Fiorella Guerra
August 19, 2025
Foundations of Programming: Python
Assignment 6

https://github.com/fiorellaguerra95/IntroToProg-Python-Mod06

Functions, Classes, and Separation of Concerns

Introduction

Assignment 6 introduced the concepts of functions, parameters, arguments, return values, and classes, with a focus on using the separation of concerns pattern. The previous assignment (Assignment 5) implemented JSON files and structured error handling, but the program logic and input/output were written directly in the script body. In this version, I refactored the course registration program by organizing code into classes (FileProcessor and IO) and functions with descriptive docstrings, making the program easier to maintain, test, and reuse.

From the Module 6 Notes, I learned:

- **Functions** group reusable logic into named blocks of code.
- **Parameters and arguments** allow functions to accept input values, while **return values** let them send results back.
- **Local vs. global variables:** Local variables exist only inside a function, while global variables persist throughout the script.
- **Classes** can be used to group related functions into logical collections.
- **Separation of concerns** is a design principle where responsibilities are divided into layers (input/output vs. data processing).

Organizing Code with Classes and Functions

To implement separation of concerns, I divided the script into two main classes:

1. **FileProcessor (Processing Layer)**
    - Handles all file interactions (read_data_from_file, write_data_to_file).
    - Uses structured error handling to manage file-related issues.
    - Returns data to be used by the rest of the program.
2. **IO (Presentation Layer)**
    - Handles all user interactions (output_menu, input_menu_choice, input_student_data, output_student_data).

- Contains output_error_messages for consistent error reporting.

This structure ensures that if the file processing logic changes (e.g., switching from JSON to CSV), I would only update the FileProcessor class without affecting the rest of the program.

Parameters, Arguments, and Return Values

This assignment highlighted how parameters (placeholders inside a function definition) and arguments (actual values passed in) allow functions to be flexible and reusable.

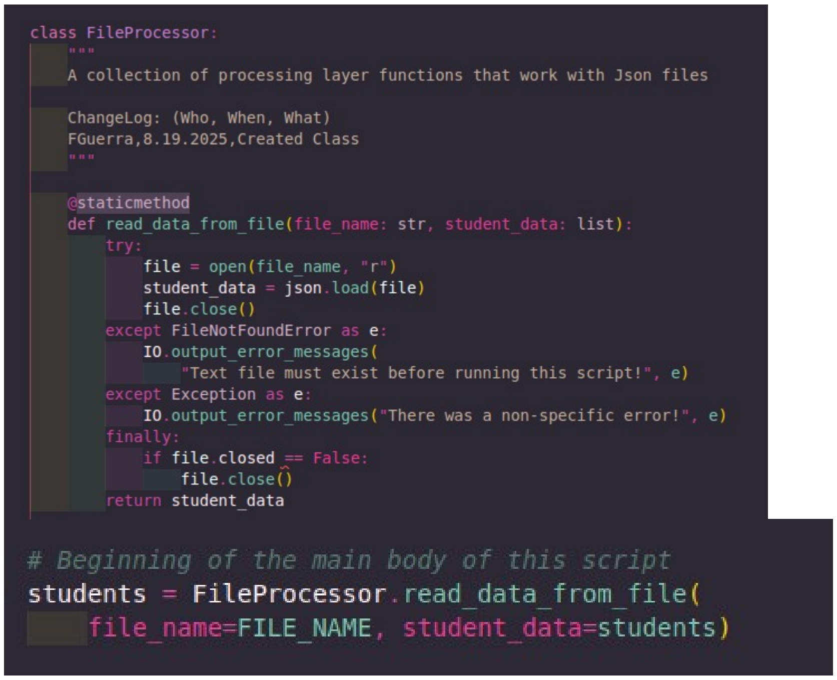
Example from FileProcessor.read_data_from_file:

```
@staticmethod
def read_data_from_file(file_name: str, student_data: list):
    # file_name and student_data are parameters
    ...
    return student_data
```

When called as:

```
students = FileProcessor.read_data_from_file(file_name=FILE_NAME,
student_data=students)
```

… the actual arguments (FILE_NAME and students) are passed in. The function returns the updated students list back to the caller.



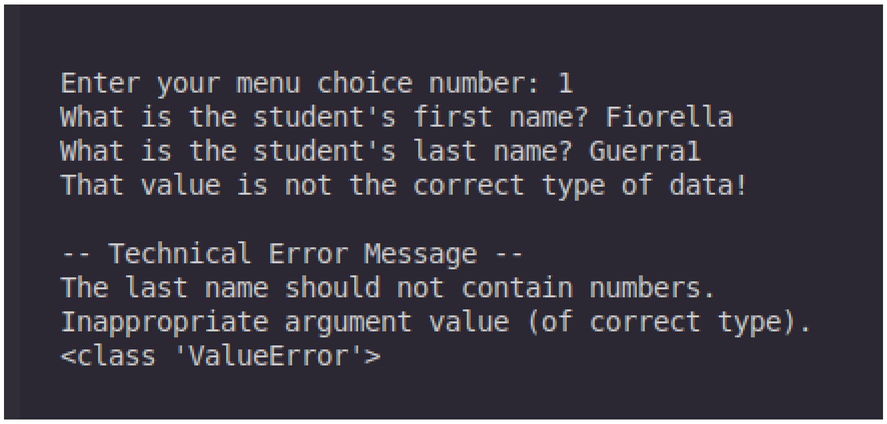
```
class FileProcessor:
    """
    A collection of processing layer functions that work with Json files

    ChangeLog: (Who, When, What)
    FGuerra,8.19.2025,Created Class
    """

    @staticmethod
    def read_data_from_file(file_name: str, student_data: list):
        try:
            file = open(file_name, "r")
            student_data = json.load(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 file.closed == False:
                file.close()
        return student_data

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

Input and Output Functions

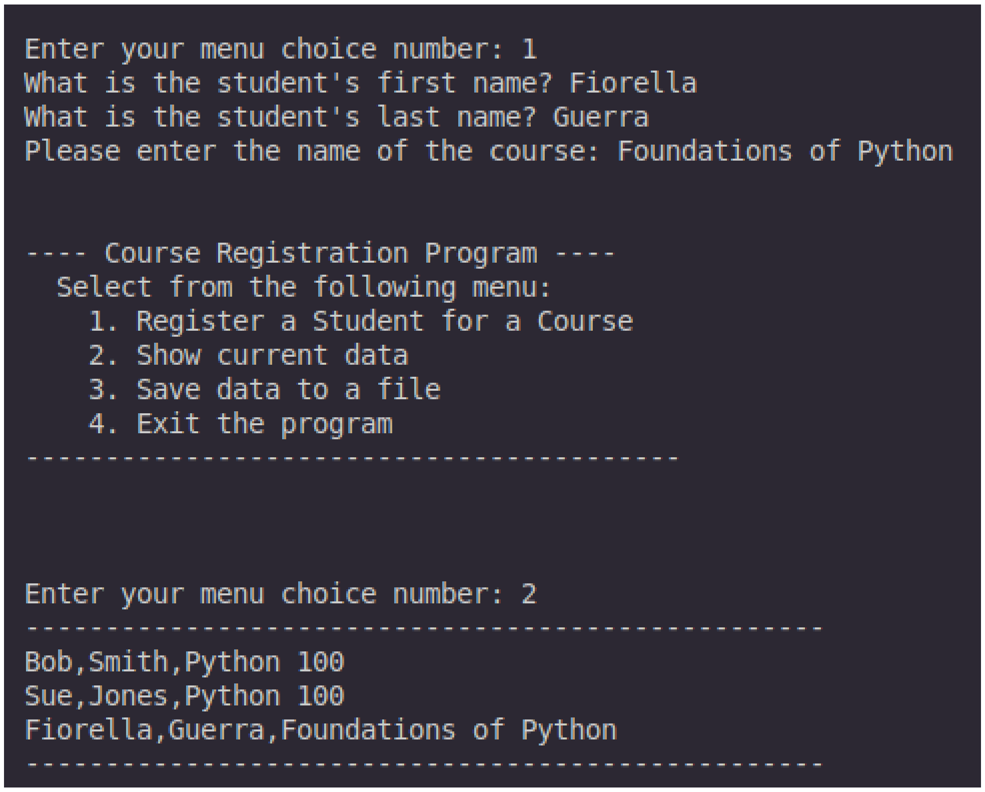
When registering students, the IO.input_student_data function validates input and appends a dictionary to the list. Another function, IO.output_student_data, loops through the list and prints it in CSV format.

```
@staticmethod
def input_student_data(student_data: list)
    try:
        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.")
```

Error:



```
Enter your menu choice number: 1
What is the student's first name? Fiorella
What is the student's last name? Guerra1
That value is not the correct type of data!

-- Technical Error Message --
The last name should not contain numbers.
Inappropriate argument value (of correct type).
<class 'ValueError'>
```



```
Enter your menu choice number: 1
What is the student's first name? Fiorella
What is the student's last name? Guerra
Please enter the name of the course: Foundations of Python


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




Enter your menu choice number: 2
--------------------------------------------------
Bob,Smith,Python 100
Sue,Jones,Python 100
Fiorella,Guerra,Foundations of Python
--------------------------------------------------
```

Summary

Through Assignment 6, I practiced:

- Writing and calling functions with parameters, arguments, and return values.
- Organizing related functions into classes.
- Applying the separation of concerns design principle to split presentation (IO) from processing (FileProcessor).
- Designing functions that work with inputs and outputs rather than relying only on global variables.

This made the course registration program modular and more professional, with clear boundaries between different responsibilities.