Greg McCaslin

March 8, 2025

IT FDN 110 A

Assignment 07

<https://github.com/gmc5/UW_Python_Winter_2025_07>

Creating Assignment 07

# Introduction

This assignment continues using a version of the course registration program which demonstrates the use of data classes with structured error handling. It defines a Person class with validation for first and last names and a Student class that extends Person by adding a course name. The program provides a menu for users to register students, display current registrations, and save data in a JSON file. The FileProcessor class handles file operations, while the IO class manages user interactions and input validation. The main loop continuously presents the menu until the user chooses to exit, ensuring proper data handling and error messaging throughout.

# Getting Started in Programming My 7th Assignment

I created an A07 folder where my assignment files reside (Figure 1.1). I also compared it to the assignment06 starter code to my assignment07 code.

A screenshot of a computer program

AI-generated content may be incorrect.

**Figure 1.1: Snapshot of assignment artifacts**

# Installing the Header

As directed in the homework module, I installed the header with the required information. I used hashtags to comment on the information so it does not get executed as code. (figure 1.2)

*# ------------------------------------------------------------------------------------------ #  
# Title: Assignment07  
# Desc: This assignment demonstrates using data classes  
# with structured error handling  
# Change Log: (Who, When, What)  
# GMcCaslin,3/8/2025,Created Script  
# <Your Name Here>,<Date>,<Activity>  
# ------------------------------------------------------------------------------------------ #*

**Figure 1.2: Header information entered the script**

# Importing of JSON Library

It was recommended that when working with JSON, to import its library at the beginning of the script. This way, the library will already be available when called upon. (figure 1.3)

A black background with white text

AI-generated content may be incorrect.

**Figure 1.3: Importing of JSON library**

# Setting the Constant Values

I set all required constant values for the code with their type (which should never change). (figure 1.4)

*MENU: str = '''  
---- Course Registration Program ----  
 Select from the following menu:   
 1. Register a Student for a Course.  
 2. Show current data.   
 3. Save data to a file.  
 4. Exit the program.  
-----------------------------------------   
'''  
FILE\_NAME: str = "Enrollments.json"*

**Figure 1.4: Constants defined in the script**

Define the Data Variables

For this assignment, some of the assignment06 variables are no longer required, leaving just 3. (figure 1.5)

*students: list = [] # a table of student data  
menu\_choice: str = "" # Hold the choice made by the user.  
  
student\_table: list = []*

**Figure 1.5: The variables used in the script**

# Person initialization and validation class

The Person class defines a model for storing and validating first and last names. It ensures that names only contain letters using an @staticmethod, is\_valid\_name, and applies validation through an @property method for first\_name and last\_name. These properties enforce capitalization and raise a ValueError if invalid input is provided. The class also includes a \_\_str\_\_ method for formatted output and a from\_json static method to create a Person instance from a JSON string. This structure maintains data integrity while allowing controlled attribute access. (figure 1.6)

class Person:  
 def \_\_init\_\_(self, first\_name: str = "", last\_name: str = ""):

self.first\_name = first\_name # Calls setter  
self.last\_name = last\_name # Calls setter

**Figure 1.6: The Person class**

**Key functions include:**

1. \_\_init\_\_(self, first\_name: str = "", last\_name: str = ""): The constructor method initializes a Person object with optional first\_name and last\_name attributes, processing them through their respective setters.
2. is\_valid\_name(name: str) -> bool: A static method that checks if a name is valid (only containing letters or being empty). The setters use this method for first\_name and last\_name.
3. first\_name (property and setter): A property that returns the first\_name with capitalized first letter. The setter validates the input using is\_valid\_name and assigns it to the private attribute \_\_first\_name.
4. last\_name (property and setter): A property that returns the last\_name with the first letter capitalized. The setter validates the input using is\_valid\_name and assigns it to the private attribute \_\_last\_name.
5. \_\_str\_\_(self): The string representation method that formats the first\_name and last\_name attributes into a space-separated string.
6. from\_json(json\_data: str): A static method that converts a JSON string into a Person object by extracting the first\_name and last\_name from the JSON data.

# Student class inherits/extends the Person class

The Student class extends the Person class and initializes a student object with first\_name, last\_name, and course\_name, ensuring the parent constructor is correctly called while setting course\_name separately. The course\_name property retrieves the value of the private attribute \_\_course\_name in a title-cased format, and its setter assigns the provided value. The \_\_str\_\_ method returns a formatted string representation of the student, combining their full name and course name. (figure 1.7)

class Student(Person):  
 def \_\_init\_\_(self, first\_name: str = "", last\_name: str = "", course\_name: str = ""):

super().\_\_init\_\_(first\_name, last\_name) # Corrected call  
self.course\_name = course\_name # Assign separately

**Figure 1.7 The Student class**

**The key functions include:**

1. \_\_init\_\_: The constructor method that initializes the first\_name, last\_name, and course\_name attributes. It also calls the parent class's constructor using super() to initialize first\_name and last\_name.
2. course\_name **(getter)**: A property method retrieves the value of the private attribute \_\_course\_name and returns it with the title case applied.
3. course\_name **(setter)**: A setter method that assigns a value to the private attribute \_\_course\_name.
4. \_\_str\_\_: The string representation method that returns a string combining the first\_name, last\_name, and course\_name attributes in a specific format: "first\_name last\_name - course\_name".

# The functions inside the Class FileProcessor

The `FileProcessor` class provides static methods for handling JSON file operations related to student data. The `read\_data\_from\_file` method reads student records from a JSON file and converts them into a list of `Student` objects, correctly handling files that are not found or decoding errors. The `write\_data\_to\_file` method saves a list of `Student` objects to a JSON file, ensuring proper formatting and handling potential file access issues. Error handling is incorporated to notify users of issues during file operations. The class is structured for modularity, making it easy to integrate into broader applications that manage student records.

**Key Functions include:**

1. read\_data\_from\_file(file\_name: str)
   1. Reads student data from a JSON file.
   2. Parses the JSON content and converts it into a list of Student objects.
   3. Handles file-related errors, such as missing files or incorrect formatting.
2. write\_data\_to\_file(file\_name: str, student\_data: list)
   1. Writes student data to a JSON file.
   2. Converts Student objects into dictionaries before saving.
   3. Includes error handling for potential file access issues.
   4. Calls IO.output\_student\_and\_course\_names() to display saved data.
   5. Handles file closure properly to prevent data corruption.

# The functions inside the Class IO

The `IO` class provides a set of static methods for handling user input and output in a presentation layer, focusing on error messages, menu interactions, and student data management. It includes functions to display error messages, print a menu, validate and retrieve a menu choice, display student and course information, and collect student details with basic validation. The `input\_student\_data` method ensures that names do not contain numbers and creates a `Student` object before adding it to the provided list. The class helps standardize user interaction while managing potential input errors effectively. This class provides a simple interface for managing student records through console-based interactions, with built-in error handling and structured data storage.

The key functions are:

1. output\_error\_messages(message: str, error: Exception = None) – Displays a custom error message to the user, optionally including technical details of an exception.
2. output\_menu(menu: str) – Prints a formatted menu of choices for the user.
3. input\_menu\_choice() – Gets and validates a menu choice from the user, ensuring it is one of the expected options (1, 2, 3, or 4).
4. output\_student\_and\_course\_names(student\_data: list) – Displays a formatted list of students and their enrolled courses.
5. input\_student\_data(student\_data: list) – Collects student information (first name, last name, and course) from the user, performs basic validation, creates a Student object, and adds it to the provided list.

These functions collectively manage user interactions and data handling within the presentation layer.

# The presentation and interaction with the student menu

This code reads student data from a file using `FileProcessor.read\_data\_from\_file(FILE\_NAME)` and enters a continuous loop to present a menu for user interaction. The menu allows the user to input new student data (`option 1`), display current student and course information (`option 2`), save the updated data back to the file (`option 3`), or exit the program (`option 4`). The loop ensures the program continues running until the user explicitly chooses to exit. Additionally, the code includes a safeguard to prompt users to select only valid menu options.

# Running from the Command Prompt

Running from the command prompt, I first showed the existing data in the file. (figure 1.7)

A screenshot of a computer program

AI-generated content may be incorrect.

**Figure 1.7: Existing student data in the file**

Next, I added another student enrollment and showed it was in memory and that the new registration entered in lower case was capitalized. (figure 1.8)

A screenshot of a computer screen

AI-generated content may be incorrect.

**Figure 1.8: Enrolled new student data to file**

After saving the file to the JSON, I confirmed that the added enrollment was in the file. (figure 1.9)

A screenshot of a computer program

AI-generated content may be incorrect.

**Figure 1.9: All enrollments shown in JSON file**

Summary

# This week while working on Assignment 07, I faced several challenges that taught me important lessons. The biggest issue was understanding and tracking the classes and functions, particularly because I wasn't correctly calling a class, which prevented a specific method or function from executing. I also encountered a data persistence issue where reading and writing to the JSON file caused all the data to be deleted unexpectedly. From these experiences, I've learned that even if the code seems to be running correctly, I need to ensure that each class and its corresponding functions are being properly executed to avoid issues.