

Name: Cathy DuPuis

Date: August 14, 2024

Course: IT FDN 110 Su 24 Foundations of Programming Python

Assignment 06 – Functions, Classes and Error Handling

Introduction

In this assignment, I made a Python program that demonstrates the use of functions, classes, and structured error handling. The program manages student registrations for a course, with functionalities for inputting, displaying, and saving student data to a file using classes and lists. This document covers the key aspects of the program, including Processing, Input and Output, and Error Handling such as for only accepting alphabetic characters for names, while emphasizing the importance of the Separation of Concerns principle.

Topic – Processing using a class

The processing logic in this program is handled primarily by the FileProcessor class (see Figure 1), which is responsible for reading from and writing to the file Enrollments.json. The class includes methods that ensure data is correctly managed for multiple program runs and handles the error messages in case there are issues opening or other technical issues with the file.

Key Components:

- **Reading Data:** The `read_data_from_file` method reads data from the JSON file into a list of dictionary rows. This data is then used throughout the program for managing student registrations.
- **Writing Data:** The `write_data_to_file` method writes the contents of the students list (which contains dictionary rows of student data) back to the JSON file. This ensures that any changes made during the program's execution are saved for future use.

Because these 2 methods don't need to change or use any specific instances of the attributes – they only need the data moved as parameters (like the `file_name` and `student_data`) then I used the `@staticmethod` to define them.

Separation of Concerns:

The processing logic is encapsulated within the FileProcessor class, ensuring that file-related operations are isolated from the rest of the program. This adheres to the Separation of Concerns principle, which promotes modularity and makes the code easier to maintain and extend.

```
class FileProcessor:
    """Processes data to and from a file"""

    1 usage
    @staticmethod
    def read_data_from_file(file_name: str, student_data: list):
        """ Reads data from a JSON file into a list of dictionary rows """
        file = None
        try:
            file = open(file_name, "r")
            student_data.clear()
            student_data.extend(json.load(file))
        except FileNotFoundError as e:
            I0.output_error_messages(f"File '{file_name}' must exist before running this script.\n")
            I0.output_error_messages("Technical error message:")
            print(e, e.__doc__, type(e), sep='\n')
        except Exception as e:
            I0.output_error_messages("Non-specific error occurred.")
            I0.output_error_messages("Technical error message:")
            print(e, e.__doc__, type(e), sep='\n')
        finally:
            if file and not file.closed:
                file.close()

    1 usage
    @staticmethod
    def write_data_to_file(file_name: str, student_data: list):
        """ Writes data from a list of dictionary rows to a JSON file """
        file = None
        try:
            file = open(file_name, "w")
            json.dump(student_data, file, indent=4)
        except Exception as e:
            I0.output_error_messages("An error occurred while saving data to the file.")
            I0.output_error_messages("Technical error message:")
            print(e, e.__doc__, type(e), sep='\n')
        finally:
            if file and not file.closed:
                file.close()
```

Figure 1. Screen shot of FileProcessor class logic PyCharm

Topic – Input and Output

Input and output operations are managed by the IO class, which handles all interactions with the user including error messages. This class is responsible for receiving input from the user, displaying menu options, and showing the current list of registered students. It also calls on the @staticmethod like the FileProcessor class. See Figure 2.

13 usages

```
class IO:
```

```
    """Handles Input and Output operations"""
```

9 usages

```
    @staticmethod
```

```
    def output_error_messages(message: str, error: Exception = None):
```

```
        """ Outputs error messages to the user """
```

```
        print(f"Error: {message}")
```

```
        if error:
```

```
            print(f"Exception: {error}")
```

Figure 2. Screen shot of code to create class IO

Key Components:

- **Input Handling:** The input_student_data method prompts the user to enter a student's first name, last name, and course name. The method includes input validation to ensure that only alphabetic characters are accepted for the names.
- **Output Handling:** The output_student_courses method displays the list of registered students. The output_menu method prints the main menu to guide the user's choices.

Separation of Concerns:

By encapsulating all input and output operations within the IO class, the program maintains a clear separation between user interaction and other program sections. This makes the code more organized and easier to understand, allowing changes

to be made to input/output processes without affecting other parts of the program.

Topic – Error Handling

Error handling is an important part of the program, ensuring that it can handle unexpected situations, such as file access issues or invalid user input. Structured error handling is implemented using try-except-finally blocks, with specific messaging for different types of errors.

Key Components:

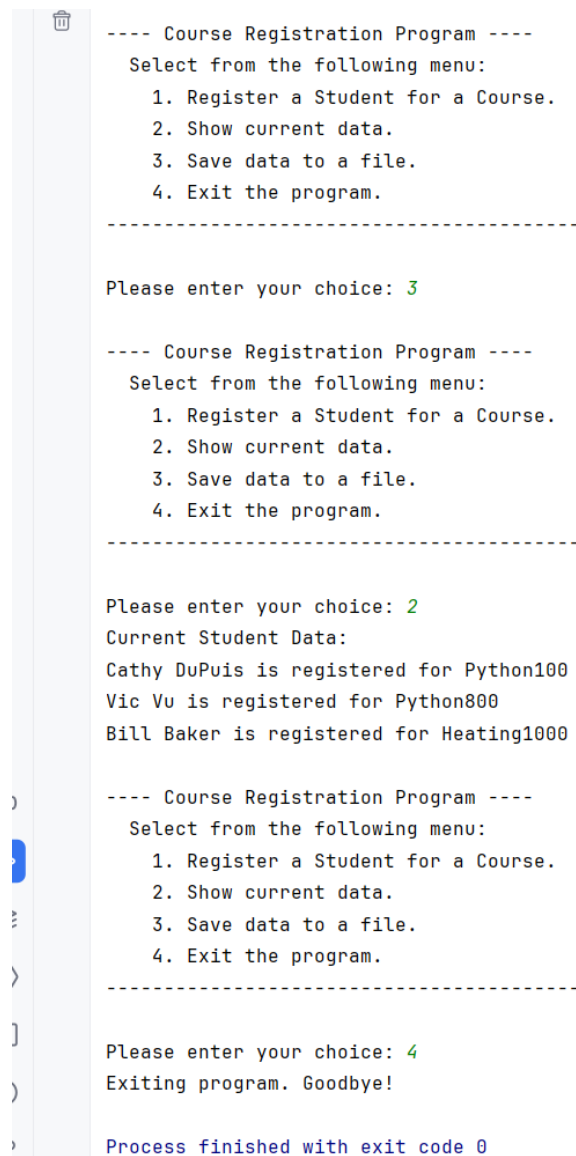
- **File Not Found Error:** If the JSON file does not exist when attempting to read data, the program catches the `FileNotFoundException` and informs the user that the file must exist before running the script. It also provides a technical error message for further troubleshooting. (see Figure 1)
- **General Exceptions:** The program includes a general Exception handler to catch any non-specific errors that may occur. This handler also outputs a detailed technical error message, which includes the exception, its documentation string, and its type.
- **Closing Resources:** The program ensures that the file is always closed after reading or writing operations, even if an error occurs. This is managed within the finally block, which guarantees that resources are properly released.

Separation of Concerns:

Error handling is written into the processing and input/output operations but is managed in a way that keeps these concerns separate. Each function handles its own errors, making it easier to isolate and manage different aspects of the program's behavior.

Topic – Testing

After inputting student registration data, the program then lists the first and last name and course name. (See Figure 3)



```
---- 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.
-----

Please enter your choice: 3

---- 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.
-----

Please enter your choice: 2
Current Student Data:
Cathy DuPuis is registered for Python100
Vic Vu is registered for Python800
Bill Baker is registered for Heating1000

---- 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.
-----

Please enter your choice: 4
Exiting program. Goodbye!

Process finished with exit code 0
```

Figure 3. Student and course details displayed with successful program exit.

Using menu choice 3, the data gets saved to Enrollements.json and can be opened in a text editor like notepad to show the dictionary list rows.

```
[
  {
    "First Name": "Cathy",
    "Last Name": "DuPuis",
    "Course": "Python100"
  },
  {
    "First Name": "Vic",
    "Last Name": "Vu",
    "Course": "Python800"
  },
  {
    "First Name": "Bill",
    "Last Name": "Baker",
    "Course": "Heating1000"
  }
]
```

Figure 4. Text file of json student data (Figure 4).

Per the assignment, I also tested in the MS Command Prompt and the program ran successfully in that environment (Figure 5). I verified that the program can register students, display current data, save data to a file, and handle multiple registrations and that it can display the dictionary data in Notepad with the key and individual string values. I forgot to have the program say the data was successfully entered for Menu Choice 3, but it is working nonetheless without that UX add-on.

```
-----  
Please enter your choice: 1  
Enter the student's first name: Maltida  
Enter the student's last name: Lingonberry  
Enter the course name: BotanyinDenali  
Student Maltida Lingonberry has been registered for the course 'BotanyinDenali'.  
      ^
```

```
---- 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.  
-----
```

```
Please enter your choice: 3
```

```
---- 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.  
-----
```

```
Please enter your choice: 4  
Exiting program. Goodbye!
```

```
Please enter your choice: 2  
Current Student Data:  
Cathy DuPuis is registered for Python100  
Vic Vu is registered for Python800  
Bill Baker is registered for Heating1000  
Maltida Lingonberry is registered for BotanyinDenali
```

```
---- 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.  
-----
```

```
Please enter your choice: 4  
Exiting program. Goodbye!
```

Figure 5. Command prompt working screen shots showing successful entry

Summary

This assignment highlights the importance of structuring code using the principles of Separation of Concerns. By dividing the program into distinct classes that handle processing, input/output, and error handling, the code becomes more modular, easier to maintain, and in theory for advanced users who look at the program and need to modify it, less prone to errors. The structured approach to error handling further ensures that the program can handle unexpected situations without abruptly breaking.

Github posting: <https://github.com/CathyD-UW/IntrotoProg-Python-Mod06>