Jessica Boardman

August 6, 2024

IT FND 110

Assignment 06

GitHub: https://github.com/jboarduw/IntroToProg-Python-Mod06

Classes and Functions

Introduction

This week I learned how to create functions inside of classes of a Python script and organize the code using the principle of “Separations of Concerns”. The script I created has two classes, one for processing data and one for presenting data to the user. Included in the script are error messages when exceptions occur or when specific user inputs are validated. This Python script started from an existing base script, from which I needed to interpret the existing script and make adjustments to meet the acceptance criteria. Upon completion of authoring the program, I was able to execute the script in both PyCharm and the MacOS terminal. The steps I took to complete Assignment 06 are expanded below.

Creating the Program

Before attempting to create the program, I read through the entire acceptance criteria in the assignment to gain a holistic view of what the program needed to do from beginning to end. I authored the Python program Assignment06.py in PyCharm. First I set up an outline of the file to include where the classes and functions were going to be placed. Based on the requirements, I kept the two constants and removed many of the variables (Figure 1.1). The

import JSON was also kept at the top of the script.

```
# ------------------------------------------------------------------------------
# Title: Assignment06_Starter
# Desc: This assignment demonstrates using functions
# with structured error handling
# Change Log: (Who, When, What)
#   RRoot,1/1/2030,Created Script
#   JBoardman, 8/6/2024, updated script to add classes and functions and SOC
# ------------------------------------------------------------------------------
import json

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

# (jnb 8.6.24) removing these variables and will call them locally
# student_first_name: str = ''  # Holds the first name of a student entered by the us
# student_last_name: str = ''  # Holds the last name of a student entered by the user
# course_name: str = ''  # Holds the name of a course entered by the user
```

Figure 1.1: Header Setup, constants, variables

Separations of Concerns

In the reading this week, we learned about the principle of “Separations of Concerns”. This principle defines how to organize code into 3 different functional layers:

- Data - manages data structures and works with files

- Processing - manages logic and does transformations of data. This layer also manages error handling.
- Presentation - displays information to users for input/output

There are several benefits to structuring Python in this way. The top three that stood out for me were maintainability, modularity so each layer can be tested independently, and reusability.

Classes

For Assignment06 we created two classes. The first class was for FileProcessor (Figure 1.2). Inside this class were the various functions that would be used when working with the JSON file to open, read and write to the file. Using classes in the JSON enabled us to group the functions and make the code easier to read and maintain.

```
class FileProcessor:
    """
    A collection of processing layer functions that work with Json files
    ChangeLog: (Who, When, What)
    JBoardman, 8.6.2024, created class and functions
    """
    # (jnb 8.6.24) When the program starts, read the file data into a list of lists (table)
    # (jnb 8.6.24) Extract the data from the file
    # (jnb 8.6.24) Call errors from IO.output_error_messages
    1 usage
    @staticmethod
    def read_data_from_file(file_name: str, student_data: list):...

    # (jnb 8.6.24) Write data to the json file
    # (jnb 8.6.24) Call errors from IO.output_error_messages
    1 usage
    @staticmethod
    def write_data_to_file(file_name: str, student_data: list):...
```

Figure 1.2: FileProcessor Class

The second class created was the IO class. Here is where all of the UI and user input/output was built (Figure 1.3). Error messaging was also built in the IO class.

```python
class IO:
    """
    A collection of presentation layer functions that manage user input and output
    ChangeLog: (Who, When, What)
    JBoardman, 8.6.2024, created class and functions
    """
    # (jnb 8.6.24) Variables inside of this class
    new_student: dict = {}  # one row of student data used in input_student_data func
    student_first_name: str = ''  # Holds the first name of a student entered by the
    student_last_name: str = ''  # Holds the last name of a student entered by the us
    course_name: str = ''  # Holds the name of a course entered by the user.

    # (jnb 8.6.24) created function to be called for any error handling messages
    7 usages
    @staticmethod
    #  check if this works for file not found
    def output_error_messages(message: str, error: Exception = None):
        """
        Formats the error message and prints on screen
        ChangeLog: (Who, When, What)
        JBoardman, 8.6.2024, created function
        """
        print(message, end="\n\n")
        if error is not None:
            print("-- Technical Error Message -- ")
            print(error, error.__doc__, type(error), sep='\n')

    # (jnb 8.6.24) created function to display menu output
    1 usage
    @staticmethod
```

Figure 1.3: IO Class

Functions

At the core of this week's learning was the application of Functions in a Python script. Functions enable the script to reuse blocks of code when the program runs. The functions are like pre-cutting all of the materials before building a fence. Then when you build the fence all you

need to do is assemble. Each function has a small (modular) piece of code that does something. The function has Parameters where variables are passed into the function. The Arguments are how values are passed into the variables of the function. Figure 1.4 shows the output_student_courses Function, which has a Parameter of student_data and an Argument of list.

```python
def output_student_courses(student_data: list):
    """
    Display Student data on screen
    ChangeLog: (Who, When, What)
    JBoardman, 8.6.2024, created function
    """
    print("-" * 50)
    for student in student_data:
        print(f'Student {student["FirstName"]} '
              f'{student["LastName"]} is enrolled in {student["CourseName"]}')
    print("-" * 50)
```

Figure 1.4 Functions, Parameters and Arguments

For Assignment06, I created the following seven functions within these two classes:

FileProcesser Class

- read_data_from_file
- write_data_to_file

IO Class

- output_error_messages
- output_menu
- input_menu_choice
- output_student_courses
- input_student_data

Body of Script

Once the classes and functions were created, it was time to assemble the script. To do this, I created a section in the script where each of the functions would be called. This section started

with reading the file, using the FileProcessor class, and then used functions from the IO class to present the menu and input/output data (Figure 1.5). With this format, it is easy to understand what the script is doing while still being able to review the functions at the beginning of the script.

```
# Beginning of the main body of this script
students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)


# Present and Process the data
while True:
    # (jnb 8.6.2024) display the menu
    IO.output_menu(menu=MENU)

    # (jnb 8.6.2024) collect menu input
    menu_choice = IO.input_menu_choice()

    # (jnb 8.6.2024) Process inputs
    # (jnb 8.6.2024) Present the current data
    if menu_choice == "1":  # Get new data (and display the change)
        students = IO.input_student_data(student_data=students)
        #IO.output_student_courses(student_data=students)  # (jnb 8.6.24) similar to Lab3 for
        continue

    # (jnb 8.6.2024) Present the current data
    elif menu_choice == "2":  # Display current data
        IO.output_student_courses(student_data=students)
        continue

    # (jnb 8.6.2024) Save the data to a file
    elif menu_choice == "3":  # Save data in a file
        FileProcessor.write_data_to_file(file_name=FILE_NAME, student_data=students)
        print("The following data was saved to file!")  # (jnb 8.6.24) outside of the functio
        IO.output_student_courses(student_data=students)
        continue
```

Figure 1.5: Body of Script

Testing

The final step in completing Assignment06 was to test the script in both PyCharm and the terminal. To run the program in PyCharm I selected the ‘Run’ button and was prompted to

select from the MENU. Going through each input value, I was able to test the entire script and with the final option ‘4’ I exited the program (Figure 1.6).

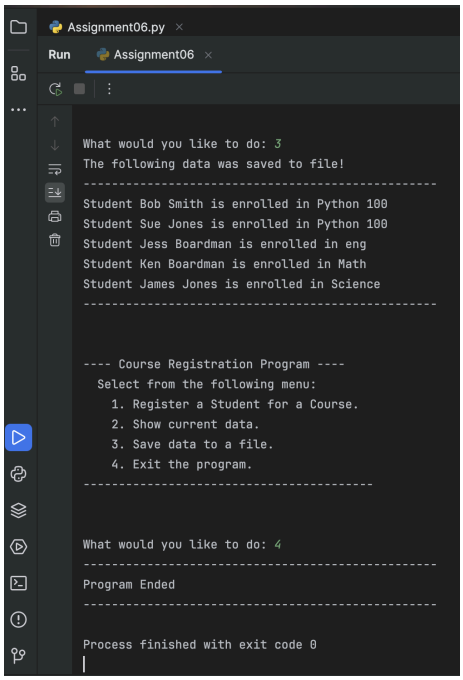

Figure 1.6: PyCharm Testing

The final testing was completed in the terminal to simulate End User Testing. After opening the terminal, I used the cd command to navigate to the correct directory. Once I was in the correct directory, I executed the program by entering python3 Assignment06.py in the terminal window. This ran the program and I was able to read the existing file, provide multiple student inputs, display the data and write to the .json (Figure 1.7).

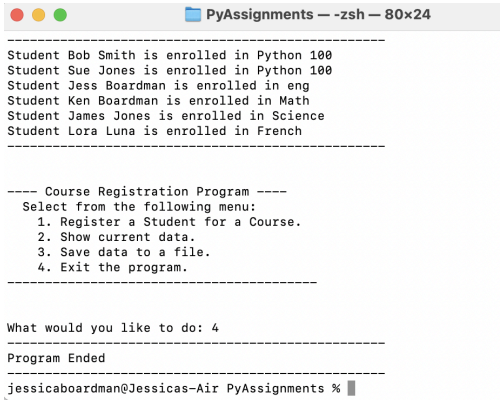

Figure 1.7: Terminal Output

Summary

This week I completed Assignment06 and was able to create functions and classes in my Python script. Applying the principle of Separations of Concerns, the script was organized into the various layers of data, processing, and presentation. Error handling was built as a function and called into the UI without the need to recreate the error handling code. The script was tested in both PyCharm and the terminal to verify it worked as expected and met all documented acceptance criteria. The final script was uploaded to Github for peer review by my classmates.