CS-500 Module Nine Project

Isaac Raymond

CS-500-11462-M01

Dr. Omar Hamdy

Python Personal Budgeting Application

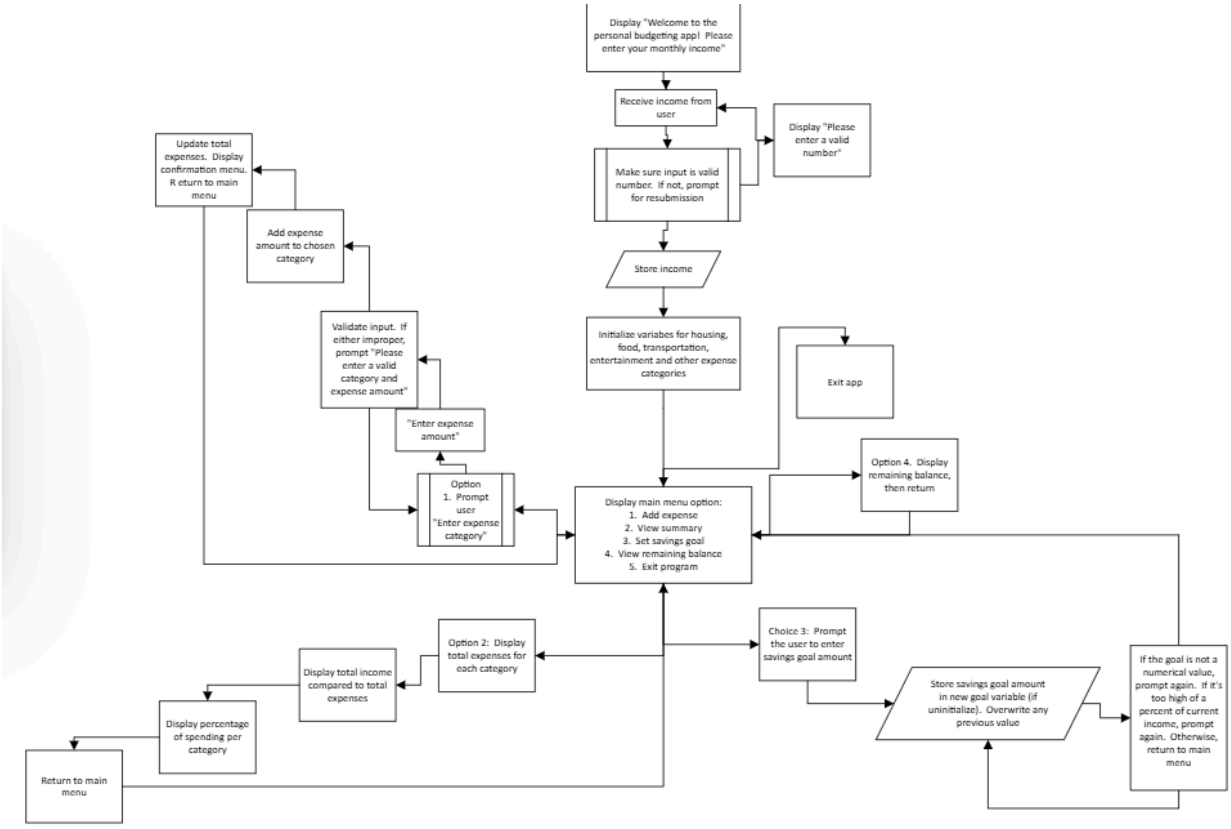
I've used Python to create a console based personal budgeting application. The program performs the basic functions of a budgeting application by asking the user to input their income and expenses, then allowing the user to make some basic budgeting planning. The program uses object oriented programming principles to create reusable and easily maintainable code. The program demonstrates introductory knowledge on input handling and data validation. It is an example of code that an introductory programmer can write, such as myself!

The program was written in a fashion. The principles of object oriented programming were put into practice. The user can store information about their income, expenses, and their goal for their budget. The most used of these functions was "get_and_validate_float".

User input has to be validated each time that it occurs, so a separate function was written to validate this input. The function "get_and_validate_float" is used throughout the code in order to make sure the next value entered by the user is a positive float. This abstraction reduces the amount of code needed as each user's numerical input is validated. It improves the reusability of the code and makes the program more modular. Any issues I had receiving user input for floats could be easily investigated as they all occurred in this function.

The object oriented programming principle of encapsulation was on display with three primary functions implemented for the user to use. The function "add_expense" allows the user to add a particular monthly expense amount to a particular expense category. The "set_income" function was used so that the user could easily adjust their income, should it change in the future. The "view_summary" function allowed the user to quickly get a summary of their budget situation.

The user's interaction with the program was encapsulated in one area of the code. The "main_menu" function is where the program receives user input, directing the user to other modules in the program. This separation of the code further emphasizes the principle of encapsulation. This can be seen in the flow chart of the program's operations: The main_menu function is at the heart of the program.



The code is meant to emphasize some basic coding standards. The function names are written in an easy-to_read fashion with names that describe the purpose of the function. Lowercase and underscores were used in the name. The code is split into functions, where each function performs one primary task. The functions are written with as little code as possible in order to complete its task, yet with enough code to make sure the function is easily read and interpreted by another programmer.

As the program is limited in its appearance and function (A console only program), only Python's built in libraries were used. The program uses Python's built-in input() and print() functions to accept user input and to display information to the console. Python's built in error handling, try/except, was used to accept only valid input and reject input that would otherwise cause the program to crash. Python's built-in dict data type was used to store income, the user's goal, and the expenses as properties of the budgeting app class.

Secure programming practices were emphasized in the project. The previously mentioned "get_and_validate_float" method ensures that the user is entering an acceptable float for the program to continue running. Also used was the "safe_input" function which checks to make sure user input is not beyond the program's memory requirements. This ensures that the program run on a variety of machines. The use of encapsulation also prevents erroneous modification of data outside the class. The values can only be modified by the methods inside the class.

The try/except blocks of code in the "add_expense" and "get_and_validate_float" methods ensure that the user is only entering a number. This prevents unintended program crashes and demonstrates proper error handling in Python code. This prevents the program from crashing if the user enters a typo, such as "20000a", or if the user enters a comma "2,000".

Per the assignment instructions, a separate unit testing file was not created. Only manual testing and validation was conducted. The program verifies the integrity of information input by

the user.

```
def main():
    app = BudgetApp()
    print("Welcome to the Personal Budgeting
        Application!\n")
    print("Testing BudgetApp...")

    app = BudgetApp()

    app.set_user_income()
    print("Income set to:", app.income)
```

```
Welcome to the Personal Budgeting Application!

Testing BudgetApp...
What's your monthly income? sdfasf
Your input is not a number. Please enter a numeric value.
What's your monthly income?
```

The program also protects the system from the entry of a value that is too large, again demonstrating input validation and defensive programming:

```
def main():
    app = BudgetApp()
    print("Welcome to the Personal Budgeting
        Application!\n")
    print("Testing BudgetApp...")

    app = BudgetApp()

    app.set_user_income()
    print("Income set to:", app.income)

    app.add_expense()
    print("Expenses:", app.expenses)
```

```
Welcome to the Personal Budgeting Application!

Testing BudgetApp...
What's your monthly income? sdfasf
Your input is not a number. Please enter a numeric value.
What's your monthly income?
    99999999999999999999999999999999999999999999999999
    99999999999999999999999999999999999999999999999999
    99999999999999999999999999999999999999999999999999
    99999999999999999999
Input too long — please keep responses reasonable.
What's your monthly income?
```

The program successfully uses the functions written in the code to perform different functions, such as displaying the user's current balance.

```python
def main():
    app = BudgetApp()
    print("Welcome to the Personal Budgeting
        Application!\n")
    print("Testing BudgetApp...")

    app = BudgetApp()

    app.set_user_income()
    print("Income set to:", app.income)

    app.add_expense()
    print("Expenses:", app.expenses)

    print("Remaining budget:", app.view_balance
        ())

    print("All manual tests completed.")


if __name__ == "__main__":
    main()
```

```
What's your monthly income? sdfasf
Your input is not a number. Please enter a numeric value
    .
What's your monthly income?
    99999999999999999999999999999999999999999999999999
    99999999999999999999999999999999999999999999999999
    99999999999999999999999999999999999999999999999999
    999999999999999999999999
Input too long — please keep responses reasonable.
What's your monthly income? 2000
Income set to $2000.00

Income set to: 2000.0

Expenses:
1. Housing
2. Food
3. Transportation
4. Entertainment
5. Other
Choose a category number: 3
Enter monthly amount for Transportation: 200
Added $200.00 to Transportation.

Expenses: {'Housing': 0.0, 'Food': 0.0, 'Transportation'
    : 200.0, 'Entertainment': 0.0, 'Other': 0.0}

Your leftover cash after expenses is: $1800.00

Remaining budget: None
All manual tests completed.
```

This project demonstrates introductory knowledge of Python, as well as introductory knowledge of object-oriented programming, secure coding practices, and basic software development standards. The code was developed in a modular fashion. This enhances the code's readability and therefore it is easily maintained. The code validates all input made by the user. The program works in a console only, but could easily be upgraded with a graphic user interface. Overall, the program demonstrates the core concepts of programming through a basic Python console program.