Jacob Widner

8/5/2024

IT FDN 110 A Su 24: Foundations of Programming: Python

Assignment06

GitHub link: https://github.com/JakeWidner/IntroToProg-Python-Mod06

Integrating Classes and Functions into Scripts

Introduction

In this document I will explain the process I used to complete the sixth assignment for this course. The script used to complete this assignment builds on what was used for the previous assignments and adds new concepts such as using classes and functions.

Designing the Code

Like the previous assignments, this script asks the user for input. This assignment asks the user to choose an option from a menu. The options include opportunities for user input, printing output to the screen, saving data to a file, and exiting the program. This assignment also reads data from a JSON file that was previously saved to append the data. An important difference between this assignment and the last assignment is that the code for this program is organized into separate classes that include functions.

The Script Header

In the instructional materials for the first module, we were introduced to the concept of the script header. The script header is usually placed at the top or "head" of the script and consists of a series of commented lines that hold useful information about the script. These elements include a title, description, and a change log.[1] The header for this assignment (Figure 1) contains the same basic information used in the previous assignments.

```
# ------------------------------------------------------------------------ #
# Title: Assignment06
# Desc: This assignment demonstrates using functions
#       with structured error handling
# Change Log: (Who, When, What)
#   JWidner,8/5/2024,Created Script
#   <Your Name Here>,<Date>,<Activity>
# ------------------------------------------------------------------------ #
```

Figure 1: Assignment06 Script Header.

[1] Root, Randal, Creating Python Scripts: Script Headers, *Mod01-Notes,* p17.

Importing JSON Library

This assignment requires the manipulation of JSON files.[2] To use JSON files in your script you must first import the module that contains the appropriate methods for manipulating JSON files[3] (Figure 2).

```
# import json module
import json
```

Figure 2: JSON Module Import

Defining the Constants

We were introduced to the concepts of variables, constants and data types[4] as well as type hints[5] in the first module. For this assignment, a set of constants were defined including the menu that will be displayed to the user and the name of the JSON file we will be using to write the data to (Figure 3).

```
# 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.json"
```

Figure 3: Assignment05 Constants.

Defining the Variables

For this assignment, only two variables were used. One variable was defined as an empty string[6] and one was defined as an empty list[7]. Additional variables would be defined and used inside functions defined later in the code. These variables would be filled with values later in the code (Figure 4).

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

Figure 4: Assignment06 Variables.

---

[2] Root, Randal. JSON Files, *Mod05-Notes,* p13.
[3] Root, Randal. JSON Files: Working with JSON Files, *Mod05-Notes,* p15.
[4] Root, Randal, Program Data: Constants, Variables and Data Types, *Mod01-Notes,* p27.
[5] Root, Randal, Data Types: Data Type Naming Conventions, *Mod01-Notes,* p33.
[6] Root, Randal. Common Types of Errors, *Mod02-Notes,* p28.
[7] Root, Randal. Data Collections: Lists, *Mod04-Notes,* p17.

Class and Function Definitions

This assignment calls for the functional code to be organized into two classes[8] with descriptive document strings[9] and seven functions[10]. All functions use the "@staticmethod" modifier to allow the class functions to be used directly.[11] The following sections will describe the classes and their associated functions.

FileProcessor Class and Associated Functions

The FileProcessor class contains two functions that deal with processing JSON files. Each function in this class requires input in the form of parameters[12] that include the file name and the list variable. Figure 5 displays the class heading and document string:

```
class FileProcessor:
    """
    Functions for JSON file processing.

    ChangeLog: (Who, When, What)
    JWidner,8/5/2024,created class
    """
```

Figure 5: FileProcessor Class heading and document string

The first function defined as part of this class in the "read_data_from_file" function. In this function, we extract data from a JSON file by opening the file in read mode, extract the data using methods imported from the JSON module, and place it inside a list[13] (Figure 6). The function then returns the list to be used by other code. This function also includes error handling in the code using a Try-Except block[14]. This allows the script to continue to run when errors are encountered. For this code section, we can print an error message to the user if the JSON file is unable to be opened using a class and function defined later in the code.

```
@staticmethod
def read_data_from_file(file_name: str, student_data: list):
    """ This function read the data from a JSON file and return it as a list.

    ChangeLog: (Who, When, What)
    JWidner,8/5/2024,created function

    :return: list of student data read from the JSON file
    """
    try:
        file = open(file_name, "r")    # Extract the data from the file
        student_data = json.load(file)  # Read from the Json file
        file.close()
    except FileNotFoundError as e:
        IO.output_error_messages("Text file must exist before running this script!", e)
    except Exception as e:
        IO.output_error_messages("There was a non-specific error!", e)
    finally:
        if not file.closed:
            file.close()
    return student_data
```

---

[8] Root, Randal. Classes and Functions, *Mod06-Notes,* p27.
[9] Root, Randal. Classes and Functions: Document Strings (Docstrings), *Mod06-Notes,* p30.
[10] Root, Randal. Organizing Your Code: Functions, *Mod06-Notes,* p2.
[11] Root, Randal. Classes and Functions: Static Classes, *Mod06-Notes,* p29.
[12] Root, Randal. Parameters, *Mod06-Notes,* p14.
[13] Root, Randal. JSON Files: Working with JSON Files, *Mod05-Notes,* p15.
[14] Root, Randal. Structured Error Handling (Try-Except), *Mod05-Notes,* p19.

Figure 6: Function for reading JSON file.

The second function in this class is used to write data into a JSON file and print out a summary of the data written to the file. This section of code utilizes the concepts of opening, writing data to, and closing a JSON file (Figure 7).[15] This section also uses error handling in case the data does not correctly save to the JSON file.

```
@staticmethod
def write_data_to_file(file_name: str, student_data: list):
    """ This function writes the data to the JSON file.

    ChangeLog: (Who, When, What)
    JWidner,8/5/2024,created function

    :return: None
    """
    try:
        file = open(file_name, "w")
        json.dump(student_data, file, indent=1)
        file.close()
        print("The following data was saved to file!")
        for student in student_data:
            print(f"Student {student["FirstName"]} {student["LastName"]} is
enrolled in {student["CourseName"]}")
    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 not file.closed:
            file.close()
```

Figure 7: Function for writing to JSON file.

IO Class and Associated Functions

The "IO" Class contains five functions used for receiving input from the user and displaying information to the user. Figure 8 displays the class heading and document string:

```
class IO:
    """
    IO Functions section

    ChangeLog: (Who, When, What)
    JWidner,8/5/2024,created class
    """
```

Figure 8: IO Class header

The first function in this class is used for error handling. The error message and error type are input as parameters and the appropriate error message is displayed to the user (Figure 9).

[15] Root, Randal. JSON Files: Working with JSON Files, *Mod05-Notes,* p15.

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

    ChangeLog: (Who, When, What)
    JWidner,8/5/2024,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 9: Error Handling Function.

The next function in this class is used to display the menu to the user. This function receives a string parameter and prints it out (Figure 10).

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

    ChangeLog: (Who, When, What)
    JWidner,8/5/2024,created function

    :return: None
    """
    print()
    print(menu)
    print()  # Adding extra space to make it look nicer.
```

Figure 10: Output Menu function.

The next function asks for the user input and returns the choice they select as a string (Figure 11). If the user enters a choice not recognized by the function, a specific error handling message is raised.

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

    ChangeLog: (Who, When, What)
    JWidner,8/5/2024,created function

    :return: string with the users choice
    """
    choice = "0"
    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
```

Figure 11: Menu Choice Function.

The next function outputs messages displaying data showing the first, last, and course name of all students registered for courses. This function receives a list of dictionaries and outputs the data in a readable format (Figure 12). I also included error handling for a key mismatch in the data. To ensure that correct messaging during error handling, I used the global[16] variable for the filename in the function.

[16] Root, Randal. Global vs. Local Variables, *Mod06-Notes,* p6.

```
def output_student_courses(student_data: list):
    """ This function displays the list of student registered for each course

    ChangeLog: (Who, When, What)
    JWidner,8/5/2024,created function

    :return: None
    """
    global FILE_NAME  # Use filename constant
    # Process the data to create and display a custom message
    try:
        print("-" * 50)
        for student in student_data:
            print(f'Student {student["FirstName"]} '
                  f'{student["LastName"]} is enrolled in {student["CourseName"]}')
        print("-" * 50)
    except KeyError as e:  # Key mismatch error handling
        print(f"Key mismatch. Please ensure keys in {FILE_NAME} are correct.")
        print(f"Key {e} not found in {student_data}")
        print(f"Correct the keys in {FILE_NAME} and restart the program.")
    except Exception as e:
        IO.output_error_messages("There was an error displaying data!", e)
```

Figure 12: Student Course Output

The final function of this class asks the user to enter data including first name, last name and course name for a student that needs to be registered for a course. The current list with student data is input via a parameter and the updated list is returned (Figure 13). This section also contains error handling to prevent the user from entering numbers as part of a first or last name. This section utilizes custom error messages[17] based on which section of the code caused the error.

```
@staticmethod
def input_student_data(student_data: list):
    """ This function gets the first name, last name, and course name from the user

    ChangeLog: (Who, When, What)
    JWidner,8/5/2024,created function

    :return: None
    """

    try:
        # Input the data
        student_first_name = input("What is the student's first name? ")
        if not student_first_name.isalpha():
            raise ValueError("The first name should not contain numbers.")

        student_last_name = input("What is the student's last name? ")
        if not student_last_name.isalpha():
            raise ValueError("The last name should not contain numbers.")

        course_name = input("Please enter the name of the course: ")
        student = {"FirstName": student_first_name,
                   "LastName": student_last_name,
                   "CourseName": course_name}
        student_data.append(student)
        print(f"You have registered {student_first_name} {student_last_name} for {course_name}.")

    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
```

[17] Root, Randal. Structured Error Handling: Raising Custom Errors, *Mod05-Notes,* p22.

Figure 13: User Input function.

Main Body of the Script

This script separates the defined functions with the functional part of the program using a concept known as Separation of Concerns (SoC)[18]. This improves code readability and organization as the code length increases.

Importing Data from the JSON File

The first code in this section uses a pre-defined function from the "FileProcessor" class to import data from the JSON file defined by the "FILE_NAME" constant into a list variable (Figure 14).

```
# Beginning of the main body of this script
# When the program starts, read the file data into table
students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)
```

Figure 14: Import data from JSON file.

The While Loop

The next section of code establishes a loop using the while loop concept we learned in a previous module[19]. This tells the script to continue to run the following code while a specific conditional statement is true and presents the menu to the user and asks for a number as a choice(Figure 15). This section of code uses two different functions defined as part of the "IO" class.

```
# Repeat the follow tasks
while True:

    IO.output_menu(menu=MENU)   # Output the menu to the user
    menu_choice = IO.input_menu_choice()   # receive menu choice from user
```

Figure 15: Establish the loop.

Data Entry

If the user selects the first option, the user is given an opportunity to enter data using a function defined in the "IO" class. The data is then saved to a previously defined list (Figure 16).

```
if menu_choice == "1":  # Get new data and print out new student data
    students = IO.input_student_data(student_data=students)
    continue
```

Figure 16: Option 1, data entry.

Presenting Data to the User

If the user chooses option 2, the data that was entered by the user and appended into a list variable is then printed out to the user using a function from the "IO" class. This is a useful troubleshooting technique which helps verify the data once it has been saved to the JSON file (Figure 17).

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

Figure 17: Print out data to the user.

---

[18] Root, Randal. The Separation of Concerns Pattern, *Mod06-Notes,* p32.
[19] Root, Randal. Loops: The while Loop, *Mod03-Notes,* p16.

Data File Manipulation

If the user chooses option 3, the data is saved to a JSON file (Figure 18). This is done using a function from the "FileProcessor" class.

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

Figure 18: File Manipulation Code.

Closing the Loop and Ending the Program

The final section of code executes if the user chooses option 4. The loop stops and the program ends (Figure 19). If the user chooses an option not recognized by the conditional statements, error handling defined in the "input_menu_choice" function of the "IO" class.

```
    # Stop the loop
    elif menu_choice == "4":  # Exit the program
        break  # out of the loop

print("Program Ended")
```

Figure 19: Final conditional statement to close the loop and end the program.

Outputs

This assignment outputs a menu to the user, asks for input from the user, and outputs the data including any data read in from the JSON file. This assignment also outputs error messages based on exception handling. To test the script, we are asked to run it using both IDE[20] and the command console[21]. For this assignment, PyCharm[22] was used as the IDE. The results of these tests can be found below in Figures 20 and 21 respectively.

[20] Root, Randal, Integrated Development Environments: IDLE (Integrated Development and Learning Environment), *Mod01-Notes,* p14.
[21] Root, Randal, The Command Shell, *Mod01-Notes,* p10.
[22] Root, Randal, PyCharm, *Mod03-Notes,* p3.

Figure 20: PyCharm Output.

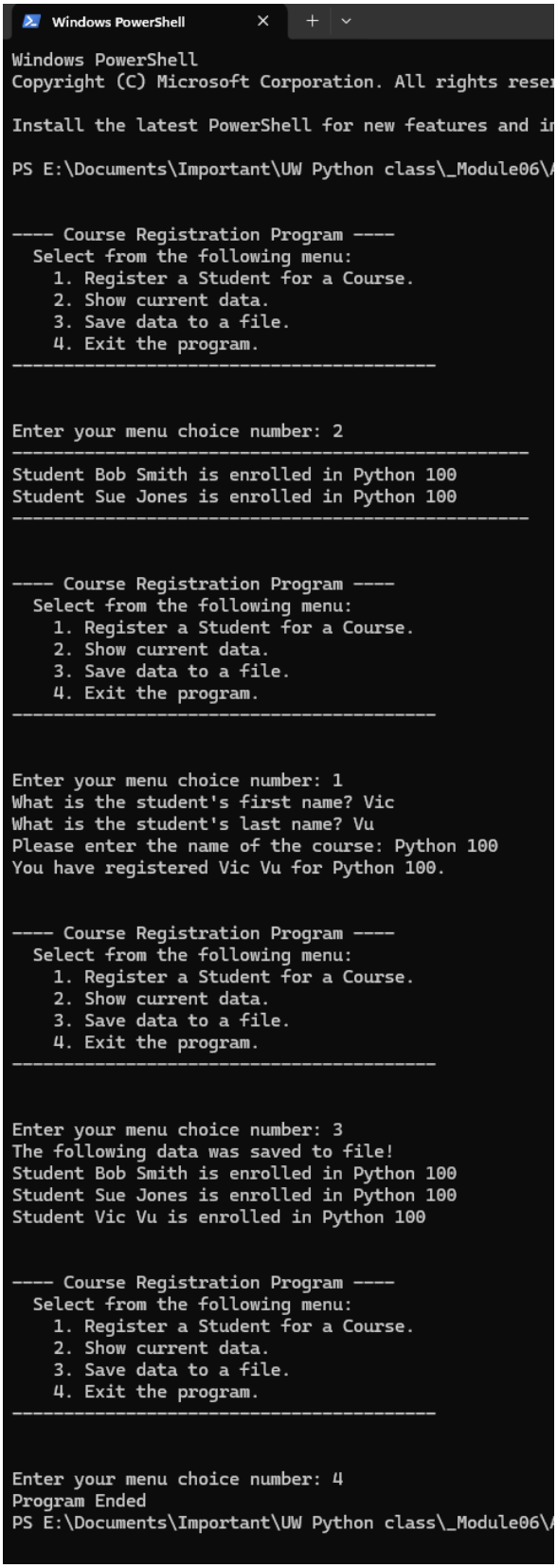

Figure 21: Command Console Output.

I then further verified the functionality of the script by opening the file generated by it using PyCharm (Figure 22).

Figure 22: Data File Output.

Both the data output to the user and the data found in the JSON file match, confirming that the script works as intended.

Summary

Using the information gained from reviewing the course materials, I was able to successfully complete this assignment and create a python script that reads data from a JSON file and then presents the user with a menu with multiple options and receives input from the user. The functionality of this assignment was the same as assignment 5. The difference is that the code was organized into classes and functions to improve code organization and flow.

This module's labs were very helpful in explaining the concepts I learned and helped me better understand the code I was building for the assignment. This module was enjoyable because a cornerstone of all programming languages is organization. My background in C and C++ granted me some familiarity with classes and functions. I look forward to building on these concepts in future modules.