CIC Assignment 1 - Report

Christopher Dalziel

November 2024

1 Introduction to Materials

The materials for this can be accessed here via my GitHub and consists of an exercise worksheet and a zip file of Python files split into exercises (or "snippets") which can be run using the online micro:bit Python editor. Corresponding sample solutions and a teacher's copy of the pdf with answers to written questions have also been provided.

These materials aim to cater to Scottish Higher students, who should be familiar with the basics of lists (or another array-like structure), such as indexing and traversing with loops, as they are a part of the National 5 curriculum [1].

The exercises cover material largely beyond what should be expected to be common knowledge for the target audience. They will rely on students' ability to read and understand documentation. The relevant sections of the documentation are linked to as part of each exercise.

All of the materials were created by me, with the notable exceptions of the `music` library native to the micro:bits (which I invoke as part of making the simpler `play()` function the students will interact with) and the interactive circle of fifths [2] which is courtesy of Rand Scullard - an incredibly useful resource for understanding the general shape of the circle of fifths, which is handy for the later exercises.

1.1 Worksheet

The worksheet is split into five exercises which are intended to be completed sequentially, with each exercise being completed and then discussed as a class.

The tasks are designed using some music theory as a way to motivate writing and analysing small programs working with lists. Music theory is a background element, and completing the tasks should not require any formal musical education.

After each code snippet is a leading question about the function of the code, intended to have the student test their code and come up with a potential answer to be brought up during the discussion.

The flaws in the code have been devised such that if they are hard to spot visually they should be easily heard when running the file.

The sample solution is a copy of this worksheet with boxes added below each exercise outlining the solution and the answer to any questions asked about the code. Teachers should be expected to ask leading questions to guide students towards this answer during discussions.

1.2 Python Files

Each of the five exercises has a corresponding folder in the `exercises` (or `solutions`) folder. Due to limitations of the micro:bit editor, files have to be manually imported per exercise.

Students will be expected to work only in the `main.py` file, with other files abstracting implementation details of helper functions which may be unnecessarily confusing. Unfortunately, these files cannot be abstracted from students so implementations will be visible.

2 Motivation

These exercises use lists and music as a background to teach students some fundamental lessons in a way that I feel is engaging. As a learner, I find lessons which have a satisfying final product always last longer in my mind than those that do not.

Material-wise, the exercises are first and foremost about debugging and working with unfamiliar tools, with a focus on reading documentation. These are skills I feel are vitally important for anybody who writes software, yet were not taught well in my formal education.

The two major activities a programmer engages in when developing software are programming and debugging. Of the two, debugging generally takes more time, and more often than not takes the most work too. From this perspective, students must experience debugging early [3].

The SQA's National 5 curriculum contains no mention of debugging, focusing purely on writing code and not necessarily on understanding that code [1]. The Higher curriculum [4] does marginally better in covering debugging methods, but does not in my opinion give it adequate time relative to its importance. The Advanced Higher curriculum [5] does something to address working with unfamiliar tools via the project, and even requires formal testing as part of projects.

While it is good that students are eventually taught these things, I often felt that what was covered in class didn't represent my practical experience.

Debugging is one of those skills that can be intensely frustrating for students to learn, but it's also perhaps the most deeply important one [6]. Trying to cover the topic in a way that reduces this frustration (in this case by making the error not "their fault") and encourages constructive habits like reading documentation should hopefully begin to build a foundation for students to handle cases of "organic" bugs they encounter in the future.

An important factor in developing the exercises was making the problems simple enough that students could be expected to either detect the faults themselves (via hand testing or logical reasoning) or understand the faults when they were discussed.

I feel that getting students started on solid ground with regard to making sure their software works as intended is an important part of their journey towards feeling confident that they can fix their code, even when it's an error that they've never seen before.

3 Related Work

There are of course other exercises with a focus on debugging in this way, but as it turns out there are not as many as I had expected. Those which were free to access and that I could find online were all part of online courses.

3.1 LaunchCode Education

LaunchCode Education is a company which provides free online programming courses which guide first-time programmers through the basics of programming to have them proficient enough by the end of the book to feel confident enough in their programming skills to enter the workplace.

Their courses on Web Development in JavaScript, Programming in C#, and Data Analysis in Python each contain a chapter [7]–[9] covering debugging basics. The final section of each of these chapters is a short debugging exercise focusing on ensuring that a rocket launch goes correctly by making certain that the code has no bugs.

The general pacing of exercises getting subsequently more difficult paired with using an engaging framing device (in this case the idea of safely launching a rocket) to make things more interesting for prospective learners makes these exercises a close match to the ones I've developed.

Early placement of debugging in these courses allows debugging to become part of software development in the minds of its students, as opposed to being un-

covered or treated like an afterthought.

While these courses are by no means examples of a debugging-centric approach, they do at least provide students a better view of debugging than many other courses do, and certainly better than no coverage at all.

3.2 Software Carpentry - Programming with Python

The Carpentries is a non-profit organisation that aims to make programming and data skills more accessible via their online courses [10]. Their three projects are Software Carpentry, Library Carpentry, and Data Carpentry. While all three are software-oriented, only the Software Carpentry course on Python programming contains an explicit debugging section.

Chapter 11 covers the general principles of debugging, establishing the general goals and procedures for debugging programs [11]. In this regard, it does very well, as the advice it gives is generally sound.

There are two exercises on the page, the first being a cooperative exercise (the merits of which cannot reasonably be examined here) and the second is a broken BMI calculator which students are tasked to fix.

This exercise arguably works, in that it does make students apply their understanding to detect bugs. Where it fails is in being rather uninspired and unmemorable. The framing device here is perhaps more representative of a real system than the rocket example used by LaunchCode or my music lesson but is very bland by comparison. This blandness makes the exercise a lot less memorable, and by extension makes the lesson attached to it less likely to be retained by students.

4 Educational Theory

4.1 PRIMMDebug

Debugging as an exercise has a fundamentally PRIMM-ish structure; involving analysing our understanding of a program (Predict) and comparing it to that program's actual behaviour (Run, Investigate), before altering the program to better match the expected behaviour (Modify).

PRIMMDebug [12] is a recently proposed extension of the PRIMM framework for teaching debugging in the classroom. The argument is that by designing exercises around fixing incorrect code students can be guided towards finding solutions in an organic way, which should give them a solid foundation for debugging their own code in future.

Exercises for PRIMMDebug start with a code chunk containing ideally a single

error and a description of the ideal functionality. Learners are guided through a series of steps in accordance with PRIMM (though with reduced attention paid to the make stage). Through repeatedly performing such exercises students should become more independent debuggers (and by extension programmers), reducing the amount of individual support they need from educators.

4.2 Debugging-First Pedagogy using TAMP

The Theory of the Applied Mind for Programming (TAMP) [13] is a framework for understanding the ways that novice programmers struggle, with the aim of guiding the design of new pedagogical approaches to attempt to address the common points of difficulties experienced by new programmers.

One such problem area is predictably debugging, which is difficult to teach in isolation as it demands a lot of attention per student from a teacher, while simultaneously being frustrating for students to learn [6], [14] - nobody likes struggling with the seemingly insurmountable.

Where frameworks like PRIMMDebug can to a certain extent reduce the split-attention problem that teachers face, a more radical restructuring of how we teach programming seems to be the best way to help intermediate learners bridge the gap and stop feeling daunted by debugging - teaching debugging first [3].

Psychologist Daniel Kahneman described two classes of thinking exercises - which he referred to as "System 1" (fast and instinctual) and "System 2" (slow and reasoned) thinking [15]. The argument provided in [3] is that debugging is a fundamentally System 2 exercise and that once learners are comfortable with programming in general it is largely handled by System 1 - our reflexive understanding of programming concepts and problem-solving tools.

The result of this is that students find themselves spending most of their time on hard mental work, which can often be frustrating and confusing. If then we were to focus earlier on developing students' debugging skills, we could simultaneously familiarise students with the general pattern of debugging (making this kind of thinking more habitual) and normalise debugging itself, making it a less intimidating concept.

4.3 Applications in my Exercise

The central idea underpinning my exercises is that debugging is both an important tool for students to learn and a useful way for those same students to learn new programming concepts. The research seems to suggest that fairly early exposure to novice programmers is the best way to benefit learners and that continual exposure will reinforce the habits and make students more confident.

I feel that giving students the opportunity to engage with the material directly

via small implementation sections is the best way to teach them to use new skills (in this case unfamiliar Python list operations) in a debugging-heavy format. By adding short discussion questions after each exercise students are encouraged to engage critically with the material they have just completed, and to recognise errors in code that they did not necessarily write.

Although not phrased as debugging questions, the actual structure of my exercises encourages learners to approach the problems at various stages of the PRIMMDebug process. They are structured so that learners are provided a sample of code which they look at, and after filling in a small section according to the specification they are encouraged to think about how the code's actual function compares to what the code is doing when they run it.

The lesson structure is inherently constructivist, as students are given an opportunity to discuss their ideas with the rest of the class with their teacher able to sway the discussion in the general direction of an answer. The hope is that these discussions give students an opportunity to apply problem-solving skills without having to worry about getting stuck without an answer.

The discussion questions are arguably the most important part of the exercises. They are designed to make students engage with PRIMM more directly, specifically with predicting, investigating, and running - with students potentially proposing modifications during discussion.

The final section I treat as largely an extension task, letting students use what they've learned and what they've worked on up to that point to create a larger program. This is very much the "Make" section of the PRIMM model, and while I feel that focusing on the other components is more important it felt like the best way to conclude the worksheet.

5 Future Work

The intent of this exercise is first and foremost to normalise debugging for students and to make it less intimidating. Doing so in a single exercise however is unrealistic, and so future work would be to create a wider range of exercises at a range of different levels.

It is my opinion that introducing similar exercises at a National 5 level would be particularly valuable, as it would introduce students as early as possible to the kind of problem-solving that debugging often requires - even without necessarily touching on tools like trace tables or step-through debuggers.

Taking the "debugging-first" mindset to its extreme, it would also be useful to have exercises designed for non-text languages such as Scratch and for simple early programming lessons like "Bee-bots".

By having exercises like these spread through a students' education we can hopefully make debugging a smaller hurdle for novices so that when they do get to the level that formal debugging tools are introduced they will already have a basic understanding of where they can apply them.

6 Conclusion

Debugging is an important part of programming and is a vital part of a student's programming education. The exercise provided lets students engage with debugging in a way which should be engaging and less frustrating than when they are working on their own code, to prepare them for handling bugs in their own code using a PRIMM-like approach.

References

[1] "SQA National 5 Specification - Computing Science." en. (2024), [Online]. Available: https://www.sqa.org.uk/files_ccc/n5-course-spec-computing-science.pdf.

[2] R. Scullard. "Interactive Circle of Fifths." (2007), [Online]. Available: https://randscullard.com/CircleOfFifths/.

[3] T. Lowe, "Debugging: The key to unlocking the mind of a novice programmer?" In *2019 IEEE Frontiers in Education Conference (FIE)*, 2019, pp. 1–9. DOI: 10.1109/FIE43999.2019.9028699.

[4] "SQA Higher Specification - Computing Science." en. (2024), [Online]. Available: https://www.sqa.org.uk/files_ccc/h-course-spec-computing-science.pdf.

[5] "SQA Advanced Higher Specification - Computing Science." en. (2024), [Online]. Available: https://www.sqa.org.uk/files_ccc/ah-course-spec-computing-science.pdf.

[6] L. Gale, "Debugging: A powerful and dangerous skill to learn," *Raspberry Pi Computing Education Centre*, 2023.

[7] "Introduction to Programming in C#, Chapter 6.8 - Exercises: Debugging." (2024), [Online]. Available: https://education.launchcode.org/intro-to-programming-csharp/chapters/errors-and-debugging/exercises.html.

[8] "Introduction to Web Development, Chapter 6 - Exercises: Debugging." (2024), [Online]. Available: https://education.launchcode.org/intro-to-web-dev-curriculum/errors-and-debugging/exercises/index.html.

[9] "Data Analysis, Chapter 10.8 - Studio: Errors and Debugging." (2024), [Online]. Available: https://education.launchcode.org/data-analysis/chapters/errors-and-debugging/studio.html.

[10] "About Us." (Nov. 2024), [Online]. Available: https://carpentries.org/about-us/.

[11] A. Bostroem, T. Bekolay, and V. Staneva. "Software Carpentry: Programming with Python, Chapter 11 - Debugging." (Jun. 2016), [Online]. Available: https://github-pages.arc.ucl.ac.uk/python-novice-gdp-penguins/instructor/11-debugging.html#know-what-its-supposed-to-do.

[12] L. Gale. "PRIMMDebug: Explore a new way of teaching debugging." (Oct. 2024), [Online]. Available: https://computingeducationresearch.org/blog-introducing-primmdebug/.

[13] T. Lowe, "A theory of applied mind of programming: Understanding the challenges in learning to program," in *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '19, Aberdeen, Scotland Uk: Association for Computing Machinery, 2019, pp. 340–341, ISBN: 9781450368957. DOI: 10.1145/3304221.3325600. [Online]. Available: https://doi.org/10.1145/3304221.3325600.

[14] P. Kinnunen and B. Simon, "Experiencing programming assignments in cs1: The emotional toll," in *International Computing Education Research Workshop*, 2010, pp. 77–85. [Online]. Available: https://api.semanticscholar.org/CorpusID:6123209.

[15] D. Kahneman, *Thinking, fast and slow*. New York: Farrar, Straus and Giroux, 2011, ISBN: 9780374275631 0374275637. [Online]. Available: https://www.amazon.de/Thinking-Fast-Slow-Daniel-Kahneman/dp/0374275637/ref=wl_it_dp_o_pdT1_nS_nC?ie=UTF8&colid=151193SNGKJT9&coliid=I3OCESLZCVDFL7.