Assignment: JustynMahen_21029112_Report

Justyn Mahen ID 21029112

Campus: Bentley, Perth

Curtin University

ISAD 1000 Introduction to Software Engineering

Coordinator: Nimalika Fernando Thudugala Mudalige

Due Date: May 31, 2022

Word Count: 4486

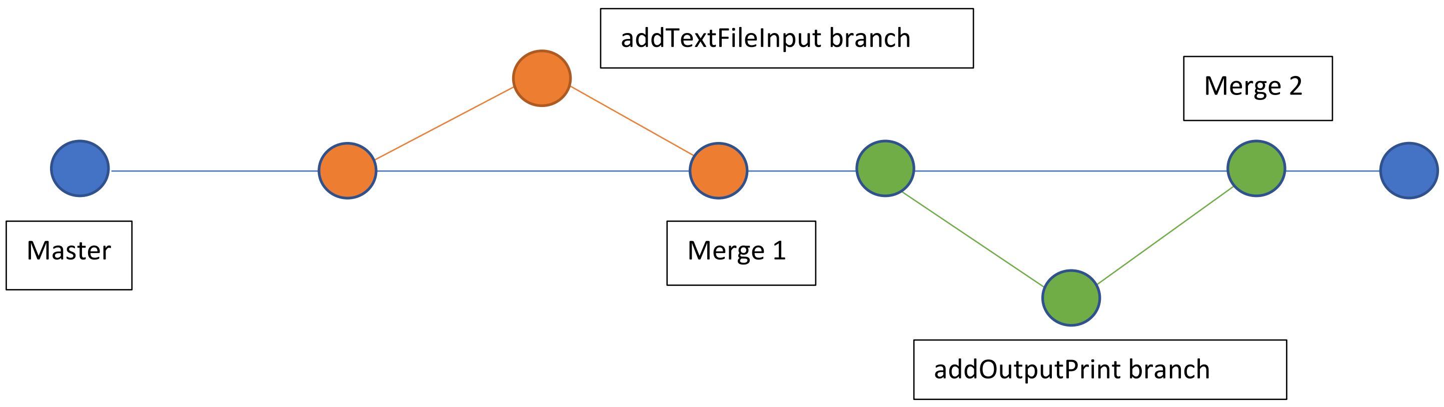
Overview

This report addresses my workings and accomplishments of each stage of the ISAD assignment. Throughout this report, collating work data will be used in conjunction with explanations to provide an easy understanding of the programs inside the git repository along with their code functionalities, modules, test designs and implementations. In addition, the git version control processes will also be outlined.

Upon reviewing the assignment, I have chosen to implement functionalities with the ability to convert strings to both upper and lower case, identify the presence of numeric values in strings, identify if a number should be considered valid , convert a string to upper or lower case after the removal of any numeric values and convert between various units of time such as seconds, minutes and hours. Each functionality has been assigned their respective names; Case Converter, Numeric Identifier, Numeric Validator, Numeric Remover and Time Converter. Every functionality reads user keyboard and file inputs, and prints outputs to both the terminal screen and a designated output file.

Version Control

Throughout this assignment, git was the chosen software for version control and it has been used to assist in keeping track of code and document changes. The diagrams below respectively illustrates a short git version control branch plan and a commit plan for the assignment:



Master Branch Commits:

Initial Commit → C1 Modules Completed → C2 Modules Completed → Final Commit

addTextFileInput Branch Commits:

Input Files Completed → Input Functionalities Completed

addOutputPrint Branch Commits:

Output Functionalities Completed → Output Files Completed

Full Steps:

1) Establish local git repository called Mahen_Justyn21029112_ISErepo
   Sub directories: code
   documents
2) Make Category 1 Modules in a file
   Commit: C1 Modules Completed
3) Make Category 2 Modules in a file
   Commit: C2 Modules Completed
4) Create addTextFileInput branch
5) Create input files
   Commit: Input Files Completed
6) Create Input Functionalities in C1 and C2 Modules
   Commit: Input Functionalities Completed
7) Merge addTextFileInput branch with master branch
8) Create addOutputPrint branch
9) Create Output Functionalities in C1 and C2 Modules
   Commit: Output Functionalities Completed
10) Run files and Output Files will be created
    Commit: Output Files Completed
11) Merge addOutputPrint branch with master branch
12) Commit: Final Commit

I have decided to use 2 other branches called addTextFileInput and addOutputPrint because I will be implementing more input functionalities into the code and would like the old version to still be present in case I have made a mistake and need to revert back. Branch addTextFileInput will be created after both Category 1 and 2 modules have been completed with user input functionalities. It will be merged back into the master branch after all modules have successfully had file input functionalities implemented. Branch addOutputPrint will be created after the merge of addTextFileInput and will be merged back into the master branch after all modules can print their outputs into txt or csv files. I have planned 2 commits to occur during each branch. One commit will be for adding the input and output files to git and the other commit will be for saving and adding the modified code files to git after their respective input and output functionalities have been implemented.

<u>In the final production of the assignment, there was once where I had forgot to add a file before committing hence I had to make another commit called 'Forgot to git add a file commit'. In addition to this, I also had to redo the git repository a few hours before the due date because the repository got corrupted and I could not zip it. This has resulted in inaccurate times recorded in the git logs.</u>

Git Logs

```
commit 5977c11af35fc66a5a0c2ce4061a495c458a8862 (HEAD -> master)
Author: Justyn Mahen <justyn2s04mahen23@gmail.com>
Date:   Tue May 31 22:55:07 2022 +0800

    Final Commit

commit 5df60b267c4e5790b85c290e23b5ae2ed575c47a (addOutputPrint)
Author: Justyn Mahen <justyn2s04mahen23@gmail.com>
Date:   Tue May 31 22:52:31 2022 +0800

    Forgot to git add a file commit

commit 6b4ac3b63e46cdd3bc12cdfb83fcbae72c45fcc9
Author: Justyn Mahen <justyn2s04mahen23@gmail.com>
Date:   Tue May 31 22:51:19 2022 +0800

    Output Files Completed

commit 540cd5bd5001c5930bca15db393426fc3f61556f
Author: Justyn Mahen <justyn2s04mahen23@gmail.com>
Date:   Tue May 31 22:44:20 2022 +0800

    Output Functionalities Completed

commit cbcd053676c410b3dfbbfb29ed25734e742c33bb (addTextFileInput)
Author: Justyn Mahen <justyn2s04mahen23@gmail.com>
Date:   Tue May 31 22:26:35 2022 +0800

    Input Functionalities Completed

commit ce60fbe8efd518de3d6605d3ea227e6910cfd011
Author: Justyn Mahen <justyn2s04mahen23@gmail.com>
Date:   Tue May 31 22:24:55 2022 +0800

    Input Files Completed

commit d6d8d36ad81303a922399ebf728c5abda7dc1a8b
Author: Justyn Mahen <justyn2s04mahen23@gmail.com>
Date:   Tue May 31 22:13:11 2022 +0800

    C2 Modules Completed
```

```
commit 51ea731137c517e9939384c1e719a9fd38051896
Author: Justyn Mahen <justyn2s04mahen23@gmail.com>
Date:   Tue May 31 22:11:31 2022 +0800

    C1 Module Completed

commit 49a85992df95b24b9fb69d1c2bd7680454945082
Author: Justyn Mahen <justyn2s04mahen23@gmail.com>
Date:   Tue May 31 22:09:07 2022 +0800

    Initial Commit
```

Preliminary description of modules

This section of the report will cover the descriptions of implemented modules. Modularity guidelines have been followed during the planning and implementation of each module. Category1.py contains modules for all functionalities in Category 1. Modules have been named Case Converter, Numeric Identifier, Numeric Validator and Numeric Remover. Category2.py contains modules for all functionalities required for Category 2. Modules have been named Hours2Mins, Mins2Secs, Secs2Mins, Mins2Hours and Time Converter. Both files contain a main() module that calls the other modules. The main() module is then run at the end of the file. All modules will be using if statements to achieve their goal and with statements to read and print file inputs and outputs. Because this section covers preliminary module descriptions, detailed explanation of the code statements will not be covered.

Module Case Converter is the first module that was implemented in the Category1.py file. The module's goal is to convert a given string to upper or lower case, hence the short and easy to understand name of 'Case Converter'. Inputs are taken via keyboard inputs and strings that are passed from a file called 'testdata.txt'. The module first asks a user to first choose if they want to convert a text to upper or lower case. It does this by asking the user to either input L or H to choose the desired case. If L is inputted, the lower case converter will be used, if H is inputted, the upper case converter will used. The module then asks for a text to be inputted which will then have its case converted and outputted unto the terminal screen. Case Converter then displays its ability to read a string from a txt file, case convert that string in accordance to the converter the user has chosen and output its result unto the terminal screen. Case Converter is the first module to be called by the main() function. Case Converter contains absolutely no coupling. This is due to no presence of global variables and control flags, and Case Converter does not use code from another module. Case Converter is also of high cohesion. This function also contains duplicate code for writing outputs to an output file. The duplicate pieces of code have been implemented due to ease of reuse and understandability of the code.

Module Numeric Identifier is the second module that was implemented in the Category1.py file. The module's goal is to identify if there are any numeric values in a given string, hence the name 'Numeric Identifier'. Inputs are pulled using keyboard inputs and strings that are passed from the testdata.txt file. This module uses list comprehensions to accomplish its goal. The module first asks the user to input a text using the keyboard. It then scans the inputted text for any numeric values. If the text contains numeric values, it outputs the numbers to the terminal in a list. However, if there are no numbers in the inputted text, the module will display ‘There are no numeric values in your text’ unto the terminal. Numeric Identifier further displays its capabilities by reading a string from a txt file, scanning said string and outputting the outcome. All outputs are printed unto the terminal screen and saved into a file called C1_outputs.txt. Once again, this module contains no coupling as no global variables and control flags are present. Numeric Identifier performs only 1 task of identifying numeric values, hence it has no control flags which suggests that there is high cohesion. Numeric Identifier further contains duplicate code for writing outputs to an output file. The duplicate pieces of code have been implemented due to ease of reuse and understandability of the code.

Module Numeric Validator is the third module that has been implemented into Category1.py. This module’s objective is to identify if a string is a valid number. This has been accomplished by converting the string to an integer and scanning if it’s a valid number. The module first asks the user to input a number between 0 to 10. The inputted string is then converted to an integer and is passed through an if statement. If the inputted number is between 0 and 10, the module verifies it as valid and prints the output into the terminal screen and saves it into C1_outputs.txt. If however, the number is not between – and 10, the module verifies the number as invalid and prints out ‘Invalid Number’ unto the terminal screen and saves ‘Invalid Number, inputted number’ into the outputs file. Like all implemented modules, Numeric Validator also reads a string from testdata.txt and feeds the string through the module. The output is also printed unto the terminal screen and saved into C1_outputs.txt. Numeric Validator contains no global variables and control flags hence there is no coupling. The module also is of high cohesion because it only performance the 1 task of validating a number. For ease of reuse and code understandability, duplicate pieces of code have been implemented for writing outputs to an output file

The final non-main module in Category1.py is called Numeric Remover. This module removes any numeric value in a string and converts the said string into upper or lower case. This module acts like a combination of both Case Converter and Numeric Identifier. Although this module has no coupling, because there are no global variables or control flags, it does have quite low cohesion because this module is doing 2 tasks. One task being removing numeric values in the input string and the other being converting the string into upper or lower case. Similar to Numeric Identifier, Numeric Remover first asks the user to input a text using the keyboard. It then scans the inputted text for any numeric values. The module then asks a user to first choose if they want to convert a text to upper or lower case. It does this by asking the user to either input L or H to choose the desired case. If L is inputted, the lower case converter will be used, if H is inputted, the upper case converter will used. Similar to all implemented modules, Numeric Remover also reads an input string from testdata.txt and feeds it through the module, displaying the output of all inputs to the terminal screen and saves them to C1_outputs.txt. This function also contains duplicate code for writing outputs to an output file. The duplicate pieces of code have been implemented due to ease of reuse and understandability of the code.

Category1.py contains a final module called main. This module simply acts like a code executor for all other modules. Although there is no specific requirement of implementing this module, it is good practice to increase modularity in the code. This function also has high coupling as is being fed code from all other modules. There is also extremely low cohesion because it is performing many tasks. However, there is no redundant code.

All modules in Category2,py are extremely similar. The first module in Category2.py has been named Hours2Mins. I have chosen this name because of the goal of the module, which is to convert hours to minutes. Like all modules in both Category1.py and Category2.py, inputs are taken both by keyboard inputs and strings that are read from a file. Hours2Mins first asks the user to input the number of hours they wish to convert to minutes. It then prints the output into the terminal screen and saves it to C2_outputs.csv. It then reads several hours from an file called testdata.csv. Category 2 modules read inputs and prints outputs to and from a csv file instead of a txt file. The inputs read from the file is then fed into the module which displays outputs both in the terminal screen and writes into C2_outputs.csv. There is no coupling in this module because there are no global variables or control flags and no other modules are fed into Hours2Mins. There is also high cohesion as it as well performs the sole task of converting hours to minutes. There is no duplicate or redundant code in this function.

The second module in Category2.py has been named Mins2Secs. I have chosen this name because of the goal of the module, which is to convert minutes to seconds. Inputs are taken both by keyboard inputs and strings that are read from testdata.csv. Mins2Secs first asks the user to input the number of minutes they wish to convert to seconds. It then prints the output into the terminal screen and saves it to C2_outputs.csv. It then reads several minutes from testdata.csv. The inputs read from the file is then fed into the module which displays outputs both in the terminal screen and writes into C2_outputs.csv. There is no coupling in this module because there are no global variables or control flags and no other modules are fed into Mins2Secs. There is also high cohesion as it as well performs the sole task of converting minutes to seconds. This function contains no duplicate or redundant code.

An inverse of the first module, Mins2Hours is the third module in this file. The goal of the module is to convert minutes to hours. Inputs are taken both by keyboard inputs and strings that are read from testdata.csv. Mins2hours first asks the user to input the number of minutes they wish to convert to hours. It then prints the output into the terminal screen and saves it to C2_outputs.csv. It then reads several minutes from testdata.csv. The inputs read from the file is then fed into the module which displays outputs both in the terminal screen and writes into C2_outputs.csv. There is no coupling in this module because there are no global variables or control flags and no other modules are fed into Mins2Hours. There is also high cohesion as it as well performs the sole task of converting minutes to hours. Duplicate and redundant code are not present in this function.

The 4th module, Secs2Mins, converts seconds to minutes. Inputs are taken both by keyboard inputs and strings that are read from testdata.csv. Secs2Mins first asks the user to input the number of seconds they wish to convert to minutes. It then prints the output into the terminal screen and saves it to C2_outputs.csv. It then reads several seconds from testdata.csv. The inputs read from the file is then fed into the module which displays outputs both in the terminal screen and writes into C2_outputs.csv. There is no coupling in this module because there are no global variables or control flags and no other modules are fed into Secs2Mins. There is also high cohesion as it as well performs the sole task of converting seconds to minutes. This function contains no duplicate or redundant code.

The final non-main module in Category2.py is called Time Converter. This module gives the user a choice to choose between all the 4 modules previously discussed. This module has low coupling because the other modules are being fed into the Time Converter. It is also of low cohesion because this module is doing 4 tasks. Time Converter first asks a user to first choose if they want to their time unit converter. It does this by asking the user to either input H, M, I or S to choose the desired converter. If H is inputted, Hours2Mins is passed through. If M is inputted, Mins2Secs is passed through. If I is inputted, Mins2Hours is passed through and if S is imputed, Secs2Mins is passed through. If the input is neither of these options, ‘Invalid Selection’ is printed unto the terminal screen. Unlike all other non-main functions in Category2.py, this function also contains duplicate code for both ease of reuse and code understandability.

Category2.py contains a final module called main. This module simply acts like a code executor for all other modules. Although there is no specific requirement of implementing this module, it is good practice to increase modularity in the code. This function also has high coupling as is being fed code from all other modules. There is also extremely low cohesion because it is performing many tasks. However, there is no redundant code.

Modularity

Using python 3 and considering the good modularity principles discussed in week 7's lecture, I have successfully implemented the modules that run without syntax errors.

Checklist

Is code hard to modify due to poor modularity?
Do global variables exist outside functions?
If there are global variables, is there a way to rewrite code to lessen them down?
If there are control flags, is there a way to rewrite code to lessen them down?
Is there high degree of unnecessary coupling in modules and functions?
Is there low degree of unnecessary cohesion in modules and functions?
If the module performs several unrelated tasks, can they be further broken up (refactored)?
Are there redundant/duplicated code that increase code complexity?
Which redundant pieces of code that do not benefit from reuse can be removed to reduce code complexity?
Can loops and arrays be used to refactor duplicate pieces of code?

Review for Category1.py and Category2.py

The code is easy to modify due to good use of modularity.
Global variables do not exist outside functions.
There are no global variables, hence there is no need to rewrite code to lessen them.
There are no control flags, hence there is no need to rewrite code to lessen them.
There is no high degree of unnecessary coupling in modules in functions. Neither is there low degree of unnecessary cohesion. Time Converter, Numeric Remover and main modules all have low cohesion and/or high coupling due to the nature of the modules and/or because there will be lesser redundant code.
Unrelated tasks cannot be further refactored to a simpler format.
There is duplicated code that increases code complexity but they will not be refactored due to benefits of reuse.
There are no redundant pieces of code that do not benefit from reuse.
Loops and arrays have been used to ease code complexity. Refactoring cannot be done because the first time code was implemented, it cannot be further refactored to a simpler format.

Issues Identified from Checklist and Review

The first version of code implementation has been extremely well done that after going through the checklist and doing a review, no issues have been identified.

Revised Module Descriptions

Not needed due to no refactoring.

Test Designs (Black Box Testing)

This section of the report presents information on equivalence partitioning or boundary value analysis (BVA) test designs for each module.

Submodule: CaseConverter

Test Design: Equivalence Partitioning

Imports: text (string)

Exports: text in upper or lower case (string)

Converts a text to upper or lower case after user has selected their case converter.

Category	Test Data	Expected Result
User selects for text goes through upper case converter.	Mahen	MAHEN
User selects for text goes through lower case converter.	Justyn Samuel Mahen	justyn samuel mahen
User does an invalid selection	I have an i20N	Invalid Selection

Submodule: NumericIdentifier

Test Design: Equivalence Partitioning

Imports: text (string)

Exports: identified numbers (string)

Identifies numeric values in a string.

Category	Test Data	Expected Result
Text contains numeric values.	My last 4 student ID digits are 9112	4,9,1,1,2
Text does not contain numeric values	I am a student at Curtin University.	There are no numerics in your text.

Submodule: NumericValidator

Test Design: BVA

Imports: number (integer)

Exports: validate (string)

Identify if a number is a valid number.

Boundary	Test Data	Expected Result

Invalid/0	Number = -5
	Number = -100
		Invalid Number
		Invalid Number

0/10	Number = 7
	Number = 10
		Your number, 7, is a valid number
		Your number, 10, is a valid number

10/Invalid	Number = 11
	Number = 100
		Invalid Number
		Invalid Number

Submodule: NumericRemover

Test Design: Equivalence Partitioning

Imports: text (string)

Exports: text in upper or lower case with numbers removed (string)

Removes any numeric values from a string then convert said string to upper or lower case after user selects the case converter.

Category	Test Data	Expected Result
User selects for text goes through upper case converter after numbers have been removed.	I want to watch Blade Runner 2049	I WANT TO WATCH BLADE RUNNER
User selects for text goes through lower case converter after numbers have been removed.	I have an i20N	i have an in
User does an invalid selection	My favourite number is 7	Invalid Selection

Submodule: Hours2Mins

Test Design: BVA

Imports: hours (integer)

Exports: mins (integer)

Calculates number of minutes in given hours.

Boundary	Test Data	Expected Result

Invalid/0	Hours = -1
	Hours = -10
		Invalid Number of Hours
		Invalid Number of Hours

0/Infinity	Hours = 5
	Hours = 2
		Minutes = 300
		Minutes = 120

Submodule: Mins2Seconds

Test Design: BVA

Imports: mins (integer)

Exports: secs (integer)

Calculates number of seconds in given minutes.

Boundary	Test Data	Expected Result
Invalid/0	Minutes = -1	Invalid Number of Minutes

0/Infinity	Minutes = 40
	Minutes = 75
		Seconds = 2400
		Seconds = 4500

Submodule: Mins2Hours

Test Design: Equivalence Partitioning

Imports: mins (integer)

Exports: hours (integer)

Calculates number of hours in given minutes.

Category	Test Data	Expected Result
Mins < 0	Mins = -1	Invalid Number of MInutes

Mins > 0	Mins = 180
	Mins = 320
		Hours = 3
		Hours = 5.33

Submodule: Secs2Mins

Test Design: Equivalence Partitioning

Imports: secs (integer)

Exports: mins (integer)

Calculates number of minutes in given secondss.

Category	Test Data	Expected Result
Secs < 0	Secs = -1000	Invalid Number of Seconds

Secs > 0	Secs = 1200
	Secs = 4030
		Minutes = 20
		Minutes = 67.17

Module: TimeConverter

Test Design: Equivalence Partitioning

Imports: convertchoice (string)

Exports: hours, mins, secs (integer)

Calculates units of time based on time converter chosen.

Category	Test Data	Expected Result
User selects Hours2Mins	Input = H	Hours2Mins

User selects Mins2Secs	Input = M	Mins2Secs
User selects Mins2Hours	Input = I	Mins2Hours
User selects Secs2Mins	Input = S	Secs2Mins
User does an Invalid Selection	Input = Z	Invalid Selection

Test Designs (White Box Testing)

This section of the report presents information on white box testing for 2 modules (Numeric Identifier and Numeric Validator).

Module: NumericIdentifer

Test Design: For-if/if not- else

Path	Test Data	Expected Result

Enter the for loop	Text = My last 4 student ID digits are 9112
	Text = I love ice cream
		Output: 'Numerics identified: [4,9,1,1,2]'
		Output: 'There are no numerics in your text'

Enter inner if part	Text = My last 4 student ID digits are 9112	Output: 'Numerics identified: [4,9,1,1,2]'
Enter outer if not part	Text = I love ice cream	Output: 'There are no numerics in your text'

Enter outer else part	Text = My last 4 student ID digits are 9112
	Text = I love ice cream
		Output: 'Numerics identified: [4,9,1,1,2]'
		Output: 'There are no numerics in your text'

Module: NumericValidator

Test Design: If-else

Path	Test Data	Expected Result
Enter the if part	Number = -1	Output: 'Invalid Number'
Enter the else part	Number = 7	Output: 'Your number, 7, is a valid number'

Test Implementation

Using python 3, I have successfully implemented the modules that run without errors. Results have been obtained into output files in the git repository.

To run the test code, simply do python3 Category1.py or Category2.py in the terminal command. Outputs will be stored in files C1_Outputs.txt and C2_Outputs.csv after the program have been fully ran. The outputted results are exactly like what have been outlined in the black and white box test designs.

Module Name	BB test design (EP)	BB test design (BVA)	WB test design	EP test code (implemented/ run)	BVA test code (implemented /run)	White-Box testing (implemented/run)
CaseConverter	Done	Not Done	Not Done	Implemented and Runs		
NumericIdentifier	Done	Not Done	Done	Implemented and Runs		Implemented and Runs
NumericValidator	Not Done	Done	Done		Implemented and Runs	Implemented and Runs
NumericRemover	Done	Not Done	Not Done	Implemented and Runs		
Hours2Mins	Not Done	Done	Not Done		Implemented and Runs	
Mins2Secs		Done	Not Done		Implemented and Runs	
Mins2Hours	Done	Not Done	Not Done	Implemented and Runs		
Secs2Mins	Done	Not Done	Not Done	Implemented and Runs		
TimeConverter()	Done	Not Done	Not Done	Implemented and Runs		

Ethics and Professionalism

The code I have designed can be used in a large software project that involves a chatbot. In this age, chatbots like alexa, siri and google home are capable of many different things such as converting units of time, identifying numbers in a sentence and much more. If my code were to be used in a chatbot, it could result in harmful effects due to the outputs being stored in a file. A worst case scenario is shown below:

Scenario: A user asks the chatbot to convert their name into upper or lower case and the chatbot would save the name in an output file. The user then tells the chatbot their last 4 student ID digits and the last 4 digits of their bank account password and it would as well be saved into an output file. Another person then hacks this chatbot, they steal the information the chatbot has saved about the user.

Who is harmed:

User can be harmed physically, financially and psychologically depending on what the hacker does with the information.

Chatbot company will have its reputation and finances harmed. Death threats or physical harm can be dealt to senior management which would harm those individuals physically, financially and psychologically.

Involvement of software: Poor chatbot design that could be intentional if company wanted to sell information it has gained on its users.

SE Mistakes: Lack of ethics and professionalism while developing product. Could also be poor auditing that did not pick up output file code because programmer could be the one who intentionally created the output file code to steal information about the users.

Conclusion

In conclusion, the implementation of the code for this assignment was handled very well as I did not have to refactor any pieces of code after going through the checklist and review. In addition, all test cases have been implemented and run without any errors. However, there was once where I had forgot to add a file before committing hence I had to make another commit called 'Forgot to git add a file commit'. In addition to this, I also had to redo the git repository a few hours before the due date because the repository got corrupted and I could not zip it. This has resulted in inaccurate times recorded in the git logs.