

Justin Spencer

May 28, 2025

IT FDN 110 A

Assignment 06

<https://github.com/JSpencer45/IntroToProg-Python-Mod06>

Assignment 06: Functions, Parameters, & Classes

Introduction

In this assignment, I utilized the concepts presented in Module06 to create a Python program that uses functions, classes, and separations of concerns in order to capture multiple user inputs for student class registration.

Creating the Code

Defining the Constants & Variables

The first step I took was to define the constants and variable that would include the menu of choices the user would pick from, the JSON file that would be read and written to, the list containing the student data and the string of menu choices that the user could make (Figure 1). Given that many of the other variables that were defined early on in the code as they were in previous modules would be made to be mutable locally, I removed many of them from the Assignment06 starter script.

```
import json
from idlelib.browse import file_open

# Define the Data 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 Data Constants
# FILE_NAME: str = "Enrollments.csv"
FILE_NAME: str = "Enrollments.json"

# Define the Data Variables and constants
students: list = [] # a table of student data
menu_choice: str = '' # Hold the choice made by the user.
```

Figure 1 - Defined constants & variables.

Processing, Presentation & Error Handling:

After defining the variables and constants, I created the Processing section of the script. This has the FileProcessor class which houses and separates the layer functions from the rest of the script functions. The read_data_from_file (Figure 2) and write_data_to_file (Figure 3) functions are in the FileProcessor class because they interface with the JSON file and work beneath the Presentation layer. To note, all functions are set to the @staticmethod in order for them to be called on in the main body of the script, which allows for the code to become more readable and easier to edit.

```
#-- Processing --#
class FileProcessor: 2 usages
    """
    A collection of processing layer functions that work with json files
    ChangeLog: (Who, When, What)
    JSpencer, 5/28/2025, Created Class
    """

    @staticmethod 1 usage
    def read_data_from_file(file_name: str, student_data: list):
        """
        This function reads json file in dict rows and checks for errors
        ChangeLog: (Who, When, What)
        JSpencer, 5/28/2025, Created Function

        :return: None
        """
        try:
            file = open(file_name, "r")
            student_data = json.load(file)
            file.close()
        except FileNotFoundError as e:
            IO.output_error_messages( message: "Text file must exist before running this script!", e)
        except Exception as e:
            IO.output_error_messages( message: "There was a non-specific error!", e)
        finally:
            if file.closed == False:
                file.close()
        return student_data
```

Figure 2 - read_data_from_file function set to the FileProcessor class.

```
@staticmethod 1 usage
def write_data_to_file(file_name: str, student_data: list):
    """
    This function opens the json file in write mode and checks for errors
    ChangeLog: (Who, When, What)
    JSpencer, 5/28/2025, Created Function

    :return: None
    """
    try:
        file = open(file_name, "w")
        json.dump(student_data, file)
        file.close()
    except TypeError as e:
        IO.output_error_messages( message: "Please check that the data is a valid JSON format!", e)
    except Exception as e:
        IO.output_error_messages( message: "There was a non-specific error!", e)
    finally:
        if file.closed == False:
            file.close()
    print("-" * 50)
    print("The following data has been saved: ")
    for student in students:
        print(f'{student["FirstName"]},{student["LastName"]},{student["CourseName"]}\n')
    print("-" * 50)
    return student_data
```

Figure 3 - write_data_to_file function set to the FileProcessor class.

Once the Processing class was complete, I moved to build the Input and Output (IO) class. This class would include displaying error messages (Figure 4), displaying the current data (Figure 5), capturing user inputs and displaying them back to the user, writing the inputs to the JSON file (Figure 6), displaying the menu (Figure 7) and closing the program.

```
-- Presentation (Input/Output) --#
class IO: 11 usages
    """
    A collection of presentation layer functions that manage user input and output
    ChangeLog: (Who, When, What)
    JSpencer, 5/28/2025, Created Class
    """
    pass

    @staticmethod 7 usages
    def output_error_messages(message: str, error: Exception = None):
        """
        This function outputs error messages
        ChangeLog: (Who, When, What)
        JSpencer, 5/28/2025, Created Function

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

Figure 4 - output_error_messages set to IO class.

```
@staticmethod 1 usage
def output_student_courses(student_data: list):
    """
    This function outputs student course registration
    ChangeLog: (Who, When, What)
    JSpencer, 5/28/2025, Created Function

    :return: None
    """

    print()
    print("-" * 50)
    for student in students:
        print(f'{student["FirstName"]},{student["LastName"]},{student["CourseName"]}\n')
    print("-" * 50)
    print()
```

Figure 5 - output_student_courses_ set to IO class.

```

@staticmethod 1 usage
def input_student_data(student_data: list):
    """
    This function inputs the student's first name, last name and registered class
    ChangeLog: (Who, When, What)
    JSpencer, 5/28/2025, Created Function

    :return: show's the user's entry back to them
    """
    try:
        student_first_name = input("Enter student's first name: ")
        if not student_first_name.isalpha():
            raise ValueError("Student first name should not contain numbers!")
        student_last_name = input("Enter student's last name: ")
        if not student_last_name.isalpha():
            raise ValueError("Student last name should not contain numbers!")
        course_name = input("Enter student's course name: ")
        student_data = {"FirstName": student_first_name,
                        "LastName": student_last_name,
                        "CourseName": course_name}
        students.append(student_data)
        print("-" * 50)
        print("The following data has been entered: ")
        for student in students:
            print(f'{student["FirstName"]}, {student["LastName"]}, {student["CourseName"]}\n')
        print("-" * 50)

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

```

Figure 6 - input_student_data set to IO class.

```

@staticmethod 1 usage
def output_menu(menu: str):
    """
    This function outputs menu choice
    ChangeLog: (Who, When, What)
    JSpencer, 5/28/2025, Created Function

    :return: None
    """
    print()
    print(menu)
    print()

@staticmethod 1 usage
def input_menu_choice():
    """
    This function captures user menu choice and displays it back
    ChangeLog: (Who, When, What)
    JSpencer, 5/28/2025, Created Function

    :return: string with the user's choice
    """
    choice = "0"
    try:
        choice = input("Enter your choice: ")
        if choice not in ("1", "2", "3", "4"):
            raise Exception("Menu choice must be 1, 2, 3, or 4!")
    except Exception as e:
        IO.output_error_messages(e.__str__())

    return choice

```

Figure 7 - Menu inputs and outputs set to IO class.

The final part of the script was the main body. This was the code that would call on all of the previously defined functions and classes and apply them the program. Since the `@staticmethod` tool was used for the functions, the follow tasks could be coded more easily and did not require redefining any variables (Figure 8).

```
# End of function definitions #

# Beginning of the main body of the script

students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)

# repeat the follow tasks
while (True):

    # Present the menu of choices
    IO.output_menu(menu=MENU)

    menu_choice = IO.input_menu_choice()

    # Input user data
    if menu_choice == "1": # This will not work if it is an integer!
        IO.input_student_data(student_data=students)
        continue

    # Present the current data
    elif menu_choice == "2":
        IO.output_student_courses(student_data=students)
        continue

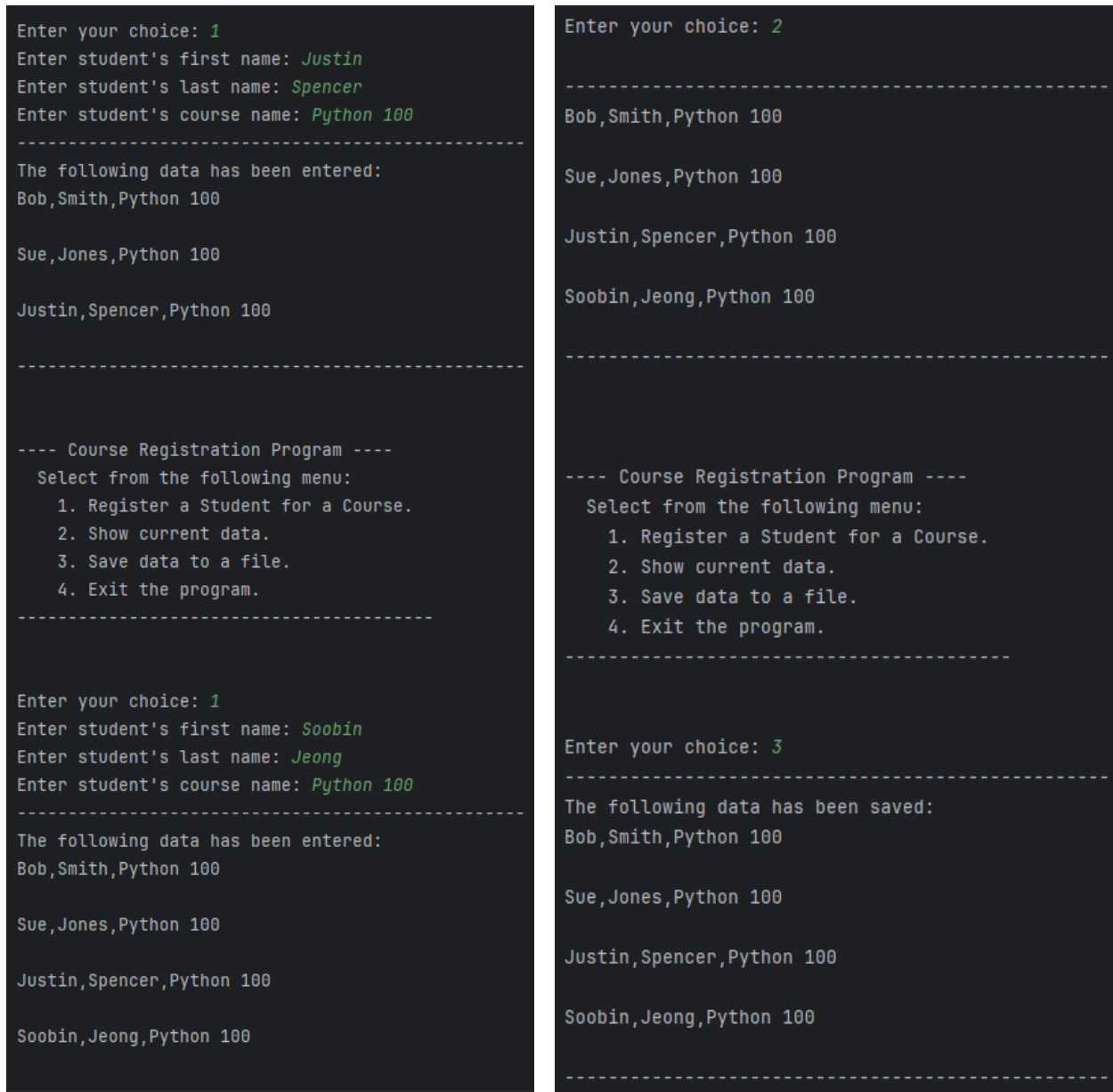
    # Save the data to a file
    elif menu_choice == "3":
        FileProcessor.write_data_to_file(file_name=FILE_NAME, student_data=students)
        continue

    # Stop the loop
    elif menu_choice == "4":
        break # out of the loop
        print("Program Ended")
    else:
        print("Please only choose options 1, 2, 3 or 4!")
```

Figure 8 - Follow tasks using while True statement & call functions.

Testing the Code

When testing in PyCharm, I tested the code using multiple entries which resulted in the program displaying both the entered names and courses (Figure 9) as well as saving them to the “Enrollments.json” file in the shared folder (Figure 10). I was also able to successfully test the script in the Command Shell which also stored and saved the user input to the program and the JSON file (Figure 11).



```
Enter your choice: 1
Enter student's first name: Justin
Enter student's last name: Spencer
Enter student's course name: Python 100
-----
The following data has been entered:
Bob,Smith,Python 100

Sue,Jones,Python 100

Justin,Spencer,Python 100

-----

---- 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 choice: 1
Enter student's first name: Soobin
Enter student's last name: Jeong
Enter student's course name: Python 100
-----
The following data has been entered:
Bob,Smith,Python 100

Sue,Jones,Python 100

Justin,Spencer,Python 100

Soobin,Jeong,Python 100

-----

Enter your choice: 2
-----
Bob,Smith,Python 100

Sue,Jones,Python 100

Justin,Spencer,Python 100

Soobin,Jeong,Python 100

-----

---- 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 choice: 3
-----
The following data has been saved:
Bob,Smith,Python 100

Sue,Jones,Python 100

Justin,Spencer,Python 100

Soobin,Jeong,Python 100

-----
```

Figure 9 - PyCharm Test with multiple entries.

```
[
  {
    "FirstName": "Bob",
    "LastName": "Smith",
    "CourseName": "Python 100"
  },
  {
    "FirstName": "Sue",
    "LastName": "Jones",
    "CourseName": "Python 100"
  }
]
```

```
[
  {
    "FirstName": "Bob",
    "LastName": "Smith",
    "CourseName": "Python 100"
  },
  {
    "FirstName": "Sue",
    "LastName": "Jones",
    "CourseName": "Python 100"
  },
  {
    "FirstName": "Justin",
    "LastName": "Spencer",
    "CourseName": "Python 100"
  },
  {
    "FirstName": "Soobin",
    "LastName": "Jeong",
    "CourseName": "Python 100"
  }
]
```

Figure 10 - PyCharm test results in JSON file.


```

Microsoft Windows [Version 10.0.26100.4061]
(c) Microsoft Corporation. All rights reserved.

C:\Users\jbspe>cd Documents\Python\PythonCourse
C:\Users\jbspe\Documents\Python\PythonCourse>cd A06
C:\Users\jbspe\Documents\Python\PythonCourse\A06>python "Assignment06.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 choice: 1
Enter student's first name: Vic
Enter student's last name: Vu
Enter student's course name: Python 100
-----

The following data has been entered:
Bob,Smith,Python 100

Sue,Jones,Python 100

Justin,Spencer,Python 100

Soobin,Jeong,Python 100

Vic,Vu,Python 100
-----

```

```

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 choice: 2
-----

Bob,Smith,Python 100

Sue,Jones,Python 100

Justin,Spencer,Python 100

Soobin,Jeong,Python 100

Vic,Vu,Python 100
-----

---- 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 choice: 3
-----

The following data has been saved:
Bob,Smith,Python 100

Sue,Jones,Python 100

Justin,Spencer,Python 100

Soobin,Jeong,Python 100

Vic,Vu,Python 100
-----

```

```

1  [
2      {
3          "FirstName": "Bob",
4          "LastName": "Smith",
5          "CourseName": "Python 100"
6      },
7      {
8          "FirstName": "Sue",
9          "LastName": "Jones",
10         "CourseName": "Python 100"
11     },
12     {
13         "FirstName": "Justin",
14         "LastName": "Spencer",
15         "CourseName": "Python 100"
16     },
17     {
18         "FirstName": "Soobin",
19         "LastName": "Jeong",
20         "CourseName": "Python 100"
21     },
22     {
23         "FirstName": "Vic",
24         "LastName": "Vu",
25         "CourseName": "Python 100"
26     }
27 ]

```

Figure 11 - CMD Shell test results in JSON file.

Summary

With the successful testing of my Python script, I believe I have demonstrated my understanding of the material presented in Module 06 in the use of functions, classes, and separations of concerns.