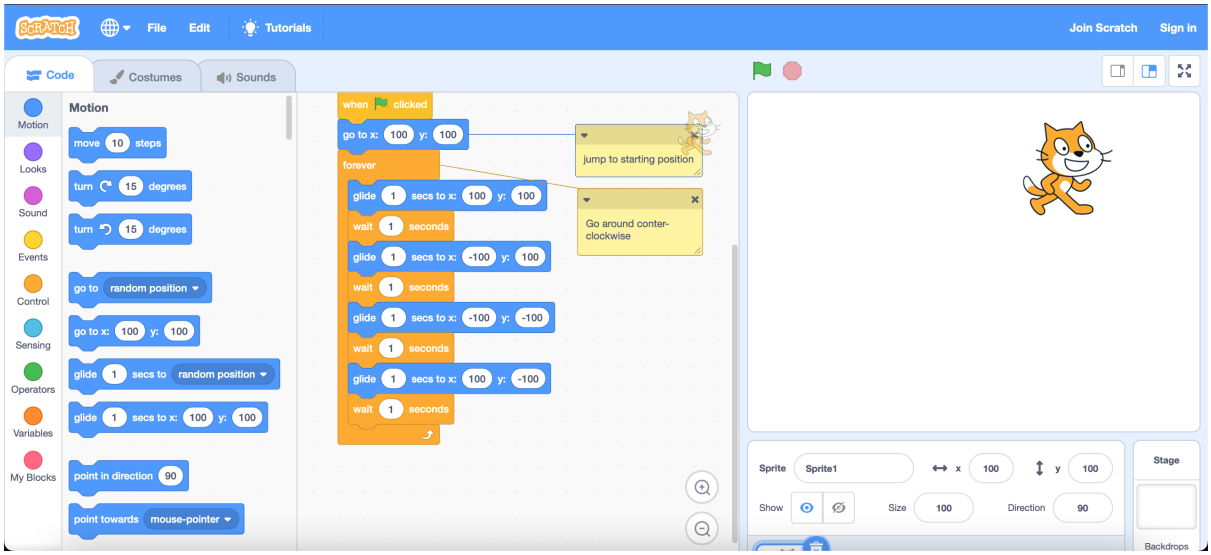


Figure 2.1: Example Scratch Program

Block-based languages, however, have considerable limitations, especially for more experienced users, who tend to prefer writing code with text-based languages [9]. Reporting errors in block-based languages can be a challenge [10]. Especially for novice users and children, feedback needs to be descriptive and specific, which would require very complex feedback systems.

Block-based languages are perceived to be more powerful and less time-consuming than block-based languages. The software industry does not make use of block-based languages, hence they are not specialised for professional use. They offer limited functionality (to avoid overwhelming beginners). Block-based code cannot be efficiently integrated into larger applications and the lack of source control makes them unreasonable to be employed in large-scale projects. Hence, text-based software remains the industry standard.

Thus, novices, using block-based editors eventually have to learn text-based programming languages. Whether is it beneficial for complete beginners to start out with block-based languages, might not make a significant difference [11]. However, precisely because block-based environments are so successful at guiding users, the transition to text-based presents a challenge [12].

2.2 Further Programming Education

A source code editor is traditionally a text editor that is designed specifically for writing software. Nowadays, there are a range of Integrated Development Environments (IDE) available that have a text editor at their core, such as Pycharm, Eclipse or Visual Studio. They offer development tools such as syntax-highlighting [13], in-line error warnings, hover help, debugging [14], automatic formatting, and suggestions to help improve workflow and

code readability. Most programming environments also include error messages on failed compilation. The usefulness of current error messages is limited [15]. As mentioned earlier, text-based editors remain the industry standard for software development and are a key requirement in later stages of computer science education.

While text-based programming environments are unquestionably powerful tools for experienced users, there is a steep learning curve for novices. The common problems beginners face are well-researched. To make things worse, there is often a disagreement [16] [17] between students and teachers about the most important problems. The connection between students' eye movements while reading and analysing code and their abilities is a promising new area [18] [19] [20], however, this is too early to base educational tools on their findings.

Altadmri and Brown [21] identified the 13 most common types of student mistakes that fall into the following categories:

- Misunderstanding (or Forgetting) Syntax,
- Type errors,
- Other Semantic Errors.

Altadmri and Brown [21] conducted their research on Java code, so some of the specific errors are not relevant to Python syntax. Likewise, there might be other common errors made by Python users that are not considered. However, the three general categories can give a general impression of the type of issues.

Mistakes related to "Misunderstanding (or forgetting) syntax" are confusing similar operators (such as '=' with '==', '&' with '&&', '|' with '||', '<' with '>'), unbalanced parentheses, using keywords as method or variable names, and forgetting parentheses after a method call;

Other mistakes related to Misunderstanding (or Forgetting) Syntax listed by Altadmri and Brown [21] are not relevant to this paper, as Python syntax eliminates certain syntactic requirements, such as semicolons at the end of statements and in for loop conditions.

Type Errors relate to invoking methods with wrong arguments or incompatible types (Number, String, List, Tuple, Dictionary) between method return and the type of variable that the value is assigned to.

Even though Python variable types are not explicitly declared at initialisation, Python is still a strongly typed language and the above errors apply to Python code.

Other Semantic Errors include calling a method with a non-void return and its return discarded, control flow can reach the end of the non-void method without returning, calling

non-static methods and a class not fulfilling all requirements of the interface it is based on.

McCall and Köllig [22] have proposed a manual classification of the errors as opposed to diagnostic messages. As a result of the analysis of student errors, a more accurate error category hierarchy was developed. The categories are the following:

- Variable: Incorrect attempt to use variable,
- Variable: Incorrect variable declaration,
- Method: Incorrect method call,
- Method: Incorrect method declaration,
- Constructor: Incorrect constructor call,
- Constructor: Incorrect constructor declaration,
- Incorrect attempted use of class or type,
- Semantic error,
- Simple syntactical error,
- Statement outside method/block,
- Uncategorized.

After analysing the differences, similarities, and other connections between the two student error categorisation methods, the following method was used to define the relationship:

- If the categories are directly related, they are marked as a primary (P).
- If the categories are indirectly related, or the relationship is not particularly significant, they are marked as a secondary (S).
- If student mistake categories are not or barely related, they are marked as not related (-).

The mapping of the most common types of student mistake categories can be seen in table 2.1.

Writing syntactically correct code for students is only part of the challenge. Students need to develop a programming mindset that involves abstraction, algorithmic thinking, decomposition, evaluation and generalisation [8].

Table 2.1: Mapping the most common types of student mistake categories

	Misunderstanding (or forgetting) syntax	Type errors	Other semantic errors
Variable: Incorrect attempt to use variable	P	P	S
Variable: Incorrect variable declaration	P	P	S
Method: Incorrect method call	P	P	S
Method: Incorrect method declaration	P	P	S
Constructor: Incorrect constructor call	P	P	S
Constructor: Incorrect constructor declaration	P	P	S
Incorrect attempted use of class or type	P	P	S
Semantic error	–	–	P
Simple syntactical error	P	S	–
Statement outside method/block	P	S	–
Uncategorised	S	S	P

2.3 Frame-Based Editing

Frame-based editors offers a potential solution to aid the transition from block-based programming to text-based languages. Frame-based programming can also be used for beginners to introduce algorithmic thinking and reduce the potential student errors of text-based approaches.

The Greenfoot project first introduced the concept of frame-based editing. The Greenfoot educational software [3] makes use of Stride, a Java-like language as the base of their editor. Java and Stride likely employ similar thinking patterns, however, there are some differences [23] in some users. Users can input text-based code or insert blocks of text through frames. The editor combines some of the benefits of both text and block-based programming [24] [25]. Students have been reported to spend less time on syntactic corrections and with non-compilable code [25] with the Greenfoot editor compared to a Java editor.

Frame-based editors as described by Kölling et al. [24] combine aspects of locks and text, by maintaining the graphical nature of black-based editors, while providing the flexibility and keyboard entry capabilities. They have discussed frame-based editing as it relates to Stride and suggested that the same principles could be extended to other languages.

For the purposes of this project, frame-based editors are defined as a transitional environment between block and frame-based editors, while maintaining aspects of both. This wider definition offers more freedom when designing the editor, as the aspects adopted from block and text-based editors do not necessarily align with those of Köllig et al. [24].

While the Greenfoot editor offers many great functionalities for programming students, it

might take some time for first-time users to get accustomed to the platform. It is offline too, which requires downloading and installing the packages. The application contains different windows for the 2D visual output of the class codes. For learners, this can enhance object-oriented thinking, as the classes are separate, but can also be confusing when first opening the application.

2.4 Pytch Application

The Pytch [26] project, developed at Trinity College Dublin, aims to provide a tool to support the transition between Scratch and Python. Similarly to Scratch, the Pytch users can work on projects with an immediate graphical output. Currently, the code editor is mainly text-based with references to the familiar Scratch-blocks. An example of a simple code written in Pytch can be seen in Fig. 2.2.

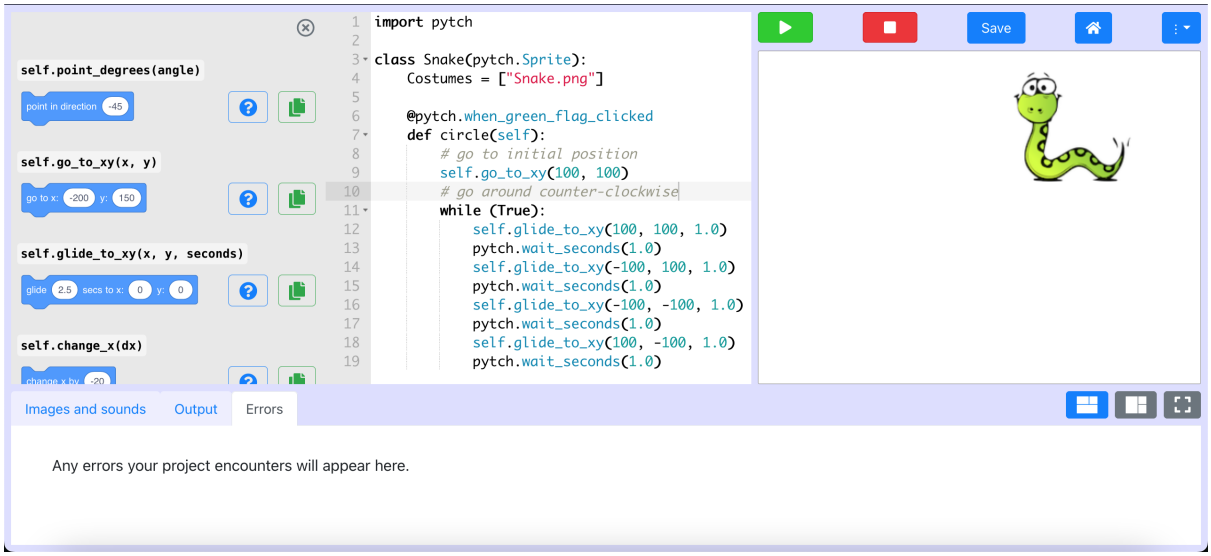


Figure 2.2: Example Pytch Program

One of the goals of the Pytch project is to explore the types of programs users can write autonomously. The development of features is driven by the user's needs and expectations and not the other way around. A block-based editor [26] has been proposed for Pytch before using Blockly [6], however, this project proposes a frame-based specifically without the limitations of Blockly.

The frame-based editor was to be integrated into the Pytch web app, so its technical framework was chosen accordingly. The FBPE had to be compatible with the technical decisions already made in the Pytch implementation.

Pytch, and thus this project, are web-based applications and are primarily based on JavaScript, Type Script and React [27]. These technologies are discussed in further detail in

Chapter 4, as they relate to the technical implementation of the FBPE.

2.4.1 Accessibility

In many software projects, accessibility is a secondary goal, or even an afterthought, something to be considered only after the application is complete. In some cases, it is disregarded completely. This is unfortunate, considering how much software developers would benefit from reaching as wide an audience as possible.

Implementing accessibility features on a finished product can be incredibly (and unnecessarily) expensive, as it sometimes requires changes to low-level components. This is completely avoidable by considering accessibility in the early stages of development.

The Web Accessibility Initiative (WAI) [28] defined a set of principles that contribute to software products being as widely available to users as possible.

Naturally, the best way to evaluate accessibility is with real users, so an emphasis should be placed on conducting workshops, surveys and questionnaires with a diverse selection of participants.

3 Design

This chapter introduces the core functionalities and constraints of the FBPE prototype that needed to be implemented to support the seamless integration with Pytch IDE. The aim was for users to be able to write simple programs and compile and run them in the Pytch environment. The chapter also provides a user interface analysis, including the colour scheme and the layout to support the target audience.

3.1 Core Requirements and Constraints

The motivation behind the project was to explore whether the frame-based approach would be a benefit to the Pytch system. This was to be achieved by designing a frame-based editor that best fit the existing features of Pytch and solved Pytch users' problems. For this reason, the core functionalities of the editor needed to be implemented, such that users could write simple programs, and compile and run them in the Pytch environment. The Pytch IDE currently makes contains a text-based editor, called Ace [29], which the FBPE was designed to replace.

Since this project had limited time and resources available, the most important features had to be determined. They were chosen, such that the prototype would allow users to write functional code, without having to type completely text-based code. Thus, a number of interactive features were selected that allow FBPE users to complete a simple tutorial and a preliminary evaluation could be conducted. The types of statements selected for the FBPE were also influenced by existing Pytch tutorials. These tutorials were already designed for students starting to transition from Scratch to Python, limiting the functionality. Thus, the number of Frame types can also benefit the users, by creating a simpler, more focused user interface.

The ease of use and aesthetic qualities of the editor was also taken into consideration. The editor was to be designed in a way to make it accessible and intuitive for as many users as possible. The design of the FBPE was largely influenced by the Greenfoot editor [24]. In Chapter 2, the key features of frame-based editors were discussed. Like Greenfoot, the FBPE developed in this project combined characteristics of text and block-based editing.

The expected target audience of the FBPE influenced the design. This project targeted users that are familiar with Scratch, but have little experience with text-based Python programming. The target audience of the FBPE was defined as ages 16 and above.

3.2 Design Solutions

The most basic feature to be implemented is the ability for FBPE users to add and delete code frames, later referred to as Frames. Unlike Greenfoot [24], where the whole program can be written through text input only, the FBPE was designed to have more structured editing. The Frames in the FBPE could not be added with keyboard input only, instead, they are added through mouse interactions. This means that the FBPE user could only modify certain parts of the code, thus leaving less room for syntactic errors.

Thus, the concept of the 'Frames Controls' was introduced, in which the Frames types would be available through a set of buttons. Adding a list of buttons with which the FBPE user could add different Frame types, could also serve as a visual reminder for the user of what Frames are available to them. With the frames as implemented in this project, there is no need for the user to memorise the complete syntax for creating for example a class or a for loop as the full code is displayed in the correct Python syntax. This allows the users to focus on the programming logic, while simultaneously familiarising themselves with the correct syntax.

Once Frames were added, they were to be editable, so the user can make useful and unique programs. Sequentially inserting Frames would have been very limiting, so some sort of cursor and re-ordering features were to be included as well.

The FBPE had to fit seamlessly into the Pytch IDE. This was done by replacing the current text-based editor with the FBPE, which determined the amount of space it could take up on the screen.

The FBPE consists of two main parts: the "Code Content Area", where the Frames are displayed and a "Frame Controls Area" that contains all the buttons to add new Frames. In Scratch, the new blocks can be dragged from the right side of the screen to the main code content area. While the drag-and-drop functionality is intuitive and easy to use, it would have introduced too much complexity for the scope of this project.

Instead of the drag-and-drop, frames could be inserted at the click of a button at the cursor location. The idea of a cursor or insertion point was borrowed from text-based editors.

The FBPE also had the opportunity to simplify the task of indentation for the users. In python, code-blocks are separated through indentation, which can be a source of errors. In the FBPE, the nesting of Frames would take care of this.

3.3 Appearance

As mentioned before, the FBPE was planned to be integrated with Pytch. The appearance of the editor was influenced and constrained by the current IDE. The colour scheme was to be in harmony with the Pytch website design, which had a light purple background, blue default buttons and red and green accents. The size of the frames was to be large enough to be easily readable, but small enough for at least 4 to 6 frames to be displayed simultaneously.

Different Layouts were explored through wire-frames and sketches can be seen in Fig. 3.1. The main elements to be placed were the Code Area with scrolling functionality, the Add Frames buttons, and other buttons (edit, save, delete).

In the initial design, the Add Frame buttons were made available by hovering at each frame. Thus, the new frames were inserted after the selected frame. This is shown in the top-left layout. However, this layout proved to be inadequate for a larger number of frame types. Abbreviations could have been used for each frame, but that would have made the navigation less clear and the users would have had to memorise additional information.

The final design, shown in the bottom-left sketch, allows for the clearest navigation and makes the best use of the space. The buttons were placed to the side of the code so that the vertical space for the code area could be maximised, and more frames could be displayed simultaneously. When additional Add Frame buttons are added, a scroll functionality can be added to the buttons' panel.

Figure 3.1: Different Layout Option Sketches for the FBPE

3.4 Design Summary

Based on the analysis above, the FBPE prototype has the following general requirements:

1. The FBPE should be size-limited to fit Pytch, and also it shall meet the compatibility requirements of its code editor.
2. The FBPE should be implemented using web-friendly technologies (JavaScript and HTML).
3. The FBPE should provide a real-time, lightweight interface to minimise lag.
4. The FBPE should be modular, reusable and support the implementation of additional functionalities in the future.

The following functional requirements should be met by the FBPE:

1. The FBPE should provide a frame-based interface which uses both
    - blocks,
    - Python-like text-based elements to support the transition to learning Python programming.
2. The FBPE should implement at least the following frame types to provide a minimum but sufficient s functionality for the developed applications:
    - Comment Frame that realises a non-executed and allows FBPE users to text to facilitate understanding of the code.
    - Simple Frames (without body, containing no other frames) realise the following functionalities:
        - variable assignment,
        - print statement
    - Sprite Specific Frames that realise the following functionalities:
        - glide or move-to,
        - stay,
        - wait.
    - Complex Frames (with body parameters that might contain one or more Frames) that realise the following functionalities:
        - logical comparison (if),

    - looping (for, while).
  - Action Frames that realise the following conditional checking-based action functionalities related to Sprite Specific Frames:
    - Sprite clicked,
    - green-flag clicked,
    - key pressed
3. From the Frames, the FBPE shall compile a Python code, which is
  - human-readable,
  - complete and accurate representation of the Frame-based code,
  - compatible with Pytch's Python compiler.

4 Implementation

This chapter introduces the implemented functionalities of the FBPE. First, it summarises the underlying technologies of the prototype editor development and the relationship between these. Second, the chapter provides an analysis of the main integration points with the Pytch web app. Third, the chapter gives a structured overview of the FBPE, which is based on Frames. Also, the Frame Representation, Frame Kinds, Frame Controls and the Editor Structure are described in detail to illustrate the capabilities of the FBPE prototype. The chapter also provides a demonstration of how the FBPE can generate Python code and the input validation.

4.1 Technologies

Since the FBPE was designed to be a part of the Pytch system, the FBPE was created using similar technologies. This ensured compatibility and made communication with the Pytch development team easier.

Both Pytch and the FBPE are based on JavaScript and HTML. The application is lightweight and can be used without initial setup or installation. It is also supported by modern browsers.

For state management, React, a component-based library was used. The main benefits of React are the concise JSX syntax and the ability to create reusable components. On top of React, the Easy Peasy library made the state management easier, by simplifying the syntax.

All JavaScript files were converted to Type Script [27], a strongly typed language, that compiles to JavaScript. By explicitly declaring types, hidden issues were revealed immediately, making the code base more robust and also easier to read.

Some icons were used as custom buttons and visual elements in the FBPE. These were selected from the Font Awesome Collection included in the React Icons library.

4.2 Pytch Integration

There are two main integration points with the Pytch web app. Firstly, the FBPE has to be rendered as an HTML object inside Pytch and appear on the screen when the user opens a project.

At the second integration point, the Python code generated by the FBPE needs to be passed to the compiler. When the user clicks on the green flag to run the program, their code needs to be converted to Python, compiled and run so that they can see the results. Alternatively, if the code is invalid, the user needs to be notified of the error.

Pytch adopted the concept of Sprites from Scratch, therefore the project inherited this capability. Sprites are an object representation of the actors that appear and move on the screen as a result of the user's program running.

The development process of the FBPE began separately from Pytch, which later had to be integrated. Fig 4.1 shows an early version of the FBPE without Pytch. Though Pytch influenced many of the design decisions and implementation of the FBPE, it can be viewed as a standalone entity.

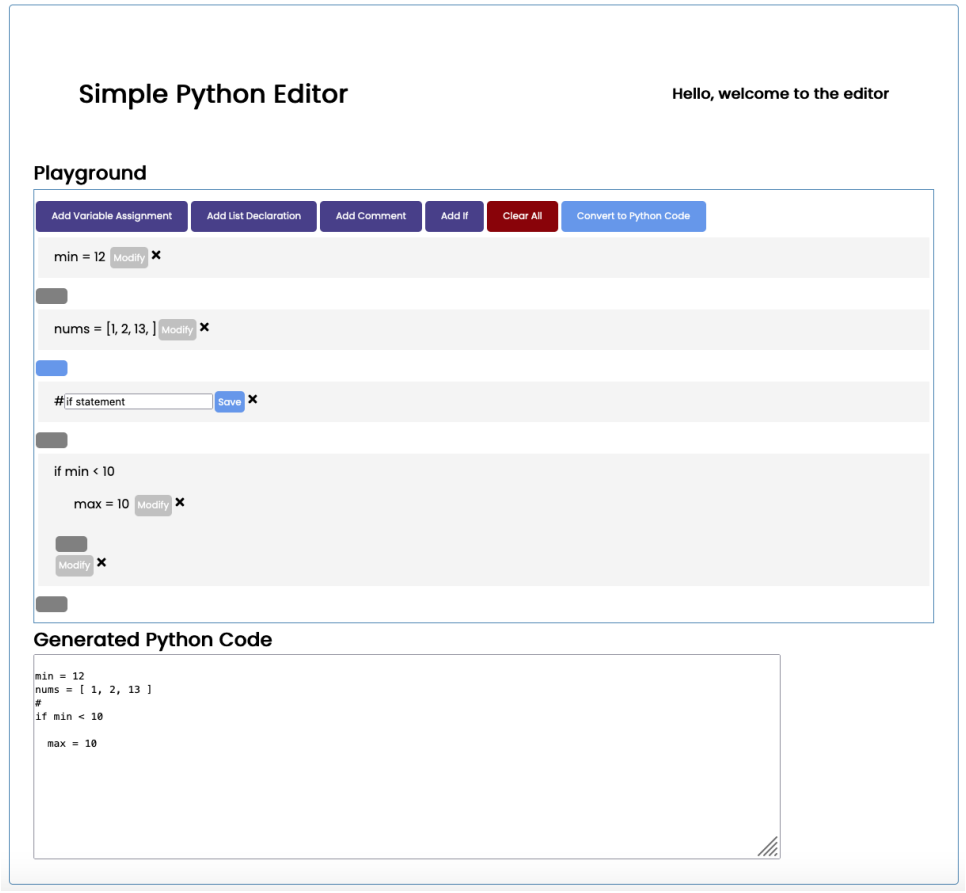


Figure 4.1: Early Version of FBPE Before Pytch Integration

4.3 Frame Representation

The fundamental units of the FBPE are the Frames. A Frame typically represents a statement or one line of Python code. Each Frame object contains information about its kind, value or values, and edit status. A Frame can represent a comment, variable assignment, function call, if/if-else block, while/for loop, function or class definition.

There are a number of functions available for users in the Pytch IDE for handling user interactions and manipulating Sprites. For easier navigation and understandability, these are represented as unique frame kinds in the FBPE. (For example, the Frame type for handing a key press would be completely separate from the Frame type for handling mouse clicks.)

The visual appearance of a Frame in the FBPE is similar to how code would appear in a text-based editor. The text in the Frames read sequentially in correct and complete Python code. However, there is a clear separation between lines of code, as each Frame is outlined. If the Frames are nested, for example, in the body of an if statement, the lower level frames that make up the body of the if statement is inside the higher-level frame (for example, the If Frame).

The Frame is made up of static and dynamic sections. The static part displayed the necessary syntax related to the Frame's functionality. For example, in Comment Frames, the "#" symbol is static, because it is an integral part of a Python comment. The dynamic value of the Frame is the section that the user can edit. This can be the comment text, the variable name and value, the function name or parameters, the class name, and the body of loops of conditional statements.

The Frames contain a Delete and an Edit/Save button respectively. With these, the status of the Frame can be modified from "saved" to "being edited" and vice versa. There are exceptions, as some special Frame types are un-editable. These are discussed in detail in later sections.

4.4 Frame Representation in Type Script

All Frames are based on the same `Frame Base` that contains a unique `id` number and a `kind` property, indicating the function of the Frame in the code. This could indicate that the Frame represents a comment, variable assignment, function call, if/if-else block, while/for loop, function or class definition. The ID is assigned to each new Frame sequentially to avoid duplicate IDs.

For keeping track of the nesting of Frames (and thus the indentation in the final code) a depth property was also added to the `Frame Base`. Similarly, they contain an `isSelected`

boolean property, which is used when rendering the index buttons. These will be discussed in detail in later sections of this chapter. The selected index and the depth of each Frame are stored indirectly in the top level of the FBPE. This leads to some code redundancy, however, it makes the rendering process simpler.

Each Frame kind has a specific base (e.g. `CommentFrameBased`, `StatementFrameBased`). The properties depend on the Frame's `kind`. These properties store the values that can be changed by the user (e.g. `commentText`, `variableName`, `variableValue`). Each Frame Kind and its parameters will be discussed in detail in later sections.

The user-inputted values are all stored as a string, regardless of the type they would be allowed to be in the Python code. For example, in the Wait Frame, the only input is the number of seconds. However, limiting the input to a number would be limiting for the users, who might wish to store the number of seconds in a variable. When a new Frame is added, they are initialised to an empty string, instead of a default value. Complex Frames, that can have nested frames inside them have a `body` property, which contains a list of Frame objects.

With the `FrameBase` and the base of the specific Frame kind, a `PreEditableFrame` can be defined. This is not the complete frame yet, as it does not have the `being-edited` property, which indicated whether the Frame is `saved` or `edited`.

4.5 Editor Structure

The Frames are stored in a JSON-like representation, as a list of objects containing all their property values. Since the frames can be nested, the structure can also be thought of as a tree data structure. An example of the tree representation can be seen in Fig. 4.2. This is similar to how code parsers convert code to ASTs.

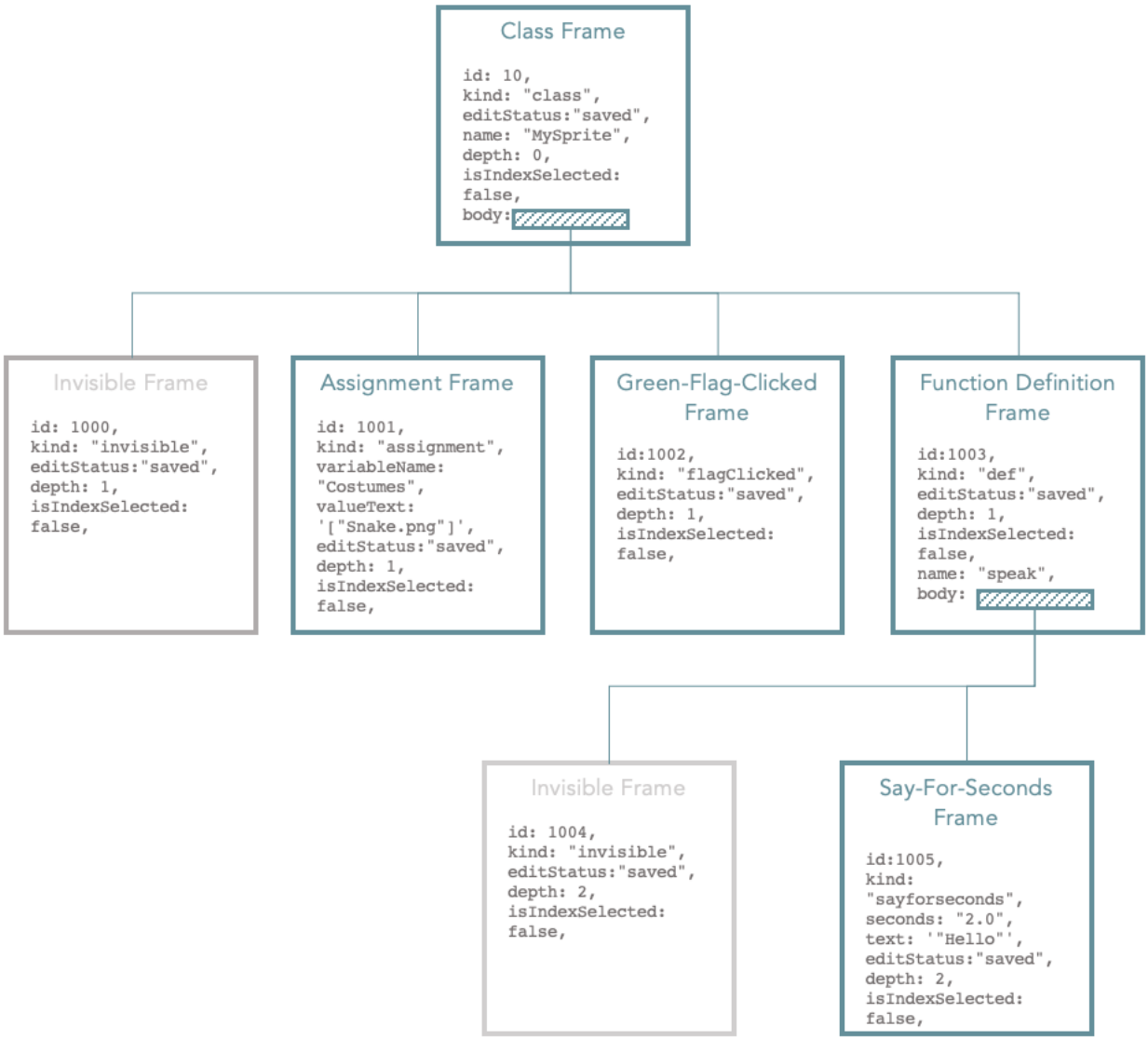


Figure 4.2: Abstract Tree Representation of a Sample Code

4.6 Frame Kinds

Each frame has a `kind` property. (The word "type" was avoided to avoid confusion with object-oriented terminology.) This determines the static text and the dynamic input parameters of the frame.

14 frame types have been implemented in the FBPE. These correspond to core Python concepts, such as comments, print statements, assignment statements, conditional statements, looping statements as well as simple Sprite actions and user inputs.

4.6.1 Comment Frame

Even though comments do not affect the compiled code, it is a useful tool in programming to help organise and annotate code. In Scratch, comments are different to other code blocks, they look like sticky notes connected to the block. In text-based Python, however, it is in-line and is only differentiated by syntax-highlighting. To prepare users for text-based programming, the FBPE comments were created as Frames.

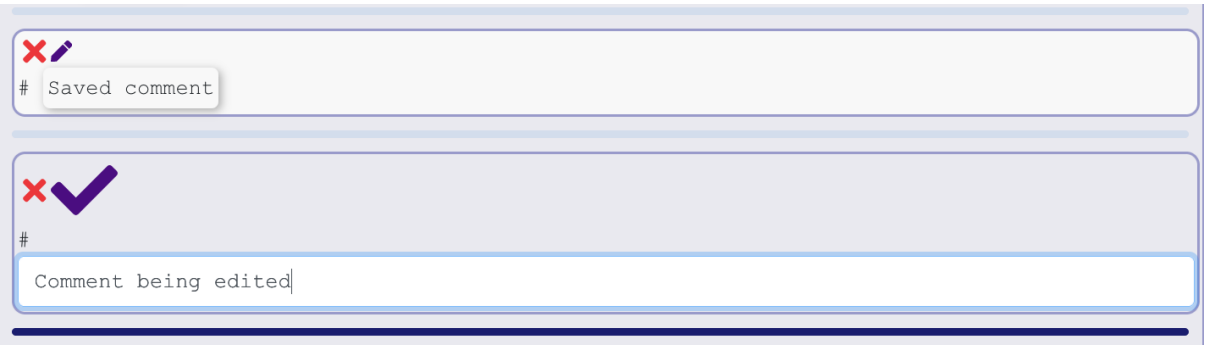


Figure 4.3: Comment Frame

4.6.2 Simple Frames

Simple Frames do not have a body, they do not contain other frames, hence they are leaf nodes in the AST. The following two simple frames are available in the FBPE:

- Variable Assignment,
- Print.

Since the datatype does not have to be specified at the variable definition in Python, the Variable Assignment Frame can be used to initialise values as well as to reassign values. To create or assign a value to a single variable, the user can type in the variable name and the value into the two text inputs in the Frame, as seen in Fig. 4.4.

More advanced options are also available in FBPE, though not trivial. A user can initialise a list by typing multiple values and adding square brackets. An example of this is shown in the second frame in Fig. 4.4. The third and fourth frames in Fig. 4.4 show examples of assigning values to multiple variables at a time.

The Print frame can be used to output a string, number or variable value. Instead of appearing in the console as it would in a text-based IDE, it will be displayed in the Pytch Output Area.

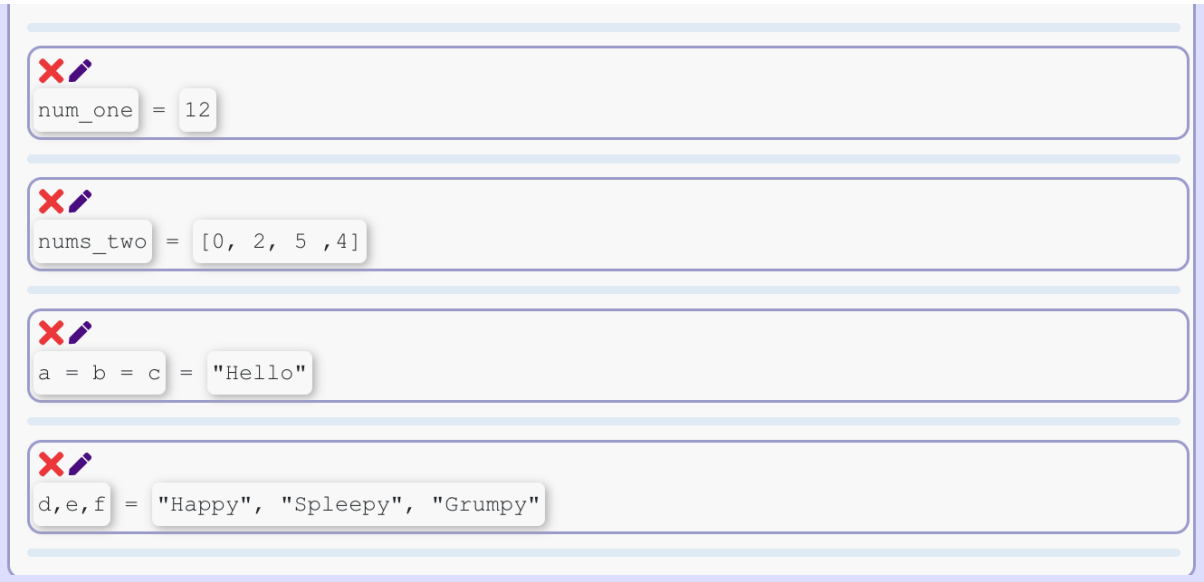


Figure 4.4: Assignment Frame Examples

4.6.3 Sprite Specific Frames

The Frames discussed in this section are for controlling Sprites on the Pytch Canvas. The Sprites can glide to a position, say something or wait for a number of seconds. In all three Frames, the user can specify the number of seconds the effect takes. These effects are:

- Glide,
- Say,
- Wait.

4.6.4 Complex Frames

Complex frames take some text input as well as contain a body parameter, that can contain one or more Frames. The complex Frame kinds implemented are as follows:

- If,
- While,
- For.

The If Frame allows conditional decisions. The While and For Frames are familiar to Scratch users from the "repeat", "repeat until" and "forever" blocks. In Python (unlike Scratch and non-object oriented languages) for-loops are usually used to iterate over a sequence (a list, dictionary, set, tuple or string). For this reason, the input for the for loop consist of two inputs, separated by the "in" keyword. One for the temporary name given to the item and the other for the sequence. The use of for loops is demonstrated in Fig. 4.5.

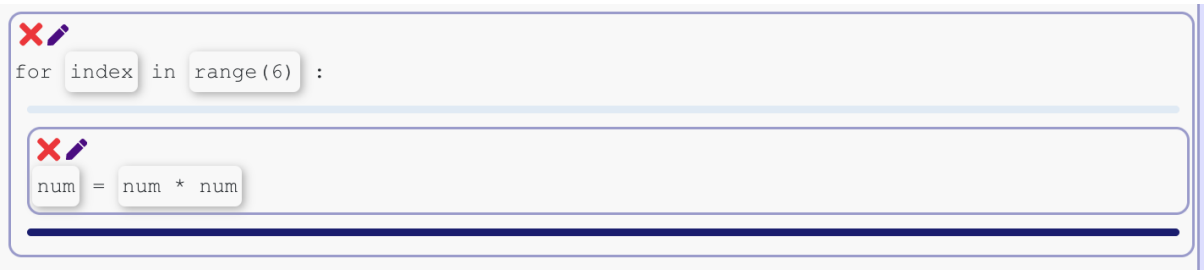


Figure 4.5: For-Loop Frame Example

4.6.5 Action Frames

Actions frames enable features that can react to user interactions. The following action frames were implemented:

- Sprite Clicked
- Green-Flag Clicked
- Key Pressed

The Sprite Clicked and Green Flag Clicked Frames are static, meaning they do not take any input from the user. For ease of use, they do not have a modify button, hence cannot be in a "being-modified" state.

4.6.6 Class and Function Definitions

In Pytch, the Sprites are represented as classes and the different actions are contained in functions. This made it necessary to add class and function definitions. In order to reduce complexity, the parameters are non-editable, so the user only needs to define the name of the class or function.

For an even more beginner-friendly editor, these could be completely hidden from the user or made uneditable, however, the program would be limited to a single Sprite.

4.6.7 Statement

As discussed previously, one of the goals of the editor was to limit initial complexity for users. There can, however, be a trade-off between ease of use and potential use cases. The Statement Frame was added as a way to work around this limitation.

While this Frame was not intended to be used by most users, it allows for more freedom, for example, to add more library import statements, function calls, break or continue statements and much more.

The Statement Frame takes a single text input that is converted to Python code directly.

The only modification applied to the text is the indentation depending on the position of the Frame.

4.7 Frame Controls

As previously discussed, new Frames can be added from the 'Add Frames' buttons on the left side of the editor. The Index allows the user to select the insertion point. There is an index button after each frame (and one at the top of the code area). On hover, the index buttons become more visible, and on click, the index button is selected and indicates the insertion point with a deep purple line. Only one index point can be selected at a time.

The index buttons are rendered as part of each Frame, so when an index is selected, the next frame is added after the corresponding frame. However, users might want to add new Frames at the top of the code, or inside bodies of complex frames, which do not contain any Frames yet. The solution to this problem was to introduce an Invisible Frame kind, which cannot be deleted or edited, its only function is to be an insertion point in the code area. Invisible Frames are automatically added to each new complex frame's body.

Most Frames can be either `saved` or `editable`. If they are in a `saved` state, they display the code they represent as solid text. For the users to edit the frame values, they can click the modify button, which is shown as a pen icon at the top-right of the frame.

Once the Frame is in an editable state, the text boxes appear, so the user can modify the values. A large tick icon takes place of the pen icon, which serves as a save button.

Only one Frame can be Editable at a time, which helps direct the user's attention. The values are being saved on every value change so that if the user decides to modify a different Frame without saving the current one, the changes will not be lost.

Some Frames have no editable component (Green Flag Clicked, Sprite Clicked), in which case the Edit button is not displayed and the frame can only be in a saved state.

In most IDEs there is a search option, however, since the FBPE runs in the browser and is a single block of code, the users can use the browser's built-in search bar.

4.8 Generating Python Code

Each frame represents a unit of Python code. However, they are stored as a list of objects, which needs to be converted into text format for the code compiler.

The FBPE generates the code when the Green Flag button is clicked. Considering the frames as an AST, the FBPE needs to run a search on the tree that reaches all nodes in the correct order. The correct method in this case is a preorder tree traversal starting at the

root of the tree. As an example, on the tree in Fig. 4.2, the Frames found are in the following order (by ID): 10, 1000, 1001, 1002, 1003, 1004, 1005.

The corresponding Python code for each Frame is generated by concatenating the static and dynamic elements together into a String. Finally, all lines are joined together, adding the necessary indentation according to their `depth` parameter.

4.9 Input validation

The aim of input validation in the FBPE was to notify users of potentially incorrect code before compilation. Input validation in the FBPE is different from traditional error messages, as they do not inform users of the type of problem. However, it most likely indicates invalid characters in the input filed or missing values.

To let the user know that the input might not be invalid, the value is displayed with a red border and a light red background as seen in Fig. 4.6. The effect is only visible on saved frames.

Figure 4.6: Input Field Validation Example

As an example, in the Variable Assignment Frame, the name of the variable can be validated, as there are a number of rules [30] that apply:

- The variable name has to start with an upper- or lowercase letter or underscore character (not a number);
- The variable name can only contain letters (alphanumeric values), numbers or underscores;

As discussed in Section 4.5.2, the Assignment Frame was intended to give users the ability to assign multiple variables at the same time. For this reason, white space, the ',' and the '=' characters should also be valid in the first input field.

The input is being validated against a Regular Expression (REGEX). The regular expression for the variable name field is:

```
'^([a-zA-Z_$\p{Zs}][0-9a-zA-Z_$,=\p{Zs}]*)$'
```

Due to time limitations, input validation has not been implemented for all Frames. Table 4.1 shows the list of Frames that have input validation so far.

For the Statement Frame, there is no need for input validation, as it is intended to give the user more freedom, or an alternative input in special cases. However, checking for an empty string can still be useful, to avoid redundant frames.

Table 4.1: Input validation Progress

Frame Type	Check for empty String	Check for Invalid characters
Statement	Yes	No
Assignment	Yes	Only for variable name
Class	Yes	Yes
Function	Yes	Yes
Wait	Yes	Yes

5 Evaluation

This chapter describes the evaluation methods proposed for the prototype. The objective evaluation of the users includes a workshop and survey. The technical evaluation methods are suitable for assessing the quality and robustness of the software.

5.1 Workshop and Survey Design

A workshop and survey were designed as a preliminary evaluation of the editor. The aim of the survey is to validate the usability of the editor with the target group and gather feedback on the key advantages and problems. An application was submitted to ethics board of Trinity College Dublin, which was sent back with requests for minor changes. Due to time limitations, the experiment fell out of the scope of this project.

In the workshop, the participants are asked to complete a short programming exercise in Pytch using the frame-based editor and complete a short questionnaire regarding their previous programming experience and their experience using the editor.

The survey takes approximately 40 minutes to complete and can be completed remotely without researcher supervision. This makes the recruitment process much easier, as the participants can complete the workshop and submit responses remotely, on their own time.

The participants were to be selected through convenience sampling from university students with enough previous programming experience to be familiar with basic concepts.

The survey was implemented in Qualtrics. The survey can be completed on any device with a browser installed, however, the workshop is recommended to be completed on a laptop or computer, as the editor has not yet been optimised for mobile devices.

At the beginning of the survey, the users are presented with the Participant Information Sheet, which describes the research, the workshop procedure, the anticipated risk and benefits, and the contact information of the researcher and the data protection officer. Following the information sheet, the informed consent form is shown. If the participant chooses to agree to the terms, they will begin the workshop. Otherwise, they will be

redirected to a page thanking them for their time and ensuring them that all of their data was removed.

5.1.1 Programming Exercises

When the participants reach the Exercises section of the workshop, they are redirected to a version of the Pytch website with the FBPE. The task is displayed in the info panel at the bottom of the screen. When the participant feels like they have completed the task to the best of their abilities, they can paste the generated Python code into the survey. They can then move on to the next exercise.

5.1.2 Survey

The survey consists of two blocks. The first block is concerned with the participant's previous programming experience. The second block is related to their experience using the frame-based python editor. The questions are based on validated scales.

5.2 Technical Evaluation

5.2.1 Manual Testing

All components were tested manually during the development process. The test flow for frame types can be seen in Fig. 5.1. In each case, the editor is firstly opened with a clean code area. Then the insertion point is selected and a new Frame is added. Since this is a general test design for all frame types, the Frame kind tested is inserted (e.g., Assignment Frame, Statement Frame).

The first test case is designed to evaluate the appearance and the most basic functionality of the Frame. If the Frame can be added, its appearance is as expected and it can be deleted, then the test has passed.

In the second test case, the input functionalities are evaluated. The Frame is completed with a valid input and then saved. The inputted value (or values) should now appear as part of the saved Frame.

The test case labelled "Generating Python Code" is designed to check whether the code passed to the compiler is complete and accurate. For testing purposes, the generated Python code is logged to the console.

The last test case is very similar to the second one, except invalid values are inputted into the text fields. The last test case only applies to the Frames in Table 4.1, for which the input validation has been implemented.

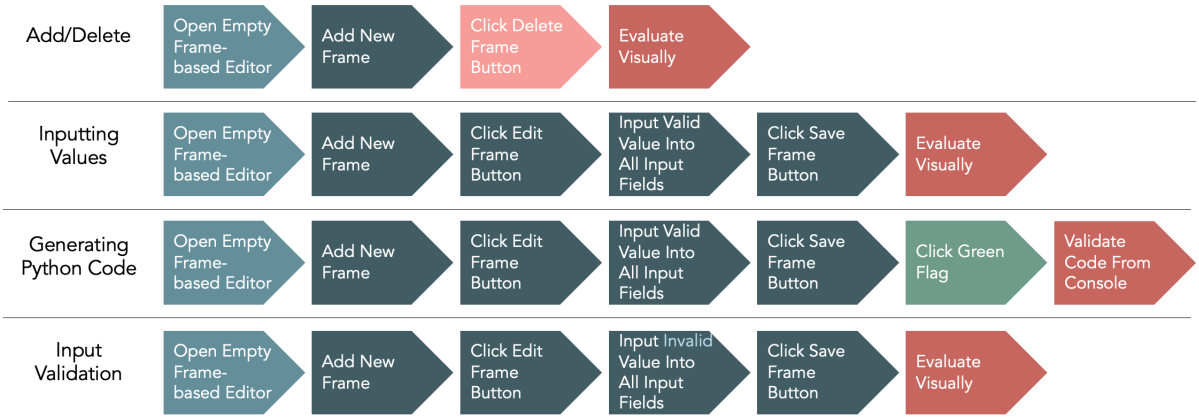


Figure 5.1: Manual Evaluation Test Case Flows for Frame Kinds

For non-editable Frames (Green Flag Clicked, Sprite Clicked) certain steps were omitted, as they do not contain an edit button or any input fields.

For complex frames (If, For, While, Function and Class definitions), additional testing was needed to validate the nesting functionality. In addition to all of the test cases described above, the same test cases were repeated for a Comment Frame inside the body of the tested frame, to ensure the flow of information is correct.

These manual tests were complete for each Frame type after every major change (usually before each Git commit). This ensures that the changes were correct and did not break any functionalities.

5.2.2 Software Testing

The code base could be made more robust with additional software testing. The initial attempts at using the unit-testing framework failed due to compatibility issues between Pytch and Jest, the unit testing library. Unfortunately, due to lack of time, the problem was not investigated further. However, both future users and developers could benefit from extensive unit testing.

Additionally, end-to-end testing could be implemented for the FBPE. For these purposes, TestCafe or Selenium would be the most suitable. Both of these testing suites have extensive documentation and training materials, and a community of users, all of which would make the testing process achievable. TestCafe also offers cross-browser and cross-platform [31] testing, which would contribute to the accessibility rating of the FBPE.

For now, the FBPE is lightweight enough that no performance issues were observed during implementation. However, in the future a performance metrics such as Website Speed, Number of Asset, Error or Failure to delivery rates should be taken into consideration.

6 Conclusion

6.1 Recommendations for Future Work

This project was part of a Master's dissertation project over two semesters. This set considerable limitations on the time and resources available for the project. The scale of this project was limited to a high-fidelity functional prototype, however, the FBPE could be developed into a fully functional product over time. This section will propose possibilities and make recommendations for future development.

The FBPE could be extended with a number of new features. User-driven software development is typically an unpredictable process, so it is hard to predict, which of these recommended features will be received well by FBPE users. However, it is still important to plan ahead and anticipate FBPE users' needs.

Firstly, additional accessibility features could be implemented. The group of potential FBPE users is currently limited by the Pytch website, however, there is still much room for improvement. Text alternatives could be added to non-text elements such as delete, save modify buttons, green and red flags. The content could be made easier to see by increasing the contrast, text and button sizes. Though in theory, the FBPE is compatible with most browsers this could be further investigated through cross-browser testing. Many interactions are currently only available with the mouse, but keyboard navigation could easily be added to the FBPE. Since there is a strong visual component to the FBPE, it is possible that the user's code would produce fast animation flashes, that could be dangerous for some users. It would require some advanced techniques and careful consideration, but a warning could be added if the Pytch Sprite's actions are expected to be triggering.

Syntax highlighting or assigning different colours to Frame types could help readability. This would be a simple change to implement technically, however, selecting appropriate and harmonising colours was out of the scope of this project.

Frame reordering is a key functionality, that was planned for the FBPE but was not implemented due to time constraints. With the Read DnD [32] library, the Frames could be moved by dragging and dropping them to a new position. Another way to help users

duplicate and reposition Frames would be through a copy, cut, and paste option. This would take less time to implement with a high potential impact on usability.

An undo-redo functionality could make the development process in the FBPE much more convenient. The Ctrl+Z and Ctrl+V commands are well known to most internet users and would be an ideal addition to the FBPE. This could be implemented by saving snapshots of previous states of the Frames.

Currently, Pytch, users can have several projects saved, that they can save and reload at their convenience. This is done by saving progress to the browser's cookies. As for now, the FBPE does not save progress, which could discourage users from tackling more complex problems.

Some of Pytch's custom functions were made available in the FBPE, however, there are almost a hundred more that could still be added. Practically speaking, since the FBPE is targeted at novice programmers only starting out with Python, a limited functionality, thus a simple user interface is also an advantage. Future research could investigate the best number of Frame types and implement them.

The concept of using the FBPE in alternative applications could also be explored further. The core structure could be used for other purposes; even for other programming languages.

6.2 Conclusion

The aim of this project was to develop a functional prototype for a Python editor. The Python editor was to be targeted at novice programmers, mainly aged 16 and older, who are familiar with programming concepts from Scratch or other languages and wish to pursue Python programming.

A frame-based approach was followed, which combines characteristics of both block-based languages and text-based languages. It allows the user to focus on the programming logic and problem solving while becoming familiar with the correct Python syntax. Thanks to the Pytch integration, the FBPE can be used to write programs that compile and take effect immediately on the screen. For users already familiar with programming concepts for Scratch or elsewhere, its features are similar enough to Scratch to be intuitive without much explanation.

A detailed evaluation model was proposed for the FBPE prototype, including a workshop, survey and a technical evaluation framework. A workshop and survey have been designed, that can be used for a preliminary evaluation. It consists of simple exercises which involve all the current functionalities of the FBPE, and a survey of the participant's programming

experience and their feedback about the editor. This could be used for a user-driven development.

The core components of a software testing framework have also been established. A manual test flow has been developed for the frames, which can help detect defects and bugs.

The FBPE has the potential for supporting its users in the transition from Scratch to Python, but more user trials are necessary. Ideally, the FBPE would be fine-tuned through an iterative cycle of user testing and development. The prototype developed in this project can be used as a basis for further development, due to its reusable components, accessible framework and scalable structure. The core structure of the FBPE could easily be expanded with other adapted to other programming languages.

Bibliography

[1] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond, "The Scratch Programming Language and Environment," *ACM Transactions on Computing Education*, vol. 10, no. 4, pp. 1–15, Nov. 2010. [Online]. Available: https://dl.acm.org/doi/10.1145/1868358.1868363

[2] K. Powers, S. Ecott, and L. M. Hirshfield, "Through the looking glass: teaching CS0 with Alice," *ACM SIGCSE Bulletin*, vol. 39, no. 1, pp. 213–217, Mar. 2007. [Online]. Available: https://doi.org/10.1145/1227504.1227386

[3] M. Kölling, "The Greenfoot Programming Environment," *ACM Transactions on Computing Education*, vol. 10, no. 4, pp. 1–21, Nov. 2010. [Online]. Available: https://dl.acm.org/doi/10.1145/1868358.1868361

[4] G. Strong and B. North, "Pytch — an environment for bridging block and text programming styles (Work in progress)," in *The 16th Workshop in Primary and Secondary Computing Education*. Virtual Event Germany: ACM, Oct. 2021, pp. 1–4. [Online]. Available: https://dl.acm.org/doi/10.1145/3481312.3481318

[5] D. Weintrop and U. Wilensky, "Using Commutative Assessments to Compare Conceptual Understanding in Blocks-based and Text-based Programs," in *Proceedings of the eleventh annual International Conference on International Computing Education Research*, ser. ICER '15. New York, NY, USA: Association for Computing Machinery, Aug. 2015, pp. 101–110. [Online]. Available: https://doi.org/10.1145/2787622.2787721

[6] E. Pasternak, R. Fenichel, and A. N. Marshall, "Tips for creating a block language with blockly," in *2017 IEEE Blocks and Beyond Workshop (B B)*, Oct. 2017, pp. 21–24.

[7] C. D. Hundhausen, S. F. Farley, and J. L. Brown, "Can direct manipulation lower the barriers to computer programming and promote transfer of training? An experimental study," *ACM Transactions on Computer-Human Interaction*, vol. 16, no. 3, pp. 13:1–13:40, Sep. 2009. [Online]. Available: https://doi.org/10.1145/1592440.1592442

[8] F. Kalelioglu, Y. Gulbahar, and D. Doğan, "Teaching How to Think Like a Programmer: Emerging Insights," Oct. 2017, pp. 18–35.

[9] D. Weintrop and U. Wilensky, "To block or not to block, that is the question: students' perceptions of blocks-based programming," in *Proceedings of the 14th International Conference on Interaction Design and Children*, ser. IDC '15. New York, NY, USA: Association for Computing Machinery, Jun. 2015, pp. 199–208. [Online]. Available: https://doi.org/10.1145/2771839.2771860

[10] L. Gusukuma, D. Kafura, and A. C. Bart, "Authoring feedback for novice programmers in a block-based language," in *2017 IEEE Blocks and Beyond Workshop (B B)*, Oct. 2017, pp. 37–40.

[11] N. Brown, C. Kyfonidis, P. Weill-Tessier, B. Becker, J. Dillane, and M. Kölling, "A Frame of Mind: Frame-based vs. Text-based Editing," in *United Kingdom and Ireland Computing Education Research conference.* Glasgow United Kingdom: ACM, Sep. 2021, pp. 1–7. [Online]. Available: https://dl.acm.org/doi/10.1145/3481282.3481286

[12] D. Krpan, S. Mladenović, and G. Zaharija, "Mediated transfer from visual to high-level programming language," *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2017.

[13] T. Beelders and J.-P. du Plessis, "The Influence of Syntax Highlighting on Scanning and Reading Behaviour for Source Code," in *Proceedings of the Annual Conference of the South African Institute of Computer Scientists and Information Technologists*, ser. SAICSIT '16. New York, NY, USA: Association for Computing Machinery, Sep. 2016, pp. 1–10. [Online]. Available: https://doi.org/10.1145/2987491.2987536

[14] A. Afzal and C. Le Goues, "A Study on the Use of IDE Features for Debugging," in *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, May 2018, pp. 114–117, iSSN: 2574-3864.

[15] R. D'souza, M. Bhayana, M. Ahmadzadeh, and B. Harrington, "A Mixed-Methods Study of Novice Programmer Interaction with Python Error Messages," in *Proceedings of the Western Canadian Conference on Computing Education*. Calgary AB Canada: ACM, May 2019, pp. 1–2. [Online]. Available: https://dl.acm.org/doi/10.1145/3314994.3325090

[16] E. Lahtinen, K. Ala-Mutka, and H.-M. Järvinen, "A study of the difficulties of novice programmers," *ACM SIGCSE Bulletin*, vol. 37, no. 3, pp. 14–18, Jun. 2005. [Online]. Available: https://doi.org/10.1145/1151954.1067453

[17] B. Simon, R. Lister, and S. Fincher, "Multi-Institutional Computer Science Education Research: A Review of Recent Studies of Novice Understanding," in *Proceedings.*

Frontiers in Education. 36th Annual Conference. San Diego, CA, USA: IEEE, 2006, pp. 12–17. [Online]. Available: http://ieeexplore.ieee.org/document/4116894/

[18] L. Budde, B. Heinemann, and C. Schulte, "A theory based tool set for analysing reading processes in the context of learning programming," in *Proceedings of the 12th Workshop on Primary and Secondary Computing Education*, ser. WiPSCE '17. New York, NY, USA: Association for Computing Machinery, Nov. 2017, pp. 83–86. [Online]. Available: https://doi.org/10.1145/3137065.3137077

[19] R. Bednarik and M. Tukiainen, "An eye-tracking methodology for characterizing program comprehension processes," in *Proceedings of the 2006 symposium on Eye tracking research & applications*, ser. ETRA '06. New York, NY, USA: Association for Computing Machinery, Mar. 2006, pp. 125–132. [Online]. Available: https://doi.org/10.1145/1117309.1117356

[20] R. Talsma, E. Barendsen, and S. Smetsers, "Analyzing the influence of block highlighting on beginning programmers' reading behavior using eye tracking," Oct. 2020, pp. 1–10.

[21] A. Altadmri and N. C. Brown, "37 Million Compilations: Investigating Novice Programming Mistakes in Large-Scale Student Data," in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '15. New York, NY, USA: Association for Computing Machinery, Feb. 2015, pp. 522–527. [Online]. Available: https://doi.org/10.1145/2676723.2677258

[22] D. McCall and M. Kölling, "A New Look at Novice Programmer Errors," *ACM Transactions on Computing Education*, vol. 19, no. 4, pp. 38:1–38:30, Jul. 2019. [Online]. Available: https://doi.org/10.1145/3335814

[23] J. Dillane, I. Karvelas, and B. Becker, *Portraits of Programmer Behavior in a Frame-Based Language*, Apr. 2022.

[24] M. Kölling, N. C. C. Brown, and A. Altadmri, "Frame-Based Editing: Easing the Transition from Blocks to Text-Based Programming," in *Proceedings of the Workshop in Primary and Secondary Computing Education*. London United Kingdom: ACM, Nov. 2015, pp. 29–38. [Online]. Available: https://dl.acm.org/doi/10.1145/2818314.2818331

[25] T. W. Price, N. C. Brown, D. Lipovac, T. Barnes, and M. Kölling, "Evaluation of a Frame-based Programming Editor," in *Proceedings of the 2016 ACM Conference on International Computing Education Research*. Melbourne VIC Australia: ACM, Aug. 2016, pp. 33–42. [Online]. Available: https://dl.acm.org/doi/10.1145/2960310.2960319

[26] G. Strong, S. O'Carroll, and N. Bresnihan, "A block based editor for Python," in *Proceedings of the 13th Workshop in Primary and Secondary Computing Education*. Potsdam Germany: ACM, Oct. 2018, pp. 1–2. [Online]. Available: https://dl.acm.org/doi/10.1145/3265757.3265788

[27] G. Bierman, M. Abadi, and M. Torgersen, "Understanding TypeScript," in *ECOOP*, 2014.

[28] W. W. A. Initiative (WAI), "Introduction to Web Accessibility." [Online]. Available: https://www.w3.org/WAI/fundamentals/accessibility-intro/

[29] "Ace - The High Performance Code Editor for the Web." [Online]. Available: https://ace.c9.io/

[30] "Python Variable Names." [Online]. Available: https://www.w3schools.com/python/gloss_python_variable_names.asp

[31] "Cross-Browser End-to-End Testing Framework | Testafe." [Online]. Available: https://testcafe.io/

[32] lvshihao, "react-flex-dnd," Feb. 2022, original-date: 2021-11-19T06:09:58Z. [Online]. Available: https://github.com/Asweallcan/react-flex-dnd

A1 Appendix

A1.1 Link to FBPE GitHub Page

https://github.com/illesie/editor-pytch-integration

A1.2 Ethics Application

Changes to the Revised Application

1. Please find the updated questionnaire in Appendix 4. Link to the updated questionnaire: https://scsstcd.qualtrics.com/jfe/preview/SV_3Qm5lZS3pBv1CLk?Q_CHL=preview&Q_SurveyVersionID=current

2. The total time of the workshop is 40 minutes. Of this, the programming task should take 30-35 minutes, and the questionnaire should take 5-10 minutes. This has been clarified in the online form, the consent form, and the participant information sheet.

School of Computer Science & Statistics
Research Ethics Application

Part A

Project Title: .....Frame based Python editor.....

Name of Lead Researcher (student in case of project work): ..... Anna Eniko Illesi.....

Name of Supervisor: .....Glenn Strong.....

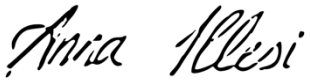
TCD E-mail: ……illesie@tcd.ie…….....……. Contact Tel No.: ….……+353 89 602 6592……

Course Name and Code (if applicable): …… CSP55E01-202122 Computer Engineering Research Project…….

Estimated start date of survey/research: …….....12/04/2022.....…...……

I confirm that I will (where relevant):

- Familiarize myself with the Data Protection Act and the College Good Research Practice guidelines http://www.tcd.ie/info_compliance/dp/legislation.php;
- Tell participants that any recordings, e.g. audio/video/photographs, will not be identifiable unless prior written permission has been given. I will obtain permission for specific reuse (in papers, talks, etc.)
- Provide participants with an information sheet (or web-page for web-based experiments) that describes the main procedures (a copy of the information sheet must be included with this application)
- Obtain informed consent for participation (a copy of the informed consent form must be included with this application)
- Should the research be observational, ask participants for their consent to be observed
- Tell participants that their participation is voluntary
- Tell participants that they may withdraw at any time and for any reason without penalty
- Give participants the option of omitting questions they do not wish to answer if a questionnaire is used
- Tell participants that their data will be treated with full confidentiality and that, if published, it will not be identified as theirs
- On request, debrief participants at the end of their participation (i.e. give them a brief explanation of the study)
- Verify that participants are 18 years or older and competent to supply consent.
- If the study involves participants viewing video displays then I will verify that they understand that if they or anyone in their family has a history of epilepsy then the participant is proceeding at their own risk
- Declare any potential conflict of interest to participants.
- Inform participants that in the extremely unlikely event that illicit activity is reported to me during the study I will be obliged to report it to appropriate authorities.
- Act in accordance with the information provided (i.e. if I tell participants I will not do something, then I will not do it).



Signed: ..... Date: .....31/03/2022.....

Lead Researcher/student in case of project work

Part B

Please answer the following questions.		***Yes/No***
Has this research application or any application of a similar nature connected to this research project been refused ethical approval by another review committee of the College (or at the institutions of any collaborators)?		No
Will your project involve photographing participants or electronic audio or video recordings?		No
Will your project deliberately involve misleading participants in any way?		No
Does this study contain commercially sensitive material?		No
Is there a risk of participants experiencing either physical or psychological distress or discomfort? If yes, give details on a separate sheet and state what you will tell them to do if they should experience any such problems (e.g. who they can contact for help).		No
Does your study involve any of the following?	Children (under 18 years of age)	No
	People with intellectual or communication difficulties	No
	Patients	No

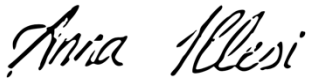
School of Computer Science and Statistics
Research Ethical Application Form

Details of the Research Project Proposal must be submitted as a separate document to include the following information:

1. Title of project
2. Purpose of project including academic rationale
3. Brief description of methods and measurements to be used
4. Participants - recruitment methods, number, age, gender, exclusion/inclusion criteria, including statistical justification for numbers of participants
5. Debriefing arrangements
6. A clear concise statement of the ethical considerations raised by the project and how you intend to deal with them
7. Cite any relevant legislation relevant to the project with the method of compliance e.g. Data Protection Act etc.

Part C

I confirm that the materials I have submitted provided a complete and accurate account of the research I propose to conduct in this context, including my assessment of the ethical ramifications.

Signed:  ..... Date: ..... 31/03/2022.....
Lead Researcher/student in case of project work

There is an obligation on the lead researcher to bring to the attention of the SCSS Research Ethics Committee any issues with ethical implications not clearly covered above.

Part D

If external or other TCD Ethics Committee approval has been received, please complete below.

External/TCD ethical approval has been received and no further ethical approval is required from the School's Research Ethical Committee. I have attached a copy of the external ethical approval for the School's Research Unit.

Signed: ..... Date: .....
Lead Researcher/student in case of project work

Part E

If the research is proposed by an undergraduate or postgraduate student, please have the below section completed.

I confirm, as an academic supervisor of this proposed research that the documents at hand are complete (i.e. each item on the submission checklist is accounted for) and are in a form that is suitable for review by the SCSS Research Ethics Committee

Signed: .....  ..... Date: ..... 31/03/2022.....
Supervisor

Completed application forms together with supporting documentation should be submitted electronically to the online ethics system - https://webhost.tchpc.tcd.ie/research_ethics/ When your application has been reviewed and approved by the Ethics committee, hardcopies with original signatures should be submitted to the School of Computer Science & Statistics, Room 104, Lloyd Building, Trinity College, Dublin 2.

CHECKLIST

Please ensure that you have submitted the following documents with your application:

1.	• SCSS Ethical **Application Form**	✓
2.	• **Participant's Information Sheet** must include the following: a) Declarations from Part A of the application form; b) Details provided to participants about how they were selected to participate; c) Declaration of all conflicts of interest.	✓
3.	• **Participant's Consent Form** must include the following: a) Declarations from Part A of the application form; **b)** Researchers contact details provided for counter-signature (your participant will keep one copy of the signed consent form and return a copy to you).	✓
4.	• **Research Project Proposal** must include the following: a) You must inform the Ethics Committee **who** your intended participants are i.e. are they your work colleagues, class mates etc. b) How will you recruit the participants i.e. **how** do you intend asking people to take part in your research? For example, will you stand on Pearse Street asking passers-by? c) If your participants are under the age of 18, you must seek both parental/guardian AND child consent.	✓
5.	• Intended **questionnaire**/survey/interview protocol/screen shots/representative materials (as appropriate)	✓
6.	• **URL** to intended on-line survey (as appropriate)	✓

Notes on Conflict of Interest

1. If your intended participants are work colleagues, you must declare a potential conflict of interest: you are taking advantage of your existing relationships in order to make progress in your research. It is best to acknowledge this in your invitation to participants.
2. If your research is also intended to direct commercial or other exploitation, this must be declared. For example, *"Please be advised that this research is being conducted by an employee of the company that supplies the product or service which form an object of study within the research."*

Notes for questionnaires and interviews

1. If your questionnaire is **paper based**, you must have the following **opt-out** clause on the top of each page of the questionnaire: "*Each question is optional. Feel free to omit a response to any question; however, the researcher would be grateful if all questions are responded to."*
2. If your questionnaire is **on-line**, the first page of your questionnaire must repeat the content of the information sheet. This must be followed by the consent form. If the participant does not agree to the consent, they must automatically be exited from the questionnaire.
3. Each question must be **optional**.
4. The participant must have the option to '**not submit, exit without submitting'** at the final submission point on your questionnaire.
5. If you have open-ended questions on your questionnaire, you must warn the participant against naming **third parties**: "*Please do not name third parties in any open text field of the questionnaire. Any such replies will be anonymised."*
6. You must inform your participants regarding **illicit activity**: "*In the extremely unlikely event that illicit activity is reported I will be obliged to report it to appropriate authorities."*

SCSS Research Project Proposal

SECTION 1 – Project Description

Title of Study: Frame-based python editor

Duration of Study: March 2022 – April 2022

Purpose of the project:

The transition from block-based programming environments (such as Scratch) can present a challenge for novice programmers. In this study an editor prototype is developed. The goal of the study is a preliminary evaluation of the usefulness of the prototype. The frame-based was proposed attempting to bridge the gap between highly-visual, intuitive block-based editors and the more abstract, text-based editors. It is intended for students, who are already familiar with basic programming concepts. The editor was proposed as a supporting tool for the Pytch system, a web-based educational environment developed at Trinity College Dublin.

The challenges beginners face when adjusting to a text-based programming environment are well documented in the literature. Numerous systems have been proposed to help novices with these initial challenges. The unique contribution of the frame-based editor comes from the fact that it serves as an introduction to python syntax while maintaining the benefits of block-based editors.

Participants:

Participants in this experiment will be recruited using convenience sampling, primarily through our network of contacts. An estimated 20 adult participants will be recruited. For the purposes of the study, participants should have at least a basic understanding of programming concepts, so students attending computer science courses in Trinity College Dublin and Milton Friedman University will be recruited. The participants are expected to be mostly young adults between 18 and 25 years old. The purpose of this study is to validate the effectiveness of the editor and to identify any major flaws or concerns. Hence, the small sample size is sufficient.

Procedures:

The study will take place through remote workshops in which participants will be asked to use the frame based editor to complete programming tasks and fill out a questionnaire.

The questionnaire is fully anonymous. Questions related to the participants' previous knowledge of programming will be included, derived from validated scales intended to measure programming self-efficacy. The participants will be asked to complete one or more programming exercises and upload their solutions in the questionnaire. The final set of questions relates to the experience using the editor. The exercise environment is not ready yet, so the link in the survey (https://www.pytch.org/app/frame-based-editor/) is invalid for now. The exercises will be similar to the Pytch tutorials (https://www.pytch.org/app/tutorials/).

The programming tasks and the questionnaire take approximately 40 minutes in total (30-35 minutes for the programming task and 5-10 minutes for the survey).

Selected references for scales used:

Abdunabi, Ramadan, Ilham Hbaci, and Heng-Yu Ku. “Towards Enhancing Programming Self-Efficacy Perceptions among Undergraduate Information Systems Students.” *Journal of Information Technology Education: Research* 18 (January 1, 2019): 185–206. https://doi.org/10.28945/4308.

Kukul, Volkan, Şahin Gökçearslan, and Mustafa Serkan Günbatar. “Computer Programming Self-Efficacy Scale (CPSES) for Secondary School Students: Development, Validation and Reliability.” *Educational Technology Theory and Practice* 7 (January 25, 2017): 158–79. https://doi.org/10.17943/ETKU.72918.

“The Effect of a Web-Based Coding Tool with Automatic Feedback on Students’ Performance and Perceptions | Proceedings of the 49th ACM Technical Symposium on Computer Science Education.” Accessed February 3, 2022. https://dl.acm.org/doi/10.1145/3159450.3159579.

Venkatesh, Viswanath, and Fred D. Davis. “A Theoretical Extension of the Technology Acceptance Model: Four Longitudinal Field Studies.” *Management Science* 46, no. 2 (2000): 186–204. https://www.jstor.org/stable/2634758.

Debriefing: The email address of the student conducting the study will be made available to each participant. In the workshop, the participants will be invited to contact the researchers and ask questions.

Link to intended questionnaire:

https://scsstcd.qualtrics.com/jfe/preview/SV_3Qm5lZS3pBv1CLk?Q_CHL=preview&Q_SurveyVersionID=current

SECTION 2 – Ethical Concerns

Potential benefits and potential harms to participants:

The well-being and safety of participants is highly important.

All questions and any participation in the study is optional. At the end of the survey, the option to withdraw from the study and have all data removed will be available.

All participant identities will be kept anonymous.

Conflicts of interest:

As we will be recruiting from our network, there is a potential conflict of interest. To reduce this risk we will discuss this openly with all participants, and remain mindful of the risk during the study.

SECTION 3 – CONFIDENTIALITY AND DATA PROTECTION

Note that it is up to the researcher (and their supervisor if relevant) to discharge their own professional and legal obligations with respect to the handling of personal data. Please consult the College Data Protection Officer (DPO) if in any doubt regarding your obligations.

3.1 Does this study involve collecting, using, accessing or sharing personal data[1]?

Yes ☐ No ☒

If **NO**, please go to section 4.
If **YES,** please list[2] all categories of personal data.
Please see checklist on secure storage available here.

Type of Data	Justification: Why do you need the data?	Data Format	Technical and Organisational Controls	Identifiable coded, or anonymised

1 Personal data is information which can identify a person. In particular: a name, address, email, telephone number, an identification number, location data, an online identifier, an IP address, a code key linking back to identifiable data etc. Please note that pseudonymised data is personal data under GDPR. Pseudonymised data means data which cannot be attributed to an individual without the use of additional information which is kept separately. (i.e. a key) . It is sometimes referred to as 'coded data'. Please note that in order to be considered personal data in our hands, Trinity must hold the key.

2 If using Personal data. Records must be kept pursuant to Article 30, GDPR; https://gdpr-info.eu/art-30-gdpr/

3.2 Does the study involve collecting, using, accessing or sharing sensitive data[3]?

Yes ☒ No ☐

If **NO,** please go to question 3. 3.
If **YES**, please list all categories of the sensitive data collected.

Please see checklist on secure storage available here.

Type of Data	Justification: Why you need the data?	Data Format	Technical and Organisational Controls	Identifiable coded, or anonymised
Experience with programming.	Used to obtain insight into connection between experience with editor and programming expertise.	Online Qualtrics survey	On local machine which uses full-disk encryption. Accessible to researcher only. On secure servers in encrypted format.	Anonymised
Python code generated from exercise solutions.	Used to evaluate editor usability by confirming how much of the exercises users were able to complete.	Online Qualtrics survey	On local machine which uses full-disk encryption. Accessible to researcher only. On secure servers in encrypted format.	Anonymised

3.3 Who determines how and why the personal and/or sensitive data is used?

(Data Controller[4] or Joint Data Controllers**)**

Provide Details: Data Protection Officer, Secretary's Office, Trinity College Dublin, Dublin 2 - dataprotection@tcd.ie

3.4 Will the personal and/or sensitive personal data be shared with any third parties[5]?

No.

3 Sensitive personal data means personal data, which poses a higher risk to the individual. It includes personal data revealing racial and ethnic origin, political opinions, religious or ideological convictions, trade union membership, criminal convictions and offences, genetic, biometric (photos, videos, audio etc.) data concerning physical or mental well-being , information relating to education, professional training employment and career history, questionnaires, Information relating to the family of the individual and the individual's lifestyle and social circumstances.

4 *Employees and students of TCD are not data controllers. TCD is the data controller for the institution. However, if other institutes jointly decide how and why the data will be used, they should also be noted as controllers here.*

5 Third parties could be collaborators (institutes/industry) or service providers (transcribers, cloud storage etc.)

If **YES,** provide details including information on the contractual arrangements in place. If **NO**, please go to question 3.5

This list should include all Data Processing Agreements with external laboratories, Cloud-based Solutions Agreements etc., and any and Data Sharing Agreements with Collaborators.

Please contact researchDPO@tcd.ie if you need assistance with agreements and/or for any transfer outside EEA (including England, Wales, Scotland or Northern Ireland).

Provide Details:

3.5 How long will you retain the personal data?

Please see good research practice guide for guidance on retention of research data. Your school should be able to advice on best practice.

10 years by supervisor for research integrity purposes in secure server in encrypted format, after which point it will be deleted.

3. 6 Will the personal data be fully anonymised or deleted after it is no longer necessary?

Yes.

See advice on secure data disposal Provide Details: Data will be deleted when no longer needed.

3.7 How will you inform participants of their rights under GDPR[6]:

A data protection section is present on information and consent sheets supplied to all participants.

Please note that the DPO's contact details must be included on any information leaflet or privacy notice if you are using personal data for your research.

Email: dataprotection@tcd.ie
Post: Data Protection Officer, Secretary's Office, Trinity College Dublin, Dublin 2, Ireland

[6] Under GDPR, these include:
- right of access;
- right to rectification;
- right to erasure;
- right to object to processing based on legitimate or public interest;
- right to data portability;
- right to object to profiling or making decisions about individuals by automated means?

APPENDIX 1 – Information Sheet

TRINITY COLLEGE DUBLIN

INFORMATION SHEET FOR PROSPECTIVE PARTICIPANTS

Frame-based python editor

We have developed a frame-based editor for python, which combines the advantages of both block-based and text-based programming. It aims to create a transitional environment from block-based editors (such as Scratch) to text-based programming languages (such as python). You are invited to participate to help evaluate how well the editor can be used for programming tasks.

This study is being carried out by an MAI student as part of their masters dissertation with the School of Computer Science and Statistics at Trinity College Dublin.

We are primarily seeking adult participants who have some experience with either text or block-based programming languages and are familiar with basic concepts of programming logic (e.g. variables, conditionals, loops, functions).

Workshop procedure
Workshop participants will complete a series of programming tasks using the Frame-Based Editor. Once you have completed or attempted each exercise, you will be asked to upload the generated python code in the questionnaire. After the exercises, you will be asked to complete a short questionnaire about your previous programming experience and your experience with the editor.

The questionnaire is fully anonymous. All data will be stored on a secure server in an encrypted format and only members of the research team will have access to it.

The workshop will take approximately 40 minutes. (The programming task should take 30-35 minutes, and the questionnaire should take 5-10 minutes.)

Anticipated risks
We do not anticipate any risks to participants.

Benefits to the participant
While participation will not benefit the participants directly, the research will support the development of better tools for programming education.

Voluntary and anonymous participation
Your participation is entirely voluntary, and you can withdraw at any time before you have submitted the questionnaire. At the end of the questionnaire, there will be an option for you to withdraw from the study and have all associated data removed. As there is no way for us (or anyone else) to identify an individual participant once they submit their data, it will no longer be possible to withdraw once you have submitted the questionnaire.

Questionnaires
All questions are optional. Feel free to omit a response to any question; however, the researcher would be grateful if all questions are responded to.

Debriefing
The researchers will be available to answer any questions about the research. If you have any queries feel free to contact illesie@tcd.ie and we will be happy to answer any questions.

Participation in analysis and publication
The data collected will be analyzed to determine how to make programming environments and supports that help people learn programming. The research results will be published in a masters dissertation at Trinity College Dublin. We also plan to publish the results of our research in academic journals and conference proceedings. We will do this in a way, that does not identify you, or any other individual participant.

Cautions about inadvertent discovery of illicit activities
While it is unlikely that illicit activities would be disclosed, if you do so, we would be obliged to report them to the appropriate authorities.

Conflicts of interest
The researchers may have made use of personal contacts in recruiting.

Contact details
The study is led by Anna Illesi (illesie@tcd.ie) and supervised by Glenn Strong (glenn.strong@tcd.ie) from the School of Computer Science and Statistics in Trinity College Dublin.

If you have any queries, feel free to contact us and we will be happy to answer questions about the experiment.

Data Protection details
Data Controllers: Trinity College Dublin

Data Protection Officer: Data Protection Officer, Secretary's Office, Trinity College Dublin, Dublin 2 - dataprotection@tcd.ie

APPENDIX 2 – Consent form

TRINITY COLLEGE DUBLIN
INFORMED CONSENT FORM

LEAD RESEARCHERS: Anna Illesi

BACKGROUND OF RESEARCH: We have developed a frame-based editor for python, which combines the advantages of both block-based and text-based programming. You are invited to participate to help evaluate how well the editor can be used for programming tasks.

PROCEDURES OF THIS STUDY:
Workshop participants will complete a series programming tasks using the Frame-Based Editor. Once you have completed or attempted each exercise, you will be asked to upload the generated python code in the questionnaire. After the exercises you will be asked to complete a short questionnaire about your previous programming experience and your experience with the editor.

The questionnaire is fully anonymous. All this data will be stored on a secure server in an encrypted format and only members of the research team will have access to it.

The workshop will take approximately 40 minutes. (The programming task should take 30-35 minutes, and the questionnaire should take 5-10 minutes.)

PUBLICATION:
The research results will be published in a masters dissertation at Trinity College Dublin. We also plan to publish the results of our research in academic journals and conference proceedings.

CONFLICTS OF INTEREST: The researchers may have made use of personal contacts in recruiting.

Individual results may be aggregated anonymously, and research reported on aggregate results.

DECLARATION:

- I am 18 years or older and am competent to provide consent.
- I have read, or had read to me, a document providing information about this research and this consent form. I have had the opportunity to ask questions and all my questions have been answered to my satisfaction and understand the description of the research that is being provided to me.
- I agree that my data is used for scientific purposes, and I have no objection that my data is published in scientific publications in a way that does not reveal my identity.
- I understand that if I make illicit activities known, these will be reported to appropriate authorities.
- I understand that I may refuse to answer any question and that I may withdraw at any time without penalty.
- I understand that if the results of the research have been published, or my data has been fully anonymised so that it can no longer be attributed to me, then it will no longer be possible to withdraw.
- **I understand that I may stop at any time, and have data destroyed by ticking the option to withdraw from the study at the end of the questionnaire.**
- I freely and voluntarily agree to be part of this research study, though without prejudice to my legal and ethical rights.
- I understand that my participation is fully anonymous and that no personal details about me will be recorded.
- I understand that if I or anyone in my family has a history of epilepsy then I am proceeding at my own risk.
- I have received a copy of this agreement.

By signing this document, I consent to participate in this study, and consent to the data processing necessary to enable my participation and to achieve the research goals of this study.

PARTICIPANT'S NAME:

PARTICIPANT'S SIGNATURE:

Date:

Statement of investigator's responsibility: I have explained the nature and purpose of this research study, the procedures to be undertaken and any risks that may be involved. I have offered to answer any questions and fully answered such questions. I believe that the participant understands my explanation and has freely given informed consent.

RESEARCHERS CONTACT DETAILS:

RESEARCHER'S SIGNATURE:

Date:

Appendix 3 – Link to online questionnaire

Preview link to questionnaire:
https://scsstcd.qualtrics.com/jfe/preview/SV_3Qm5lZS3pBv1CLk?Q_CHL=preview&Q_SurveyVersionID=current

Appendix 4 – Questionnaire

Q1

FRAME-BASED PYTHON EDITOR

TRINITY COLLEGE DUBLIN

INFORMATION SHEET FOR PROSPECTIVE PARTICIPANTS

We have developed a frame-based editor for python, which combines the advantages of both block-based and text-based programming. It aims to create a transitional environment from block-based editors (such as Scratch) to text-based programming languages (such as python). You are invited to participate to help evaluate how well the editor can be used for programming tasks.

This study is being carried out by an MAI student as part of their masters dissertation with the School of Computer Science and Statistics at Trinity College Dublin.

We are primarily seeking adult participants who have some experience with either text or block-based programming languages and are familiar with basic concepts of programming logic (e.g. variables, conditionals, loops, functions).

Workshop procedure
Workshop participants will complete a series of programming tasks using the Frame-Based Editor. Once you have completed or attempted each exercise, you will be asked to upload the generated python code in the questionnaire. After the exercises, you will be asked to complete a short questionnaire about your previous programming experience and your experience with the editor.

The questionnaire is fully anonymous. All data will be stored on a secure server in an encrypted format and only members of the research team will have access to it.

The workshop will take approximately **40 minutes.**
(The programming task should take 30-35 minutes, and the questionnaire should

take 5-10 minutes.)

Anticipated risks
We do not anticipate any risks to participants.

Benefits to the participant
While participation will not benefit the participants directly, the research will support the development of better tools for programming education.

Voluntary and anonymous participation
Your participation is entirely voluntary, and you can withdraw at any time before you have submitted the questionnaire. At the end of the questionnaire, there will be an option for you to withdraw from the study and have all associated data removed. As there is no way for us (or anyone else) to identify an individual participant once they submit their data, it will no longer be possible to withdraw once you have submitted the questionnaire.

Questionnaires
All questions are optional. Feel free to omit a response to any question; however, the researcher would be grateful if all questions are responded to.

Debriefing
The researchers will be available to answer any questions about the research. If you have any queries feel free to contact illesie@tcd.ie and we will be happy to answer any questions.

Participation in analysis and publication
The data collected will be analyzed to determine how to make programming environments and supports that help people learn programming. The research results will be published in a masters dissertation at Trinity College Dublin. We also plan to publish the results of our research in academic journals and conference proceedings. We will do this in a way, that does not identify you, or any other individual participant.

Cautions about inadvertent discovery of illicit activities
While it is unlikely that illicit activities would be disclosed, if you do so, we would be obliged to report them to the appropriate authorities.

Conflicts of interest

The researchers may have made use of personal contacts in recruiting.

Contact details

The study is led by **Anna Illesi (illesie@tcd.ie)** and supervised by **Glenn Strong (glenn.strong@tcd.ie)** from the **School of Computer Science and Statistics at Trinity College Dublin**. If you have any queries, feel free to contact us and we will be happy to answer questions about the experiment.

Data Protection details

Data Controllers: Trinity College Dublin
Data Protection Officer: Data Protection Officer, Secretary's Office, Trinity College Dublin, Dublin 2 - dataprotection@tcd.ie

TRINITY COLLEGE DUBLIN
INFORMED CONSENT FORM

LEAD RESEARCHER: Anna Illesi

BACKGROUND OF RESEARCH: We have developed a frame-based editor for python, which combines the advantages of both block-based and text-based programming. You are invited to participate to help evaluate how well the editor can be used for programming tasks.

PROCEDURES OF THIS STUDY: Workshop participants will complete a series programming tasks using the Frame-Based Editor. Once you have completed or attempted each exercise, you will be asked to upload the generated python code in the questionnaire. After the exercises you will be asked to complete a short questionnaire about your previous programming experience and your experience with the editor.

The questionnaire is fully anonymous. All this data will be stored on a secure server in an encrypted format and only members of the research team will have access to it.

The workshop will take approximately 40 minutes. (The programming task should take 30-35 minutes, and the questionnaire should take 5-10 minutes.)

PUBLICATION: The research results will be published in a masters dissertation at Trinity College Dublin. We also plan to publish the results of our research in academic journals and conference proceedings.

CONFLICTS OF INTEREST: The researchers may have made use of personal contacts in recruiting. Individual results may be aggregated anonymously, and research reported on aggregate results.

DECLARATION:

- **I am 18 years or older and am competent to provide consent.**
- **I have read, or have been read to me, a document providing information about this research and this consent form. I have had the opportunity to ask questions and all my questions have been answered to my satisfaction and understand the description of the research that is being provided to me.**

- **I agree that my data is used for scientific purposes and I have no objection that my data is published in scientific publications in a way that does not reveal my identity.**
- **I understand that if I make illicit activities known, these will be reported to appropriate authorities.**
- **I understand that I may refuse to answer any question and that I may withdraw at any time without penalty.**
- **I understand that if the results of the research have been published, or my data has been fully anonymised so that it can no longer be attributed to me, then it will no longer be possible to withdraw.**
- **I understand that I may stop at any time, and have data destroyed by choosing the option to withdraw from the study at the end of the questionnaire.**
- **I freely and voluntarily agree to be part of this research study, though without prejudice to my legal and ethical rights.**
- **I understand that my participation is fully anonymous and that no personal details about me will be recorded.**
- **I understand that if I or anyone in my family has a history of epilepsy then I am proceeding at my own risk.**
- **I have received a copy of this agreement.**
- ***By selecting 'consent' I assent to continue with this survey this document, I consent to participate in this study, and consent to the data processing necessary to enable my participation and to achieve the research goals of this study.***

Statement of investigator's responsibility: I have explained the nature and purpose of this research study, the procedures to be undertaken and any risks that may be involved. I have offered to answer any questions and fully answered such questions. I believe that the participant understands my explanation and has freely given informed consent.

- ○ I consent, begin the study
- ○ I do not consent, I do not wish to participate

Exercises using the Frame-Based Editor

Please click the link below and follow the instructions to complete the programming

exercises:

https://www.pytch.org/app/frame-based-editor/

The Programming Exercises should take approximately 30-35 minutes.

Once you are finished, please copy the generated code from your solution here:

Question Tour Block 2

Questionnaire

(This will take approximately 5-10 minutes)

The next few questions are realted to your previous progamming experience.

How long have you been programming for?

- I have never programmed before
- Less than 3 months
- Between 3 months and 1 year
- Between 1 year and 2 years
- More than 2 years

Which programming languages are you most familiar with?

Please rate your confidence in doing the following programming related tasks.

	Not at all confident	Mostly not confident	Neither confident nor unconfident	Mostly confident	Absolutely confident
I know how to use the programming variables.	○	○	○	○	○
I can explain my idea of software project step by step.	○	○	○	○	○
I can solve a problem via mulitple different solutions.	○	○	○	○	○
I can investigate the knowledge that is required for solving a programming problem.	○	○	○	○	○
I could debug (correct all the errors) a long and complex program that I had written and make it work.	○	○	○	○	○
I could rewrite lengthy and confusing portions of code to be more readable and clearer.	○	○	○	○	○
I could mentally trace through the execution of a long, complex multi-file program given to me.	○	○	○	○	○
I could write a program that computes the average of any given number of numbers.	○	○	○	○	○

Block 3

Questionnaire

The next few questions are realted to your experience with the frame-based editor.

Please Rate how much you agree with the statements below.

	Strongly Disagree	Disagree	Undecided	Agree	Strongly Agree
The interaction with the frame-based editor is clear and understandable.	○	○	○	○	○
The frame-based editor offers help when I find a task difficult.	○	○	○	○	○
The frame-based editor offers all functionality I would expect from an educational code editor.	○	○	○	○	○
The frame-based editor is useful for learning to program in python.	○	○	○	○	○
Using the frame-based editor improves my abilities as a python programmer.	○	○	○	○	○
The frame-based editor helps me solve tasks more easily than text-based editors.	○	○	○	○	○

The last set of questions are open ended. We appreciate any feedback about your exepriences with the editor.

Have you faced any difficulties during the exercise? If yes, please explain.

What features (if any) did you find particularly useful?

Was there a missing feature from the editor that you would have found useful during the exercise?

Do you have any suggestions for improving the editor?

End of Survey Confirmation

If you are happy with your answers, please confirm your consent for them to be recorded. You may instead wish to exit without submitting your answers.

- ○ I am happy with my answers and wish them to be recorded.
- ○ I do not wish my responses to be recoded.

Block 4

A1.3 Ethics Application Status

TCD Research Ethics WebApp

Frame-based Python Editor

Current Status	Submission date	Last Status Update	Academic Supervisor / Lead Researcher	Application Number
Assigned to Reviewers; Pending Comments	Thursday, February 10, 2022 - 13:58	Tuesday, April 19, 2022 - 15:10	gstrong	20220206

No workflow transitions are possible at this time.

Final Comments from the Research Ethics Committee

The Research Ethics Committee have reviewed your application and requested that you make the following amendments. Please clearly highlight where you have made the changes in your revised application. Please provide a list of numbered responses to each issue and highlight in the revised proposal all changes made:

- Submit online questionnaire.
- In documentation time is estimated to be 40 minutes. On the initial section online this is stated as 30 – 35 minutes. Align to whichever you think it will be.

Status:
Assigned to Reviewers; Pending Comments

Timeline of state changes for this application

Tuesday, April 19, 2022 - 15:10

State change:
from *Submitted to REC; Awaiting Reviewer Assignment*
to ***Assigned to Reviewers; Pending Comments***

Tuesday, April 19, 2022 - 15:10

State change:
from *Submitted to Supervisor; Pending Review*
to ***Submitted to REC; Awaiting Reviewer Assignment***

Tuesday, April 19, 2022 - 15:09

State change:
from *REC Amendments Required*
to ***Submitted to Supervisor; Pending Review***

Tuesday, March 29, 2022 - 13:31

State change:
from *Commented by Reviewers; Pending REC Meeting*
to ***REC Amendments Required***

Thursday, March 24, 2022 - 21:31

State change:
from *Assigned to Reviewers; Pending Comments*
to ***Commented by Reviewers; Pending REC Meeting***

Monday, February 14, 2022 - 15:36

State change:
from *Submitted to REC; Awaiting Reviewer Assignment*
to ***Assigned to Reviewers; Pending Comments***

Friday, February 11, 2022 - 16:16

State change:
from *Submitted to Supervisor; Pending Review*
to ***Submitted to REC; Awaiting Reviewer Assignment***

Thursday, February 10, 2022 - 13:59

State change:
from *Ready for Submission to Supervisor for Review*
to ***Submitted to Supervisor; Pending Review***

Review

Thursday, February 10, 2022 - 13:59

State change:
from *Draft*
to ***Ready for Submission to Supervisor for Review***

Thursday, February 10, 2022 - 13:58

State change:
from *(creation)*
to ***Draft***

Trinity College Dublin, The University of Dublin
College Green
Dublin 2, Ireland
T: +353 1 896 1000