# Python Assessment Part 2 – Prototype Artefact

## Craig Adam

## COCS50733

## Staffordshire University

## Jan 2021

# Contents

[Introduction 2](#_Toc58856256)

[Design 3](#_Toc58856257)

[Structure 3](#_Toc58856258)

[General Design Principles 3](#_Toc58856259)

[Main menu 5](#_Toc58856260)

[Configuration 5](#_Toc58856262)

[Calculator 6](#_Toc58856263)

[Schedule 7](#_Toc58856264)

[Difficult Word Checker 8](#_Toc58856265)

[Higher or Lower 9](#_Toc58856266)

[Hangman 11](#_Toc58856269)

[Quiz 12](#_Toc58856270)

[Notes 14](#_Toc58856270)

[Implementation 15](#_Toc58856271)

[Testing Plan 16](#_Toc58856272)

[Conclusion 17](#_Toc58856273)

[Bibliography 18](#_Toc58856274)

[Appendices 19](#_Toc58856274)

[Appendix A - main.py test plan 19](#_Toc58856275)

[Appendix B - Calculator (calculator.py) feature module test plan 22](#_Toc58856276)

[Appendix C - Daily schedule (schedule.py) feature module test plan 37](#_Toc58856277)

[Appendix D - Difficult word checker (checker.py) feature module test plan 45](#_Toc58856278)

[Appendix E - Higher or lower (higher\_or\_lower.py) feature module test plan 50](#_Toc58856278)

[Appendix F - Hangman (hangman.py) feature module test plan 57](#_Toc58856279)

[Appendix G - Quiz (quiz.py) feature module test plan 63](#_Toc58856280)

[Appendix H - Notes (notes.py) feature module test plan 72](#_Toc58856280)

# Introduction

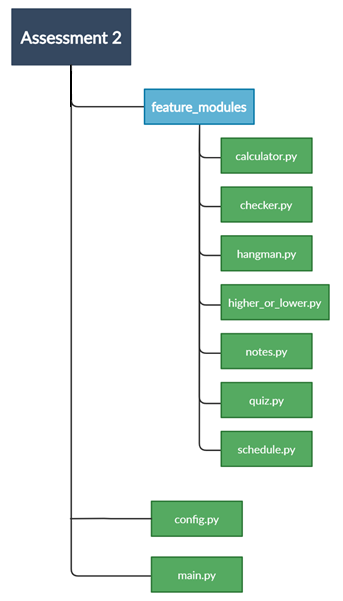
This artefact aims to detail the design, implementation and testing of a program created for the ‘Staffordshire University Early Education Department’, written in the Python programming language. The program is to be used by primary school students to assist them with everyday school tasks.

The following is a list of requirements and functionality:

* General
  + Easy to use with a basic user interface
  + Graceful error handling
  + Fully featured ‘help’ system, providing instructions about what to do next
* Menu
  + Ability to choose functionality
  + Open the chosen feature
* Calculator
  + Perform mathematical calculations
  + Store and return values (memory)
* Schedule
  + Display the days classes
  + Edit the schedule and display updated schedule
* Difficult Word Checker
  + Correct and change certain difficult words to the correct word
  + For this prototype, the word ‘structure’ is to be changed to ‘building’
* Higher or Lower Game
  + Played against the computer
  + First to guess three in a row wins
  + Ability to replay the game
* Hangman Game
  + Randomly selected five letter word
  + Eight changes to guess correctly
  + Ability to replay the game
* Two additional custom modules, justified in the design segment of this report
  + Quiz – Various multiple-choice quizzes available, scored out of five
  + Notes – Create, read, edit and delete notes

# Design

Structure

The program will follow a simple directory structure, as shown in Fig. 1.

**Fig. 1**

This structure shows each feature being placed into its own python file, inside the feature\_modules directory. By taking a modular approach the program is split up into “smaller files for easier maintenance” (The Python Tutorial – 3.9.0 Documentation, 2020). Each of these features can be written and tested individually, and future improvements would not impact the other modules.

In the route directory, the main.py module serves as the entry into the program, whilst the config.py file contains necessary setup information, along with shared functionality and variables accessed by any of the feature modules.

General Design Principles Applied to all Modules

Where possible, a simple numeric user interface should be used, allowing a user to select an activity based on a number. Removing the need to worry about spelling, or complicated inputs, will allow users to easily navigate the program. Where words are used (such as “exit” to leave a feature at any time), the words will be case-insensitive.

The need for graceful error handling is to be implemented inside all the modules. Downey (2012, p. 7) states “There is evidence that people naturally respond to computers as if they were people… when they are obstinate or rude, we respond to them the same way”. With the requirement to encourage the development of the children, the program should provide positive error messages. Incorrect inputs will use positive language, and friendly and encouraging icons will be used such as smiley faces and thumbs up symbols.

Each feature module will be built as a class, following object orientated programming (OOP) paradigms. Utilising OOP will allow individual modules to be modified without effecting the others, effectively decoupling the modules to allow for easier testing and maintenance (Hunt and Thomas, 2000, p. 140). Additionally, data encapsulation within the class will help to keep the variables and methods contained together and hidden from external sources.

Each feature module runs off a non-deterministic loop. This ensures that a user stays inside the module, only returning to the main menu once the user chooses to exit. Fig. 2 shows how this loop is used to control each feature, allowing children to enter and exit each feature in the same easy to use way.

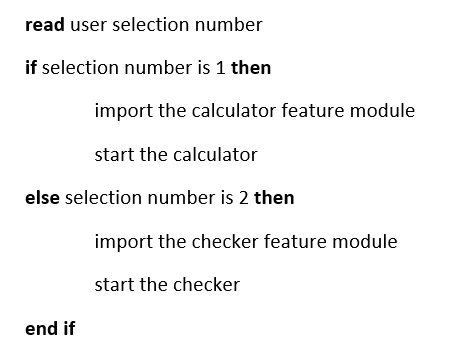
Diagram

Description automatically generated

**Fig. 2**

Main menu

A user will select the feature they require using a number to feature mapping, with the menu importing and running the selected feature module. Fig. 3 demonstrates this selection, with different numbers used to access the feature modules.



**Fig. 3**

Configuration

The config.py file contains shared functions and variables, which can be used across feature modules. Certain variables will be accessed from config.py such as the hangman.py word list and the daily schedule data for schedule.py, allowing a teacher to edit data in one location. Furthermore, this file can be used to store the calculator.py current memory value, ensuring if the calculator module is closed, the memory value persists if the user decides to go back into the calculator during the session.

Styling can be controlled through the config.py file. Individual modules can import and call functions from config.py to provide modifications to text output, allowing words to be bold, underlined or a different colour, adding visual interest to keep children engaged.

Calculator

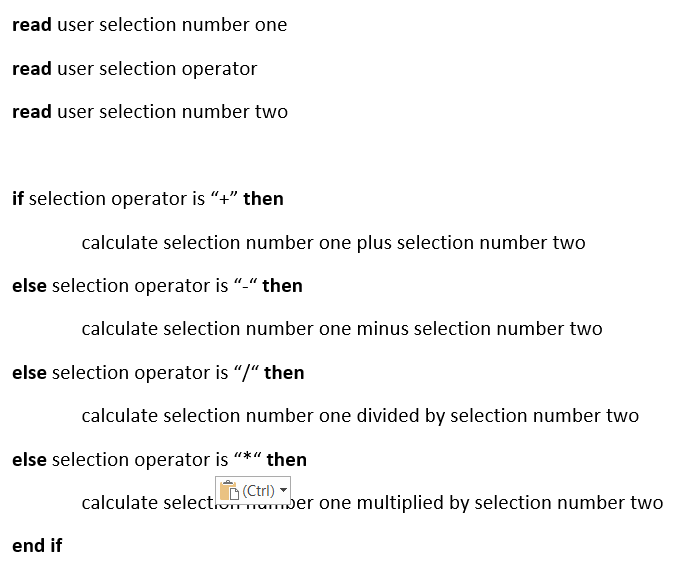
The calculator allows users to calculate addition, subtraction, multiplication and division. Additionally, the previous user input can be stored, then recalled at any time. Fig. 4 shows the logical flow of the calculator.

A picture containing text, clock

Description automatically generated

**Fig. 4**

The calculation process involves three user inputs. The first and third inputs take in the numbers the user wishes to perform maths on, with the second input taking in the operator. Once the calculation has been output, the process restarts. Fig. 5 is a pseudocode representation of the main logic following user inputs.



**Fig. 5**

Schedule

The schedule allows users to quickly see their daily lesson schedule, split into periods and corresponding subjects. Students can easily select a period to edit by typing its number, and then selecting from a list of subjects to replace the current subject, as shown by the flow diagram in Fig. 6. The schedule data is stored inside a dictionary; using the period number and the current subject for that period as key value pairs. This allows for the subjects to be easily changed, by selecting a period and mutating its corresponding value to the new subject string.

**Fig. 6**

Diagram

Description automatically generated

Difficult Word Checker

The difficult word checker allows users to enter a phrase and have certain words changed to the correct word; in the case of this prototype, ‘structure’ is always changed to ‘building’. The words the program looks for, and the words it outputs instead, are stored as dictionary key value pairs. Fig. 7 shows the logic behind the word swap; a deterministic loop checks each dictionary key against the input sentence to see if the sentence contains the key, and if so, swapping the key for its corresponding value in the sentence.

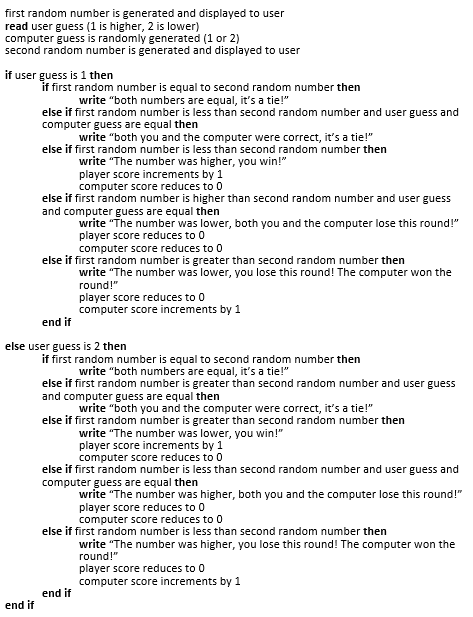
**Fig. 7**

Diagram

Description automatically generated

Higher or Lower

The higher or lower game allows a user to play against the computer to guess if the next number will be higher or lower, with the first player to win three rounds in a row being declared the winner. Fig. 8 shows pseudocode of the main logic, where nested if/else statements are used to check the user selection against the two randomly generated numbers, and against the computer’s selection.



**Fig. 8**

Fig. 9 shows the feature modules logical flow.

**Fig. 9**

Diagram

Description automatically generated

Hangman

The hangman game selects a random five letter word from the word bank in the config.py file. The user is then prompted to guess a letter, with guessed letters stored in a list. When a letter is correctly guessed the letter is then revealed in the random word, with the game ending either when the entire word has been revealed, or the user has unsuccessfully tried to guess eight times. Methods are used to check if the guessed letter is contained in the random word, to reveal correct characters in the word, and to check if the game loop should end. Fig. 10 shows the logical flow of the module.

Diagram

Description automatically generated

**Fig. 10**

Quiz

The first of the custom modules is a quiz, where user can choose which subject they want to take a quiz about. The question bank is then randomised to add replay value, and users are given five multiple choice questions, with a score out of five given after the questions have been attempted.

The quiz feature can be used by teachers in multiple ways. Quizzes can be used as revision tools, a way to help children retain important information, and a way of seeing any potential weak points in the children’s knowledge, all whilst keeping them engaged. As justification for this module Karpicke et al. (2014, pp. 198 - 199) summarise research from a wide array of academic papers documenting research on retrieval-based learning for children, stating that children ‘can learn educationally relevant facts from multiple-choice tests’.

The module will make heavy use of 2D lists. Fig. 11 demonstrates how these data structures can be used, with the chosen question index number being used to generate a bank of questions and check the users answer.

Diagram

Description automatically generated

**Fig. 11**

Fig. 12 shows the logical flow of the module.

Diagram

Description automatically generated

**Fig. 12**

Notes

The second custom module is a note taking program, where users can store helpful information. This will provide a place for children to store information such as their locker combination, friends telephone numbers or school trip dates. Note taking apps appear to have many benefits for children:

Note-taking apps are extremely beneficial for kids in school, helping them to prepare study material, review assignments and flesh out ideas .... Kids who have been diagnosed with ADHD, Dyslexia, and other types of learning disabilities, will find some of the more functional note-taking apps to be necessary learning tools (Politelli, 2013).

This notes module will have basic functionality; however, it will serve as an introduction to note taking apps, allowing children to transition to more feature rich versions as they continue through their school years. A dictionary is used to store the notes title as the key, and the content as the value, allowing users to easily create, edit or delete notes.

Diagram

Description automatically generatedFig. 13 demonstrates the logic and functionality of the module.

**Fig. 13**

# Diagram Description automatically generatedImplementation

During the implementation the overall project structure was modified allowing for persistent data storage and ease of use. Fig. 14 shows an updated structure, whereby external data required by feature modules is stored inside the ‘data’ directory.

**Fig. 14**

In order to allow data to be stored persistently between sessions, text files have now been utilised to save data. This is prevalent in the notes.py file, where notes are now written to the notes.txt file. Having notes not being persist between sessions made the prototype lack important functionality, making the module redundant.

Using text files has the added benefit of being easier to use for teachers. Teachers can easily edit the schedule.txt file to update children’s daily schedules, without the need to understand the basics of Python and programming.

# Testing Plan

Each modules test plan consists of tests to ensure all incorrect inputs are gracefully handled so that children are not put off from using the application in future. Additionally, each test plan ensures that a user can type in ‘exit’ at any time to return to the main menu, or ‘help’ to ensure the user is given correct guidance. Individual tests are then carried out to ensure correct functionality for each feature module.

For the test plans and outputs for each test see the following appendices:

* Appendix A: Menu
* Appendix B: Calculator
* Appendix C: Schedule
* Appendix D: Checker
* Appendix E: Higher or Lower
* Appendix F: Hangman
* Appendix G: Quiz
* Appendix H: Notes

# Conclusion

The prototype has achieved the requirements and functionality laid out in the introduction of this report. The overall prototype features a fully working easy to use menu system, allowing users to choose from a selection of educational resources and games. The application appears easy to use and provides helpful feedback if an input it incorrectly entered. Furthermore, children can type in ‘exit’ or ‘help’ during any of the feature modules to return to the main menu or receive tailored assistance in the same uniform manor.

Future improvements could include a graphical user interface (GUI) which could be built utilising ‘wxPython, Tkinter, and Qt’ (Downey, 2012, p. 201). This prototype has made efforts to keep children engaged using colour, text effects and icons; however, a GUI would prove useful to encourage children to use and continue using the application. Another key improvement for the finished version would be robust documentation, allowing teachers to easily update information such as the daily schedule, or add quiz questions. Whilst the prototype was primarily built with the end user in mind, it is important that teachers are comfortable and confident to input data in a simple and easy way. Individual modules could be enhanced with more functionality, such as the schedule module providing a weekly schedule, or the quiz module allowing students to choose how many questions they want in the quiz. Overall, the prototype has set a solid foundation for these changes, and functionality could be easily expanded on thanks to the well-structured OOP design of each module.

# Bibliography

Docs.python.org (2020)***The Python Tutorial – Python 3.9.0 Documentation***. [Online]Available at: <https://docs.python.org> [Accessed: 20th Nov 2020].

Downey, Allen B. (2012) ***Think Python***, Sebastopol, CA: O'Reilly Media.

Hunt, A and Thomas, D. (2000) ***The Pragmatic Programmer: From Journeyman to Master***, Reading, Mass: Addison-Wesley.

Karpicke, J. et al. (2014) Retrieval-based learning: The need for guided retrieval in elementary school children. ***The Journal of Applied Research in Memory and Cognition****,*[Online] vol. 3, no. 3, pp. 198-206. Available from: https://learninglab.psych.perdue.edu [Accessed: 11th Dec 2020].

Martin, R. (2009) ***Clean Code: A Handbook of Agile Software Craftmanship***, Upper Saddle River, NJ: Prentice Hall.

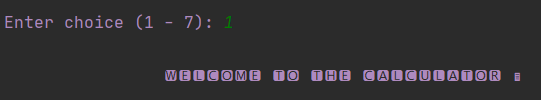
Politelli, N. (2013) ***Learning Works for Kids***. Available at: <https://learningworksforkids.com/2013/08/best-note-taking-apps-for-kids/> [Accessed: 12th Dec 2020].

## Appendices

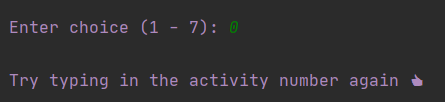
Appendix A - main.py test plan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No. | Input | Reason | Expected | Actual |
| Test 1:  Feature selection | First Selection: 1 (Calculator) | Test selecting 1 shows that the menu system can successfully open a feature module. | Option 1 selects calculator; user is taken to the calculator feature module | Welcome to the Calculator  See Fig. A1 for screenshot |
| Test 2:  Input out of scope | First Selection: 0 | Test selecting a number out of scope. Only numbers 1 – 7 should open feature modules. | Value 0 is input  An error message is printed to the user letting them know to try again | Try typing the activity number again (1-7)  See Fig. A2 for screenshot |
| Test 3:  Help | First Selection: “help” | Test selecting “help”, ensuring user is given a helpful message to guide them. | Value “help” is input  A useful message is shown guiding the user how to use the current feature (menu system) | You are currently in the application menu. To enter an activity, type its number (1-7)  See Fig. A3 for screenshot |
| Test 4:  Exit | First Selection: “exit” | Test selecting “exit” showing that the application gracefully shuts down. | Value “exit” is input  A goodbye message is shown, then the program stops | Have a great day (smiley face)  See Fig. A4 for screenshot |
| Test 5:  Help case sensitive | First Selection: “HELP” | Test to see if case sensitivity has been considered. User Interface should be easy to use and ignore case sensitivity. | Value “HELP” is input  A useful message is shown guiding the user how to use the current feature (menu system) | You are currently in the application menu. To enter an activity, type its number (1-7)  See Fig. A5 for screenshot |
| Test 6:  Exit case sensitive | First Selection:“EXIT” | Additional test to check that user inputs are not case sensitive, as user interface should be easy to use for children. | Value “EXIT” is input  A goodbye message is shown, then the program stops | Have a great day (smiley face)  See Fig. A6 for screenshot |

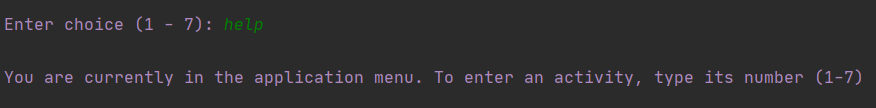
**Fig. A1 –** Successful feature selection from menu



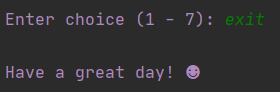
**Fig. A2 –** User selects a feature out of scope



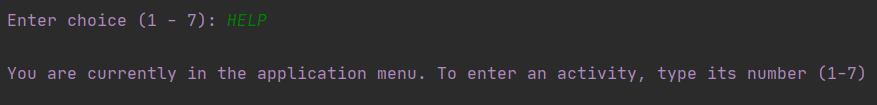
**Fig. A3** – User requests help



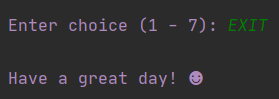
**Fig. A4 –** User exits program



**Fig. A5 –** User request help (uppercase)



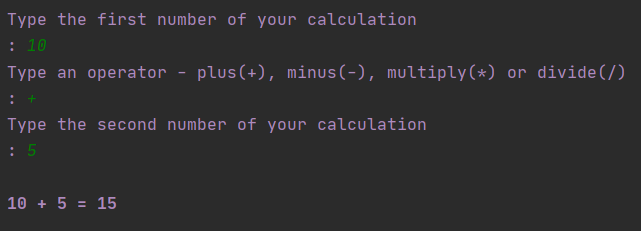
**Fig. A6 –** User exits program (uppercase)



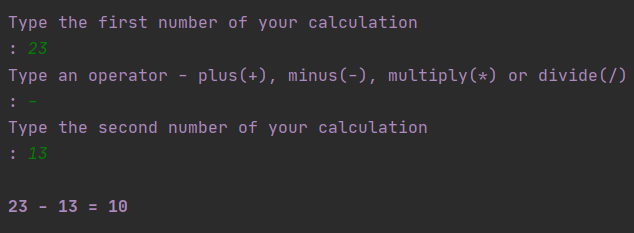
Appendix B - Calculator (calculator.py) feature module test plan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No. | Input | Reason | Expected | Actual |
| Test 1:  Addition | First Selection: 10  Second Selection: “+”  Third Selection: 5 | Test ensures simple addition is working. Test selects full integers and the plus operator. | 10 + 5 = 15 | 10 + 5 = 15  See Fig. B1 for screenshot |
| Test 2:  Subtraction | First Selection: 23  Second Selection: “-”  Third Selection: 13 | Test ensures simple subtraction is working. Test selects full integers and the minus operator. | 23 – 13 = 10 | 23 – 13 = 10  See Fig. B2 for screenshot |
| Test 3:  Multiplication | First Selection: 23  Second Selection: “\*”  Third Selection: 10 | Test ensures simple multiplication is working. Test selects full integers and the multiplication operator. | 23 \* 10 = 230 | 23 \* 10 = 230  See Fig. B3 for screenshot |
| Test 4:  Division | First Selection: 12  Second Selection: “/”  Third Selection: 6 | Test ensures simple division is working. Test selects full integers and the division operator. | 12 / 6 = 2 | 12 / 6 = 2  See Fig. B4 for screenshot |
| Test 5:  Floats | First Selection: 10.234  Second Selection: “+”  Third Selection: 5.34567 | Test selecting numbers which are not full integers. Test ensures calculations with floats accurate. | 10.234 + 5.34567 =  15.57967 | 10.234 + 5.34567 = 15.57967  See Fig. B5 for screenshot |
| Test 6:  Exit | First Selection: “exit” | Test selecting “exit” showing that the application quits back to the menu module. | Value “exit” is input  A goodbye message is shown, then the program returns to the main menu options | Goodbye from the calculator (smiley face)  See Fig. B6 for screenshot |
| Test 7:  Exit second selection | First Selection:10  Second Selection: “exit” | Test selecting “exit” on the second input, showing that the application quits back to the menu module. | Value 10 is input Value “exit” is input  A goodbye message is shown, then the program returns to the main menu options | Goodbye from the calculator (smiley face)  See Fig. B7 for screenshot |
| Test 8:  Exit third selection | First Selection:10  Second Selection: “+”  Third Selection: “exit” | Test selecting “exit” on the third input, showing that the application quits back to the menu module. | Value 10 is input, Value “+” is input  Value “exit” is input  A goodbye message is shown, then the program returns to the main menu options | Goodbye from the calculator (smiley face)  See Fig. B8 for screenshot |
| Test 9:  Exit case sensitive | First Selection:“EXIT” | Test selecting “EXIT” showing that the application quits back to the menu module, regardless of text case sensitivity. | Value “EXIT” is input  A goodbye message is shown, then the program returns to the main menu options | Goodbye from the calculator (smiley face)  See Fig. B9 for screenshot |
| Test 10:  Help | First Selection:“help” | Test selecting “help” showing that the application provides guidance to the user based on current module location. | Value “help” is input;  A relevant message is shown, then the program returns to the first calculation number input | You are in the calculator! Type in the first number of your calculation  See Fig. B10 for screenshot |
| Test 11:  Help second selection | First Selection:47  Second Selection: “help” | Test selecting “help” on the second input, showing that the application provides guidance to the user based on current module location, in this case the operator selection. | Value 47 is input  Value “help” is input  A relevant message is shown, then the program returns to the select operator input | You are in the calculator! Type an operator (plus(+), minus(-), multiply(\*), divide(/))  See Fig. B11 for screenshot |
| Test 12:  Help third selection | First Selection:56  Second Selection: “/”  Third Selection: “help” | Test selecting “help” on the third input, showing that the application provides guidance to the user based on current module location, in this case second number input. | Value 56 is input, Value “/” is input Value “help” is input  A relevant message is shown, then the program returns to the second number selection | You are in the calculator! Type the second number of your calculation  See Fig. B12 for screenshot |
| Test 13:  Help case sensitive | First Selection:32  Second Selection:  “HELP” | Test selecting “HELP” showing that the application provides user a useful message, regardless of text case sensitivity. | Value 32 is input Value “HELP” is input  A relevant message is shown, then the program returns to the operator selection | You are in the calculator! Type an operator (plus(+), minus(-), multiply(\*), divide(/))  See Fig. B13 for screenshot |
| Test 14:  Memory first input number and memory recall | First Selection: 999  Second Selection: “-“  Third Selection: “memory” | Test selecting “memory” showing that information is given to the user about the memory functionality provided. | Value 999 is input, Value “-“ is input Value “memory”  A list of memory function instructions is printed out to the user | Type ‘m+’ to add the last input number to the memory  Type ‘m-‘ to minus the last input number from the memory  Type ‘mr’ to recall the memory number  Type ‘mc’ to clear the memory number  See Fig. B14 for screenshot |
| Test 15:  Memory second input number and case sensitive | First Selection: 12  Second Selection: “MEMORY” | Test selecting “memory” as the operator input, showing that memory keyword is detect regardless of case sensitivity. Additionally, test proves keyword can be types at different points in the calculator to print information to user. | Value 12 is input  Value “MEMORY” is input  A list of memory function instructions is printed out to the user | Type ‘m+’ to add the last input number to the memory  Type ‘m-‘ to minus the last input number from the memory  Type ‘mr’ to recall the memory number  Type ‘mc’ to clear the memory number  See Fig. B15 for screenshot |
| Test 16:  Memory no input number | First Selection: “m+”  Second Selection:“mr” | Test selecting “m+” as the first input, before showing memory number with “mr” input. Proves memory features do not add a number until a number input has been given. | Value “m+” is input  Value “mr” is input  Memory value is displayed as 0 | Current memory value is: 0  See Fig. B16 for screenshot |
| Test 17:  Memory plus user number input one | First Selection: 5  Second Selection: “m+”  Third Selection: “mr” | Test selecting “m+” as the second input after a valid first number input. Proves memory features work on the previous valid user input number. Additionally, shows memory recall gives correct number. | Value 5 is input  Value “m+” is input  Value “mr” is input  Memory value is displayed as 5 | Current memory value is: 5  See Fig. B17 for screenshot |
| Test 18:  Memory minus user number input two | First Selection: 12  Second Selection: “\*”  Third Selection: 10  Fourth Selection: “m-”  Fifth Selection: “mr” | Test selecting “m-“ after a second valid number input has been given. Additionally, shows memory recall provides the correct expected answer. | Value 12 is input  Value “\*” is input  Value 10 is input  Answer calculates to 120  Value “m-“ is input  Value “mr” is input  Memory value is displayed as -10 (memory starting figure of 0, minus 10 = -10) | Current memory value is: -10  See Fig. B18 for screenshot |
| Test 19:  Memory maths | First Selection: 55  Second Selection: “m+”  Third Selection: “mr”  Fourth Selection: “m+”  Fifth Selection: “mr”  Sixth Selection: “mc” | Test memory add functionality by adding the first input (55) together using the memory functions. Proves that memory functions can add, display and be cleared. | Value 55 is input  Value “m+” is input  Value “mr” is input  Memory Value displayed as 55  Value “m+” is input  Value “mr” is input  Memory value displayed as 110  Value “mc” is input  Value “mr” is input  Memory value displayed as 0 | Current memory value is: 0  See Fig. B19 for screenshot |
| Test 20:  Memory maths mixed | First Selection: 12  Second Selection: “m-”  Third Selection: “mr”  Fourth Selection: “\*”  Fifth Selection: “24”  Sixth Selection: “m+”  Seventh Selection: “mr” | Test memory functions can add and minus from the memory number, using different input numbers. | Value 12 is input  Value “m-“ is input  Value “mr” is input  Memory value is displayed as -12  Value “\*” is input  Value “24” is input  12 \* 24 = 288 printed  Value “m+” is input  Value “mr” is input  Memory value is displayed as 12 | Current memory value is: 12  See Fig. B20 for screenshot |
| Test 21:  Persistent session memory value | First Selection: 10  Second Selection: “m+”  Third Selection: “exit”  Forth Selection: 1  Fifth Selection: “mr” | Test adding a memory value remains persistent over the applications session. The memory value should remain when the calculator is loaded back up by the user, as long as the entire application is not stopped. | Value 10 is input  Value “m+” is input  Value “exit” is input’  Value 1 is input (re-enter calculator feature module)  Value “mr” is input  Memory value is displayed as 10 | Current memory value is: 10  See Fig. B21 for screenshot |
| Test 22:  Invalid input | First Selection: “abcdefg” | Test selecting a string value instead of an expected number input. Proves unexpected values are handled gracefully. | Value “abcdefg” is input  A useful helper message is displayed to the user requesting they try again | Please try typing an input again, type ‘help’ for tailored assistance (thumbs up)  See Fig. B22 for screenshot |
| Test 23:  Dividing by Zero | First Selection: 1  Second Selection: “/”  Third Selection: 0 | Test selecting zero as the second user input, showing that division by zero is handled gracefully | Value 1 is input  Value “/” is input  Value 0 is input  A useful message is displayed telling the user they are unable to divide by zero | “Cannot divide by zero, try another sum”  See Fig. B23 for screenshot |

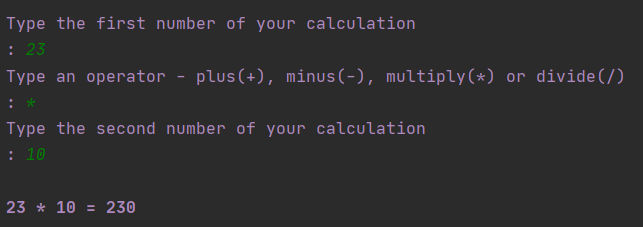
**Fig. B1 –** Simple addition



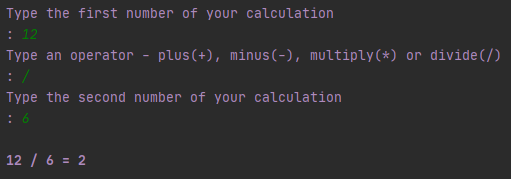
**Fig. B2 –** Simple subtraction



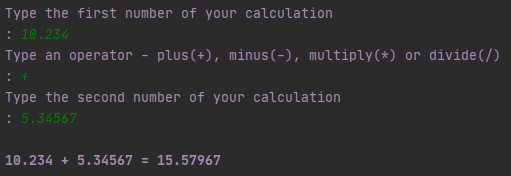
**Fig. B3** – Simple multiplication



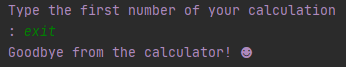
**Fig. B4 –** Simple division



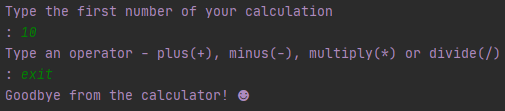
**Fig. B5 –** Sum involving complex float numbers



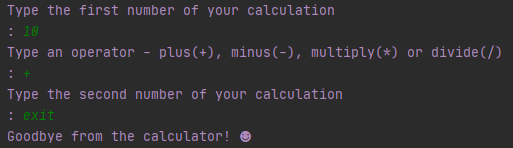
**Fig. B6 –** Exit from first input

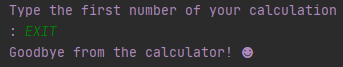


**Fig. B7 –** Exit from second (operator) input

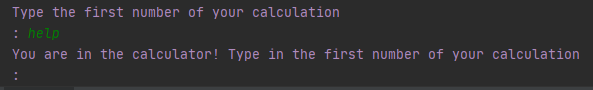


**Fig. B8 –** Exit from third (second number) input

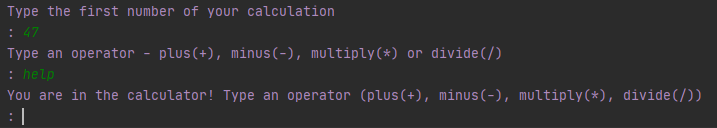


**Fig. B9 –** User exits program (uppercase)

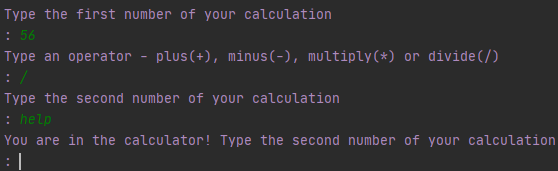
**Fig. B10 –** Help from first input



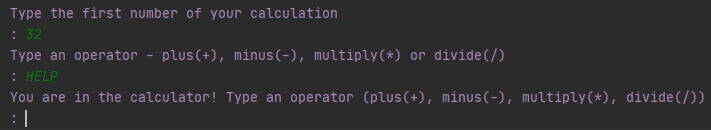
**Fig. B11 –** Help from second (operator) input



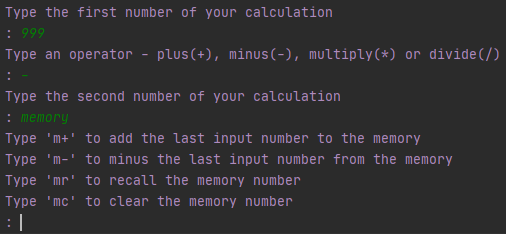
**Fig. B12 –** Help from third (second number) input



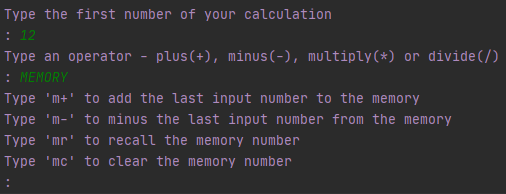
**Fig. B13 –** Help requested uppercase



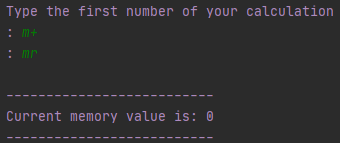
**Fig. B14 –** Memory helper



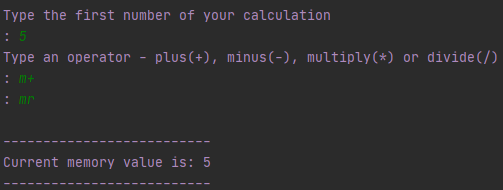
**Fig. B15 –** Memory helper uppercase



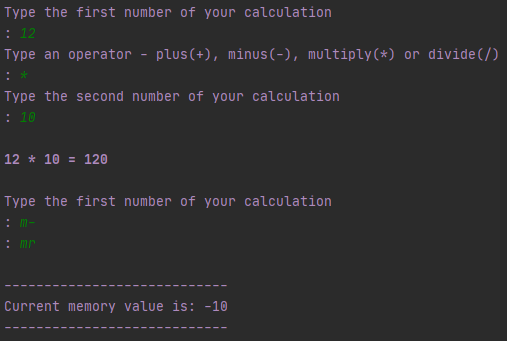
**Fig. B16 –** Memory functions first input



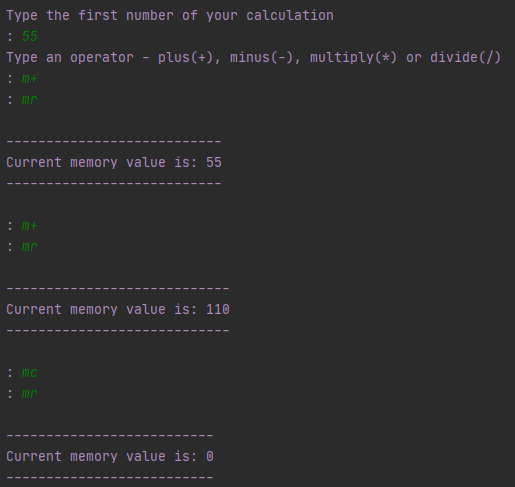
**Fig. B17 –** Memory function add and recall as first input



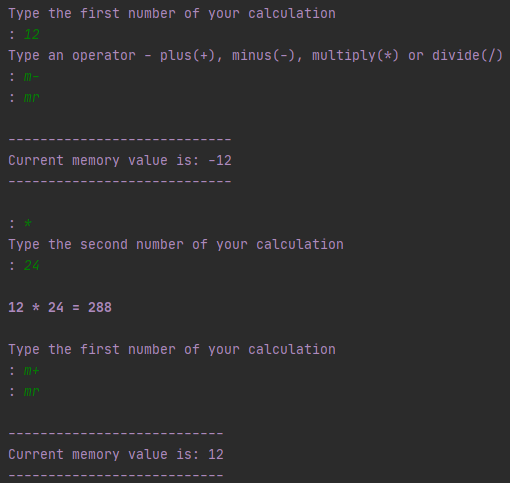
**Fig. B18 –** Memory function minus and recall as second input



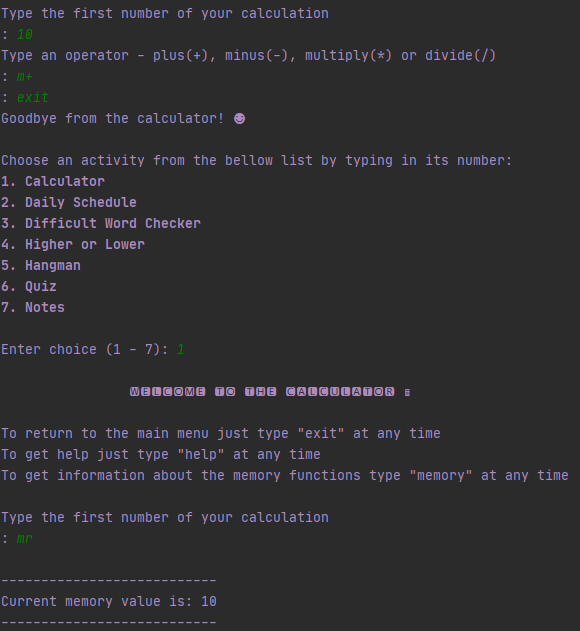
**Fig. B19 –** Memory addition and clearing



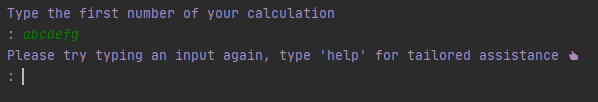
**Fig. B20 –** Memory subtracting and adding



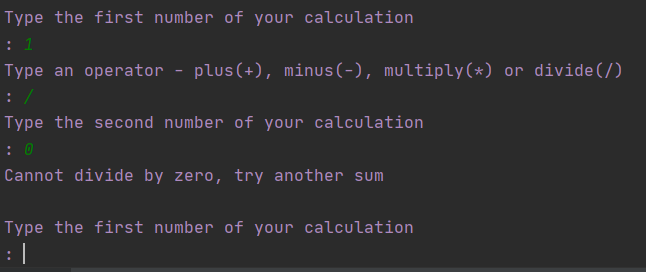
**Fig. B21 –** Persistent session memory value



**Fig. B22 –** Unexpected string value



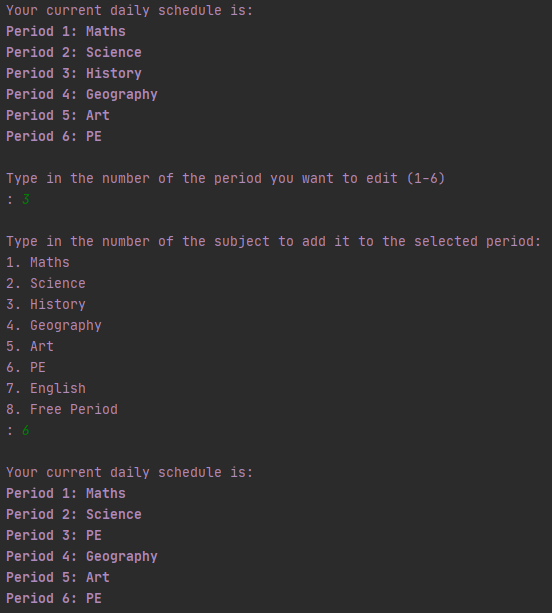
**Fig. B23 –** Zero division exception



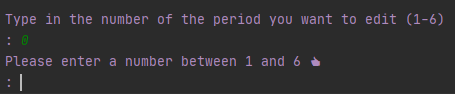
Appendix C - Daily schedule (schedule.py) feature module test plan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No. | Input | Reason | Expected | Actual |
| Test 1:  Schedule change | First Selection: 3 (Period 3)  Second Selection: 6 (PE) | Test selecting 3 shows that the module can successfully access a chosen period from the list, additionally selecting 6 next shows that the module can correctly select the correct subject as a replacement. | Value 3 is input  Value 6 is input  Daily schedule is output with period 3 showing subject 6 (PE) | Period 6: PE  See Fig. C1 for screenshot |
| Test 2:  Invalid input integer | First Selection: 0 | Test selecting a number out of scope. Only numbers 1 – 6 should select a period. | Value 0 is input  A useful message is printed to the user letting them know to try again using a valid number (1-6) | Please enter a number between 1 and 6 (thumbs up)  See Fig. C2 for screenshot |
| Test 3:  Invalid input string | First Selection: “random string” | Test selecting a string value instead of a number. String values are not accepted and should be handled gracefully. | Value “random string” is input  A useful message is printed to the user letting them know to try again using a number | Please try typing in a number again (thumbs up)  See Fig. C3 for screenshot |
| Test 4:  Help | First Selection: “help” | Test selecting “help”, ensuring user is given a helpful message to guide them. | Value “help” is input  A useful message is shown guiding the user how to use the current feature - enter a period number to edit | You are currently in the daily schedule! Type in the number of the period you would like to edit (1-6)  See Fig. C4 for screenshot |
| Test 5:  Exit | First Selection: “exit” | Test selecting “exit” showing that the module gracefully stops, and user returns to main menu selection screen. | Value “exit” is input  A goodbye message is shown, then the module returns to the main menu | Goodbye from the daily schedule! (smiley face)  See Fig. C5 for screenshot |
| Test 6:  Help case sensitive | First Selection: “HELP” | Test to see if case sensitivity has been considered. User Interface should be easy to use and ignore case sensitivity. | Value “HELP” is input  A useful message is shown guiding the user how to use the current feature - enter a period number to edit | You are currently in the daily schedule! Type in the number of the period you would like to edit (1-6)  See Fig. C6 for screenshot |
| Test 7:  Exit case sensitive | First Selection:“EXIT” | Test selecting “exit” showing that the module gracefully stops, and user returns to main menu selection screen. | Value “EXIT” is input  A goodbye message is shown, then the module returns to the main menu | Goodbye from the daily schedule! (smiley face)  See Fig. C7 for screenshot |
| Test 8:  Help second selection | First Selection: 2(Period 2)  Second Selection: “help | Test selecting “help” on the second input, showing that the application provides guidance to the user based on current module location, in this case the new subject selection menu. | Value 2 is input  Value “help” is input  A useful message is printed to the user stating that they should choose a subject number to go into the selected period | You are in the daily schedule! Type in the number of the class you would like to change your current period selection to (1-8)  See Fig. C8 for screenshot |
| Test 9:  Exit second selection | First Selection: 4 (Period 4)  Second Selection: “exit” | Test selecting “exit” as the second input, showing that the module gracefully stops, and user returns to main menu selection screen. | Value 4 is input  Value “exit” is input  A goodbye message is shown, then the module returns to the main menu | Goodbye from the daily schedule! (smiley face)  See Fig. C9 for screenshot |
| Test 10:  Persistent data | Fist Selection: 1 (Period 1)  Second Selection: 8 (Free Period)  Third Selection: “exit”  Fourth Selection: 2(daily schedule) | Test editing a period to show data persistent over the applications session and in the schedule.txt file for long term persistence when the entire application is closed. The modified schedule remains when the daily schedule module is loaded back up by the user, and the updated entry is in the .txt file | Value 1 is input  Value 8 is input  Period 6 becomes “Free Period” instead of “Maths”  Value “exit is input  Module closes and returns to main menu  Value 2 is input  Daily Schedule module is reloaded, showing Period 1 as “Free Period” | Period 6: Free Period  See Fig. C10 for screenshot of the output, and Fig. C11 for screenshot of the txt file before and after |

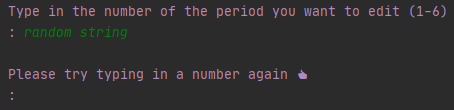
**Fig. C1 –** Swap period 3 from history to PE



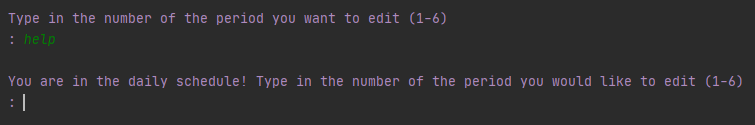
**Fig. C2 –** Number not in range



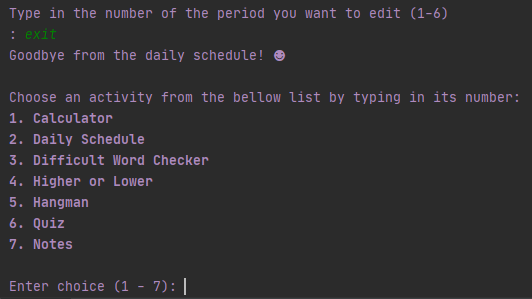
**Fig. C3 –** String input



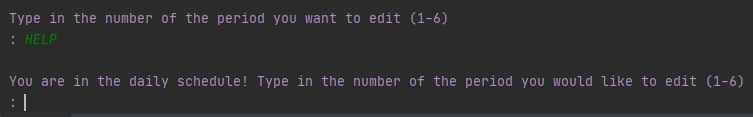
**Fig. C4 –** Help from first input



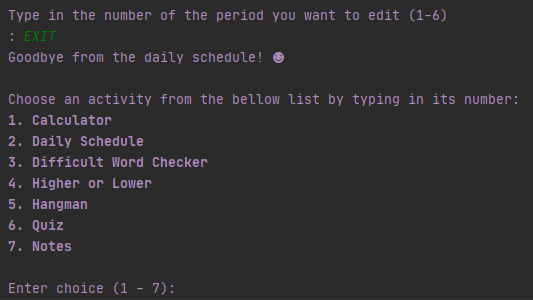
**Fig. C5 –** Exit



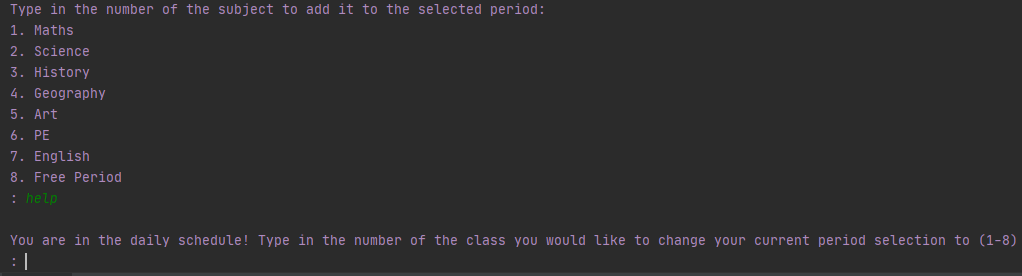
**Fig. C6 –** Help uppercase



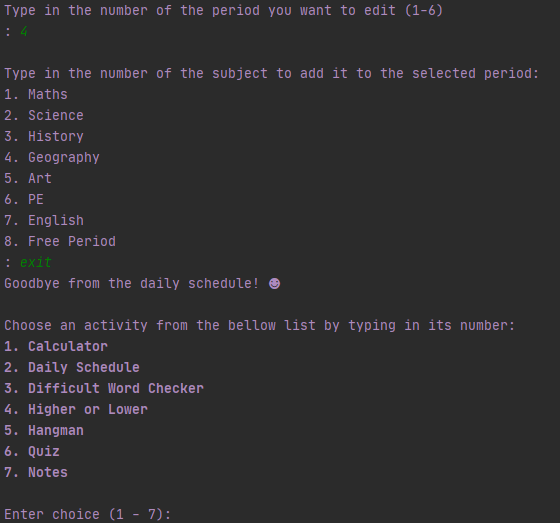
**Fig. C7 –** Exit uppercase



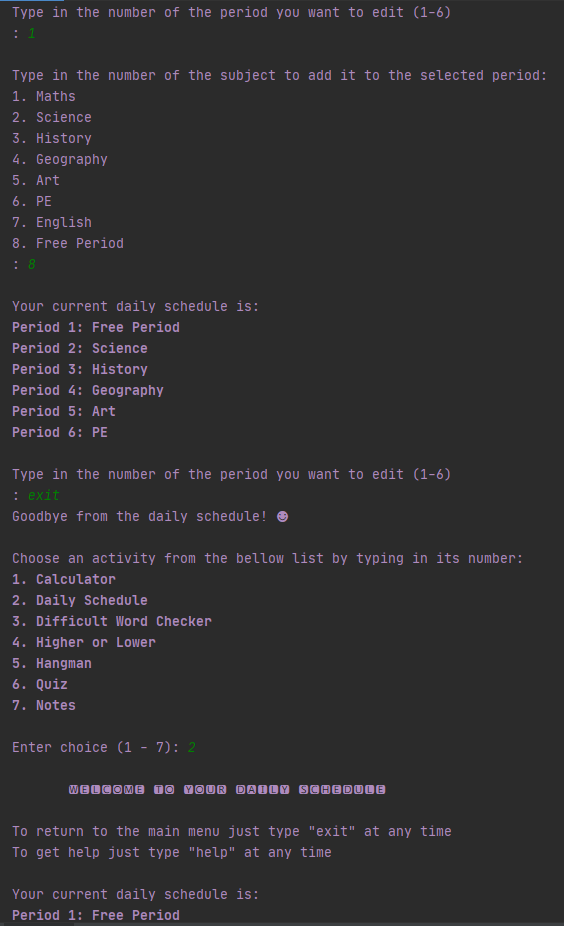
**Fig. C8 –** Help requested from second input



**Fig. C9 –** Exit from second input

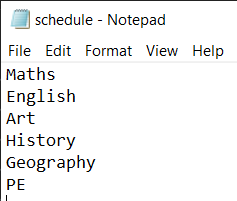
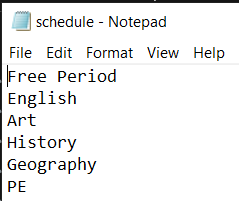


**Fig. C10 –** Persistent session schedule



**Fig C11 –** Persistent data

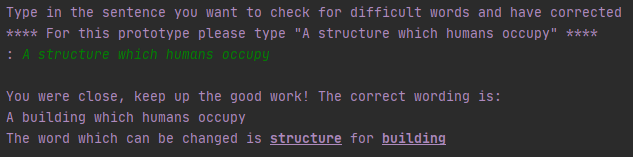
**Before Edit After Edit**



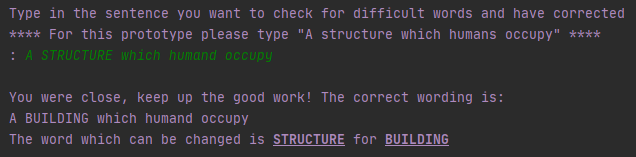
Appendix D - Difficult word checker (checker.py) feature module test plan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No. | Input | Reason | Expected | Actual |
| Test 1:  Word found and corrected | First Selection: “A structure which humans occupy” | Test selecting “A structure which humans occupy” shows that the prototype can identify the word ‘structure’ and swap it for the word ‘building’. | Value “A structure which humans occupy” is input  An encouraging message is printed to the user showing the corrected sentence and the words which were changed | You were close, keep up the good work! The correct wording is: A building which humans occupy  The word which can be changed is structure for building  See Fig. D1 for screenshot |
| Test 2:  Word found and corrected case sensitivity | First Selection: “A STRUCTURE which humans occupy | Test that program is easy to use and not case sensitive. Children may make mistakes when typing or may get stuck in caps lock. | Value “A STRUCTURE which humans occupy” is input  An encouraging message is printed to the user showing the corrected sentence and the words which were changed | You were close, keep up the good work! The correct wording is: A BUILDING which humans occupy  The word which can be changed is STRUCTURE for BUILDING  See Fig. D2 for screenshot |
| Test 3:  Word not in the dictionary | First Selection: 0 | Test selecting a value out of the prototypes scope, proving that only certain words can be found and corrected. | Value 0 is input  A useful message is printed to the user letting them know that no changes have been found | Unable to find any words to swap, well done!  See Fig. D3 for screenshot |
| Test 4:  Multiple word occurrences | First Selection: “Structure, STRUCTURE, structure” | Test selecting the word ‘structure’ multiple ways to ensure each time the word is swapped to building. Ensures the module can swap out the word in any sentence not just the given test sentence. | Value “Structure, STRUCTURE, structure” is input  An encouraging message is printed to the user showing the corrected sentence and the words which were changed | You were close, keep up the good work! The correct wording is: Building, BUILDING, building  The word which can be changed is Structure for Building  See Fig. D4 for screenshot |
| Test 5:  Help | First Selection: “help” | Test selecting “help”, ensuring user is given a helpful message to guide them. | Value “help” is input  A useful message is shown guiding the user how to use the current feature - enter a sentence to have it checked | You are currently in the difficult word checker! Type in the sentence you would like to have checked  See Fig. D5 for screenshot |
| Test 6:  Exit | First Selection: “exit” | Test selecting “exit” showing that the module gracefully stops, and user returns to main menu selection screen. | Value “exit” is input  A goodbye message is shown, then the module returns to the main menu | Goodbye from the difficult word checker! (smiley face)  See Fig. D6 for screenshot |
| Test 7:  Help case sensitivity | First Selection: “HELP” | Test to see if case sensitivity has been considered. User Interface should be easy to use and ignore case sensitivity. | Value “HELP” is input  A useful message is shown guiding the user how to use the current feature - enter a sentence to have it checked | You are currently in the difficult word checker! Type in the sentence you would like to have checked  See Fig. D7 for screenshot |
| Test 8:  Exit case sensitivity | First Selection:“EXIT” | Test selecting “exit” showing that the module gracefully stops, and user returns to main menu selection screen. | Value “EXIT” is input  A goodbye message is shown, then the module returns to the main menu | Goodbye from the difficult word checker! (smiley face)  See Fig. D8 for screenshot |

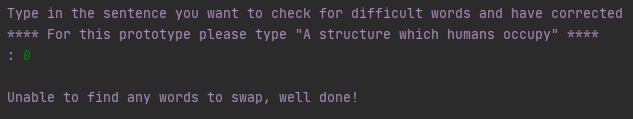
**Fig. D1 –** Correct word swap



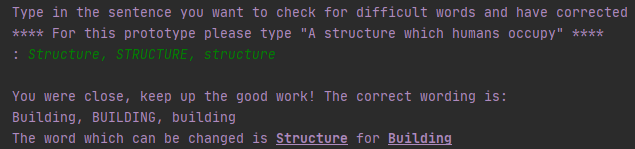
**Fig. D2 –** Case sensitivity



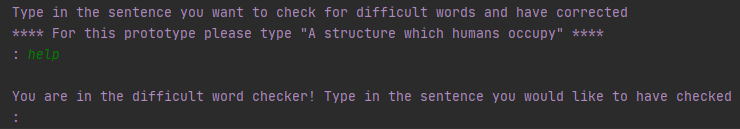
**Fig. D3 –** Correct string input



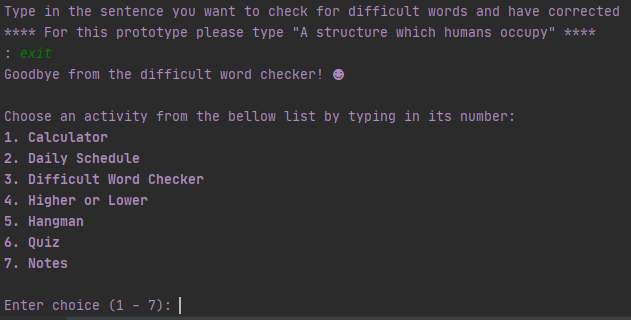
**Fig. D4 –** Word detected in any sentence



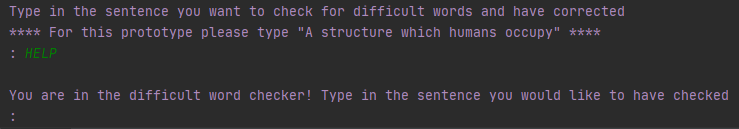
**Fig. D5 –** Help



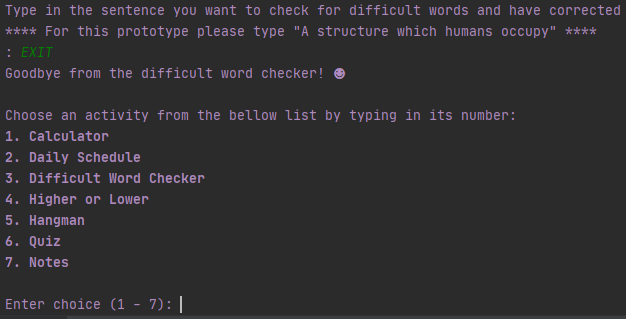
**Fig. D6 –** Exit



**Fig. D7 –** Help uppercase



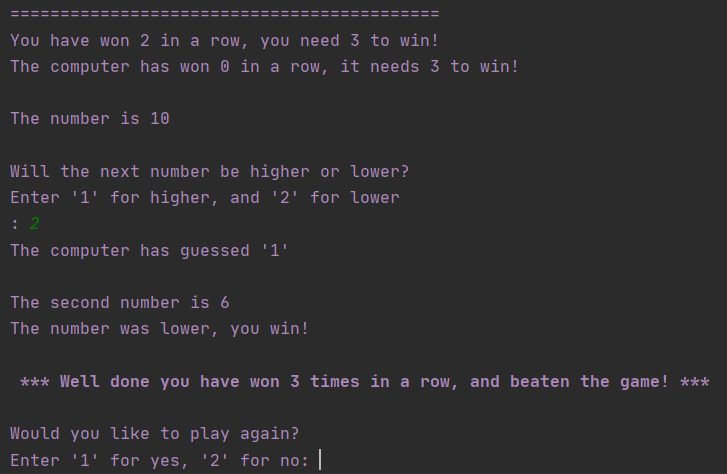
**Fig. D8 –** Exit uppercase



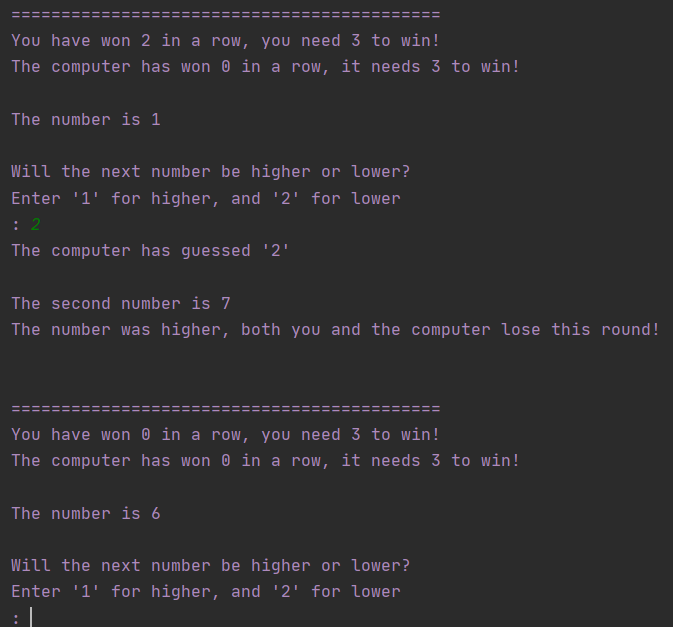
Appendix E - Higher or lower (higher\_or\_lower.py) feature module test plan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No. | Input | Reason | Expected | Actual |
| Test 1:  Correct answers | First Selection: Correct  Second Selection:Correct  Third Selection:Correct | Test selecting three correct answers in a row, proving that player counter correctly increments, and the player wins when it reaches 3. | Correct value is guessed three times in a row  Each time the win counter increases until the player has won three games in a row  Player is given a congratulations message and asked if they want to play again | \*\*\* Well done you have won 3 times in a row, and beaten the game! \*\*\*  Would you like to play again?  See Fig. E1 for screenshot |
| Test 2:  Mixed answers | First Selection: 1 – Correct  Second Selection: 1 - Incorrect | Test selecting a correct guess, followed by an incorrect guess. Test shows that the player score correctly drops back to 0 on an incorrect guess, and the computer score correctly increments. | Correct value is guessed, and player score increments to 1  Incorrect guess is input, and player score returns to 0 whilst computer score goes up 1 | The number was higher, both you and the computer lose this round!  See Fig. E2 for screenshot |
| Test 3:  Incorrect answers | First Selection: False  Second Selection:False  Third Selection: False | Test selecting three incorrect answers in a row, proving that computer counter correctly increments, and the computer wins when it reaches 3. | Incorrect value is guessed three times in a row  Each time the computer wins counter increases until the computer has won three games in a row  Player is given a losing message and asked if they want to play again | \*\*\* Unlucky, the computer has won 3 times in a row! \*\*\*\*  Would you like to play again?  See Fig. E3 for screenshot |
| Test 4:  Equal numbers (tie) | First Selection: 2 (lower) | Test to ensure if the first number matches the second number, that round is a draw and the score counters remain the same. Additionally, test shows that the computers score doesn’t change either when a draw occurs. | Program gives number 7  Value 2 (lower) is input by player  Value 2 is guessed by computer  Program second random number is also a 2  Program let user know the numbers are equal, so the round is a tie. Neither player nor computer score changes | Both numbers where equal, that’s a tie!  See Fig. E4 for screenshot |
| Test 5:  Tie | First Selection: 1 (higher) | Test to ensure if the player input and the computer input are the same, neither score is incremented as the round is a tie. | Program gives number 3  Value 1 (higher) is input  Computer generated input is 1 (higher)  Program second random number is generated as 8  Program let user know the numbers are equal, so the round is a tie. | Both you and the computer were correct, that’s a tie!  See Fig. E5 for screenshot |
| Test 6:  Help | First Selection: “help” | Test selecting “help”, ensuring user is given a helpful message to guide them. | Value “help” is input  A useful message is shown guiding the user how to use the current feature – guess higher or lower | You are currently in the higher or lower game! Type in ‘1’ to guess higher or ‘2’ to guess lower  See Fig. E6 for screenshot |
|  | First Selection: “exit” | Test selecting “exit” showing that the module gracefully stops, and user returns to main menu selection screen. | Value “exit” is input  A goodbye message is shown, then the module returns to the main menu | Goodbye from the higher or lower game! (smiley face)  See Fig. E7 for screenshot |
| Test 7:  Help case sensitivity | First Selection: “HELP” | Test to see if case sensitivity has been considered. User Interface should be easy to use and ignore case sensitivity. | Value “HELP” is input  A useful message is shown guiding the user how to use the current feature – guess higher or lower | You are currently in the higher or lower game! Type in ‘1’ to guess higher or ‘2’ to guess lower  See Fig. E8 for screenshot |
| Test 8:  Exit case sensitivity | First Selection:“EXIT” | Test selecting “exit” showing that the module gracefully stops, and user returns to main menu selection screen. | Value “EXIT” is input  A goodbye message is shown, then the module returns to the main menu | Goodbye from the higher or lower game! (smiley face)  See Fig. E9 for screenshot |

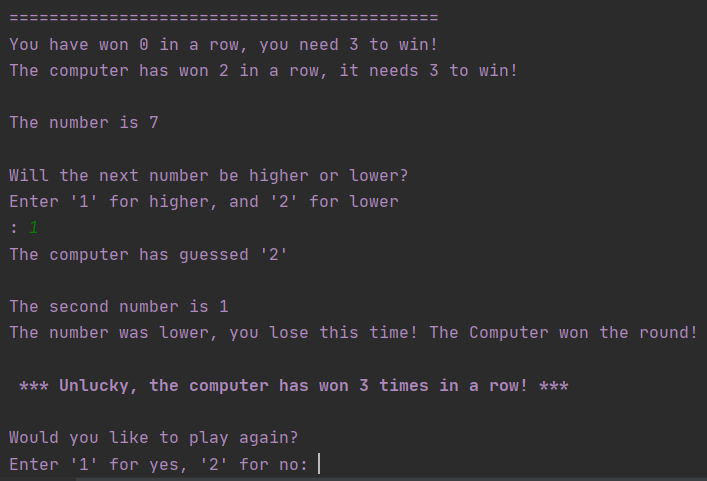
**Fig. E1 –** Win



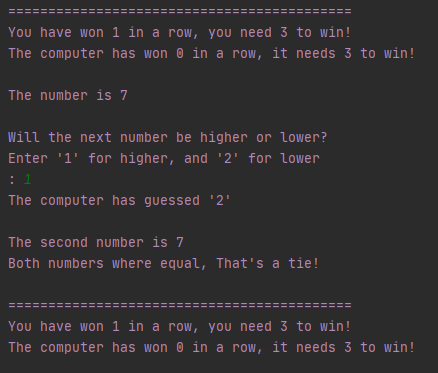
**Fig. E2 –** Mixed



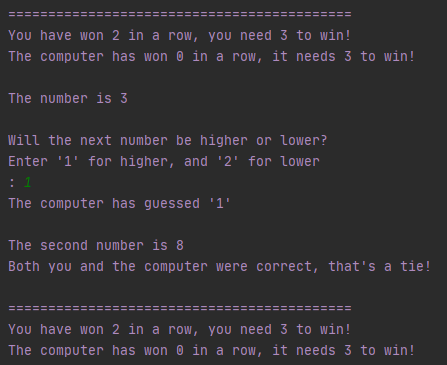
**Fig. E3 –** Computer win



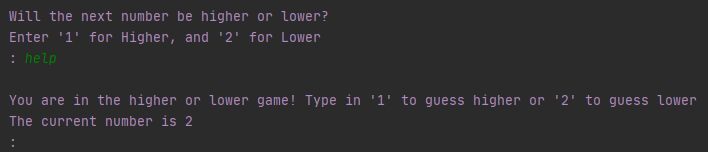
**Fig. E4 –** Tie (same number)



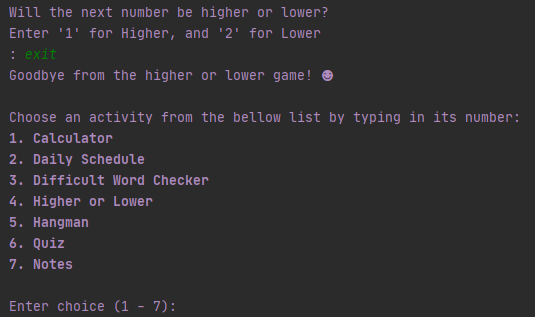
**Fig. E5 –** Tie (same selection choice)



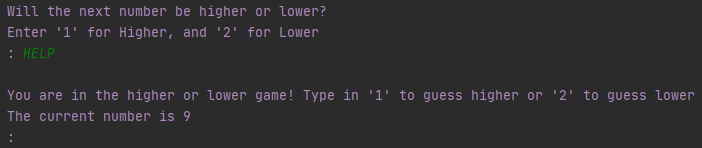
**Fig. E6 –** Help



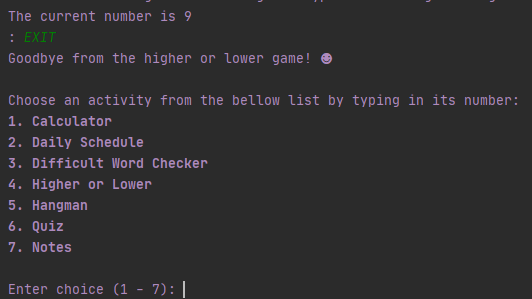
**Fig. E7 –** Exit



**Fig. E8 –** Help case sensitive



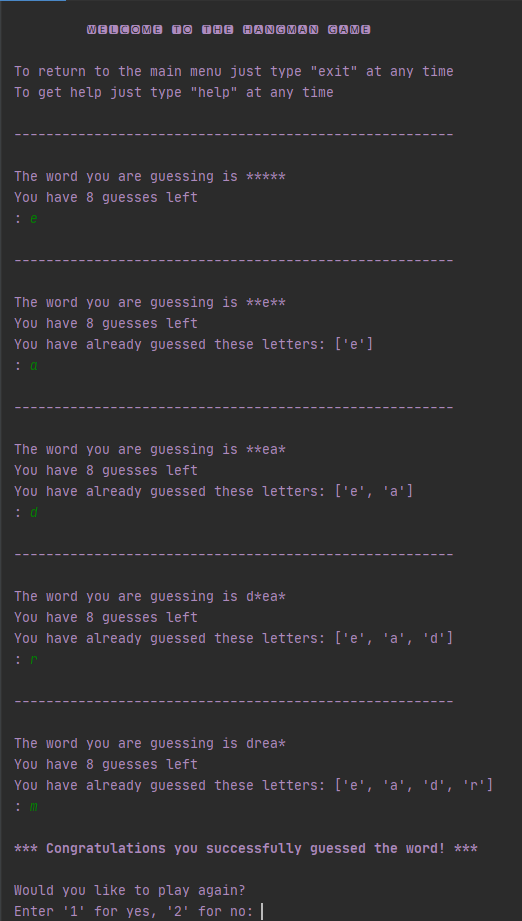
**Fig. E9 –** Exit case sensitive



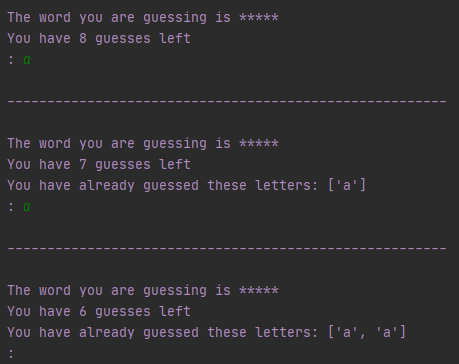
Appendix F - Hangman (hangman.py) feature module test plan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No. | Input | Reason | Expected | Actual |
| Test 1:  Correct | First Selection: ‘e’  Second Selection: ‘a’  Third Selection: ‘d’  Forth Selection: ‘r’  Fifth Selection: ‘m’ | Test selecting five correct answers fully reveals the word and declares user successful. Test proves that the correct characters in the guessed word are revealed on input, and the list of guessed letters correctly populates.  Additionally, the test shows that on a correct guess, the counter does not decrease. | Correct value is guessed 5 times in a row  Each time the guessed letter is revealed to the player in the word to guess.  The guessed letter is added to an array of letters so that the user can see what they have already input  Player is given a congratulations message and asked if they want to play again | \*\*\* Congratulations you have successfully guessed the word! \*\*\*  Would you like to play again?  See Fig. F1 for screenshot |
| Test 2: Double guess | First Selection: ‘a’  Second Selection: ‘a’ | Test selecting the same character more than once, proving that the number of guesses does decrease, and the letter is added to the guessed letter array so the user can see that they typed the same letter in again. | Value ‘a’ is input incorrectly  Guesses decrease from 8 to 7  Value ‘a’ is added to the letters guessed array and shown to the user  Value ‘a’ is input’  Guesses decreased from 7 to 6  Value ‘a’ is again added to the letters guessed array and shown to the user | You have already guessed these letters: [‘a’, ‘a’]  See Fig. F2 for screenshot |
| Test 3:  Incorrect | First Selection: ‘x’  Second Selection: y  Third Selection: ‘z’  Forth Selection: ‘q’  Fifth Selection: ‘w’  Sixth Selection: ‘1’  Seventh Selection: ‘\*’  Eighth Selection: ‘3’ | Test selecting 8 wrong guesses, showing that each time the guess counter decreases by 1, and the player loses the game after 8 tries. | Value ‘x’ input  Value ‘y’ input  Value ‘z’ input  Value ‘q’ input  Value ‘w’ input  Value ‘1’ input  Value ‘\*’ input  Value ‘3’ input  Game ends with a message after 8 unsuccessful letter guesses  The correct word is printed to the user so that they can see what they were trying to guess  The user is then asked if they want to play again | \*\*\* You didn’t manage to guess the word this time! \*\*\*  The word was smile  Would you like to play again?  See Fig. F3 for screenshot |
| Test 4:  Help | First Selection: “help” | Test selecting “help”, ensuring user is given a helpful message to guide them. | Value “help” is input  A useful message is shown guiding the user how to use the current feature – type in a character to guess the word | You are currently in the hangman game! Type in a single letter to guess the word  See Fig. F4 for screenshot |
| Test 5:  Exit | First Selection: “exit” | Test selecting “exit” showing that the module gracefully stops, and user returns to main menu selection screen. | Value “exit” is input  A goodbye message is shown, then the module returns to the main menu | Goodbye from the hangman game! (smiley face)  See Fig. 61 for screenshot |
| Test 6:  Help case sensitivity | First Selection: “HELP” | Test to see if case sensitivity has been considered. User Interface should be easy to use and ignore case sensitivity. | Value “HELP” is input  A useful message is shown guiding the user how to use the current feature – type in a character to guess the word | You are currently in the hangman game! Type in a single letter to guess the word  See Fig. F5 for screenshot |
| Test 7:  Exit case sensitivity | First Selection:“EXIT” | Test selecting “exit” showing that the module gracefully stops, and user returns to main menu selection screen. | Value “EXIT” is input  A goodbye message is shown, then the module returns to the main menu | Goodbye from the hangman game! (smiley face)  See Fig. F6 for screenshot |
| Test 8:  Invalid input | First Selection: ‘aa’ | Test shows that when more than one character is input, the user is told you can only enter one character at a time. | Value ‘aa’ is input  User is given a message stating to only put in a single character, and to try again | You can only input a single letter, please try again (thumbs up)  See Fig. F7 for screenshot |

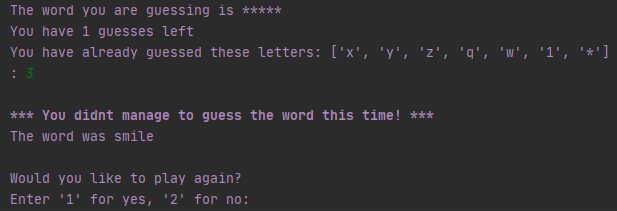
**Fig. F1 –** Win



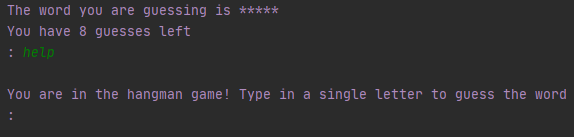
**Fig. F2 –** Selection repeat



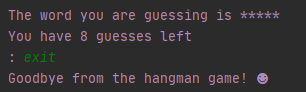
**Fig. F3 –** Lose



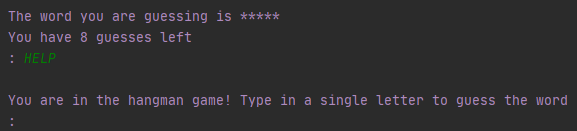
**Fig. F4 –** Help



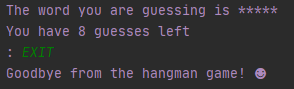
**Fig. F5 –** Exit



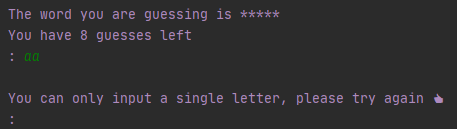
**Fig. F6 –** Help case sensitive



**Fig. F7 –** Exit case sensitive

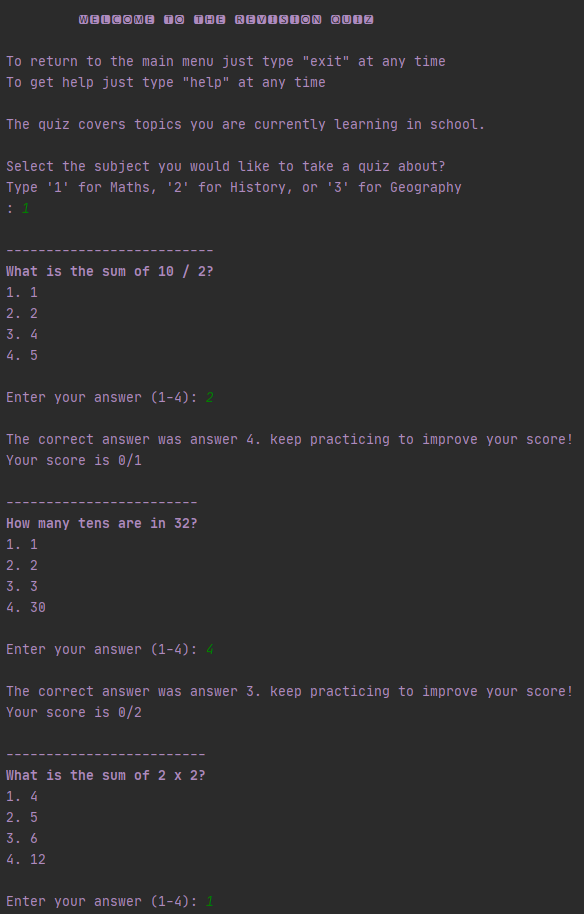
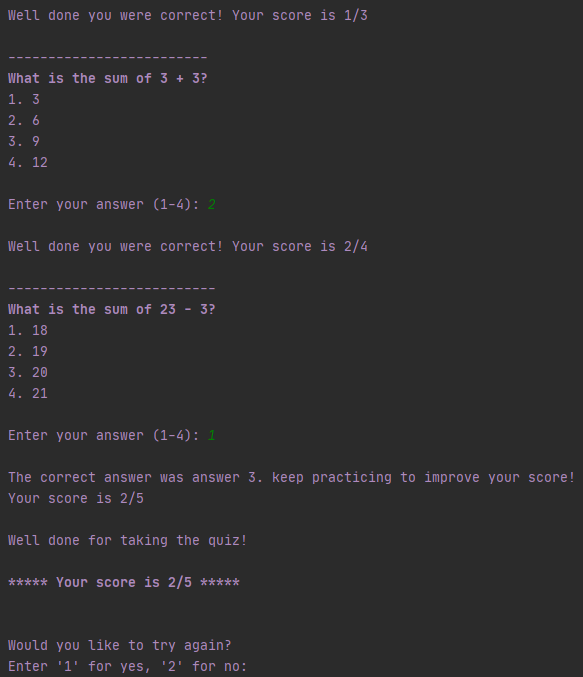


**Fig. F8 –** Multi character input

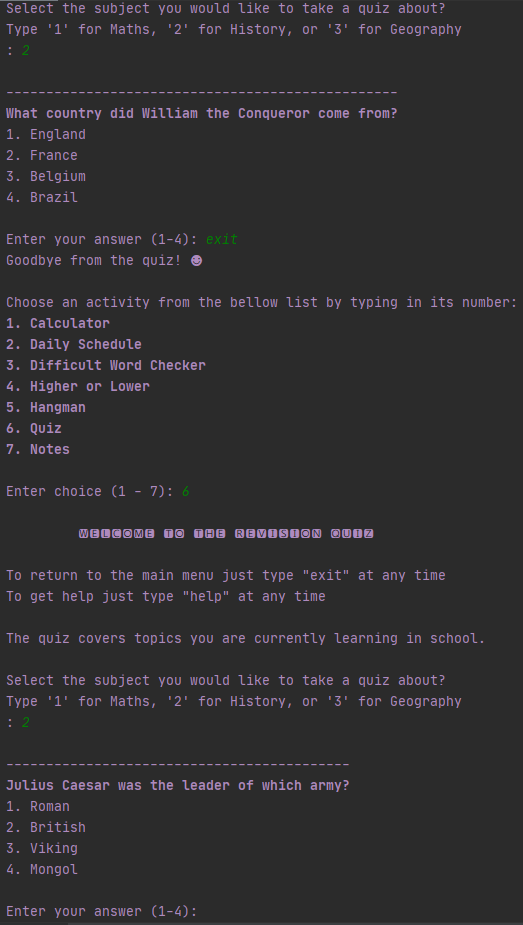


Appendix G - Quiz (quiz.py) feature module test plan

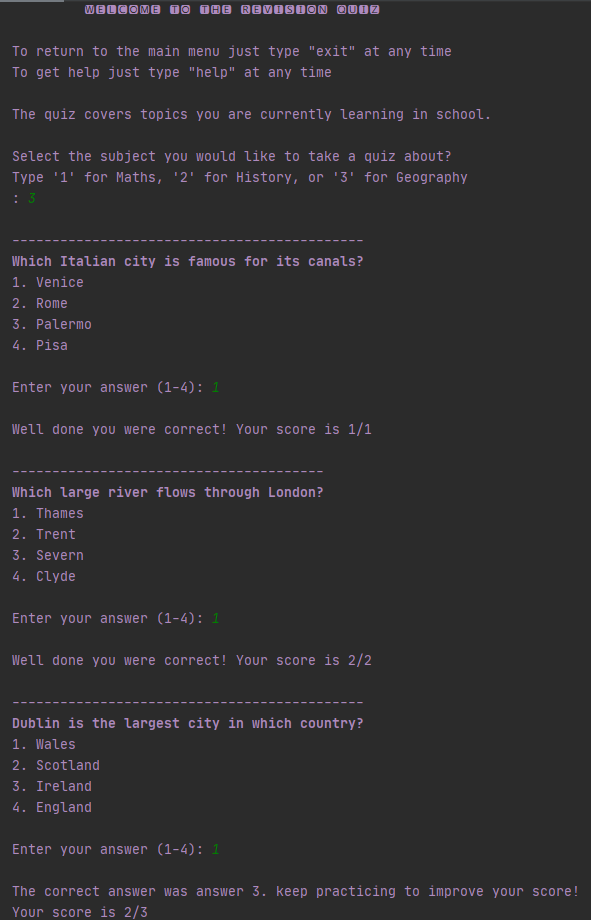
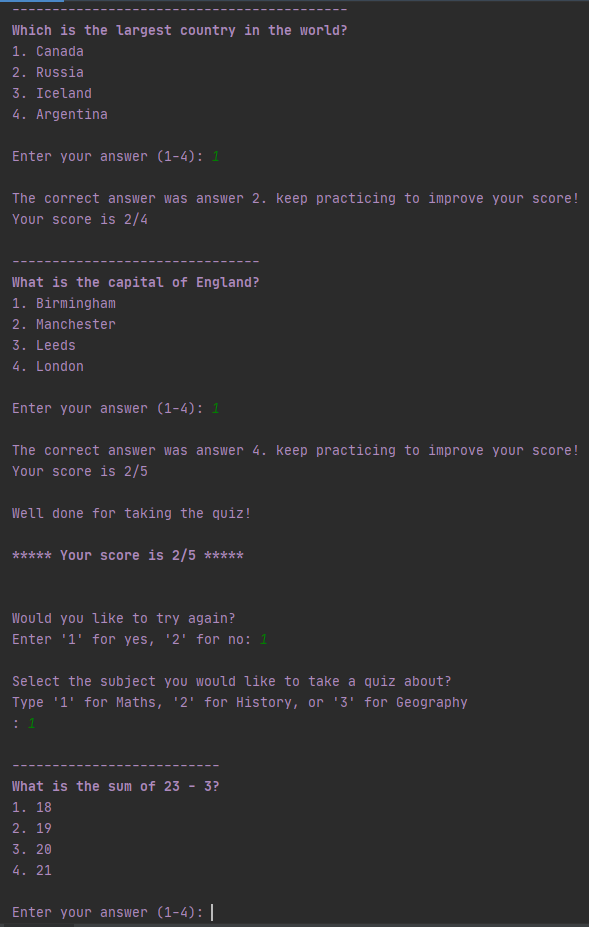
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Input | Reason | Expected | Actual |
| Test 1:  Mixed score | First Selection: 1 (Maths)  Second Selection: 2 (incorrect)  Third Selection: 4(incorrect)  Forth Selection: 1 (correct)  Fifth Selection: 2 (correct)  Sixth Selection: 1 (incorrect) | Test selecting five answers with a mixture of right and wrong answers. This test proves that the score is correctly incremented, and that questions are not repeated.  Additionally, the test proves that a subject quiz can be successfully entered. | Value 1 is input  Maths quiz is given  Value 2 is input  Answer is wrong, score 0/1 is given  Value 4 is input  Answer is wrong, score 0/2 is given  Value 1 is input  Answer is correct,  score 1/ 3 is given  Value 2 is input  Answer is correct,  score 2/4 is given  Value 1 is input  Answer is wrong,  score 2/5 is given  Total score given as 2/5  User is given a well-done message for taking the quiz and asked if they want to try again | Well done for taking the quiz!  \*\*\*\*\* Your score is 2/5 \*\*\*\*\*  Would you like to try again?  See Fig. G1 for screenshot |
| Test 2:  Random questions | First Selection: 2 (History)  Second Selection: exit  Third Selection: 6 (quiz)  Forth Selection: 2 (History) | Test selecting the same quiz topic twice showing that the questions are randomised  Additionally, test shows that ‘exit’ successfully leaves the quiz module | Value 2 is input History quiz starts  Value ‘exit’ is input  Quiz module exits back to the main module  Value 6 is input  Quiz module starts  Value 2 is input History quiz starts with a different first question | The first question given is:  What country did William the Conqueror come from?  The first question on re-entering the quiz was:  Julius Caesar was the leader of which army?  See Fig. G2 for screenshot |
| Test 3:  Retry | First Selection: 3 (geography)  Second Selection: 1  Third Selection: 1  Forth Selection: 1  Fifth Selection: 1  Sixth Selection: 1  Seventh Selection: 1 (restart)  Eighth Selection: 1 (maths) | Test selecting retry. The test proves that the quiz feature successfully starts a new quiz of the users choosing after finishing a quiz. | Value 3 input  Geography quiz starts  Value 1 input 5 times as the answer  Quiz ends stating user score  The user is then asked if they want to play again  Value 1 (play again) is entered  The user is given a choice of which quiz topic they would like  Value 1 input  Maths quiz starts | \*\*\* Your score is 2 / 5 \*\*\*  Would you like to play again?  Enter ‘1’ for yes. ‘2’ for no.  User selects 1 to try again, then selects 1 for Maths  What is the sum of 23 – 3?  See Fig. G3 for screenshot |
| Test 4:  Help | First Selection: “help” | Test selecting “help”, ensuring user is given a helpful message to guide them. | Value “help” is input  A useful message is shown guiding the user how to use the current feature – type in a number to choose the quiz topic | You are in the revision quiz! Type in a number to choose the quiz topic  See Fig. G4 for screenshot |
| Test 5:  Help second  input | First Selection: 1 (maths)  Second Selection: “help” | Test selecting “help” during the quiz, ensuring user is given a helpful message to guide them. | Value 1 is input  Maths quiz starts  Value “help” is input  A useful message is shown to the user, guiding them to select and answer number | You are in the revision quiz! Choose your answer by typing is number (1-4)  See Fig. G5 for screenshot |
| Test 6:  Help case sensitivity | First Selection: “HELP” | Test to see if case sensitivity has been considered. User Interface should be easy to use and ignore case sensitivity. | Value “HELP” is input  A useful message is shown guiding the user how to use the current feature – type in a number to choose the quiz topic | You are in the revision quiz! Type in a number to choose the quiz topic  See Fig. G6 for screenshot |
| Test 7:  Exit case sensitivity | First Selection:“EXIT” | Test selecting “EXIT” showing that the module gracefully stops, and user returns to main menu selection screen. Test proves that exit is detected regardless of case sensitivity making module easy to use. | Value “EXIT” is input  A goodbye message is shown, then the module returns to the main menu | Goodbye from the quiz! (smiley face)  See Fig. G7 for screenshot |
| Test 8:  Invalid input quiz selection | First Selection: ‘aa’ | Test not selecting an expected answer, showing that when more than one character is input, the user is prompted to try again. | Value ‘aa’ is input  User is given a message asking them to try again, giving the numbers to type for each topic to help them out | Please try again, type ‘1’ for Maths, ‘2’ for History and ‘3’ for Geography (thumbs up)  See Fig. G8 for screenshot |
| Test 9:  Invalid input answer selection | First Selection: 1 (maths)  Second Selection: ‘aa’ | Test not selecting a valid answer during a quiz question, proving that the user is guided to select an answer in range. | Value 1 is input  Maths quiz starts  Value ‘aa’ is input  User is given a message to type a number between 1 and 4 to select and answer | Please try again, type the number of the answer you wish to enter between 1 – 4 (thumbs up)  See Fig. G9 for screenshot |

**Fig. G1 –** Score 

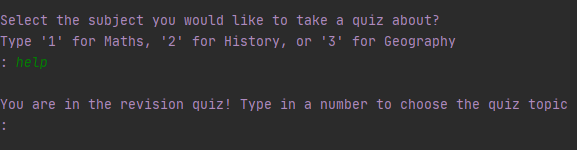
**Fig. G2 –** Randomise



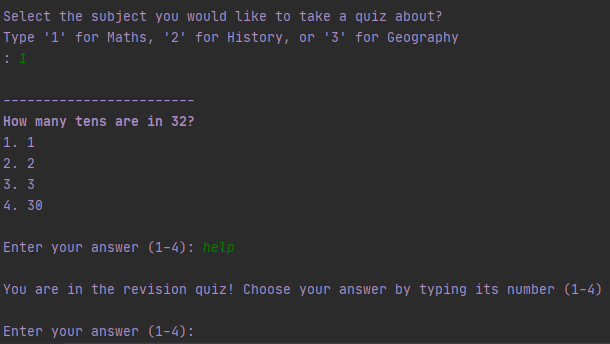
**Fig. G3 –** Retry



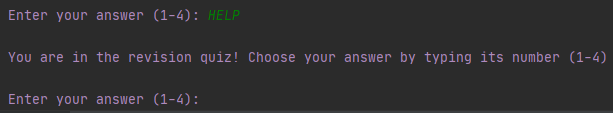
**Fig. G4 –** Help (first selection)



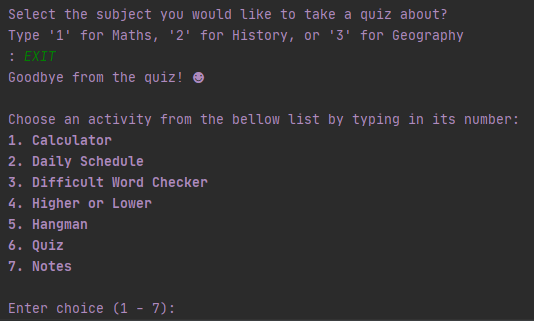
**Fig. G5 –** Help (second selection)



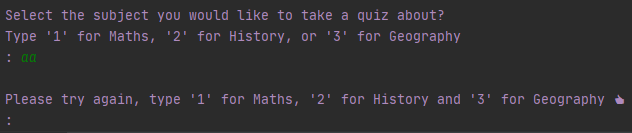
**Fig. G6 –** Help case sensitive



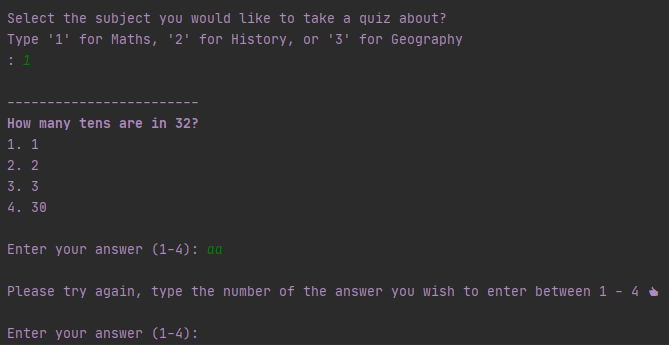
**Fig. G7 –** Exit case sensitive



**Fig. G8 –** Selection out of range (first selection)



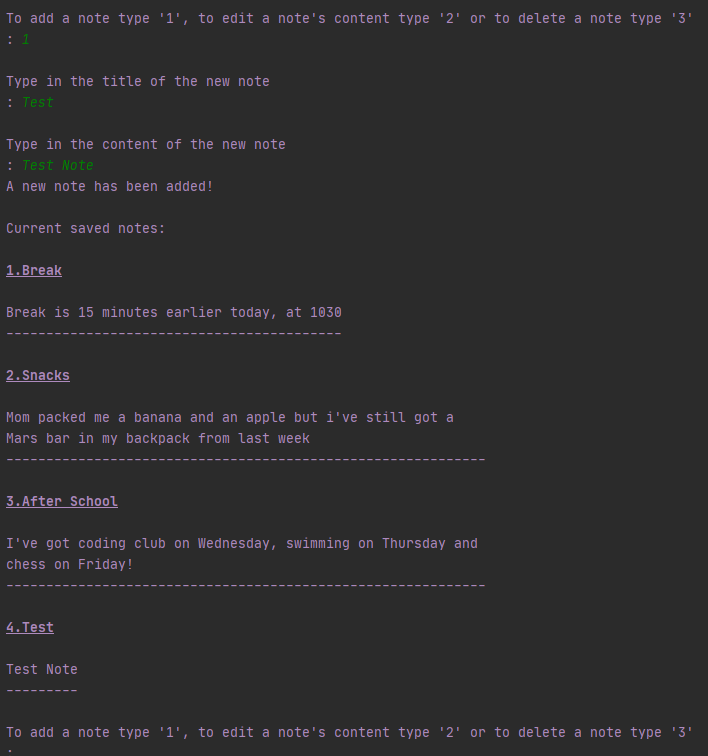
**Fig. G9 –** Selection out of range (second selection)



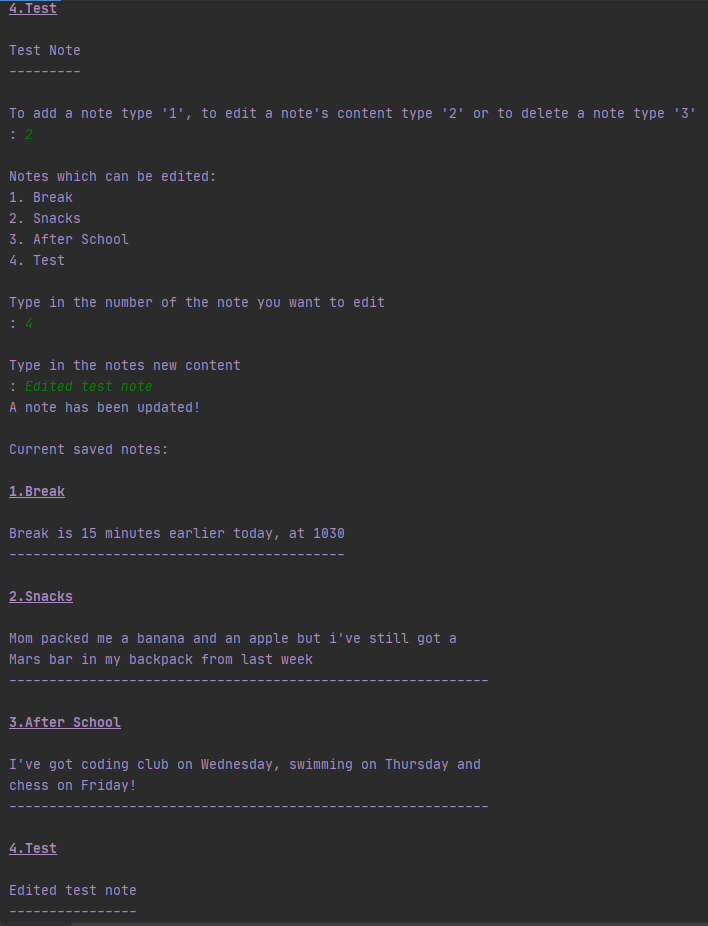
Appendix H - Notes (notes.py) feature module test plan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No. | Input | Reason | Expected | Actual |
| Test 1:  New note | First Selection: 1 (new note)  Second Selection: ‘Test’  Third Selection:‘Test note’ | Test adding a new note, proving that notes can be successfully added to the application | Value 1 is input  User prompt to add title of new note  Value ‘Test’ is input  User prompt to add new note content  Value ‘Test Note’ input  All notes, including new note are printed for the user | 4. Test  Test Note  See Fig. H1 for screenshot |
| Test 2:  Edit note | First Selection: 2 (edit)  Second Selection: 4 (test note)  Third Selection: “Edited test note” | Test editing a current note, ensuring a user can easily select a note and edit its contents. | Value 2 is input  User is prompted to input the notes number to edit it  Value 4 is input  Test note is selected  Value “Edited test note” is input  All notes are printed, showing the new contents of the test note | 4. Test  Edited Test Note  See Fig. H2 for screenshot |
| Test 3:  Delete note | First Selection: 3 (delete)  Second Selection: 4 (test note) | Test selecting delete, showing that the chosen note is then removed for the notes list. | Value 3 input  User is prompted to enter number of note to delete  Value 4 is input  Test note it deleted  All notes are printed, with test note no longer in the notes list | Note 1 – Break  Note 2 – Snacks  Note 3 – After School  Note 4 – Test, is no longer visible to the user  See Fig. H3 for screenshot |
| Test 4:  Persistent data | Fist Selection: 4 (note 4)  Second Selection: “Persistent data test”  Third Selection: “exit”  Fourth Selection: 7(notes module) | Test editing a note to show data persistent over the applications session and in the notes.txt file for long term persistence when the entire application is closed. The modified note remains when the notes module is loaded back up by the user, and the updated entry is in the .txt file | Value 1 is input  Value “Persistent data test” is input  Notes are displayed, with note 4’s content being updated to “Persistent data test”  Value “exit” is input  The module gracefully exits with a goodbye message to the user  Value 7 is input  The notes module is loaded and outputs all the notes, with note 4 displaying the correct content | 4. Test  Persistent data test  See Fig. H4 for screenshot of the output, and Fig. H5 for screenshot of the txt file before and after |
| Test 5:  Invalid input | First Selection: “aaa” | Test selecting a note which does not exist, with the input being out of range. Test confirms that incorrect inputs are handled gracefully | Value “aaa” is input  Value is not a valid input (only 1, 2 or 3)  Program provides user with a helpful hint, asking to input a number | Please try typing a number again (thumbs up)  See Fig. H6 for screenshot |
| Test 6:  Help | First Selection: “help” | Test selecting “help”, ensuring user is given a helpful message to guide them. | Value “help” is input  A useful message is shown guiding the user how to use the current feature – instructions given how to add, edit or delete a note | You are in the notes feature! To add a note type ‘1’, to edit a note type ‘2’ or to delete a note type ‘3’  See Fig. H7 for screenshot |
| Test 7:  Help second selection and case sensitivity | First Selection: 1 (add note)  Second Selection: ‘HELP’ | Test selecting “HELP” as the second input. Test shows that tailored help is given depending on what the user is doing in the application (in this case adding a note).  Additionally, test proves that case sensitivity is not an issue. | Value 1 is input  Value ‘HELP’ is input  A useful message is shown guiding the user how to use the current feature – instructions given to type the new notes title | You are in the notes feature! Type in the new notes title  See Fig. H8 for screenshot |
| Test 8:  Exit case sensitivity | First Selection: ‘EXIT’ | Test selecting “EXIT” showing that the module gracefully stops, and user returns to main menu selection screen. Test proves that exit is detected regardless of case sensitivity making module easy to use. | Value “EXIT” is input  A goodbye message is shown, then the module returns to the main menu | Goodbye from notes! (smiley face)  See Fig. H9 for screenshot |

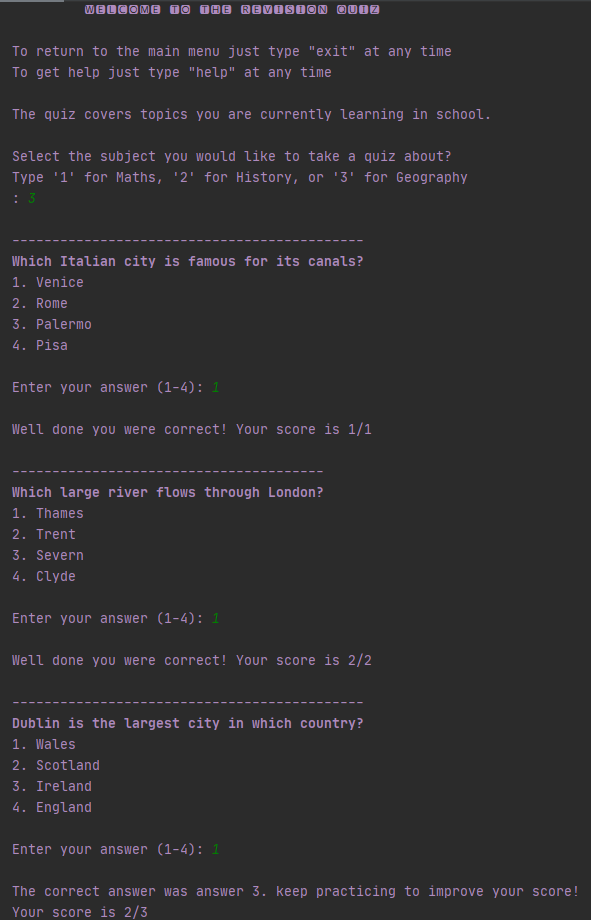
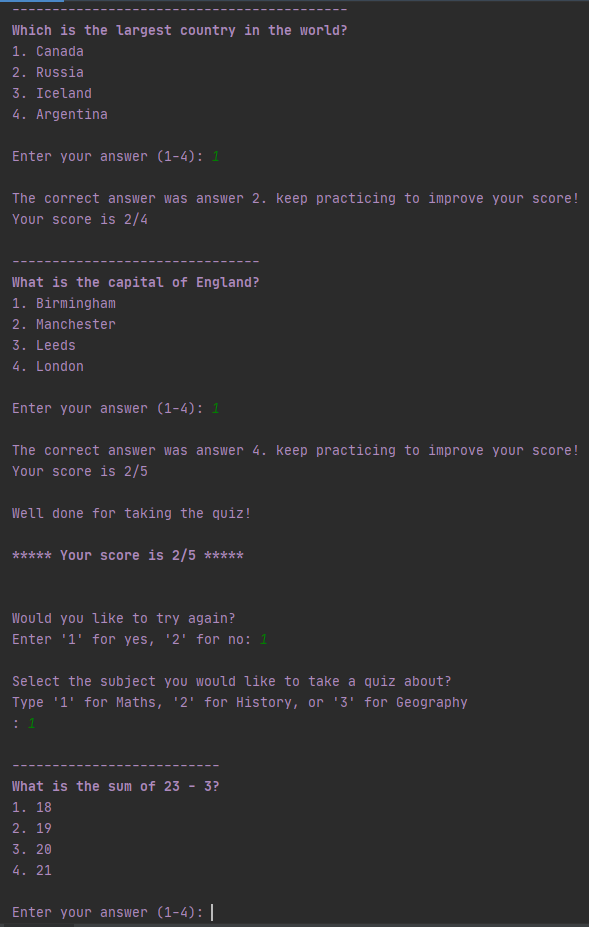
**Fig. H1 –** Add note

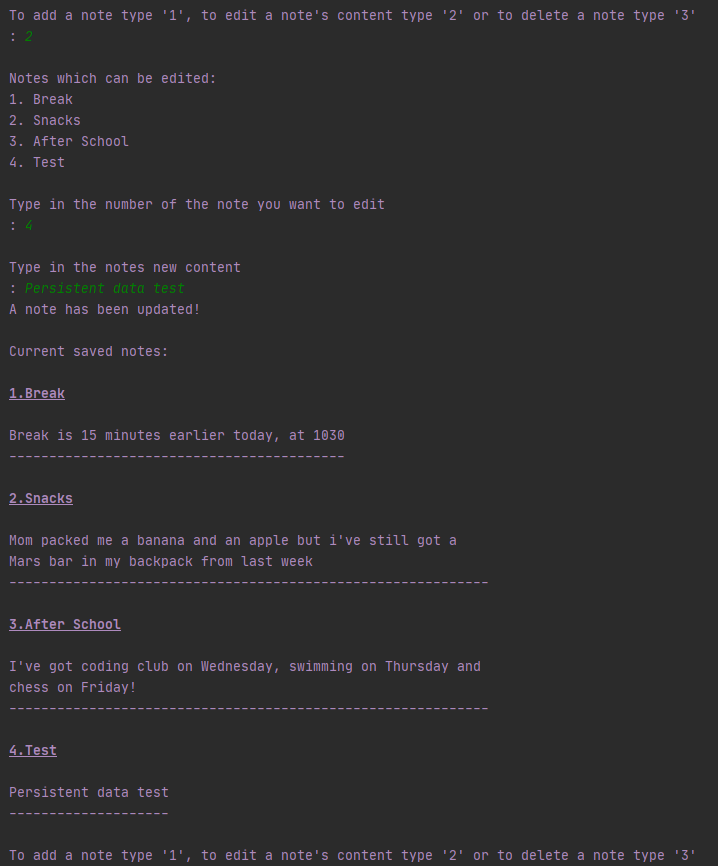


**Fig. H2 –** Edit Note



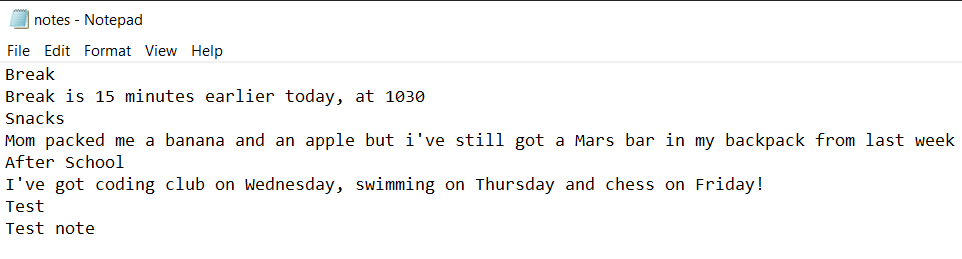
**Fig. H3 –** Delete note



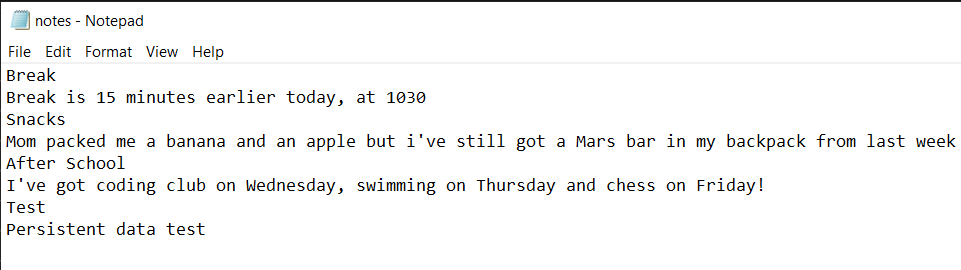
******Fig. H4 –** Persistent data session

**Fig. H5 –** Persistent data txt

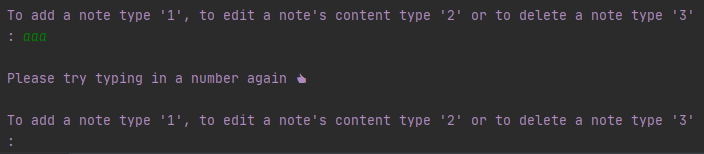
**Before edit**



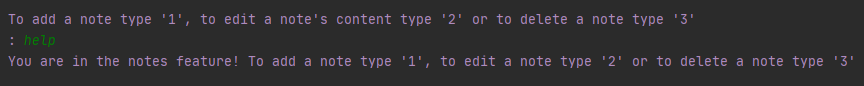
**After edit**

****

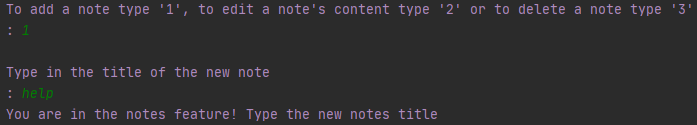
**Fig. H6–** Invalid choice



**Fig. H7 –** Help



**Fig. H8 –** Tailored help and case sensitivity



**Fig. H9 –** Help case sensitivity

