The Financial Calculator Project

Overview

During this course, you have learned a huge amount about computer science and Python programming in general. In this project, you will design, plan, and implement a medium-to large-scale final project involving financial literacy. You can work individually, or with a partner. To enhance the efficacy of the financial application, you will also learn how to read and write data from and to a CSV (Comma Separated Value) file.

Think about the most recent times when you used a calculator. When and why did you need to use a calculator? What are some financial scenarios where we need to record and analyze lots of data?

Here are some possible candidates for your project:

- Student loan calculator and tracker
- Holiday budget planner, currency convertor, and expense tracker
- Personal budget and expense/income tracker
- Post-secondary "Dream School" financial cost calculator and analyzer and planner

When a scientist or financial analyst is focused on crunching numbers and analyzing results, they prefer skipping the graphical interface, and working with data that is created/provided in a simple text format. One popular input data format is the CSV (Comma Separated Value) file that can be created by a spreadsheet program like excel. Python has a built in CSV parser library that we can use. It is also common practice in business applications to interface data with a program like Excel, which is why the CSV format is convenient. Learning about formatting the output, such as displaying numbers in 2 decimal places, would also be helpful in this project. Here are some resource links:

https://realpython.com/python-formatted-output/
https://realpython.com/python-csv/

Details

1. Project Phases

This project will be significantly larger in scope than any of your previous assignments. Rather than being given a well-defined specification, *YOU* will be setting the requirements for your project by coming up with an idea, fleshing out the details, and defining the steps

necessary to complete your program. To help you through this process, there will be several steps to this project. You must complete all steps for your project to be successful. In fact, *half* of your grade will be based not on how well your program works, but on how well you completed the design and planning process.

The project includes these applied design phases:

1. *Understanding Context* - Conduct user centred research to understand design opportunities and barriers
2. *Ideating* - Establish point of view, generate and prioritize ideas, analyze competing social, ethical, and sustainability considerations; list out user stories and features
3. *Prototyping* - Choose an appropriate form, scale, and level of detail for prototyping multiple ideas; analyse the design for the life cycle, and construct prototype, making changes to tools, materials, and procedures as needed; record iterations of prototyping
4. *Testing* - Identify feedback, develop an appropriate way to test prototype, collect feedback and critically evaluate design and makes changes; iterate or abandon design idea
5. *Making* - Task project management processes (task management schedule) to work collaboratively to coordinate production, ensuring that your software goals are met
6. *Sharing* - Share progress; design how to promote product with end-users; critically reflect on design thinking and processes, identify new design goals and future work

Software development process is iterative and agile. The actual coding is not done until the **Making** phase. Before the designer dives into coding, it is crucial to spend enough time flushing through the idea and doing some sketches of what using the program will look like.

The **Prototyping** phase is meant to be quick and efficient, using materials that are easily available. This can be a pencil and paper sketch of a flowchart, and sketch of the software interface, what things will look like when the client uses the software. What does input (and data format) does it prompt from the user? What output will it generate?

The **Testing** phase here refers to testing the prototype design, and not seeing if your code works. It means getting others (possibly potential users) to look at your prototype and see if its usage makes sense. Give the tester a scenario, and ask them what they would do, and then tell them what would happen next. The tester's response and feedback should be observed and used to make a new iteration of the design.

2. Progress Tracking

In the **Prototyping** phase, you will complete the Final Project Organizer. You can skip Part 2 if you choose not to implement classes in this project. After the **Testing** phase, you will create a final project schedule using the Final Project Development Plan. These documents will be your guides in the development **Making** phase and will help you stay on track and aware of your progress. Throughout the development phase of the project, you will be expected to keep your spec and plan up-to-date and adjust as you get ahead or behind, as requirements change, or as tasks or features get re-prioritized. At the end of each coding day, your spec and plan documents should be updated to reflect the current state of your project, and you will check in with an instructor at least once a week to make sure things are on track.

3. Implementation Requirements

Complexity and Creativity

One of the main goals of this project is to allow you to unleash your creativity and allow you to create something of interest to you. To achieve this, your project must show some level of creativity or personalization that makes it your own. Simply creating your own version of some existing application will not fully meet this requirement. For both the complexity and creativity requirements, you should check in with the teacher early and often to ensure your project is in line with our expectations.

Documentation and Style

Your program must be well-written, well-documented, and readable. Writing code with good style is always a good idea, but in a project of this size and scope, following style guidelines will help you keep your thoughts organized and make it easier to keep track of your progress, pick up where you left off each day, and find and fix bugs. In particular, though this is certainly not a comprehensive list, pay attention to the following:

- Organizing your functions/code so that they can be read and comprehended easily
- Giving your functions, variables, lists, and dictionaries descriptive and meaningful names
- Using the right data type (`string`, `int`, `bool`, `float`, `dictionary`, `class`) for each situation
- Include comments to describe the structure of your program and track your progress
- Avoiding redundancy with good use of loops, functions, and/or lists, and/or dictionaries, and/or classes
- Practicing good procedural decomposition and abstraction

Required Python Elements

1. A clear way to start the program, and clear prompts or instructions for any user interaction
2. At least one loop, variable, function, and list, and more as necessary or appropriate. Each of these must be used correctly and meaningfully
    - creating a list that contains a single element just to meet this requirement will not earn points
3. At least one user interaction - this can be prompting for information using ask, responding to key presses or mouse movements, or any other action that keeps the user involved
4. Use of the CSV file format to store data

Required Checkpoints

At least three times during the project period, you should check in with your teacher to ensure that your project is on track, that you are meeting the project requirements, and that you have the answers to any questions that might have arisen during your work. Your teacher work with you to set up a schedule for these checkpoints, but it is **your responsibility** to ensure that the meetings take place.

Suggested Pacing Plan

- Day 1 - project introduction, team formation (if working with partner), brainstorm, prepare some survey or interview questions to help validate your calculator idea.
- Day 2 - prototype and testing, sketch out solution and input/outputs, do a small example of what the input CSV would look like, and what the output would look like; get feedback from peers
- Day 3 - start of learning how to read and write data with CSV files, re-visit your design to see that it still make sense, ie
    - Is it easy and convenient for the user?
    - Does the sequence of events make sense?
    - Is the output meaningful and useful?
- Day 4-6 – implement the program according to your plan; test small amounts of code at a time and revisit the initial design plan to see if everything still makes sense for the user
- Day 7 - share on the presentation day