Mariusz Sokol

Mar, 12, 2025

Writing a paper

Assingnment 07

Github link: https://github.com/Mariusz-uw/IntroToProg-Python-Mod07

Introduction

This Python script implements a structured Course Registration Program utilizing object-oriented programming principles.

Program Script

This script presents featured Python program.

```
# ------------------------------------------------------------------------------ #
# Title: Assignment07
# Desc: This assignment demonstrates using data classes
# with structured error handling
# Change Log: (Who, When, What)
#   RRoot,1/1/2030,Created Script
#   <Mariusz Sokol>,<3, 12, 2025>,<Activity>
# ------------------------------------------------------------------------------ #
```

Figure 1: Script for the presented Python program.

Module import

Imported module necessary for handling json data in this program.

```
import json
```

Figure 2: Importing json module.

Person class and Student class

Person class represents a person with basic attributes. The properties “student_first_name” and “student_last_name” default to empty strings.

Student class is one that inherits properties from Person class and extends it by adding course_name attribute that defaults to empty string as well.

```python
class Person:
    """Represents a person with basic attributes."""

    def __init__(self, student_first_name: str = "", student_last_name: str = ""):
        self.student_first_name = student_first_name
        self.student_last_name = student_last_name

class Student(Person):
    """
    Extends Person to include course enrollment information.
    """
    def __init__(self, student_first_name: str = "", student_last_name: str = "",
                 course_name: str = ""):
        self.student_first_name = student_first_name
        self.student_last_name = student_last_name
        self.course_name = course_name
```

Figure 3: Person class and Student class.

FileProcessor class and its functions

FileProcessor class handles file-related input and output operations. Method “read_data_from_file” reads student enrollment data from a specified JSON file “Enrollments.json” and populates a given list with “Student” objects. It also implements structured error handling to catch and report any file-writing issues through “ IO.output_error_messages ”.

Method “ write_data_to_file “ writes current student data back to the specified JSON file. It also utilizes structured error handling to catch and report any file-writing issues through “ IO.output_error_messages ”.

```python
class FileProcessor:
    """
    Handles file input and output operations.
    """
    @staticmethod
    def read_data_from_file(file_name: str, student_data: list):
        """Reads student data from a JSON file into a list of Student objects."""
        try:
            with open(file_name, "r") as file:
                student_data.clear()
                data = json.load(file)
                for item in data:
                    student_data.append(Student(**item))
        except Exception as e:
            IO.output_error_messages("Error reading file.", e)

    @staticmethod
    def write_data_to_file(file_name: str, student_data: list):
        """Writes student data to a JSON file."""
        try:
            with open(file_name, "w") as file:
                json.dump([student.__dict__ for student in student_data], file)
            IO.output_student_courses(student_data)
        except Exception as e:
            IO.output_error_messages("Error writing to file.", e)
```

Figure 4: FileProcessor class, methods to process data and error handling code.

IO class and its functions

IO class manages all interactions with the user such as input and output operations. Method “ output_error_messages “ displays custom error messages along with technical exception details. Method “ output_menu “ prints the main menu to the user. Method “ input_menu_choice “ captures user input for menu selections. Method “ output_student_courses ” lists enrolled students and their courses clearly formatted. Method “ input_student_data “ prompts the user to enter first name, last name, and course name. It also includes structured error handling for validation of entered data (first and last name must contain letters only). Catches errors and provides detailed messages to the user.

```python
class IO:
    """
    Handles user interface input/output operations.
    """
    @staticmethod
    def output_error_messages(message: str, error: Exception = None):
        """Displays an error message with optional technical details."""
        print(f"Error: {message}")
        if error:
            print(f"Technical details: {error}")

    @staticmethod
    def output_menu(menu: str):
        """Displays the menu."""
        print(menu)

    @staticmethod
    def input_menu_choice():
        """Gets a menu choice from the user."""
        return input("Enter your choice: ")

    @staticmethod
    def output_student_courses(student_data: list):
        """Displays the current student and course enrollments."""
        print("-" * 50)
        for student in student_data:
            print(f'{student.student_first_name} {student.student_last_name}'
                  f' is enrolled in {student.course_name}')
        print("-" * 50)

    @staticmethod
    def input_student_data(student_data: list):
        """Gets student data from the user and adds it to the list."""
        try:
            first_name = input("Enter student's first name: ")
            if not first_name.isalpha():
                raise ValueError("First name must contain only letters.")
            last_name = input("Enter student's last name: ")
            if not last_name.isalpha():
                raise ValueError("Last name must contain only letters.")
            course_name = input("Enter course name: ")
            student_data.append(Student(first_name, last_name, course_name))
            print(f"Successfully registered {first_name} {last_name}"
                  f" for {course_name}.")
        except ValueError as e:
            IO.output_error_messages(str(e), e)
        except Exception as e:
            IO.output_error_messages("Unexpected error while entering"
                                     " student data.", e)
```

Figure 5: IO class and methods that handle user interactions. Exception code handles error occurrences.

Choice Menu

Showing the actual text that is printed at program start for the user and every time the control flow comes back after looping through the program corresponding to users choice.

```
MENU = '''\n---- Course Registration Program ----
    Select from the following menu:\n
1. Register a Student for a course
2. Show current Data
3. Save data to a file
4. Exit the program\n'''
FILE_NAME = "Enrollments.json"
students = []
```

Figure 6: Choice Menu displayed to the user.

Control flow

Loads data from " Enrollments.json " into a list of " students " objects. It then enters an infinite loop presenting a menu with four options:

1. Includes structured validation on first and last names.
2. Shows a list of students and which courses they enrolled in.
3. Writes data and displays stored content.
4. Exits the program gracefully.

```
FileProcessor.read_data_from_file(FILE_NAME, students)
```

```
while True:
    IO.output_menu(MENU)
    choice = IO.input_menu_choice()

    if choice == '1':
        IO.input_student_data(students)
    elif choice == '2':
        IO.output_student_courses(students)
    elif choice == '3':
        FileProcessor.write_data_to_file(FILE_NAME, students)
    elif choice == '4':
        print("Exiting program.")
        break
    else:
        print("Invalid choice. Please choose a valid option.")
```

Figure 7: Control flow code handles what part of the script is run.

Command Prompt Test 1

The entire program tested in Command Prompt.

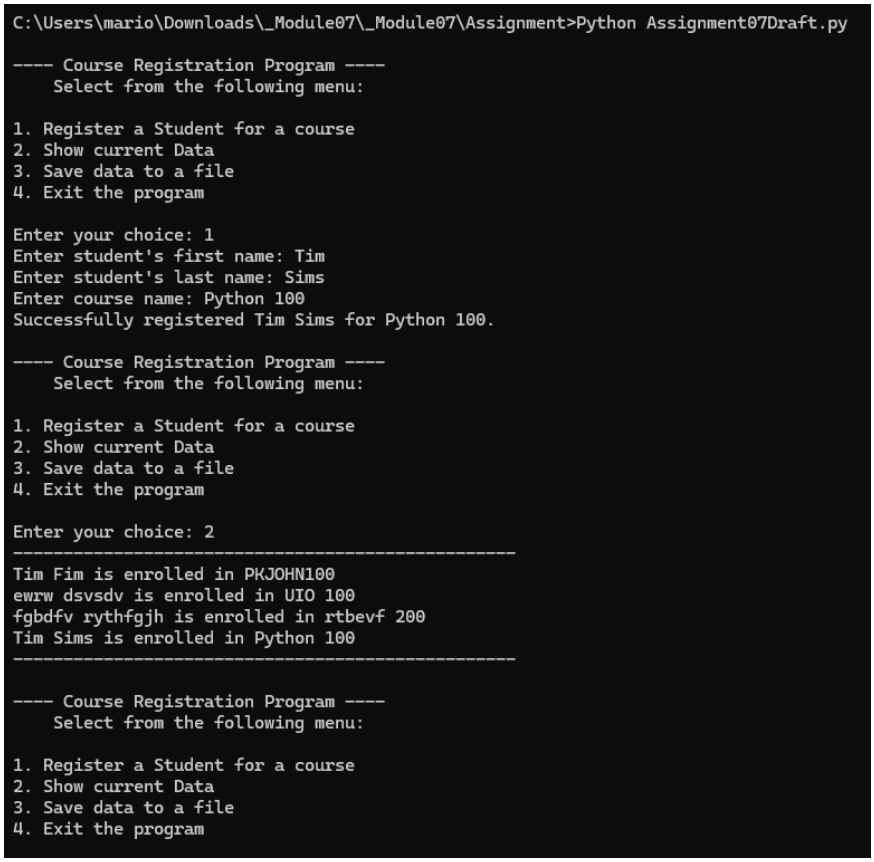

Figure 8: Command Prompt test.

Command Prompt Test 2

Last section of the program run in Command Prompt for testing.

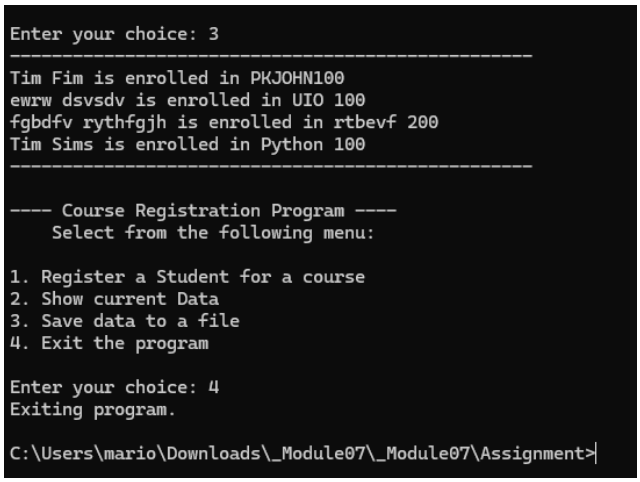

Figure 9: Command Prompt test

Summary

This program presentation showed how classes and functions/methods can keep code organized in sections so that perhaps others can later work on the code as well.