Python Coding Guide Assignments

NOTE: Apprentice should follow the criteria and complete the number of questions mentioned.

1. SOLID Principles

Criteria: Complete all the questions. No need to store any data. Just play around with objects

- **[Single-Responsibility Principle (SRP)]** Implement a simple program to interact with the library catalog system. Create a Python class **Book** to represent a single book with attributes: Title, Author, ISBN, Genre, Availability (whether the book is available for borrowing or not). Create another Python class **LibraryCatalog** to manage the collection of books with following functionalities:
  - Add books by storing each book objects (Hint: Create an empty list in constructor and store book objects)
  - get book details and get all books from the list of objects

  Lets say, we need a book borrowing process (what books are borrowed and what books are available for borrowing). Implement logics to integrate this requirement in the above system. Design the classes with a clear focus on adhering to the Single Responsibility Principle(SRP) which represents that **"A module should be responsible to one, and only one, actor."**

- **[Open-Closed Principle (OCP)]** Download the python file from this link. Suppose we have a Product class that represents a generic product, and we want to calculate the total price of a list of products. Initially, the Product class only has a price attribute, and we can calculate the total price of products based on their prices.

  Now, let's say we want to add a discount feature, where some products might have a discount applied to their prices. To add this feature, we would need to modify the existing Product class and the calculate_total_price function, which violates the Open/Closed Principle. Redesign this program to follow the Open-Closed Principle (OCP) which represents “Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification.”

- **[Liskov Substitution Principle (LSP)]** Download the python file from this link. In this file, there is an implementation of a banking system for account handling. There is a savings account and a checking account class. The checking account inherits the savings account as both have the same functionality and the checking account allows overdrafts (allow processing transactions even if there is not sufficient balance). Redesign this program to follow the Liskov Substitution Principle (LSP) principle which represents that **“objects should be replaceable by their subtypes without altering how the program works”.**

- **[Interface Segregation Principle (ISP)]** Download the python file from this link. Suppose we have an interface called PaymentProcessor that defines methods for processing payments and refunds. Then we have a class called OnlinePaymentProcessor that implements the PaymentProcessor interface. However, some parts of our system only need to process payments and do not handle refunds. Redesign this program to follow the Interface Segregation Principle (ISP) principle which represents that **"Clients should not be forced to depend upon methods that they do not use. Interfaces belong to clients, not to hierarchies."** (Hint: Create two different classes in which one class use interfaces for process payment and another class can process and refund payment both)

- **[Dependency Inversion Principle (DIP)]** Download the python file from this link. Suppose we have a NotificationService class that is responsible for sending notifications. The NotificationService class directly depends on the EmailSender class to send emails.

  In this implementation, the NotificationService class directly depends on the EmailSender class, which violates the Dependency Inversion Principle. The high-level NotificationService should not depend on the low-level EmailSender, as it tightly couples the classes together.

  Redesign this program to follow the Dependency Inversion Principle (DIP) principle which represents that **"Abstractions should not depend upon details. Details should depend upon abstractions."**

2. Design Patterns

Criteria: Complete all the questions

- **[Factory Design Pattern]** Build a logging system using the Factory Design Pattern. Create a LoggerFactory class that generates different types of loggers (e.g., FileLogger, ConsoleLogger, DatabaseLogger). Implement methods in each logger to write logs to their respective destinations. Show how the Factory Design Pattern helps to decouple the logging system from the application and allows for flexible log handling.

- **[Builder Design Pattern]** Design a document generator using the Builder Design Pattern. Create a DocumentBuilder that creates documents of various types (e.g., PDF, HTML, Plain Text). Implement the builder methods to format the document content and structure according to the chosen type. Demonstrate how the Builder Design Pattern allows for the creation of different document formats without tightly coupling the document generation logic.

- **[Singleton Design Pattern]** Implement a configuration manager using the Singleton Design Pattern. The configuration manager should read configuration settings from a file and provide access to these settings throughout the application. Demonstrate how the Singleton Design Pattern ensures that there is only one instance of the configuration manager, preventing unnecessary multiple reads of the configuration file.

3. Coding conventions

Criteria: Complete all the questions

- Create a Python program that manages student records. The program should have the following functionalities:
  - Create a function that can add new students to the records with their student_id, name, age, and grade. The records should be saved to “json” file and each time new record is added, it should be saved to same “json” file
  - Allow searching for a student by student_id or name. The data should return age and grade from the saved file.
  - Allow updating a student's information by using student_id or name(age or grade)

  Ensure to follow PEP8 Coding Guidelines for following criterias:

  - Proper Indentation
  - Maximum Line Length
  - Prescriptive Naming conventions (Package and Module Names, Class Names, Exception Names, Global Variable Names, Function and Variable Names, Method Names and Instance Variables, Constants)
  - Source File Encoding while accessing the JSON file
  - Add NumPy Docstring to each function

4. Unittest

Criteria: Complete all the questions

- Create a function that validates email addresses based on following set of rules:
  - Proper email format such as presence of “@”, no space in the address
  - Presence of valid email providers such as yahoo, gmail and outlook. Make sure there are no no disposable email providers such as yopmail.

  Write unit tests to validate different email addresses against the rules, including valid and invalid addresses (Use unittest module).

- Design a function that takes a list of numerical data and performs calculations for mean, median and standard deviation. Write unit tests to verify the correctness of the statistical calculations for various inputs, including edge cases like an empty list or a list with one element (Use unittest module).

5. Debugging

Criteria: Complete all the questions

- Download the python file from this link. It is an implementation of an e-commerce system where products can be added, customers can be created and then products can be added to the cart. Find the cases where the logic to handle adding or removing products from cart fails. Handle these cases after debugging.

  Note: Use Python Debugger (pdb) to debug the whole code with the necessary “breakpoint()” function. Use different syntax such as where (w), next (n), step (s), continue (c), print (p) etc.

  (Hint: Negative quantity should not be allowed to add to cart and non present product should not be allowed to remove)

- Similar to the previous question, debug the given python file from this link. It is an implementation of a food ordering system.

  Note: Use vscode “Run and Debug” with necessary breakpoints.

  (Hint: Negative quantity should not be allowed and non present food item should not be allowed to remove)

6. Linting and code formatting

Criteria: Complete all the questions

- Use **“pylint”** module to identify and fix the problems in the scripts from previous assignments mentioned below:

  - [Day 2] Intermediate Level Python Programming: Check and fix all problems from the topics: **Exception Handling, File I/O, Object Oriented Programming**

  - [Day 3] Python Coding Guide Assignments**:** Check and fix all problems from topic “**Design Patterns**”

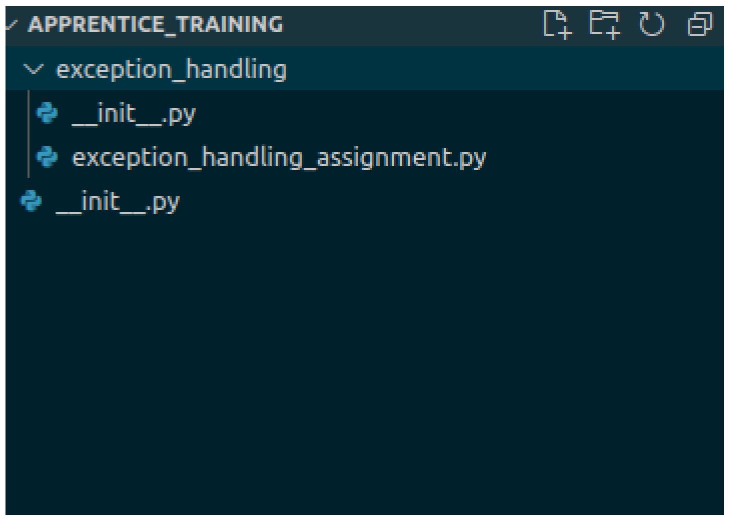
- Create a package containing the below mentioned scripts from previous assignments. Each assignments should be individual package

  - **Exception Handling, File I/O, Object Oriented Programming** from Day 2

  - **Design Pattern** from Day 3

  Use Flake8 to check all the errors in this package and fix all the problems.

Note: Create package as shown in below screenshot.



```
APPRENTICE_TRAINING
  exception_handling
    __init__.py
    exception_handling_assignment.py
  __init__.py
```