Beginner Budget Tracker Documentation

Miguel Ruiz, Laraib Zia, Sarwat Kazmi

Project Concept

Our project is a budget tracker that allows a user to conduct various calculations and produces visualizations related to their financial records based on the information they've entered. The goal of this project was to help users get started with budgeting and saving when it comes to their finances.

Repository Files and Descriptions

`spending.py:`

This file contains code for running the program by first gathering financial information through user input, followed by performing calculations to display a breakdown of the user's financial information through graphs and charts

`spending_test.py:`

This is the test script for `spending.py` which contains unit tests for the following methods:

- `init`
- `spend_check`
- `interest`
- `spending_allocation`
- `expenses`
- `habits`

`expense_sheet.csv:`

This CSV file is a record of the user's variable and fixed expenses created after the user has entered their financial information through the `'user_expenses'` method.

`finances.docx:`

This document is a record of the user inputted information that has been checked to ensure that the user is entering the correct numbers with the associated spending records.

Running Beginner Budget Tracker from the Command Line

1. Using the '`argparser`' library for a command-line parser, the only required argument is the user entering their total yearly income after the name of the script. Total income must be a float type.
2. The user is then asked to enter the name of a fixed expense that month followed by the cost of that fixed expense, also a float type.
3. Once the user is done entering any fixed expenses, they may type "done" when prompted for the name.
4. Repeat steps 2 and 3 for any variable expenses.
5. Once each type of expense is entered, they will be added to a respective 'fixed' and 'variable' dictionaries, and subsequently converted to a pandas dataframe object. The

dataframe produces, `'expense_sheet.csv'`, containing each expense type. The user will be able to view the DataFrame in the terminal as well as in the CSV file.

Using and Interpreting Beginner Budget Tracker

`spend_check()`: this method reports how much the user has spent on their combined fixed and variable expenses each month, and checks whether their spending is above or below their monthly earnings. Monthly earnings are computed as `total_income/12`.

`input_info()`: this method prompts the user to enter the name of a loan, the principal amount, the interest rate per year, the duration of the pay period for the loan in months, and finally whether it's compounded or simple interest. Results in each loan entered and associated total amount of interest that will be paid by the end of the loan period.

`expenses()`: this method generates a bar graph depicting the user's percentage of their total yearly income that goes towards their fixed and variable expenses.

`input_habits()`: following `expenses()`, the user is prompted to enter a spending habit (i.e. smoking cigarettes), and how much they spend on that habit, and once they're done entering habits, they can type 'done'. If they don't want to enter any habits, they can type 'none'. The inputted habits are added to a dictionary, and a pie chart is generated, representing how much each habit costs yearly, how much out of the user's total income it accounts for, and the total amount of money that could be saved if the user cut down on certain habits.

The program's final output is a Microsoft Word Document containing all of the above results.

Written Testing Procedure

`user_expenses()`:

First prompt: 'FIXED expenses, name?' *`// Asking user for a fixed expense (i.e. rent)`*

Entering any **string** will move the user forward to the next prompt. Entering any other variable will ask the user to re-enter the fixed expense.

Second prompt: 'How much? (dollars and cents)' *`// Asking user for associated cost of fixed expense`*

Entering a **float** will move the user forward to the next prompt; however, entering an **int** is acceptable and will also move the user forward. Again, entering any other value will prompt the user to re-enter their information.

The program will cycle between both questions; once the user wishes to move to the next phase, they can then type '**done**' and the program will then repeat this same procedure for the variable expenses.

Typing '**done**' at any point will move the user forward to the next set of questions, but take note that without at least one entry of fixed expenses and one entry of variable expenses, the **spend_check()**, **spending_allocation()**, and **expenses()** methods will not function. The following message will appear when this happens:

'Please enter both your fixed and variable expenses to run this program.'

If the user types '**done**' when asked to enter both expenses, the following message will appear:

'Please enter FIXED and VARIABLE expenses to continue, goodbye.'

Then the program will then terminate.

`input_info()`:
******Entering '**skip**' at the initial prompt will allow the user to skip this method. Any other entry will allow the user to proceed.**

First prompt: 'Enter NAME of loan: ' *`// Asking user for loan type (i.e. student loans)`*
Entering any **string** will move the user forward and any other variable will ask the user to re-enter their information.

Second prompt: 'Enter the PRINCIPAL (dollars and cents): ' *`// Asking for a principal amount`*
Entering any **float** will move the user forward and entering an **int** is also acceptable. Any other variable will return the user to the first prompt where they will have to restart and re-enter their information.

Third prompt: 'Enter the RATE (% of the year): ' *`// Asking for the interest rate`*
Entering any **float** will move the user forward and entering an **int** is also acceptable. Any other variable will return the user to the first prompt where they will have to restart and re-enter their information.

Fourth prompt: 'Enter the TIME (months): ' *`// Asking for the time period involved`*
Entering any **int** will move the user forward. Any other variable will return the user to the first prompt where they will have to restart and re-enter their information.

Fifth prompt: 'Enter SIMPLE or COMPOUND: ' *`// Asking for the type of interest`*
The user must enter either '**SIMPLE**' or '**COMPOUND**' (case-sensitive) to move forward. Any other words or any other variable will return the user to the first prompt where they will have to restart and re-enter their information.
Entering '**SIMPLE**' will return a value and end the method; entering '**COMPOUND**' will prompt the user to pick between 3 more options.

Final prompt: ‘Enter MONTHLY, QUARTERLY, or YEARLY: ‘ `// Asking for the pay period of the loan`

The user must enter either ‘**MONTHLY’, ‘QUARTERLY’,** or ‘**YEARLY’** (case-sensitive) to move forward. Any other words or variables will return the user to the first prompt where they will have to restart and re-enter their information.

`input_habits()`:
First prompt: ‘Enter NAME of habit (one word): ‘ *`// Asking user for a habit (i.e. smoking)`*
Entering any **string** will move the user forward but any other variable will ask the user to re-enter their information.

Second prompt: ‘How much do you spend monthly to supplement this habit? (enter dollars and cents): ‘ *`// Asking user how much they spend on the habit`*

Similar to the input in **user_expenses()**, the program will prompt the user to enter a dollar amount that corresponds to their listed habit.

Entering a **float** will move the user forward and entering an **int** is also acceptable. Any other value will prompt the user to re-enter their information.

The program will continue to ask the user to enter their habits and can end the requests by typing ‘**done**’. The user can also skip this part of the program by typing ‘**none**’ at the first prompt.

Annotated Bibliography

Reference to pie chart created in `Graphs` *class method,* `habits()`:

- "Basic Pie Chart." *Matplotlib 3.2.1 Documentation*, matplotlib.org/3.2.1/gallery/pie_and_polar_charts/pie_features.html.

Reference to how to use the `'docx'` *library to write to a Microsoft Word file:*

- "Docx." *Python*, python-docx.readthedocs.io/en/latest/.

Reference to bar chart created in `Graphs` *class method,* `expenses()`:

- "Matplotlib.pyplot.subplots." *Matplotlib 3.2.1 Documentation*, matplotlib.org/3.2.1/api/_as_gen/matplotlib.pyplot.subplots.html.

Reference to `'pyplot'` *in unit tests in* `Graphs` *class methods:*

- DiNardo, Vince DiNardoVince. "Pyplots in Unit Tests." *Stack Overflow*, 1 Mar. 1968, stackoverflow.com/questions/51429465/how-to-mock-pyplot-show-in-python-to-prevent-showing-plots.

Reference to testing inputs:

- "Testing inputs in Python3." *Andressa.dev*, https://andressa.dev/2019-07-20-using-pach-to-test-inputs/