

Part 1: Object-Oriented Student Management System

Overview

This assignment requires you to develop a Student Management System (SMS) using Python and Object-Oriented Programming (OOP) concepts. The project is designed to help you understand Python functions, modules, OOP principles (encapsulation, inheritance), exception handling, and file storage.

Project Objectives

By completing this assignment, you will learn how to:

- Apply **OOP principles** such as **encapsulation, inheritance, and modularity**.
 - Implement **Python functions, modules, and exception handling**.
 - Develop a **menu-driven system** to manage student records.
 - Use **file storage** to persist student data.
-

Project Structure

Your project should follow the structure below:

```
student_management/  
|-- main.py # Entry point for the application  
|-- student_operations.py # Contains core functionalities  
|-- database.py # Stores student records  
|-- validation.py # Validates student data  
|-- exceptions.py # Custom exceptions  
|-- storage.py # File handling operations  
|-- models/  
|   |-- student.py # Student base class with encapsulation  
|   |-- undergraduate.py # Undergraduate subclass
```

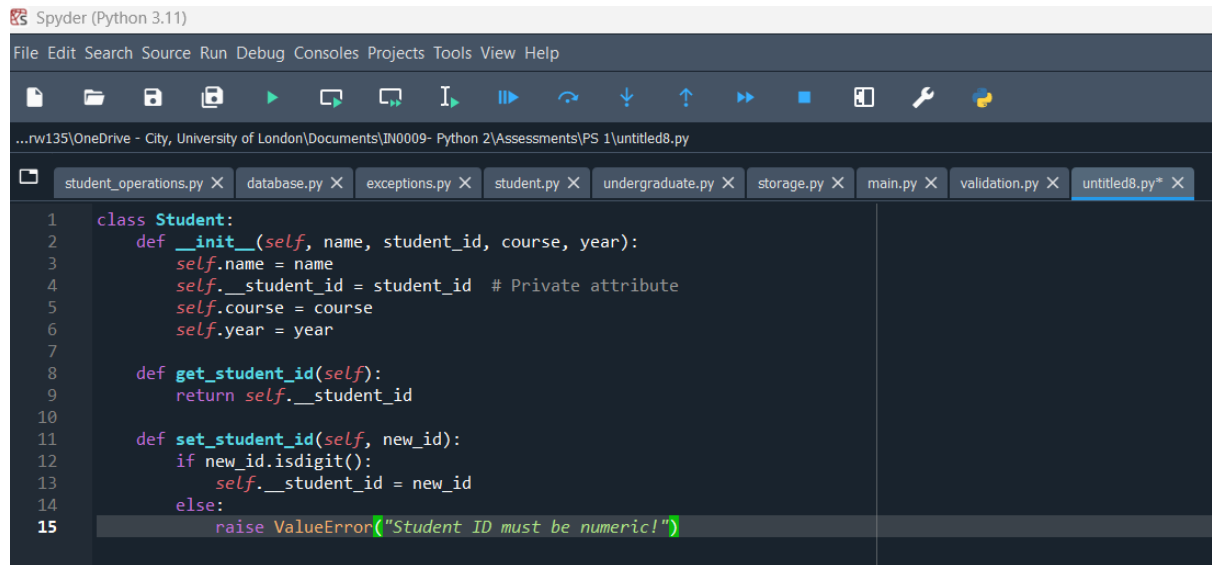
Task 1: Implement Student Class with Encapsulation

Requirements:

- Create a Student class with **encapsulation**.
- Define **private attributes** (`__student_id`) and provide **getter** (`get_student_id()****`) and **setter** (`set_student_id()****`) methods.

- Implement **methods** for updating course and retrieving student details.

Hint: Refer to **Week 3 - OOP Basics** for encapsulation principles.



```

1  class Student:
2      def __init__(self, name, student_id, course, year):
3          self.name = name
4          self.__student_id = student_id # Private attribute
5          self.course = course
6          self.year = year
7
8      def get_student_id(self):
9          return self.__student_id
10
11     def set_student_id(self, new_id):
12         if new_id.isdigit():
13             self.__student_id = new_id
14         else:
15             raise ValueError("Student ID must be numeric!")

```

Task 2: Implement Inheritance in an Undergraduate Class

Requirements:

- Create an Undergraduate subclass that inherits from Student.
- Add an **extra attribute** (minor) to Undergraduate students.
- Override the get_details() method to include minor specialisation.

Hint: Refer to **Week 4 - OOP Inheritance** and use super() to extend Student.

Task 3: Develop Student Management Operations

Requirements:

- Implement CRUD operations (**add, delete, update, and list students**) in student_operations.py.
- Use the validate_student_id() function before adding students.
- Store students in a **global list (students****)**.

Hint: Refer to **Week 2 - Python Functions & Data Structures** for handling lists and modular functions.

Task 4: Implement Exception Handling

Requirements:

- Create **custom exceptions** (InvalidIDException, DuplicateStudentIDException) in exceptions.py.
- Use exception handling in student_operations.py to prevent errors from breaking the program.

Hint: Refer to **Week 5 - Error Handling & Exceptions** and use try-except blocks to catch and handle exceptions gracefully.

Task 5: Implement File Storage for Student Records

Requirements:

- Use storage.py to **save and load student data**.
- Store student records in a **JSON file (students.json****)**.
- Ensure data persistence when the program restarts.

Hint: Refer to **Week 7 - File Handling & Data Storage**. Convert student objects to dictionaries before saving them to a JSON file.

Task 6: Create an Interactive Menu-Driven System

Requirements:

- Implement a main.py script that provides a **menu-based interface**.
- Users should be able to **add, delete, update, and list students** interactively.
- Ensure **error handling** for invalid inputs.

Hint: Refer to **Week 1 - Python Basics & User Interaction**. Use a while True loop with a user input-driven menu for interactive execution.

Submission Requirements

* Complete Python scripts for all modules.

- * Encapsulation and Inheritance correctly implemented.
- * File handling and Exception handling are properly used.
- * Menu-driven interface functional.
- * Commented and well-documented code.

This document provides the detailed implementation tasks and requirements for your Student Management System assignment, along with hints referencing weekly materials to guide your approach.