

Michael Kemery

2024.08.06

Foundations of Programming, Python

Assignment_06

[branchingTacos/IntroToProg-Python-Mod06 \(github.com\)](https://github.com/branchingTacos/IntroToProg-Python-Mod06)

Assignment 06 – Classes and Functions

Introduction

This week, we concentrated on enhancing the course registration program without significant new functionality. We particularly aimed to streamline the course registration program, focusing on refactoring the codebase to improve its structure and maintainability. The previous iteration of the program, while functional, lacked modularity and was difficult to manage as it grew in complexity. By introducing functions and classes, we aimed to create a more organized and scalable solution that aligns with best practices in software design.

Main Program

Overall, I divided the work into three main tasks (with testing occurring throughout):

1. Review the initial script and the requirements document
2. Outline and define the different classes and functions that we needed to implement in the program
3. Consider other modifications and or quality of life changes that could help the user experience when running the program.

This week we needed to take last week's program and refactor it using classes and functions. In the end we created two classes: `FileProcessor` and `IO`. Within the `FileProcessor` class we only had a single function: `read_data_from_file()`. The intent of this function was to read in the data file and load the contents into the local variable: `student_data`. The `IO` class is composed of 7 functions:

1. `output_error_messages(message: str, error: Exception = None)` – Displays custom error messages to the user.
2. `output_menu(menu: str)` – Displays the menu choices to the user.
3. `input_menu_choice()` – Gets a menu choice from the user.
4. `input_student_data(student_data: list)` – Prompts the user to enter student data.
5. `write_data_to_file(file_name: str, student_data: list)` – Writes the student data to a file.

6. `output_student_courses(student_data: list)` – Outputs the student information, with an option to sort by course name (sort option discussed in ‘Modifications to the Program section’)
7. `sort_output(data)` - Sorts the output by course name (discussed in ‘Modifications to the Program section’)

Ultimately, refactoring the program this way does not change the underlying behavior of the program, rather it helps make the code organizationally easier to understand, making the code easier to maintain/update. It can also make functions more portable as they are encapsulated into a class that could be used in other programs.

As one can see from Code Snippet 1 – the overall code has been dramatically simplified. When a user selects option 1 to register a student, we call the `input_student_data()` function from the IO class and then print the data with the `output_student_courses()` function from the same class.

Code Snippet 1

```
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":
    print() # for easier reading between prompt and output
    IO.output_student_courses(student_data=students)
    continue

# Save the data to a file
elif menu_choice == "3":
    IO.write_data_to_file(FILE_NAME, students)

# Stop the loop
elif menu_choice == "4":
    break
```

Confusion may be created by using the same variable names internal to and external to the functions. The variables are scoped appropriately and the computer does not get confused, however having several versions of “`student_data`” could make it confusing for someone reviewing the code itself.

Since the underlying behavior of the program didn’t change too much since last week, I focused primarily on the organization of the program rather than its behavior.

Modifications to the program:

While I wasn’t intending to extend the program beyond the requirements and some of the previous functionality, during testing and reviewing the output, I considered sorting the data by `course_name`. This consideration was likely influenced by my professional experience as a researcher, where I regularly analyze market-level and user-level data from various perspectives.

To clarify, I did not modify the data written to the file; instead, I wanted to view the output from another perspective. Therefore, I included an additional function, `sort_output()` (Code Snippet 2), and modified the `output_student_courses()` function.

To achieve this, I used the sorted function in Python, which was guided by a lambda function. The lambda function ([Python Lambda \(w3schools.com\)](https://www.w3schools.com/python/python_lambda.asp)) is essentially an anonymous function that performs a singular task or expression, though can take multiple arguments. In this scenario, the lambda function works as the sort key.

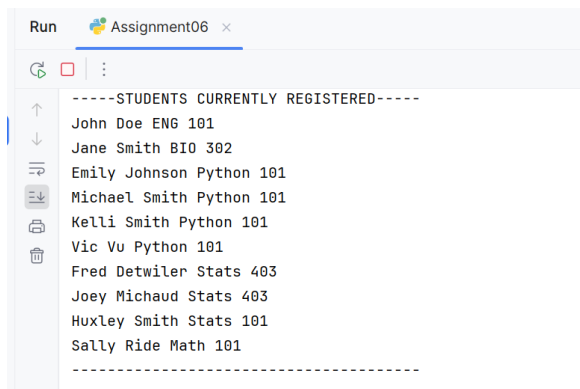
Code Snippet 2

```
def sort_output(data):  
    sort_data = sorted(data, key=lambda x: (x['CourseName'], x['FirstName']))  
    #print(sort_data["FirstName"], sort_data["LastName"], sort_data["CourseName"])  
    return sort_data
```

To view the modified output, I first ask the user if they'd like to see the data sorted by course name and included conditional logic to sort. Then the `output_student_courses()` function will call the `sort_output()` function, if the user indicates that they would like to see the data sorted. Image 1 shows the differences between the two different output while Code Snippet 3 shows how this function works.

Image 1 – difference between output (default and sorted by course_name)

Default version of output:



```
Run Assignment06 x  
-----STUDENTS CURRENTLY REGISTERED-----  
John Doe ENG 101  
Jane Smith BIO 302  
Emily Johnson Python 101  
Michael Smith Python 101  
Kelli Smith Python 101  
Vic Vu Python 101  
Fred Detwiler Stats 403  
Joey Michaud Stats 403  
Huxley Smith Stats 101  
Sally Ride Math 101  
-----
```

Sorted version of output:

```
Would you like to view the registered students sorted by course name? (Y/N)y

-----STUDENTS CURRENTLY REGISTERED-----
BIO 302
Jane Smith
-----
ENG 101
John Doe
-----
Python 101
Emily Johnson
Kelli Smith
Michael Smith
Vic Vu
-----
Stats 101
Huxley Smith
-----
Stats 403
Fred Detwiler
Joey Michaud
-----
```

Code Snippet 3

```
def output_student_courses(stu_data: list):
    choice = input("Would you like to view the registered students sorted by course
name? (Y/N)")
    print()
    # define header for output
    my_message = ("-" * 5 + "STUDENTS CURRENTLY REGISTERED" + "-" * 5)
    if choice in ("Y", "y", "N", "n"):

        if choice in ("Y", "y"):
            sort_data = IO.sort_output(stu_data)
            course_dict = {} ## creating a new dictionary - locally scoped
            for student in sort_data:
                course_name = student["CourseName"]
                if course_name not in course_dict:
                    course_dict[course_name] = []
                course_dict[course_name].append(f"{student['FirstName']}
{student['LastName']}")

            print(my_message)
            for course_name, students in course_dict.items():
                print(course_name)
                for student in students:
                    print(student)
                print("-" * len(my_message))
        else:
            # keep original order / don't sort
            sort_data = stu_data
            my_message = ("-" * 5 + "STUDENTS CURRENTLY REGISTERED" + "-" * 5)
            print(my_message)
            for student in sort_data:
                print(student["FirstName"], student["LastName"],
student["CourseName"])
                print("-" * len(my_message))
    else:
        print("Invalid choice. Please enter 'Y' or 'N'.")
```

While the use of the `sort_output()` function to sort data by `course_name` is a decent first step, it is essential to consider the separation of concerns within this codebase for better clarity and maintainability. The `output_student_courses()` function should ideally be split into two separate functions:

1. **Default Output Function:** outputs the data in the order it was inputted and written to the file.
2. **Sorted Output Function:** handles the sorting of data by `course_name` before outputting it.

By isolating these different behaviors into distinct functions, it becomes much easier for someone to understand what each function does when reviewing the code. This separation also adheres to the principles discussed this week, where each function has a single responsibility, making the codebase more modular and easier to maintain.

Summary

This week we refactored the course registration program to improve its structure and maintainability by introducing functions and classes and adhering to the separation of concerns. With this focus the codebase is better organized and modular, facilitating easier updates and enhancements.