Rebecca Bergh
7/28/2024
Foundations of Programming: Python
Assignment 05
https://github.com/rlbergh/IntroToProg-Python-Mod05

Assignment 05

Introduction

The starter file was tricky this week. It was set up with some initial code that was intended for use with a csv file. And the starter .json file had some inconsistencies that led to some errors. It was good to be able to diagnose those errors relatively easily and be able to fix them.

Rearranging the file

When I first opened the file, I decided to rearrange the variables to match the assignment instructions and also to put all of the original code in a ''' comment ''' so that I could reference it as I was making my own code, but not get confused about what was old and what was new. This helped me set aside the original code without deleting it, and forced me to write out all of my original code instead of just modifying the existing lines. Ultimately I did end of deleting the original code when I was finished, but it helped me gain perspective as I moved through the required functions of the program.

Using the 'try' function

Last week I ran into an issue when I tested my program without the csv file in the same directory folder. This week, I decided to try to fix that by using the try statement. I needed to implement this first before I got started on the rest of the program because once I was writing data to the "Enrollments.json" file, it would have been more difficult to test the "try" part.

Ultimately, I wanted to attach a Boolean variable to whether the "Enrollments.json" file existed. I saw some references to a separate module you can important to be able to do this, but I thought I could try just using the "try" function instead and depending on the type of error, set my variable value. To my surprise, it worked well the first time. (Stack Overflow, https://stackoverflow.com/questions/20652527/python-try-except-with-of-if-else, 2024) (External Site)

Using the try function, I opened the json file in "read" mode. If it didn't exist, then an error would be thrown. After "trying" to open the file and close it, I wrote an "except" line looking for the "File Not Found" error. If it found that *error*, then I would change my Boolean variable "file_exists" to "False." If it didn't find that error, then it would move on to reading the file into the program, storing the existing data. My variable originally had a value of "True" in order to make this work properly.

If the error was found — and the Boolean variable was equal to "False — it would then open the file in "write" mode, creating the file since it doesn't exists. It would also print the line "Setting up enrollment file…" to the end user.

Doing this "try" function allowed me to test whether the file existed first before overwriting any existing data, and let the end user know a new file was being created.

```
# test whether the json file exists
try:
    file = open(FILE_NAME,'r')
    file.close()
# if it doesn't, set variable to false
except FileNotFoundError:
    file_exists = False
# if the file exists, read it and store the existing data in students list
if file_exists:
    file = open(FILE_NAME,'r')
    students = json.load(file)
    file.close()
    for student_data in students:
        print(f'{student_data["FirstName"]} {student_data["LastName"]} is registered
for {student_data["CourseName"]}.')
#if the file doesn't exist, create it and close it
else:
    print('Setting up enrollment file...')
    file = open(FILE_NAME,'w')
    file.close()
```

One big miss

However, one major flaw that I hadn't anticipated until I more thoroughly tested this code was that if the file existed, but it was blank, it would give me an error. That's because my logic said that if the file existed, then open it in "read" mode and load the data into the program. But if there is no data, then it will give an error.

To fix this, I set out to create a nested if/else statement. My next logical check would be whether there was a first character of the json file or not. (Geeks For Geeks, https://www.geeksforgeeks.org/check-if-a-text-file-empty-in-python/, 2024) (External site)

I checked if the first character (first_char) was blank, and if it was, then it would simply pass onto the next thing in the program. If the file wasn't blank, then it would load the data into the program. This doesn't cover, however, if the file is not in the right format with the same keys as my program. So, while I think this was a good learning experience, it isn't the best solution. I think opening the file in append mode would be far more effective. See code on next page.

```
try:
    file = open(FILE_NAME,'r')
    file.close()
# if it doesn't, set variable to false
except FileNotFoundError:
    file_exists = False
# if the file exists, read it
if file_exists:
    # if the file exists, see if it's empty or not
    file = open(FILE_NAME,'r')
    # read first character
    first_char = file.read(1)
    # if it's empty, do nothing
    if not first_char:
        pass
    else:
        file = open(FILE_NAME, 'r')
        students = json.load(file)
        file.close()
        for student_data in students:
            print(f'{student_data["FirstName"]} {student_data["LastName"]\
                } is registered for {student_data["CourseName"]}.')

# if the file doesn't exist, create it and close it
else:
    print('Setting up enrollment file...')
    file = open(FILE_NAME,'w')
    file.close()
```

Error handling

I actually enjoy doing my own error handling and I think it's because of my background in journalism. I was pretty good at anticipating questions from readers, which helped me in interviews. The same is true for error handling: you have to anticipate what error-prone humans might do with inputs, file integrity, etc.

Adding in error handling for typos adds a layer of integrity and security to a program. In this case, I did a lot of error handling at the beginning with my try statements and various extra variables for the file existing and whether the file was blank or not. With the inputs, there was more nesting necessary. But this time it was nesting "try" functions instead of "if" functions.

If the end user types a number in with the first name, then it throws an error and also asks the end user to try again. Once it passes that test, the same try statement occurs for the last name. But for those to happen in the right order and for it to give the end user another opportunity to enter either name, I needed to nest the function together (Figure 3).

```
76      # Get data inputs and show result
77      if menu_choice == "1":
78          try:
79              student_first_name = input("Enter the student's first name: ")
80              if not student_first_name.isalpha():
81                  raise ValueError("First name can only have alphabetic characters")
82          except ValueError as e:
83              print(e)
84              student_first_name = input("Try entering the student's first name again: ")
85          finally:
86              try:
87                  student_last_name = input("Enter the student's last name: ")
88                  if not student_last_name.isalpha():
89                      raise ValueError("Last name can only have alphabetic characters")
90              except ValueError as e:
91                  print(e)
92                  student_last_name = input("Try entering the student's last name again: ")
93              finally:
94                  course_name = input("Enter the course name: ")
95              student_data = {"FirstName": student_first_name, "LastName": student_last_name,"CourseName": course_name}
96              students.append(student_data)
97              # print result
98      print(f'You have registered {student_first_name} {student_last_name} for {course_name}.')
99      continue
```

Figure 1: Line 78 starts my "try" statement, but the "finally" section starts a new "try" statement for the last name input.

Breaking lines in f strings

One, seemingly small, issue I encountered while writing a longer f string — using keys from the dictionary made is longer — was where to put the line break. I kept getting a large space between my student's full name and the rest of the print message when I tested the program. I realized I needed to put my line break somewhere inside the curly braces or the brackets for it not to do that. Figure 2 shows the code and result before, and Figure 3 shows the code and result after the fix.

```
72
73      elif menu_choice == "2":
74          for student_data in students:
75              print(f'{student_data["FirstName"]} {student_data["LastName"]}\
76               is registered for {student_data["CourseName"]}.')
77
```

```
What would you like to do? 2
Bob Smith             is registered for Python 100.
Sue Jones             is registered for Python 100.
```

Figure 2: Line 75 has a line break after the curly braces, which inserted a bunch of spaces (or tabs) into my print message (bottom section of this figure).

```
72
73      elif menu_choice == "2":
74          for student_data in students:
75              print(f'{student_data["FirstName"]} {student_data["LastName"]\
76                  } is registered for {student_data["CourseName"]}.')
77
```

```
What would you like to do? 2
Bob Smith is registered for Python 100.
Sue Jones is registered for Python 100.
```

Figure 3: With the line break now inside the curly braces, there is no longer a huge space between the student's name and the rest of the string.

Summary

After testing my code in various scenarios — when no json file was present in the directory, when a blank json file was present, and when a not-blank json file was present — I reflected on how I was able to solve a vexing conundrum using some simple logic and short lines of code. I think there are modules that help in situations like this, but I really wanted to rely on the "vanilla" version of Python instead of turning to a module at the first sight of trouble. I'm glad I was able to figure it out with some gentle help from the internet.