Assignment 2: Pair Programming

Assignment Handout

Due date: Sunday, March 14 by **11:59pm**
This assignment is done **in pairs.** Note that you should have created your pairing before starting the assignment. You can reuse your pairing from A1 but will need to create a new group in Quercus for this assignment anyway. You may lose up to 5% if you don't sign up for your pair properly as instructed. If you don't have a pair use Piazza to find someone to work with. If you are having difficulty finding a pair, please email us at csc301-2021-01-assignment@cs.toronto.edu **ASAP.**

Overview

In this assignment, you will focus on some core elements of the **process** of producing software. By completing this assignment, you will work on:

1. Pair programming
2. Testing
3. Clean Coding

Starter Files/GitHub Repo

You will be submitting your assignment entirely on GitHub.

You and your partner must join a pair on Quercus by navigating to the 'People' tab, and clicking 'Assignment 2 Pairs' (similar to what you did in Assignment 1). The number that you have chosen will be your assigned pair number
You must either create a repository or join an existing repository that your partner has created.
Please use the following team name structure:
<pair-number>-<member1-github-id>-<member2-github-id>.
The words "assignment2-pair" will be prepended automatically to your repo name. You must create your team repository using the following link:
https://classroom.github.com/g/6YuMEGKk

You will use this repository for assignment 2 only. **Note that this repository is not related to your project or assignment 1**. Note that the template (.md document) for running the assignment will be pre-loaded in your assignment. You will be submitting your assignment entirely on GitHub.

You will be responsible for ensuring that your repo is up to date (with your submission on the **master** branch) at the time the assignment is due. You will be responsible for any delays introduced if you do not join the course GitHub org in time.

Pair Programming

Pair Programming is a common practice in software engineering as described in the lecture on "Agile". You can set up remote pair programming as well.
In this assignment, you will perform **a prescribed version** of pair programming for **two features** of the program. You only have to pair program as prescribed for **these two features.** For the rest of the features, it is up to you how you want to finish them with your partner. Here's how we'd like you to do pair programming:

- Decide roles (driver/navigator) and the feature to be developed (they should be big enough to spend 2 to 3 hours thinking about implementation and then coding) . Together break down the feature to multiple "checkpoints" and design the solution on paper. Estimate how long it will take you to finish this feature with the proposed design. This design will be the guide in coding your solution. Once you are happy with your design, the driver (using their own GitHub account) starts coding the solution while the navigator watches and helps whenever needed (no checking phones or going to get coffee). After the allocated time is done (could be 30-90 minutes, you can take a break and switch roles). Repeat this enough times until the feature is done. Experiment with different variations to see what works best for you. You can then move on to the second feature and **switch starting roles.**
    - This includes the **tests** for the features you chose - make sure they are pair programmed as well.
- In your **`README.md`**
    - Should explain which features were pair programmed
    - Should explain who the driver and navigator was for different parts of the features
    - Should give a **reflection** on how it went, and what you liked and disliked about this process
- The commits should reflect who the driver was for each feature
    - **Any** work done on a particular feature should **only** be done by the driver for this feature - we will check!

Specification: COVID Monitor

For this assignment, you will be creating an interface (API or CLI) for COVID-19 data**.**
There is not much starter code to start with - you will have to take the specifications of the app, and use the concepts from class to write a **Python program** to make it work.
You will also write an **interface** to demonstrate the functionality of your app. This can be done in two ways:

- Using a Flask API: If you go with this option, you can use the starter code provided as your starting point
- Using CLI: If you go with this option, you can simply start from scratch and treat it as a standalone Python app that you can call from your terminal.

Either way, you need to write clear instructions of how to run your flask server and work with it on the browser.

Going off of our theme that Software Development is not just coding, you will notice that the interface itself is not that difficult. We know that you can probably hack something together quickly that more or less works most of the time.
However, that is not what professionals do. You will have to think about and reflect on why you chose certain elements, including:

- How to represent objects
- The relationships between objects (coupling, cohesion)
- Design of functions
- Using **design patterns effectively**
    - You can use the ones we mentioned in class, or others
- Clean coding practices

The Starter Code

The starter code is meant to be set up as a **Flask** project in your IDE. You can start by running **main.py** in the CovidMonitor folder**,** which will set up a simple server similarly to the ones shown during lectures and tutorials. The starter code also comes with a file called **unit_tests.py** in the tests folder. You can use the commands described in the Readme.md file included in the starter code.

There is only one Python file in the starter code, `main.py`, which will have the main method that will start the program. You will have to add to it to fulfill the requirements of the program below (you can replace the contents of the main method entirely as well). We will run your code from this main method, so do not get rid of it.
The rest of the design is up to you!
You should make effective use of **design patterns and unit tests** that we talked about in class. You will be graded on program design!

Requirements

Your program needs to be able to do the following **while the app is running** (we do not require external persistence in files apart from what's listed below).

1. Add a new data file: You can assume the file is consistent and follows the format specified here. You can use one of the files in this GitHub repo. The files have two possible formats:
    - Time Series: You should implement this first. The user should be able to send a csv file to your programs as a data source. You should be able to parse and store the data. Please note that you are not asked to create a database; however, doing so, can be an additional functionality.
    - Daily reports: These are more detailed files that take in the data one day at a time. Your program needs to be able to take this information in as well.

    You can choose how you want to expose your application to run and the endpoints for your application. With both of these methods, you need to send a success/failure response following the standard.
2. Update existing files: If a file is uploaded again, you should update the data you store to reflect the new information
3. Query data by one or more countries, provinces/states, combined_keys: The data can be any of the following:
    - Deaths
    - Confirmed
    - Active
    - Recovered

The data above can be requested for any one day or a period of time.

4. Your program should support returning the data in multiple formats:
    - JSON
    - CSV
    - Text (printed)
    - **Bonus:** You can use libraries like Matplotlib to plot the returned data instead.

Testing

You will design tests (unit and integration) to test your code functionality.
It is up to you how you test your code, but in order to receive full marks, you should test with **at least 75% line coverage** of the repo with **meaningful tests and test names as discussed during lectures.** You can learn more about code coverage here.

Code Craftsmanship

You must have good programming and formatting style in your code. You are free to use tools like Linters, IDE tools, etc. to help you. Mention in your README.md which tools you used to help you create clean code.

Handing in your work

We will grade the most recent version (and look at the commit history) of your submission on the **master branch** of your pair's repo. You may lose marks if your master doesn't have your latest updates.
Make sure everything is updated there, including your README.md.

Grading

You will be graded on the following criteria.

- **Pair Programming** (15 marks):
  - Your process explained in README.md
  - Your commit history reflecting your process explanation
  - Your reflection about your process, including positives and negatives
- **Program design** (20 marks)
  - The design patterns you chose to use
    - The description of why you chose to use them for your implementation
  - Relationships between objects: code cohesion, coupling
  - Function design
- **Functionality** (25 marks): We will test it and look at your code to see your implementation of the required features.
- **Tests** (20 marks)
  - Thorough testing of the required features
  - At least 75% line coverage of the repo with meaningful tests, test names, and assertions.
- **Code Craftsmanship** (10 marks)
  - Programming and formatting style is good
  - Let us know what tools you used (Linters, IDE tools, etc.)
- **README.md explanation and organization** (10 marks): Your README should be well formatted and organized. The TA should be able to use your app easily based on the instructions you provide. You may lose marks if your TA cannot easily run your application.