University of Glasgow | School of Computing Science

Honours Individual Project Dissertation

Level 4 Project: A Reflective Learning Tool for Novice Programmers

James Orr

March 21, 2024

Abstract

Self-reflection is a fundamental academic practice that is often overlooked in Computing Science education. A vast amount of research has shown the positive impact self-reflection has for novice students. For this project, I have developed a series of Jupyter Notebook extensions which allow novice programmers to practice self-reflection. A further tool was implemented to give teachers a snapshot of the overall reflections students are leaving on a specific Notebook.

An evaluation was completed to better understand how students and lectures could impact from using the tool. The results indicate that the tool was somewhat helpful for learners but did not lead to an overall statistical improvement. However, further analysis indicated that students who used the Reflective Learning Tool were able to apply their learning to new questions better than students not using the tool. A small lecturer evaluation indicated that the summarised report based on student reflections was helpful and could be used by lecturers to help improve teaching.

Education Use Consent

I hereby grant my permission for this project to be stored, distributed and shown to other University of Glasgow students and staff for educational purposes. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Signature: James Orr Date: 21 March 2024

Acknowledgements

I would like to thank a number people who made this project possible. Firstly, I would like to thank the project supervisor, Dr Jeremy Singer. Dr Singer went above and beyond to help support me with the project. His insights and guidance throughout have been pivotal to the success of the project. I would also like to give a special thanks to Dr Steve Draper who was an invaluable source of information during the research and design process.

Without the 14 participants who took part in the user evaluation, I would not have been able to understand the impact of the Reflective Learning Tool. I would like to thank the following participants: Emily Mynett, Barbara Orr, Louise Orr, Bryn Kelly, Gregory Hast, Sean Andrew, Graham Salway, Daniel Campbell, Ossian Newby, Selina Sarantila, Olivia Vaaranmaa, Christian Matheson, Mateusz Kapustka and Marta Zarantonello.

Contents

1 Introduction

1.1 Motivation

Reflection is a pivotal part of a learner's academic journey; even more so for novice learners. Capturing the imaginations of students and then keeping them engaged with a subject is often difficult. Through self-reflection, students can structure their thoughts and feelings in a way which helps rationalise their approach to learning.

Computing Science is a very challenging and often daunting subject to undertake for beginners. Many stereotypes and misconceptions can lead to students having a lower sense of belonging as discussed by Mooney et al. (2020). Furthermore, the frustration of a program not working, or the thought of having to relearn a problem which is coupled with negative emotions can be detrimental to a learner's journey. Discussed in Chapter 2, there is a distinct lack of computing-specific reflective research. Although there is a vast amount of general reflective research, there appears to be no computing tool which allows students to carry out self-reflection whilst coding. The set of self-reflective learning tools developed as part of this project allow students to mitigate these concerns. Through identifying emotions throughout lab work, and then prompting learners to reflect on their emotions, the tool gives a structured framework for novice programmers to reflect and learn from their experience to become more rounded computer scientists.

Reflective data from students can also be a valuable source of information for teachers. Through reading how their students are interacting and feeling whilst completing their work, lecturers themselves can reflect on the content they are teaching. Student data such as an overall feeling about the work, and time to complete the work is valuable feedback which students may not feel comfortable giving in a face-to-face setting. The tool allows for anonymous reflections to be relayed back to lecturers, giving the opportunity to collect raw feedback about a computing lab.

1.2 Aim

The main project aim is to facilitate reflection for beginner programmers. This has been done by the implementation of a series of reflective extensions embedded in Jupyter Notebooks. Students can reflect *in-action* and *on-action*; meaning that students can reflect whilst they are completing a specific part of a notebook and they can reflect after they have completed the notebook as a whole. This provides learners with the opportunity to build an actionable plan to help them learn more effectively and build from experience.

After discussion and input from several University of Glasgow Level 1 and 2 lecturers (see Chapter 3), it became apparent that student reflections could be used by teachers to help understand how students are getting on with course content. Specifically, the tool allows students who otherwise may not feel comfortable sharing their real feelings about a topic a mechanism to record their concerns and reflect on them.

An evaluation was conducted to understand the impact on learners using the set of extensions. The main questions that the evaluation attempted to answers were:

- Do learners perform better overall when they use the reflective learning tool compared with similar students that do not?
- How do students use the tool?
- Do students gather reflections in the same way?
- Do students using the tool preform better when faced new problems and concepts?

A small study with 14 participants was conducted, with a control and an experimental group. The independent variable was the use the Reflective Learning Tool. Participants completed a 1-month Python course. An initial quiz was compared against a post-course quiz to quantify how well the students learned. Results did not indicate a statistical difference overall, but qualitative data suggested that students found the tool helpful. However, the results did indicate that students using the Reflective Learning Tool were better at applying their learning to new concepts. Q14 and Q15 from the second quiz were not covered in the course and required participants to apply their learning onto new problems. The combined totals for Q14 and Q15 indicate a statistical difference between the two groups, p-value = 0.02.

After completing the course, four participants from the experimental group were interviewed to better understand how they used the tool. The participants used the tool in similar ways, usually selecting a negative emotion when they found a question difficult and selecting a positive emotion at the end of the Notebook. Students appeared to use the emotion selector and reflective note tool throughout. They indicated that the timer caused stress and was detrimental to their learning as they felt rushed to finish.

The other intended users are lecturers and teachers. A further evaluation was carried out to better understand if the tool would help lecturers. The main questions in the lecturer study were:

- Is it beneficial to know how the students are feeling about a Notebook?
- Would lecturers benefit from a brief summary of student reflections?
- If lecturers want a summary, how accurate is the summary?
- Can the lecturer tool help the teacher to self-reflect?

Due to time constraints, only one lecturer was able to evaluate this tool. The general feedback indicates that the tool is helpful and would be used by lecturers. Discussed in Chapter 6, the tool creates an accurate and precise summary of student reflection. The lecturer went on to suggest that they would use the tool to reflect on their lecture content and address concerns if there were common issues from the summarised report.

2 Background

This chapter builds upon the aims outlined in Chapter 1. A summary of related work is presented and discussed. The key points covered include a brief introduction to the different type of reflection, reflection within computing science, and a discussion about what technology can be used to implement a self-reflective programming tool for novice computing scientists.

2.1 Reflection: An Introduction

2.1.1 Reflection based on the experience

Education and reflection go hand-in-hand; reflection is a key aspect of education that is sometimes overlooked. There are several different approaches to reflection as an educational tool. One of which is based on Kolb's work in 1984 which looked at the learning experience and its relationship with self-reflection to help learners understand abstract concepts. Kolb (1984) developed a four-stage reflective cycle which takes learning and applies it to new experiences. This starts with a concrete experience, reflection of the experience, making conclusions, and finally applying what has been learned onto new problems. Following this, Gibbs (1988) extended Kolb's work and developed a six-stage reflective cycle. This started with an experience, students would then contemplate their thoughts and feelings about that experience, evaluate the impact of the experience, analyse to understand the situation, conclude what has been learnt from the experience, and finally devise an action plan to deal with similar situations. Both Kolb and Gibbs developed patterns that focus on trying to understand what had happened and give an opportunity for students to pick out where they went wrong themselves. The key points are that the students themselves identified issues and then evaluated them. Although Gibbs adds relevant additions to Kolb's reflective practices the fundamental rules that the student must identify and decide an appropriate response is present throughout.

The experience-based view on reflection argues that reflections allow students to understand key concepts and generalise main principles from an experience (Kolb 1984). In-order to articulate an experience, learners must have a mechanism to record their immediate reflections on an event. The use of journals and feedback prompts to help guide students has been looked at previously

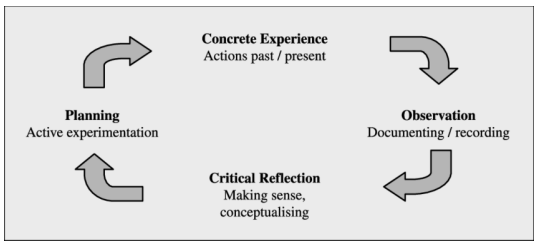


Figure 2.1: *Kolb's Experiential Learning Cycle. Source: Dummer et al. (2008) adapted from Kolb (1984)*

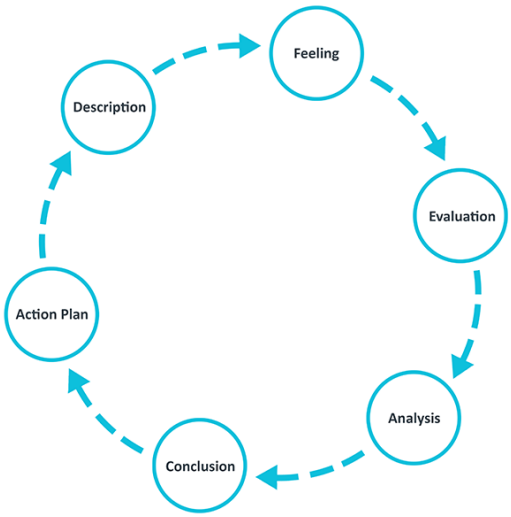


Figure 2.2: *Gibbs' Reflective Learning Cycle. Source: University of Edinburgh (2020) adapted from Gibbs 1988*

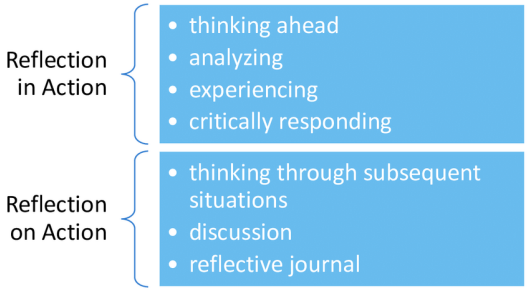


Figure 2.3: *Schon's reflection-in-action and reflection-on-action. Source: Gordon (2016) adapted from Schön (1983)*

by Roskos et al. (2001). Reflection is an important aspect of an academic journey but has been shown to be a difficult concept for beginners to understand and apply to their practices. This may have been due to a lack of well-documented information on beginner students and teachers not practicing reflection themselves(Roskos et al. (2001)). Whatever the case may be, reflection does help beginners learn at a higher level and prompts deeper inter-subject knowledge connections.

2.1.2 Reflection-in and on-action

Whilst looking at when the learner reflects, Schön (1983) argues that "reflection in-action" occurs by surprise. Reflection happens when an event unfolds unexpectedly. The immediate reflection from an unexpected result can then be built upon further by "reflecting after the event" as the student attempts to understand the foreign concept and adopt an actionable strategy to deal with similar situations in the future. Schön's (1987) other major book builds upon these ideas further, discussing the concept on "reflection on-action". Schön's reflective framework prompts students to reflect as events happen and reflect afterwards. His arguments are that reflection *in* and *on-action* should be the basis of a new shareable body of knowledge which students can use to learn from their lived experiences.

Capturing an *in-action* reflection requires a system which users can quickly note their immediate feelings and reflective comments. This would need to be usable and accessible quickly so the student can act immediately. Further system functionality would allow users to reflect *on-action* after the experience has taken place. Prompting the user to reflect on what they would do differently if they had to complete similar tasks. Moreover, the system would prompt a user to process how they might use new information gained from the experience when faced with similar situations in the future.

2.2 Education and Experience

Work by John Dewey (1998), Education and Experience, looks at the deep connection between experienced events and their relationship with education. Noting the delicate connection between a positive experience which may arouse curiosity, and the desire to continue their studies further. Dewey's view shows the danger of deterring a potential student from pursuing further research within the subject because of a negative experience. This could perhaps be when an exercise was too challenging and advanced that the student was completely lost, turning them away from the feeling of curiosity and warmness to a new subject. If experience "arouses curiosity, strengths initiative; and sets up desires and purposes" it leads to a positive outlook and carries the student further in the field when they may otherwise drop off. In-order to maintain students attention each present experience should have a purpose or else they become redundant. The aim is for students to reach an academic maturity to the extent where they understand their current learning will have a favourable effect on future practice.

Applying Dewey's ideas we see that the learning experience does not ever end for some; "education as growth" shows that education should be continuous and inter-disciplinary. By introducing reflective concepts in one subject, students may apply reflective practices to their other studies. Overall, leading to better rounded students which have reached academic maturity .

2.3 Self-assessed Learning

Reflections are closely linked to self-assessment and both terms are often used interchangeably. Definitions of self-assessment describe it as a process were learners evaluate their own progress in a subject area and compare it against a set of standards to which they aspire (Andrade and Valtcheva (2009)). Self-assessed learning is not a new concept and is applied throughout academic institutes. Students generally have a positive attitude toward self-assessment (Andrade and Du (2007)), mainly due to the perceived benefit of actively critiquing their own work. Suggesting that students active engagement with self-reflection may be negatively impacted if the reflective tool does not pose a positive impact to their learning.

As discussed by Hunt (1982), student self-assessment during the "learning stage" can be used to indicate the level at which the students are learning. During his 1982 study, Hunt looked at the effects of dynamic feedback from students during a lecturer. This was distributed via a selection of buttons representing different levels of sureness which students were prompted to press at various stages during lectures. Allowing the students to keep a record of how "sure" they felt about a concept or answer and gave an indication to the lecturer as to how comfortable the class were about the new material. The results from the study showed that there was a higher level of learning with the self-assessed group of participants, and that there was an even higher acquisition of knowledge when students completed self-assessment after answering questions on the material. Hunt's work indicates that students learn better when they are asked to reflect on how "sure" they feel after completing questions. Furthermore, there appears to have been no

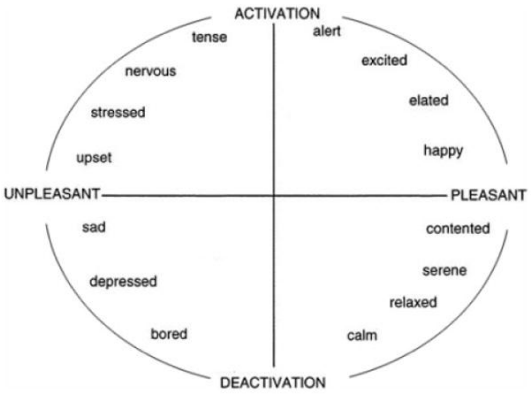


Figure 2.4: *A graphical representation of the circumplex model of affect with the horizontal axis representing the valence dimension and the vertical axis representing the arousal or activation dimension.Russell et al. (2005).*

negative aspect to how the participants used the tool compared with the control group. The tool was easy to understand and simple to use so the students could apply self-assessment without it hindering the actual learning.

2.4 Emotions as Reflective Parameters

Emotions are the basic classification of experienced feelings; they categorise complex feelings into single words. There are a variety of emotion lists with different classifications and lengths. However, the most popular list, often known as "The Big Six", was developed by Ekman et al. (1969). These universal emotions are: happiness, surprise, fear, sadness, disgust and anger.

Building from Ekman's work, Russell et al. (2005) developed a binary emotion model shown as Figure 2.4. Russell's model represents emotions in two-dimensions: Arousal, as activation and deactivation, and Valence, as pleasantness and unpleasantness. The arousal dimension refers to how excited or apathetic an emotion is for an individual. Valence refers to emotions which evoke positive or negative experiences. Emotions can be used to quickly and effectively identify a complex feeling which can then be used as a basis to reflect upon.

2.5 Reflection in Computing Science

Discussed by Collins and Brown (1988), the computer itself is an inherently reflective machine as they naturally keep track of actions used to complete a given task. Furthermore, reflective practice is common practice in the software profession with code-review a standard practice for software teams. It is however a skill which is sometimes overlooked by novice computer scientists. The benefits of reflection has been shown to improve computing students' understanding of concepts and assist in problem solving (Chng (2018)). Specifically, journals are helpful for students to inform lecturers on difficulties where students are uncomfortable speaking with them face-to-face.

Building from Schön (1983), Boyle (2004) looks at how reflective learning can be applied to computing science. Boyle adopted a blended approach to learning new coding languages. His research has shown that visible and graphical outputs lead to marked improvements of how well students learnt new coding languages. Throughout the study, Boyle noted that how to teach students "new abstractions" such as new languages is a particularly vacant area in the broader

Computer Science education sphere. Simply put, Boyle believes that we do not know how to teach coding to novice programmers. The challenge with learning a new language is that the task is too vast and expansive to accurately argue that there is a particular technique or way to go about learning. Each student's learning journey is different and how they learn is unique, it is therefore important to acknowledge that novice programmers require time for them to develop their own way of learning. This could be achieved through the application of self-reflective practices when novice programmers learn a new coding language.

Reflection can also produce valuable feedback from a student. Student feedback is referenced by Fincher and Robins (2019) as an important element of evaluating of teaching. In the context of Computing Science, this suggests that lecturers can use student feedback on their experience with course material as a valuable metric. Fincher and Robins (2019) further discuss how self-reflection from a teacher can be evoked by student feedback. Students feedback can be used alongside self-reflection from a teacher to help improve a Computing Science course. There is, however, a danger to over rely on student feedback as there may be a number of students who contribute to a softening of the teaching because they are not as academically invested as the teacher.

2.6 Beginner Friendly Technologies

Selecting the appropriate technology for novice programmers is an important and difficult task. Tools like Scratch have been shown to be good starting points for beginner programmers (Sim and Lau (2018)). The drag and drop nature allows students to see high level concepts in action without the need for accurate syntax. Although, Scratch is often used in high school and primary school settings so may not be appropriate for university level novices. Active learning through online tools is one of the more popular methods to help novice programmers get started (Sim and Lau (2018)). We must also take into consideration the coding language the user may encounter. Python is one the most common language for beginners (Khoirom et al. (2020)), so an IDE which has a beginner friendly interface, is extendable, and can compile Python would be an ideal starting point for a reflective tool extension. Furthermore, a blended approach to learning a coding language with more emphasis on a sequence of programs producing visible graphical outputs has been shown to have marked improvements of up to 23% compared to student who did not complete a blended approach (Boyle (2003)).

2.6.1 Jupyter Notebook

Jupyter Notebook is the most used system for interactive literate programming (Shen (2014)). Designed to make data analysis easier to share, document, and replicate, Jupyter Notebook is an online IDE which supports a variety of languages such as Julia, R, Javascript, C and Python (Pimentel et al. (2019)). Jupyter Notebook's most common use it to support Interactive Python Notebooks (.ipynb files). These allow users to insert Markdown text based cells and Python cells within a .ipynb file.

First coined by Knuth (1984), *literate programming*, relates to combing code and natural language to allow developers to state the thoughts underneath a program's logic. Intuitively, the design allows parts of a Notebook to be compiled with an immediate visual representation of results and text. Notebooks can also be easily employed to deliver tutorial or lab content to students due to its editable nature and east to use interface. These factors make Jupyter Notebook a highly appealing IDE to initially start with for beginner programmers. It is already widely used across many academic institutions such as the University of Glasgow.

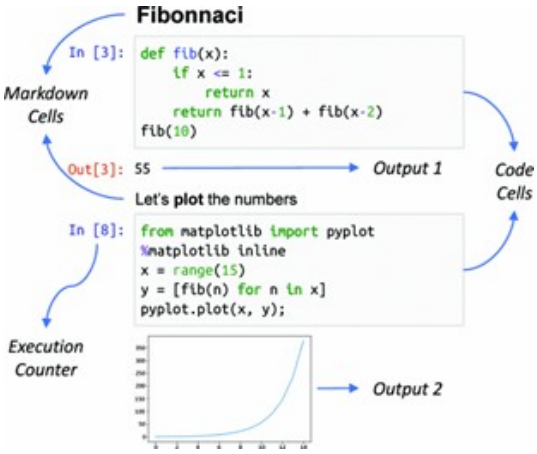


Figure 2.5: *Labeled Example of a Jupyter Notebook from Li et al. (2023)*

2.6.2 NBExtension

Jupyter Notebook does not offer official support for creating extensions or plug-ins. There is however an unofficial Jupyter Notebook extension framework known an NBExtensions, Jupyter-contrib project contributors (accessed 2024). The unofficial Jupyter Notebook extensions are currently an open source GitHub repository and can be installed following a set of instructions found on their website. For the purposes of creating a reflective tool extension, using NBExtension allows the tool to be created iteratively and allows the tool to be distributed after development.

2.7 Summary

Self-reflection is a vast academic field which has been shown to help improve students' learning. There are a number of reflective cycles and practices which can be applied to a student's learning journey. The practices all stem from a lived experience and build action plans based on feeling recall. Although there are many self-reflective techniques and practices there is a lack of tool to support self-reflection for novice programmers. A reflective learning tool should be able to support students to reflect *in* and *on-action*, as well as guiding students on a reflective cycle after they have completed the lab. Furthermore, the tool should be an integrated coding accompaniment and not break the student focus when completing computing labs. Jupyter Notebook is an ideal framework to build the reflective extension.

3 Analysis

In this chapter we discuss the requirements that the reflective learning tool should fulfil to meet the aims of the project laid out in Chapter 1. The requirements build upon the research discussed in Chapter 2. Furthermore, after creating several user stories and consulting various Glasgow University Computing Science lecturers we identified the main stakeholders of the tool as the learner, educator, and creator of the resource.

3.1 User stories

3.1.1 User Story 1: Sara, Novice Computing Science Student

Sara is a Level 1 Computing Science student at the University of Glasgow. She has no prior experience with any programming language or related concepts. She is doing a course which teaches beginner level Python. Sara lacks confidence to ask for support in her labs as she is worried about being judged. She is looking for a tool where she can review her previous work and try to learn from it. As she is a beginner she may get confused with a complex design.

3.1.2 User Story 2: John, Level 2 Computing Science Student

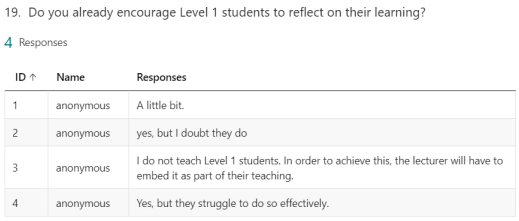
John is a Level 2 student at the University of Glasgow. He is currently learning Java for the first time. Last year he learnt Python as his first language with the help of the Reflective Learning Tool. John would like to use some of the same techniques he used to learn Python and apply them to new languages he is learning now.

3.1.3 User Story 3: Prof. Smith, Computing Science Lecturer

Prof. Smith is concerned that her Level 1 beginner students are not engaging with the lab material she has set. She would like to know how long students are spending on labs and how they are feeling after completing them. Prof. Smith wants to know if there are any questions or topics that students are struggling with so she can help.

3.2 Lecturer Survey

During the early development stage I created a short video outlining a basic prototype and asked a number of Level 1 Computing Science lecturers at the University of Glasgow to fill out a short questionnaire after watching. The responses helped inform how lecturers and tutors may benefit from having access to reflective information such as average time spent, and overall lab work feedback from the students. Responses showed that lecturers would like to know the average times students spent on the lab, and they would like to know how the students feel about the content after the complete it. The lecturers all voted to receive this information via a .csv file which can be accessed via Microsoft Excel as spreadsheet data. One key suggestion was to use an AI tool to summarise the feedback for a specific lab, this would prevent lecturers from having to read


19. Do you already encourage Level 1 students to reflect on their learning?

4 Responses

ID ↑	Name	Responses
1	anonymous	A little bit.
2	anonymous	yes, but I doubt they do
3	anonymous	I do not teach Level 1 students. In order to achieve this, the lecturer will have to embed it as part of their teaching.
4	anonymous	Yes, but they struggle to do so effectively.

Figure 3.1: *Do lecturers encourage reflection already?*

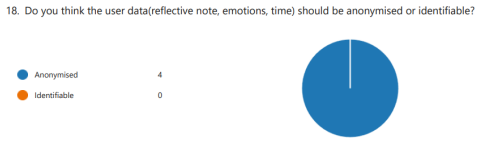


Figure 3.2: *Should the data be identifiable?*

through potentially hundreds of responses. The feedback also included some critiques of certain features; one lecturer suggested that the timer may cause more stress and could lead to a lower quality of work if the students are just trying to complete the tasks quickly but not necessarily to a high level. Furthermore, the lecturers all agreed that the data should be anonymised as the identity of each response is not relevant for a summary of how the students are feeling about each lab overall.

3.3 Requirements

Requirements can be split into two group; functional requirements relate to features that a user should be able to perform with the Reflective Learning Tool, and non-functional requirements relate to quality constraints that the system should meet overall. The MoSCoW priority method has been used to indicate the importance of each requirement.

3.3.1 Functional Requirements

1. Beginner-level tool which incorporates a simple interface. - **Must**
2. Deployment on Jupyter Notebook. -**Must**
3. Reflective note. - **Should**
    a. This will be a markdown cell which a user can input text.
    b. There will be a text prompt to help the user reflective on their experience.
4. Emotion tracking. - **Should**
    a. The user will be prompted to select from a drop-down menu of preset emotions.
    b. The user will be able to save their response.

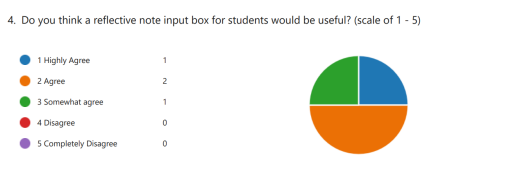


Figure 3.3: *Do lecturers think a reflect input would be useful for students?*

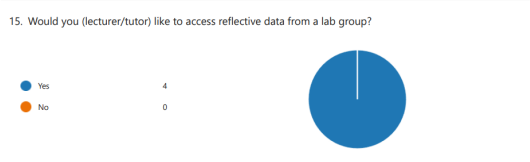


Figure 3.4: *Would lecturers want to have access to student's reflective inputs?*

5. Time tracking. - **Should**
    a. There will be a stopwatch at the top of the notebook page.
    b. The stopwatch will have a start, reset and save function.
6. The tool should guide the user through a reflective cycle based on their inputs after the student has completed the lab session. - **Could**
7. The data will be available to lecturers and tutors. - **Would**
    a. The information from a specific lab will be available as a .csv file.
    b. The reflective feedback for a lab will be summarised for tutors and lecturers.
    c. General information like the average time spent on the lab will be available for tutors and lecturers.

3.3.2 Non-Functional Requirements

1. The tool will be integrated with the standard Jupyter Notebook interface. - **Must**
2. Users will be able to revisit their reflective responses after completing the lab. - **Must**
3. Users will be able to upload their reflective data within 5 seconds after entering. - **Should**
4. Lecturers will be able to generate the reflective summary and accompanying .csv file within 5 seconds after selection. - **Should**
5. The implementation should support at least 200 user, the average size of a Glasgow University Level 1 computing class. - **Could**
6. The reflective data should be stored securely to avoid data breaches. - **Would**

3.4 Summary

This chapter has discussed the process carried out to create the final set of functional and non-functional requirements. Building from Chapter 2, I developed a set of user stories to better understand the users of the tool. Furthermore, after identify the key stakeholders as learners, teachers and creators of the Notebooks, I ran a lecturer survey to identify what teachers would want from the tool. A final set of requirements was constructed to meet the needs of both learners and teachers

4 Design

This chapter will discuss the high-level design of the reflective tool. I will discuss how the Jupyter Notebook extensions will appear and what they will do. There are two main intended users; learners and lecturers. Below I have discussed the design of the extensions in relation to the learner users and the lecturer users. Several figures are used to display the initial wireframes.

4.1 Learner Tool Bar Extensions

The main components of the tool will be the reflective tool bar buttons which users can use to help perform self-reflection whilst completing a Jupyter Notebook lab. The full set of extension wireframes on the Notebook toolbar can be seen by Figure 4.1. As laid out in the analysis and requirements overview, the extension should be simple to use and easy to understand. Each of the buttons will require an on-hover prompt to remind the user what they do, the choice of icon will also be important to help the students identify which button they are looking for.

4.1.1 Timer

The timer, as seen in Figure 4.2 will require a start and save button. In order to minimise risks that students could accidentally pause the timer and submit a time which is not accurate, I have decided to not adopt a pause mechanism. Instead, there will be an option to reset the timer back to its start mode and time (00:00:00). The timer will be visible to the user to help the user understand how long they are spending on a task and the Notebook overall. However, in future iterations the timer could be hidden from view and automatically start when the user opens the file if the timer is becoming more of a hindrance to students than it is a benefit. This may be the case if students start getting competitive with the timer and completing lab work becomes a race, not a helpful practical resource to reinforce lecture material.

4.1.2 Reflective Note

A reflective note button (Figure 4.3) would allow students to keep a local diary-like log of immediate reflections throughout their Notebook. Discussed in the background and analysis chapters, reflections are often immediate and fleeting, so we require a feature where students can quickly capture their reflection in a short note based form relative to which part of the Notebook they are working on. The feature is intended to be used several times throughout the Notebook so saving all of the reflective notes will not be beneficial to the lecturers - an overall reflective response based on the lab as a whole will be relevant to the lecturers. A text prompt will be required to display alongside the input cell to remind the user of the purpose of the reflective note

4.1.3 Emotion Selector

The emotion selector, shown in Figure 4.4, will go alongside the other extensions in the tool bar as a drop-down selection of potential emotions. This would allow lectures to quantify the different and similar emotions students are feeling throughout each lab. Each emotion change

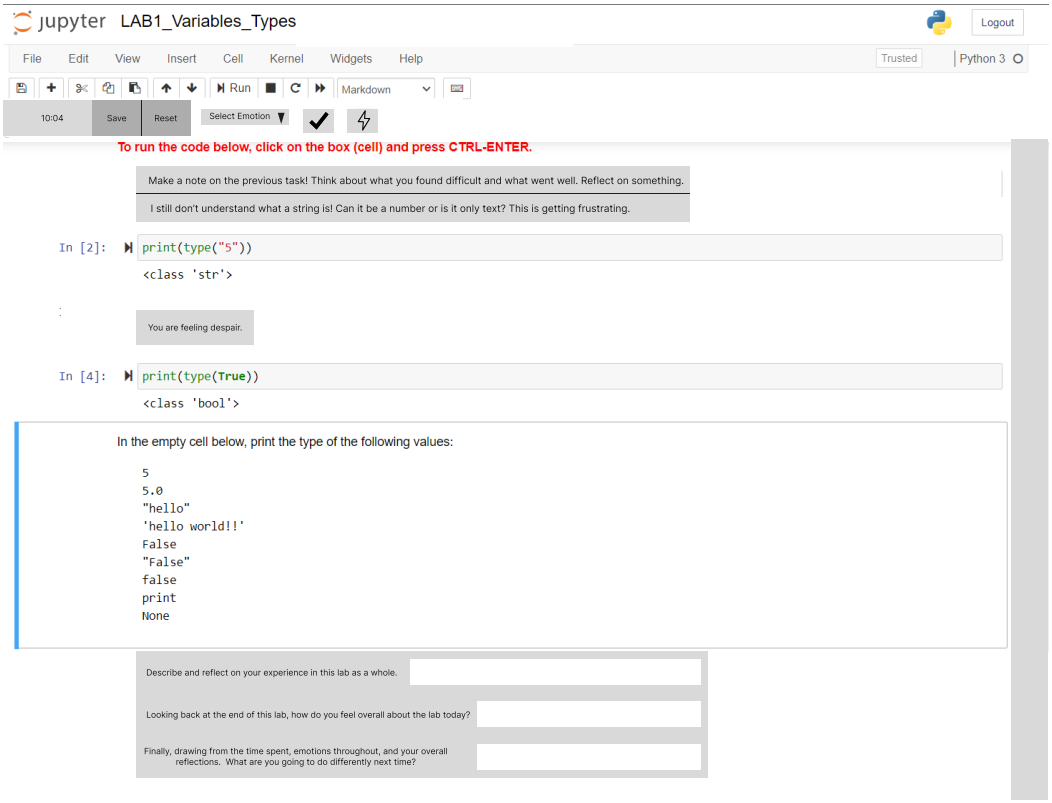


Figure 4.1: *Jupyter Notebook toolbar with full set of learner extension wireframes.*

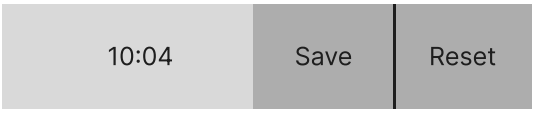


Figure 4.2: *Timer Wireframe*

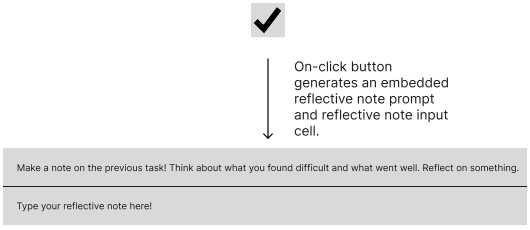


Figure 4.3: *Reflective Note Wireframe*

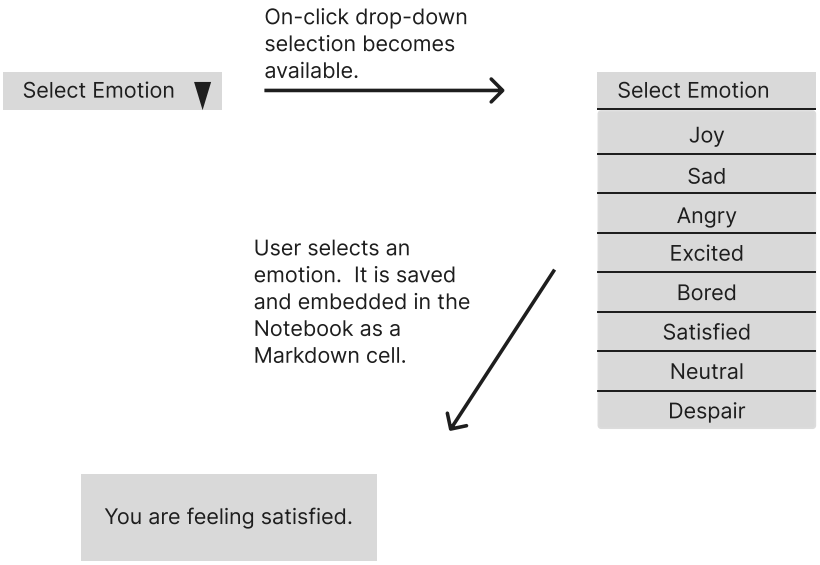


Figure 4.4: *Emotion Selector Wireframe*

will therefore require a save functionality so they can be accessed when lecturers generate the summaries. Based on discussions with a lecturer in the School of Psychology at Glasgow University and analysis of previous literature, I have decided on the following list of emotions the users can choose from a drop-down selection: Joy, Sad, Angry, Excited, Bored, Satisfied, Neutral, and Despair.

4.1.4 Reflective Journey

As identified previously, self-reflection can be only be completed after the experience has taken place. Based on the immediate feelings and notes recorded during a lab session, users will be prompted to carryout a reflective cycle at the end of the labs. This will be triggered by selecting the end of Notebook reflection button, shown by Figure 4.5. After selecting, the user will be prompted to summarise their feelings and reflect on how they found the lab as a whole. This piece of user input will be saved to a database and used to generate overall reflective feedback for the lecturers. The user will then be prompted to apply Gibbs' (1988) reflective cycle onto their own lab experience. Their responses are saved locally in the .ipynb file so can be accessed at a later date by students.

4.2 Lecturer Tool Bar Extension

The lecturer functionality will be used by experienced users so will not require as much focus on simplicity. Similar to the learner toolbar extensions, the lecturer functionality will be triggered via a Jupyter Notebook button. The accompanying wireframe shown by Figure 4.6 shows an example use case and summary report from a fictional *lab 7*. Upon selection, the lecturer will be required to input a specific lab number, they will be given a summary of the reflective feedback from that lab, the average time taken to complete the lab, and all of the raw data (emotions, timers, reflective feedback notes) as part of a .csv file.

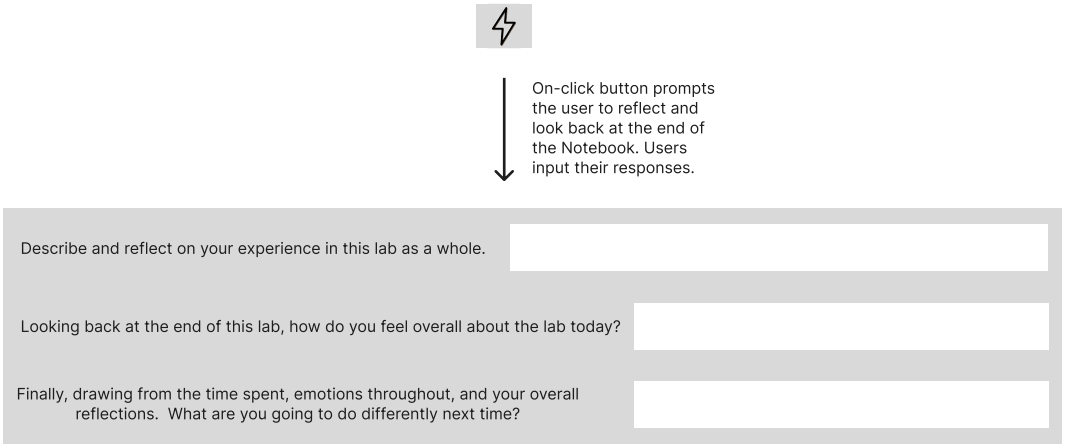


Figure 4.5: *End of Lab Reflections Wireframe*

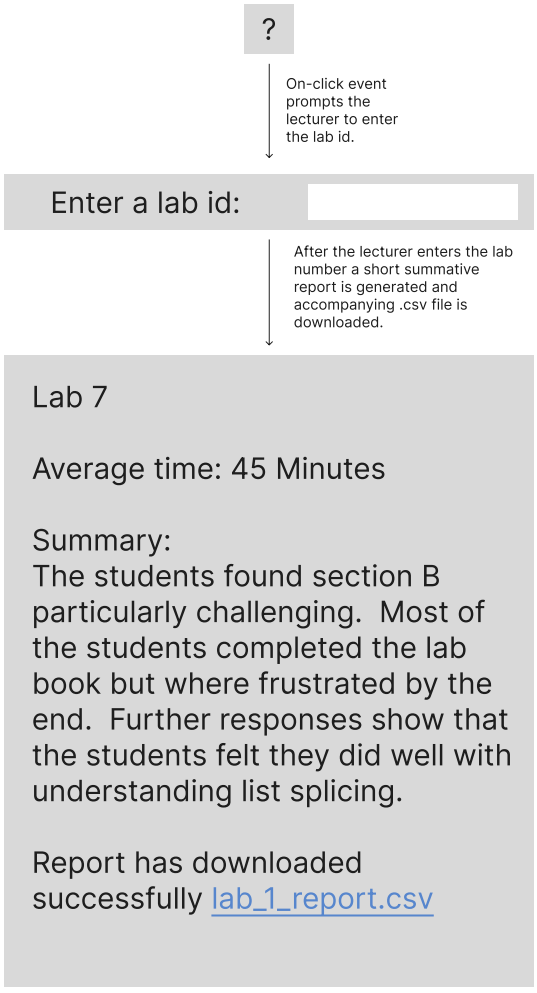


Figure 4.6: *Lecturer Report Walkthrough Wireframe*

4.2.1 AI Summary

As noted by a lecturer during the survey outlined in Chapter 3, it can often be tedious for teachers to scroll through potentially hundreds of individual pieces of feedback to get a feel for how the students are feeling about a lab. In order to mitigate this, I will implement a large language model (LLM) API to summarise all of the reflective feedback for each specific lab. This will allow the lecturer to get a short but accurate representation of the overall feeling the students have about a lab as a whole. There are many common LLMs currently available which could be used, for the purposes of this project I will be looking for an in-expense and scalable solution. The following implementation chapter (Chapter 5) will discuss this in more detail.

4.2.2 Report Generation

Alongside the AI summary, a report summarising a specific lab will be required for lecturers. The report will consist of an average time taken to complete the lab and the raw data from the database. This will allow lecturers to get all the relevant data captured by the reflective tool. Each time the lecturer selects this button, the raw data from the lab will be automatically downloaded as three .csv files into the current directory. This allows them to open the data in spreadsheet applications like Microsoft Excel for their own analysis. The raw data can be used to verify the AI generated summary if required.

4.3 Summary

This chapter has discussed the high level design for the reflective tool. Outlined, are four learner extensions which allow students to reflect *in* and *on-action*, and one teacher tool which allows lecturers to see how a lab group are coping with a Notebook. Accompanying wireframes, help visualise the functionality and layout of the extensions. Notabley, Figure 4.1, outlines an overall view of how a learner's Jupyter Notebook toolbar will appear after the tool has been installed.

5 Implementation

The chapter includes an overview of the extensions created for both learners and teachers. I discuss the general template for Jupyter Notebook extensions and how I created them. Further detailed about system intrinsic functions like the use of Magic commands are discussed as well. A number of listings and figures help outline the implementation. Technical achievements such as the incorporation of a LLM to summarise data and Firestore database interactions are also explained. The tool is made up Javascript and Python code, with roughly 550 lines of code.

5.1 Jupyter Notebook Extensions

The full reflective tool is made up of five Jupyter Notebook extensions. Learners can make use of four reflective extensions; a timer, reflective notes tool, emotion selector and an end-of-lab reflective walk-through. Teachers are prompted to use the lecturer toolbar extension which generates a summary of all the data from a given lab. Each extension can be added individually through manual installation. Below is an overview of the functionality and use cases for each extension developed as part of the Reflective Learning Tool.

5.1.1 Timer

The timer can be viewed in Figure 5.3, denoted by the red border. Once installed learners can start, reset and save the time spent on a lab. The start button will begin the timer whilst the reset option restarts the timer back to 00:00:00. At the end of the Notebook users are expected to select the save button which prompts the user to enter the relevant lab ID number. The timer information is saved to the database and can be viewed in the Notebook (.ipynb) file.

5.1.2 Reflective Note

Shown by Figure 5.3, the reflective note example is highlighted by a yellow border. The functionality allows users to click the button and a Markdown input cell is added to the learner's Notebook. Learner's are prompted to add an inline reflective note about their immediate experience. The note is saved as part of the Notebook and can be accessed by students when they re-open the .ipynb file. An on-hover message displays to help users remember the purpose of the button.

5.1.3 Emotion Selector

The emotion selector can be seen in Figure 5.4, highlighted with a green border. Users are prompted with a selection of emotions after they click the Select Emotion button. After learners click an emotion it is displayed in the Notebook and saved to the database after a lab ID is inputted by the learner.

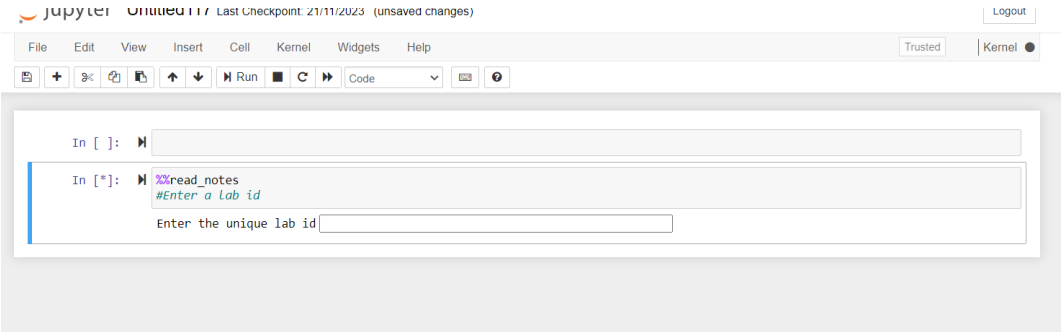


Figure 5.1: *After clicking the teacher lecturer report button denoted by a question mark, teachers are prompted to input the relevant lab ID.*

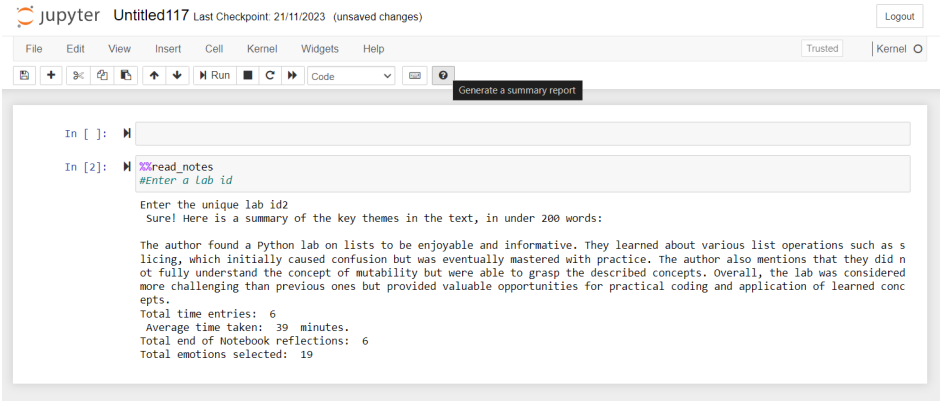


Figure 5.2: *An AI summary is generated for teachers to read a summarised snapshot of the learners overall reflections. A further three excel spreadsheets are created in the teacher's directory which contain the raw database entries for emotions, overall reflections and times that students inputted for the specific lab.*

5.1.4 End-of-lab Reflective Walk-through

Learners are prompted to end their lab Notebook by selecting the end-of-lab reflection button from the tool bar. Shown by a dark blue border in Figure 5.4, the extension prompts users to reflective on the lab as a whole. Making use of the reflections gathered along the way, the extension prompts the user to recap their feelings and overall thoughts about the Notebook. The overall reflection is recorded to the database after the user successfully complete the end-of-lab walk-though. Similar to the other learner extensions, the inputs are saved to the Notebook as well as the database so learners can revisit them.

5.1.5 Lecturer Reports and Summary

Shown by Figures 5.1 and 5.2, the lecturer toolbar button will generate a summary of learners overall reflections. This allows teachers to get a shortened summary of key themes. The teacher is told the average time taken by students as well as the total end-of-lab reflections given by students. Three Excel files denoting the raw emotion, reflections, and time data from students will be downloaded to the current working directory. This allows lecturers the opportunity to see specific comments and times from their students. Further information such as submission timestamps are also visible in the Excel spreadsheets.

```
var exampleButton = function () {
   Jupyter.toolbar.add_buttons_group([
      Jupyter.keyboard_manager.actions.register ({
         'help': 'Informative message about button functionality', //On-hover
             message
         'icon' : 'fa-bolt', //Button symbol
         'handler': functionCallExample //On-click event
      },)
   ])
}
exampleButton();
```

Listing 5.1: *Jupyter Notebook Button Template*

5.2 Front-end

Each Notebook extension's source code is saved in its own folder; the folders contain at least a main.js file and a description.yml file. Main.js handles the front end view whilst description.yml describes the extension. The JavaScript front-end files follow the template structure from Listing 5.1. NBExtensions make use of functions such as `insert_cell_below('celltype')`, and `execute-cell()` to manipulate the .ipynb files. A detailed example shown by Listing 5.2, shows the front-end design for the embedded reflective note extension. The examples show how the main.js files were used to indicate the icon and on-hover help message. As well, the example shows how further framework specific functions like `Jupyter.notebook.select_next();` and `set_text("Here is some text");` were used to add basic functionality to the extensions.

All the buttons have some functionality after they are clicked. The reflective note button is the only extension which does not read or write to the database. After clicking the reflective note button, an embedded Markdown cell with text is added to the Notebook, there is an accompanying Markdown cell were users are asked to enter their reflective note. This functionality is strictly local to the Notebook and has been implemented using only JavaScript code. The other extensions make use of server side Python function executions to complete more complex tasks such as reading and writing from the database.

5.3 Installation

Each extension is installed individually as per standard NBExtension guidelines from Jupyter-contrib project contributors (accessed 2024). The tool is not currently available as part of a package, meaning that each Notebook extension must be installed and then enabled manually on a users machine. Installation is completed through the Anaconda Prompt for Windows users or the relevant terminal directory location for Mac users. The terminal prompt directory should point to the location of the extension folder. The following will install and enable a Jupyter Notebook extension:

```
jupyter nbextension install <nbextension folder> --user

jupyter nbextension enable <nbextension folder/main>
```

Further detail about installation can be found in the accompanying ReadMe.md and Manual.md

```
var insertDescriptionCell = function() {
   Jupyter.notebook.
   insert_cell_below('markdown').
   set_text("Make a note on the previous task! Think about what you found
       difficult and what went well. Reflect on something");
   Jupyter.notebook.select_next();
   Jupyter.notebook.execute_cell();
   insertInputCell()
  };

var insertInputCell= function() {
   Jupyter.notebook.
   insert_cell_below('markdown').
   set_text("Type your reflective note here!");
   Jupyter.notebook.select_next();
  };

var reflectiveNoteButton = function () {
   Jupyter.toolbar.add_buttons_group([
      Jupyter.keyboard_manager.actions.register ({
         'help': 'Add note for self reflection',
         'icon' : 'fa-check',
         'handler': insertDescriptionCell
      },)
   ])
}
reflectiveNoteButton();
```

Listing 5.2: *Reflective Note JavaScript cell creation functions*

```
var executeReflectiveTool = function() {
   Jupyter.notebook.
   insert_cell_below('code').
   set_text("%load_ext autoreload\n%autoreload 2\n%run reflective_tool");
   Jupyter.notebook.select_next();
   Jupyter.notebook.execute_cell();
   // Execute Magic in Jupyter cell so we can call
   // call functions from reflective_tool.py
   // from the user's Jupyter server.
   // Then hide the current cell.
   Jupyter.notebook.get_selected_cell().element.hide();
   getUserInput()
  };

  var getUserInput= function() {
   Jupyter.notebook.
   insert_cell_below('code').
   // Call end_lab_reflection() from reflective_tool.py
   set_text("%%end_lab_reflection\n#Describe your experience");
   Jupyter.notebook.select_next();
   Jupyter.notebook.execute_cell();
   };

  // Add Toolbar button
  var endReflectionButton = function () {
     Jupyter.toolbar.add_buttons_group([
        Jupyter.keyboard_manager.actions.register ({
           'help': 'End of Notebook reflection',
           'icon' : 'fa-bolt',
           'handler': executeReflectiveTool
        },)
     ])
  }
// Run on start
endReflection();
```

Listing 5.3: *End of lab reflection JavaScript code - shows interaction with reflectiveTool.py and use of Magic*

Figure 5.3: *Timer running and embedded reflective note example*

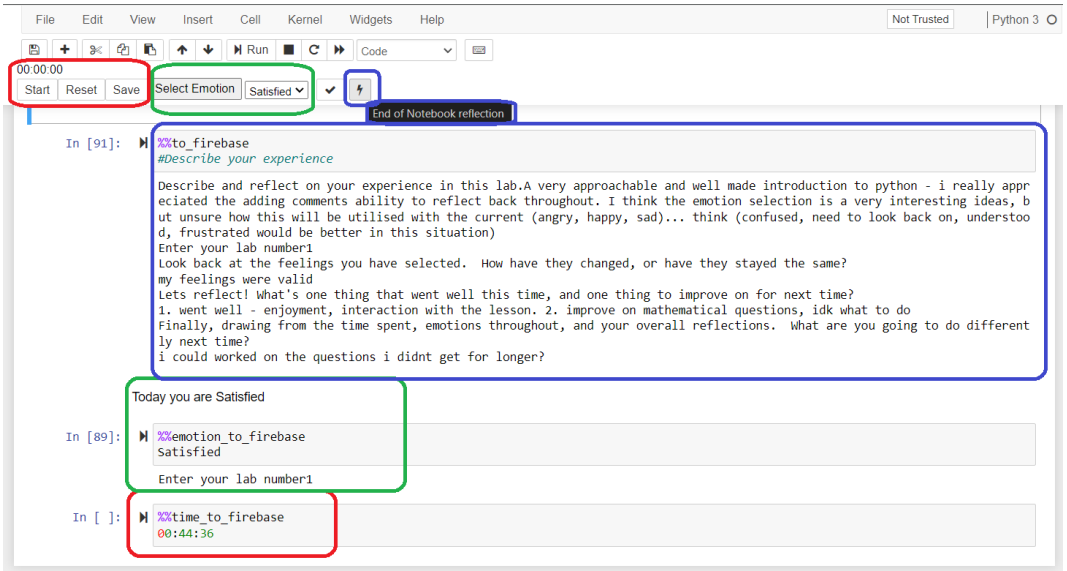


Figure 5.4: *End of lab book example. Timer, emotion, and end of lab reflections have been submitted.*

```
def generate_summary(reflective_data,lab):
   llama2_13b =
       "meta/llama-2-13b-chat:f4e2de70d66816a838a89eeeb621910adffb0dd0baba3976c96980970978018d"
   os.environ["REPLICATE_API_TOKEN"] = "r8_GnYKyT4fcwVrYH3bGc6nVuMJ0VZKDhX0LfmpR"
   pre_prompt = "In a third-party formal tone, make a summary of the key themes
       in the following text. Keep it below 200 words please."
   output = replicate.run(
   llama2_13b,
   input={"prompt": (pre_prompt+reflective_data), "max_new_tokens":1000}
   )
   print ("".join(output))
   FirebaseExtension.time_summary(lab)
```

Listing 5.4: *Llama 2 API call to generate a summary reflective notes.*

files.

5.4 System Intrinsic Functions

Figure 5.6 outlines the interactions and data flow that may occur when students and teachers use the Reflective Learning Tool. Outlined below are the intrinsic functions which allow the tool to complete complex tasks. Jupyter Notebook does not allow JavaScript extensions to access data from Markdown or Code cells. In-order to use a student's input I employed Magic commands to garner inputs local to the Notebook and employ functionality through Jupyter server side Python code execution. This means that all student interactions with the Reflective Learning Tool is only available within the Notebook, but by employing Magic commands I have successfully created the ability to read and write the user inputted reflective data with a Firestore database.

5.4.1 Magic

The extensions make use of a Python class to execute system deliverables such as saving reflective responses to a database, external API usages, and more complex reflective features. I achieved this by using Magic commands, further details can be found from IPython Development Team (accessed 2024). Magic commands are used to complete server-side computations invoked from the Notebook. They allow users to execute functions from different Python files within their own Notebook. Magic function calls are embedded in plain text Notebook cells via `set_text("");`, an example can be found in Listing 5.3. Each extension is effectively a cosmetic front; the heavy lifting is done via Python functions found in reflectiveTool.py.

5.4.2 Llama 2 - AI Summary

The teacher extension makes use of a large language model (LLM), Llama 2, to help summarise the students overall reflections. The LLAMA Development Team (accessed 2024), open source LLM, Llama 2, is free for research and commercial use. A Python API call takes place when the Magic command calls `generate_summary(reflective_data, lab)` executed from the Jupyter Server, this function is dentoted by Listing 5.4. Further details of are shown in the accompanying sequence diagram, Figure 5.6. Alongside the raw reflective data, a prompt is added to the full text passed to the LLM. The prompt informs the LLM that I want to get a summary under 200 words of the following reflective data; the prompt can be changed as required.

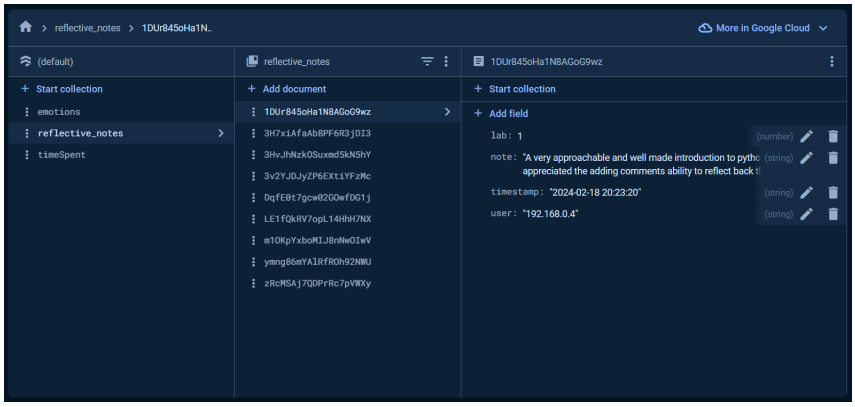


Figure 5.5: *Example overall reflective note database entry*

5.5 Firestore Database

The database used to save each reflective parameter is a standard, free to use Firestore database. There are three collections of information: emotions, reflective notes and time spent. Each document saved in the collections includes the user inputted lab number, a timestamp (YYYY-MM-DD 00:00), user IP address, and accompanying raw data like the emotion selcted, time, or overall reflection. An example collection is shown by Figure 5.5. The extensions read and write to the database through Magic executed Python code. Furthermore, Firestore offers Identity and Access Managment (IAM) to help manage databases as standard. This would allow the database to be secure and scalable in the future. Further documentation can be found through Google (2024).

Python functions from refleciveTool.py, access the Firestore database using the corresponding database project JSON file. The file includes project information such as the private key and ID used to gain access to the database. The implementation does not encrypt or hash this information, making it vulnerable to attacks. As the tool required further focus on functionality I was unable to securely implement a solution which mitigated a database access credentials leak.

5.6 Summary

This chapter has outlined the implementation detail for the series of extensions which make up the Reflective Learning Tool. Each extension included a JavaScript front-end, created under the Notebook extension structure outlined by Jupyter-contrib project contributors (accessed 2024). Magic commands are used to execute Python functions for four out of the five extensions. These include API calls and database interactions. A working back end Firestore database has been setup to save student responses. The lecturer extension accesses relevant entries from the database to generate a summarised report and accompanying Excel spreadsheets of raw reflective data.

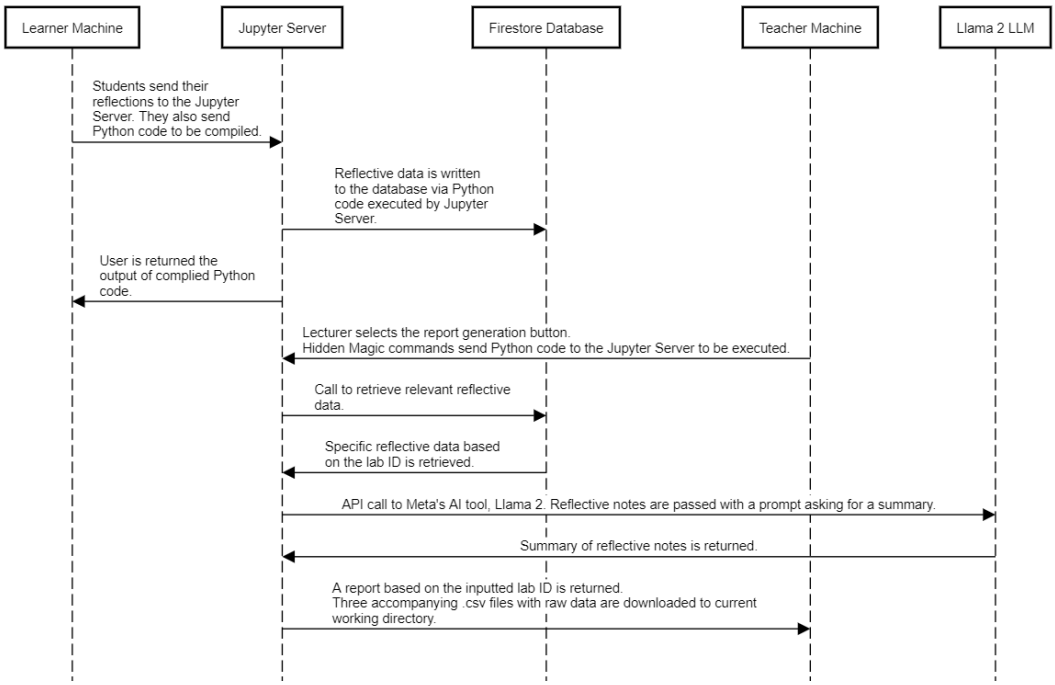


Figure 5.6: *Firestore Database Sequence Diagram*

6 Evaluation

This chapter discuss the two studies conducted as part of the Reflective Learning Tool evaluation. The larger, user study involved a between subject study conducted with 14 total participants. Practices carried out in the learner study are based on experimental guidance from Ko and Fincher (2019). After being split into a control and experimental group, participants completed three 30-minute Jupyter Notebooks over one month. Quizzes were completed before and after to quantify the difference in learning between the two groups. Further post-course semi-structured interviews were carried out on four participants which used the Reflective Learning Tool. The second study looked to understand how lecturers may use the tool and if the tool would be of benefit. I gathered qualitative feedback from one University of Glasgow lecturer.

6.1 Learner User Study

6.1.1 Overview

The main focus of the evaluation was to understand if the tool would be useful for novice programmers. In-order to conduct a suitable evaluation, the learner user study required a between study experiment. I created three 30 minute Jupyter Notebook labs to be completed over three weeks. Participants were split into two groups; a group completing the Python course with the Reflective Learning Tool extensions installed and a control group using the standard Jupyter Notebook interface. Overall, 14 participants took part in the study, with two equal groups of seven. Participants completed a pre-course quiz and post-course quiz. Both quizzes where distributed via Microsoft Forms, they included a mix of multiple choice and text based responses. Each quiz was graded out of 20 and had a fixed completion time of 20 minutes.

Participants attended installation workshops at the start of the experiment and were given further information about the study. There were a series of tutorial sessions throughout the study to keep track of how the participants were finding the content as they completed the Python course. Although, all of the participants completed the majority of the course in their own time.

Both quizzes were designed to have a similar difficulty. There was a mix of multiple choice and text based questions in both quizzes. The design of the second, post-course quiz included a number of challenging questions which required students to apply their learning onto new problems. These questions being Q14 and Q15. The course did not specifically teach students how to complete these questions but the Notebooks covered concepts which would allow students to apply their learning to the questions. This allowed students to show if they had gone beyond the scope of the course and learned deeper than surface level. I wanted to understand which (if any) participants could apply their learning to new scenarios.

6.1.2 Pilot Study

After designing the user study, I ran a pilot study to better understand how long users would take to complete the Notebooks and quizzes. The pilot included one participant who completed the

Figure 6.1: *Picture taken from the pizza event.*

first quiz and then Notebook I designed for the course. The participant took over 45 minutes to complete the Notebook, and received seven out of 20 in the first quiz. The quiz score was a good indication that the questions where hard enough to allow participants a chance to improve over the course. I adapted the first Notebook based on the pilot study, with the aim to have students spend 30 minutes completing each Notebook. I adapted the further two Notebooks to last a similar length of time for participants.

6.1.3 Recruitment

As the tool was intended for novice programmers and involved investing a significant amount of time, recruiting was particularly challenging. A number of posters were distributed across the Glasgow University campus in the initial search to market the study to beginner developers (Appendices B). The poster did initially draw a number of responses but no-one that responded to the poster took part in the experiment.

Recruitment was also challenging as colleagues and other members of the Computing Science school did not fit the *novice* criteria. The study relied on mainly students from other disciplines to complete the course. As a result, the participants were a spread of friends, family and friends-of-friends.

6.1.4 Limitations

The study required participants to give up time over a number of week. As most of the participants where Glasgow University students there was a challenge to accurately simulate how novice students may interact with a Python course using the reflective tool. Due to time constraints and other course work deadlines, participants did not complete the Notebooks weekly over three weeks as originally planned. I briefed the participants during the initial setup up sessions to aim for a Notebook a week over three week, with each Notebook designed to take 30 minutes. Participants gave positive responses at the time but I later discovered that most participants were not following this suggested timing. I extended the course by one week and arranged a pizza event, shown by Figure 6.1. The purpose of this was to allow the participants a final opportunity to engage with the Notebook content and complete the post-course quiz.

Other limitations included the lack of demographic diversity in the participant pool. I avoided including any computing peers in the study to prevent external factors such as previous experience, but some participants had completed basic computing courses previously. Although the students which had some experience still showed a similar level of improvement with the rest of the group when comparing the quiz difference trend. Furthermore, twelve out of the fourteen participants were university students, indicating general lack of academic diversity within the group.

6.1.5 Results

Table 6.1 and Table 6.2 outline the quiz scores for both groups of participants. Overall the first quiz included 14 participants, only 13 completed the final quiz as one participant from the control group did not finish the course (Turtle). Although all the participants in the experimental group completed the final quiz, one participant (Smiley) did not complete any of the course Notebooks so I have denoted their performance with a line through their row in the Table 6.2. I have used the 12 scores (control = six participants and experimental = six participants) to calculate the following results.

Paired t-testing was carried out as both groups had the same dependent variable, quiz score difference. I used the SciPy ttsest_ind Python function to calculate a p-value. The results do not show statistical difference between the two groups quiz differences: the p-value is 0.39.

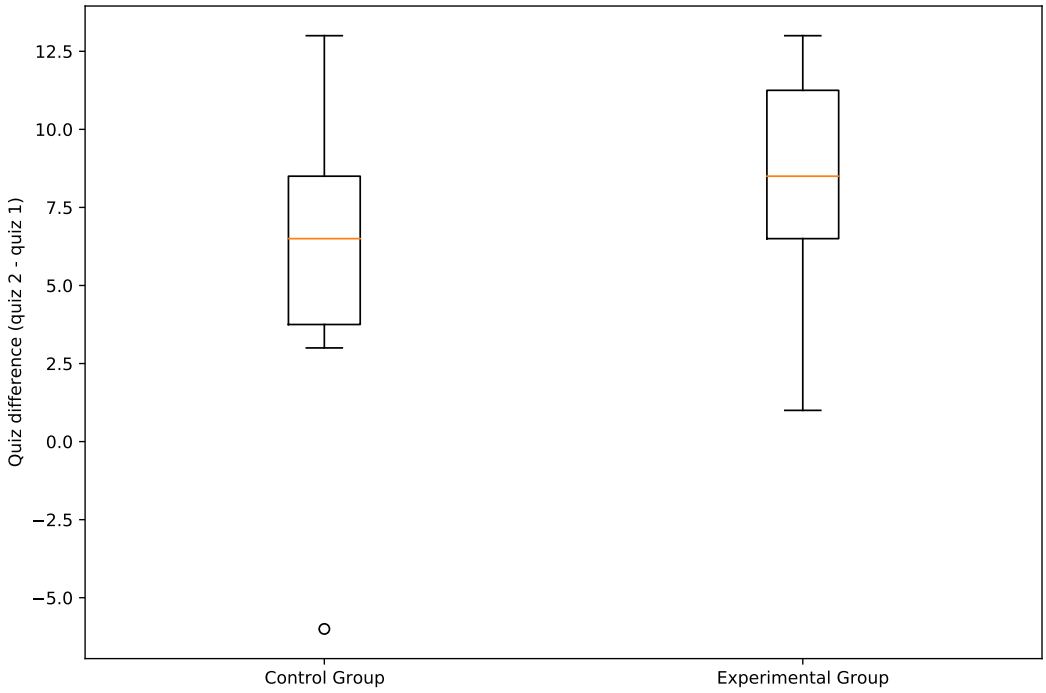


Figure 6.2: *Control group and experimental group quiz data difference.*

Although, the group using the Reflective Learning Tool performed better overall. The mean increase between the first and second quiz was 8.17 marks for the group with the Reflective Learning Tool compared with an average increase of 5.33 marks for the control group.

Looking at the more challenging questions in the second quiz, Q14 and Q15, the experimental group performed better than the control group. Out of the six participants in the experimental group, all six completed Q14 correctly and five completed Q15 correctly. The control group results show that just two participants completed Q14 and only one participant completed Q15 correctly. Results for the control and experimental group are displayed in Tables 6.3 and 6.4. As outlined previously, the inclusion of these question was to allow participants to show a deeper level of understanding and course engagement. The results indicate that the group using the Reflective Learning Tool were able to handle new concepts and apply their learning to new scenarios better than the control group.

Comparing the total points each participant received for only Q14 and Q15, we see there is statistical difference between the control and experimental groups (p-value=0.02). Although, we are only using two questions to complete the calculations, the results indicate that the Reflective Learning Tool led to students performing better when having to apply their skills to new scenarios. Figure 6.3 shows the difference between each group's total score for Q14 and Q15. As the study was conducted with a small number of participants, I would recommend a further study with more participants be conducted to avoid accusations of data dredging.

6.1.6 Post Course Interviews

I held a pizza party at the end of the course to thank all the participants that took part, and to allow participants a final chance to use the tool if that had not already done so. The event also allowed me to interview a number of participants that used the tool whilst completing the

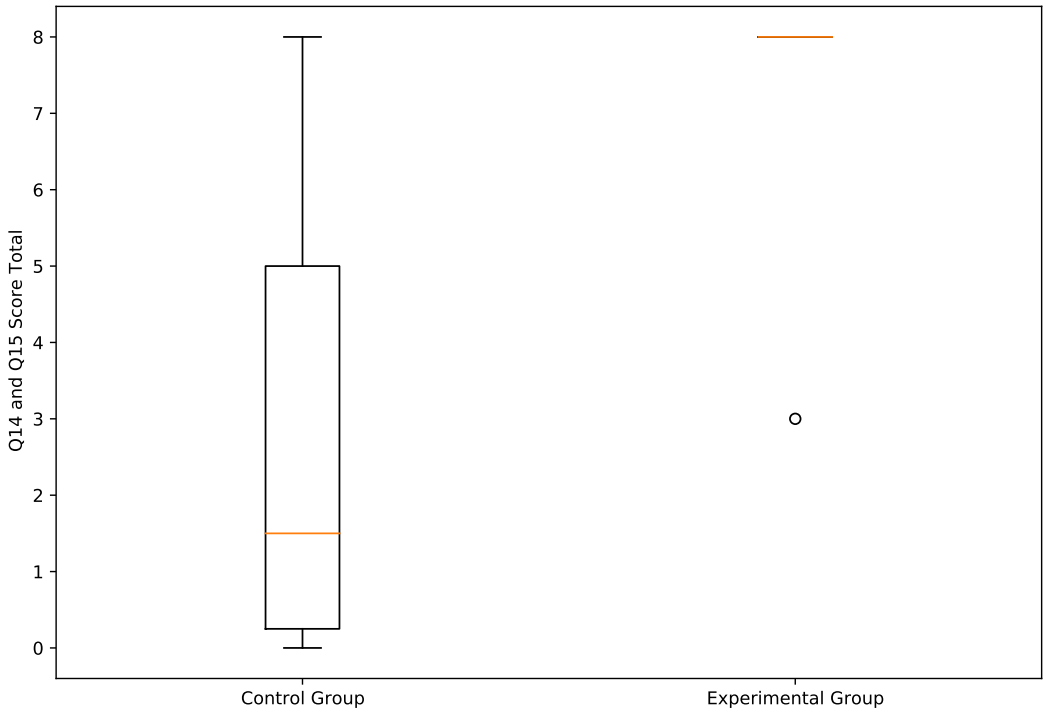


Figure 6.3: *2nd quiz Q14 and Q15 total for control and experimental groups.*

Participant Nickname	1st Quiz Score	2nd Quiz Score	Quiz Score Difference (+/-)
Grumpy Cat	5	18	+13
Jellyfish	0	9	+9
Silent Green	2	9	+7
Pepperoni Man	6	12	+6
Byres Road	0	3	+3
Bug	9	3	-6
~~Turtle~~	~~0~~	~~?~~	~~?~~

Table 6.1: *Control Group Quiz Scores*

Participant Nickname	1st Quiz Score	2nd Quiz Score	Quiz Score Difference (+/-)
Honk	7	20	+13
Giraffe	5	17	+12
Big Mac	8	17	+9
Arosos	11	19	+8
Mr Boombastic	14	20	+6
Ozzboss	11	12	+1
~~Smiley~~	~~1~~	~~2~~	~~+1~~

Table 6.2: *Experimental Group Quiz Scores*

beginner level course. Interviews followed a loose semi-structured format with the main aim being to understand how users interacted with the tool and to identify drawbacks with the design.

Participant Nickname	2nd Quiz Q14 Score	2nd Quiz Q14 Score	Q14 and Q15 Total
Grumpy Cat	3	5	8
Pepperoni Man	3	3	6
Silent Green	1	0	1
Jellyfish	0	0	0
Byres Road	0	0	0
Bug	0	0	0
~~Turtle~~	~~?~~	~~?~~	~~?~~

Table 6.3: *Control Group 2nd Quiz Q14 and Q15 Scores*

Participant Nickname	2nd Quiz Q14 Score	2nd Quiz Q14 Score	Q14 and Q15 Total
Honk	3	5	8
Giraffe	3	5	8
Big Mac	3	5	8
Arosos	3	5	8
Mr Boombastic	3	5	8
Ozzboss	3	0	3
~~Smiley~~	~~0~~	~~0~~	~~+1~~

Table 6.4: *Experimental Group 2nd Quiz Q14 and Q15 Scores*

Four participants which used the Reflective Learning Tool where interviewed.

Overall there was a feeling that the participants would use a similar tool in the future. The participants (n = 3) used both the reflective note and emotion selector together throughout. After further clarification, users would generally select an emotion relating to how an exercise would make them feel, and then write an accompanying reflective note about the experience. The participants (n = 4) further noted the perceived usefulness of being able to return to their emotions and reflections when revising content. They indicated that they would identify which sections they found frustrating or made them angry when revising content.

All of the participants that used the reflective tool in the study were Glasgow University students from non-computing related fields. Although, some participants (n = 3) identified a computing related cross over with their subject. This was mostly the use of Python for graphing and statistical analysis. Furthermore, the participants noted that the opportunity to take notes may help improve their understanding. For many students (n = 3) in other disciplines, they are given statistical problems to solve but no formal practical coding teaching. The tool allows students to make there own notes as they go through a lab and may lead to a better overall quality of learning.

Participants also noted a number of usability drawbacks with the tool. Feedback suggested that the layout of the buttons could have been clearer with better labels. One suggestion was that the icons were unclear and words may have been more appropriate to describe each button's action. Although there is an on-hover help message describing the functionality, participants were still unsure about each extensions purpose. This may indicate that the icons are not suitable and the on-hover messages are not helpful.

One participant, coming from a Physics background noted that they could not write equations as part of the reflective note. They specifically referenced Lab 1, where participants looked at operators and based Python equations. Although the course did not cover complex maths

problems, the participant informed me that they wrote out equations on paper and used them as a form of reflective note to keep track of their learning. They did not use the embedded reflective note tool to keep notes during the Notebook, but kept notes in a form which suited them better. A maths formula language such as Latex may be suitable to allow students to record calculations.

6.2 Lecturer Evaluation

After real learner reflections were gathered in the above user study, I evaluated the lecturer tool. With the guidance of my supervisor, I distributed screenshots (Figure 5.2) showing the summarised report and teacher specific functionality of the tool. Lecturers were also given the raw data to compare against. Due to time constraints and other mitigating factors, only one lecturer responded. The lecturer teaches honors level Computing Science courses which contain large numbers of students.

6.2.1 Qualitative Results

Overall, the feedback suggested that reflective data from students would be useful. The lecturer noted that the LLM "summarised the student comments very well". Furthermore the lecture indicated they would use this type of summary to enhance their lectures. They went on to comments that "this sort of summary is a really good use of LLMs". The lecturer also noted that "integrating the ability for student to reflection along with the exercises also reduces survey fatigue".

The comments made by the lecturer indicate that the tool would be useful and lecturers would benefit from seeing a brief summary of student reflections. Furthermore, the summary was accurate but due to the relatively low sample size this would need to be investigated further with a larger amount of reflective data.

The summary indicated that students were unsure about new concepts like mutability, the lecturer noted that they would "probably at the very least add in a recap of mutability into the start of the next lecture". Showing that the tool could be used to help prompt self-reflection for the teacher. After reading the short summary the lecturer was able to identify a common issue which they said they would address with the students.

6.3 Summary

The results from the student user study indicate that the learner tool did not lead to an overall statistically different improvement. However, focusing on Q14 and Q15 from the 2nd quiz, participants with the tool did preform significantly better when faced with unseen concepts. This would indicate that the tool may invoke a deeper level of learning as participant using the tool were able to apply their learning onto new scenarios whilst the control group could not. As mentioned, the sample size would need to be increased to better understand the effects of using the Reflective Learning Tool. The qualitative data showed that the learner tool was helpful and would be used by participants in the future. Particularly, students noted that bookmarking challenging topics with negative emotions would indicate to them which parts of a Notebook to revise. Although, there were a number of usability concerns around the purpose of each tool bar button. One participant suggested the incorporation of words and more informative icons to better indicate the purpose of each extension.

Feedback about the teacher tool was gathered from one lecturer. They suggested that the teacher tool provided an accurate and useful summary of student reflective feedback. The lecturer

indicated that they would make use of the tool by identifying common issue about a lab specific issue and then address the concern in class. Overall, the teacher tool allows lecturers to get an accurate snapshot of how their students are feeling about a lab. Lecturers can then use this as a form of student feedback to reflect on their own teaching.

7 Conclusion

7.1 Summary

The project successfully developed a series of Jupyter Notebook extensions which make up the Reflective Learning Tool for beginner programmers and teachers. The tool allows students to reflect whilst they are experiencing and after the event. Furthermore, students are guided on a reflective cycle at the end of Notebooks which follows practices laid out by Gibbs (1988). A body of previous and relevant work has been laid out in Chapter 2, showing the wide field of self-refection as a part of education. The background chapter showed that self-reflection can be a highly positive practice, especially for novices. The tool I have developed has attempted to address the lack of self-reflective support for beginner programmers.

The final tool incorporated a teacher specific extension which generates a summarised report for lecturers to accurately and quickly get a feel for how students are getting on with a Notebook. This was a direct result of the lecturer survey responses collected during the early analysis stage. After addressing the key stakeholders as teachers and learner, a set of functional and non-functional requirements were laid out in Chapter 3. These requirements addressed the aim from Chapter 1 and employ ideas researched in Chapter 2.

Chapters 4 and 5 detail the process I went through to implement the final tool. Wireframes were used to assist implementation of the Jupyter Notebook toolbar extensions. The implementation process involved a number of challenges, the overarching challenge was the lack of official Jupyter Notebook extension support. This meant that the tool had to be developed under unofficial NBExtension guidance. Furthermore, adding functionality to the extensions was a challenge as Jupyter Notebooks do not allow for local user inputs to be accessed via Javascript or Python files. This led to Magic commands being used to allow database and API interactions invoked by the local Jupyter server through embedded Notebook cells.

The evaluation discussed in Chapter 6, included a between subject user study conducted over five weeks, and a qualitative study looking to get feedback from University of Glasgow lecturers about the teacher tool. A number of results and takeaways can be drawn from the user study; mainly that reflection was helpful for novice programmers but we cannot be statistically conclusive about it. However, the results indicated that students using the Reflective Learning Tool preformed better when given new problems. The experimental group were able to apply their learning to new problems whilst the control group could not. Overall, the study sample size ($n = 14$) was too small to definitively state the effects of using the Reflective Learning Tool. Although, through user semi-structured interview I have found a number of themes and trends which would indicate that using the tool would benefit novice computing scientists. The ability to highlight which section or question evoked a negative emotion was noted by participants as a starting point if they were to revisit the Notebook.

I gathered feedback from a Glasgow University lecturer to help evaluate the teacher tool. After supplying them with a screenshot showing the LLM summarised report and raw reflection data,

the lecture supplied feedback. Overall, the feedback indicated that the tool would benefit teachers and allow them to get an accurate overview of how their students are feeling.

7.2 Future Work

7.2.1 Further Study

As noted during the evaluation in Chapter 6, there were several limitations and external factors which may have prevented the user study from producing statistically significant results. I would suggest that a future user study involving a larger number of participants over a longer period timer would be beneficial. Ideally, this would involve a beginner Level 1 university computing class who receive weekly Jupyter Notebook lab work. There would be an increased incentive for students to complete course content as well as an increased level of demographic variance in the larger sample pool. Furthermore, this would allow different groups to use specific extensions to better understand the impact of using each specific reflective extension. Sets of participants could use a limited selection or just one reflective extension, for example only using the emotion selector, and then experimenters could compare the impact of using one extension over the other.

The larger sample size would allow the lecturer tool to be evaluated with more data as well. Due to the limited number of participants in the initial study (total = 14), I was only able to garner feedback from lecturer based on six (number of participants from experimental group who completed the Notebooks) user inputs. Furthermore, I would recommend a study which includes several lecturers and teachers.

7.2.2 Different IDEs

The tool was developed for Jupyter Notebook as it was commonly used by novice programmers. Future work could look at developing a reflective tool which could be used across different IDE platforms (VS Code, Intellij, Eclipse). Users may want to code with a language not supported by Jupyter Notebook but still want to reflect using the tool.

7.2.3 Teacher Tool

The LLM generated report did not include emotion changes and timer data, it only summarised overall reflective feedback. Future iterations may want to investigate how including emotion changes and timer data may effect the output. Moreover, if there are further reflective parameters gathered from student, future work could look at the impact on adding them to the LLM prompt to be summarised. Different LLMs could be used to summarise data as well.

7.2.4 Further Self-Reflection Extensions

A further extension of this project could incorporate a system which builds from previous reflections. This could be in the form of chat-bot study tool where the tool would use previous reflection from users to generate a reflective action plan. Using previous reflections from labs to help build a more tailored experience for users could address the different ways users practice self-reflection.

As noted in Chapter 6, one user was not able to record calculations as part of their reflective note. A further extension could incorporate a Latex maths input to allow users to record calculations. This could be its own extension or implemented as part of the current reflective note tool.

7.3 Reflection

Overall, the project allowed me the opportunity to showcase the skills I have learned over the past four years at the University of Glasgow. Particularly, I enjoyed coordinating the user study for the Reflective Learning Tool. Running the evaluation allowed me to get a taste for how researchers and lecturers complete studies. As a result, I would like to come back to academia in the future. The project has also shown me the importance of self-reflection as a learning practice. Whilst completing the project, I have been applying reflective practices to my own learning. As a result, I have seen the benefits of apply reflective practices to my own studies this academic year.

A Completed Ethics Checklist Form

School of Computing Science
University of Glasgow

Ethics checklist form for assessed exercises (at all levels)

This form is only applicable for assessed exercises that use other people ('participants') for the collection of information, typically in getting comments about a system or a system design, or getting information about how a system could be used, or evaluating a working system.

If no other people have been involved in the collection of information, then you do not need to complete this form.

If your evaluation does not comply with any one or more of the points below, please contact the Chair of the School of Computing Science Ethics Committee (matthew.chalmers@glasgow.ac.uk) for advice.

If your evaluation does comply with all the points below, please sign this form and submit it with your assessed work.

---

1. Participants were not exposed to any risks greater than those encountered in their normal working life.

   Investigators have a responsibility to protect participants from physical and mental harm during the investigation. The risk of harm must be no greater than in ordinary life. Areas of potential risk that require ethical approval include, but are not limited to, investigations that occur outside usual laboratory areas, or that require participant mobility (e.g. walking, running, use of public transport), unusual or repetitive activity or movement, that use sensory deprivation (e.g. ear plugs or blindfolds), bright or flashing lights, loud or disorienting noises, smell, taste, vibration, or force feedback

2. The experimental materials were paper-based, or comprised software running on standard hardware.

   Participants should not be exposed to any risks associated with the use of non-standard equipment: anything other than pen-and-paper, standard PCs, laptops, iPads, mobile phones and common hand-held devices is considered non-standard.

3. All participants explicitly stated that they agreed to take part, and that their data could be used in the project.

   If the results of the evaluation are likely to be used beyond the term of the project (for example, the software is to be deployed, or the data is to be published), then signed consent is necessary. A separate consent form should be signed by each participant.

   Otherwise, verbal consent is sufficient, and should be explicitly requested in the introductory script.

4. No incentives were offered to the participants.

   The payment of participants must not be used to induce them to risk harm beyond that which they risk without payment in their normal lifestyle.

5. No information about the evaluation or materials was intentionally withheld from the participants.
   Withholding information or misleading participants is unacceptable if participants are likely to object or show unease when debriefed.

6. No participant was under the age of 16.
   Parental consent is required for participants under the age of 16.

7. No participant has an impairment that may limit their understanding or communication.
   Additional consent is required for participants with impairments.

8. Neither I nor my supervisor is in a position of authority or influence over any of the participants.
   A position of authority or influence over any participant must not be allowed to pressurise participants to take part in, or remain in, any experiment.

9. All participants were informed that they could withdraw at any time.
   All participants have the right to withdraw at any time during the investigation. They should be told this in the introductory script.

10. All participants have been informed of my contact details.
    All participants must be able to contact the investigator after the investigation. They should be given the details of both student and module co-ordinator or supervisor as part of the debriefing.

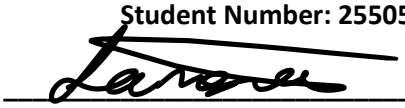
11. The evaluation was discussed with all the participants at the end of the session, and all participants had the opportunity to ask questions.
    The student must provide the participants with sufficient information in the debriefing to enable them to understand the nature of the investigation. In cases where remote participants may withdraw from the experiment early and it is not possible to debrief them, the fact that doing so will result in their not being debriefed should be mentioned in the introductory text.

12. All the data collected from the participants is stored in an anonymous form.
    All participant data (hard-copy and soft-copy) should be stored securely, and in anonymous form.

Course and Assessment Name: Level 4 Project COMSCI4025P (Single Hons)

Student's Name: James Orr

Student Number: 2550508

Student's Signature 

Date __18/3/24____

B Recruitment Poster

2550508o@student.gla.ac.uk

Learn Beginner Python for Free

Online Courses to Enhance Your Python Skills

I have created a Jupyter Notebook reflective learning tool and need your help to evaluate it! Participants will benefit from learning basic Python programming skills which they can add to their CV. The study is aimed at novice programmers. If you are interested and want to learn more please email me by 10th of February 2024, 2550508o@student.gla.ac.uk

More Information

2550508o@student.gla.ac.uk

James Orr

C Quizzes and Results

20 minutes

Evaluation Quiz 1

Please take this quiz under formal closed book exam condition. You are allowed to use a calculator. Do not refer to online or offline help. I also ask you do not discuss your answers to the quiz with any other participants until after the experiment has concluded.

Remember the quiz is intended to be challenging and is going to be used as a baseline metric. Take your time and if you are unsure about a question then skip it.

If for any reason you are interrupted during the quiz, email me (2550508o@student.gla.ac.uk) to make further arrangements.

* Required

1

Please enter a unique nickname. This will help identify the difference in both of your quizzes whilst keeping your identity anonymous. For example, favorite fictional character or pick a pronoun and a colour e.g. Mr Pink

Please use the same nickname for both quizzes. *

2

How can you determine the type of a variable? (1 Point)

- ◯ Print out the value and determine the data type based on the value printed.
- ◯ Use the type function.
- ◯ Use it in a known equation and print the result.
- ◯ Look at the declaration of the variable.

3

What value is printed when the following statement executes?

print(int(53.8734)) (1 Point)

- Nothing is printed. It generates a runtime error.
- 53
- 54
- 53.785

4

What is printed when the following statements execute?

```
day = "Thursday"
day = 32.5
day = 19
print(day)
```
(1 Point)

- Nothing is printed. A runtime error occurs.
- Thursday
- 32.5
- 19

5

What is printed when the following statements execute?

```
n = input("Please enter your age: ")
 # user types in 18
print ( type(n) )
```
(1 Point)

- <class 'str'>
- <class 'int'>
- <class 18>
- 18

6

What is the value of the following expression:

16 - 2 * 5 // 3 + 1 (1 Point)

- 14
- 24
- 3
- 13.667

7

After the following statements, what are the values of x and y?

```
x = 15
y = x
x = 22
```
(1 Point)

- x is 15 and y is 15
- x is 22 and y is 22
- x is 15 and y is 22
- x is 22 and y is 15

8

What is printed?

```
country = "Scotland"
print(country[1:4])
```
(1 Point)

- "Scot"
- "Sco"
- "cote"
- "cot"
- Error

9

What is printed?

```
numbers = [2,4,6,8]
print(numbers[2])
```
(1 Point)

- 2
- 4
- 6
- 8
- Error

10

What is printed?

```
age = 18

if age < 3:
    print("baby")
elif age < 13:
    print("child")
elif age < 20:
    print("teen")
elif age < 60:
    print("adult")
else:
    print("OAP")
```
(1 Point)

- child
- teen adult OAP
- Error
- teen adult
- teen

11

What is printed?

```
myList = [2,7,4,3]
result = 0

for x in myList:
   result += x

print(result)
```
(1 Point)

- ◯ 16
- ◯ 3
- ◯ Error
- ◯ 0
- ◯ 4

12

The following code contains a number of errors. There should be a list *x* of integers, and an integer *y*. Each integer in the list *x* should be compared against *y*, if it is the same value as *y*, output *Same*. If it is bigger than *y*, output *Bigger*. Otherwise, output *Smaller*.

Identify each error, stating its line number, then explain how you would correct it. * (10 Points)

```
1  x = (3,5,4,5,6,2)
2  y is 4
3  for x in i:
4  if i = y:
5      print(Same)
6  elif y > i:
7      print('Bigger")
8  else
9      "Smaller"
```

Evaluation Quiz 1

14	5.6	10:55	Active
Responses	Average Score	Average time to complete	Status

1. Please enter a unique nickname. This will help identify the difference in both of your quizzes whilst keeping your identity anonymous. For example, favorite fictional character or pick a pronoun and a colour e.g. Mr Pink (0 point)

   Please use the same nickname for both quizzes.

14
Responses

Latest Responses
"Mr Turtle"
"jellyfish1"
"Silent Green"

2 respondents (**14**%) answered **Mr** for this question.

2. How can you determine the type of a variable? (1 point)
   46% of respondents (6 of 13) answered this question correctly.

Print out the value and determi...	1	
Use the type function.	6	✓
Use it in a known equation and ...	0	
Look at the declaration of the v...	6	

3. What value is printed when the following statement executes? (1 point)

   print(int(53.8734))
   17% of respondents (2 of 12) answered this question correctly.

Nothing is printed. It generates ...	3	
53	2	✓
54	6	
53.785	1	

4. What is printed when the following statements execute? (1 point)

```
day = "Thursday"
day = 32.5
day = 19
print(day)
```

50% of respondents (6 of 12) answered this question correctly.

Nothing is printed. A runtime er...	4	
Thursday	2	
32.5	0	
19	6	✓

5. What is printed when the following statements execute? (1 point)

```
n = input("Please enter your age: ")
# user types in 18
print ( type(n) )
```

25% of respondents (3 of 12) answered this question correctly.

<class 'str'>	3	✓
<class 'int'>	2	
<class 18>	1	
18	6	

6. What is the value of the following expression: (1 point)

```
16 - 2 * 5 // 3 + 1
```

25% of respondents (3 of 12) answered this question correctly.

14	3	✓
24	1	
3	3	
13.667	5	

7. After the following statements, what are the values of x and y? (1 point)

```
x = 15
y = x
x = 22
```

57% of respondents (8 of 14) answered this question correctly.

x is 15 and y is 15	0	
x is 22 and y is 22	5	
x is 15 and y is 22	1	
x is 22 and y is 15	8	✓

8. What is printed? (1 point)

```
country = "Scotland"
print(country[1:4])
```

8% of respondents (1 of 12) answered this question correctly.

"Scot"	6	
"Sco"	1	
"cote"	0	
"cot"	1	✓
Error	4	

9. What is printed? (1 point)

```
numbers = [2,4,6,8]
print(numbers[2])
```

0% of respondents (0 of 13) answered this question correctly.

2	4	
4	8	
6	0	✓
8	0	
Error	1	

10. What is printed? (1 point)

```
age = 18

if age < 3:
    print("baby")
elif age < 13:
    print("child")
elif age < 20:
    print("teen")
elif age < 60:
    print("adult")
else:
    print("OAP")
```

50% of respondents (7 of 14) answered this question correctly.

child	0	
teen adult OAP	1	
Error	1	
teen adult	5	
teen	7	✓

11. What is printed? (1 point)

```
myList = [2,7,4,3]
result = 0

for x in myList:
    result += x

print(result)
```

17% of respondents (2 of 12) answered this question correctly.

16	2 ✓
3	1
Error	3
0	5
4	1

12. The following code contains a number of errors. There should be a list *x* of integers, and an integer *y*. Each integer in the list *x* should be compared against *y*, if it is the same value as *y*, output *Same*. If it is bigger than *y*, output *Bigger*. Otherwise, output *Smaller*. (10 points)

Identify each error, stating its line number, then explain how you would correct it.

14
Responses

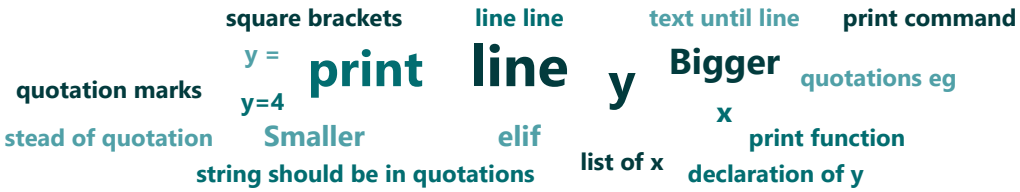
Latest Responses

"Unsure of errors or how to correct them"

"i is not an integer. i is and imaginary number. i = square root of -1. line 1 does ...

"I have no idea"

8 respondents (**57**%) answered **line** for this question.



Evaluation Quiz 2 - Group X - Reflective Tool

Please take this quiz under formal closed book exam condition. You are allowed to use a calculator. Do not refer to online or offline help. I also ask you do not discuss your answers to the quiz with any other participants until after the experiment has concluded.

Remember the quiz is intended to be challenging. Take your time and if you are unsure about a question then skip it.

If for any reason you are interrupted during the quiz, email me (2550508o@student.gla.ac.uk) to make further arrangements.

* Required

1

Please enter the same unique nickname you entered for quiz 1. This will help identify the difference in both of your quizzes whilst keeping your identity anonymous. For example, favorite fictional character or pick a pronoun and a colour e.g. Mr Pink

Please use the same nickname for both quizzes. *

2

How many Notebooks did you complete *

- ◯ 1
- ◯ 2
- ◯ All 3
- ◯ None, what lab books?

3

What is the data type of 'this is what kind of data'? * (1 Point)

- ◯ Character
- ◯ Integer
- ◯ Float
- ◯ String

4

What is the value of the following expression:

2 ** 2 ** 3 * 3 * (1 Point)

- 768
- 128
- 12
- 256

5

What is printed?

```
place = 'Glasgow'
print(place[::2]) *
```

(1 Point)

- Error
- Gagw
- l
- a
- ow
- lso

6

What is printed?

```
a = "hello"
b = 4
c = [2,4,6]
d = True
myList = ['a', c, b, 4, False]

print(myList[1]) *
```

(1 Point)

- "hello"
- 4
- False
- True
- a
- [2,4,6]

7

What will happen when this code is executed?
* (1 Point)

```
number = 5
while number <= 5:
    if number < 5:
        number = number + 1
    print(number)
```

- Nothing because the while loop is never executed
- Error
- The value of number will be printed exactly 5 times
- The program will loop indefinitely
- The value of number will be printed exactly 1 time

8

What is the value of the following expression:

19 + 4 * 2 // 5 + 3 * (1 Point)

- 23.6
- 23
- 20
- 12
- 12.2

9

What will be printed after the following code is executed?

* (1 Point)

```
s = 0
x = [0, 2, 4]
y = [1, 3, 5]
for i in range(len(x)):
    for q in range(len(y)):
        s += y[q]
print(s)
```

- 27
- Nothing, the program will get stuck in an infinite loop
- 9
- 15
- 0
- [1, 3, 5, 0, 2, 4]

10

What is a correct syntax to output "Hello World" in Python? * (1 Point)

- print("Hello World)
- print('Hello World')
- p("Hello World")
- consolo.log('Hello World');

11

How do you insert comments in Python code? * (1 Point)

- //This is a comment
- /#This is a comment#/
- /*This is a comment*/
- #This is a comment

12

What will be printed after the following code is executed?
* (1 Point)

```
number = 3
print(number%2 == 1)
```

- ◯ False
- ◯ 1
- ◯ True
- ◯ Error
- ◯ 3
- ◯ 0

13

Describe the difference between a **for** loop and a **while** loop. * (2 Points)

14

Can you adapt the following code to use indefinite iteration? If yes, rewrite the Python code to use indefinite iteration.

* (3 Points)

```
for i in range(10):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

15

Write Python code which ask a user to input their age as a whole number, then print "*You are really old*" if the input age is over 30 or print "*You are still in your prime!*" if the age is 30 and under. *
(5 Points)

Evaluation Quiz 2 - Group Y

6	9.0	Active
Responses	Average Score	Status

1. Please enter the same unique nickname you entered for quiz 1. This will help identify the difference in both of your quizzes whilst keeping your identity anonymous. For example, favorite fictional character or pick a pronoun and a colour e.g. Mr Pink (0 point)

   Please use the same nickname for both quizzes.

   6
   Responses

   Latest Responses

   "Bug"

   "Cat (I dont remember I'm so sorry but i think it was something cat related)"

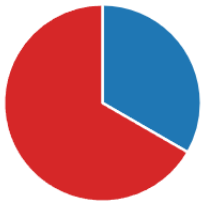
   "ByresRoad"

2. How many Notebooks did you complete (0 point)

1	0
2	1
All 3	5
None, what lab books?	0

3. What is the data type of 'this is what kind of data'? (1 point)
   67% of respondents (4 of 6) answered this question correctly.

Character	2	
Integer	0	
Float	0	
String	4	✓



4. What is the value of the following expression: (1 point)

   2 ** 2 ** 3 * 3
   50% of respondents (3 of 6) answered this question correctly.

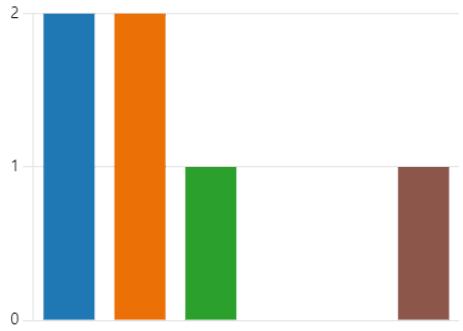
768	3	✓
128	1	
12	0	
256	2	

5. What is printed? (1 point)

```
place = 'Glasgow'
print(place[::2])
```

33% of respondents (2 of 6) answered this question correctly.

Error	2	
Gagw	2	✓
l	1	
a	0	
ow	0	
lso	1	



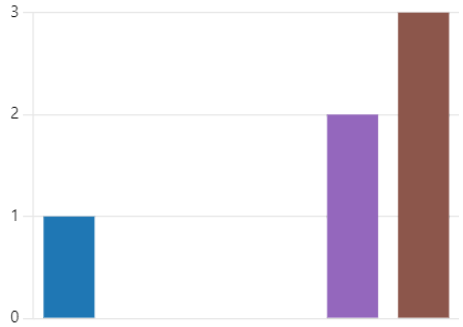
6. What is printed? (1 point)

```
a = "hello"
b = 4
c = [2,4,6]
d = True
myList = ['a', c, b, 4, False]

print(myList[1])
```

50% of respondents (3 of 6) answered this question correctly.

"hello"	1	
4	0	
False	0	
True	0	
a	2	
[2,4,6]	3	✓



7. What will happen when this code is executed? (1 point)

33% of respondents (2 of 6) answered this question correctly.

Nothing because the while loop...	1	
Error	2	
The value of number will be prin...	0	
The program will loop indefinitely	2	✓
The value of number will be prin...	1	

8. What is the value of the following expression: (1 point)

19 + 4 * 2 // 5 + 3

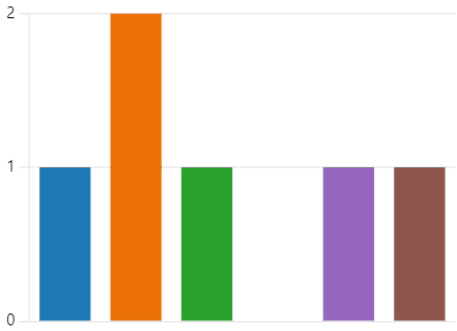
50% of respondents (3 of 6) answered this question correctly.

23.6	1	
23	3	✓
20	0	
12	1	
12.2	1	

9. What will be printed after the following code is executed? (1 point)

17% of respondents (1 of 6) answered this question correctly.

27	1	✓
Nothing, the program will get st...	2	
9	1	
15	0	
0	1	
[1, 3, 5, 0, 2, 4]	1	



10. What is a correct syntax to output "Hello World" in Python? (1 point)

83% of respondents (5 of 6) answered this question correctly.

print("Hello World)	0	
print('Hello World')	5	✓
p("Hello World")	1	
consolo.log('Hello World');	0	

11. How do you insert comments in Python code? (1 point)

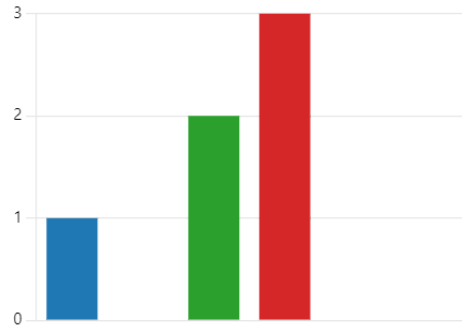
100% of respondents (6 of 6) answered this question correctly.

//This is a comment	0	
/#This is a comment#/	0	
/*This is a comment*/	0	
#This is a comment	6	✓

12. What will be printed after the following code is executed? (1 point)

33% of respondents (2 of 6) answered this question correctly.

False	1
1	0
True ✓	2
Error	3
3	0
0	0



13. Describe the difference between a **for** loop and a **while** loop. (2 points)

6
Responses

Latest Responses

"While loop stops when a condition is met."

"A while loop repeats a block of statements while a specific stated condition is t..."

"Don't know"

14. Can you adapt the following code to use indefinite iteration? If yes, rewrite the Python code to use indefinite iteration. (3 points)

5
Responses

Latest Responses

"no"

"n = -1 while n < 9: n += 1 print(n)"

"don't know"

15. Write Python code which ask a user to input their age as a whole number, then print "*You are really old*" if the input age is over 30 or print "*You are still in your prime!*" if the age is 30 and under. (5 points)

6
Responses

Latest Responses

"I can't remember doing this."

"age = input('Please enter your age:') age = int(age) if age > 30: print('You are r..."

"#age print "You are really old" >30 print "You are still in your prime" <=30"

Evaluation Quiz 2 - Group X - Reflective Tool

7	15.3	Active
Responses	Average Score	Status

1. Please enter the same unique nickname you entered for quiz 1. This will help identify the difference in both of your quizzes whilst keeping your identity anonymous. For example, favorite fictional character or pick a pronoun and a colour e.g. Mr Pink (0 point)

   Please use the same nickname for both quizzes.

   7
   Responses

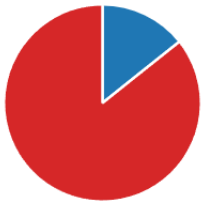
   Latest Responses
   "Emily"
   "Arosos"
   "Ossboss"

2. How many Notebooks did you complete (0 point)

1	0
2	0
All 3	5
None, what lab books?	2

3. What is the data type of 'this is what kind of data'? (1 point)
   86% of respondents (6 of 7) answered this question correctly.

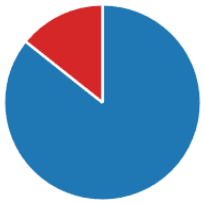
Character	1	
Integer	0	
Float	0	
String	6	✓



4. What is the value of the following expression: (1 point)

   2 ** 2 ** 3 * 3
   86% of respondents (6 of 7) answered this question correctly.

768	6	✓
128	0	
12	0	
256	1	

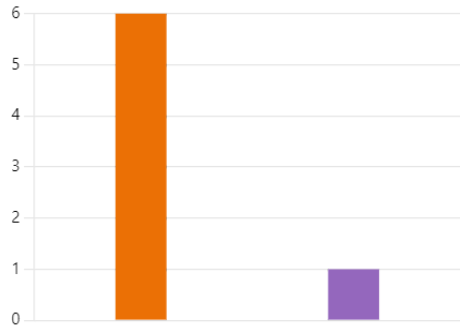


5. What is printed? (1 point)

```
place = 'Glasgow'
print(place[::2])
```

86% of respondents (6 of 7) answered this question correctly.

Error	0	
Gagw	6	✓
l	0	
a	0	
ow	1	
lso	0	



6. What is printed? (1 point)

```
a = "hello"
b = 4
c = [2,4,6]
d = True
myList = ['a', c, b, 4, False]

print(myList[1])
```

86% of respondents (6 of 7) answered this question correctly.

"hello"	0	
4	0	
False	1	
True	0	
a	0	
[2,4,6]	6	✓

7. What will happen when this code is executed? (1 point)

43% of respondents (3 of 7) answered this question correctly.

Nothing because the while loop...	1	
Error	0	
The value of number will be prin...	0	
The program will loop indefinitely	3	✓
The value of number will be prin...	3	

8. What is the value of the following expression: (1 point)

19 + 4 * 2 // 5 + 3

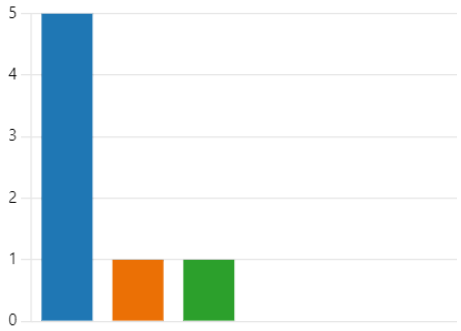
86% of respondents (6 of 7) answered this question correctly.

23.6	1	
23	6	✓
20	0	
12	0	
12.2	0	

9. What will be printed after the following code is executed? (1 point)

71% of respondents (5 of 7) answered this question correctly.

27	5	✓
Nothing, the program will get st...	1	
9	1	
15	0	
0	0	
[1, 3, 5, 0, 2, 4]	0	



10. What is a correct syntax to output "Hello World" in Python? (1 point)

86% of respondents (6 of 7) answered this question correctly.

print("Hello World)	1	
print('Hello World')	6	✓
p("Hello World")	0	
consolo.log('Hello World');	0	

11. How do you insert comments in Python code? (1 point)

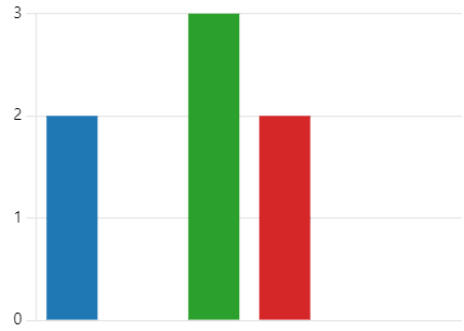
86% of respondents (6 of 7) answered this question correctly.

//This is a comment	0	
/#This is a comment#/	1	
/*This is a comment*/	0	
#This is a comment	6	✓

12. What will be printed after the following code is executed? (1 point)

43% of respondents (3 of 7) answered this question correctly.

False	2
1	0
True ✓	3
Error	2
3	0
0	0



13. Describe the difference between a **for** loop and a **while** loop. (2 points)

7
Responses

Latest Responses

"A for loop will be fixed. It will execute a body of code a fixed number of times. ...

"A for loop will run for a finite number of known cycles or the number cycles co...

"a while loop executes indefinitely until a predefined parameter is met"

14. Can you adapt the following code to use indefinite iteration? If yes, rewrite the Python code to use indefinite iteration. (3 points)

7
Responses

Latest Responses

"yes i = 0 while i< 10: print(i) i = i + 1 "

"i = 0 while i <= 9: print(i) i+=1 "

"i=0 while i <=10: print (i) i +=1 "

15. Write Python code which ask a user to input their age as a whole number, then print "*You are really old*" if the input age is over 30 or print "*You are still in your prime!*" if the age is 30 and under. (5 points)

7
Responses

Latest Responses

"userAge = int(input("Whats your age?")) if userAge <31: print("You are still in y...

"age = int(input('What is your age, as a whole number?')) if age > 30: print('Yo...

"I swear this was not taught on the course"

D Sample Notebook from User Study

Lab 2: Lists

There are some lab questions at the end to help reinforce your new Python skills. Your solutions are not graded so do not worry if you can not finish them all.

Remember

You have the reflective tool bar enabled. This means that you can record your time spent, write reflective notes, and update how you are feeling. Please start the timer when you open this Notebook and click the save button when you are finished. Use the reflective note button to add notes throughout. I would also recommend using the emotion selector at the start and end of the lab. Once finished with the Notebook, click the lightning bolt button on the tool bar to complete your evaluation and reflections. Always remember to input the correct lab number when asked.

Data structures

Organising data into **data structures** is a key part of programming. Most languages support a number of different types of data structures which **aggregate** (collect together) simple data like numbers, strings or other data structures. They are sometimes called **compound** data structures because they are made up of elements.

Lists

Lists are probably the most important compound data type in Python and are very widely used. They are a **sequence type** and represent an ordered sequence of values. Sequence types are a specialised kind of **collection type** (they impose **order**); a **collection type** just means a data type that holds multiple elements.

I'll use **element** to mean a value that can be in a list, and **sequence** to refer to any ordered collection of **elements**. For example the list

```
[1,2,3]
```

is a **sequence** of **elements** 1, 2 and 3.

Syntax

Lists have a **literal syntax** (this means you can directly create a list in a single step in Python).

Just put values between square brackets `[ ]` and separate with commas:

In [1]:
```
singleElement_list = [1]
intList = [5, 6, 7, 8]
stringList = ["first", "last"]

# note that we can put lists inside lists using this syntax
nestedList = [1, [2, [3]], 4]

# we might write the card hand a
# ace-of-hearts, ace-of-clubs, king-of-clubs, king-of-spades
# like this, as a list of four pairs
twoPair = [["a", "hearts"],
           ["a", "clubs"],
           ["k", "clubs"],
           ["k", "spades"]]

emptyList = []
```

In [2]:
```
intList = [1, 2, 3]
stringList = ["one", "two", "three"]

# note that it's fine for a list to hold more lists
# or any other collection type
listOfLists = [intList, stringList]

print(intList)
print(stringList)
print(listOfLists)
```

```
[1, 2, 3]
['one', 'two', 'three']
[[1, 2, 3], ['one', 'two', 'three']]
```

In [3]:
```
mixedList = [1, 2, "three", "four", 5, intList]
print(mixedList)
```

```
[1, 2, 'three', 'four', 5, [1, 2, 3]]
```

Length

The length of a list -- the number of values it contains -- can be returned using `len()`. `len()` actually works for **any** sequence type, e.g. for strings.

In [ ]:
```
a = []
print(a, len(a))
b = [1]
print(b, len(b))
c = [1,2,3]
print(c, len(c))
```

In [ ]:
```
## tricky: this list has one element -- which is itself a list
d = [[1,2,3]]
print(d, len(d))
```

Indexing

To access a single elements of a list, we use square brackets after the list. This is called **indexing**.

In [ ]:
```
elements = ["air", "earth", "fire", "water"]
print(elements[0])  # first element of a
print(elements[3])  # last element of a
```

0-based indexing

Python indices lists beginning at 0 (not 1!), so the first element of a list is indexed by [0] and the last element is (len(list)-1).

In [ ]:
```
print(elements[4]) # this is an error at runtime -- there are only 4 elemen
ts!
```

Negative-indices

If you use a negative index, Python treats it as counting backwards from the end of the list. For example `elements[-1]` means the last element in the list; `elements[-2]` means the second-to-last, and so on.

In [ ]:
```
print(elements[-1]) # you might think this would be an error... but it is
n't!
print(elements[-2])
```

Slicing

As well as **indexing** which extracts a specific element, Python lets you **slice** sequence types like lists. This makes it very easy to pick out subsections of a list. Slicing "chops out" a subsequence from a sequence. It works for lists, strings, arrays and many other sequence types.

Slicing uses the syntax:

```
my_list[start:end]
```

and will return all elements starting at `start` and ending at **but not including** `end`.

In [16]:
```
a = list(range(20))   # range creates a list of numbers 0..19

print("a", a)
```

```
a [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

In [ ]:
```
print("0:5 ", a[0:5])   # creates a new list with
                        # first five elements of a

# We can just omit the start
# it will default to 0
print(":5 ", a[:5])     # creates a new list
                        # with first five elements of a
```

In [17]:
```
print("5:10", a[5:10])  # 6th to 10th element
```

```
5:10 [5, 6, 7, 8, 9]
```

In [ ]:
```
print("10:", a[10:])  # After the tenth element
```

In [ ]:
```
print(":-5 ", a[:-5])   # omitting the first index means the same as zero;
                # negative numbers work exactly as in indexing
                # this takes everything except the last five elements
```

In [ ]:
```
print("-5: ", a[-5:])    # omitting the last index means until the end
                # this takes the last five elements only
```

Advanced slicing ¶

You can specify **three-element** slices in Python:

```
my_list[a:b:c]
```

which means take the elements from `a:b` (as before), **taking only every `c` th element**

In [ ]:
```
a = list(range(20))   # range creates a list of numbers 0..19

print("0:10", a[0:10])   # simple 2 element slice
print("0:10:1", a[0:10:1]) # exactly the same as a[0:10]
print("0:10:2", a[0:10:2]) # every second element of a[0:10]
print("0:10:20", a[0:10:20]) # every twentieth element of a[0:10];
print("::3", a[::3]) # every third element of a
# note that the count starts at zero, so we will always get the first eleme
nt
```

Slicing backwards

If the step is negative, we will step **backwards** through the list. This is exactly how the `range()` function worked in defining `for` loops.

In [ ]:
```
a = list(range(20))

print(a[10:0:-1]) # every element of a[0:10] in reverse order
print(a[10:0:-2]) # every second element of a[0:10] in reverse order.

## shorthand to reverse a list
print(a[::-1])   # the whole list, in reverse order
```

Item and slice assignment

We can assign directly to list elements. This has an indexed list on the LHS and any value on the RHS

```
list[index] = value
```

In [ ]:
```
l = [1,2,3]
l[0] = "Wonderful"
print(l)
```

This also works with slices, as long as the LHS and RHS have a matching number of elements.

```
mylist[a:b] = otherlist
```

In [ ]:
```
# you can assign to slices -- as long as they have the same size on both
# the left and right hand sides
l[0:2] = ["seven", "nine"]
print(l)
```

In [ ]:
```

```

In [19]:
```
l1 = [1,2,3,4,5]
l2 = ["A", "B", "C", "D", "E"]
l1[2:4] = l2[3:5]
print(l1)
```

```
[1, 2, 'D', 'E', 5]
```

In [20]:
```
x = list(range(20))
print(x)
y = [0] * 10
print(y)
# assign 0 to every even element
x[::2] = y
print(x)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 1, 0, 3, 0, 5, 0, 7, 0, 9, 0, 11, 0, 13, 0, 15, 0, 17, 0, 19]
```

Joining

The `+` operator is **overloaded** for use with lists (or in fact any sequence). This means that using `+` on lists will join them together. Note that `+` does **not** add each elements of a lists to another; it joins two lists into a new one.

Joining of sequences is called **concatenation** in computer science; the `+` operator is sometimes called the "concatenation operator" when used on lists.

In [ ]:
```
chinese_elements = ['earth', 'fire', 'water'] + ["wood", "metal"]
print(chinese_elements)
```

In [ ]:
```
a = [1, 2, 3]

print(a + a + a)  # we can concatenate many lists
```

Lists in Python

Dynamically sized

Python lists are dynamically sized: this means you can add and remove elements as you wish, without having to declare the size of the list in advance.

In [ ]:
```
a = [1, 2, 3]
a.append(4)  # we can just append values as we want
print(a)
```

In [ ]:
```

```

Adding using append()

As well as creating new lists, lists can be modified **in place**. **This is a major difference from types we have seen so far**. We can add new elements to a list with `.append()`

In [21]:
```
l = []
l.append(1)
l.append(2)
l.append(3)
print(l)
```

```
[1, 2, 3]
```

This doesn't make a new list. It changes the elements of the existing list. This is a very important difference.

Append is not +

In [ ]:
```
a = [1,2,3]
b = a+a
print("b", b)

c = []
c.append(a)
c.append(a)
print("c", c)
```

Mutability

Note that we have actually changed the lists using `.append()`. The property of a data structure being changeable is called **mutability** (the ability to **mutate** or change). This mutability of `list`s has important consequences, which we'll discuss below.

Copying and references

Let's return to the idea of **mutability**. In Python, simple types like numbers and strings are **immutable** (they cannot be changed after they have been created).

In [ ]:
```
## integers
a = 32
b = a
b = b + 1
print("a=",a)
print("b=", b)
print()

## strings
c = "hello"
d = c
d += " world"
print("c=",c)
print("d=", d)
```

But lists are **mutable**; they can be changed after they have been created. The list itself lives off in some bit of memory -- variables just **point at the** list. They are **references** to the list that has been created.

If multiple variables point at one single list then making a change to the list will appear for every variable that refers to it!

In [ ]:
```
a = [1,2,3]
b = a
b.append("!")

# note: both a and b will be [1,2,3,"!"]
# because they both refer to the *same* list
print("a=",a)
print("b=",b)
```

Copying

If you want to work a list without affecting existing variables which refer to it, you need to make a **copy** of the list.

This is true for all data structures in Python (it's true for numbers and strings too, but there is no way to modify them, so no need to ever copy them).

Handy note: slicing a list returns a new list with the same elements.

Thus, the syntax [:] (a slice taking the whole list) can be used to copy a list:

In [ ]:
```
a = [1,2,3]
b = a[:]          # create a new list with the same entries as are in a
b.append("!")

# Now, a and b refer to *different* lists
print("a=",a)
print("b=",b)
```

You can test two lists for **equality**, which tests if they have the same elements:

In [ ]:
```
a = [1,2,3]
b = [1,2,3]
print(a==b)
```

But if you want to test if a list is a **copy** of another, you can use the `is` operator. This tests if two variables refer to the same value, not if their elements are equal.

Make sure you understand this difference!

In [ ]:
```
a = [1,2,3]
b = a      # A *reference* to a
c = a[:]   # A *copy* of a

print("a==b", a==b)
print("a==c", a==c)
print("a is b", a is b)
print("a is c", a is c)
```

In [ ]:
```
# This means that if we changed b, the value of a would change as well.
# But has no effect on c, which is a separate list
b.append("sentinel")
print(a)
print(b)
print(c)
```

Operators

Operators like `+` (concatenate) return new lists and do **not** modify lists in place:

In [ ]:
```
a = [1,2,3]
b = [4,5,6]
c = a + b         # c is now a new list which has the elements of a and b
#c.append("!")
print(a)
print(b)
print(c)
```

Mutable vs immutable operations

In Python, there is a simple rule:

- If a function (e.g. `a.sort()`) changes a data structure **in place** (i.e. **mutates** it), it always returns `None`
- If it does not change the original data structure, then it **creates an entirely new data structure** (e.g. `sorted(a)`), fills it with elements from the original list and returns the new data structure.
- Operators like +, * etc. always return new lists, and don't modify lists in place.

You should follow this convention in any code that you write!

The * operator on lists

Somewhat less usefully, the `*` operator is also overloaded; it repeats a list multiple times. The left operand must be a list and the right operand must be an integer:

In [ ]:
```
print(a*3)   # same as a + a + a
print(a*1)   # one repetition, same as a[:]
print(a*0)   # empty list -- note that it is quite ok to have an empty list
```

In [22]:
```
a = [1,2,3,4]
b = a[1:3]
b[1] = 5
print(a)
print(b)
```

```
[1, 2, 3, 4]
[2, 5]
```

Lists in Python

Dynamically sized

Python lists are dynamically sized: this means you can add and remove elements as you wish, without having to declare the size of the list in advance.

In [ ]:
```
a = [1, 2, 3]
a.append(4)  # we can just append values as we want
print(a)
```

In [25]:
```
word = "hello "
newWord = word
word += "there"

print(word)
print(newWord)

wordList = ["hello"]
newWordList = wordList
wordList += ["there"]

print(wordList)
print(newWordList)
```

```
hello there
hello
['hello', 'there']
['hello', 'there']
```

List operations

We can do many things with lists:

- index and slice `l[0]` `l[0:5]`
- join `list_a + list_b`
- add elements `list_a.append("one")`
- remove elements `list_a.remove("one")` or `del list_a[0]`
- sort `sorted(list_a)`
- test for membership `elt in list_a`
- find the index of elements `list_a.find("one")`
- copy them `list_a.copy()`

In [ ]:
```
l = [[1,2,3],
     [4,5,6],
     [7,8,9]]

print(l[0][0], l[0][1], l[1][0])
```

In [ ]:
```
# This would be the same as doing
# l[0][1] is just the same as
row = l[0]
elt = row[1]
print(elt)
```

Lab Exercises - Lists

1. Manipulating lists

Before running the following snippets of code try and predict what the answer will be.

In [30]:
```
numbers = [2,5,4,6,8,1,3,9,7,0]
```

A. Indexing

In [55]:
```
print(numbers[3])
```

In [1]:
```
print(numbers[9])
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-0a43a5f7864f> in <module>
----> 1 print(numbers[9])

NameError: name 'numbers' is not defined
```

In [57]:
```
print(numbers[-1])
```

In [58]:
```
print(numbers[-5])
```

In [59]:
```
# what does this print?

print(len(numbers))
```

B. Slicing

In [123]:
```
print(numbers[2:4])
```

In [62]:
```
print(numbers[0:6])
```

In [63]:
```
print(numbers[:6])
```

In [64]:
```
print(numbers[-4:-1])
```

In [65]:
```
print(numbers[-4:])
```

C. Stepping

In [67]:
```
print(numbers[::2])
```

In [68]:
```
print(numbers[1::2])
```

In [69]:
```
print(numbers[1:8:3])
```

In [70]:
```
print(numbers[::-1])
```

2. Colours

Without using a loop, using the following definition of **colours**, print out the following:

1. The first element of `colours`
2. The last element of `colours`
3. Every even element of `colours` (i.e. elements with even indices)
4. Every odd element of `colours`
5. The third to the sixth element of `colours`, inclusive
6. The last five elements of colours
7. Every third element of the first eight elements of `colours`, in reverse order (starting with the eighth element).

In [1]:
```
colours = ["red", "black", "orange", "yellow",
           "blue", "cyan", "green", "purple", "gray", "white"]
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Simple list operations

You are given three lists `x,y,z`.

Write code that will:

- join the lists into one list in the order `z,x,y`,
- and store the result in a variable `zxy`.
- then create a *copy* of `zxy` and call it `zxy_copy`.
- Append the string `"in the sunshine"` to the end of `zxy_copy`.

In [ ]:
```
# These are the lists you are given
x = [1,2,3]
y = [0,0,0]
z = [1,9,6,9]
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Bibliography

H. Andrade and Y. Du. Student responses to criteria-referenced self-assessment. *Assessment & Evaluation in Higher Education*, 32(2):159–181, 2007. doi: 10.1080/02602930600801928. URL https://doi.org/10.1080/02602930600801928.

H. Andrade and A. Valtcheva. Promoting learning and achievement through self-assessment. *Theory Into Practice*, 48(1):12–19, 2009. doi: 10.1080/00405840802577544. URL https://doi.org/10.1080/00405840802577544.

T. Boyle. Design principles for authoring dynamic, reusable learning objects. *Australasian Journal of Educational Technology*, 19(1), 2003.

T. Boyle. Technology and the reflective practitioner. *Effective learning and teaching in computing*, pages 182–188, 2004.

S. I. Chng. Incorporating reflection into computing classes: models and challenges. *Reflective Practice*, 19(3):358–375, 2018. doi: 10.1080/14623943.2018.1479686. URL https://doi.org/10.1080/14623943.2018.1479686.

A. Collins and J. S. Brown. The computer as a tool for learning through reflection. In *Learning issues for intelligent tutoring systems*, pages 1–18. Springer, 1988.

J. Dewey. *Experience and Education*. Kappa Delta Pi lecture series. Kappa Delta Pi, 1998. ISBN 9780912099354. URL https://books.google.co.uk/books?id=nkNaAAAAYAAJ.

T. Dummer, I. Cook, S. Parker, G. Barrett, and A. Hull. Promoting and assessing 'deep learning' in geography fieldwork: An evaluation of reflective field diaries. *Journal of Geography in Higher Education - J GEOGR HIGHER EDUC*, 32:459–479, 09 2008. doi: 10.1080/03098260701728484.

P. Ekman, E. Sorenson, and W. Friesen. Pan-cultural elements in facial displays of emotion. *Science (New York, N.Y.)*, 164:86–8, 05 1969. doi: 10.1126/science.164.3875.86.

S. A. Fincher and A. V. Robins. *Foundations*, page 79–322. Cambridge Handbooks in Psychology. Cambridge University Press, 2019.

G. Gibbs. *Learning by Doing: A guide to teaching and learning methods*. Oxford Polytechnic: Oxford., 1988.

Google. Firestore documentation, 2024. URL https://firebase.google.com/docs/firestore. Accessed: March 15, 2024.

E. Gordon. 'the good, the bad, and the ugly.' a model for reflective teaching practices in coaching pedagogy.". *Strategies*, 30, 01 2016. doi: 10.1080/08924562.2016.1251866.

D. P. Hunt. Effects of human self-assessment responding on learning. *Journal of Applied Psychology*, 67(1):75–82, 1982. doi: https://doi.org/10.1037/0021-9010.67.1.75.

IPython Development Team. Ipython magic commands, accessed 2024. URL https://ipython.readthedocs.io/en/stable/interactive/magics.html. Accessed: March 18, 2024.

Jupyter-contrib project contributors. Installing jupyter notebook extensions, accessed 2024. URL https://jupyter-contrib-nbextensions.readthedocs.io/en/latest/install.html. Accessed: Feb 21, 2024.

S. Khoirom, M. Sonia, B. Laikhuram, J. Laishram, and T. D. Singh. Comparative analysis of python and java for beginners. *Int. Res. J. Eng. Technol*, 7(8):4384–4407, 2020.

D. E. Knuth. Literate programming. *The computer journal*, 27(2):97–111, 1984.

A. J. Ko and S. A. Fincher. *A Study Design Process*, page 81–101. Cambridge Handbooks in Psychology. Cambridge University Press, 2019.

D. A. Kolb. *Experiential learning: Experience as the source of learning and development*. FT press, 1984.

H. Li, F. Bao, Q. Xiao, and J. Tian. Codepod: A namespace-aware, hierarchical jupyter for interactive development at scale. 01 2023. doi: 10.48550/arXiv.2301.02410.

LLAMA Development Team. LLAMA2 documentation, accessed 2024. URL https://llama.meta.com/llama2. Accessed: March 18, 2024.

C. Mooney, A. Antoniadi, I. Karvelas, L. Salmon, and B. A. Becker. Exploring sense of belonging in computer science students. ITiCSE '20, page 563, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450368742. doi: 10.1145/3341525.3393974. URL https://doi.org/10.1145/3341525.3393974.

J. F. Pimentel, L. Murta, V. Braganholo, and J. Freire. A large-scale study about quality and reproducibility of jupyter notebooks. In *2019 IEEE/ACM 16th international conference on mining software repositories (MSR)*, pages 507–517. IEEE, 2019.

K. Roskos, C. Vukelich, and V. Risko. Reflection and learning to teach reading: A critical review of literacy and general teacher education studies. *Journal of Literacy Research*, 33(4):595–635, 2001. doi: 10.1080/10862960109548127.

J. Russell, J. A. Posner, and B. S. PETERSON. The circumplex model of affect: An integrative approach to affective neuroscience, cognitive development, and psychopathology. *Development and Psychopathology*, 17(3):715–734, 2005. doi: 10.1017/S0954579405050340.

D. A. Schön. *The reflective practitioner: How professionals think in action*. Basic Books, New York, 1983.

D. A. Schön's. *Educating the reflective practitioner: Toward a new design for teaching and learning in the professions*. Jossey-Bass., 1987.

SciPy. scipy.stats.ttest_ind. URL https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_ind.html. Accessed: March 20, 2024.

H. Shen. Interactive notebooks: Sharing the code. *Nature*, 515(7525):152–152, 2014.

T. Y. Sim and S. L. Lau. Online tools to support novice programming: A systematic review. In *2018 IEEE Conference on e-Learning, e-Management and e-Services (IC3e)*, pages 91–96, 2018. doi: 10.1109/IC3e.2018.8632649.

University of Edinburgh. Gibbs' reflective cycle. 11 2020.