Nathan Thomas

November 18, 2025

IT FND 110 A

Assignment 6

https://github.com/nthomas28/IntroToProg-Python_Mod06

Programming with Functions and Classes

Introduction

For our assignment this week, we were tasked with using **classes** and **functions** to clean up our Assignment 5 programs, making them more developer-friendly. To accomplish this, we broke down our code into separate functions that we imbedded into either the class **FileProcessor** or the **IO** class. Our **IO** class is designated to contain our functions used for student data collecting and error messaging, while the **FileProcessor** class will contain our **json** file functions.

Defining Constants & Variables

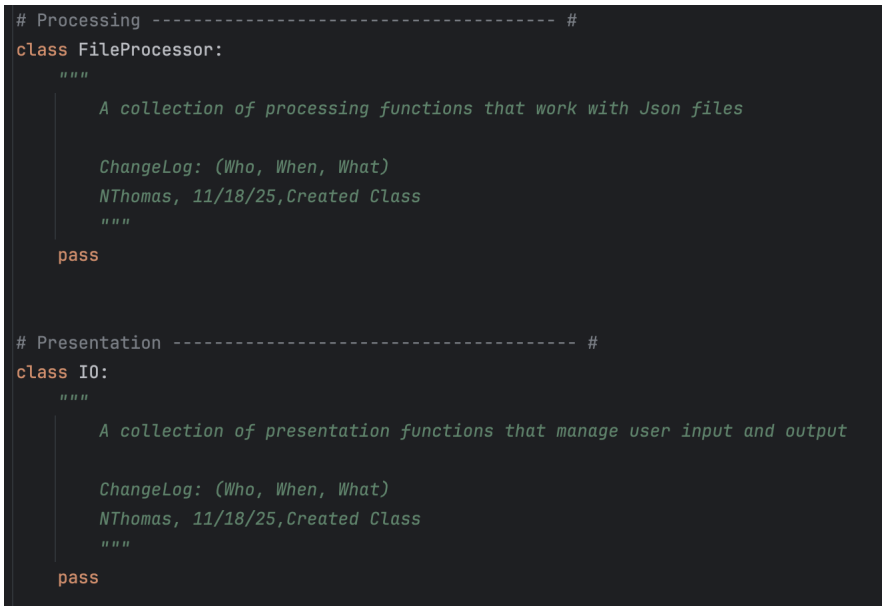
While our constants remained the same from our Assignment 5 program, we worked with fewer variables initially for this program. This program utilizes local variables that were defined as parameters for our various functions. The variables **students** and **menu_choice** are considered global variables, which would be utilized throughout our program.

```
# 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"

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

Figure: The constants and global variables used in this program.

Defining Our Classes

The majority of our program will be contained within the classes, **FileProcessor** and **IO**. Our **FileProcessor** class will contain the functions that would gather (read) the data from our **json** file as well as save (write) to our **json** file, depending upon a menu selection. The **IO** class would encompass the student data gathering from our student registration menu prompt, as well as the error handling for this program. I initially included the code **pass** as a means to skip over classes while I was programming. These were removed once I began adding our functions to these classes.

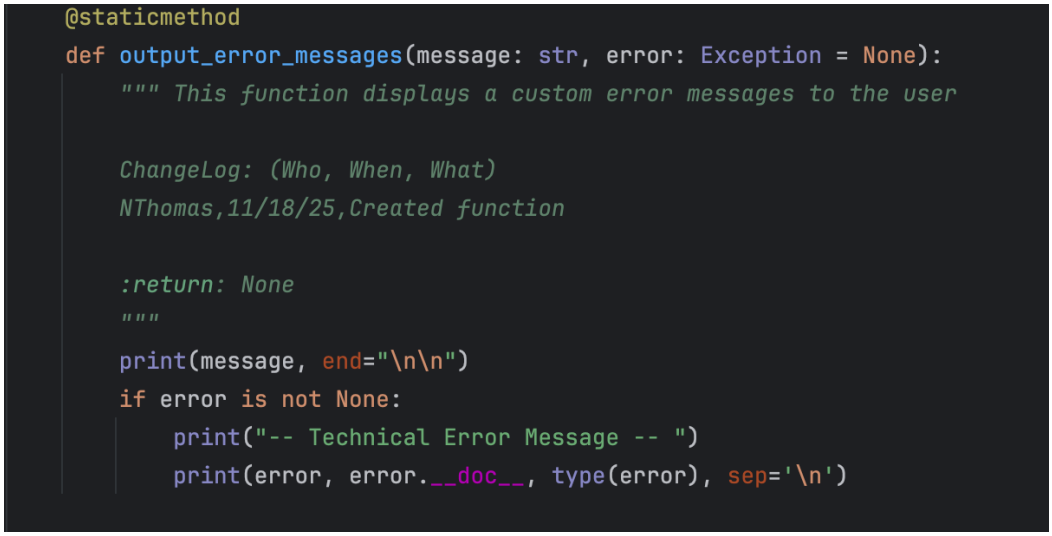


```
# Processing ------------------------------------- #
class FileProcessor:
    """
        A collection of processing functions that work with Json files

        ChangeLog: (Who, When, What)
        NThomas, 11/18/25,Created Class
        """
    pass


# Presentation ------------------------------------- #
class IO:
    """
        A collection of presentation functions that manage user input and output

        ChangeLog: (Who, When, What)
        NThomas, 11/18/25,Created Class
        """
    pass
```

Figure: These are the classes used to organize our functions.

IO Functions

The first function to be included in this section was our **output_error_messages()**, which used the parameters **message** and **error.** This function operates by displaying an error message whenever prompted by our program, due to potential user error.



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

        ChangeLog: (Who, When, What)
        NThomas,11/18/25,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: The function to produce generalized messages for error popups.

Next, I programmed the function to prompt our menu, **output_menu().** This function uses the parameter **menu,** which correlates to our constant **MENU**.

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

    ChangeLog: (Who, When, What)
    NThomas,11/18/25,Created function

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

Figure: The menu function, which presents our menu to a user.

To finish off our menu for users, we needed a function for a user's input. For this program we called this function **input_menu_choice()**, which didn't include a parameter. This function contains a local variable called **choice** with the initial value of "0", which was used instead of a parameter. A user is then prompted to enter a valid menu option. This function also refers to our previous **output_error_messages()** function, displaying an error message if an invalid menu option is entered.

```python
@staticmethod  1 usage
def input_menu_choice():
    """ This function gets a menu choice from the user

    ChangeLog: (Who, When, What)
    NThomas,11/18/25,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: The menu selection function that allows a user to interact with our menu.

Our next function calls upon the data currently stored into our **students()** list variable from our **json** file. This data is presented to a user in the form of a list of registered students in a **csv** format.

```
@staticmethod  2 usages
def output_student_courses(student_data: list):
    """ This function presents student course data to the user

    ChangeLog: (Who, When, What)
    NThomas,11/18/25,Created function

    :return: None
    """
    # Process the data to create and display a custom message
    print("-" * 50)
    for student in student_data:
        print(f"{student['FirstName']}, {student['LastName']}, {student['CourseName']}")
    print("-" * 50)
    print()
```

Figure: The code used to present our current list of registered students.

Our final function that has been included in our **IO** class is **input_student_data()** with the parameter **student_data**. This function prompts a user for student registration information which is then added to our growing list of student registrations. This function also calls upon our **output_error_messages()** function if an invalid name is enter for student registration.

```python
@staticmethod  1 usage
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)
    NThomas,11/18/25,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)
    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: This is the code used to prompt a student's registration data and add the data to our student registration list.

FileProcessor Functions

The **FileProcessor** class functions all involve access to our **json** file, which is why they have been grouped together. The first function included in this class is our **read_data_from_file()**, which uses the parameters **file_name** and **student_data**. This function will extract (read) the data currently stored in our **json** file and save that data to our **student_data** parameter. This function also references our **output_error_messages()** function to present an error message if our **json** file doesn't already exist.

```python
@staticmethod  1 usage
def read_data_from_file(file_name: str, student_data: list):
    """ This function reads data from a json file

    Changelog: (Who, When, What)
    NThomas, 11/18/25,Created function
    param: file_name
    return: student_data

    """
    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: The code used to read the data from our json file and save it to student_data.

The last function for our **FileProcessor** class is our **write_data_to_file()** function, with the parameters **file_name** and **student_data.** This function allows our program to save any new student registration information from **student_data** to our **json** file in an easy to read format. This function also includes the possibility for an error message to display, call upon our **output_error_messages()** function from our **IO** class.

```python
@staticmethod  1 usage
def write_data_to_file(file_name: str, student_data: list):
    """
    This function writes data to a json file
    Changelog: (Who, When, What)
    NThomas, 11/18/25,Created function

    return: student_data
    """

    try:
        file = open(file_name, "w")
        json.dump(student_data, file, indent=2)
        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()
```

Figure: The code used to save our student registration data to our json file.

Conclusion

The use of classes and functions makes program organization much more intuitive compared to our previous assignments. We can divide our functions into different classes based on the similarities that functions have in relation to our program. In this program, our functions were separated into **FileProcessor** for the processing functions and **IO** for the presentation functions, error messaging included. This level of organization enables us to simply call upon a class and a specific function when coding the functionality of our program menu. We simply need to associate a menu option to the corresponding function that we programmed.

```python
# When the program starts, read the file data into a list of lists (table)
students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)

# Present and Process the data
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!
        students = IO.input_student_data(student_data=students)
        IO.output_student_courses(student_data=students)
        continue

    # Present the current data
    elif menu_choice == "2":
        IO.output_student_courses(student_data=students)  # Added this to improve user experience
        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")
```

Figure: This code demonstrates the calls to specific classes and functions within this program. (i.e. FileProcessor.write_data_to_file().)