Name: Mohiuddin Khan
Date: November 26, 2025
Course: Python 100 (Foundations of Programming: Python)

Assignment 07 – Working with data classes (Person & Student), functions, JSON files, properties, inheritance, and structured error handling

Introduction

In this assignment, I built on the course registration program developed in earlier modules by integrating several object-oriented programming techniques explored in Module 07. Specifically, I learned how to use constructors to initialize class objects, designed data classes for better data management, implemented properties with validation for encapsulation, and applied inheritance to create a clear relationship between Person and Student classes. Throughout this process, I relied on the Mod07 lecture notes, the assignment guidelines, and the "Python Programming for the Absolute Beginner" textbook by Dawson to guide my understanding of how object-oriented features can make Python programs more organized and maintainable.

The main objective of Assignment 07 was to create two new data classes—Person and Student—using inheritance to demonstrate that a Student is a specialized form of Person. I updated the FileProcessor and IO classes so they could handle Student objects effectively, enforced property validation to protect data integrity, and continued to follow the Separation of Concerns principle by keeping each class's responsibilities distinct. In the sections below, I'll walk through the key steps I followed and highlight relevant portions of my code to illustrate how these concepts were put into practice..

Steps I Took to Complete the Assignment

1. Designing Data Classes and Implementing Constructors

To begin with, I reviewed how Python classes act as blueprints that bundle attributes and methods together. A constructor (__init__) initializes an object's data, ensuring each object begins in a valid and predictable state. Based on this, I created the '**Person'** class, which stores and validates a first and last name.

The validation rules required alphabetic characters only, which I handled through property setters. This approach protects the integrity of the data and keeps the Person class responsible only for managing person-related information.

Excerpt from the Person Class

```
class Person:
    def __init__(self, first_name: str = "", last_name: str = ""):
        self.first_name = first_name
        self.last_name = last_name

    @property
    def first_name(self) -> str:
        return self.__first_name.title()

    @first_name.setter
    def first_name(self, value: str) -> None:
        if value.isalpha() or value == "":
            self.__first_name = value
        else:
            raise ValueError("The first name should not contain numbers or symbols.")
```

This structure allowed my Person objects to consistently maintain validated, well-formatted data.

2. Building the Student Class Using Inheritance

Next, I extended the Person class to create a **Student** class. Module 07 emphasizes using inheritance to build subclasses that reuse existing functionality while adding new, specialized features. Student inherits all characteristics of Person and introduces an additional attribute: '**course_name'**.

The Student constructor calls the Person constructor using super (), and I added a property setter to validate the course name. I found this 'super ()' in the textbook, and it's logical to use this function here. This ensures that anyone registering for a course cannot submit an empty or invalid entry.

Excerpt from the Student Class

```
class Student(Person):
    def __init__(self, first_name: str = "", last_name: str = "", course_name: str = ""):
        super().__init__(first_name=first_name, last_name=last_name)
        self.__course_name = ""
        if course_name != "":
            self.course_name = course_name

    @property
    def course_name(self) -> str:
        return self.__course_name

    @course_name.setter
    def course_name(self, value: str) -> None:
        value = value.strip()
        if value == "":
            raise ValueError("The course name cannot be empty.")
        self.__course_name = value
```

Using inheritance streamlined the class structure and allowed Student objects to behave exactly like Person objects with additional capabilities.

3. Applying Separation of Concerns with FileProcessor and IO Classes

A major goal of this assignment was to continue using the Separation of Concerns pattern, which keeps the program well-organized. Instead of mixing input, output, validation, and file processing together, each responsibility is handled by its own class.

3.1. FileProcessor Class

Handles reading and writing JSON files and converting rows into Student objects.

3.2. IO Class

Manages user prompts, displays information, and handles formatting and output messages.

Excerpt – Reading Student Data from JSON

```
@staticmethod
def read_data_from_file(file_name: str, student_data: list) -> list:
    try:
        file = open(file_name, "r")
        list_of_dict_rows = json.load(file)
        student_data.clear()

        for row in list_of_dict_rows:
            student = Student(first_name=row["FirstName"],
                              last_name=row["LastName"],
                              course_name=row["CourseName"])
            student_data.append(student)

    except Exception as e:
        IO.output_error_messages("Error reading the file.", e)
    finally:
        if file is not None and not file.closed:
            file.close()
    return student_data
```

Excerpt – Registering Student Input

```
@staticmethod
def input_student_data(student_data: list) -> list:
    try:
```

```
            student = Student()
            student.first_name = input("Enter the student's first name: ").strip()
            student.last_name = input("Enter the student's last name: ").strip()

            course = input("Please enter the name of the course: ").strip()
            if course == "":
                raise ValueError("The course name cannot be empty.")
            student.course_name = course

            student_data.append(student)

        except Exception as e:
            IO.output_error_messages("Error: There was a problem with your entered data.", e)

    return student_data
```

This organization keeps input/output logic separate from data handling and processing, making the program more professional and easier to maintain.

4. Tested Major Assignment Requirements

To make sure I met all the requirements listed in Mod07-Assignment, I performed the following tests:

a. The program takes the user's input for a student's first, last name, and course name.
b. The program displays the user's input for a student's first, last name, and course name.
c. The program saves the user's input for a student's first, last name, and course name to a JSON file. (checked this in PyCharm or a simple text editor like Notepad or TextEdit.)
d. The program allows users to enter multiple registrations (first name, last name, course name).

e. The program allows users to display multiple registrations (first name, last name, course name).
f. The program allows users to save multiple registrations to a file (first name, last name, course name).
g. The program runs correctly in both **PyCharm and** from the **console or terminal**.

5. Adding Comments and Code Clarity

Throughout the script, I focused on readability and clarity, which is emphasized both in the module content and in the textbook.

Specific things I did:

- Used a clear header block at the top with title, description, and change log.
- Added section comments so someone reading the code can quickly see the structure.
- Chose descriptive variable names like students, student_data, course_name, and menu_choice instead of single letters.
- Used consistent casing for dictionary keys ("FirstName", "LastName", "CourseName") to match the guidance in the notes about key names and JSON compatibility.
- Printed user-friendly messages for both normal operations and errors, keeping technical info in a clearly labeled “Technical Error Message” section.

This matches the general best practices: comment “why”, use readable names, and structure the code with logical sections.

6. Documentation and Submission

After finishing and testing the program, I completed the documentation and submission steps described in Mod07-Assignment:

a. **Knowledge Document (this file)**
- Wrote this document following the professional template and section headings.
- Described what I did, showed key code sections, and explained how the program meets the requirements.

b. **GitHub Repository**

- Created a public repository named **IntroToProg-Python-Mod07**

(link: https://github.com/KhanEnv/IntroToProg-Python-Mod07**)**.

- Uploaded:
    - Assignment07.py
    - Enrollments.json with sample data
    - This knowledge document (as a PDF).
    - Several knowledge based discussions in a separate file in PDF.
- Committed the changes so the instructor and classmates can review the work.

c. **Canvas Submission**

- Placed the Python file, the JSON file, and this document into a folder called A07.
- Zipped the folder and uploaded the .zip file to the Canvas Assignment07 page.

d. **Discussion Board (for review)**

- Posted the GitHub link to the "Assignment 07 Documents for Review!" discussion board so peers can access it for informal review.

Summary

Working through Assignment 07 really opened my eyes to how powerful object-oriented programming can be when building real applications. I got hands-on experience designing custom classes with constructors, adding properties for data validation, and using inheritance to make my code more flexible and organized. By following the Separation of Concerns approach, I was able to keep the student data, user interaction, and file processing each in their own neat sections of the program.

As a result, my course registration program feels much more solid and professional. The different parts work together smoothly, and making changes is less stressful because everything has its place. Learning these concepts feels like a big step toward writing code that's not just functional, but maintainable at a software engineering level.

Citations

- Assignment07.py (M. Khan, 2025).
- Module 07 Notes – "Functions, Classes, and Separation of Concerns."
- Module 07 Assignment Instructions.

- Dawson, M. (2010). *Python Programming for the Absolute Beginner* (3rd ed.). Course Technology PTR.
- Python Built-in Functions: https://docs.python.org/3/library/functions.html