Python Money Manager

Team 7

Sam Song, Vedika Jajodia, Nathan Carney, Justin Koehn, Darius Driggers

12/15/19

ISTM 3119 – Introduction to Programming

Table of Contents

User Interface and Operation Manual…….. 3

Problem Analysis…….. 7

Solution……. 7

Challenges……. 8

Main Expense Calculator…….. 9

[Code explained]

Email Notifications……13

[Code explained]

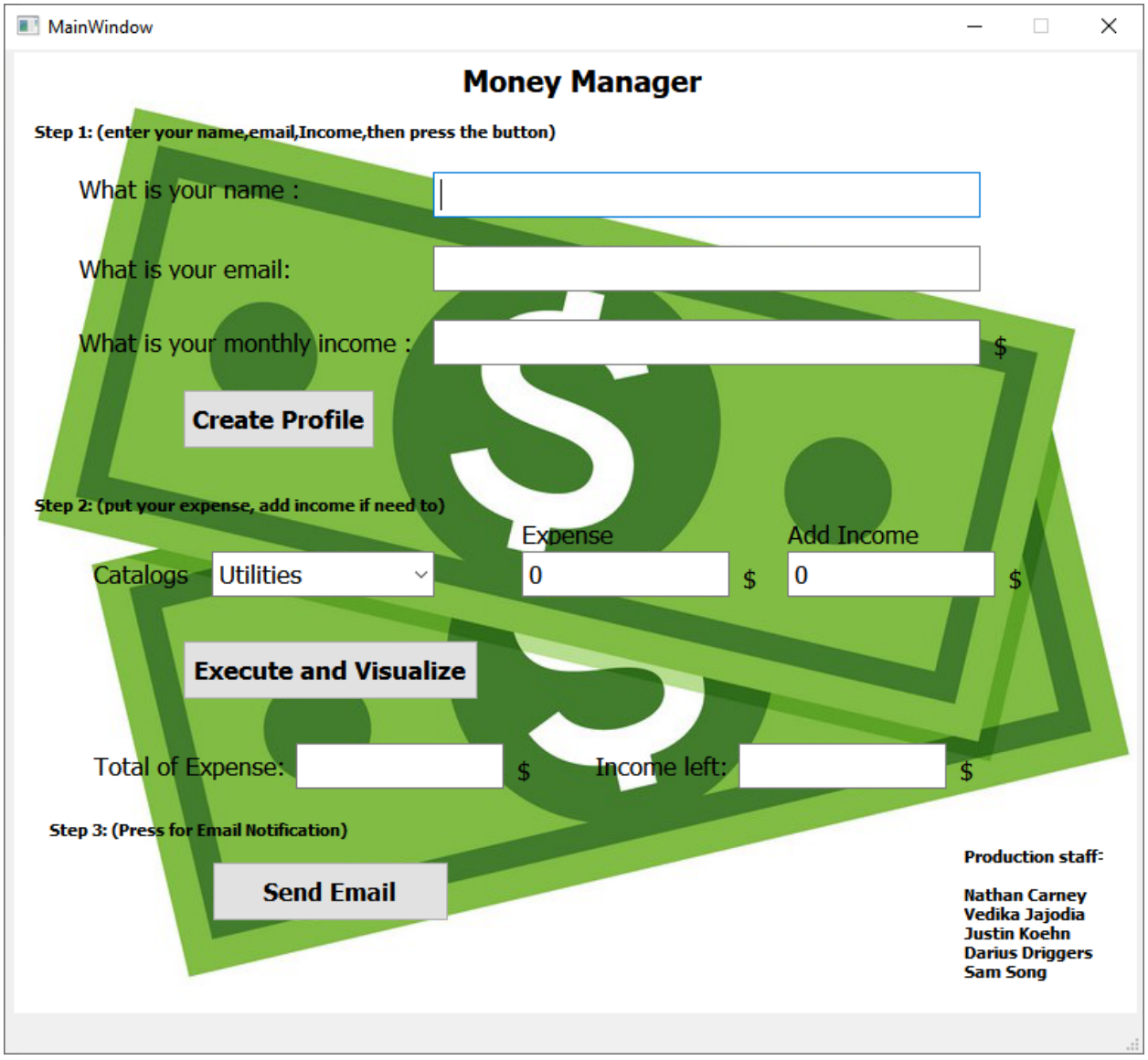
Graphs and Visualisations feature…….14

[Code explained]

Sources……...16

User Interface introduction & operation manual :

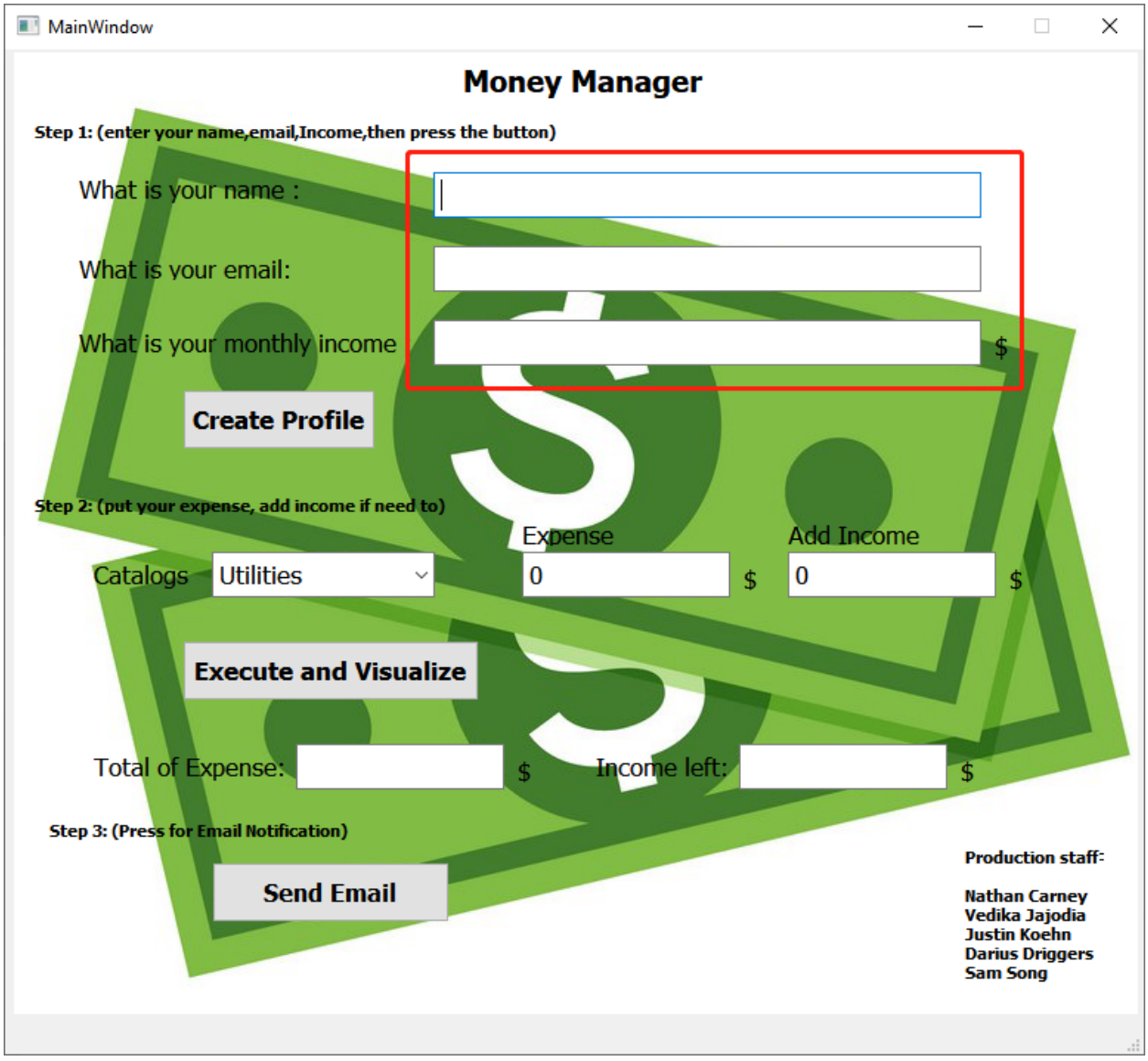
The app looks like the picture below:



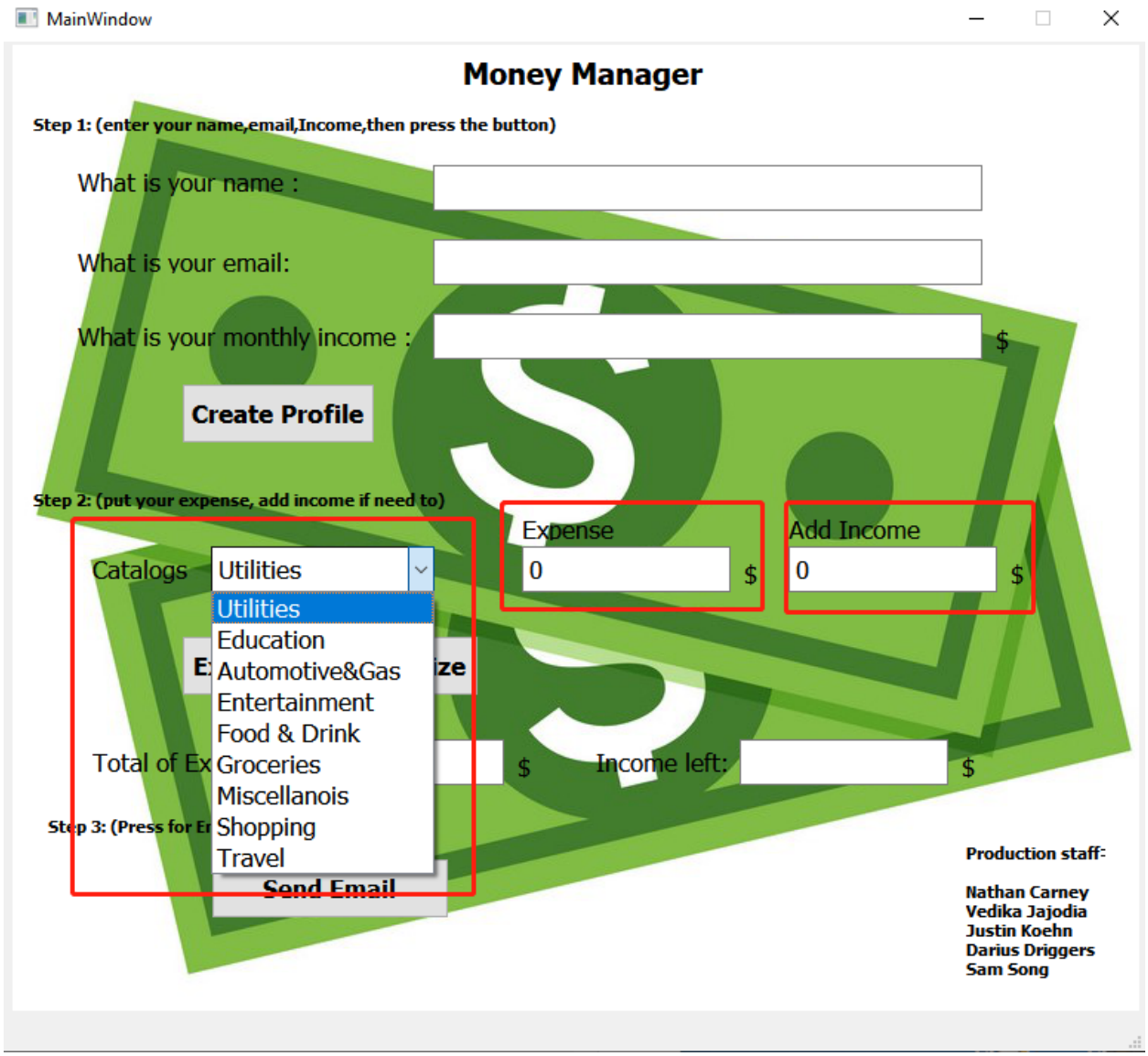
Input function are mostly in first step area,the program can send email report and warning window for spending surpass issue.

The operation are three steps to follow:

1. Enter the user's personal information(name,email,monthly-income) and press the Create Profile button.

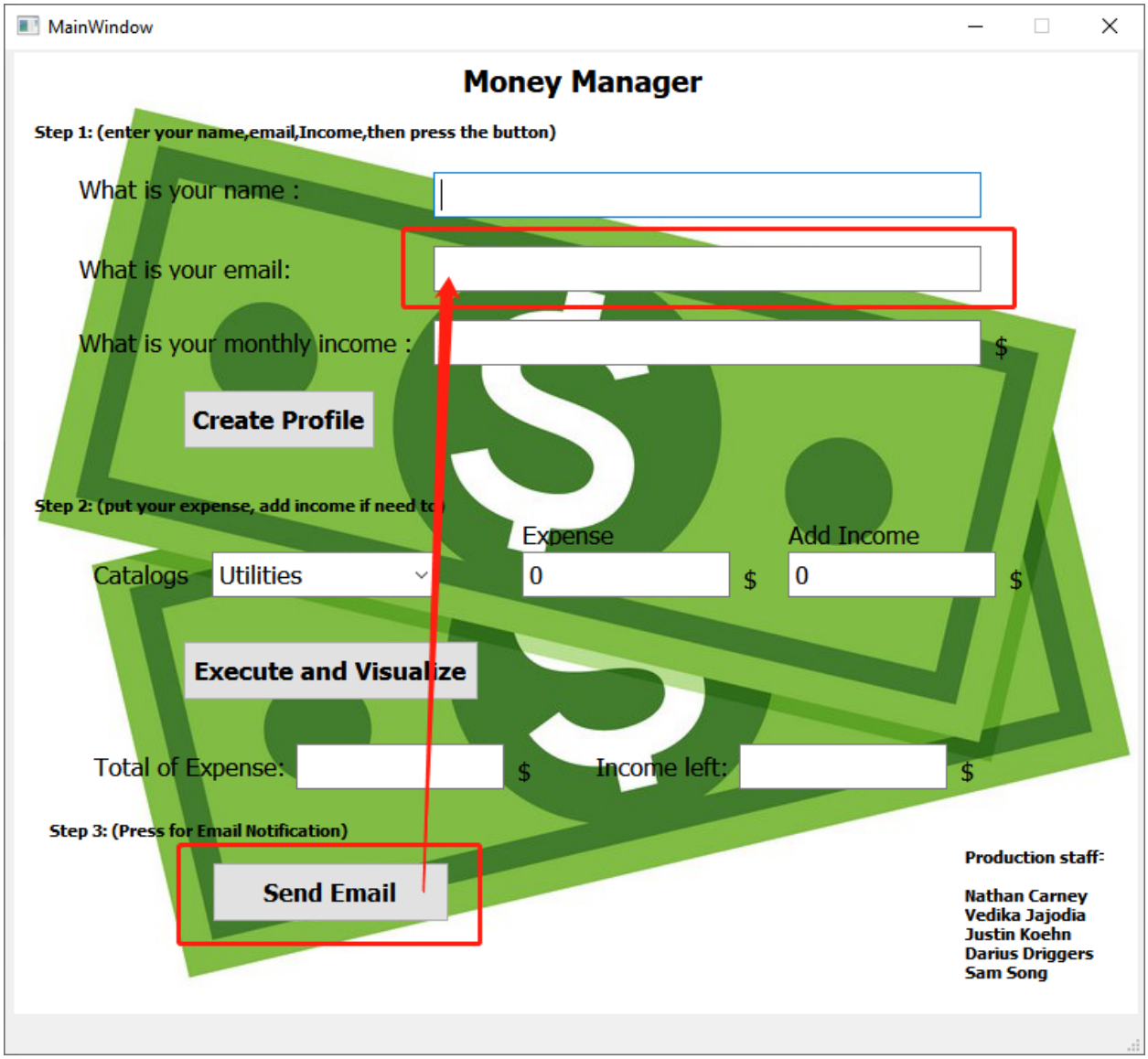


2. Choosing the Catalog for the expense, Enter the expense amount and add-income amount for money expenditure.after finish all the input, press button “Execute and Visualize” the picture will should in the console area.



3. Step three is an optional feature, If the user wants have email report,press button “Send Email”. The program will send an email to user’s email provided above.

The user will receive the report in their email-boxes.



The program has many other features like if user didn't enter name or monthly-income, there will a pop out window for reminder. Same as if user's expense is surpass their left income.

Problem Analysis

A recent survey conducted by Standard and Poor's Financial Literacy Branch has revealed some startling realities about the players in the world's most powerful economy. According to this 2015 report an astounding 43% of Americans are considered to be financially illiterate. Researchers found that this massive portion of the populace could not answer simple questions regarding debt, interest rates and inflation. We live in a time where Americans need this basic financial information more than ever. Four out of five workers are currently living paycheck. Their biggest money related challenges as revealed by a 2019 CFPB survey are "losing track of expenses", and "failing to stick to my budget." In the past decade consumer finance groups and NGO's have led the charge in spreading financial knowledge to the public but their efforts often fall on deaf ears. A large portion of these groups' funding has gone into academic papers and seminars many of which fail to capture the attention of the busy American. Our team research has revealed that consumers are looking for a product that integrates seamlessly into their daily life and allows them to take advantage of financial knowledge without having to read a textbook.

Solution

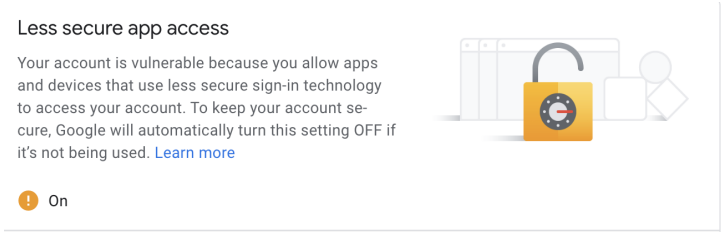
Our signature offering, The Python Money Manager, does just this. Our full service money management application allows users to input their expenses throughout the day right from their smartphone. Once a user has logged in they can directly enter

their expenses after they choose an expense category from the drop down menu. After the expenses are inputted users may visualize their spending in a fully colored pie chart that breaks their expenses down by category ranging from shopping to automotive. Users will also be able to see how much more they are allowed to spend for the day given their predefined budget parameters. Lastly, users will receive personalized emails that outline all of their spending. The Python Money Manager delivers unquestionable value to the average American and helps them tackle the challenges of budgeting and Finance in a simple fashion.

Challenges

A challenge with our program is that is not able to access the user's bank information and transactions. As a result, useful operation of the program relies solely on the user to manually input the income and expense information. Also, if there is any user error where monetary amounts are entered incorrectly, the subsequent expense report created will be inaccurate.

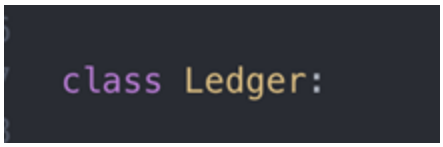
The send email button as originally a challenge for us for two reasons. First, the function was failing due to Gmail not granting the program access to send emails. In order to successfully bypass this, we had to allow third party apps to access our email account (pictured to the right). Secondly, after access to the account was granted, we were



getting an error that stated “SMTPServer Disconnected: connection unexpectedly closed”. We were able to fix this error by using a secure sockets layer (smtplib.SMTP_SSL) to keep the internet connection secure.

The main expense calculator code -

Class definition:



Firstly, we defined a class and each object (instance) of the class will serve as a Ledger account for an individual. The Python Money Manager has been designed such that multiple users/individuals can make their respective accounts and be able to track their spending. Every individual’s credits and debits in different spending categories can be kept track of in this way.

Init method:

The Init method is the constructor of the class Ledger. It is called every time a new object (instance) of the class is created. In the context of the project, it initializes

various attributes like name, category, income, spending_category etc. to their default value.

spending _categories:

The spending_categories variable is a dictionary associated with the project. If the user selects an input category type which is not present in spending_categories, the new category type will be automatically added to the dictionary.

The Python Money Manager has been conceptualized keeping a typical University student in mind and their expenses/income per day, per week and, eventually per month. Hence, the various relevant categories we are taking into consideration are:

1. **Automotive & Gas**
2. **Utilities**
3. **Education**
4. **Entertainment**
5. **Food & Drink**
6. **Groceries**
7. **Miscellaneous**
8. **Shopping**
9. **Travel**

Credit Method:

```
def credit(self, amount):  # credited amount
    self.income += amount
    self.surplus = self.income - self.Texpense
    return self.Texpense, self.surplus
```

We have taken into consideration that a user might have additional income during the course of their day/week/month in the form of refunds, cash backs, interest, etc.

The credit method adds the credited amount specified in the calling argument to the income. It further calculates the surplus left and returns the Texpense (Total expense) and surplus values (Income *less* Total expense).

Debit Method:

```
def debit(self, amount, category): #amount of one debit transaction, category of transaction

    if (amount > self.surplus):
        print("You have exceeded your monthly income")
        bankrupt = True
    else:
            self.Texpense +=amount
            self.surplus = self.income - self.Texpense

            if(category in self.spending_categories):
                self.spending_categories[category] += amount
            else:
                    self.spending_categories[category] = amount
            return self.Texpense, self.surplus;
```

The Debit method records the debit transaction performed by the user. The method initially checks whether the user can “afford” this transaction by comparing the transaction amount with the surplus of the user. If the transaction amount is greater than the user’s surplus, it notifies the user that he can’t afford this transaction.

If the transaction is possible, it calculates the remaining surplus and adds the amount to the respective category specified by the user. Essentially, keeps adding the dollar amount of the expense to the respective category ‘key’ in the dictionary (spending_categories). The dollar amounts act as ‘values’ in the dictionary.

Importantly, this method returns the Total expense amount incurred by the user and the surplus left of the Income left.

getTotalExpense Method:

```
def getTotalExpense(self):
    total_expense = self.Texpense
    return total_expense
```

This ‘Getter method’ returns the total expense incurred by the user.

getIncome() Method:

```
def getIncome(self):
    income = self.income
    return income;
```

This ‘Getter method’ returns the Total income of a user.

getCategoryExpense() Method:

```
def getCategoryExpense(self, category):
    string = "" + self.name + "'s spending for " + category+ " is " + str(self.spending_categories[category])
    return string
```

This ‘Getter method’ takes the category type as an argument and returns the total expense incurred by the user in that particular category.

getCategoryDic() Method:

```
def getCategoryDict(self):
    return self.spending_categories
```

The ‘Getter method’ returns the spending_category dictionary.

Email Notifications

```
def send_alert(self):
    import smtplib
    from email.mime.text import MIMEText
    from email.mime.multipart import MIMEMultipart
    #self.add_expense_and_income_bt()

    a = "Money Manager report for "+name+":"
    b = str(user.getCategoryDict())


    mylist =[a,b]

    fromaddr = "istm3119@gmail.com"
    toaddr = email
    msg4 = MIMEMultipart()
    msg4['From'] = fromaddr
    msg4['To'] = toaddr
    msg4['Subject'] = "Money Manager Alert"

    body = "\r\n".join(mylist)
    msg4.attach(MIMEText(body, 'plain'))

    server = smtplib.SMTP_SSL(host='smtp.gmail.com',port= 465)

    server.login("istm3119@gmail.com", "Python3119")
    text = msg4.as_string()
    server.sendmail(fromaddr, toaddr, text)
```

This function listed above is called when the “Send email” button is clicked by the user. In order for this function to work correctly, we first had to import SMTPlib. SMTPlib creates a Simple Mail Transfer Protocol session which is used to send emails to any valid email address. We then had to set up the server by listing the host address (smtp.gmail.com) and port number for the given host address. Once the server was set up, we provided the login information for the email address that would be used to send

the requested email alert to the user. Using the MIMEMultipart class from Python, we were able to define the “From”, “To”, and “Subject” email fields, and compose the body of the email with the dictionary that contained the user’s categories and expenses.

Graph and Visualization Feature

In terms of the graph and visualization code, our first step was to create a temporary variable called “temp” that stored the user spending categories dictionary. The user.spending_catagories dictionary was previously created and stores the available categories we want to plot. After initializing an empty variable and storing it as “plotdic”, we created a for loop. The for loop omits the categories that are equal to zero. This is done to remove the categories without any associated expenses from the pie chart. The associated expenses for the non-zero dictionaries are removed and stored in the “expense” variable so we can utilize them to create the pie chart. In addition, the dictionary keys for the non-zero dictionaries are used to create the correct labels for the pie chart (see code below).

Code:

```
#Create Pie Chart to show only categories that are non zero
        temp=user.spending_categories
        plotdic = {}
        for x, y in temp.items():
            if y != 0:
                plotdic[x] = y
        labels=plotdic.keys()
        expense=plotdic.values()
```

After the necessary variables are defined and stored, we utilized the online graphical package Matplotlib to execute our graph. The first Matplotlib command we used was the “ax1.pie” command. To specify the correct input, we entered the “expense” variable that we created above as the first for “ax1.pie” command. The “expense” variable stores the user input values that are stored in the user.spending_catagories dictionary. The next argument for “ax1.pie” is responsible for the labels of the graph. Because we had already created a labels variable above, we set the second argument equal to our “labels” variable. The “ax1.pie” command also has a function called “autopct” which converts our inputs into percentages. Then we used the “ax1.axis” command, with the argument “equal” to ensure that the pie was drawn as a circle with equal parts. Finally, we displayed the graph using the “plt.show” command (see code below).

Code:

```
        fig1, ax1 = plt.subplots()

        ax1.pie(expense, labels=labels, autopct='%1.0f%%',
                shadow=False, startangle=90)
        ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

        plt.show(fig1)
```

Sources

Consumer Financial Protection Bureau. (n.d.). Retrieved from https://www.consumerfinance.gov/.

Standard & Poor's: Americas. (n.d.). Retrieved from https://www.standardandpoors.com/en_US/web/guest/home.

Kazarinoff, P. D. (n.d.). Bar Charts and Pie Charts. Retrieved from https://problemsolvingwithpython.com/06-Plotting-with-Matplotlib/06.06-Bar-Charts-and-Pie-Charts/.

Student, K. M. F., KattamuriMeghna, & Student, F. (2018, May 14). Python: Introduction to Matplotlib. Retrieved from https://www.geeksforgeeks.org/python-introduction-matplotlib/.

(n.d.). Retrieved from https://www.youtube.com/watch?v=XkNItvjjVc4&list=LLT2OblTLllLq4aVCn-xWaVw&index=5&t=1050s.

Dharmkar, R. (2018, January 12). What is difference between self and __init__ methods in python Class? Retrieved from https://www.tutorialspoint.com/What-is-difference-between-self-and-init-methods-in-python-Class.

Qt Designer Manual. (n.d.). Retrieved from https://doc.qt.io/qt-5/qtdesigner-manual.html.