Task 3 Evidence

This task requires me to design an application feature using Python, Pandas, and a supplied data file in .csv format.

The data file has data for 4 weeks and defines data from 4 different income sources. The task requires me to define and design the processes to be used to get the sum of income for a named income source.

I will use a 'top-down' decomposition to design the solution elements. Using this approach allows the implementation of code structures that are smaller and easier to manage. Each code unit can by developed by a range of developers with different skillsets, and can be tested, developed and maintained without needing to change other code structures as each should do 1 job only, and do it robustly and well.

As an overview, the single task has these steps:

1. Launch the application.
2. Show a main menu.
3. Get user choice (here it must be '1'.)
4. Show a menu of valid income sources
5. Get the income source required
6. Load the data file to the data for the income source
7. Process the data to calculate the sum of the income source takings
8. Return the sum of the income to the user as a terminal massage

It is likely that more application features will be required after this task, and these are represented in Figure 1 below.

In this document I have introduced some pseudocode terms not included in the specification, such as PRINT, TRY, EXCEPT to help with clarity.

'PRINT' is a fork from SEND

'TRY' and 'EXCEPT' are language specific in error trapping and handling.

Paul McKillop LL-12349876
Isle of Wight College

APPLICATION OVERVIEW

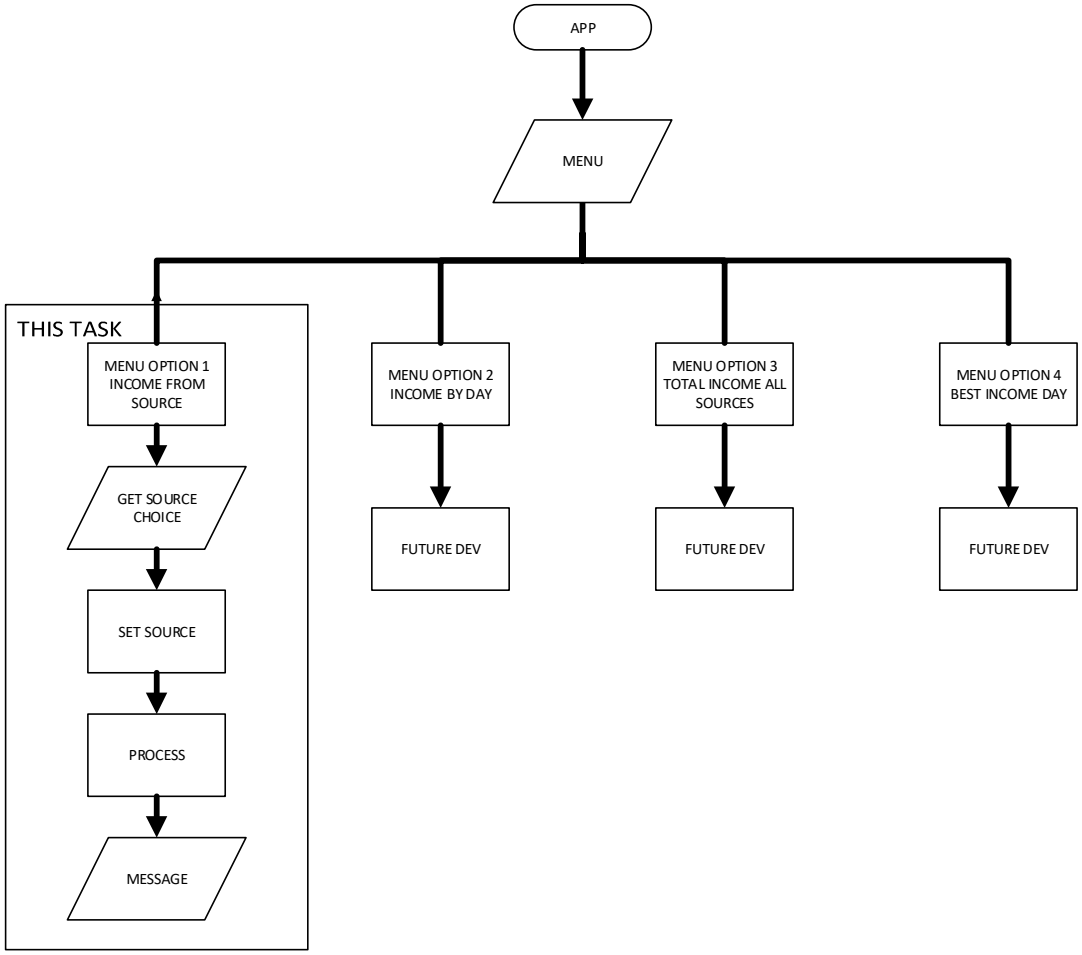


Figure 1 Application overview

Paul McKillop LL-12349876
Isle of Wight College

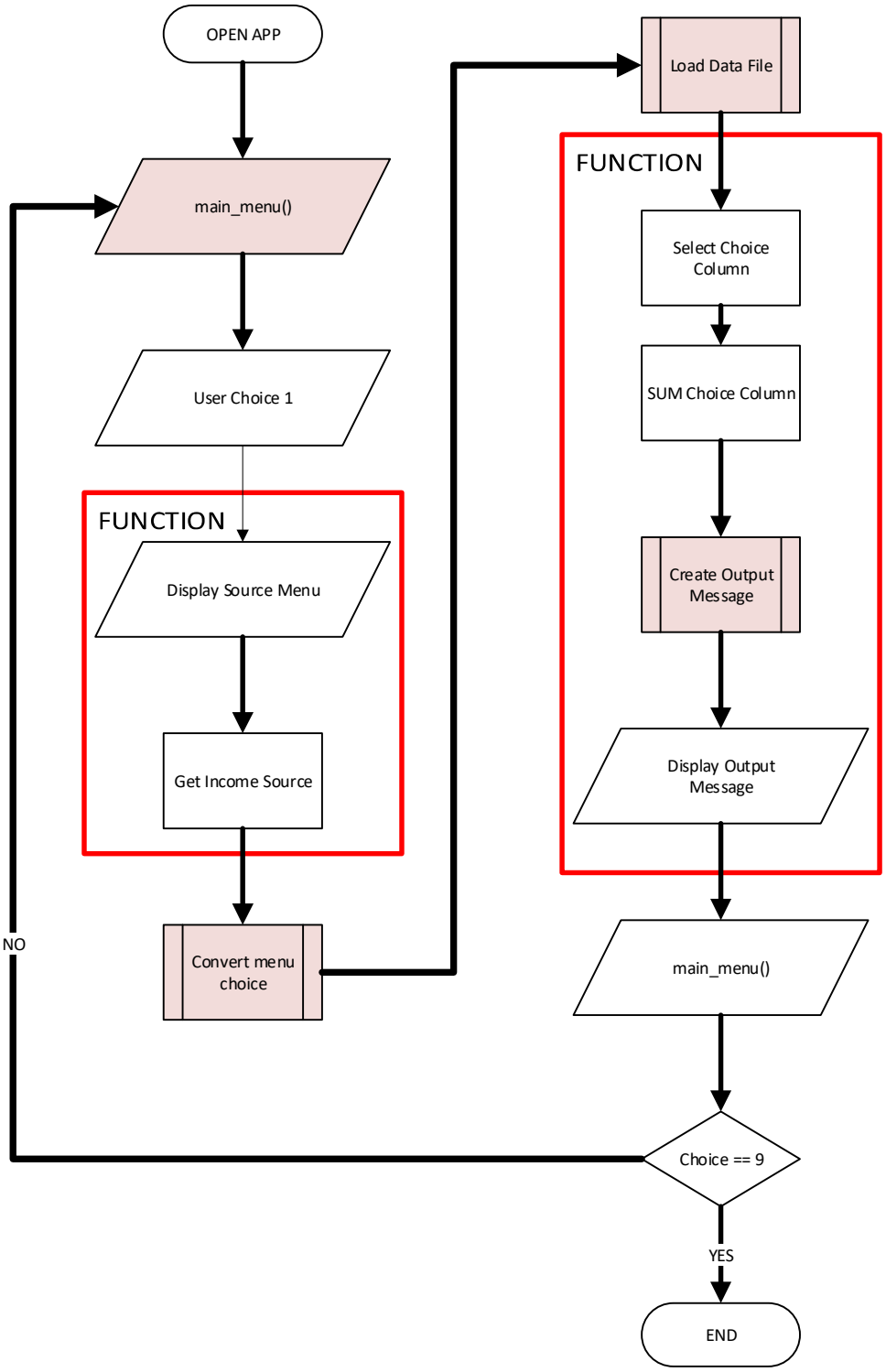
The next section of the document will define the algorithms to be implemented to complete the Menu Option 1 requirements only.

I will use either a flowchart or a pseudocode definition for the algorithms. In some cases, it is easier to use one style, rather than the other. For example, I find it easier to use pseudocode for code that uses a selection construct with multiple branching options. This is because flowcharts can be complicated and difficult to trace through in these cases.

One the other hand, there are advantages in showing an algorithm as a flowchart. The human brain has a well-developed capacity to interpret and understand large amounts of visual, graphical information quickly if it is not too complicated.

Both algorithm design methods should also allow non-technical readers to understand how the algorithm will work, even if they do not know how to implement it as software.

In Figure 2 below, I describe the Menu Option 1 processes in more detail and start to identify options for decomposition.



MODULARISATION

Figure 2 Top-down decomposition stage 1

Here I define the algorithms:

Much of the code is defined in functions. These will only run when they are called in the code. The application needs some code to start with.

```
1   WHILE True DO
2       SET choice = main_menu()
3       SEND message to say which choice has been made
4
5       IF choice == 1 THEN
6           convert_menu_choice()
7           process_total_source_income(choice)
8       ELIF choice == 2 THEN
9           SEND message 'Not yet implemented'
10      ELIF choice == 3 THEN
11          SEND message 'Not yet implemented'
12      ELIF choice == 4 THEN
13          SEND message 'Not yet implement
14      ELIF choice == 9 THEN
15          SEND message 'You have chosen to exit
16          SEND message 'Goodbye'
17
18          break
19      ELSE SEND message 'Invalid choice. Try again'
20  END WHILE
21  END
22
23
24
25
```

Figure 3 The application menu loop

Figure 3 above shows the process that will run when the application is launched. The process will loop until a valid choice is made, or the user elects to exit the application by choosing option 9.

You will see that the first instruction, on line 2 above launches the main_menu.

It then checks which menu choice is made through a selection construct using IF .. ELIF.. ELIF.. ELSE.

Here, if option 1 has been chosen, The convert_menu_choice() function will run. Then the process_total_source_income() function will use its return value to process the data.

Here, the main menu is shown.

```
PROCEDURE main_menu()
BEGIN PROCEDURE

    SET flag = True

    WHILE flag DO
        SEND Menu Header to terminal
        SEND Menu Options to terminal

        SET choice = INPUT("Enter your number choice")

        TRY int(choice)

        EXCEPT
            SEND error message invalid choice to terminal
            SET flag = True

        ELSE
            SEND choice Success message to terminal
            SET flag = False

        RETURN choice

    END WHILE

END PROCEDURE
```

Figure 4 The main_menu() function

Figure 4 above shows how the main menu will show the user can select a top-level option. For this task, there is only one option for processing the data. It would be in option 1. I have also said that there should be a menu option that allows the user to exit the application gracefully.

When option 1 has been selected, we have to show users which income sources there are, and they have to choose one.

```
PROCEDURE source_menu()
BEGIN PROCEDURE

    SET flag = True

    WHILE flag DO
        PRINT Header section
        PRINT '1. Tickets'
        PRINT '2. Gift Shop'
        PRINT '3. Snack Stand'
        PRINT '4. Pictures'
        SET choice = INPUT('ENTER your choice as a number')

        TRY int(choice)
        EXCEPT:
            PRINT Error message
            SET flag = True
        ELSE:
            PRINT Success message
            SET flag = False

    END WHILE

    RETURN choice
END PROCEDURE
```

Figure 5 Source selection menu

The function in Figure 5 above will return the number chosen by the user for the income source they want to sum. This has to be converted to a string value in another function shown in Figure 6 below for the next stage of processing.

Processing for the data needs us the reference the actual column name of the income source, so the number chosen by the user is converted to a text value that has the exact text name of the column. This is the value passed to the algorithm that calculates the sum of the income for that source.

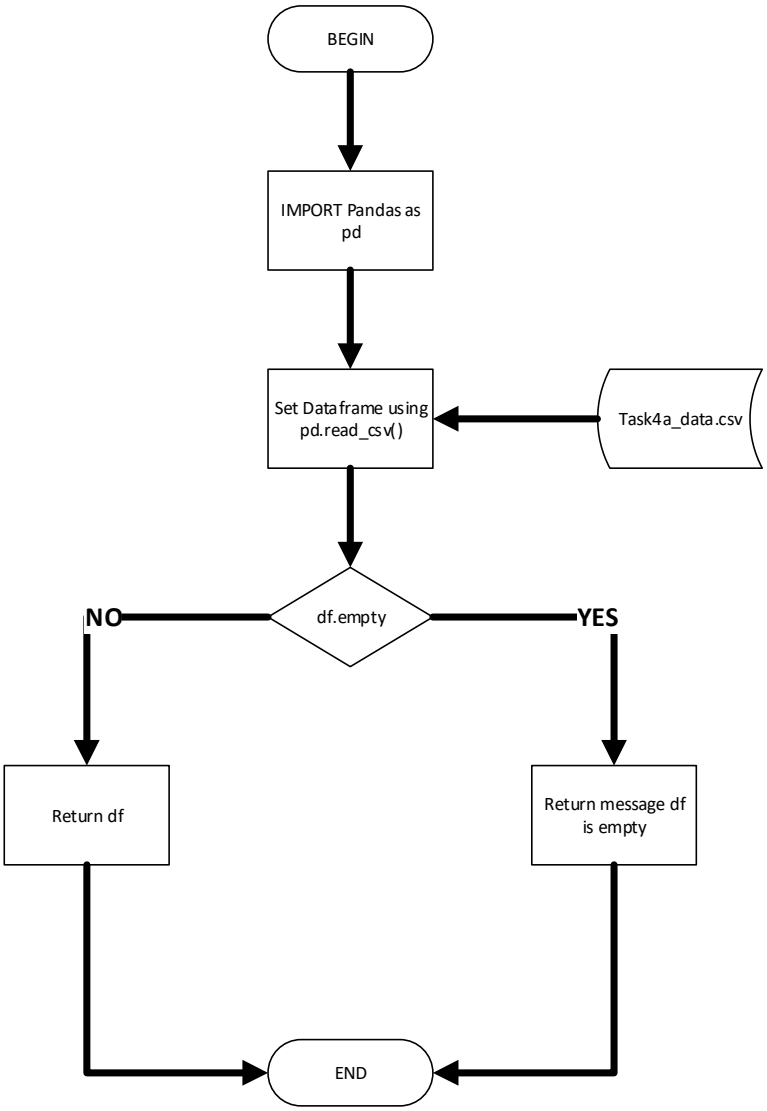
```
PROCEDURE convert_menu_choice(menu_choice)
BEGIN PROCEDURE
    IF menu_choice == 1 THEN
        SET source_choice = "Tickets"
    ELIF menu_choice = 2 THEN
        SET source_choice = "Gift Shop"
    ELIF menu_choice == 3 THEN
        SET source_choice = "Smack Stand"
    ELIF menu_choice == 4 THEN
        SET source_choice = "Pictures"
    END IF

    RETURN source_choice

END PROCEDURE
```

Figure 6 Convert menu number to text

Figure 6 above shows the conversion. It receives the income source number chosen as an argument and then converts that to a valid column name. It should be noted that this code does not have any error trapping or handling features.

We now have the value we need to return the data required from Menu Option 1, the sum of the income for the source which is now available as a string.

First, we use the Pandas library to load the data into RAM.



Function load_data()

Figure 7 The load_data algorithm

Figure 6 above shows an effective use of modularization. The application may require that the data is loaded in different places, at several stages of the application's use. By modularizing it, we can rely on the process being correct as it includes error trapping. It would be implemented in Python as follows:

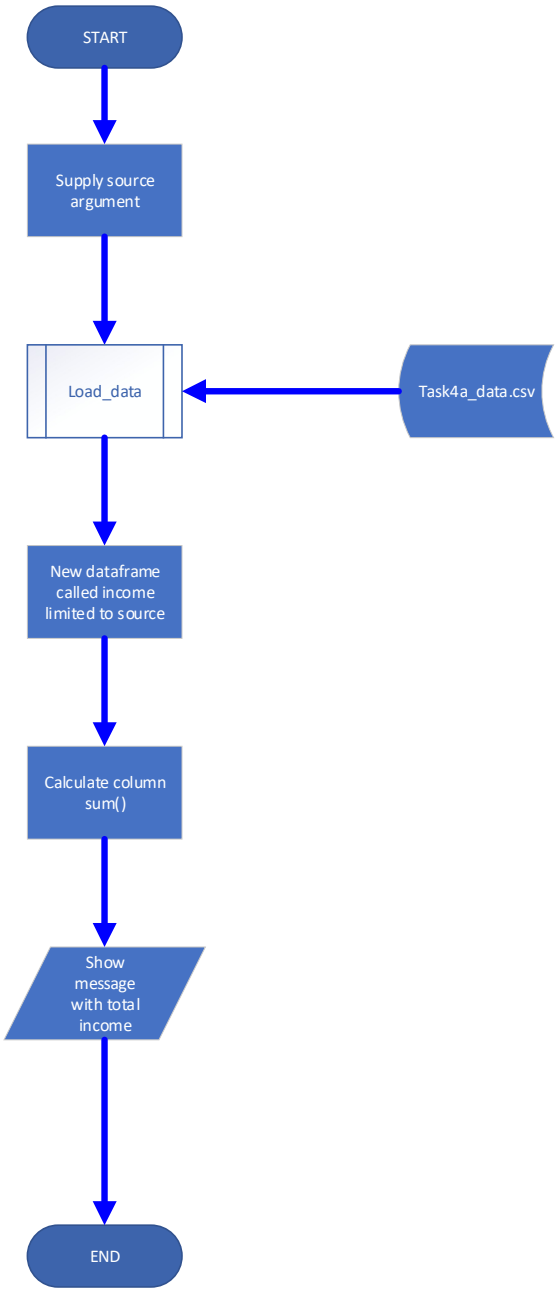
```
# Load the data file to RAM
def load_data():
    df = pd.read_csv("Task4a_data.csv")
    if not df.empty:
        return df
    else:
        print("No data found. Nothing loaded")
```

Figure 8 The Python code.

Figure 7 shows that the Pandas library alias ‘pd’ is being used to access its features.

The data to be analysed is now in RAM and is available to the application for processing.

Having got an income source choice from the user, we can process it:



Process Total Source Income

Figure 9 Processing the data when choice is defined

Figure 8 above shows that the load-data function defined above in Figure 7 is called to get the income data into RAM. The symbol used represents a ‘stored procedure’. The symbol to the side is for a data source. In this case, it’s a .csv file.

We load this data into a dataframe named ‘df’. Best practice says that we should now leave that intact and create a new dataframe if we weant to manipulate it.

We create a new dataframe with just the data we need calling it ‘income’.

We sum the income values in the income source column.

We can then tell the user what the total income for the selected source is.

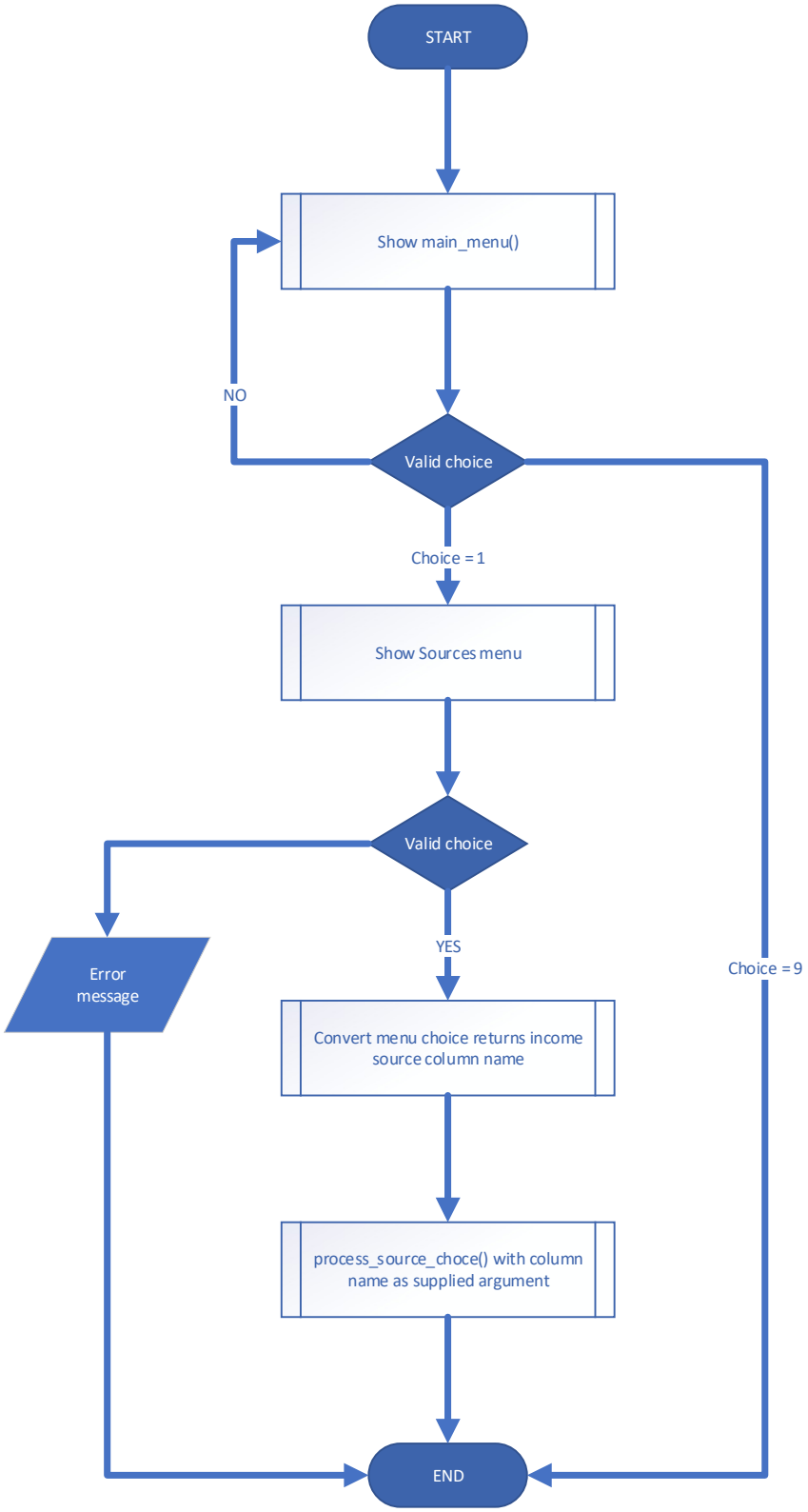
It could be useful to have a ‘standard’ message function that we can reuse in other parts of the application.

This is another decomposition and modularization process. This is a very simple structure, but by doing it this way, you remove a lot of risks of errors even if they are just mis-spellings. Here is the pseudocode in Figure 10 below.

```
1  PROCEDURE income_message(label, value)
2  BEGIN PROCEDURE
3      SET message = "The toal for {} was £{}".format(label,value)
4      RETURN message
```

Figure 10 Standard message function

The whole menu option is represented in this flowchart:



Menu Option 1 Processing