Jeremy Peters
November 12, 2024
IT FDN 110 B
Assignment 06
Files in GitHub

Functions, Classes & Structured Error handling

Introduction

In this lesson, we dove into the subjects of classes, functions, and structured error handling. I had to use many references to information in the lesson to finish this one with the ramp-up in complexity. I chose to reuse the code from my previous assignment and spent too much time struggling to get my custom messaging and color coding to work. Unfortunately, I had to lose my color coding of the output because I couldn't get the custom messaging and color coding to work together. I was determined to keep it though and found the *colorama* class online that allowed me to keep the text colorization.

Importing Classes

After the header, we open with importing the class items that we will need to leverage later in the code. We had to use *import json* so we could leverage the functionality in that class to read and write data from a JSON file type.

```
def read_data_from_file(file_name: str, student_data: list):
    """
    A function to read data from a file

    :param file_name: name of the file
    :param student_data: list of students
    """
    try:
        with open(file_name, "r") as file:
            student_data = json.load(file)
            if isinstance(student_data, list):
                students.extend(student_data)
    except Exception as e:
        IO.output_error_messages(
            message=CustomMessage.read_file_error, error=e)
    return student_data
```

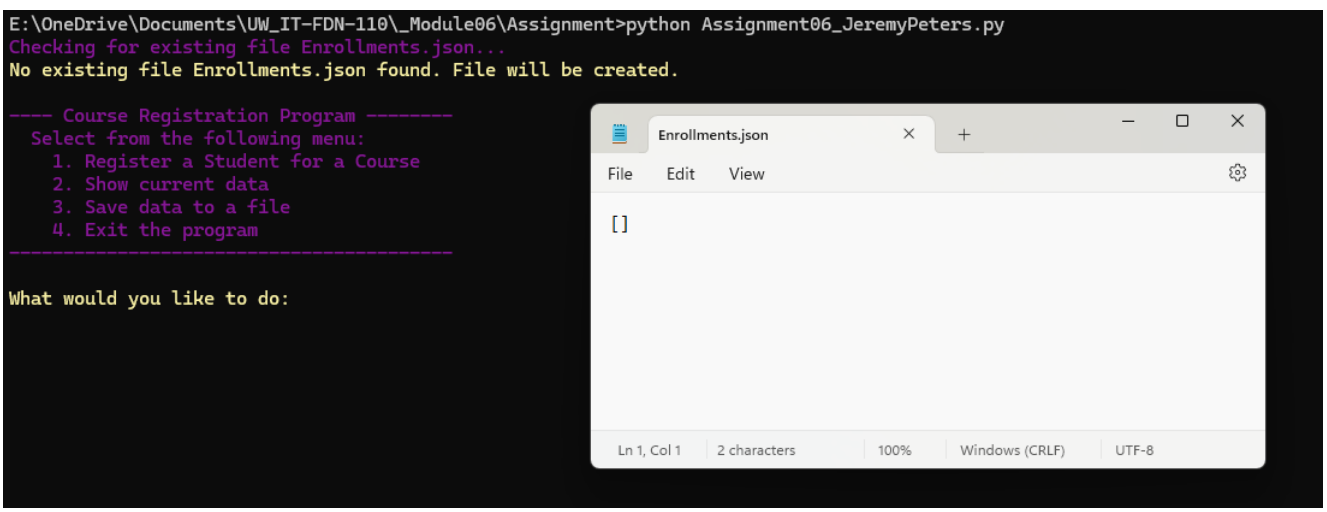
In my **read_data_from_file** function above, I'm populating the **student_data** variable with data from the JSON file with **json.load(file)**. I had some problems using the *.append* method with data from the JSON file, which caused the data to be appended as a single object. To resolve this, I found some information online that allowed me to run an *if* statement, and instead of using the *.append* method, I used the *.extend* method, which properly structured the variable as a *list* of *lists* as expected.

In addition to importing the JSON class, I also used **import os**, which allowed me to run a check to see if the JSON file exists at the start of the program's execution.

```python
    @staticmethod  1 usage
    def filecheck():
        """
        Checks that the file exists and if not, creates an empty file.
        """

        try:
            print(Fore.MAGENTA + f"Checking for existing file {FILE_NAME}..."
                  + Style.RESET_ALL)
            if (not os.path.exists(FILE_NAME) or os.path.getsize(FILE_NAME)
                    == 0):
                print(CustomMessage.no_file_create_it)
                with open(FILE_NAME, "w") as file:
                    file.write("[]")
            else:
                print(CustomMessage.file_exists)
        finally:
            pass
```

I wanted to keep the functionality from my previous assignment code, so I added the **file_check** function, which checks to see if the JSON file already exists and whether it has any data in it. I did this because it isn't enough to have an empty file. It also needs to be a valid JSON file by including the "**[]**" string in it. If the file doesn't exist at all or if the file size is zero, I create it with the expected JSON base by adding the "**[]**" string.



I also used **from colorama import Fore, Style**, which allowed me to keep my text colorization functionality without having to rewrite an entire function just for that purpose. I had spent too much time trying to convert my previous colorization functionality to a function, with nothing to show for it, that I instead chose to leverage existing public code.

```
class CustomMessage:  12 usages
    """
    Class for storing reusable messages
    """
    file_exists: str = (Fore.LIGHTYELLOW_EX + f"File {FILE_NAME} already \
exists. Skipping file creation." + Style.RESET_ALL)
    no_file_create_it: str = (Fore.LIGHTYELLOW_EX + f"No existing file \
{FILE_NAME} found. File will be created." + Style.RESET_ALL)
    prompt_firstname: str = (Fore.LIGHTYELLOW_EX + f"Please enter the \
student's first name: " + Style.RESET_ALL)
    prompt_lastname: str = (Fore.LIGHTYELLOW_EX + f"Please enter the \
student's last name: " + Style.RESET_ALL)
    prompt_coursename: str = (Fore.LIGHTYELLOW_EX + f"Please enter the course\
 name: " + Style.RESET_ALL)
    no_data: str = (Fore.LIGHTCYAN_EX + f"You have not entered any data.\n\
Try starting with starting option 1." + Style.RESET_ALL)
    alpha_only: str = (Fore.LIGHTCYAN_EX + f"Student name should only contain\
 alphabetic characters." + Style.RESET_ALL)
    ascii_only: str = (Fore.LIGHTCYAN_EX + f"Course name should only contain \
ascii characters." + Style.RESET_ALL)
    valid_choices: str = (Fore.LIGHTCYAN_EX + f"Invalid choice. Please try \
again." + Style.RESET_ALL)
    registered_students: str = (Fore.MAGENTA + f"The following students are \
registered:" + Style.RESET_ALL)
    read_file_error: str = (Fore.RED + f"Error reading contents of \
{FILE_NAME}." + Style.RESET_ALL)
```

I leveraged the *colorama* class heavily in the **CustomMessage** class I created. While this block looks a little messy, doing this meant the rest of my code was clean and legible.

Classes and Functions

In this lesson, we learned that classes are used to group similar functions together. For this assignment, we used the **FileProcessor** class, which grouped the **read_data_from_file** and **write_data_to_file** functions, along with my own **file_check** function, together. We also used the **IO** class, which grouped the **output_error_messages**, **output_menu**, **input_menu_choice**, **input_student_data**, and **output_student_courses** functions together, and I've provided details on my **CustomMessage** class above.

For each of the functions added to a class, we included an **@staticmethod** statement above it, which tells Python that the function does not interact with the class itself, but allows us to call the function using the "class.function" syntax. The ".function" portion is the "static method", similar to the builtin class methods we're already using, like *.open* in *file.open()*, which is why we declare it as such.

Structured Exception Handling

In my program, I'm really only handling two exceptions, but that covers the two types that I'm catching in other functions. I've collected the *Exception* and *ValueError* type exceptions into the **output_error_messages** function and added some of the *colorama* color coding to colorize them as errors. Using colors for errors and warning messages is standard in most console programs and familiar to most users.

```
    @staticmethod  9 usages
    def output_error_messages(message: str, error: Exception = None):
        """
        A function that handles error messages

        :param message: message data passed to the function
        :param error: Exception data passed to the function
        """
        print(message)
        if error is not None:
            print(Fore.LIGHTRED_EX + "-- Error Message -- ")
            print(error, error.__doc__, type(error), sep="\n" +
                                                    Style.RESET_ALL)
        elif error == ValueError:
            print(Fore.RED + f"-- Error -- " + Style.RESET_ALL)
            print(Fore.RED + error.__str__() + Style.RESET_ALL)
```

```python
    @staticmethod  2 usages (1 dynamic)
    def input_student_data(student_data: list):
        """
        A function that handles the user input

        :param student_data: Stores the students list data
        """

        try:
            while True:
                try:
                    student_first_name = input(CustomMessage.prompt_firstname)
                    if not student_first_name.isalpha():
                        raise ValueError(CustomMessage.alpha_only)
                    student_first_name = student_first_name.title().strip()
                    break
                except ValueError as e:
                    IO.output_error_messages(e.__str__())

            while True:
                try:
                    student_last_name = input(CustomMessage.prompt_lastname)
                    if not student_last_name.isalpha():
                        raise ValueError(CustomMessage.alpha_only)
                    student_last_name = student_last_name.title().strip()
                    break
                except ValueError as e:
                    IO.output_error_messages(e.__str__())

            while True:
                try:
                    course_name = input(CustomMessage.prompt_coursename)
                    if not course_name.isascii():
                        raise ValueError(CustomMessage.ascii_only)
                    course_name = course_name.title().strip()
                    break
                except ValueError as e:
                    IO.output_error_messages(e.__str__())

            student = {"FirstName": student_first_name,
                       "LastName": student_last_name,
                       "CourseName": course_name}
            students.append(student)
            print(Fore.MAGENTA + f"You have added {student_first_name} \
{student_last_name} for course {course_name} to the registration list." \
+ Style.RESET_ALL)
        except Exception as e:
            IO.output_error_messages(message=Fore.RED + f"There was a \
non-specific error!\n" + Style.RESET_ALL, error=e)
```

In my **input_student_data** function, I want the user to input specific information. When they enter unintended data, I raise a *ValueError* type exception and pass the information to the **IO.output_error_messages** class & function to present the appropriate message to the user. I'm also catching any other exceptions with Exception, passing a friendly error and any exception details to the **output_error_messages** function.

```
What would you like to do: 2
You have not entered any data.
Try starting with starting option 1.

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

What would you like to do: 3
The following was saved to file:
You have not entered any data.
Try starting with starting option 1.

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

What would you like to do: |
```

Descriptive Document Strings

In this lesson, we also learned about adding documentation to our class and function code. This provides contextual help while working with the code.

```
    @staticmethod  1 usage
    def write_data_to_file(file_name: str, student_data: list):
        """
        A function to write data to a file

        :param file_name: The name of the file
        :param student_data: The list of students
        """
        try:
            with open(file_name, "w") as file:
                json.dump(student_data, file)
                print(Fore.LIGHTYELLOW_EX + f"The following was saved to \
file:" + Style.RESET_ALL)
                IO.output_student_courses(student_data=students)
                if student_data == str():
                    raise ValueError(CustomMessage.no_data)
        except ValueError as e:
            IO.output_error_messages(e.__str__())
        except Exception as e:
            IO.output_error_messages(e.__str__())
```

In my **write_data_to_file** function above, I've added a description of the function, as well as documentation on the parameters that can be passed to the function, **file_name**, and **student_data**. This allows me to hover over calls to this function elsewhere in my code and get a description of the function and the parameters, saving me time from finding the section of code to figure out what it does.

Summary

I admit that I was “spinning my wheels” on the color coding, which delayed my progress and completion of this assignment. That said, this lesson and assignment were enjoyable because we’re getting closer to writing programs instead of scripts. It was hard to unpack everything I had been building to this point, and then restructure it into classes and functions, but I feel like I’m getting a good feel for how to create them and, possibly more importantly, how to research solutions for problems encountered.

When I run my program, it more-or-less does the same thing it did last lesson without using classes and functions. By writing it with classes and functions, it makes the code much more legible and reusable going forward.

```
E:\OneDrive\Documents\UW_IT-FDN-110\_Module06\Assignment>python Assignment06_JeremyPeters.py
Checking for existing file Enrollments.json...
File Enrollments.json already exists. Skipping file creation.

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

What would you like to do: 1
Please enter the student's first name: vic
Please enter the student's last name: VU
Please enter the course name: pYtHoN 100
You have added Vic Vu for course Python 100 to the registration list.

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

What would you like to do: 2
Vic Vu is enrolled in 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
----------------------------------------

What would you like to do: 3
The following was saved to file:
Vic Vu is enrolled in 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
----------------------------------------

What would you like to do: 4
Program closed successfully
```

Enrollments.json

File Edit View


```
[
  {
    "FirstName": "Vic",
    "LastName": "Vu",
    "CourseName": "Python 100"
  }
]
```


Ln 1, Col 1 | 88 characters | 100% | Windows (CRLF) | UTF-8