COSC470 Report

Lucy Turner
ltt19@uclive.ac.nz
University of Canterbury

Tim Bell - Supervisor
tim.bell@canterbury.ac.nz
University of Canterbury

October 25, 2019

1 Abstract

With digital technology education becoming a larger part of the New Zealand school curriculum it is important to ensure that our teachers are well equipped to provide quality education to New Zealand's children. Unfortunately, a large amount of the teachers implementing the new education standards have little background in programming and other computer science skills. A SENG402 student in 2018 at the University of Canterbury built the foundations of the online programming education platform called codeWOF in order to help primary and secondary school teachers maintain their programming skills. This research paper investigates whether teachers experience decay of their programming skills when they go unused, what the usage trends of the codeWOF are platform and what types of common programming mistakes are made by codeWOF users. To investigate these questions a literature review and a 64 day long study of codeWOF users were undertaken, where data relating to each question submission by study participants was collected for analysis. The literature review clearly showed that skill decay does occur amongst teachers who do not use their programming skills regularly. The 20 study participants collectively made 1336 attempts on the questions available to them. From these attempts I have identified areas of further research and development for codeWOF in the future, including the addition of automated email reminders and how might affect usage metrics, and further improvement of the Parson's problem interface. User retention rates indicates that

codeWOF provides value to it's users, though this conclusion is limited due to the sample size.

Contents

2 Introduction

Computer science and computational thinking have been introduced into the New Zealand school curriculum under “Digital Technologies”, starting with NCEA Levels 1, 2 and 3 between 2011 and 2013 (Thompson, Bell, Andreae, & Robins, 2013, p. 2), and is in the primary and high school curriculums starting from next year - 2020 (*New digital technologies for schools and kura*, n.d.). As a consequence of this introduction, existing teachers have needed to up-skill rapidly to be able to teach their students. In 2011, 48% of the teachers implementing the new standards had “rudimentary” programming skills, and 7% had no programming skills (Thompson et al., 2013, p. 3). There are many tools available to these teachers for learning programming, including online and in-person courses.

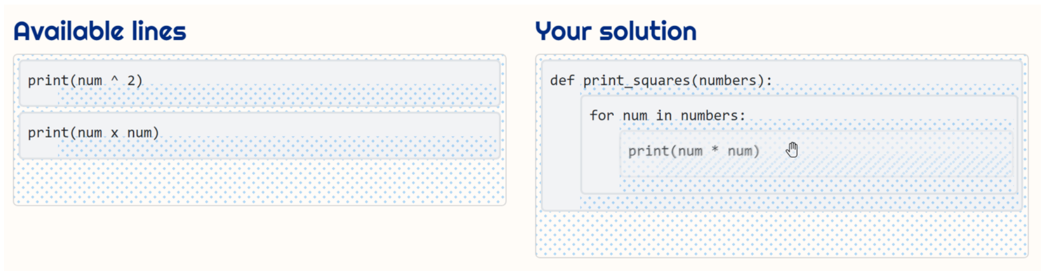
Teachers from intermediate and secondary schools are able to use text-based languages to teach the digital technologies curriculum, however there are periods of time where the teacher may not be exercising their programming skills. There are many potential reasons for this, such as not teaching programming over the summer holidays, or only using text-based languages at certain points during the year. A common issue appears to be that when teachers come back to programming after having a break from it, they struggle to remember the syntax of the language, as well as common patterns, e.g. updating a list.

Research Questions

1. Do teachers experience decay of their programming skills if they are not using them regularly?
2. What are the usage trends of codeWOF?
3. What type of programming mistakes do users make when using codeWOF?

This project investigated whether teachers struggle to maintain their programming skills if they are not using them consistently. The codeWOF system (*codeWOF*, n.d.), originally known as BitFit, that was started in 2018 at the University of Canterbury by Jessica Robertson, and which is currently being worked on by Maree Palmer, aims to make it easier for teachers to retain their programming skills. It does this by regularly prompting them to do small programming problems in Python. Regular practice of skills has already been shown to help decrease skill decay in other areas (Mitchell et al., 2011; Smith & Scarf, 2017; Baturay, Yıldırım, & Daloğlu, 2009). Python was chosen as it is a popular programming language which was used by 78% of the teachers surveyed by Thompson et al. (2013), so it is a relevant choice as the first language that codeWOF supports.

There are three different types of problems on CodeWOF: functions, programs, Parsons problems (Parsons & Haden, 2006), and debugging problems. For standard problems a full program is written and submitted by the user. Parsons problems are problems where the lines of code for the solution are given in the wrong order, and must be re-arranged to solve the problem. Debugging problems are where a faulty program is presented to the user and the mistake(s) in this given program must be fixed to solve the problem.



```
print(num ^ 2)
print(num x num)
```

```
def print_squares(numbers):
    for num in numbers:
        print(num * num)
```

Figure 1: Screenshot of a Parson's problem solving area on codeWOF

We care about the usage trends of codeWOF as we can use this information to help target the gamification that Maree has introduced (Palmer & Bell (Supervisor), 2019). This includes the points associated with answering a question, email reminders, and question difficulty. User retention can also be an indicator of how useful a website is to it's users, so is one of the metrics that I have investigated.

The types of programming mistakes that people make when using codeWOF are worth investigating as some of them could be related to codeWOF itself - like bugs in codeWOF, misleading or broken questions or could be problems that we can help users fix. Knowing the other mistakes that users make can also be helpful when compiling a list of resources to point users to when they are stuck.

2.1 Related Work

2.1.1 Skill Decay

Skill decay exists when skills are not used on a regular basis. An example of this is from Anderson, Gaetz, and Masse (2011) where first aid skills of first responders in the workplace were measured and were found to decay rapidly after original training. Teachers specifically have been shown to suffer from skill decay of the teaching skills they were taught when studying, both while they were still studying and once they had begun teaching professionally (Scheeler, 2008). Skill decay has also been shown to increase over time by Ellington, Surface, Blume, and Wilson (2015) and Arthur Jr, Bennett Jr, Stanush, and McNelly (1998). Some of the other factors that have been shown to increase skill decay include the degree of overlearning that the person had undertaken, "motivation and individual differences in skill retention", and it was found that tasks such as "physical, natural, and speed-based tasks were less susceptible to skill loss than cognitive, artificial, and accuracy-based tasks" (Arthur Jr et al., 1998). A theory proposed by Kim, Ritter, and Koubek (2013) explains skill retention based on the ACT-R theory of learning where learning declarative and procedural knowledge is done at the same time as forgetting, but that once procedural knowledge is solidified it decays more slowly than declarative knowledge.

2.1.2 Practice and Retention

Practice and refresher courses have been shown to improve (i.e. decrease) skill decay rates in areas such as complex motor skills used in surgery by Mitchell et al. (2011) and by Kluge and Frank (2014) for machinery start-up processes. Smith and Scarf (2017) found that "in both adults and children, spacing generally enhances the generalization of learning and the retention of words, grammar, and skills". Smith and Scarf provided an explanation for this saying "spaced repetitions enhance the consolidation of memories to a greater extent than massed repetitions and providing time for memories to consolidate enhances the consolidation/reconsolidation of additional learning that can befit into the same framework, resulting in faster learning and better retention". Comings (1995) also found that post-literacy programs helped literacy skill retention of adults in developing countries, saying that "people will probably lose their skill if they do not use it, and the provision of a program that encourages the use of literacy skills will allow new readers to retain or improve those skills".

Different types of practice have been shown to effect skill decay (Stevens & Gist, 1997; Sauer, Hockey, & Wastell, 2000) and a larger range of problem types have been shown to help teachers reduce skill decay (Rose, 1994). Rose says "provision should be made to ensure that teaching skills are practised in a range of contexts. Classrooms vary on a range of dimensions and it is important that the stimuli of a single training setting do not acquire exclusive stimulus control over a teaching skill". Baturay et al. (2009) specifically looked at how a web-based spaced repetition program with a range of problem types influenced skill decay of vocabulary. This is especially relevant to the maintenance of programming skills and Research Question 2 as there are parallels between learning the vocabulary of a natural language and learning a programming language. This study also tested fill-in-the-blank/cloze questions which are similar to Parson's problems (Parsons & Haden, 2006) and is also web-based practice like codeWOF. Overall, it was found that the web-based system "proved to be effective in increasing the retention of participants' vocabulary through spaced repetitions".

Retention of a variety of skills has been shown over varying amounts of time. Mitchell et al. (2011) investigated retention of a surgical skills and found no difference between practising the skill every week for four weeks, or every month for four months. Bahrick, Bahrick, Bahrick, and Bahrick (1993) found that practising a English and foreign language word pairs at intervals of 14 days had similar effects of improving long-term retention as 56 days. Bloom and Shuell (1981) found that high school students practising french vocabulary for 10 minutes a day over three days had improved retention compared to students practising for 30 minutes in one day. It is unclear based on existing literature what the optimal amount of time between programming practice sessions would be. This is an area of research that requires further investigation.

3 Methodology

3.1 Research Question 1

After performing the above literature review on skill decay, I decided that further investigation was not necessary to answer Research Question 1.

3.2 Research Questions 2 & 3

To identify usage trends of codeWOF and identify the programming mistakes made by users, I collected attempts made by users on codeWOF. Attempts were only collected from users who opted-in to this study, and users who chose not to opt-in were still able to use codeWOF without any penalty.

For each attempt made on codeWOF by a user in the study I collected:

- The date and time of the attempt
- Their user ID (a number)
- The ID of the question they were answering (a number)
- The code they submitted
- Whether the code passed the tests or not (True/False)

Participants were recruited through the Digital Technologies Teachers Aotearoa (DTTA) noticeboard, which had approx. 900 members when the advertisement was released. Members include primary, intermediate, and secondary school teachers of Digital Technologies, as well as industry, university and professional development providers. Both codeWOF and the study were advertised to DTTA members.

After a user signed up to codeWOF, they could opt-in to this study through a link on their dashboard. From there they were presented with an information sheet and a consent form. On the consent form they needed to confirm that they were at least 18 years old, and that they had programmed in Python before. Participants had to confirm that they had used Python before because only the submissions of people practising Python were being analysed, not the submissions of people learning Python for the first time. Most, if not all, of the participants in the study would have been teachers as they were the main targets of our advertising, but non-teachers were also permitted to participate in the study.

After a participant had read and agreed to the information and consent forms, no extra effort was required from them. They only needed to continue to use the website normally. This was intentional to try and make it as easy as possible to participate in the study so that I would have enough participants to be able to draw conclusions.

Automated email reminders were intended to be introduced to codeWOF as part of Maree's project (Palmer & Bell (Supervisor), 2019), however they were not able to be integrated due to time constraints. For the participants in this study I manually sent out these reminder emails so that they would remember to use the website.

I had originally planned to investigate whether different types of questions (functions, programs, Parson's or debugging) had an effect on programming skill maintenance. However, I was unable to recruit enough participants to be able to answer this question. Because of this initial research focus, there are three groups of participants in my study. The first group answered standard questions - functions and programs only. The second group answered standard questions and Parson's problems. And the third group answered standard questions and debugging problems. Participants were assigned into one of the three groups upon signing up to the study. The second group (functions, programs and Parson's) contained 7 participants, and the other two contained 8 participants each.

Even though participants in the study did not all have access to the same types of questions, I do not believe that this would have a large effect on the usage trends, or on the types of mistakes found in the solution.

3.2.1 Preparation of codeWOF

The University of Canterbury Computer Science Education Research Group (CSERG) were heavily involved in preparing codeWOF for release, especially Jack Morgan. The CSERG also provided valuable feedback on questions written for codeWOF and on the interface of the website. I wrote approximately half of the questions available to be answered on codeWOF, and the CSERG was also involved in writing and organising the rest of the questions.

4 Results & Discussion

4.1 Research Question 1

As stated in the methodology section, after performing the literature review above on skill decay, I decided that further investigation was not necessary to answer Research Question 1. It is clear based on the literature review that decaying of skills occurs when they are not being used. It is also clear that regular practice of skills increases long and short-term retention of that skill. Therefore, I can conclude that teachers would experience decay of their programming skills when they are not being used regularly.

This provides a solid foundation for the future of codeWOF as it shows that codeWOF, as a practice tool for programming, would help teachers (and other people) to maintain their programming skills when they would not otherwise be used.

4.2 Research Question 2

4.2.1 Overall Statistics

The 20 teachers who participated in the study made 1336 attempts on the questions available to them, with 529 questions attempted collectively. The study ran for 64 days, from the 16th of August to the 18th of October. Two participants never answered any questions, and two participants finished all of the questions that were available to them towards the end of the study. Participants in the group that answered standard questions only were able to answer a maximum of 55 questions. Participants in the group that answered standard and Parson's questions were able to answer a maximum of 80 questions. And participants in the group that answered standard and debugging questions were able to answer a maximum of 78 questions.

4.2.2 Number of Questions Answered

I calculated the total number of questions attempted by each user, and the distribution of these values can be seen in figure 2. The number of questions attempted per user is positively skewed. One quarter of participants attempted between 0 and 3.2 questions while in the study. This is expected as not all participants will continue to engage with the website after they first sign up. Some reasons for this may include not having enough time to use the website, forgetting about the website (e.g. the reminder emails might have been flagged as spam), or finding that the website is not what they expected or they do not find it useful. Another quarter of the participants attempted between 39.5 and 78 questions while in the study. This does not necessarily indicate that they answered a question every 1 to 2 days as the participants were in the study for varying lengths of time. A distribution similar to figure 2 might be seen in usage data outside of the study but the sample size of 20 participants is too small to predict that. Compared to this study, there would probably be a greater proportion of users who answer less questions in the total population of codeWOF as a result of selection bias. I would

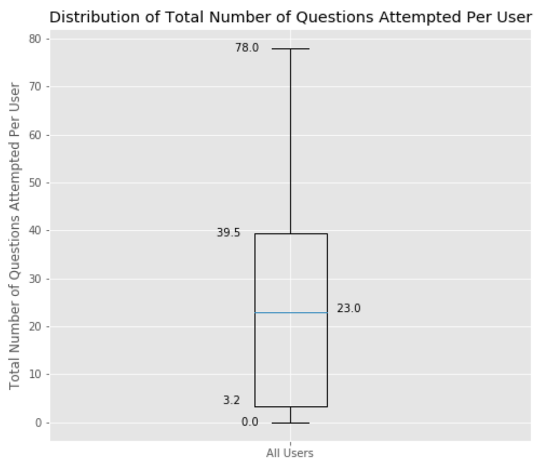


Figure 2: Distribution of total number of questions attempted by each user

expect users who are more engaged with codeWOF to be more likely to opt in to this study.

I also calculated the mean number of questions that each user attempted per day that they were in the study (figure 3). Half of the participants answered on average between 0.4 and 1.2 questions per day. This is equivalent to answering approximately one question every 1 to 2 days. However, the participants tended to use the website for both massed and spaced practice.

Figure 4 provides a good visualisation of some of the types of users we expect to see using codeWOF. Some users attempt questions consistently, shown on the graph as lines that continue to increase over time and occasionally flatten out. Others can be seen to“binge” on questions soon after they sign up, and at some other points in time. Because the aim of codeWOF is to encourage people to regularly practice small programming questions, on each user's dashboard there are two recommended questions for that day (figure 5). This feature exists to try and nudge users towards doing small amounts of work each day, rather than massed practice. However, these massed practices can still be useful if done repeatedly.

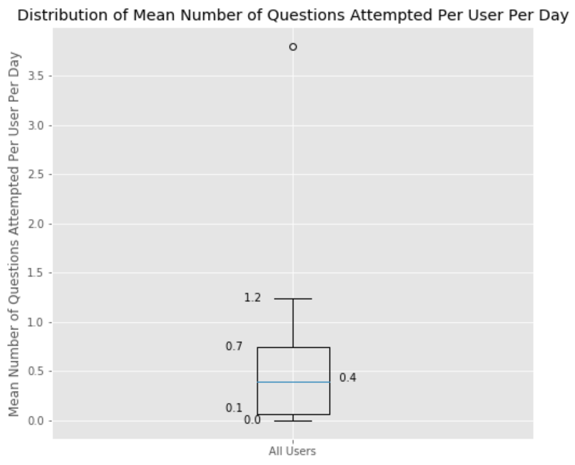


Figure 3: Distribution of mean number of questions attempted by each user per day

4.2.3 Number of Attempts Made

Figure 6 shows the distribution of how many attempts participants made per question. The number of attempts participants made on a question appears to decay exponentially. Of the 529 questions attempted collectively, over half (286 questions) were attempted only once, and the rest had between 2 and 29 attempts. Only 6 of the questions that had one attempt failed the tests, the other 280 passed on the first attempt. This means that 53% of all questions were answered correctly on their first attempt, and 47% failed their first attempt.

When designing the gamification of codeWOF (Palmer & Bell (Supervisor), 2019), Maree originally considered a points system where each incorrect attempt would decrease the amount of points that could be earned by a question. This approach was replaced by a system where each question correctly answered is worth 10 points, with an extra two points available if the question was answered correctly on the first attempt. The reasoning behind this was that losing points for each incorrect attempt could be discouraging.

Some of the questions on codeWOF are harder than others, and therefore require more attempts. The original points system would have awarded less points to users who answer difficult questions that require more attempts,

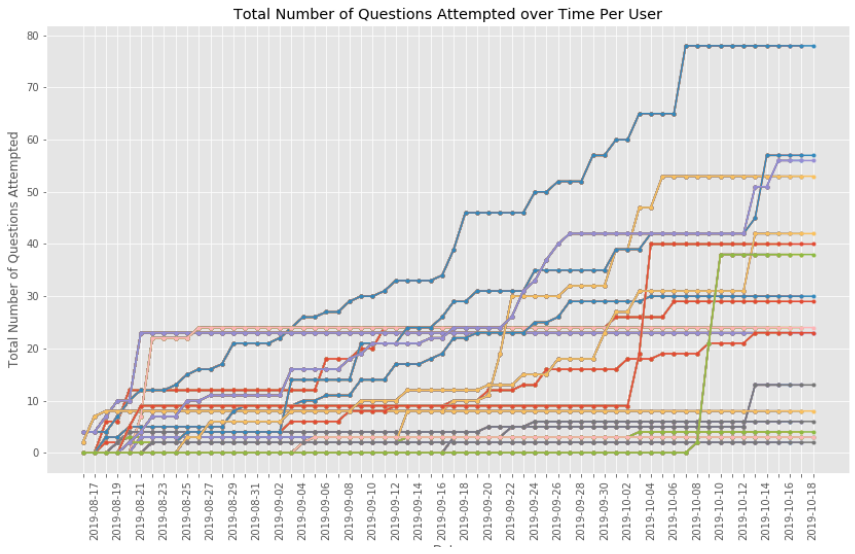
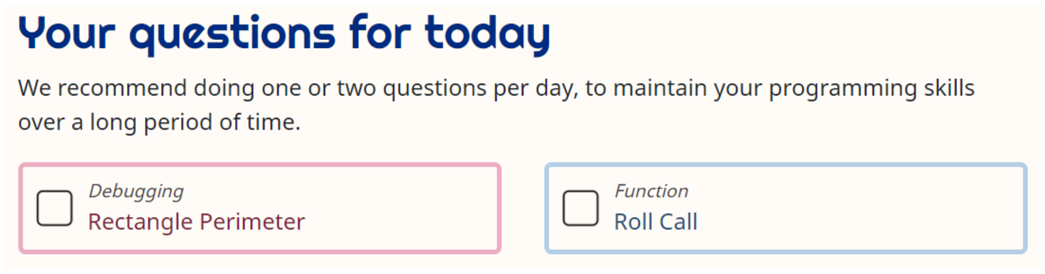


Figure 4: Total number of questions attempted over time per user (cumulative)

than it would to users who answer easier questions that can be answered with less attempts. The new points system is much more appropriate for this case, as well as because a large portion of questions (47%) require more than one attempt.

4.2.4 Number of Attempts Per Day

In figure 7, from the 16th of August to the 22nd of August we saw a large spike in usage because of people signing up to codeWOF and “bingeing” the system. The codeWOF domain was automatically blocked by some school firewalls until we asked for it to be white-listed, so some teachers were not able to access codeWOF for the first two days. On the 13th of October we saw the 3rd highest number of attempts made during the study. This is likely because the school holidays finished on the 13th of October, so teachers were getting some last practice in before a busy first week back teaching. After this, the number of attempts dropped, and over the last three days of the study there were no attempts made. After the initial rush of use when the website was launched, there was approximately consistent level of attempts


Your questions for today

We recommend doing one or two questions per day, to maintain your programming skills over a long period of time.

Figure 5: Screenshot of recommended questions section of codeWOF dashboard

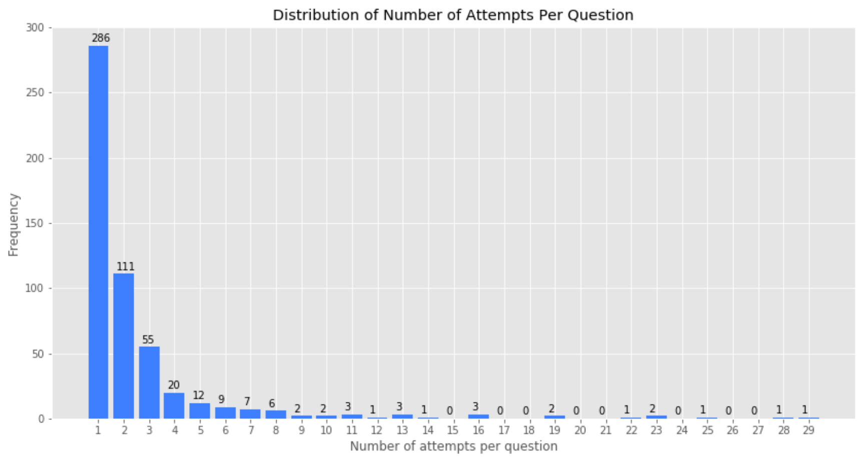


Figure 6: Distribution of number of attempts made for each question

made per day.

I sent out reminder emails on Mondays/Tuesdays and Thursdays/Fridays (see table 1 in the appendix for the precise dates). I only sent a reminder email to those users who had not answered a question in the previous two days. I did this so that each user would receive a maximum of two emails per week. I did not want the users to receive so many emails that they would start to ignore them, and I wanted the reminders to be relevant to whether they were actually using the website or not. More investigation is needed in the future to determine how often the reminder emails should be sent, and what days of the week they should or should not be sent out.

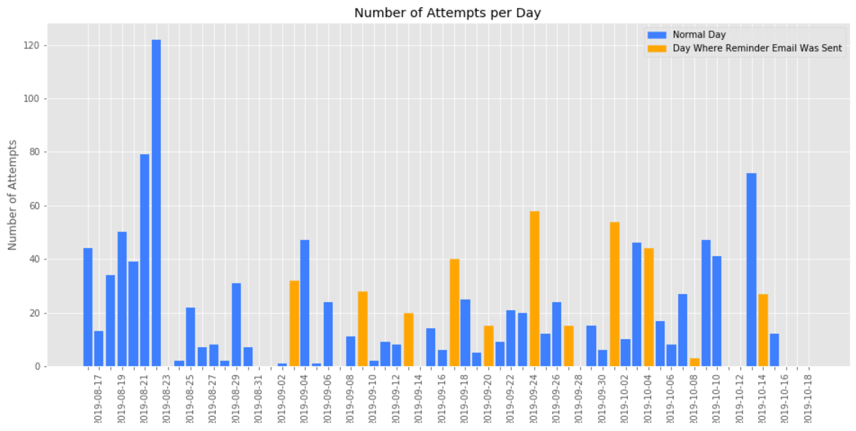


Figure 7: Number of attempts made per day, including annotation of when reminder emails were sent

With the information gathered it would be hard to draw conclusions about whether the reminder emails impacted the use of codeWOF, though I would expect that they had a positive effect on usage rates.

4.2.5 Attempts Per Day of Week

In figure 8 it was interesting to see that the least usage was on Saturdays considering that the reminder emails were sent on Mondays/Tuesdays and Thursdays/Fridays. Intuitively, I expected that the usage of the site would increase on day of the reminder email, as well as the day after the email (assuming some people would take longer to check and act on emails), but in this case the usage drops significantly the day after the email (Saturday). Again, I am unable to conclude any relationship between the reminder emails and usage rates without controlling for other factors.

Furthermore, if this trend were to continue within a larger group of people, I would recommend removing the streaks (introduced in Maree's project (Palmer & Bell (Supervisor), 2019)) from weekends. If users are unwilling to use the website on Saturdays, the motivational effect of getting or maintaining a streak may be diminished if Saturdays were to count towards a streak. Another option would be to aim for 5 days of using the website per week, and then it could become more flexible to the schedules of the user. This could also be more realistic for the students who are also using the website

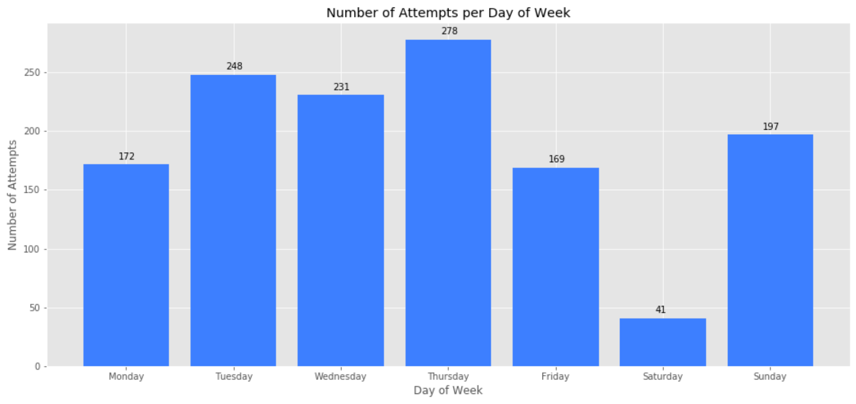


Figure 8: Number of attempts made per day of the week

as the usage of codeWOF may only be in the classroom, so the students may not use it on the weekend.

4.2.6 Retention

For this study I considered retention (figure 9) to be the percentage of users who came back on a specific number of days since their first question attempt in the study or any time after that day. Users who signed up to the study, but did not make an attempt during the study were considered to be part of the total users for day 0, but were not counted as retained for day 0. Retention was affected by two users running out of questions. No one answered any questions on the last three days of the study, which is why the retention rate dropped to 0. Because people did not all sign up on the same day, the end of the school holidays fell on different days of use for different people, so when people stopped using the website during the first week of term 4 it would have been around days 49 to 55, which coincides with a significant decrease in the retention rate. Before that point, the retention rate was decreasing, which is to be expected, but was sitting at around 50%. It may be high compared to retention rate bench marks because the people in the study may be more likely to be those who use the site more often. In comparison, the average retention rate 8 weeks after a user's first event on a website is approximately 20% (*What's A Good Retention Rate | Topics*, n.d.).

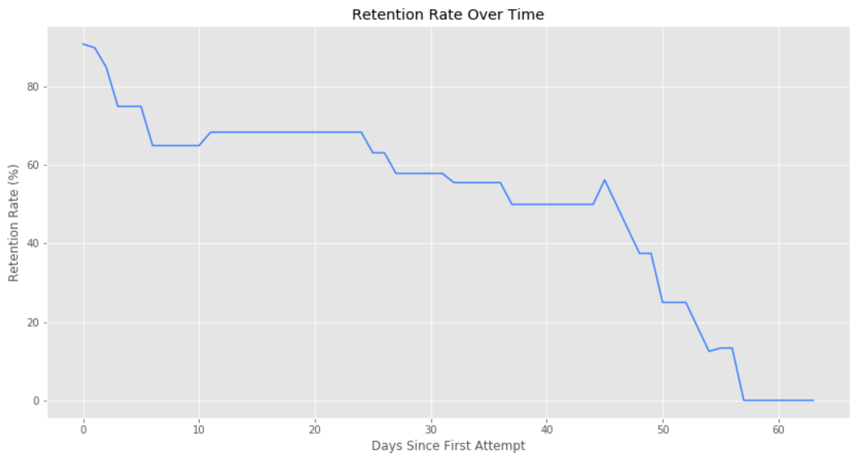


Figure 9: User retention over time

4.3 Research Question 3

To find out what types of programming mistakes users make when using codeWOF I analysed twenty-six out of the thirty-six questions that took an average of two or more attempts to answer. These questions had approximately 570 attempts between them, and I noted the errors in each of the attempts and grouped them into thirty different categories. Six of the problem types were related to codeWOF (figure 10), and the other twenty-four were related to question understanding or Python fluency (figure 13).

4.3.1 Problems Related to codeWOF

Including tests Fifty of the attempts contained tests in the submitted code. When this happens the tests written by the user and the tests in the table below the question both get run. This results in output that does not match the output expected by codeWOF, so the attempt does not pass codeWOF's tests. I suggest two methods for minimising this problem. One option is to make it clearer on codeWOF that it is not necessary to write your own tests. However, writing and testing your own code is also an important skill to practice. The second option would be to provide functionality where a user can add their own tests in addition to the tests provided by codeWOF so that the user can also practice writing tests.

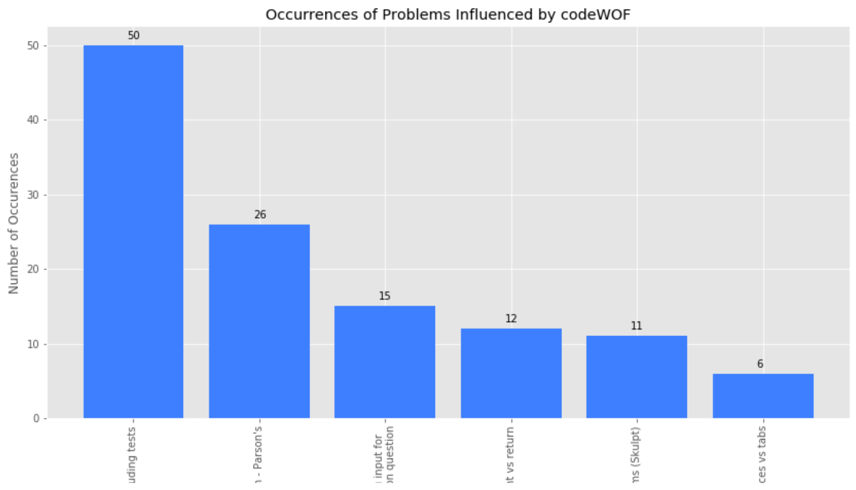
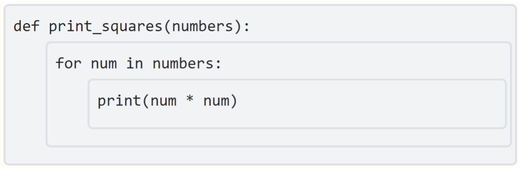


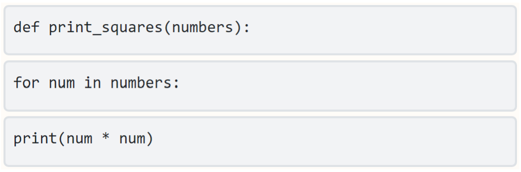
Figure 10: Occurrences of problems related to codeWOF



```
def print_squares(numbers):
    for num in numbers:
        print(num * num)
```

Figure 11: The `print_squares` function answered correctly

Indentation - Parson's Twenty-six of the attempts on Parson's questions had incorrect indentation. An example of this is the `print_squares` function answered correctly in figure 11, and incorrectly because of indentation as shown in figure 12. I have grouped these separately to other question types because the interface for Parson's problems is different to that of the other questions (figure 1). Because only one third of the participants were able to answer Parson's questions, I would expect that if all the participants were allowed to answer Parson's questions then this indentation issue would be observed more often. Teachers from the UC Computer Science Education Research Group have used this interface and have reported that nesting the blocks of code correctly (to achieve the correct indentation) is more difficult than when answering other problem types. Further work on the Parson's problem interface is needed to make it easier to use. One way to do this could be a compulsory introduction video that would be played before the



```
def print_squares(numbers):
for num in numbers:
print(num * num)
```

Figure 12: The `print_squares` function answered using incorrect indentation

first time someone answers a Parson's problem.

Using input for function question Fifteen attempts used the `input` function when answering function questions (where the `input` function did not need to be used). Most of these were used as part of test cases that the user had written, so if the test cases were no longer included by the user or if they became static as part of the test cases below the question, then it is likely that this issue would also become less frequent.

Print vs Return When codeWOF was first released we noticed that questions were frequently being failed because a `print` statement would be used instead of a `return` statement and vice versa. After this I changed the text of all questions on codeWOF so that print or return was made bold in the question text to make it more obvious. This did not have a noticeable effect on how many times this mistake was made, but this is to be expected given that the mistake only occurred twelve times within the attempts I analysed.

Rounding problems (Skulpt) Skulpt (*Skulpt - Python. Client Side.*, n.d.) is a client side implementation of Python that runs in a browser. codeWOF uses Skulpt to execute the provided tests on the code written by the user on their machine. Unfortunately there is an issue with Skulpt where numbers are rounded differently than in plain Python. This means that in some cases, code which passes the tests when run locally in Python do not pass the tests when run using Skulpt. One question on codeWOF (at the time of writing this report) experienced this problem - Print Score, where using string formatting would result in some floats being printed with decimal places and some without.

Spaces vs Tabs It was brought to our attention by one of the teachers in the study that using a mixture of spaces and tabs for indentation resulted in a confusing error message. To counteract this, whenever the tab character is pressed four spaces are entered instead, and if a tab character still makes it in (e.g. through copy and paste) we show an error message that describes that only spaces should be used for indentation and any tabs need to be replaced. This fix was released after the end of the study.

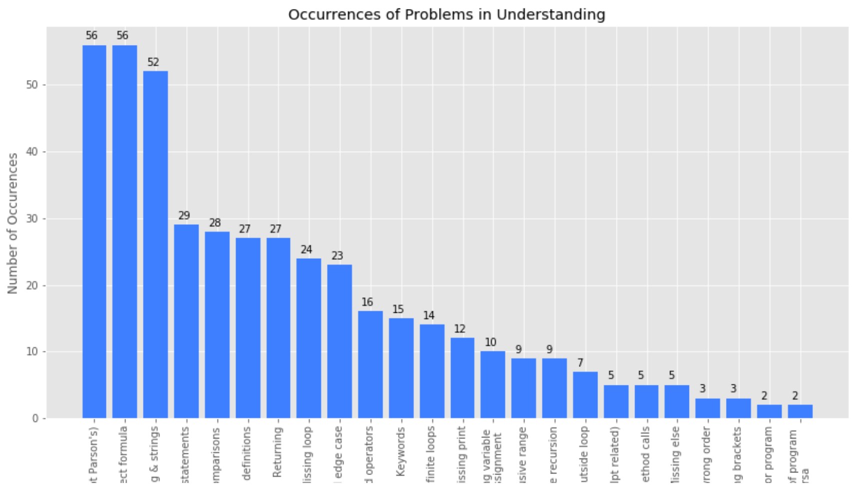


Figure 13: Occurrences of problems in question understanding or Python fluency

4.3.2 Problems Related to Question Understanding or Python Fluency

Indentation (not Parson's) Fifty-six of the attempts analysed contained incorrect indentation when answering questions that were programs, functions, or debugging questions. Many of these attempts were on function questions where none of the code was indented at all. This may be because many of the participants are still beginners in Python and have written more programs than functions when learning to program. It is common to learn how to write programs first when learning Python, and then writing functions is considered harder and is taught later. If this is the case then these participants would have more experience writing programs where less indentation is needed and may forget that the body of a function requires indentation.

Regularly practising writing code on codeWOF would help these users to remember how to indent their code correctly.

Incorrect formula Fifty-six of the attempts contained what I have described as an error relating to an incorrect formula. For example, if a question required the user to write a function that calculated a percentage grade using the arguments `points` and `total_possible_points`, a correct formula would be `percentage = (points / total_possible_points) * 100`, and an incorrect formula would be `percentage = points / total_possible_points`. These errors are expected as it is easy to misread a question, or forget a step in a calculation.

Printing, formatting & strings Fifty-two of the attempts contained errors to do with printing, strings, and string formatting. Examples of this include trying to concatenate an integer and a string, forgetting that using a comma to concatenate adds an extra space character, and forgetting characters at the end of a string - especially full stops. codeWOF could help users to spot some of these problems. One method to mitigate this error would be to add in a button that highlights the differences between the expected output and the actual output of a program or function. This would help in cases where there is an extra space or missing character that could be hard to find.

Order of statements Twenty-nine of the attempts I analysed contained errors where the lines of code on their own were correct, but the ordering of these lines were incorrect. An example of this is to write a function `print_add_10(number)` that adds ten to a number and prints it. Calling `print_add_10(5)` should print `15`. If the code submitted was:

```
def print_add_10(number):
    print(number)
    number += 10
```

Then it is the order of lines which is incorrect. The correct ordering of the lines is:

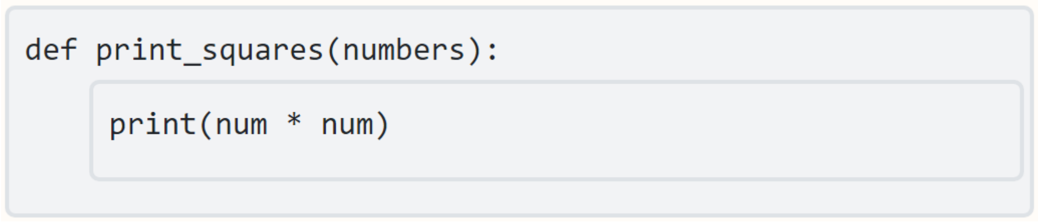
```
def print_add_10(number):
    number += 10
    print(number)
```

I saw this type of error in all of the question types, not just Parson's problems. An option to help with this type of mistake could be to include a link to an interactive debugger somewhere on codeWOF - likely with a range of other tools and tutorials. A website like Python Tutor (*Visualize Python, Java, JavaScript, C, C++, Ruby code execution*, n.d.) could be an appropriate choice as it helps to visualise the execution of the code and keep track of variables and their values.

Comparisons Twenty-eight of the attempts used a comparison which was incorrect for the situation. Common mistakes included using `>` or `<` instead of `>=` or `<=`, using `=` instead of `==` when checking equality. These mistakes are expected, and can easily be the result of misreading a question, or accidentally missing a character when typing.

Function definitions Twenty-seven of the attempts contained invalid function definitions. Problems with these include forgetting the `def` keyword, too many or too few arguments in the function definition, or forgetting the function definition entirely. I did not expect to see any mistakes of this nature, but making mistakes like these would make sense if the users did not have much practice with functions, as discussed in relation to indentation. To help users who are re-learning things like writing functions, having links to beginner Python tutorials somewhere on the website could be useful.

Returning Twenty-seven of the attempts contained errors to do with returning a value from a function. These errors were things like having no `return` statement when the question asked for value to be returned, returning from inside a loop instead of outside it, and having a `return` statement for when a condition is true, but not for when the condition is false.

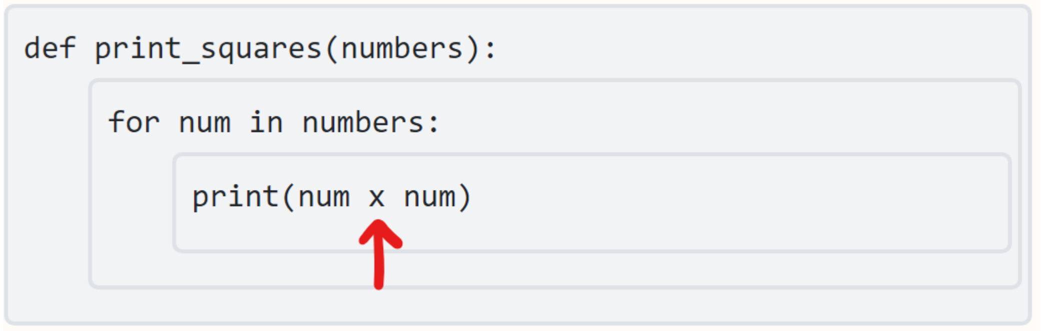


```
def print_squares(numbers):
    print(num * num)
```

Figure 14: The `print_squares` function answered without a loop

Missing loop Twenty-four of the attempts did not use a loop when one was required. For example, a Parson's problem asked the user to create a function that printed the squares of all the numbers in a list. Even though there was a for loop available in the lines of code given to the user (see figure 11), some of the attempts for this question did not use it, and tried to print the loop variable squared even though the loop variable did not have a value yet (see figure 14).

Failed edge case All twenty-three of the attempts that fell under this category were answering a question where a function returned whether a given year was a leap year or not. These attempts failed the edge case where if the year is evenly divisible by 100 then it is only a leap year if it is also divisible by 400. This question is one of the hardest questions on the website, so it would be helpful if we could make sure that it is not shown to the user too early in their use of codeWOF so that we do not discourage them. One way of doing this would be the skill areas that Maree considered in her gamification project of codeWOF (Palmer & Bell (Supervisor), 2019).



```
def print_squares(numbers):
    for num in numbers:
        print(num x num)
```

Figure 15: The `print_squares` function answered with an invalid operator

Invalid operators Sixteen of the attempts that I analysed contained invalid operators. An example of this is the Parson's problem where each number in a list needs to be squared and printed. The character `x` was used to try to multiply a number by itself (see figure 15), rather than `*` (see figure 11). The line `print(num x num)` was provided in the "Available lines" as a distractor, so this mistake is to be expected. The distractor block was provided as an example of what not to do to try and reinforce the meaning of different operators.

Keywords Fifteen of the attempts had mistakes to do with keywords/reserved words in Python. Some of these included using lowercase `true`, `false` or `none` instead of `True`, `False` and `None`. Others included using the strings `"True"` or `"False"` rather than booleans, or trying to use `pass` as a variable. To make the difference between keywords and other code clearer, the colour of keywords in the codeWOF editor could be modified to make them easier to recognise.

Infinite loops Fourteen of the attempts contained infinite loops. One of the advantages of using Skulpt to run the Python code is that these infinite loops happen in the user's browser, so cannot affect other users of codeWOF. Another advantage is that a `TimeLimitError` is raised when this happens, so the user does not have to deal with terminating the infinite loop manually.

Missing print Twelve of the attempts were missing print statements. Either no print statements at all, or missing a print statement for a particular branch of logic.

Referencing variable before assignment Ten of the attempts tried to use a variable before it had been assigned a value (see figure 14).

Inclusive vs exclusive range Nine of the attempts included errors where the user assumed that the `range` function was inclusive of the stop number, when it is actually exclusive which resulted in off-by-one errors.

Infinite recursion Nine of the attempts to answer the factorial function mistakenly used infinite recursion. The recursive case used `x * factorial(x)` instead of `x * factorial(x - 1)`. Interestingly, many of the answers to this question used recursion, even though we did not mention or require it for the question.

Operations inside/outside loop Seven of the attempts made mistakes where operations that should have been inside the loop were outside the loop and vice versa. An example of this is a question where a function has to count down from a given number, and then needs to print `Blast off!`. When `print("Blast off!")` is inside the loop it prints for every iteration, instead of only at the end of the countdown.

Problems seen 5 times or less

- Five of the attempts had mistakes with rounding floats that were not related to Skulpt's rounding issue. These include trying to use rounding functions that had not been imported, or that rounded up when it needed to always round down.
- Five attempts had mistakes when calling methods, where the brackets of the method call were not included.
- Five attempts had `if` statements where a value was returned inside it, but there was no `else` statement to return a value if the statement was `False`.
- Three attempts for a program question asked for inputs in the opposite order than specified in the question text.
- Three of the attempts analysed had missing/uneven brackets. While this is not a large issue, we could consider adding support for automatic bracket completion.
- Two attempts for program questions did not use the `input` function. This is likely because some of the program questions say that "Your program should ask for ..." rather than explicitly stating that the program should take input from the user.
- Two attempts were written as the wrong question type, one where a program was submitted for a function question, and another where a function was submitted for a program question.

Other Observations I was surprised to see that some "easy" questions took more than two attempts on average to answer. These include `doubler`; a function that doubles a number, `print_codewof`; a function that prints the string `"Welcome to codeWOF!"`, and `greet`; a function that prints `"Hello {name}!"` where `name` is the argument of the function. I observed an interesting range of formulas used to double a number, including $n * 2$, $n + n$,

$\frac{(n+n)^2}{n^2}$, $\frac{n^2+n^2}{n^2}$, $\frac{(n^2+n^2)*n}{n^2}$, $\frac{\texttt{int}(n^2+n^2)*n}{n^2}$, and $\texttt{int}(\frac{(n^2+n^2)*n}{n^2})$. This is an example of how many ways a single operation can be overcomplicated, and there were also examples of how larger functions and programs can be overcomplicated.

A potential solution for helping someone who has overcomplicated their code would be giving the user the option to see the sample solution as an alternative way to answer the same question. This would need to be after the user has answered the question correctly, otherwise it could easily be abused. The sample solutions for the questions on codeWOF have already been written, so they would only need to be made visible to the user.

5 Conclusions & Future Work

5.1 Research Question 1

Teachers do experience decay of their programming skills if they are not being used regularly. I can conclude this because existing literature supports that decay of skills occur when they are not being used, and that regular practice of skills increases short and long-term retention of that skill.

This serves as evidence that codeWOF, as a practice tool for programming, would help teachers (and other people) to maintain their programming skills when they would not otherwise be used.

Further investigation could determine what spacing between practice sessions can give enough benefit of increased skill retention, and also do not require too much time from the user.

5.2 Research Question 2

A range of usage behaviours of codeWOF were observed, from participants who did not answer any questions during the study, participants who used codeWOF for massed practice and then stopped using it, to participants who consistently used codeWOF (some of which finished all of the questions available to them).

Email reminders would be a very useful tool to ensure that users do not forget about codeWOF, and more investigation could be done on how often these emails should be sent and whether emails on particular days of the week help more than others.

The high user retention that codeWOF has, at least within in the study group, indicates that we are probably providing a tool that our users find valuable. Next steps could include discussions with our users around why they use codeWOF, and ways that they think it could be improved.

5.3 Research Question 3

Main recommendations from analysing programming mistakes made by users of codeWOF:

- Users including test cases in their code could be helped by either making it clear that writing your own tests is not necessary, or by adding functionality for users to be able to add their own tests
- Users making mistakes with indentation of Parson's problems could be helped with a compulsory introduction video before they attempt a Parson's problem for the first time and further work on making the Parson's problem editor easier to use
- To help users who have trouble printing and formatting string, a button that highlights the differences between the characters of the expected output and actual output could be useful
- Skill areas and question difficulty ratings would a valuable area of future research, to ensure that more difficult practice questions are not presented to the user too early so that they are not discouraged when they find it much more difficult.
- Reference to websites like Python Tutor for visualising code execution would be helpful, as well as a range of other tools and tutorials for Python
- Some users could find it helpful to see sample solutions after they have answered the question correctly to help them simplify their code if they have overcomplicated it.

References

Anderson, G. S., Gaetz, M., & Masse, J. (2011, February). First aid skill retention of first responders within the workplace. *Scandinavian Journal* *of Trauma, Resuscitation and Emergency Medicine*, *19*(1), 11. Retrieved 2019-04-17, from `https://doi.org/10.1186/1757-7241-19-11` doi: 10.1186/1757-7241-19-11

Arthur Jr, W., Bennett Jr, W., Stanush, P. L., & McNelly, T. L. (1998, March). Factors That Influence Skill Decay and Retention: A Quantitative Review and Analysis. *Human Performance*, *11*(1), 57–101. Retrieved 2019-05-30, from `https://doi.org/10.1207/s15327043hup1101_3` doi: 10.1207/s15327043hup1101_3

Bahrick, H. P., Bahrick, L. E., Bahrick, A. S., & Bahrick, P. E. (1993). Maintenance of Foreign Language Vocabulary and the Spacing Effect. *Psychological Science*, *4*(5), 316–321. Retrieved 2019-05-30, from `http://www.jstor.org/stable/40063054`

Baturay, M., Yıldırım, S., & Daloğlu, A. (2009). Effects of Web-Based Spaced Repetition on Vocabulary Retention of Foreign Language Learners. *Eurasian Journal of Educational Research*(34), 17–36.

Bloom, K. C., & Shuell, T. J. (1981). Effects of Massed and Distributed Practice on the Learning and Retention of Second-Language Vocabulary. *The Journal of Educational Research*, *74*(4), 245–248. Retrieved 2019-09-24, from `http://www.jstor.org/stable/27539823`

codeWOF. (n.d.). Retrieved 2019-10-25, from `https://www.codewof.co.nz/`

Comings, J. P. (1995, January). Literacy skill retention in adult students in developing countries. *International Journal of Educational Development*, *15*(1), 37–45. Retrieved 2019-05-13, from `http://www.sciencedirect.com/science/article/pii/0738059394E0006A` doi: 10.1016/0738-0593(94)E0006-A

Ellington, J. K., Surface, E. A., Blume, B. D., & Wilson, M. A. (2015, January). Foreign Language Training Transfer: Individual and Contextual Predictors of Skill Maintenance and Generalization. *Military Psychology*, *27*(1), 36–51. Retrieved 2019-05-02, from `https://doi.org/10.1037/mil0000064` doi: 10.1037/mil0000064

Kim, J. W., Ritter, F. E., & Koubek, R. J. (2013, January). An integrated theory for improved skill acquisition and retention in the three stages of learning. *Theoretical Issues in Ergonomics Science*, *14*(1), 22–37. Retrieved 2019-05-27, from `https://doi.org/10.1080/1464536X.2011.573008` doi: 10.1080/1464536X.2011.573008

Kluge, A., & Frank, B. (2014, February). Counteracting skill decay: four refresher interventions and their effect on skill and knowledge retention in a simulated process control task. *Ergonomics*, *57*(2), 175–190. Retrieved 2019-05-30, from `https://doi.org/10.1080/00140139.2013.869357` doi: 10.1080/00140139.2013.869357

Mitchell, E. L., Lee, D. Y., Sevdalis, N., Partsafas, A. W., Landry, G. J., Liem, T. K., & Moneta, G. L. (2011, January). Evaluation of distributed practice schedules on retention of a newly acquired surgical skill: a randomized trial. *The American Journal of Surgery*, *201*(1), 31–39. Retrieved 2019-05-13, from `http://www.sciencedirect.com/science/article/pii/S0002961010006057` doi: 10.1016/j.amjsurg.2010.07.040

New digital technologies for schools and kura. (n.d.). Retrieved 2019-10-25, from `http://www.beehive.govt.nz/release/new-digital-technologies-schools-and-kura`

Palmer, M., & Bell (Supervisor), T. (2019). *SENG402 Research Project - Gamification of programming skill maintenance for teachers (Unpublished)* (Unpublished doctoral dissertation). University of Canterbury, New Zealand.

Parsons, D., & Haden, P. (2006). Parson's Programming Puzzles: A Fun and Effective Learning Tool for First Programming Courses. In *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52* (pp. 157–163). Darlinghurst, Australia, Australia: Australian Computer Society, Inc. Retrieved 2019-04-04, from `http://dl.acm.org/citation.cfm?id=1151869.1151890` (event-place: Hobart, Australia)

Rose, D. J. (1994). *The effect of practice on the acquisition and maintenance of teaching skills.* (Doctoral dissertation, University of Canterbury). Retrieved 2019-05-02, from `https://ir.canterbury.ac.nz/handle/10092/813`

Sauer, J., Hockey, G. R. J., & Wastell, D. G. (2000, December). Effects of training on short- and long-term skill retention in a complex multiple-task environment. *Ergonomics*, *43*(12), 2043–2064. Retrieved 2019-05-30, from `https://doi.org/10.1080/00140130010000893` doi: 10.1080/00140130010000893

Scheeler, M. C. (2008). Generalizing Effective Teaching Skills: The Missing Link in Teacher Preparation. *Journal of Behavioral Education*, *17*(2), 145–159. Retrieved 2019-05-02, from `http://www.jstor.org/stable/41824430`

Skulpt - Python. Client Side. (n.d.). Retrieved 2019-10-24, from `-http://`

skulpt.org

Smith, C. D., & Scarf, D. (2017, June). Spacing Repetitions Over Long Timescales: A Review and a Reconsolidation Explanation. *Frontiers in Psychology*, *8*. Retrieved 2019-05-30, from https://doaj.org

Stevens, C. K., & Gist, M. E. (1997). EFFECTS OF SELF-EFFICACY and GOAL-ORIENTATION TRAINING ON NEGOTIATION SKILL MAINTENANCE: WHAT ARE THE MECHANISMS? *Personnel Psychology*, *50*(4), 955–978. Retrieved 2019-04-17, from https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1744-6570.1997.tb01490.x doi: 10.1111/j.1744-6570.1997.tb01490.x

Thompson, D., Bell, T., Andreae, P., & Robins, A. (2013). The Role of Teachers in Implementing Curriculum Changes. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (pp. 245–250). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/2445196.2445272 (event-place: Denver, Colorado, USA) doi: 10.1145/2445196.2445272

Visualize Python, Java, JavaScript, C, C++, Ruby code execution. (n.d.). Retrieved 2019-10-24, from http://www.pythontutor.com/visualize.html#mode=edit

What's A Good Retention Rate | Topics. (n.d.). Retrieved 2019-10-25, from https://mixpanel.com/topics/whats-a-good-retention-rate/

Appendices

A Email Reminder Schedule

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
Sept 1	2	$\underline{3}$	4	5	6	7
8	$\underline{9}$	10	11	12	$\underline{13}$	14
15	16	$\underline{17}$	18	19	$\underline{20}$	21
22	23	$\underline{24}$	25	26	$\underline{27}$	28
29	30	$\underline{\text{Oct 1}}$	2	3	$\underline{4}$	5
6	7	$\underline{8}$	9	10	$\underline{11}$	12
13	$\underline{14}$	15	16	$\underline{17}$	18	19

Table 1: Schedule of when email reminders were sent - underlined dates are when an email was sent.