Assignment 1 – Copilot Foundations: Testing and Optimizing

Context

Given that you all likely have differing experience levels with utilizing Copilot (or similar) tools, the first few assignments will serve to give you baseline experience and help you understand what you can do with the tool before we begin our larger assignments. This one will focus on test-driven development and code optimization.

Part 1: Starting with the desired output (10 points)

In this section, we'll work backwards and show that sometimes it can be better to begin with how you want the end result to look and work backwards from there. This can be especially helpful for complex assignments where you aren't quite sure where to begin. A description of what you need to do is below:

You are provided with a **test suite** that validates three key functions:

1. **Cleaning and structuring raw staff email data** – The data contains typos, extra characters, and inconsistencies that need to be corrected.

2. **Generating birthday and work anniversary messages** – The system should format and send messages based on properly structured data.

3. **Handling edge cases where no messages should be sent** – Ensuring the program only sends messages when appropriate.

Your task is to write the necessary Python functions to ensure **all tests pass**. You should use AI tools to assist you and should be manually writing very little code.

Part 2: Rapid test creation (10 Points)

In your jobs post-graduation, a great deal of your time will be spent on the unglamorous part of coding that is writing tests. For nearly every update to the codebase that you make, you will have to implement corresponding test cases, which can often take longer than the actual work you set out to do. This can be an excellent use of Copilot and AI tools generally to both decrease your time spent outside of problem solving and quite possible achieve a higher level of test coverage. A description of what you need to do is below:

You are provided with a **pre-written API** that handles user authentication. This API allows users to:

1. **Register** (PUT /register) – Store a username and hashed password.
2. **Authenticate** (GET /authenticate) – Check if a username/password pair is valid.
3. **Update Password** (PUT /update-password) – Change an existing user's password.
4. **Deactivate Account** (PUT /deactivate) – Prevent a user from logging in.

Your task is to use AI-assisted tools like GitHub Copilot to **generate a test suite** that ensures this API functions correctly under various conditions.

Using the VS Code extension Python Test Explorer, run your tests with coverage. The coverage of this file should be >90% given the content to ensure proper functionality.

Part 3: Optimizing / restructuring existing code (10 Points)

Despite what we all imagine when we become software engineers, not all of your work will be focused on implementing the coolest, shiniest new functionality. In fact, it is entirely possible that most of your time will be spent modernizing and optimizing existing code. Especially at large, legacy companies, this code can be a complete mess and there may be no one left who actually wrote it initially. Therefore, it is very important to be able to parse through code and understand what's happening in order to improve it. Thankfully, AI can do this much more efficiently than we can. A description of what you need to do is below:

You are given a **fully functional but extremely inefficient and unreadable Python program**. This code contains:

- Deeply nested conditionals that are hard to follow.
- Unnecessary recomputations and performance issues.
- Hardcoded logic instead of reusable functions.
- Terrible variable names and zero documentation.

The program processes a dataset of numbers and fruit prices while performing various calculations and transformations. However, due to poor design, it suffers from slow performance, difficult-to-maintain logic, and redundant computations.

Your task is to **refactor and optimize the code** so that:

• It is modular, with clear function responsibilities and reusable logic.

• It improves runtime performance by reducing redundant computations.

• It enhances readability, making it easy to understand and maintain.

• It eliminates unnecessary deep nesting, making decisions clearer.

• It includes proper variable naming and documentation.

The final program should provide users with specific fruit-related information, including:

• An alphabetical list of available fruits and their prices

• The cheapest available fruit (best value)

• The most expensive available fruit (worst value)

• The total cost of a given quantity of different fruits (you choose)

• Any other insights you think would help someone shopping for fruit with this data

Submitting

To submit this assignment, ensure that your updated code is pushed to GitHub Classroom.

Additionally, include a screenshot of your test coverage from section 2 on the Canvas assignment.