Michael Kemery

2024.07.29

Foundations of Programming, Python

Assignment_05

Assignment 05 – Working with Files and Lists

Introduction

This week we focused on continuing to advance the course registration program that we've been working on for the last couple of weeks. Our goal was to refine the program by incorporating additional features and improvements. The objectives this week were to include data handling of a JSON file, incorporate error handling, as well as working with different data structures (e.g., dictionaries, lists, tuples). Although we focused on these improvements, the overt behavior of the program itself did not change too much for the user.

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

1. Make the required modifications to ensure the program behaves as outlined in the requirements document
2. Add the error handling calls
3. Any other modifications and or quality of life changes

The first step was to import the JSON library and modify the import call to read the JSON file into the `students` object. This object is typed as a list but rather than using it as a 1-dimensional list, we loaded data into it and used it as a 2-dimensional list (or more colloquially a 'table'). It's also important to ensure that the keys in the dictionary match the field names in the JSON file, as discrepancies can lead to issues.

Now, when a user selects option 1 to register a student, the goal is to append this new data to the `students` object. However, before that takes place, we reserve this data into a dictionary (see Code Snippet 1). It's important to ensure that the keys in the dictionary match the field names in the JSON file, as discrepancies can lead to issues (see Appendix 1 for an example of how having different object names can create issues).

Code Snippet 1:

```
student_first_name = input("Enter the student's first name: ")
student_last_name = input("Enter the student's last name: ")
course_name = input("Please enter the name of the course: ")

student_data = {"FirstName": student_first_name,
                "LastName": student_last_name,
                "CourseName": course_name}
students.append(student_data)
```

As with last week, when the user selects option 2, we want to print to console both the data just collected and the data in RAM. However, since we are using a dictionary, accessing the data is slightly different than last week: a dictionary is accessed using a key, while a list is accessed with an index and does not contain keys. Next, when the user selects option 3, all data is saved to a file, and if option 4 is selected, the user exits the program.

Modifications to the program:

This week, I wanted to modify the program to search if a student was already enrolled in a specific class. To do this, I created a Boolean variable called `found` (`found: bool`). As the program iterates through the list of students (`for student in students{}`), it checks for an identical match across all three variables (see code snippet 2). If an identical match was found, the program prints a message telling the user the student already is enrolled. If the student was not already enrolled, the student's data is appended to the `students` object.

Code Snippet 2

```
found = False
for student in students:
    if (student["FirstName"] == student_first_name and
        student["LastName"] == student_last_name and
        student["CourseName"] == course_name):
        found = True
        break

if found:
    print(f"{student_first_name} {student_last_name} is already enrolled in
{course_name}.")
else:
    student_data = {"FirstName": student_first_name,
                    "LastName": student_last_name,
                    "CourseName": course_name}
    #  student_data = [student_first_name,student_last_name,course_name]
    students.append(student_data)
    print(f"You have registered {student_first_name} {student_last_name} for
{course_name}.")
```

This is a good first step, but it’s possible for two students to have the same first and last name and be enrolled in the same course. To address this potential issue, ideally, each student would have an assigned ID that we could include in the registration process. However, if students do not have a pre-assigned unique identifier, I would assign each student an ID number and ask the user to confirm if they want to include a duplicate entry. I did not pursue this modification for this assignment, but if I were to implement it, I would likely create it as a separate function. This approach would make the duplicate-detection code easier to modify in the future without affecting the entire program. It would also make the code more portable to future programs that might face similar data challenges.

Summary

This week, we made significant progress in refining the course registration program we’ve been developing. We focused on enhancing data handling by integrating JSON file support, implementing robust error handling, and working with different data structures.