Name: Parth Dave

Date: August 20, 2025

Course: IT FDN 110 A Su 25: Foundation of Programming: Python

Assignment: Assignment06

Introduction

This week, I built upon both advanced concepts and fundamental building blocks of programming. I focused on how functions and classes contribute to writing modular, maintainable, and reusable code.

Functions are the building bloc of modular programming, as they allow complex logic to be broken into smaller, manageable units. They improve readability, reusability, and promote a modular approach. parameters and default parameters, allows functions to be more flexible and predictable, reducing redundancy. Moreover, return arguments ensure that a function not only performs an action but also communicates results back further enhancing reusability.

Understanding global vs local variables was also key. While local variables are limited to the scope of a function and are easier to maintain, global variables can introduce unexpected side effects, making code harder to manage. Avoiding unnecessary global variables ensures that code is cleaner, more predictable, and less prone to bugs.

Another important concept is encapsulation, a fundamental principle of programming. Encapsulation helps bind data and behavior into a single unit, through classes. It restricts variable access to controlled scopes, allowing developers to modify or interact with data in a safe and predictable way. Encapsulation reduces reliance on global data, leading to cleaner and more maintainable programs.

Classes play an important role in organizing code by grouping functions, variables, and constants under a single block. This structure is particularly useful in larger projects, as it enforces modularity, improves readability, and simplifies maintenance. A special case is the static method, where behavior does not depend on the instance of a class. In Python, the @staticmethod decorator allows direct access to such methods without creating a class object, improving clarity and efficiency when instance data is not needed.

To further improve code reliability, Python provides docstrings, which are written when defining a class or function. Docstrings serve as inline documentation, making code self-explanatory and easier to maintain.

Finally, I explored Separation of Concerns (SoC), an essential software engineering principle. SoC emphasizes dividing code into distinct, self-contained components, where each part is responsible for a specific functionality. This practice encourages modularity, reduces duplication, simplifies debugging, and enhances scalability. By following SoC, developers can build robust, testable, and maintainable systems.

Description

In this week's assignment, we built upon the work from Module 5 by reorganizing the code to use classes with static methods, promoting a more modular structure.

Following the Separation of Concerns (SoC) principle, two main classes were implemented:

- FileProcessor Class
  - Handles all file-related operations such as reading and writing.
  - Includes appropriate error-handling methods to ensure reliability and robustness.
- IO Class
  - Contains multiple functions with local variables for safer execution.
  - Provides functionality for printing menus, capturing user input, registering student data, and displaying stored records.

To improve maintainability and readability, docstrings were added to each class and function. These inline documentation blocks enhance the reliability of the code by making its purpose and behavior clear for future use and collaboration.

Code Snippet:

Code repository on GitHub: https://github.com/ppdave19/IntroToProg-Python-Mod06.git

```
# ------------------------------------------------------------------------------------------ #
# Title: Assignment06
# Desc: A Course Registration Program using JSON, functions, and structured error handling.
# Change Log: Parth Dave, 08/20/2025, Modified Lab Code Modul06-Lab03 and Completed Assignment
# ------------------------------------------------------------------------------------------ #

import json

# Constants

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
-----------------------------------------
'''

# Define the program's data

FILE_NAME: str = "Enrollments.json"
menu_choice: str = ''
students: list = []

# Defining - Class called - FileProcessor

class FileProcessor:
   """ Created Class
   Added function to read and write to JSON files
   A collection of processing layer functions that work with Json files
   ChangeLog: Parth Dave, 08/20/25, File IO handling class created
   """

   # When the program starts, read the file data into table
   # Extract the data from the file
   # Read from the Json file

   @staticmethod
   def read_data_from_file(file_name: str, student_data: list):
       """Reads student data from a JSON file into a list of dictionaries

       Args:
           file_name (str): The name of the file to read from
           student_data (list): The list to store the data into
       """

       file = None
       try:
           file = open(file_name, "r")
           data = json.load(file)
           student_data.extend(data)
       except FileNotFoundError as e:
           IO.output_error_messages("Text file must exist before running this script!", e)
       except json.JSONDecodeError as e:
           IO.output_error_messages("File is empty or has invalid JSON format.", e)
       except Exception as e:
           IO.output_error_messages("There was a non-specific error!", e)
       finally:
           if file.closed == False:
               file.close()

   @staticmethod
   def write_data_to_file(file_name: str, student_data: list):
       """Writes the student data to a JSON file

       Args:
           file_name (str): The name of the file to write to
           student_data (list): The data to write to the file
       """
       file = None
       try:
           file = open(file_name, "w")
           json.dump(student_data, file)
           print("The following data was saved to the file:")
           IO.output_student_courses(student_data)
       except TypeError as e:
           IO.output_error_messages("Please check that the data is a valid JSON format", e)
       except Exception as e:
           IO.output_error_messages("There was a non-specific error!", e)
       finally:
           if file.closed == False:
               file.close()

class IO:
   """
   Created IO Class: A collection of presentation layer functions that manage user input and
output
```

```
    Added Function for Error Handling
    Added Function to print menu
    Added Function to input user choice
    Added Function to input Student Registration Data
    Added function to output current registration data

    ChangeLog:
    Parth Dave,08/20/2025,Created Class

    """
    pass

    @staticmethod
    def output_error_messages(message: str, error: Exception = None):
        """This function displays custom error messages to the user

        Args:
            message (str): Error message to display
            error (Exception, optional): The exception object
        """
        print(message, end="\n\n")

        if error is not None:
            print("-- Technical Error Message -- ")
            print(error, error.__doc__, type(error), sep='\n')

    @staticmethod
    def output_menu(menu: str):
        """This function displays the a menu of choices to the user

        changelog: Parth Dave, 08/20/2025
        Args:
            menu (str): The menu string to display

        """
        print()
        print(menu)
        print() # Adding extra space to make it look nicer.

    @staticmethod
    def input_menu_choice() -> str:
        """This function gets a menu choice from the user

        Returns:
            str: The user's menu choice
        """
        try:
            choice = input("Enter your menu choice number: ")
            if choice not in ("1","2","3","4"):  # Note these are strings
                raise Exception("Please, choose only 1, 2, 3, or 4")
        except Exception as e:
             IO.output_error_messages(e.__str__())  # Not passing the exception object to avoid
the technical message

        return choice

    @staticmethod
    def input_student_data(student_data: list):
        """This function gets the first name, last name, and ccourse from the user
        Args:
            student_data (list): The list to append new student record to
        """
        try:
            first_name = input("Enter the student's first name: ")
            if not first_name.isalpha():
                raise ValueError("First name must only contain letters.")

            last_name = input("Enter the student's last name: ")
            if not last_name.isalpha():
                raise ValueError("Last name must only contain letters.")

            course_name = input("Enter the course name: ")
            if not course_name:
                raise ValueError("Course name cannot be empty.")

            student = {"FirstName": first_name,
                       "LastName": last_name,
                       "CourseName": course_name}
            student_data.append(student)

            print(f"\nSuccessfully registered {first_name} {last_name} for {course_name}.\n")

        except ValueError as e:
            IO.output_error_messages("That value is not the correct type of data!", e)
        except Exception as e:
            IO.output_error_messages("There was a non-specific error!", e)
        return student_data

    @staticmethod
    def output_student_courses(student_data: list):
        """This function displays current registration list

        Args:
            student_data (list): The list of student dictionaries
        """
        if not student_data:
            print("No student registrations found.\n")
            return

        print("\nCurrent Student Enrollments:")
        print("-" * 50)
        for student in student_data:
            print(f"{student['FirstName']}, {student['LastName']}, {student['CourseName']}")
        print("-" * 50 + "\n")

# Beginning of the main body of this script

# Load existing data from the file at the start
FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)

# Repeat the follow tasks

while True:
    IO.output_menu(menu=MENU)
    menu_choice = IO.input_menu_choice()

    if menu_choice == "1": # Enter Student Registration
        IO.input_student_data(student_data=students)
        continue
    elif menu_choice == "2": # Display current data
        IO.output_student_courses(student_data=students)
        continue

    elif menu_choice == "3": # Save data in a file
        FileProcessor.write_data_to_file(file_name=FILE_NAME, student_data=students)
        continue

    elif menu_choice == "4": # End the program
        print("Exiting the program. Goodbye!")
        break # out of the while loop
```

Sample Output

```
C:\Users\parth\Downloads\_Module06-1\_Module06\Assignment\A06.py


---- 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
-----------------------------------------


Enter your menu choice number: 1
Enter the student's first name: Parth
Enter the student's last name: Dave
Enter the course name: Python-601

Successfully registered Parth Dave for Python-601.



---- 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
-----------------------------------------


Enter your menu choice number: 2

Current Student Enrollments:
--------------------------------------------------
Bob, Smith, Python 100
Sue, Jones, Python 100
Parth, Dave, Python-601
--------------------------------------------------



---- 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
----------------------------------------


Enter your menu choice number: 4
Exiting the program. Goodbye!

Process finished with exit code 0
```

Summary

This week, I strengthened my understanding of functions, parameters, encapsulation, classes, and separation of concerns, which are all crucial for writing modular, reusable, and scalable code. Study material and corresponding video and lab exercises were especially helpful in reinforcing these fundamental concepts, allowing me to grasp and apply them more effectively.