ALGORITHM & PROGRAMMING

FINAL PROJECT REPORT

Lecturer: Jude Joseph Lamug Martinez D4017

Kimberly Mazel
Class L1BC
2502022250

Faculty of Computing and Media
Computer Science Major
Binus International University
2022

Table of Contents

Project Specifications **2**

Objective 2

Concept 2

Solution Design **3**

Driver.py 3

IncomeStatement.py and SOFP.py 5

SalaryPredict.py 16

Evidence of working program **18**

Summary display of each section 18

Calculated Conclusions 20

Graphs 21

CSV Files 21

Reset Program 23

Validations 23

1. Project Specifications

1.1. Objective

Accounting is part of everyone’s daily life, regardless of their lifestyle or career. The most fundamental statements are the Income Statement and Statement of Financial Position. Although they are relatively simple, creating and formatting them remains a tedious task. This project aims to generate and format the statements for the user based on their input, as well as predict their salary based on their years of experience through machine learning. It is a simple personal financing program that prioritizes the ease of use and convenience for the user.

1.2. Concept

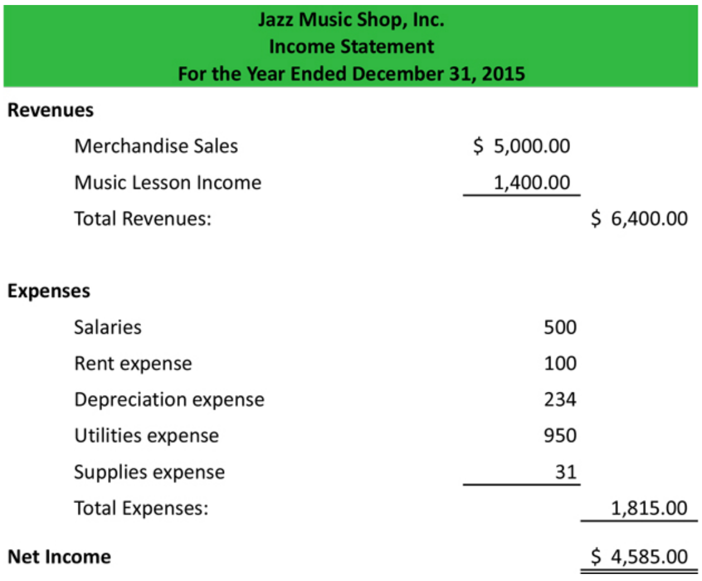
The user will be asked to choose between 3 options that the program will run:

A. Creating an Income Statement
B. Creating a Statement of Financial Position
C. Salary Prediction

Once the user decides, the program will run the respective code.

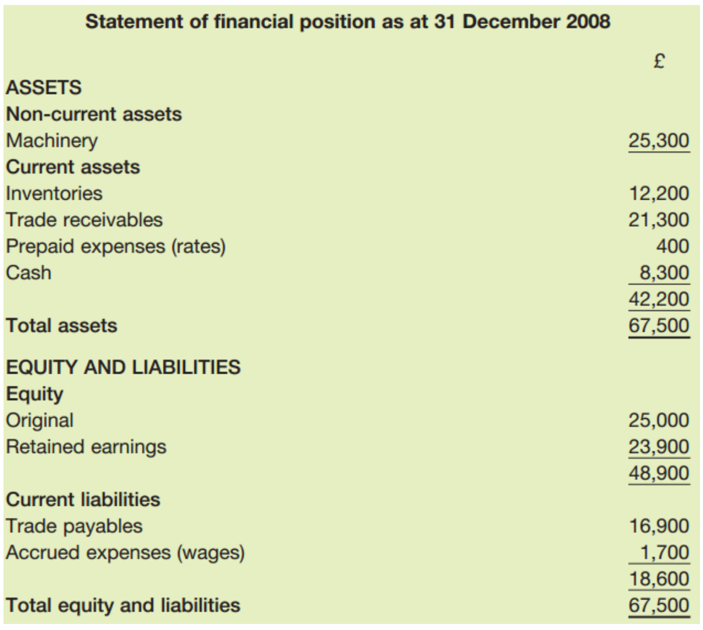
The code for the Income Statement and the Statement of Financial Position have a similar concept. The program will ask the user to input the name and value for each section (Revenue and Expenses for Income Statement, Assets, Equity and Liabilities for Statement of Financial Position). It will then calculate the totals of each section and display the conclusion for the chosen statement (Profit/Loss for Income Statement and Balanced/Unbalanced for Statement of Financial Position). Additionally, the program offers the option to export the statement in CSV file format, providing the user with their formatted statement (see Images 1.1 and 1.2 on the next page). Exclusive to the Income Statement option, the program allows the user to display the difference of the inputted revenue and expenses in a bar chart. This is not available for the Statement of Financial Position as the sections are meant to be equal (balanced).

If the user selects the Salary Prediction option, the program will ask the user for their years of experience. The code relies on machine learning based on a dataset from Kaggle.com that consists of the average salaries based on their years of experience. The program will compare the inputted years to the dataset through linear regression and output the predicted salary in US dollars.



Jazz Music Shop, Inc. Income Statement For the Year Ended December 31, 2015		
Revenues		
Merchandise Sales	$ 5,000.00	
Music Lesson Income	1,400.00	
Total Revenues:		$ 6,400.00
Expenses		
Salaries	500	
Rent expense	100	
Depreciation expense	234	
Utilities expense	950	
Supplies expense	31	
Total Expenses:		1,815.00
Net Income		$ 4,585.00

Image 1.1 Income Statement



Statement of financial position as at 31 December 2008	£
ASSETS	
Non-current assets	
Machinery	25,300
Current assets	
Inventories	12,200
Trade receivables	21,300
Prepaid expenses (rates)	400
Cash	8,300
	42,200
Total assets	67,500
EQUITY AND LIABILITIES	
Equity	
Original	25,000
Retained earnings	23,900
	48,900
Current liabilities	
Trade payables	16,900
Accrued expenses (wages)	1,700
	18,600
Total equity and liabilities	67,500

Image 1.2 Statement of Financial Position

2. Solution Design

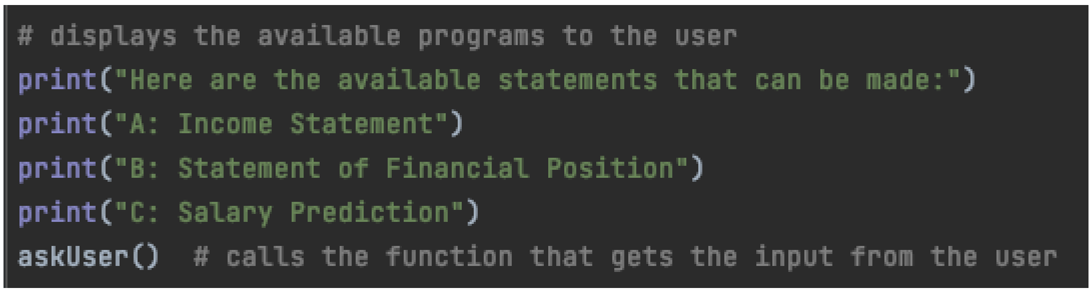
There are 7 files in the program, 4 of those being codes. The codes consist of:

a. Driver.py - The code that will be run. Asks the user what they would like to run and calls the class from the respective code.
b. IncomeStatement.py - Accepts input for revenue and expenses, calculates the profit or loss of the user and outputs the formatted Income Statement.
c. SOFP.py - Accepts input for assets (current and non-current), equity and liabilities. Calculates the total for assets and total for equity and liabilities to output the formatted Statement of Financial Position.
d. PredictSalary.py - Accepts years of experience input from the user and displays the predicted salary based on the dataset.

The remaining 3 files are supporting CSV files needed for the code to run:

a. IncomeCSV.csv - CSV file where the Income Statement will be written to.
b. SOFPCSV.csv - CSV file where the Statement of Financial Position will be written to.
c. Salary_Data.csv - Dataset from Kaggle.com consisting of the years of experience and respective average salary.

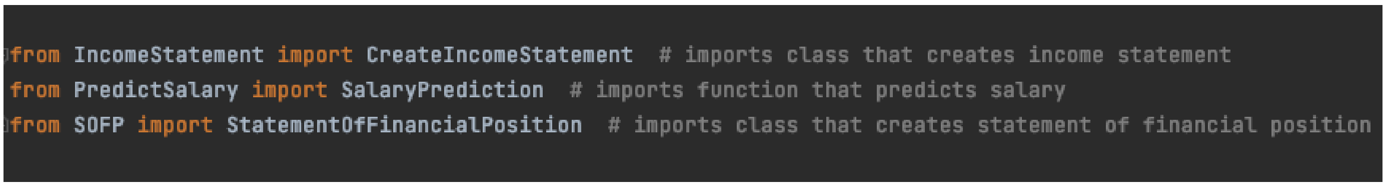
2.1. Driver.py



```python
# displays the available programs to the user
print("Here are the available statements that can be made:")
print("A: Income Statement")
print("B: Statement of Financial Position")
print("C: Salary Prediction")
askUser()  # calls the function that gets the input from the user
```

Image 2.1 Snippet of Driver.py code

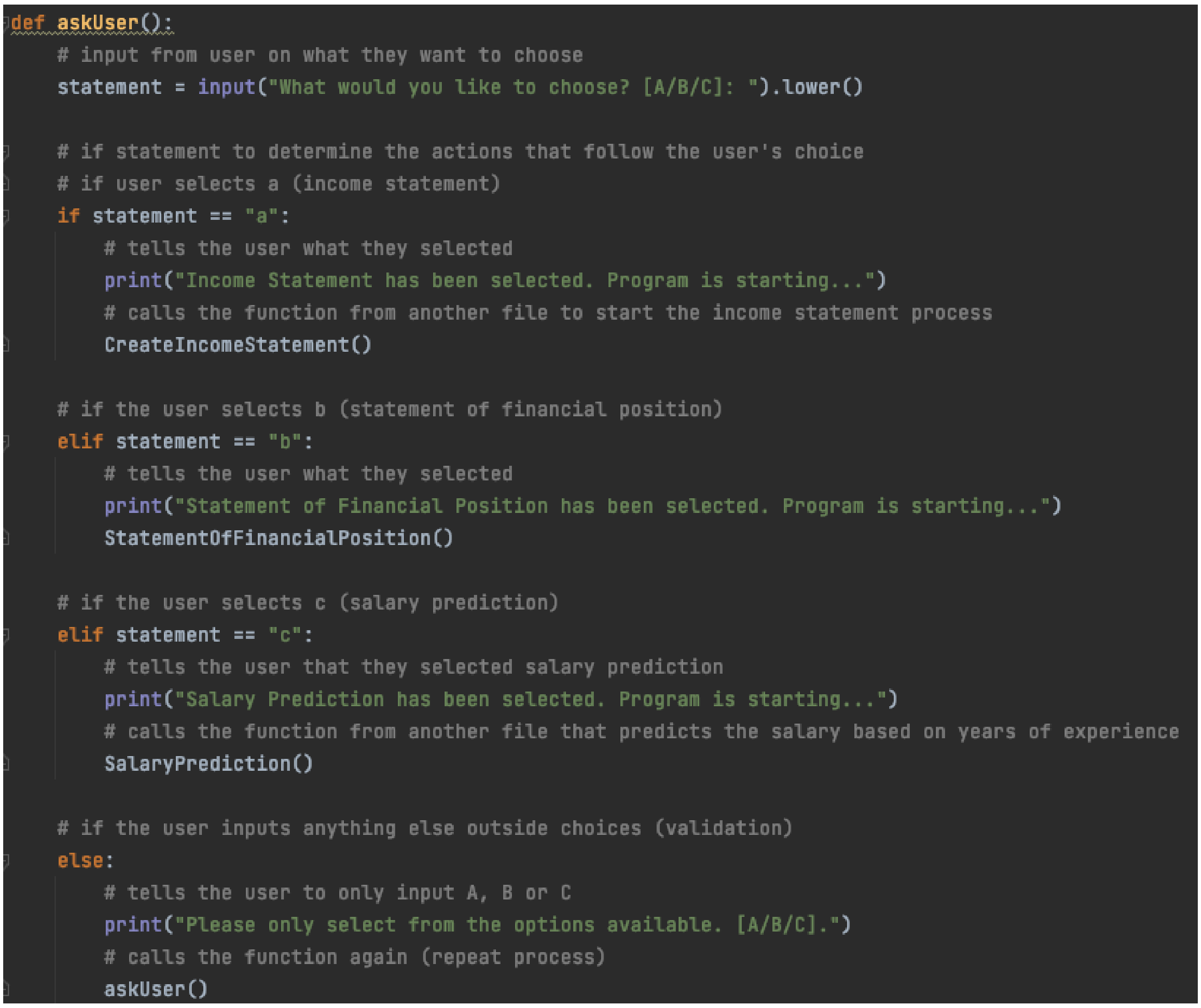
Image 2.1 shows the code that will display the options to the user. It then calls askUser(), a custom-built function that will run the next code depending on the user's input.



```python
from IncomeStatement import CreateIncomeStatement  # imports class that creates income statement
from PredictSalary import SalaryPrediction  # imports function that predicts salary
from SOFP import StatementOfFinancialPosition  # imports class that creates statement of financial position
```

Image 2.2 Import lines of Driver.py

Image 2.2 are the first 3 lines of the Driver.py file that imports the classes and functions from the other 3 files in order to call them based on the user's chosen input.



```python
def askUser():
    # input from user on what they want to choose
    statement = input("What would you like to choose? [A/B/C]: ").lower()

    # if statement to determine the actions that follow the user's choice
    # if user selects a (income statement)
    if statement == "a":
        # tells the user what they selected
        print("Income Statement has been selected. Program is starting...")
        # calls the function from another file to start the income statement process
        CreateIncomeStatement()

    # if the user selects b (statement of financial position)
    elif statement == "b":
        # tells the user what they selected
        print("Statement of Financial Position has been selected. Program is starting...")
        StatementOfFinancialPosition()

    # if the user selects c (salary prediction)
    elif statement == "c":
        # tells the user that they selected salary prediction
        print("Salary Prediction has been selected. Program is starting...")
        # calls the function from another file that predicts the salary based on years of experience
        SalaryPrediction()

    # if the user inputs anything else outside choices (validation)
    else:
        # tells the user to only input A, B or C
        print("Please only select from the options available. [A/B/C].")
        # calls the function again (repeat process)
        askUser()
```

Image 2.3 askUser() function from Driver.py

Image 2.3 shows the askUser() function that was previously mentioned. It consists of if-else statements that call the class or function from the respective file, depending on the user's input. The function is able to call the classes from the other files due to the import lines in Image 2.2. It prints a statement informing the user which option they have chosen and proceeds to run the code.

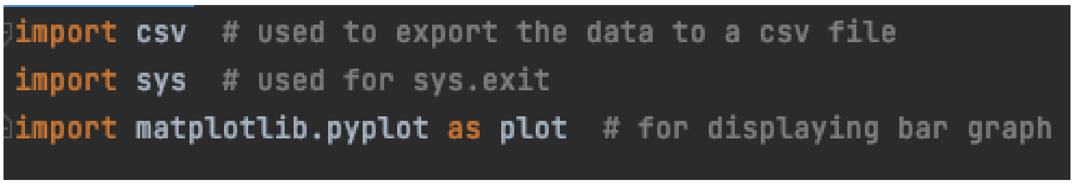
In the askUser() function, it starts off with the variable statement that prompts the user for their chosen option. The .lower() at the end of the line is a built-in function that returns any string into all lowercase letters. This is to ensure that the code will run regardless of whether the user enters “A” or “a” as the function will return it as a lowercase letter before it is used in the if-else statements.

There are 4 pathways that the code provides through the if-else statement:

1. The first if statement is for the A option, which is the income statement. The == operator compares the statement variable and the string “a”. If the user had inputted “a” for their option, choosing the Income Statement, the operator would be True, thus the lines in that if statement would be run. The program will print “Income Statement has been chosen. The program is starting…” so that the user will be aware of the choice that they made. It will then call the class CreateIncomeStatement from IncomeStatement.py.
2. The next elif statement is for the B option, which is the statement of financial position. If the statement variable does not fit the first if statement, it will move on to the next. For this statement, it compares the input with “b”. If it is True, the code will inform the user of their choice and call the class StatementOfFinancialPosition from the SOFP.py file. If it is False, it will be passed on to the next line.
3. Similarly to the last elif statement, it compares the input to “c”, and if it is True, it will once again inform the user of their choice and call the function SalaryPrediction() from the PredictSalary.py file.
4. The last else statement is for any input that is not A, B, or C. Since there are only 3 options that the program provides, it should not be possible for the user to enter any other options. This else statement acts as a validation to ensure that. If the user enters invalid input, for example, “D” or 3, it will print “Please only select from the options available. [A/B/C],” to inform the user that their input was invalid. It then calls the askUser() function that runs the code again and asks the user for their choice again.

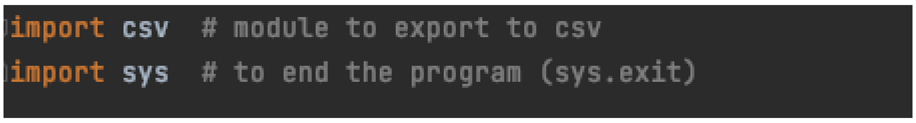
2.2. IncomeStatement.py and SOFP.py

As the codes for IncomeStatement.py and SOFP. py are similar, this section of the report will break down the similar functions for both files.



```
import csv  # used to export the data to a csv file
import sys  # used for sys.exit
import matplotlib.pyplot as plot  # for displaying bar graph
```

Image 2.4 Modules used for IncomeStatement.py

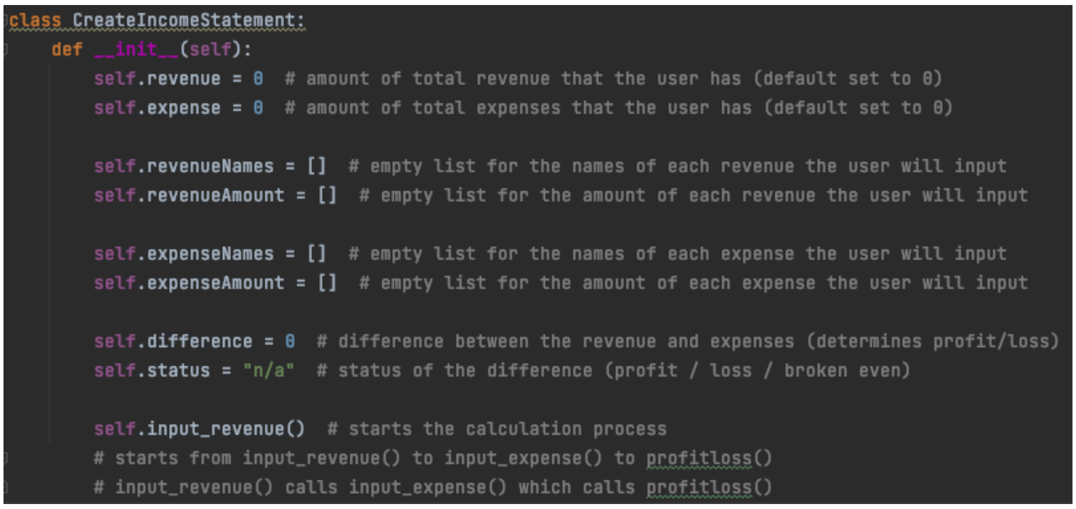


```
import csv  # module to export to csv
import sys  # to end the program (sys.exit)
```

Image 2.5 Modules used for SOFP.py

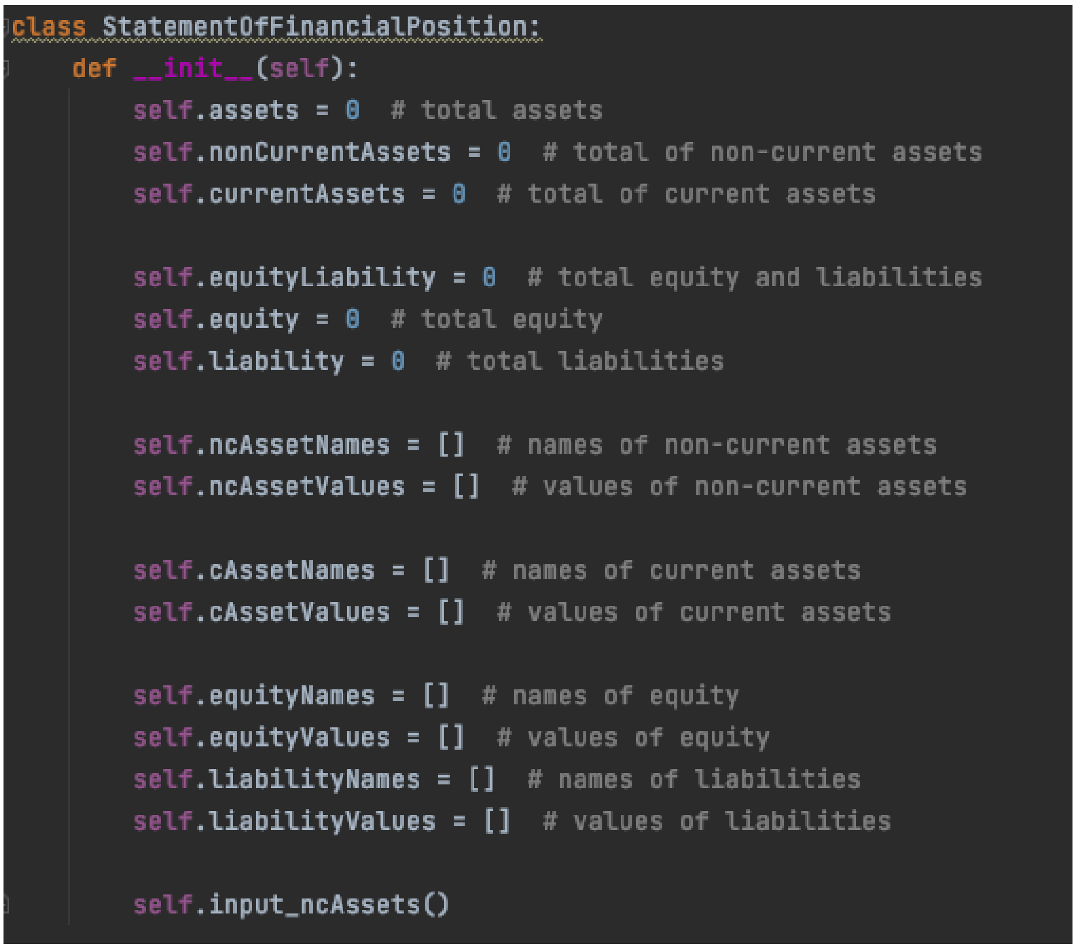
The modules used for the files consist of:

a. csv - Used for reading and writing CSV files
b. sys - Module to manipulate the virtual environment (used to exit the code through sys.exit)
c. matplotlib - Module to display data through graphs (used to produce bar charts)



```python
class CreateIncomeStatement:
    def __init__(self):
        self.revenue = 0  # amount of total revenue that the user has (default set to 0)
        self.expense = 0  # amount of total expenses that the user has (default set to 0)

        self.revenueNames = []  # empty list for the names of each revenue the user will input
        self.revenueAmount = []  # empty list for the amount of each revenue the user will input

        self.expenseNames = []  # empty list for the names of each expense the user will input
        self.expenseAmount = []  # empty list for the amount of each expense the user will input

        self.difference = 0  # difference between the revenue and expenses (determines profit/loss)
        self.status = "n/a"  # status of the difference (profit / loss / broken even)

        self.input_revenue()  # starts the calculation process
        # starts from input_revenue() to input_expense() to profitloss()
        # input_revenue() calls input_expense() which calls profitloss()
```

Image 2.6 CreateIncomeStatement class and __init__ function



```python
class StatementOfFinancialPosition:
    def __init__(self):
        self.assets = 0  # total assets
        self.nonCurrentAssets = 0  # total of non-current assets
        self.currentAssets = 0  # total of current assets

        self.equityLiability = 0  # total equity and liabilities
        self.equity = 0  # total equity
        self.liability = 0  # total liabilities

        self.ncAssetNames = []  # names of non-current assets
        self.ncAssetValues = []  # values of non-current assets

        self.cAssetNames = []  # names of current assets
        self.cAssetValues = []  # values of current assets

        self.equityNames = []  # names of equity
        self.equityValues = []  # values of equity
        self.liabilityNames = []  # names of liabilities
        self.liabilityValues = []  # values of liabilities

        self.input_ncAssets()
```

Image 2.7 StatementOfFinancialPosition class and __init__ function

As shown in Images 2.6 and 2.7, both files have similar contents for the __init__ function. First, variables for the total of each section are set to 0 by default. These sections consist of:

Income Statement:
- Revenue (self.revenue for total of revenue inputted)
- Expense (self.expense for total of expense inputted)

Statement of Financial Position:

- Assets (self.assets for total of assets inputted, both current and non-current)
    - Non-current assets (self.nonCurrentAssets for total of non-current assets)
    - Current assets (self.currentAssets for total of current assets only)
- Equity and Liabilities (self.equityLiability for total of both)
    - Equity (self.equity for total of equity alone)
    - Liability (self.liability for total of liability alone)

Next, there are two lists for each section, one for the name and one for the value of each input. For example, Wages is stored in the name list and 1000 is stored in the value list. They are stored in order of the input so the index for each respective name and value are the same. Wages will have the 0 index and 1000 will have the 0 index as well.

Example lists:
self.revenueNames = [“Wages”, “Salary”, “Sales”, ‘Income”]
self.revenueAmount = [1000, 200, 350, 400]

Here are the following list variables from their respective files:

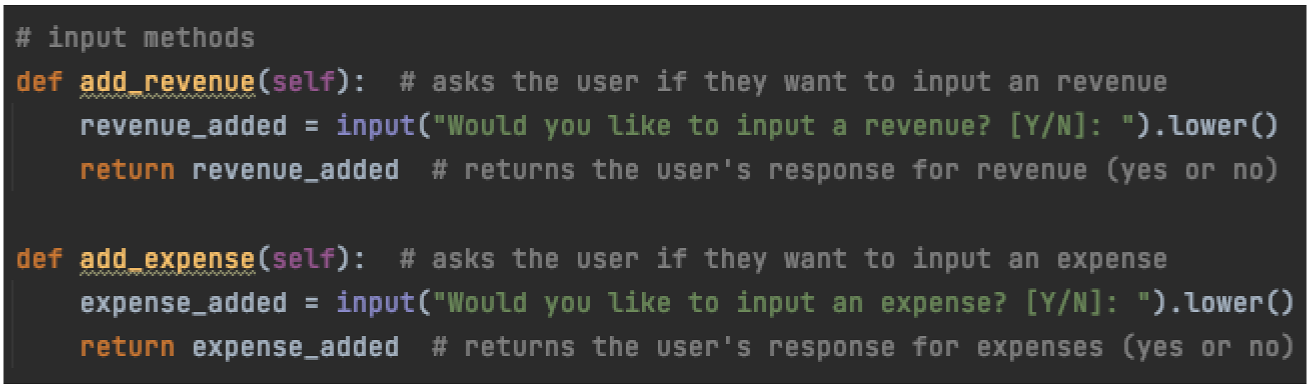
Income Statement:

- Revenue
    - self.revenueNames (list that stores each name of the inputted revenue)
    - self.revenueAmount (list that stores each value of the inputted revenue)
- Expense
    - self.expenseNames (list that stores each name of the inputted expense)
    - self.expenseAmount (list that stores each value of the inputted expense)

Statement of Financial Position

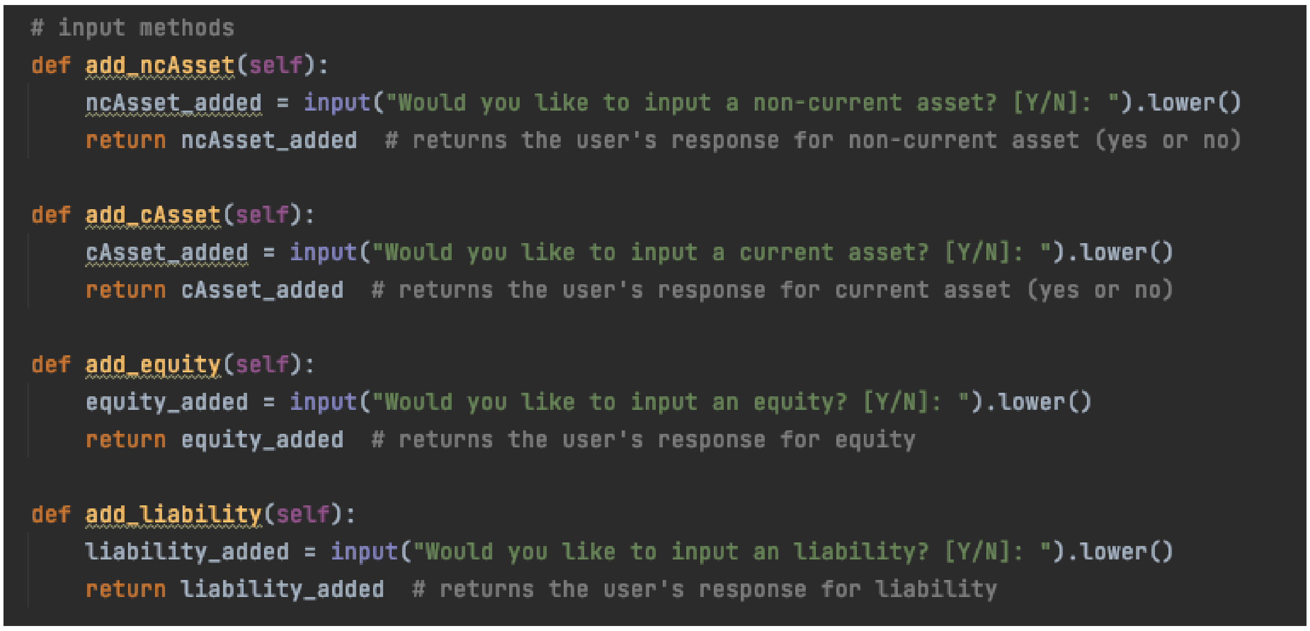
- Non-current assets
    - self.ncAssetNames (list that stores each name of the non-current assets)
    - self.ncAssetValues (list that stores each value of the non-current assets)
- Current assets
    - self.cAssetNames (list that stores each name of the current assets)
    - self.cAssetValues (list that stores each value of the current assets)
- Equity
    - self.equityNames (list that stores each name of the inputted equity)
    - self.equityValues (list that stores each value of the inputted equity)
- Liability
    - self.liabilityNames (list that stores each name of the inputted liability)
    - self.liabilityValues (list that stores each value of the inputted liability)

Next are functions that ask the user if they would like to input something for each section. The functions are similar except that the variables are changed according to each section. For example, the input text for revenue is “Would you like to input a revenue? [Y/N]: “ while the input text for the expense is “Would you like to input an expense? [Y/N]: “. The logic behind the functions is the same, as they all return the user’s input which is “Y” or “N” (or the lowercase versions of them). Similarly to the Driver.py file, .lower() is used to ensure that both uppercase and lowercase letters are accepted.



```python
# input methods
def add_revenue(self):  # asks the user if they want to input an revenue
    revenue_added = input("Would you like to input a revenue? [Y/N]: ").lower()
    return revenue_added  # returns the user's response for revenue (yes or no)

def add_expense(self):  # asks the user if they want to input an expense
    expense_added = input("Would you like to input an expense? [Y/N]: ").lower()
    return expense_added  # returns the user's response for expenses (yes or no)
```

Image 2.8 Input methods from IncomeStatement.py

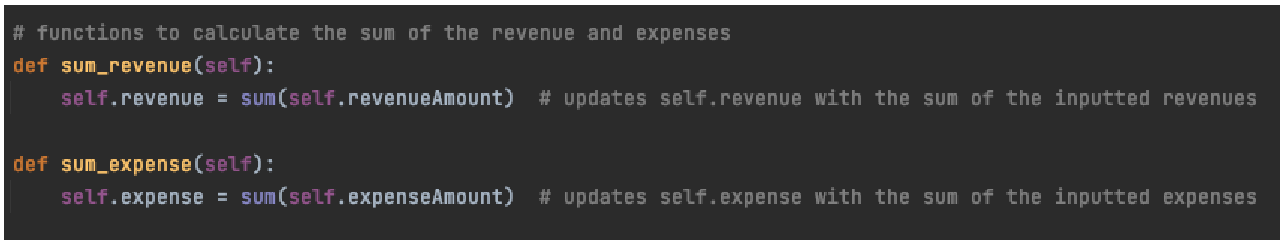


```python
# input methods
def add_ncAsset(self):
    ncAsset_added = input("Would you like to input a non-current asset? [Y/N]: ").lower()
    return ncAsset_added  # returns the user's response for non-current asset (yes or no)

def add_cAsset(self):
    cAsset_added = input("Would you like to input a current asset? [Y/N]: ").lower()
    return cAsset_added  # returns the user's response for current asset (yes or no)

def add_equity(self):
    equity_added = input("Would you like to input an equity? [Y/N]: ").lower()
    return equity_added  # returns the user's response for equity

def add_liability(self):
    liability_added = input("Would you like to input an liability? [Y/N]: ").lower()
    return liability_added  # returns the user's response for liability
```

Image 2.9 Input methods from SOFP.py

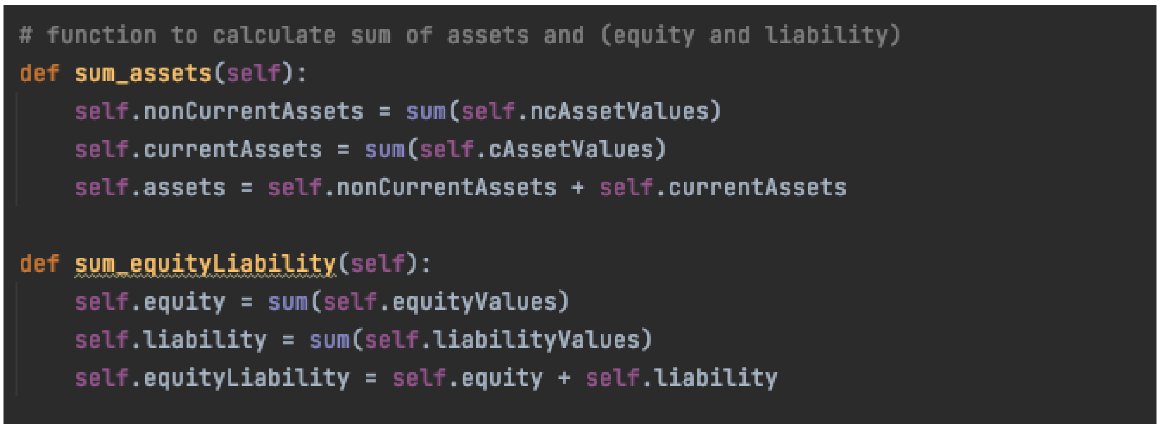
Next are methods that calculate the sum of each section and update it to the total variables that were explained before (example: self.revenue). It makes use of the built-in function sum() to add every element in the list that contains the values of the section.

For the IncomeStatement.py file, only the sum of the revenue and the sum of the expenses are calculated, while in the SOFP.py file, the sum of the non-current asset, the current asset, the assets as a whole, the equity, the liability and the equity and liability together are calculated. However, both files only have 2 methods.



```python
# functions to calculate the sum of the revenue and expenses
def sum_revenue(self):
    self.revenue = sum(self.revenueAmount)  # updates self.revenue with the sum of the inputted revenues

def sum_expense(self):
    self.expense = sum(self.expenseAmount)  # updates self.expense with the sum of the inputted expenses
```

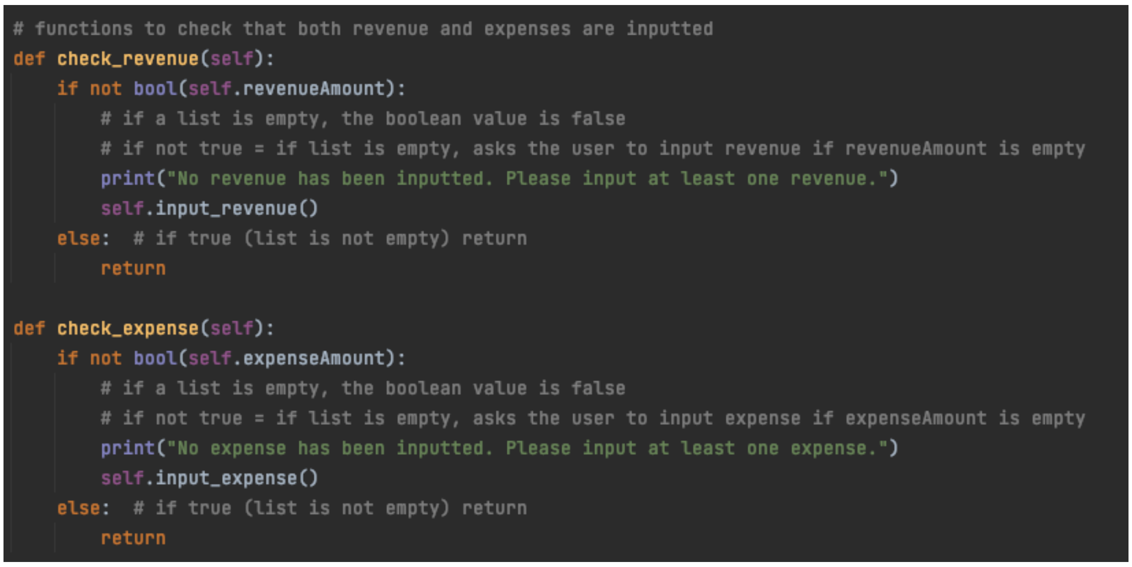
Image 2.10 Sum methods from IncomeStatement.py



```python
# function to calculate sum of assets and (equity and liability)
def sum_assets(self):
    self.nonCurrentAssets = sum(self.ncAssetValues)
    self.currentAssets = sum(self.cAssetValues)
    self.assets = self.nonCurrentAssets + self.currentAssets

def sum_equityLiability(self):
    self.equity = sum(self.equityValues)
    self.liability = sum(self.liabilityValues)
    self.equityLiability = self.equity + self.liability
```

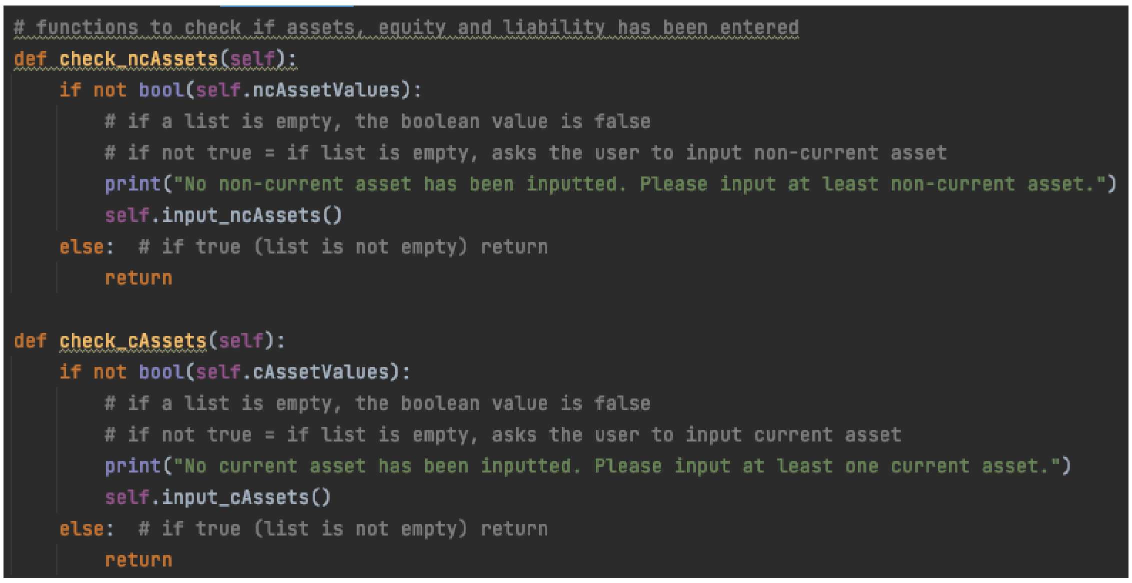
Image 2.11 Sum methods from SOFP.py

In order to make sure that there is at least one input for each section, a custom method is made. The method works through if statements that check if there is something inside the list of values for each section. It uses boolean values in which False is returned if the list is empty. If the list is empty, it prints a statement informing the user that they have not inputted anything and calls the function used to input a name and value.

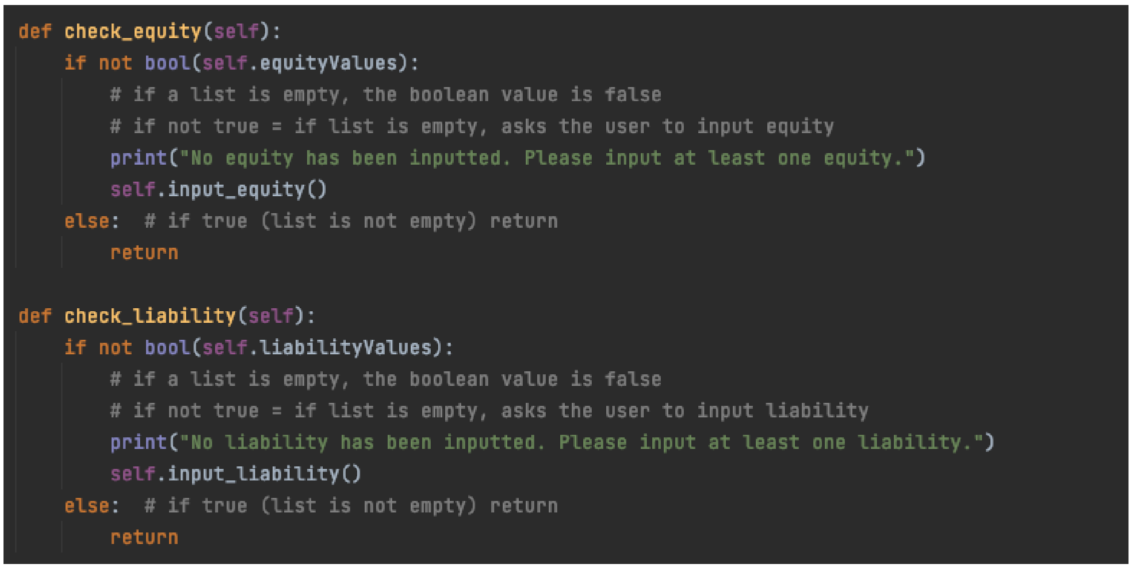


```python
# functions to check that both revenue and expenses are inputted
def check_revenue(self):
    if not bool(self.revenueAmount):
        # if a list is empty, the boolean value is false
        # if not true = if list is empty, asks the user to input revenue if revenueAmount is empty
        print("No revenue has been inputted. Please input at least one revenue.")
        self.input_revenue()
    else:  # if true (list is not empty) return
        return

def check_expense(self):
    if not bool(self.expenseAmount):
        # if a list is empty, the boolean value is false
        # if not true = if list is empty, asks the user to input expense if expenseAmount is empty
        print("No expense has been inputted. Please input at least one expense.")
        self.input_expense()
    else:  # if true (list is not empty) return
        return
```

Image 2.12 Check methods from IncomeStatement.py

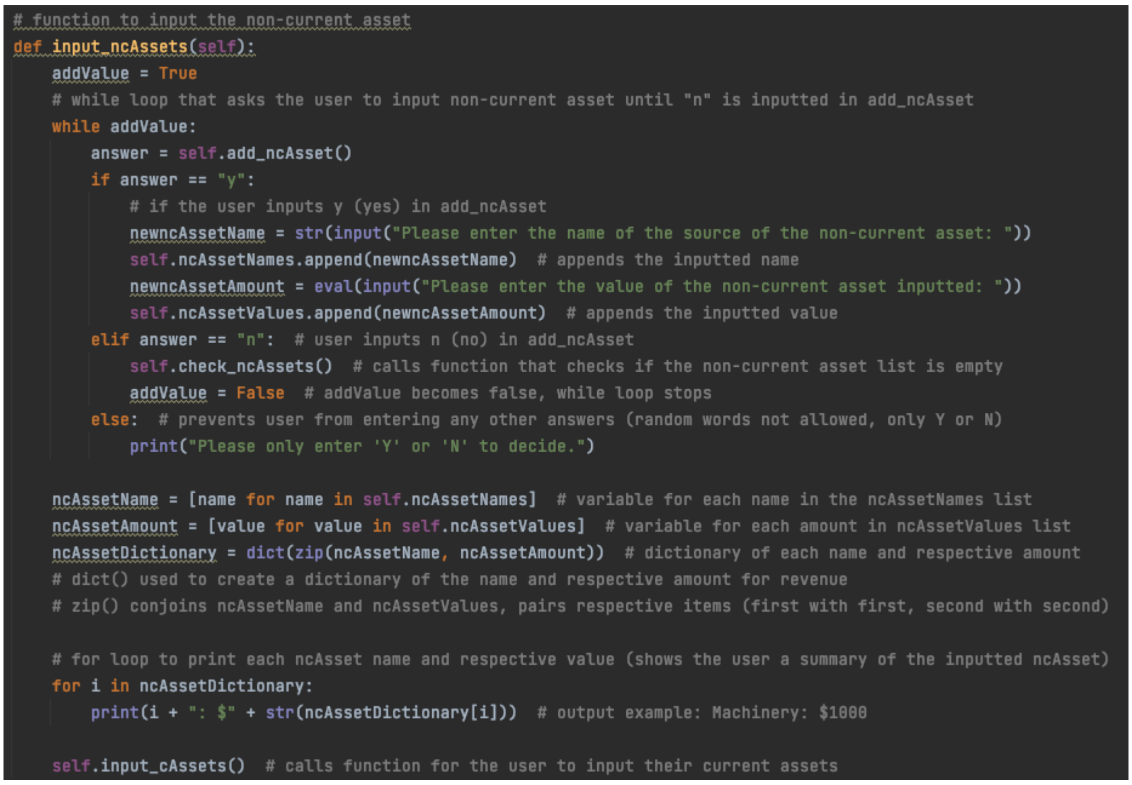


```python
# functions to check if assets, equity and liability has been entered
def check_ncAssets(self):
    if not bool(self.ncAssetValues):
        # if a list is empty, the boolean value is false
        # if not true = if list is empty, asks the user to input non-current asset
        print("No non-current asset has been inputted. Please input at least non-current asset.")
        self.input_ncAssets()
    else:  # if true (list is not empty) return
        return

def check_cAssets(self):
    if not bool(self.cAssetValues):
        # if a list is empty, the boolean value is false
        # if not true = if list is empty, asks the user to input current asset
        print("No current asset has been inputted. Please input at least one current asset.")
        self.input_cAssets()
    else:  # if true (list is not empty) return
        return
```



```python
def check_equity(self):
    if not bool(self.equityValues):
        # if a list is empty, the boolean value is false
        # if not true = if list is empty, asks the user to input equity
        print("No equity has been inputted. Please input at least one equity.")
        self.input_equity()
    else:  # if true (list is not empty) return
        return

def check_liability(self):
    if not bool(self.liabilityValues):
        # if a list is empty, the boolean value is false
        # if not true = if list is empty, asks the user to input liability
        print("No liability has been inputted. Please input at least one liability.")
        self.input_liability()
    else:  # if true (list is not empty) return
        return
```

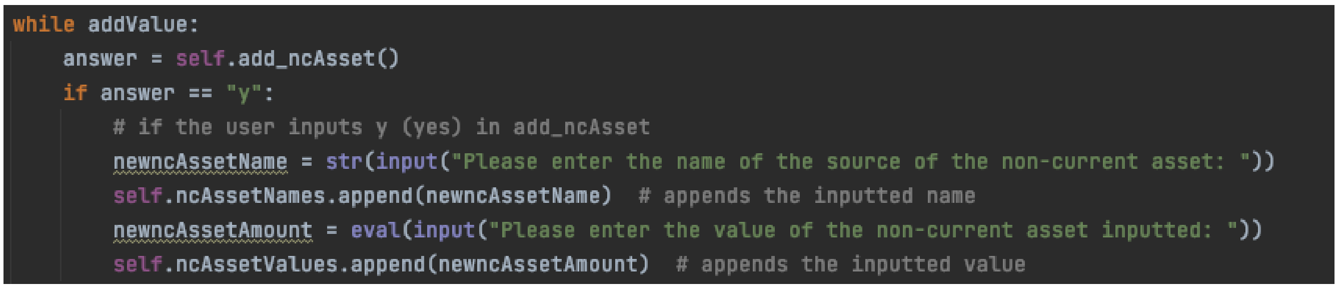
Images 2.13 Check methods from SOFP.py



```python
# function to input the non-current asset
def input_ncAssets(self):
    addValue = True
    # while loop that asks the user to input non-current asset until "n" is inputted in add_ncAsset
    while addValue:
        answer = self.add_ncAsset()
        if answer == "y":
            # if the user inputs y (yes) in add_ncAsset
            newncAssetName = str(input("Please enter the name of the source of the non-current asset: "))
            self.ncAssetNames.append(newncAssetName)  # appends the inputted name
            newncAssetAmount = eval(input("Please enter the value of the non-current asset inputted: "))
            self.ncAssetValues.append(newncAssetAmount)  # appends the inputted value
        elif answer == "n":  # user inputs n (no) in add_ncAsset
            self.check_ncAssets()  # calls function that checks if the non-current asset list is empty
            addValue = False  # addValue becomes false, while loop stops
        else:  # prevents user from entering any other answers (random words not allowed, only Y or N)
            print("Please only enter 'Y' or 'N' to decide.")

    ncAssetName = [name for name in self.ncAssetNames]  # variable for each name in the ncAssetNames list
    ncAssetAmount = [value for value in self.ncAssetValues]  # variable for each amount in ncAssetValues list
    ncAssetDictionary = dict(zip(ncAssetName, ncAssetAmount))  # dictionary of each name and respective amount
    # dict() used to create a dictionary of the name and respective amount for revenue
    # zip() conjoins ncAssetName and ncAssetValues, pairs respective items (first with first, second with second)

    # for loop to print each ncAsset name and respective value (shows the user a summary of the inputted ncAsset)
    for i in ncAssetDictionary:
        print(i + ": $" + str(ncAssetDictionary[i]))  # output example: Machinery: $1000

    self.input_cAssets()  # calls function for the user to input their current assets
```

Image 2.14 Input method for non-current assets from SOFP.py

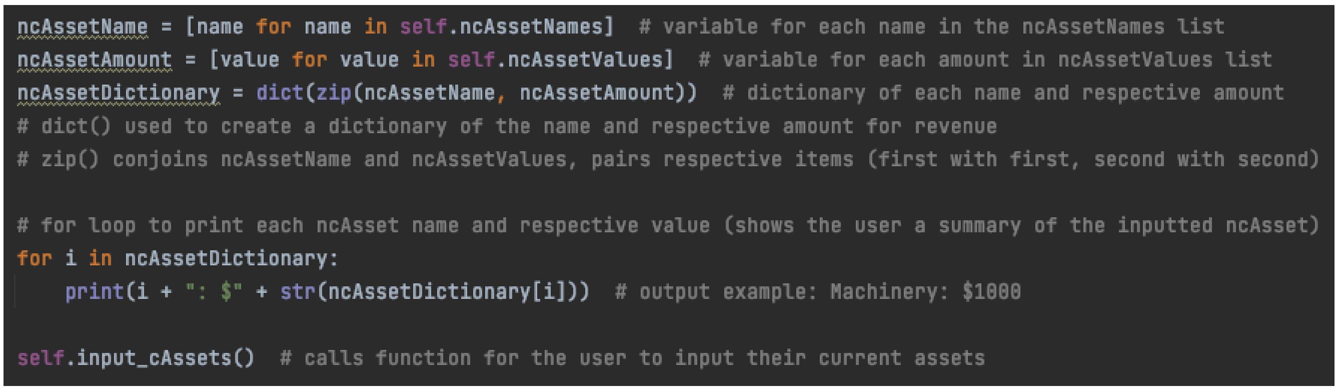
The input methods are similar for all sections, the only differences are the variables used and the input text. The logic is the same for all input methods in which a while loop is used to continuously let the user input a name and value until the user inputs N (no) in the ask methods. After the user inputs “n”, the check methods are called to ensure that the lists are not empty for the respective section. A variable is declared before the loop called addValue and assigned a boolean value True. This is to make sure that the while loop continues looping until addValue is False, which is done when the user inputs “n”, stopping the loop. The method makes use of if statements based on the user’s input from the add methods. There is also an else statement that acts as validation, similarly to the Driver.py file in which no other letters besides Y or N are accepted.



```python
while addValue:
    answer = self.add_ncAsset()
    if answer == "y":
        # if the user inputs y (yes) in add_ncAsset
        newncAssetName = str(input("Please enter the name of the source of the non-current asset: "))
        self.ncAssetNames.append(newncAssetName)  # appends the inputted name
        newncAssetAmount = eval(input("Please enter the value of the non-current asset inputted: "))
        self.ncAssetValues.append(newncAssetAmount)  # appends the inputted value
```

Image 2.15 Close up snippet of if statement from input method for non-current assets

As shown in image 2.15, two variables are declared for each section, one for the name and one for the value. The inputted name is then appended to the name list variable that was declared in the __init__ method. The same is done to the inputted value. The str() function is to ensure that the user inputs a string for the name, while the eval() function allows the user to enter an integer, for example, 1000, or a float, for example, 10.50. This is necessary as the value inputted is money.



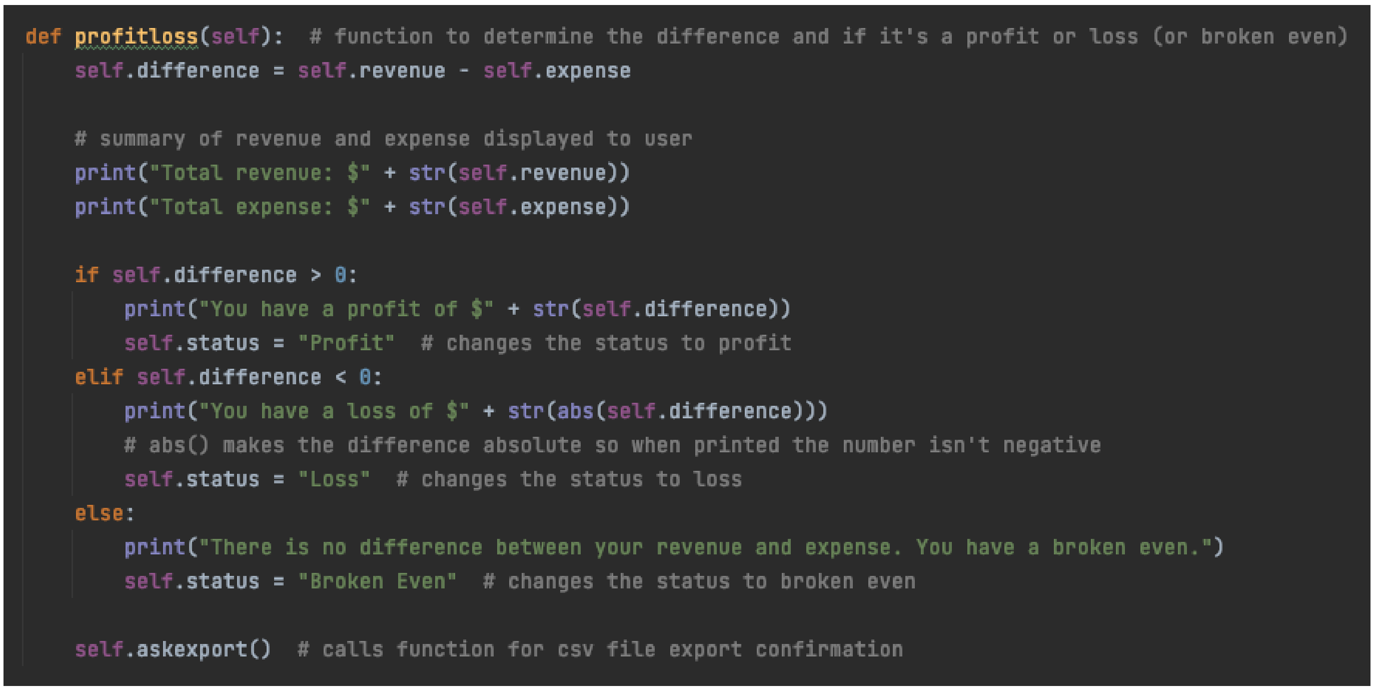
```python
ncAssetName = [name for name in self.ncAssetNames]  # variable for each name in the ncAssetNames list
ncAssetAmount = [value for value in self.ncAssetValues]  # variable for each amount in ncAssetValues list
ncAssetDictionary = dict(zip(ncAssetName, ncAssetAmount))  # dictionary of each name and respective amount
# dict() used to create a dictionary of the name and respective amount for revenue
# zip() conjoins ncAssetName and ncAssetValues, pairs respective items (first with first, second with second)

# for loop to print each ncAsset name and respective value (shows the user a summary of the inputted ncAsset)
for i in ncAssetDictionary:
    print(i + ": $" + str(ncAssetDictionary[i]))  # output example: Machinery: $1000

self.input_cAssets()  # calls function for the user to input their current assets
```

Image 2.16 Close up snippet of for loop statement from input method for non-current assets

After the if statements, variables are called for each name in the name list and for each value in the value list. This is so that it could be used to create a dictionary that uses the name as a key and the money value as an item. The function zip() pairs it together based on the index, which is possible as each name shares the same index with its respective value. The dict() function is used to create the dictionary.

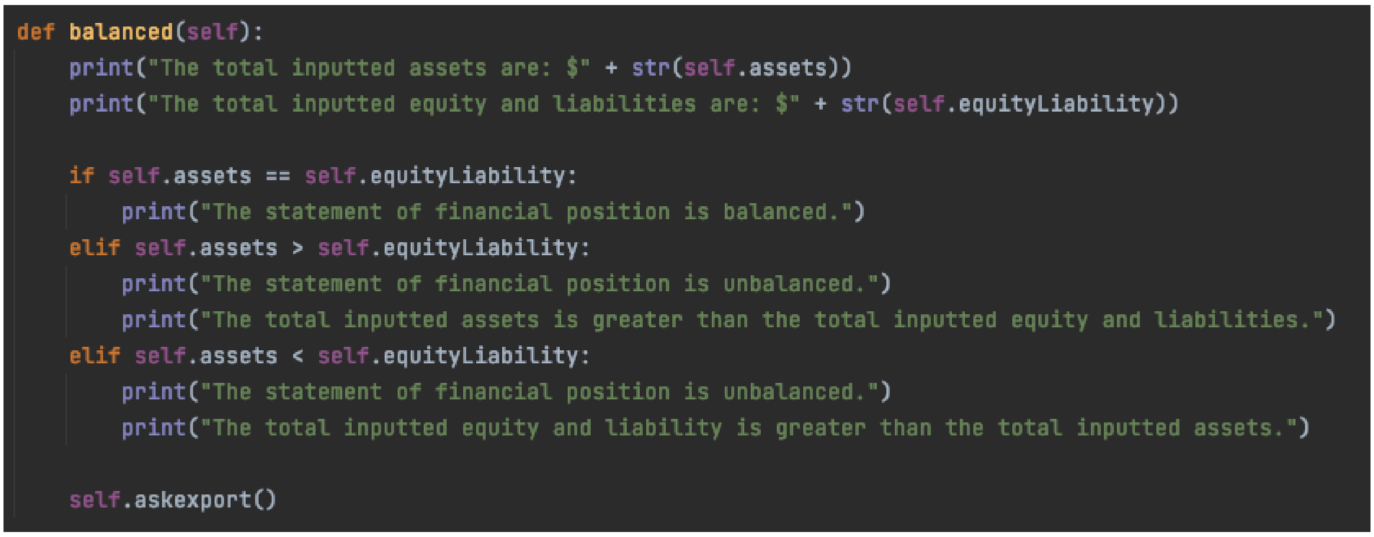
As shown in image 2.16, there is a for loop afterwards that makes use of the dictionary to print the name and value of each input. This is to give the user a summary of their input from each section. Afterwards, it will call the function for the next section. If it is the last section, it will call the function that will either calculate the profit/loss (for income statement) or check if the totals are balanced (for the statement of financial position) and display it to the user.

As shown in image 2.17 (next page), the profitloss method is done to calculate the difference between the revenue and the expenses to determine the profit or loss that the user has. It makes use of an if statement to tell the user if they gained money (profit), lost money (loss) or neither (broken even). The method prints the total revenue and total expenses that the user had inputted as a summary, afterwards it prints the profit or loss they had made. Once it is done, it calls the function that asks the user if they want to export the data to a CSV file.



```python
def profitloss(self):  # function to determine the difference and if it's a profit or loss (or broken even)
    self.difference = self.revenue - self.expense

    # summary of revenue and expense displayed to user
    print("Total revenue: $" + str(self.revenue))
    print("Total expense: $" + str(self.expense))

    if self.difference > 0:
        print("You have a profit of $" + str(self.difference))
        self.status = "Profit"  # changes the status to profit
    elif self.difference < 0:
        print("You have a loss of $" + str(abs(self.difference)))
        # abs() makes the difference absolute so when printed the number isn't negative
        self.status = "Loss"  # changes the status to loss
    else:
        print("There is no difference between your revenue and expense. You have a broken even.")
        self.status = "Broken Even"  # changes the status to broken even

    self.askexport()  # calls function for csv file export confirmation
```

Image 2.17 profitloss() method to calculate and display the user’s profit or loss

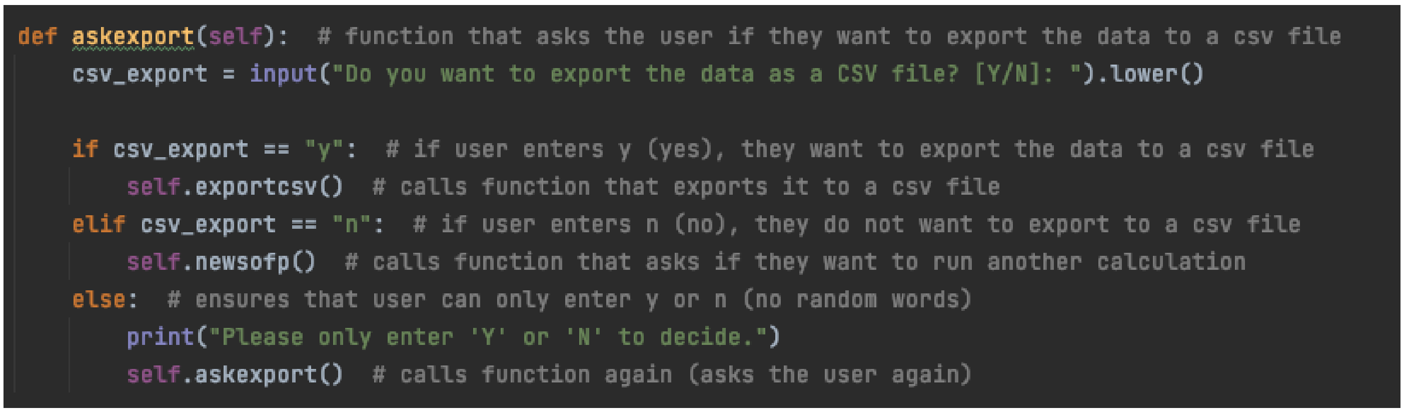


```python
def balanced(self):
    print("The total inputted assets are: $" + str(self.assets))
    print("The total inputted equity and liabilities are: $" + str(self.equityLiability))

    if self.assets == self.equityLiability:
        print("The statement of financial position is balanced.")
    elif self.assets > self.equityLiability:
        print("The statement of financial position is unbalanced.")
        print("The total inputted assets is greater than the total inputted equity and liabilities.")
    elif self.assets < self.equityLiability:
        print("The statement of financial position is unbalanced.")
        print("The total inputted equity and liability is greater than the total inputted assets.")

    self.askexport()
```

Image 2.18 balanced() method to calculate and display whether or not the statement is balanced

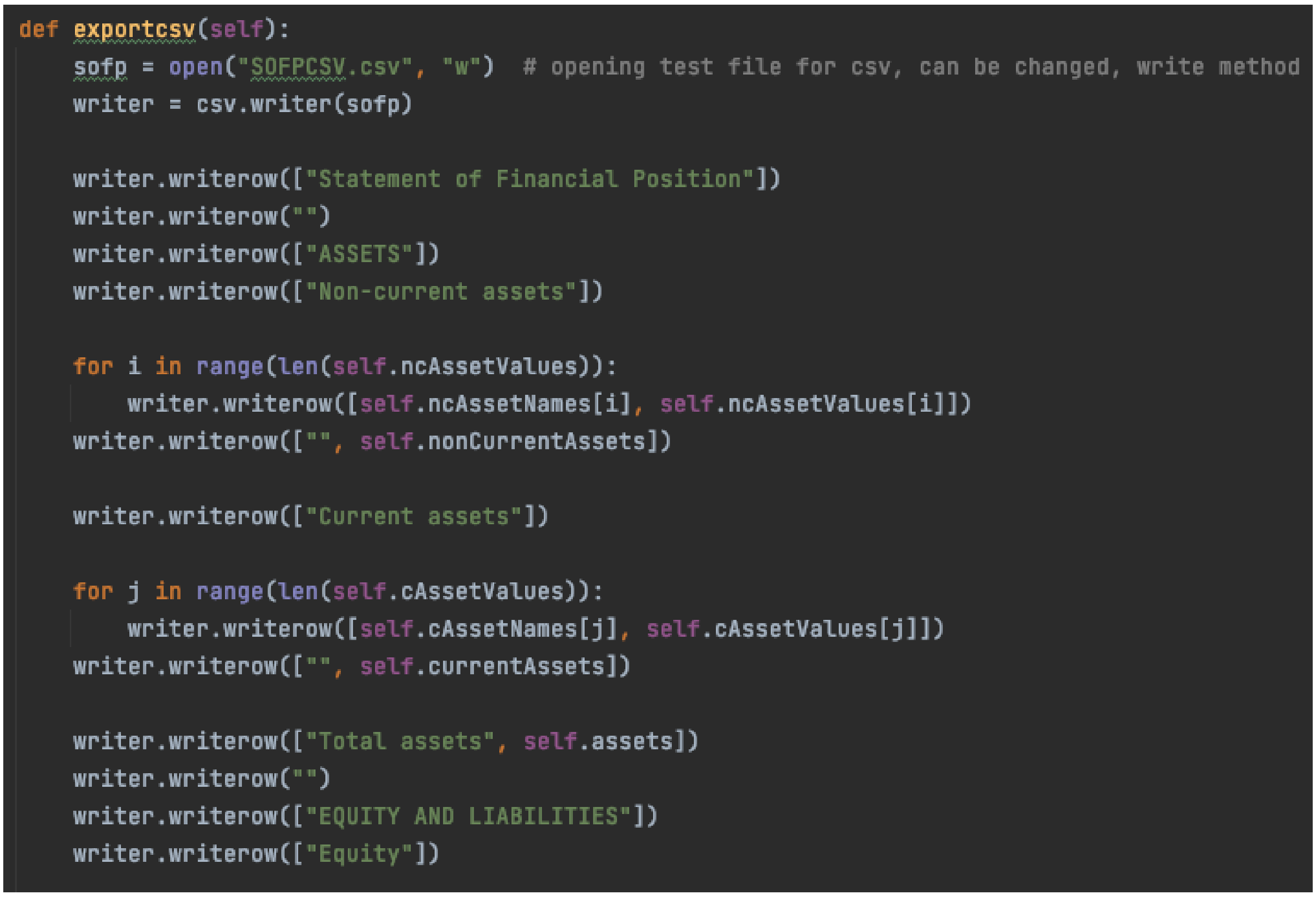
As the statement of financial position is not made to calculate the user’s profit or loss, a different method is used to check if the totals are balanced. Similarly to the profitloss() method, it first displays the totals of the assets and the totals of the equity and liabilities as a summary. It then uses if statements to conclude and print whether or not the statement is balanced. It also tells the user which section has a greater value. Once it is done, it calls the function that asks the user if they want to export the data to a CSV file.

Both files use the same method to offer the option of CSV exporting to the user. The method uses if statements based on the user’s input. If the user wants to export the data, it will call the function that will do the actual writing. If the user does not, it will either ask the user if they want to display the data in a chart (income statement) or it will ask the user if they want to run the program again.



```python
def askexport(self):  # function that asks the user if they want to export the data to a csv file
    csv_export = input("Do you want to export the data as a CSV file? [Y/N]: ").lower()

    if csv_export == "y":  # if user enters y (yes), they want to export the data to a csv file
        self.exportcsv()  # calls function that exports it to a csv file
    elif csv_export == "n":  # if user enters n (no), they do not want to export to a csv file
        self.newsofp()  # calls function that asks if they want to run another calculation
    else:  # ensures that user can only enter y or n (no random words)
        print("Please only enter 'Y' or 'N' to decide.")
        self.askexport()  # calls function again (asks the user again)
```

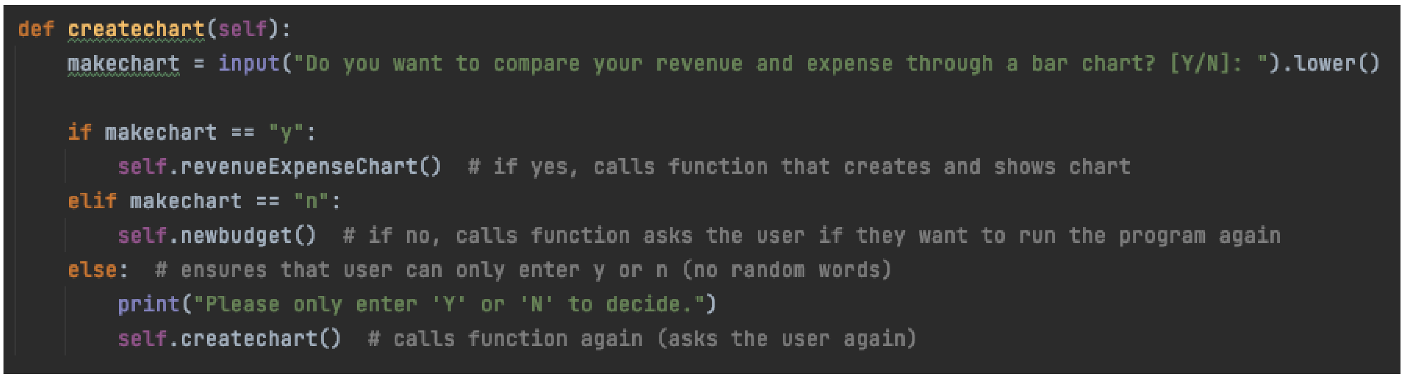
Image 2.19 askexport() method to ask the user if they would like to export to CSV file



```python
def exportcsv(self):
    sofp = open("SOFPCSV.csv", "w")  # opening test file for csv, can be changed, write method
    writer = csv.writer(sofp)

    writer.writerow(["Statement of Financial Position"])
    writer.writerow("")
    writer.writerow(["ASSETS"])
    writer.writerow(["Non-current assets"])

    for i in range(len(self.ncAssetValues)):
        writer.writerow([self.ncAssetNames[i], self.ncAssetValues[i]])
    writer.writerow(["", self.nonCurrentAssets])

    writer.writerow(["Current assets"])

    for j in range(len(self.cAssetValues)):
        writer.writerow([self.cAssetNames[j], self.cAssetValues[j]])
    writer.writerow(["", self.currentAssets])

    writer.writerow(["Total assets", self.assets])
    writer.writerow("")
    writer.writerow(["EQUITY AND LIABILITIES"])
    writer.writerow(["Equity"])
```

Image 2.20 Snippet of method that writes the data to a CSV file from SOFP.py

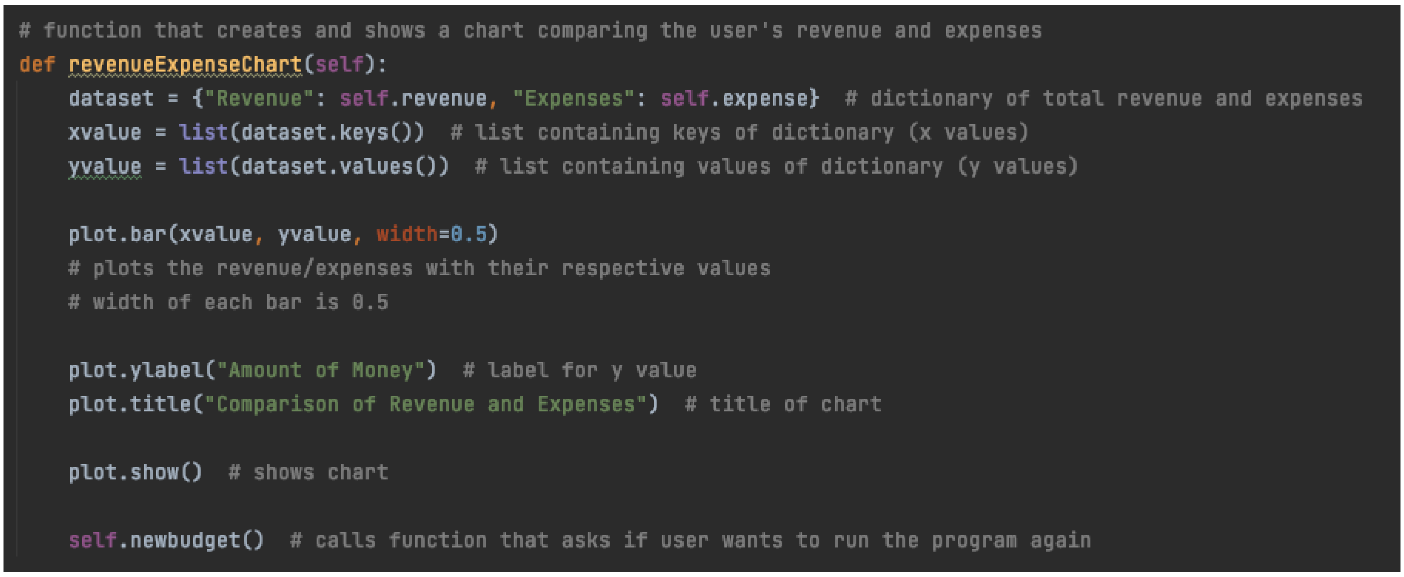
Image 2.20 is a part of the method that is used to write the data to a CSV file. The file is opened and the write method is chosen. Afterwards, using the .writerow() function allows the method to write the headers for each section. The for loop is used to write each name and respective value from each section. After each section, the total is written. The file is then closed using .close() and the next function is called.



```python
def createchart(self):
    makechart = input("Do you want to compare your revenue and expense through a bar chart? [Y/N]: ").lower()

    if makechart == "y":
        self.revenueExpenseChart()  # if yes, calls function that creates and shows chart
    elif makechart == "n":
        self.newbudget()  # if no, calls function asks the user if they want to run the program again
    else:  # ensures that user can only enter y or n (no random words)
        print("Please only enter 'Y' or 'N' to decide.")
        self.createchart()  # calls function again (asks the user again)
```

Image 2.21 Method that asks the user if they would like to create a chart

The next function for the IncomeStatement.py is the createchart() method. Similarly to the ask functions, it asks the user if they would like to create a bar chart that displays the revenue and expenses so that the user can properly see the difference between them. If the user inputs “y”, the method that creates and displays the chart is called. If the user inputs “n”, the function that asks the user if they would like to run the program is called. Since the statement of financial position is meant to be balanced, it directly calls this function as a chart is not needed to be displayed.

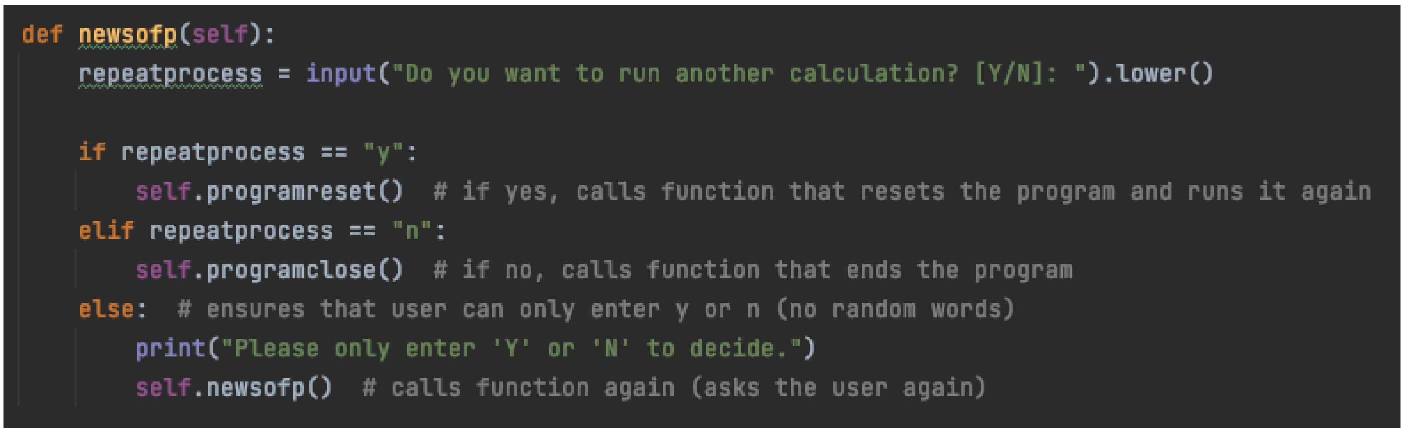


```python
# function that creates and shows a chart comparing the user's revenue and expenses
def revenueExpenseChart(self):
    dataset = {"Revenue": self.revenue, "Expenses": self.expense}  # dictionary of total revenue and expenses
    xvalue = list(dataset.keys())  # list containing keys of dictionary (x values)
    yvalue = list(dataset.values())  # list containing values of dictionary (y values)

    plot.bar(xvalue, yvalue, width=0.5)
    # plots the revenue/expenses with their respective values
    # width of each bar is 0.5

    plot.ylabel("Amount of Money")  # label for y value
    plot.title("Comparison of Revenue and Expenses")  # title of chart

    plot.show()  # shows chart

    self.newbudget()  # calls function that asks if user wants to run the program again
```

Image 2.22 Method that creates and displays the bar graph

The chart method uses matplotlib to generate and display the bar graph using the revenue total and the expense total. First, the dataset variable is made and a dictionary of the totals are assigned to it. The keys of the dictionary is the label (revenue, expenses) while the values are the respective totals. The keys are then set for the x values and the values are set for the y values, through the use of .keys() and .values(), which are methods to get the key and value from a dictionary.

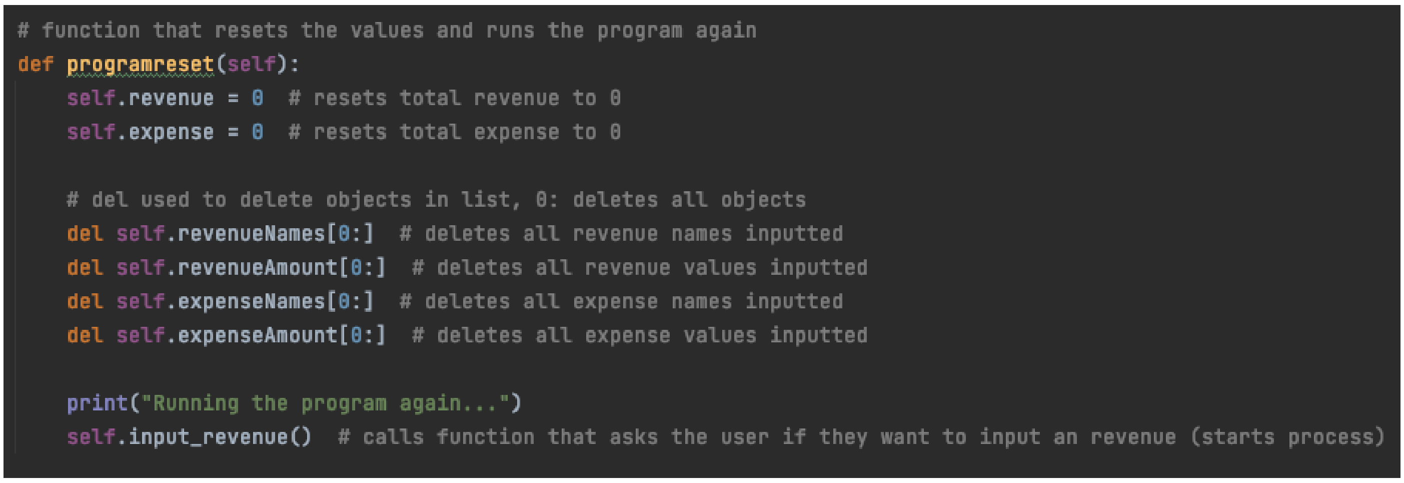
The bar graph is then plot through the use of matplotlib’s pyplot, which takes in the xvalue, yvalue and the width of each bar, which has been set to 0.5. As the label for the x value is already set, only the label of the y value is needed, which is done through .ylabel() with the text “Amount of Money”. The plot title is also set to “Comparison of Revenue and Expenses”. The graph can then be displayed through .show() function. Once it is done, the next function is called.



```python
def newsofp(self):
    repeatprocess = input("Do you want to run another calculation? [Y/N]: ").lower()

    if repeatprocess == "y":
        self.programreset()  # if yes, calls function that resets the program and runs it again
    elif repeatprocess == "n":
        self.programclose()  # if no, calls function that ends the program
    else:  # ensures that user can only enter y or n (no random words)
        print("Please only enter 'Y' or 'N' to decide.")
        self.newsofp()  # calls function again (asks the user again)
```

Image 2.23 Method that asks the user if they want to run the program again from SOFP.py

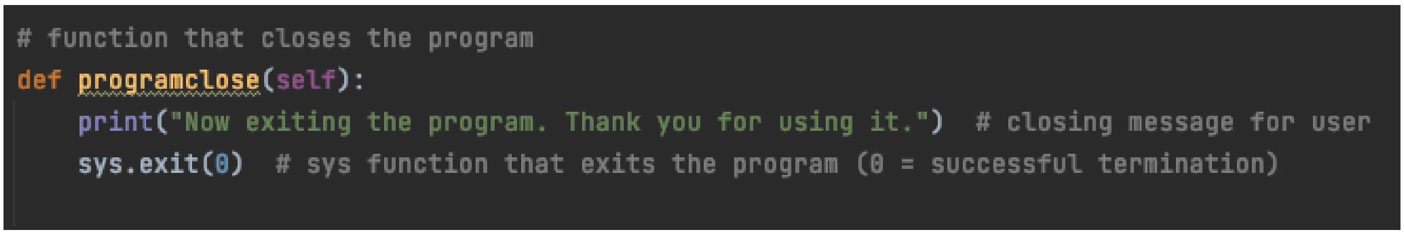
The method shown in image 2.23 is the next function that is called after the chart method in the IncomeStatement.py file, and directly called after the CSV method in the SOFP.py file. It asks the user if they would like to run the program again and calls the reset program function if they do. If they don’t the function that closes the program is called. The else statement is used as validation to ensure that only “Y” or “N” is inputted.



```python
# function that resets the values and runs the program again
def programreset(self):
    self.revenue = 0  # resets total revenue to 0
    self.expense = 0  # resets total expense to 0

    # del used to delete objects in list, 0: deletes all objects
    del self.revenueNames[0:]  # deletes all revenue names inputted
    del self.revenueAmount[0:]  # deletes all revenue values inputted
    del self.expenseNames[0:]  # deletes all expense names inputted
    del self.expenseAmount[0:]  # deletes all expense values inputted

    print("Running the program again...")
    self.input_revenue()  # calls function that asks the user if they want to input an revenue (starts process)
```

Image 2.24 Method that resets the program to be run again

The custom-made programreset() method is done to reset all the variables back to its default. First the integer variables are set back to 0. Then, using the del keyword, it removes all the objects in the list variables so that the lists are empty once again. It will then tell the user that the program is running again and calls the first function to start the flow once more. The method works the same for both files IncomeStatement.py and SOFP.py, the only difference is the variables that are included in the method. However, the logic behind it is the same.

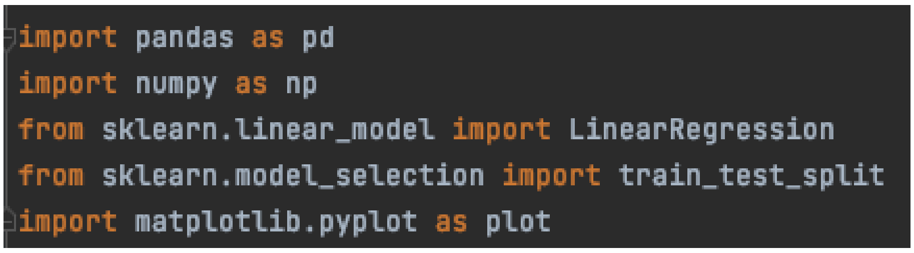


```python
# function that closes the program
def programclose(self):
    print("Now exiting the program. Thank you for using it.")  # closing message for user
    sys.exit(0)  # sys function that exits the program (0 = successful termination)
```

Image 2.25 Method that closes the program

The programclose() method is called if the user does not want to run the program again. It makes use of the sys module to close the program through sys.exit. As the last process was an if statement, sys.exit is needed to terminate it as the else statement was needed for validation. The method tells the user that the program is being closed and exits the program.

2.3. SalaryPredict.py

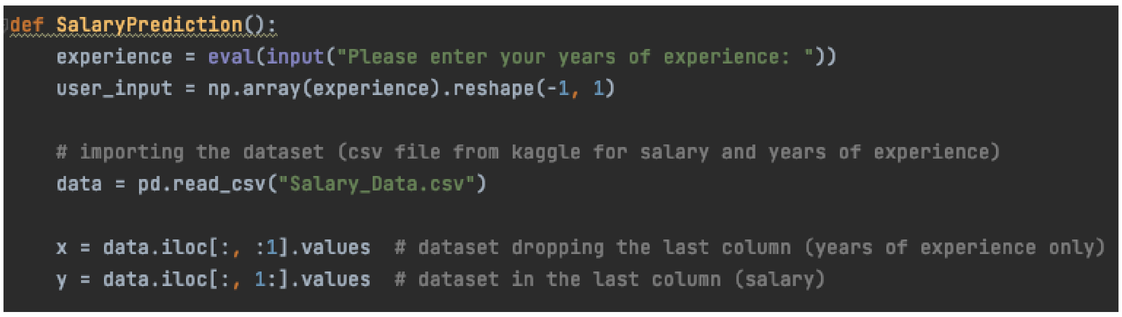


```python
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plot
```

Image 2.26 Modules used in SalaryPredict.py

The modules used in the salary prediction file consist of:

a. Pandas - library used for data manipulation and analysis
b. NumPy - library used for large arrays and datasets
c. Sklearn - machine learning library
d. Matplotlib - used to plot and display graphs

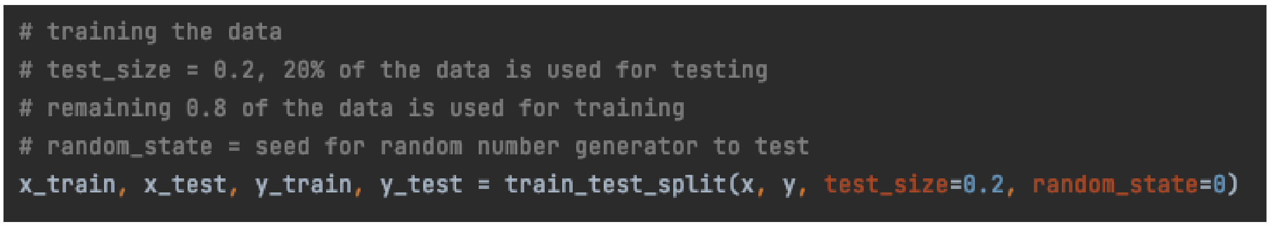


```python
def SalaryPrediction():
    experience = eval(input("Please enter your years of experience: "))
    user_input = np.array(experience).reshape(-1, 1)

    # importing the dataset (csv file from kaggle for salary and years of experience)
    data = pd.read_csv("Salary_Data.csv")

    x = data.iloc[:, :1].values  # dataset dropping the last column (years of experience only)
    y = data.iloc[:, 1:].values  # dataset in the last column (salary)
```

Image 2.27 Snippet of SalaryPrediction function from SalaryPredict.py

First, a variable is made to accept input from the user of their years of experience. The user's input is then made into an array through NumPy as it will not be able to be used in the predict function if it remains a float. This is because the .pred function compares the input to the arrays of salaries and their respective years from the dataset.

The dataset from Kaggle.com is then read through the use of pandas and assigned to the variable data. Pandas allow this to happen using the .read_csv function which, as the name suggests, reads CSV files into the dataframe. The x and y values are then set using .iloc, a pandas function that uses indexes for selecting the position of the data. The x value is the years of experience, while the y value is the salaries.

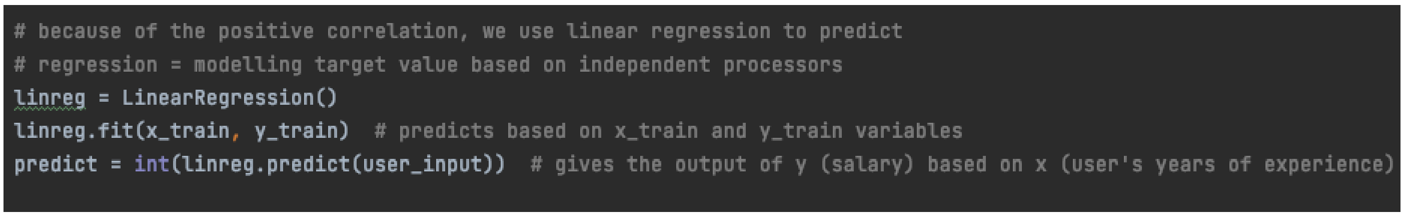


```python
# training the data
# test_size = 0.2, 20% of the data is used for testing
# remaining 0.8 of the data is used for training
# random_state = seed for random number generator to test
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)
```

Image 2.28 Snippet of testing and training from SalaryPredict.py

Image 2.28 shows that the x values and y values are then used for testing and training. Training is needed in machine learning in order to teach an algorithm. Testing is then done to test the data and compare the results, proving that the algorithm is working. This is done by plotting graphs of the trained data and the actual data to see if the prediction is accurate enough.

The line shown in image 2.28 assigns values to the variables x_train, x_test, y_train and y_test from the resulting array of the train_test_split function. It splits the data set into two parts, the training and testing data for each value (x and y). The test size chosen is 20% of the actual data, while the remaining 80% is used for training. Before the data is split, it can be shuffled by the function (random_state), however, by default, it is set to 0.

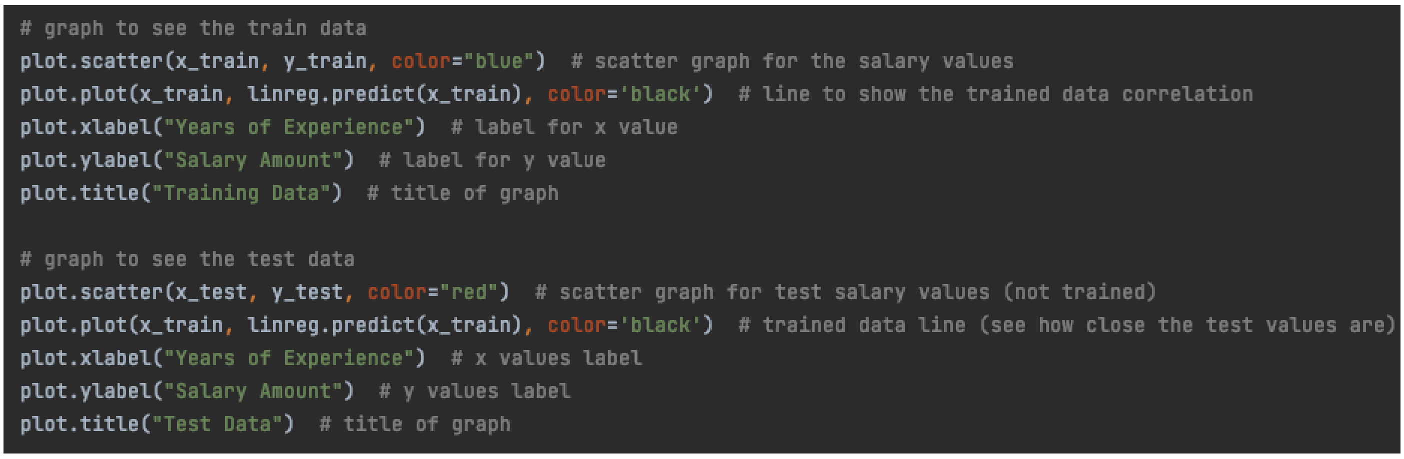


```
# because of the positive correlation, we use linear regression to predict
# regression = modelling target value based on independent processors
linreg = LinearRegression()
linreg.fit(x_train, y_train)  # predicts based on x_train and y_train variables
predict = int(linreg.predict(user_input))  # gives the output of y (salary) based on x (user's years of experience)
```

Image 2.29 Linear Regression used to predict the salary in SalaryPredict.py

Based on the dataset, the salary amount and the years of experience are directly proportional. The line graph is positively correlated, so the more years of experience they have, the higher the average salary. Thus, linear regression is used to predict the salary.

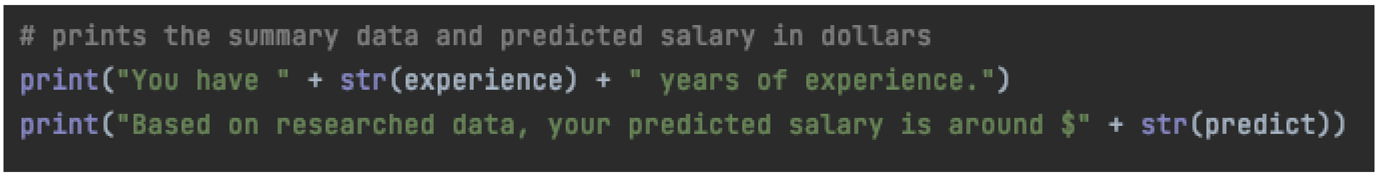
As it is based on a graph, the .fit() method is needed to find the coefficients of the equation for the algorithm, based on the trained x and y values. For example, the gradient of a graph is calculated by y = mx + c. The salary can then be predicted using the .predict() method.



```
# graph to see the train data
plot.scatter(x_train, y_train, color="blue")  # scatter graph for the salary values
plot.plot(x_train, linreg.predict(x_train), color='black')  # line to show the trained data correlation
plot.xlabel("Years of Experience")  # label for x value
plot.ylabel("Salary Amount")  # label for y value
plot.title("Training Data")  # title of graph

# graph to see the test data
plot.scatter(x_test, y_test, color="red")  # scatter graph for test salary values (not trained)
plot.plot(x_train, linreg.predict(x_train), color='black')  # trained data line (see how close the test values are)
plot.xlabel("Years of Experience")  # x values label
plot.ylabel("Salary Amount")  # y values label
plot.title("Test Data")  # title of graph
```

Image 2.30 Graph code for trained data and test data from SalaryPredict.py

To ensure that the machine learning is accurate, two graphs are plotted. One graph plots the trained data as a scatter plot, and best fit line to show the pattern and correlation of the data. The second graph plots the same best fit line, but the scatter plot is of the test data. This is to check how close the test data is to the best fit line. As this is not necessary for the user to see, the plot.show() line was deleted but the code is kept for reference.



```
# prints the summary data and predicted salary in dollars
print("You have " + str(experience) + " years of experience.")
print("Based on researched data, your predicted salary is around $" + str(predict))
```

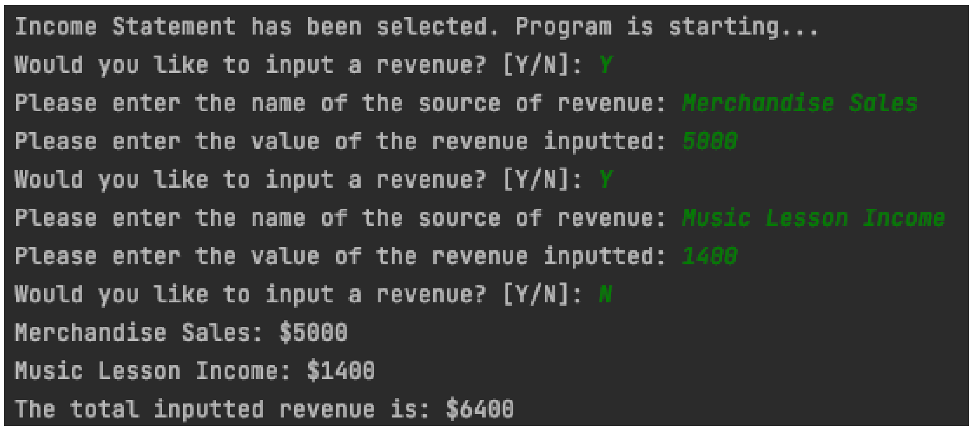
Image 2.30 Print statements in SalaryPredict.py

The code ends with print statements that display years of experience the user has and the predicted salary it has calculated based on the dataset.

3. Evidence of working program

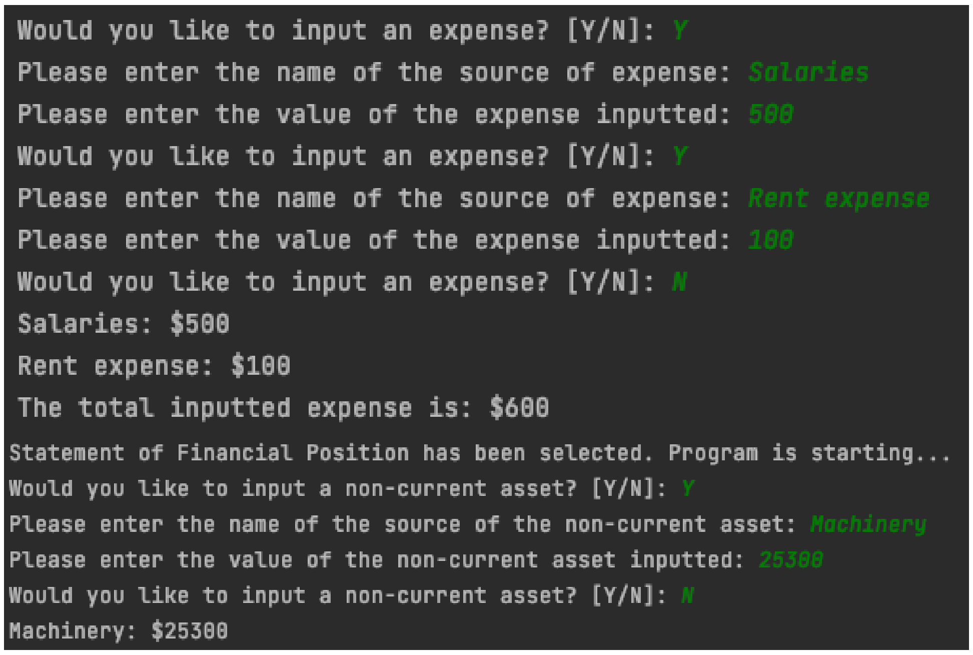
3.1. Summary display of each section

3.1.1. Income Statement (Revenue and Expenses)



```
Income Statement has been selected. Program is starting...
Would you like to input a revenue? [Y/N]: Y
Please enter the name of the source of revenue: Merchandise Sales
Please enter the value of the revenue inputted: 5000
Would you like to input a revenue? [Y/N]: Y
Please enter the name of the source of revenue: Music Lesson Income
Please enter the value of the revenue inputted: 1400
Would you like to input a revenue? [Y/N]: N
Merchandise Sales: $5000
Music Lesson Income: $1400
The total inputted revenue is: $6400
```

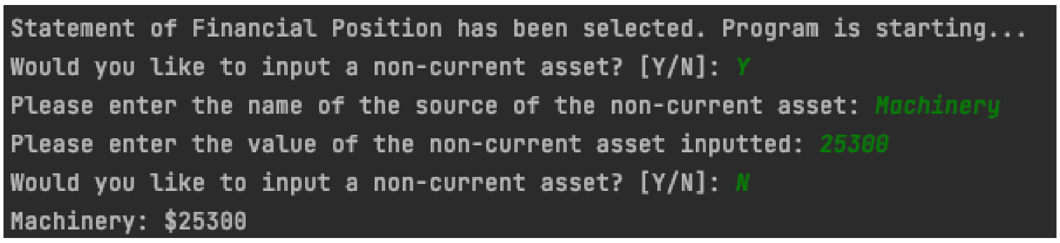
Image 3.1 Revenue input and output



```
Would you like to input an expense? [Y/N]: Y
Please enter the name of the source of expense: Salaries
Please enter the value of the expense inputted: 500
Would you like to input an expense? [Y/N]: Y
Please enter the name of the source of expense: Rent expense
Please enter the value of the expense inputted: 100
Would you like to input an expense? [Y/N]: N
Salaries: $500
Rent expense: $100
The total inputted expense is: $600
Statement of Financial Position has been selected. Program is starting...
Would you like to input a non-current asset? [Y/N]: Y
Please enter the name of the source of the non-current asset: Machinery
Please enter the value of the non-current asset inputted: 25300
Would you like to input a non-current asset? [Y/N]: N
Machinery: $25300
```

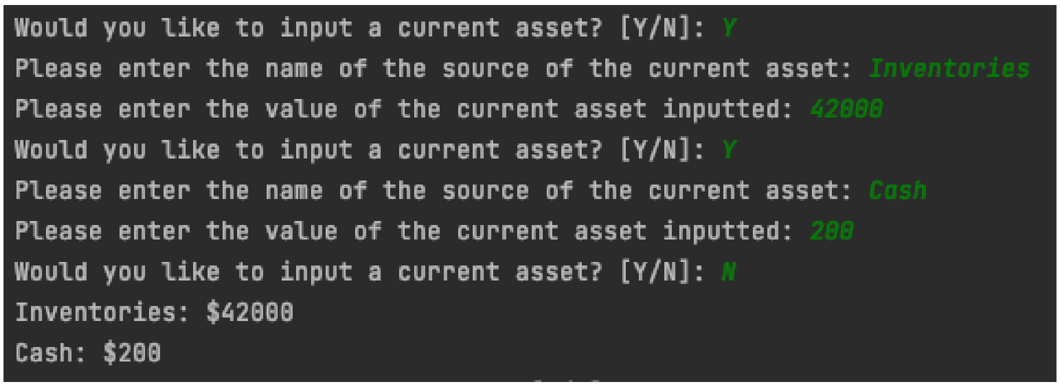
Image 3.2 Expense input and output

3.1.2. Statement of Financial Position



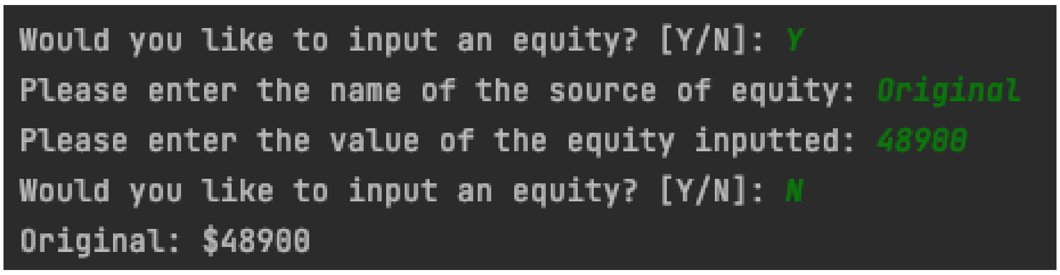
```
Statement of Financial Position has been selected. Program is starting...
Would you like to input a non-current asset? [Y/N]: Y
Please enter the name of the source of the non-current asset: Machinery
Please enter the value of the non-current asset inputted: 25300
Would you like to input a non-current asset? [Y/N]: N
Machinery: $25300
```

Image 3.3 Non-current asset input and output



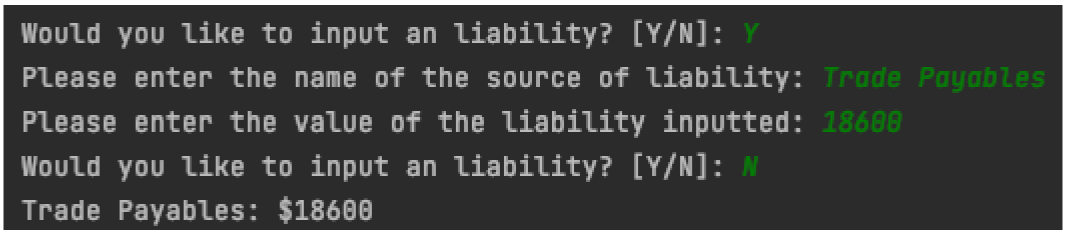
```
Would you like to input a current asset? [Y/N]: Y
Please enter the name of the source of the current asset: Inventories
Please enter the value of the current asset inputted: 42000
Would you like to input a current asset? [Y/N]: Y
Please enter the name of the source of the current asset: Cash
Please enter the value of the current asset inputted: 200
Would you like to input a current asset? [Y/N]: N
Inventories: $42000
Cash: $200
```

Image 3.4 Current asset input and output



```
Would you like to input an equity? [Y/N]: Y
Please enter the name of the source of equity: Original
Please enter the value of the equity inputted: 48900
Would you like to input an equity? [Y/N]: N
Original: $48900
```

Image 3.5 Equity input and output

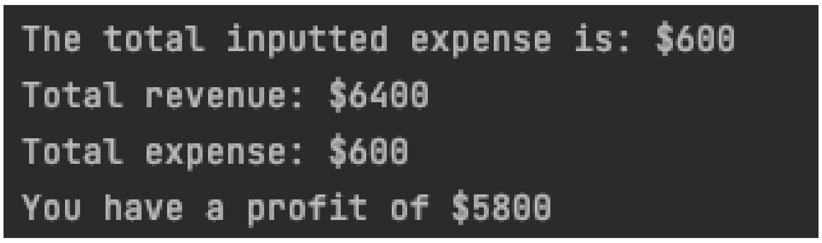


```
Would you like to input an liability? [Y/N]: Y
Please enter the name of the source of liability: Trade Payables
Please enter the value of the liability inputted: 18600
Would you like to input an liability? [Y/N]: N
Trade Payables: $18600
```

Image 3.6 Liability input and output

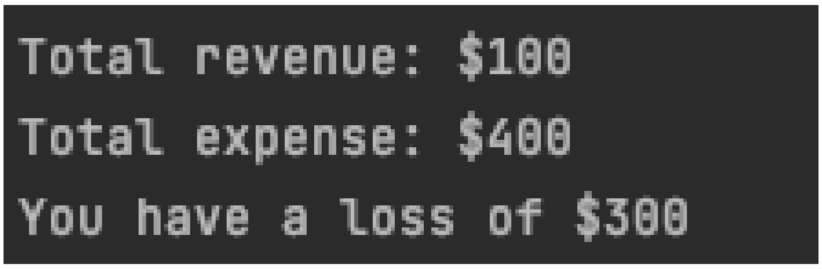
3.2. Calculated Conclusions

3.2.1. Income Statement



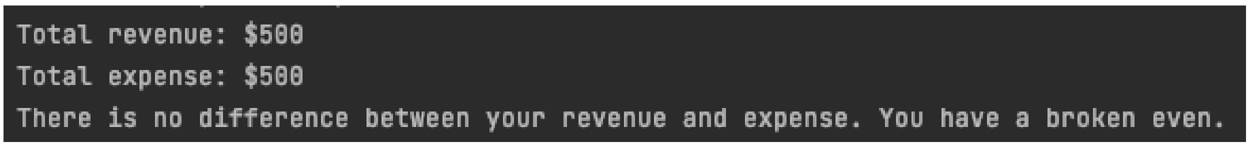
```
The total inputted expense is: $600
Total revenue: $6400
Total expense: $600
You have a profit of $5800
```

Image 3.7 Profit conclusion of IncomeStatement.py



```
Total revenue: $100
Total expense: $400
You have a loss of $300
```

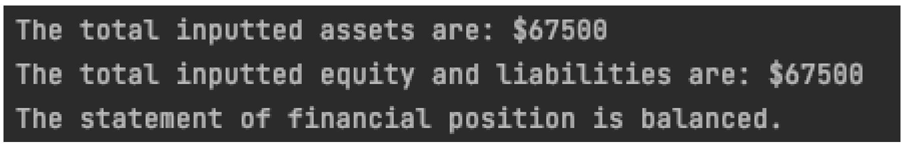
Image 3.8 Loss conclusion of IncomeStatement.py



```
Total revenue: $500
Total expense: $500
There is no difference between your revenue and expense. You have a broken even.
```

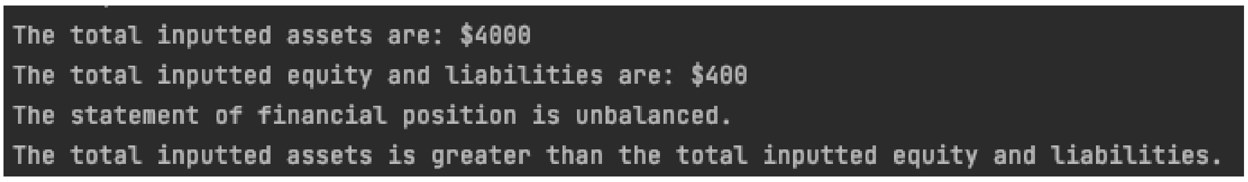
Image 3.9 Broken even conclusion of IncomeStatement.py

3.2.2. Statement of Financial Position



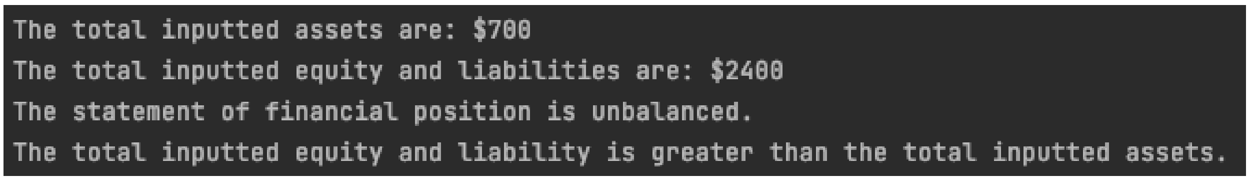
```
The total inputted assets are: $67500
The total inputted equity and liabilities are: $67500
The statement of financial position is balanced.
```

Image 3.10 Balanced conclusion of SOFP.py



```
The total inputted assets are: $4000
The total inputted equity and liabilities are: $400
The statement of financial position is unbalanced.
The total inputted assets is greater than the total inputted equity and liabilities.
```

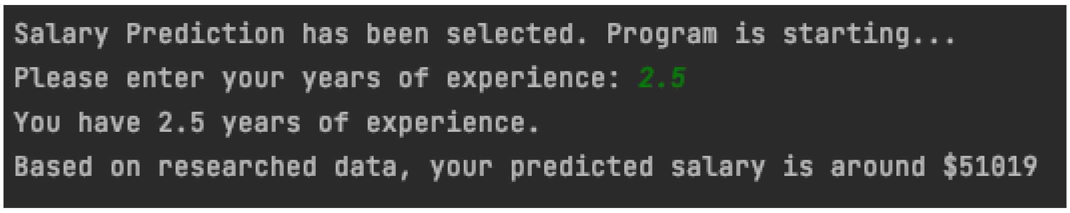
Image 3.11 Unbalanced with assets greater in SOFP.py



```
The total inputted assets are: $700
The total inputted equity and liabilities are: $2400
The statement of financial position is unbalanced.
The total inputted equity and liability is greater than the total inputted assets.
```

Image 3.12 Unbalanced with equity and liability greater in SOFP.py

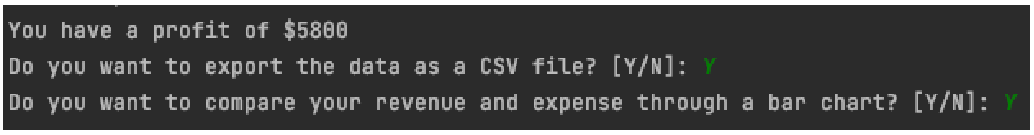
3.2.3. Salary Prediction



```
Salary Prediction has been selected. Program is starting...
Please enter your years of experience: 2.5
You have 2.5 years of experience.
Based on researched data, your predicted salary is around $51019
```

Image 3.13 Salary prediction conclusion

3.3. Graphs



```
You have a profit of $5800
Do you want to export the data as a CSV file? [Y/N]: Y
Do you want to compare your revenue and expense through a bar chart? [Y/N]: Y
```

Image 3.14 IncomeStatement.py program asking for bar chart plotting

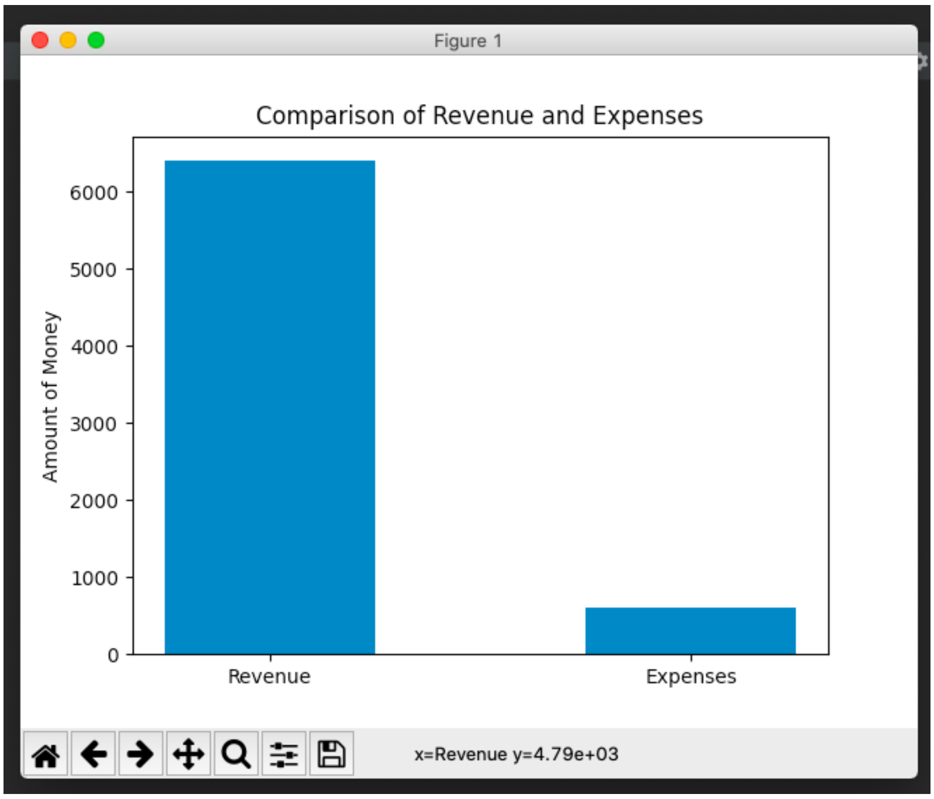
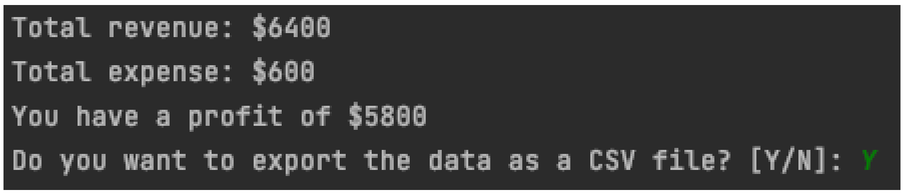


Image 3.15 Graph plotted and displayed by matplotlib

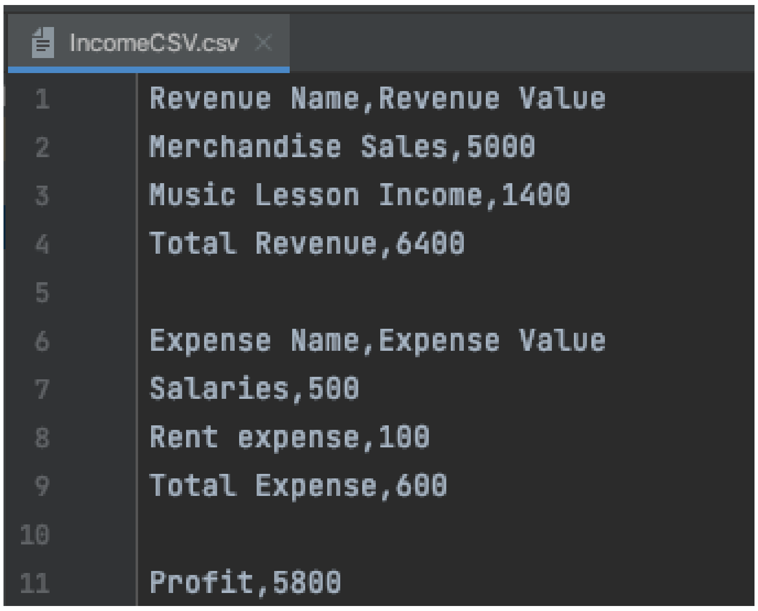
3.4. CSV Files

3.4.1. Income Statement



```
Total revenue: $6400
Total expense: $600
You have a profit of $5800
Do you want to export the data as a CSV file? [Y/N]: Y
```

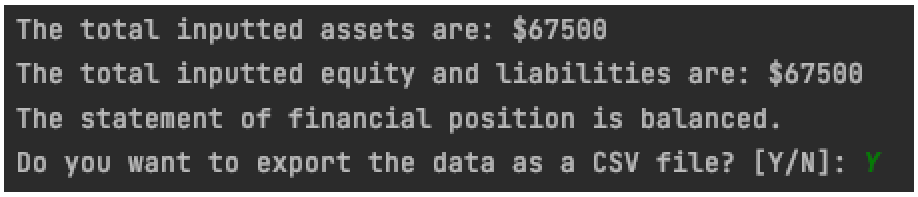
Image 3.16 IncomeStatement.py asking for CSV export



```
IncomeCSV.csv
Revenue Name,Revenue Value
Merchandise Sales,5000
Music Lesson Income,1400
Total Revenue,6400

Expense Name,Expense Value
Salaries,500
Rent expense,100
Total Expense,600

Profit,5800
```

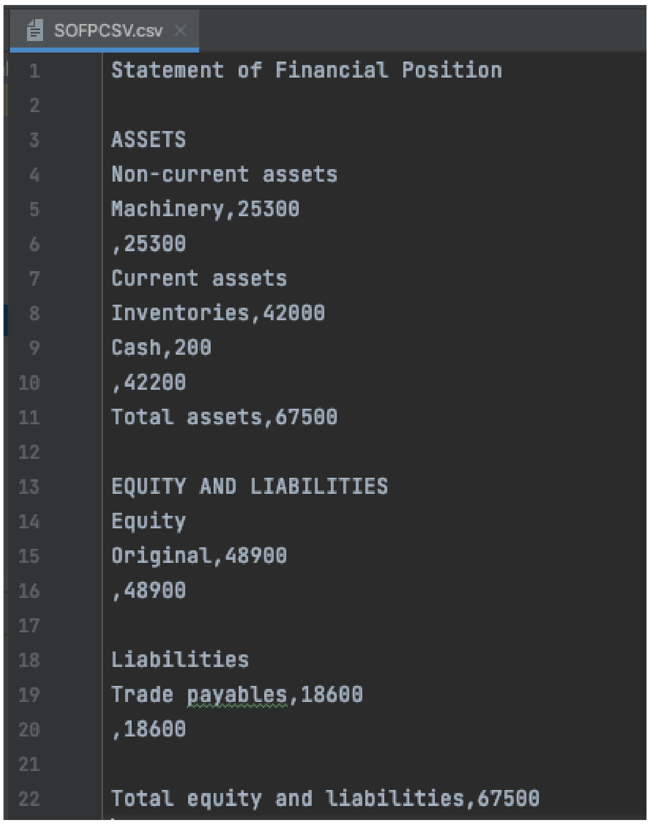
Image 3.17 Written CSV file for IncomeStatement.py

3.4.2. Statement of Financial Position



```
The total inputted assets are: $67500
The total inputted equity and liabilities are: $67500
The statement of financial position is balanced.
Do you want to export the data as a CSV file? [Y/N]: Y
```

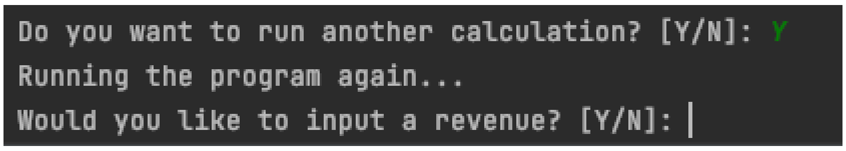
Image 3.18 SOFP.py asking for CSV export



```
SOFPCSV.csv
Statement of Financial Position

ASSETS
Non-current assets
Machinery,25300
,25300
Current assets
Inventories,42000
Cash,200
,42200
Total assets,67500

EQUITY AND LIABILITIES
Equity
Original,48900
,48900

Liabilities
Trade payables,18600
,18600

Total equity and liabilities,67500
```

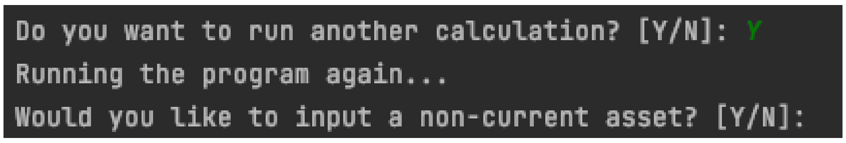
Image 3.19 Written CSV file for SOFP.py

3.5. Reset Program



```
Do you want to run another calculation? [Y/N]: Y
Running the program again...
Would you like to input a revenue? [Y/N]: 
```

Image 3.20 Reset program and run again for IncomeStatement.py

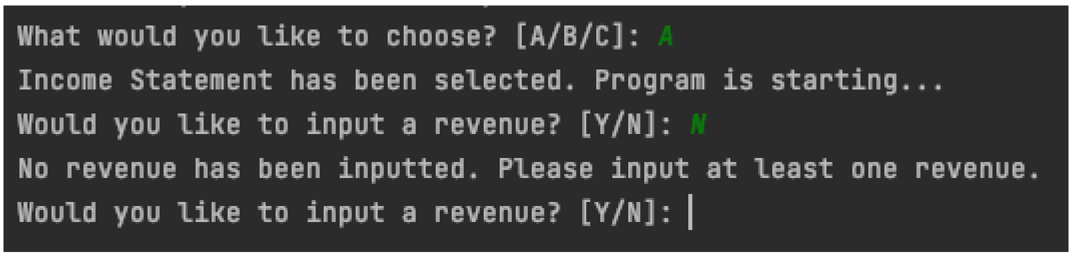


```
Do you want to run another calculation? [Y/N]: Y
Running the program again...
Would you like to input a non-current asset? [Y/N]:
```

Image 3.21 Reset program and run again for SOFP.py

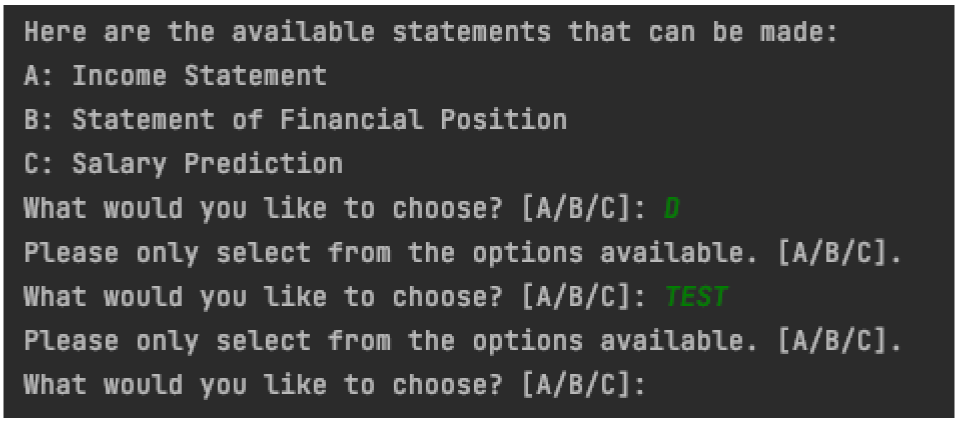
3.6. Validations

Validation remains constant throughout all the files so the screenshots will only be from one file instead of the same one from different files.



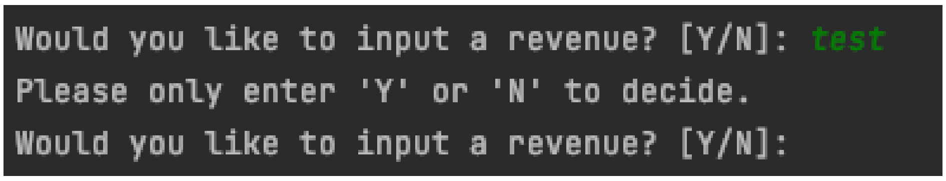
```
What would you like to choose? [A/B/C]: A
Income Statement has been selected. Program is starting...
Would you like to input a revenue? [Y/N]: N
No revenue has been inputted. Please input at least one revenue.
Would you like to input a revenue? [Y/N]: 
```

Image 3.22 Validation input only accepts A, B, C in Driver.py (lowercase accepted)



```
Here are the available statements that can be made:
A: Income Statement
B: Statement of Financial Position
C: Salary Prediction
What would you like to choose? [A/B/C]: D
Please only select from the options available. [A/B/C].
What would you like to choose? [A/B/C]: TEST
Please only select from the options available. [A/B/C].
What would you like to choose? [A/B/C]:
```

Image 3.23 Validation that all sections have at least one input (also in SOFP.py)



```
Would you like to input a revenue? [Y/N]: test
Please only enter 'Y' or 'N' to decide.
Would you like to input a revenue? [Y/N]:
```

Image 3.24 Validation to ensure that only Y and N are accepted (lowercase too)