**Task 4b – Evaluation**

The solution that I have produced for Task 4a successfully meets a number of the requirements requested by RBSX Group, including what they expect the system to be capable of, as well as the requirements that are necessary for the end users.

**System Requirements**

The initial requirements requested by the program were that the program should allow users to convert between currencies based on up to date figures. In order to achieve this, I re-purposed the existing code provided and turned it into a module to be imported and called upon within my own program.

Text

Description automatically generated with medium confidence

This function, although brief, calls the program that has been created already and gives the users the option to enter their desired currency conversion. Within this program, additional changes were made in order to further fulfil these initial requirements – for example, (1) spelling errors prevented the program from reading the appropriate columns in the csv file, and (2) options to select USD – GBP and GBP – USD were not available.

A picture containing text

Description automatically generated(1)

(2)



These changes fulfilled the first bullet point to ‘allow users to input an amount in one currency and convert this to another currency based on most up to date figures.’

Text

Description automatically generated with low confidenceThe updated requirements requested that the program should identify patterns and trends for both USD – GBP and GBP – USD, which formulated the key part of my own program.

The code above produces a simple menu and associated dictionary that allows users to choose a conversion, and then reads the CSV file column with this heading. Below is an example of how the program then displays this information.

1. Graphical user interface, text

   Description automatically generated
2. Text

   Description automatically generated with medium confidence

In the first piece of code, the program calculates the average conversion rate for the last 4 months through a loop that adds the conversion rate on each row until it reaches the top, and then calculates the average after. The second piece of code also produces an average but this time for a given month.

In this example, it rounds the average to 2 decimal places as the given decimal places in the CSV file are too long and since currencies round to 2 decimal places normally, it made the most sense to output in this way.

**User Requirements**

The user requirements for the solution indicate that it must be easy to use, display information in a meaningful way, and make use of textual, numerical and graphical output in a way relevant to the end user, and throughout my program I have attempted to achieve this.

Firstly, in order to fulfil the requirement of being ‘easy to use’, I made sure that wherever a user input was used, the program would instead use an intuitive naming system to replace potentially long or complicated inputs.

1. Text

   Description automatically generated with medium confidence
2. Text

   Description automatically generated
3. 

In the above options, the inputs are altered so that they are easy to read and write – using numbers to select various options, or ‘y’ and ‘n’ instead of yes or no. This way the program is far easier to interact with for end users and improves their experience.

Secondly, I made sure that there was appropriate error handling throughout the program and options were available to make it as easy as possible for the user to understand what they had to do and how they could progress through the program.

1. 
2. A picture containing logo

   Description automatically generated
3. 

The above examples demonstrate how error handling is implemented in some instances, including an example of informative error messages that let the user know what was incorrect about their input. Where error message give little info, it is because input messages like (3) specify the types of inputs allowed.

Inputs (2) and (3) demonstrate how the implemented code was adjusted to improve error handling, removing the possibility for negative numbers, adding the additional information as new information was added to the displayed table.

Error handling was implemented across multiple inputs, to cover for:

* Option selection – selecting a number outside the available range or a random character that doesn’t correspond to a feature in the program.
* Negative numbers – when choosing the amount to convert, the value cannot be below zero.

The module of time was imported and is used a number of times to create delays after error messages are sent, so that the user has time to read and understand what was incorrect about their input. If there was no delay, the user may misplace their inputs and not realise what was wrong.

Other user accessibility options include (1) options to view information again, to prevent the need to return all the way to the start menu and (2) input messages that break apart information to stop output messages from displaying in blocks of text.

1.  

Input (1) mentioned here also uses the .upper() method so their input, y or n, is not case sensitive, improving user accessibility and avoiding further errors with user inputs.

All the information displayed to the user is in a textual format, with meaningful information displayed alongside that contextualises the values that are being displayed – currently averages for the last week, the last month, each month recorded and the last 4 months in total are given.

**Potential Adjustments**

There are a number of changes and alterations that could be made to the program that could further develop its usability. Firstly, from a functionality perspective, the program only displays information in a limited textual way. Other features/libraries or modules could be added to improve the way that this information is displayed, including more graphical formats such as through a graph.

Secondly, the program could be structured more efficiently as currently although it is separated into functions, there are few of them with many of the calculations taking place within one whole function – this could be separated further to make it more readable and improve the ability to adjust and alter the code in the future.

Thirdly, an attempt was made to add an additional option to the trend viewer, so that you can exit to the main menu from the view trends sub-menu – however, because a while loop was already used in order to introduce a feature that allowed users to return to the sub-menu, this proved to be difficult to introduce.

Fourthly, while all inputs feature error handling that prevent incorrect inputs, these error messages do not detail what was wrong with the original input – further development may include specifying what was wrong with the input, or mentioning the requirements for the input again. However, whilst this does not appear, every input message does include in brackets the valid inputs.

Other changes may also include format changes, to make every aspect of the program more in line with each other, such as making sure the trend selector appears in a similar format to the menu layouts used in the implemented code.

A picture containing table

Description automatically generated(1, my created menu)

(2, implemented code menu)

Text, letter

Description automatically generated

Commonality with the use of hashtags surrounding the menus, numbering systems etc – however menu (1) is capitalised to improve visibility, includes a label at the top so the user knows which menu they are viewing. These changes in menu (1) could be carried over to menu (2).