Alicia Davis
20 May 2024
IT FDN 110 B
Assignment 06

# Functions & Classes

## Introduction

This assignment builds on last week's assignment of creating a menu program. Instead of writing the code inline for each menu option, I created classes to group my code according to the separation of concerns, and functions that are called in the main body of code. Additionally, we're leveraging a JSON file now instead of a csv file. The code along with this knowledge document is also hosted on my GitHub repository here.

## Creating the Script

Similar to the last assignments, I start my script off with a header, importing the exit and JSON functions and defining the constants and variables. The constants stayed the same as last week (minus the csv is now a JSON), however because I leveraged functions to define local variables, the number of global variables is down to two (Figure 1.1).

```
import json
from sys import exit

# Constants and global variables
MENU: str = """
    ---- Course Registration Program ----
    Select from the following menu:
    1. Register a student for the course
    2. Show current data
    3. Save data to file
    4. Exit the program
    -----------------------------------
"""
FILE_NAME: str = "enrollments.json"

menu_choice: str = ""
students: list = []
```

**Figure 1.1: Updated import functions, constants and global variables.**

Next I defined the class "FileProcessor", which is the data layer of the code. This class contains two functions, and is meant to handle the data/file processing for the program. Because this class contains functions that can't modify the class, they are both marked as "staticmethod" (Figure 1.2).

```
# Data Layer / File processing block
class FileProcessor:
    """Class to handle data storage and retrieval"""

    @staticmethod
    def read_data_from_file(file_name: str, student_data: list):···

    @staticmethod
    def write_data_to_file(file_name: str, student_data: list):···
```

**Figure 1.2: FileProcessor class with two staticmethod functions.**

The first FileProcessor function is for reading data from a file, while the second is for writing data to a filel. Both contain built-in error handling logic, however because error handling messaging is handled in my next class, these functions call those functions rather than containing their own print messages. Both functions also leverage the import JSON library to process files and I created dictstrings to define the class and each function (Figure 1.3).

```python
# Data Layer / File processing block
class FileProcessor:
    """Class to handle data storage and retrieval"""

    @staticmethod
    def read_data_from_file(file_name: str, student_data: list):
        """Read file and load to json and return list"""

        try:
            with open(file_name) as file:
                student_data = json.load(file)
        except FileNotFoundError as error_message:
            IO.output_error_messages("\nFile not found.", error_message)
        except Exception as error_message:
            IO.output_error_messages("\nUnknown Error. Please contact support. ", error_message)
        return student_data

    @staticmethod
    def write_data_to_file(file_name: str, student_data: list):
        """Write data to json file"""

        try:
            with open(file_name, "w") as file:
                json.dump(student_data, file)
        except FileNotFoundError as error_message:
            IO.output_error_messages("\nFile not found.", error_message)
        except Exception as error_message:
            IO.output_error_messages("\nUnknown Error. Please contact support.", error_message)
```

**Figure 1.3: Read and write functions under the FileProcessor class with built in error-handling logic.**

The next class is the presentation layer or IO (input/output) block. This contains all of the code that will get input from and give output to a user. Similar to the FileProcessor class, each function doesn't change the class and is therefore marked as "staticmethod" (Figure 1.4).

```python
# Presentation Layer / IO block
class IO:
    """Class to handle user input and output"""

    @staticmethod
    def output_error_messages(message: str, error: Exception = None):...

    @staticmethod
    def output_menu(menu: str):...

    @staticmethod
    def input_menu_choice():...

    @staticmethod
    def input_student_data(student_data: list):...

    @staticmethod
    def output_student_courses(student_data: list):...
```

**Figure 1.4: IO class containing five functions.**

The first IO function is to handle error messaging across the program. This function is called in several other functions in the program and provides standardized error messaging that provides the error, the error docstring and the type of the error separated by a carriage return (Figure 1.5)

```python
@staticmethod
def output_error_messages(message: str, error: Exception = None):
    """Standardized error messages for program"""

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

**Figure 1.5: Error messaging function in IO class.**

The next two functions output the menu constant to the user and process the user's menu input respectively. The input menu function also contains error handling logic for if a user selects an invalid menu option. As will the FileProcessor functions, instead of re-writing the error message handling, it calls the standard error messaging function (Figure 1.6).

```
@staticmethod
def output_menu(menu: str):
    """Display menu options to user"""

    print(menu, end="\n\n")

@staticmethod
def input_menu_choice():
    """Process user's menu choice"""

    choice = "0"
    try:
        choice = input("Choose a menu option (1-4): ")
        if choice not in ("1", "2", "3", "4"):
            raise Exception("Invalid option.  Please choose between 1-4.")
    except Exception as error_message:
        IO.output_error_messages("\nUnknown Error.", error_message)
    return choice
```

**Figure 1.6: Menu output and input functions.**

The next function is for consuming the user's input data for student registration (name and course). This contains specific error handling in case the user tries to enter an empty value across all fields or a non-alphanumeric value for the name field. Once the user inputs correctly formatted data, it creates a dictionary list and appends it to the student_data dictionary (Figure 1.7).

```
@staticmethod
def input_student_data(student_data: list):
    """Process user's input and append to dictionary"""

    try:
        student_first_name = input("Enter the student's first name: ")
        if len(student_first_name) == 0:
            raise ValueError("First name can't be empty.")
        if not student_first_name.isalpha():
            raise ValueError("First name can't contain non-alphanumeric values.")
        student_last_name = input("Enter the student's last name: ")
        if len(student_last_name) == 0:
            raise ValueError("Last name can't be empty.")
        if not student_last_name.isalpha():
            raise ValueError("Last name can't contain non-alphanumeric values.")
        course_name = input("Enter the course name: ")
        if len(course_name) == 0:
            raise ValueError("Course name can't be empty.")
        student = {
            "FirstName": student_first_name,
            "LastName": student_last_name,
            "CourseName": course_name
        }
        student_data.append(student)
        print()
        print(f"You have registered {student_first_name} {student_last_name} for {course_name}.")
    except ValueError as error_message:
        IO.output_error_messages("\nInvalid Entry.  See details below.", error_message)
    except Exception as error_message:
        IO.output_error_messages("\nUnknown Error. Please contact support.", error_message)
```

**Figure 1.7: Input student data function with error handling.**

The last IO class function is to output the current data to the user. This function takes the data in the student_data dictionary and loops through each value returning it as a comma-separated string to the user (Figure 1.8).

```
@staticmethod
def output_student_courses(student_data: list):
    """Output current data to user"""

    print("-"*50)
    print("The current data is: ")
    for student in student_data:
        print(f"{student["FirstName"]}, {student["LastName"]}, {student["CourseName"]}")
    print("-"*50, end="\n\n")
```

**Figure 1.8: Output student data function.**

The last block of code is the processing layer or the execution block. This is indicated by both my comment and the "__main__" name which is used to define the top-level of the code. The code functions the same way as last week's assignment as the file is loaded as soon as the program begins and then loops through the menu options as chosen by the user until the user chooses to end the program.

```
# Processing Layer / Execution block
if __name__ == "__main__":
    students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)

    while True:
        IO.output_menu(MENU)
        menu_choice = IO.input_menu_choice()
        match menu_choice:
            case "1":
                IO.input_student_data(student_data=students)

            case "2":
                IO.output_student_courses(student_data=students)

            case "3":
                FileProcessor.write_data_to_file(file_name=FILE_NAME, student_data=students)
                print("INFO: Registrations have been saved.")

            case "4":
                print("Program Ended.")
                exit()
```

**Figure 1.9: Execution block of code pulling in all functions across classes and defining the arguments for each function's parameters.**

# Testing the Script

I tested my code in both VSCode's terminal and IDLE. My testing followed the same steps as last week. I checked that my error messages for the file not existing (Figure 2.1) and invalid options entered as part of menu option 1 worked as expected (Figure 2.2).

```
PS C:\Users\adavi\OneDrive\Documents\UW\FDN110 - Foundations of Python\Module06\Assignment> python assignment06.py

File not found.

--- Error Details ---
[Errno 2] No such file or directory: 'enrollments.json'
File not found.
<class 'FileNotFoundError'>
```

**Figure 2.1: File not found error message when launching program.**

```
Choose a menu option (1-4): 1
Enter the student's first name:

Invalid Entry.  See details below.

--- Error Details ---
First name can't be empty.
Inappropriate argument value (of correct type).
<class 'ValueError'>
```

```
Choose a menu option (1-4): 1
Enter the student's first name: Alicia
Enter the student's last name:

Invalid Entry.  See details below.

--- Error Details ---
Last name can't be empty.
Inappropriate argument value (of correct type).
<class 'ValueError'>
```

```
Choose a menu option (1-4): 1
Enter the student's first name: Alicia
Enter the student's last name: Davis
Enter the course name:

Invalid Entry.  See details below.

--- Error Details ---
Course name can't be empty.
Inappropriate argument value (of correct type).
<class 'ValueError'>
```

```
Choose a menu option (1-4): 1
Enter the student's first name: 4

Invalid Entry.  See details below.

--- Error Details ---
First name can't contain non-alphanumeric values.
Inappropriate argument value (of correct type).
<class 'ValueError'>

    ---- Course Registration Program ----
    Select from the following menu:
    1. Register a student for the course
    2. Show current data
    3. Save data to file
    4. Exit the program
    ----------------------------------------
```

**Figure 2.2: Error messages for blank and non-alphanumeric inputs for menu option 1.**

Once valid data is entered, menu option 2 displays the current data (Figure 1.3).

```
Choose a menu option (1-4): 2
-------------------------------------------------
The current data is:
Bob, Smith, FDN 110
Sue, Jones, FDN 110
Alicia, Davis, FDN 110
Thierry, Henry, FDN 110
David, Beckham, FDN 110
-------------------------------------------------
```

**Figure 2.3: Current data in the "student" list.**

Menu option 3, creates the enrollments.json file and saves the new user input to the file (Figure 2.4).

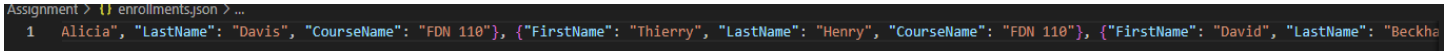


**Figure 2.4: JSON file with data from user input.**

Any input outside 1-4 still results in an invalid option message (Figure 2.5), and option 4 still ends the program (Figure 2.6).

```
Choose a menu option (1-4): 5

Unknown Error.

--- Error Details ---
Invalid option.  Please choose between 1-4.
Common base class for all non-exit exceptions.
<class 'Exception'>
```

**Figure 2.5: Invalid option**

```
Choose a menu option (1-4): 4
Program Ended.
```

**Figure 2.6: Program Ended**

Similar to in VSCode's terminal, my testing followed the same steps in IDLE, except since the file exists from my testing in terminal, I don't receive the error message that the file can't be found. I do receive the correct error message for invalid inputs in menu option 1 though (Figure 2.7).

```
Choose a menu option (1-4): 1
Enter the student's first name:

Invalid Entry.  See details below.

--- Error Details ---
First name can't be empty.
Inappropriate argument value (of correct type).
<class 'ValueError'>
```

```
Choose a menu option (1-4): 1
Enter the student's first name: Harry
Enter the student's last name:

Invalid Entry.  See details below.

--- Error Details ---
Last name can't be empty.
Inappropriate argument value (of correct type).
<class 'ValueError'>
```

```
Choose a menu option (1-4): 1
Enter the student's first name: Harry
Enter the student's last name: Kane
Enter the course name:

Invalid Entry.  See details below.

--- Error Details ---
Course name can't be empty.
Inappropriate argument value (of correct type).
<class 'ValueError'>
```

```
Choose a menu option (1-4): 1
Enter the student's first name: H4rry

Invalid Entry.  See details below.

--- Error Details ---
First name can't contain non-alphanumeric values.
Inappropriate argument value (of correct type).
<class 'ValueError'>
```

**Figure 2.7: Error messages for blank and non-alphanumeric inputs for menu option 1.**

Once valid data is entered, I can see these users were added to my master list when selecting option 2 (Figure 2.8).

```
Choose a menu option (1-4): 2
-------------------------------------------------
The current data is:
Bob, Smith, FDN 110
Sue, Jones, FDN 110
Alicia, Davis, FDN 110
Thierry, Henry, FDN 110
David, Beckham, FDN 110
Harry, Kane, FDN 110
Manuel, Neuer, FDN 110
-------------------------------------------------
```

**Figure 2.8: New users added to master list.**

Option 3 saves the data as comma-delimited to the existing csv file (Figure 2.9 & Figure 2.10).

```
Choose a menu option (1-4): 3
INFO: New registrations have been saved to file.
```

**Figure 2.9: Message to user that new input has been saved.**

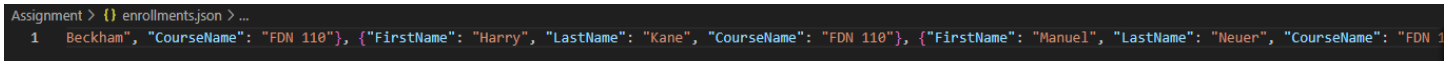


**Figure 2.10: Updated JSON File**

Any input outside 1-4 still results in an invalid option message (Figure 2.11), and option 4 still ends the program (Figure 2.12).

```
Choose a menu option (1-4): 5

Unknown Error.

--- Error Details ---
Invalid option.  Please choose between 1-4.
Common base class for all non-exit exceptions.
<class 'Exception'>
```

**Figure 2.11: Invalid option**

```
Choose a menu option (1-4): 4
Program Ended
>>>
```

**Figure 2.12: Program Ended**

# Summary

I found that starting from scratch like Luis showed in class by building an outline and placing “pass” for each function / option in the execution block was easier for me than trying to start with the assignment starter that was part of our class files. This made me really think about what each portion was really trying to accomplish and helped me understand it all better. I also decided to build a smaller program using functions to calculate the radius of a sphere and area of a triangle based on one of the YouTube videos we watched as part of this assignment. Stepping outside of the registration program repetition also helped me understand what we’re trying to accomplish with this lesson.