**CT4021 Introduction to Programming Fundamentals**

**Assignment 001**

Technical Report

for the Expense Management Program

**Student Name**: Nathaniel Clarke

**Student Number**: s1906563

**Course**: Computer and Cyber Security

**Submission Date**: Thursday, 16 January 2020

**University of Gloucestershire**

Contents

[Introduction 3](#_Toc29977684)

[Design 4](#_Toc29977685)

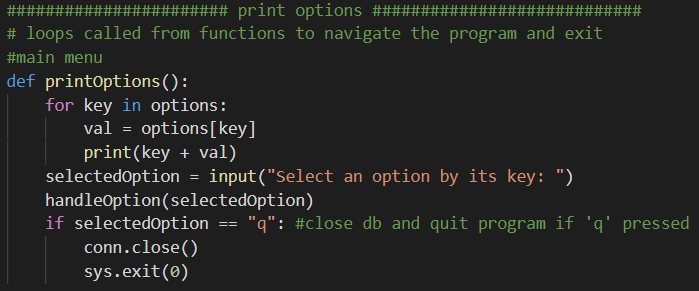
[Testing 5](#_Toc29977686)

[Appendix 2 - User Manual 6](#_Toc29977687)

[References 7](#_Toc29977688)

# Introduction

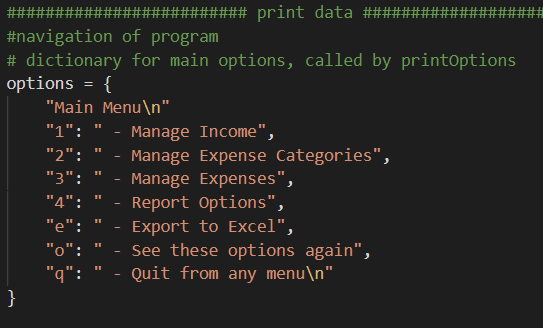
# Design

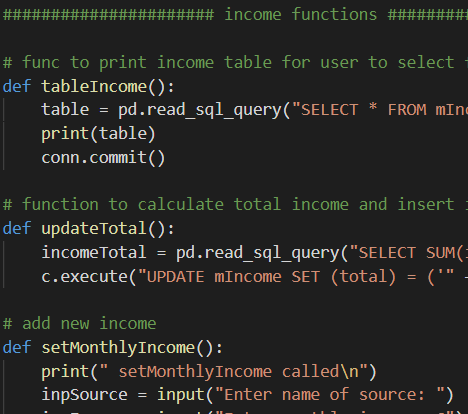
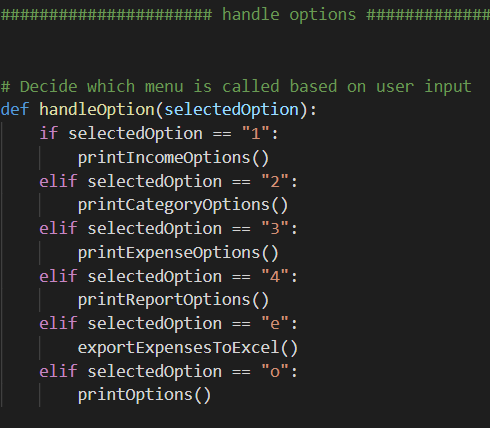
****

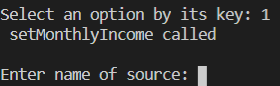
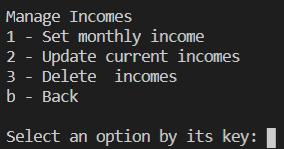
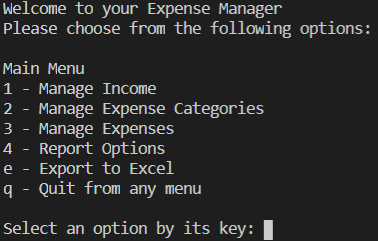
The software has been designed so that users can easily navigate the program and quickly access its functions.

The code has been built using dictionaries for the menu that get called by a corresponding function. When the program is first started the main menu function calls the options dictionary and, based on user input, the next print function is called. Functions that are in the same category or perform the same function are grouped together with title in a line of “#”s to distinguish. This keeps the program tidy and helps the programmer or others to easily understand what’s going on.

Every task the software must execute has been put into a function to be easily called upon. The handleOption functions near the top of the code decide what task is called based on the user input.

****





This is what all that code coming together looks like in the terminal

# Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ref.** | **Scope** | **Expected Results** | **Actual Results** | **Fixes** |
| 001 | Program start   1. Correct input 2. Incorrect input | Main menu brought up, input takes user to corresponding menu, incorrect input just loops to make input again. User can navigate through menus | As expected | N/A |
| 002 | Quit program from any menu   1. ‘q’ input in menu | Database closed, program ended | As expected | N/A |
| 003 | Set monthly income   1. Input source name 2. Input income | Values saved to dB, total income calculated and updated, user alerted/brought back to menu | As expected | N/A |
| 004 | Add new category   1. Input category name | Category saved to dB, user alerted/brought back to menu | As expected | N/A |
| 005 | Add new expense   1. Input expense name 2. Input category 3. Input cost 4. Input date | If category input matches existing category - add expense to dB and calculate then update over/under for expense and category  If category input invalid - user alerted, back to menu | To add over/under Numpy float could not be concatenated as integer | Dataframe value had to be converted to string first |
| 006 | Update income   1. Select source to change 2. Enter new values | If input matches existing - old values replaced with new ones in dB, user alerted  If input invalid – user alerted, back to menu | Error inserting into dB | Was trying to use INSERT with WHERE clause instead of UPDATE |
| 007 | Update category   1. Select category to change 2. Input new name | If input matches existing category – update category name in dB  If input invalid – user alerted, back to menu | As expected | N/A |
| 008 | Update expenses   1. Select expense to change 2. Input new name, category, cost | If expense and category input match existing – update values  If expense or category input invalid – user alerted, back to menu | Over/under calculation values undefined | Sql execute query had to be above calculation logic |
| 009 | Delete income, category, expense   1. In according function, select income, category or expense to delete | If income, category or expense input exists – delete target entry from dB table  If input invalid – user alerted, back to menu | Delete expense did not update catTotal | Selecting only cost of 1 expense, not whole category. Changed select query to get sum of all expenses in category |
| 010 | Export to excel   1. Input ‘e’ in main menu | All dB tables exported nicely into excel sheet, user alerted and taken back to menu | Data exported but tables overlapping | Start each table with a gap between last |
| 011 | Generate expense report of all expenses   1. Option 1 in report menu | Pdf report generated in reports folder with name ‘All Expenses’, user alerted and taken back to menu | Error finding directory | Removed ‘/’ before reports in pdf named |
| 012 | Generate expense report by date   1. Select view by range or day 2. If by range – input dates to find expenses between 3. By day – input 1 date | Report generated in reports folder with dates in title, all expenses between specified dates or from specified day shown in bar chart, user alerted and taken back to menu | As Expected | N/A |
| 013 | Generate expense report by category   1. Input category | Report generated in reports folder with category in title, all expenses under category shown in bar chart, user alerted and taken back to menu | As expected | N/A |
| 014 | Generate report of over/under for category   1. Input category 2. Input dates to view between | Report generated in reports folder with category in title for over/under, all expenses matching category and date parameters shown in bar chart | Only one expense being obtained from dB | Changed ‘fetchone’ to ‘fetchall’ in sql query |

# Conclusion

To improve the software next time I would:

1. Have better validation on the date inputs, can easily still be incorrect. Change it so it uses a date-time function to format it properly and split up the year, month and day
2. Create functions with different parameters rather than several functions that do slightly different things, more efficient
3. Add a line for over/under in the other reports instead of whole new report
4. Change colours of the different menus to more easily identify them and make the program look a bit nicer. One step further would be to create a GUI for the program.

# References

Check exists

<https://stackoverflow.com/questions/43246384/pythonsqlite3-if-statement-to-see-if-value-is-in-database>

SQLite db

<https://www.sqlitetutorial.net/sqlite-python/creating-database/>

<https://www.sqlitetutorial.net/sqlite-python/?fbclid=IwAR1D-QLVuo6-sUxtf8IaqLqEtTvOKNKsvfdq739s6dWdxcLm5OFvRWuUlp8>

Dictionary

<https://www.w3schools.com/python/python_dictionaries.asp>

Pandas print table

<https://stackoverflow.com/questions/37051516/printing-a-properly-formatted-sqlite-table-in-python>

XlsxWriter

<https://xlsxwriter.readthedocs.io/example_pandas_multiple.html>

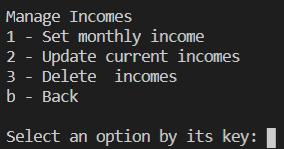
<https://stackoverflow.com/questions/32957441/putting-many-python-pandas-dataframes-to-one-excel-worksheet>

# Appendix 1 - User Manual

## 1 – Main Menu

This is what the user will see upon program start. From here they can navigate to sub-menus by inputting the number of the menu they wish to go to. A user can press ‘q’ from any menu to quit the program. ‘e’ will export all data currently stored in the software’s database to an excel spreadsheet, saved in the reports folder of the program directory. Every function will bring a user back to the main menu upon successful completion.

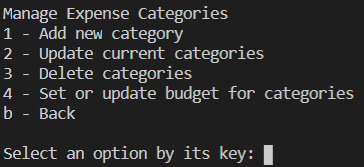
## 2 – Income functions

Once a user has navigated to the income many they have the option to set a new monthly income, update current incomes, delete old ones, or go back to the main menu. Adding a new income will ask the user for a source name followed by its income. It will then tell the user their new income has been saved.

Updating an income will display a table of current incomes to choose to update and ask the user for their input of which. If their input is an existing income then the program will ask for the new values. It will then save them and tell the user, just like adding a new one.

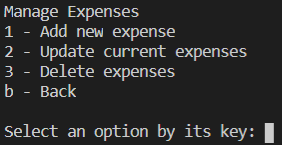
Deleting works in a similar way to updating in that a table of current incomes will appear for the user to choose from to delete. It will then ask their input and, provided the input is an existing income, delete the matching entry from the database.

## 3 – Category Functions

This is where users can manage the categories for expenses to be sorted into. It comes with the same add, update and delete functionalities as the income menu. Adding a new category will ask a user for a name and then a budget. These will be saved together in the database.

Updating and deleting work the same way as the other update/delete functions of the program – a table is displayed to choose which category to update or delete and a user will enter the desired category and it is deleted or continues to ask for a new category name. The update category and budget functions are separate because a user might update a budget frequently but a category itself is more likely to stay the same. Updating a budget also works in the same way.

## 4 – Expense Functions

This is where the user will come to add, update or delete their expenses. Adding a new expense will ask first for a name, then category, cost, and date. If the category entered exists then the new expense will be saved to the database with all those values: if not then the user will be told there is no such category and taken back to the expense menu