Keegan McGuire

8/6/2025

Foundations of Python Programming

Mod05 - Assignment05

https://github.com/Keegan-McGuire/IntroToProg-Python-Mod05

Error Handling, JSON, and GitHub

Integrating JSON files and uploading to a GitHub repository

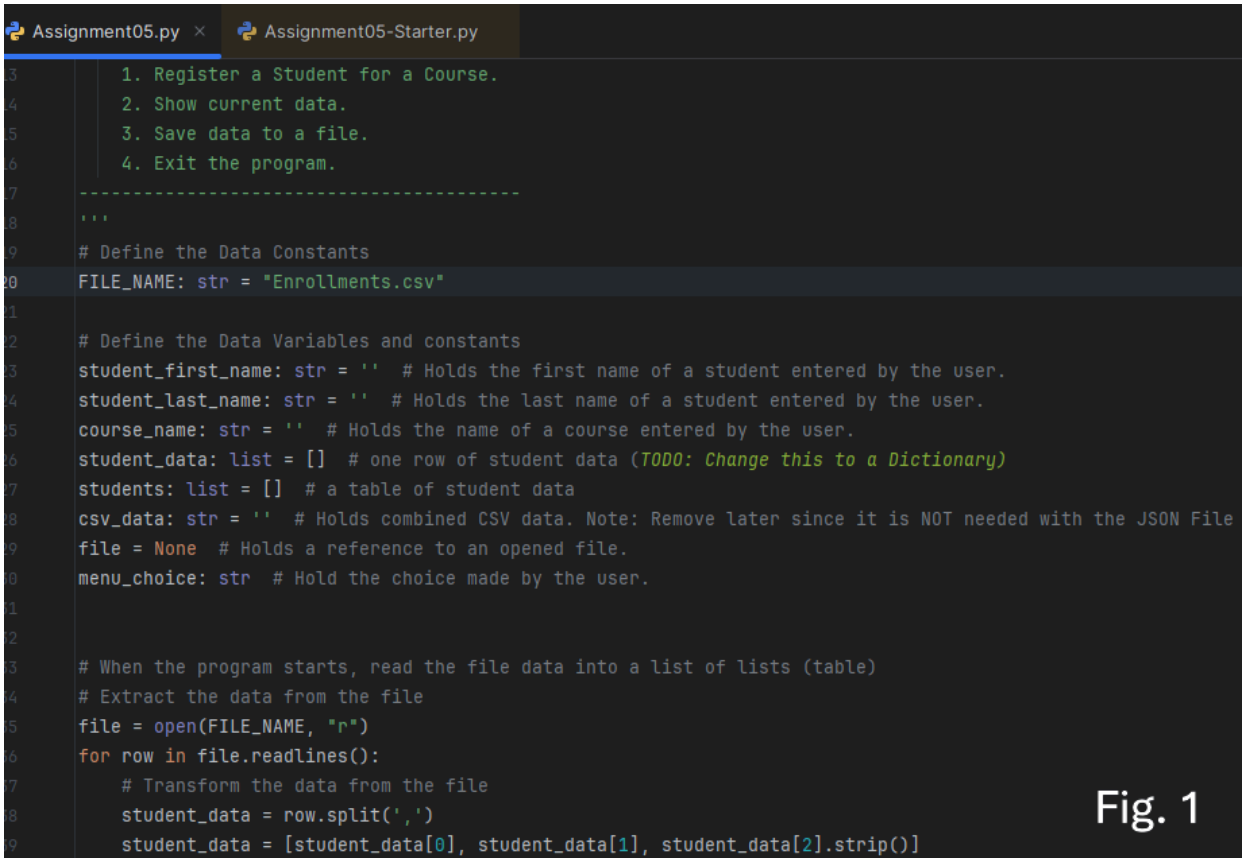
This week, the goal for our ever improving file registration program was to upgrade its functionality to include a module – specifically, the JSON module – and integrate error handling to give users easily parsed feedback when things go wrong. In addition, we would be uploading these files to be shared in a repository on GitHub, allowing our peers to quickly review and comment on our solutions.

For me personally, this process would ultimately go significantly smoother than last week's effort, although there were still plenty of interesting hurdles to overcome.

A Commitment to Upgrades

One important change to how I handled this week's assignment was to ensure that I followed the instructions Professor Root gave in the Wednesday Zoom session, where he explicitly said that the goal here was to upgrade existing code, as opposed to writing all new code.

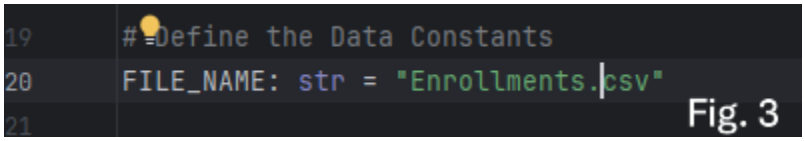
While that was a simple enough task, it ran counter to the panic rebuild I had used last week after my attempt at upgrading the starter code quickly ran into problems. This week, I tried my hardest to move slowly in upgrading the existing starter code and test every step along the way. To start, I moved over the starter code from the Assignment-05Start.py file (Fig. 1) and got started.



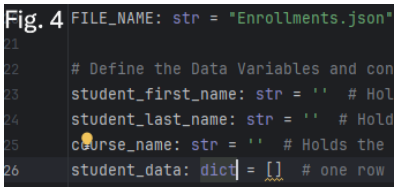
From there, I wanted to move step by step through the acceptance criteria, so I started with making sure my constants were updated. This meant changing simple items like file type we would use to store our data (Fig. 2, Fig. 3).

- The constant **FILE_NAME: str** is set to the value "Enrollments.json"
- The constant values do not change throughout the program.

Fig. 2



It also meant going through the variables and changing the values that needed to reflect the changes we would make along the way, such as the **student_data: dict** value. These changes were also indicated by the instructions in the commented out notes provided by the starter file (Fig. 4, Fig 5). I also decided to add the code that would import JSON (Fig. 6) at this point.



```
student_data: list = []  # one row of student data (TODO: Change this to a Dictionary)
students: list = []  # a table of student data
csv_data: str = ''  # Holds combined CSV data. Note: Remove later since it is NOT needed with the JSON File
```
Fig .5

```
import json
```
Fig. 6

Back on the Menu

With those simple steps out of the way, I decided to continue following the assignment criteria linearly. The next step was to start working on the input options provided by Menu item 1, so I started there.

We already had the structure from last week (Fig. 7), but I knew I had seen an updated format in one of our labs from this week (Fig. 8) that would cooperate with the dictionary JSON format needed. To that end, I moved what I found in the examples over to my program (Fig. 9).

```
# Input user data
if menu_choice == "1":  # This will not work if it is an integer!
    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 = [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}.")
    continue
```
Fig. 7

```
file = open(FILE_NAME, "r")
for row in file.readlines():
    # Transform the data from the file
    student_data = row.split(',')
    student_data = {"FirstName": student_data[0],
                    "LastName": student_data[1],
                    "GPA": float(student_data[2].strip())}
    # Load it into our collection (list of lists)
    students.append(student_data)
file.close()
```
Fig. 8

```
        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)
```
Fig. 9

I wanted to test my code, but quickly realized that there wasn't much I could do without the correct files in place, so I deviated from my plan to work from bullet point to bullet point to add the required JSON file from the assignment module (Fig. 10) to my assignment folder (Fig. 11).

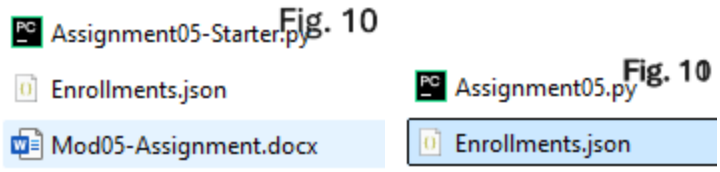


Fig. 10

Fig. 10

With those in place, I also opted to replace the section that was formerly reading the .csv file with one that would open the .json file, which I also found an example of in the text (Fig. 12). I also opted to comment out the old code in the spirit of keeping in line with the example and the upgrade – I'm not sure how this would be handled in a real upgrade, it seems cleaner to remove the code altogether, but I'm sure our peer review will help cover that.

5. **Locate and comment out** the code that opens and reads the csv data into the table.

Fig. 12

```
# file = open(FILE_NAME, "r")
# for row in file.readlines():
#     # Transform the data from the file
#     student_data = row.split(',')
#     student_data = {"FirstName": student_data[0],
#                     "LastName": student_data[1],
#                     "GPA": float(student_data[2].strip())}
#     # Load it into our collection (list of lists)
#     students.append(student_data)
# file.close()
```

6. **Add** code that opens the json file, dumps the data into the students table variable and closes the file again.

```
file = open(FILE_NAME, "r")
students = json.load(file)
file.close()
```

Upgrading the rest of the Menu

With that in place, it was time to try and upgrade the basic functionality of the remaining menu items and finish the input/output section of the assignment criteria. I loaded up the existing framework for Menu item 2 (Fig. 13), and went to look for a way to upgrade that to cooperate with our new dictionary framework (Fig. 14).

Fig. 13

```
73      elif menu_choice == "2":
74
75          # Process the data to create and display a custom message
76          print("-"*50)
77          for student in students:
78              print(f"Student {student[0]} {student[1]} is enrolled in {student[2]}")
79          print("-"*50)
80          continue
```

To read the data created by the code you provided, which writes a list of dictionaries (table) to a JSON file named "data.json," you can use the following code to read and process that JSON data:

```
import json
# Open the JSON file for reading
file = open("data.json", "r")
data = json.load(file)
# Now 'data' contains the parsed JSON data as a Python list of dictionaries
for item in data:
    print(f"ID: {item['ID']}, Name: {item['Name']}, Email: {item['Email']}")
```

Fig. 14

Following this example, I upgraded the format to reference the keys in the JSON file, FirstName, LastName, and CourseName (Fig. 15). These changes seemed to be successful (Fig. 16), the program was running as expected with the two new menu items – although running it through Menu item 3 (Fig. 17) without making modifications would lead to some unexpected challenges.

```
75          # Process the data to create and display a custom message
76          print("-"*50)
77          for student in students:
78              print(f"Student {student["FirstName"]} {student["LastName"]} is enrolled in {student["CourseName"]}")
79          print("-"*50)
80          continue
```

Fig. 15

```
What would you like to do: 1
Enter the student's first name: Rory
Enter the student's last name: Gamble
Please enter the name of the course: Python100
You have registered Rory Gamble for Python100.

---- 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.
----------------------------------------

What would you like to do: 2
--------------------------------------------------
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Rory Gamble is enrolled in Python100
```

Fig. 16

```
What would you like to do: 3
Traceback (most recent call last):
  File "C:\Users\keega\Desktop\UW PCE\Foundations of Progr
    csv_data = f"{student[0]},{student[1]},{student[2]}\n"
                ~~~~~~~~^^^
KeyError: 0

Process finished with exit code 1
```

Fig. 17

It would be a minute before I encountered the issue this created, I decided to move on to the processing section of the assignment criteria. I had already set my program to read enrollment.json, so it was time to tackle Menu item 3. I went through our labs and notes to find an instance of code that looked like it would accomplish what I needed (Fig. 18), and then applying it to our code with the correct values (Fig. 19).

```
# Write the contents to a file for each row in the table
file = open("data.json", 'w')
json.dump(table, file)
file.close()
```

Fig. 18

```
82      # Save the data to a file
83      elif menu_choice == "3":
84          file = open(FILE_NAME, "w")
85          json.dump(students, file)
86          file.close()
87          print("The following data was saved to file!")
88          for student in students:
89              print(f"Student {student["FirstName"]} {student["LastName"]} is enrolled in {student["CourseName"]}")
90          continue
```

Fig. 19

However, it was at this stage that I would discover the consequences of running my program through steps that had not been updated when I received an unexpected error (Fig. 20). As I began to investigate this, it soon became clear that I had accidentally wiped the data from my .json file by running the old code from Menu item 3 (Fig. 21).

```
  File "C:\Users\keega\Desktop\UW PCE\Foundations of Programming Python\_Module05\IntroToProg\Assignment05.py", line 49, in <module>
    students = json.load(file)
  File "C:\Users\keega\AppData\Local\Programs\Python\Python313\Lib\json\__init__.py", line 293, in load
    return loads(fp.read(),
        cls=cls, object_hook=object_hook,
        parse_float=parse_float, parse_int=parse_int,
        parse_constant=parse_constant, object_pairs_hook=object_pairs_hook, **kw)
  File "C:\Users\keega\AppData\Local\Programs\Python\Python313\Lib\json\__init__.py", line 346, in loads
    return _default_decoder.decode(s)
           ~~~~~~~~~~~~~~~~~~~~~~~~^^^
  File "C:\Users\keega\AppData\Local\Programs\Python\Python313\Lib\json\decoder.py", line 345, in decode
    obj, end = self.raw_decode(s, idx=_w(s, 0).end())
               ~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\keega\AppData\Local\Programs\Python\Python313\Lib\json\decoder.py", line 363, in raw_decode
    raise JSONDecodeError("Expecting value", s, err.value) from None
json.decoder.JSONDecodeError: Expecting value: line 1 column 1 (char 0)
```

Fig. 20

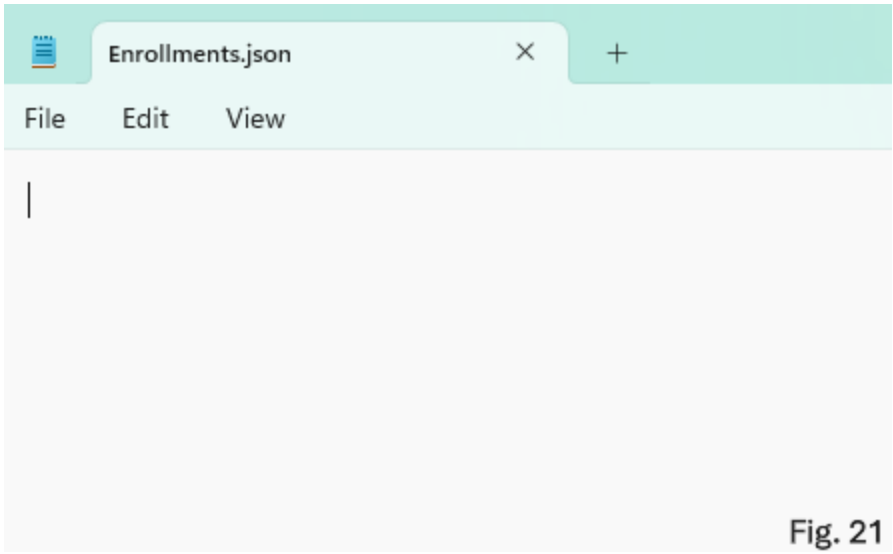


Fig. 21

To fix this, I repopulated my json file using the starter data from the original enrollments file. Thankfully, that worked – it was even able to save my new data and update the file (Fig. 22).

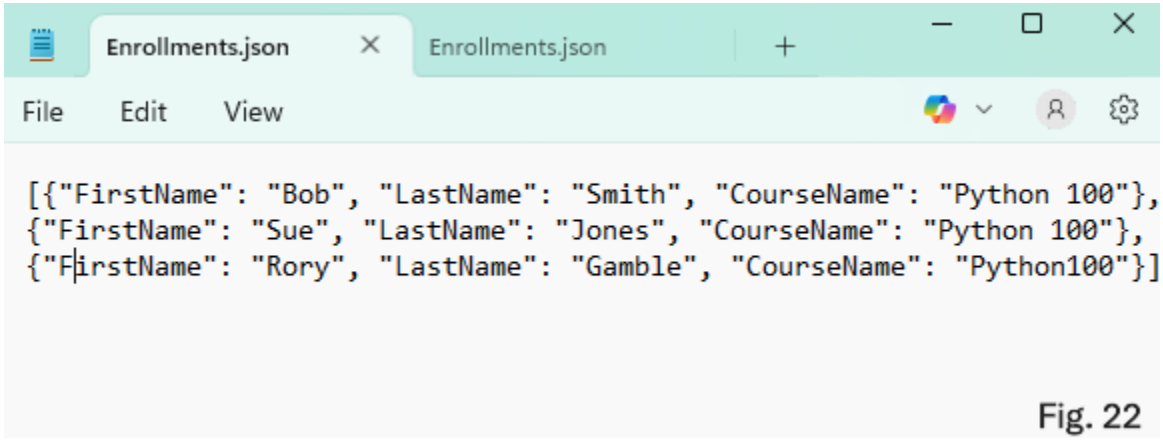


Fig. 22

With all that in place, and Menu item 4 remaining largely the same, it was time to move on to error handling.

Adding Structured Error Handling

With all my menu items seemingly working with basic commands, it was time to add some error messages for basic potential user errors. I started with the code that would read my JSON file, which currently looked very simple (Fig. 23). I found an example from our text that would introduce the error handling needed (Fig. 24), and updated my program accordingly (Fig. 25).

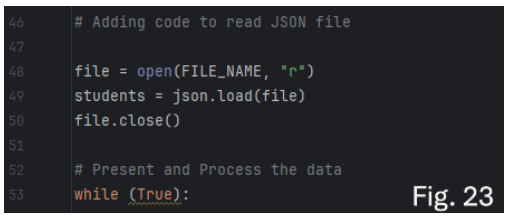


Fig. 23

4 **Add** a **try** statement **before** that code, indent the code, **then add** the following exception handling **after** the code.

```
try:

    file = open(FILE_NAME, "r")
    students = json.load(file)
    file.close()

except FileNotFoundError as e:
    print("Text file must exist before running this script!\n")
    print("-- Technical Error Message -- ")
    print(e, e.__doc__, type(e), sep='\n')
except Exception as e:
    print("There was a non-specific error!\n")
    print("-- Technical Error Message -- ")
    print(e, e.__doc__, type(e), sep='\n')
finally:
    if file.closed == False:
        file.close()
```

Fig. 24

```
49      try:
50          file = open(FILE_NAME, "r")
51          students = json.load(file)
52          file.close()
53
54      except FileNotFoundError as e:
55          print("Text file must exist before running this program!\n")
56          print("-- Technical Error Message -- ")
57          print(e, e.__doc__, type(e), sep='\n')
58      except Exception as e:
59          print("There was a non-specific error!\n")
60          print("-- Technical Error Message -- ")
61          print(e, e.__doc__, type(e), sep='\n')
62      finally:
63          if file.closed == False:
64              file.close()
```

Fig. 25

Next I updated my first Menu item (Fig. 26, 27) following an example I found in our notes.

10. **Add** a **try** statement **before** that code, indent the code, **then add** the following validation and exception handling **after** the code.

```
try:
    # Input the data
    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.")

    student_last_name = input("What is the student's last name? ")
    if not student_last_name.isalpha():
        raise ValueError("The last name should not contain numbers.")

    try:  # using a nested try block to capture when an input cannot be changed to a float
        student_gpa = float(input("What is the student's GPA? "))
    except ValueError:
        raise ValueError("GPA must be a numeric value.")

    student_data = {"FirstName": student_first_name,
                    "LastName": student_last_name,
                    "GPA": float(student_gpa)}
    students.append(student_data)
except ValueError as e:
    print(e)  # Prints the custom message
    print("-- Technical Error Message -- ")
    print(e.__doc__)
    print(e.__str__())
except Exception as e:
    print("There was a non-specific error!\n")
    print("-- Technical Error Message -- ")
    print(e, e.__doc__, type(e), sep='\n')
```

Fig. 26

```
76          # Adding structured error handling
77          try:
78              student_first_name = input("Enter the student's first name: ")
79              if not student_first_name.isalpha():
80                  raise ValueError("The first name should not contain numbers.")
81
82              student_last_name = input("Enter the student's last name: ")
83              if not student_last_name.isalpha():
84                  raise ValueError("The last name should not contain numbers.")
85
86              course_name = input("Please enter the name of the course: ")
87              # Changed to dictionary row
88              student_data = {"FirstName": student_first_name,
89                              "LastName": student_last_name,
90                              "CourseName": course_name}
91              students.append(student_data)
92              print(f"You have registered {student_first_name} {student_last_name} for {course_name}.")
93              continue
```

Fig. 27

And finally changed Menu item 3 to fulfill my last error handling acceptance criteria (Fig 28, 29), once again closely adhering to the examples provided in the material.

```
try:
    file = open(FILE_NAME, "w")
    json.dump(students, file)
    file.close()
    continue
except TypeError as e:
    print("Please check that the data is a valid JSON format\n")
    print("-- Technical Error Message -- ")
    print(e, e.__doc__, type(e), sep='\n')
except Exception as e:
    print("-- Technical Error Message -- ")
    print("Built-In Python error info: ")
    print(e, e.__doc__, type(e), sep='\n')
finally:
    if file.closed == False:
        file.close()
```

Fig. 28

```
115         elif menu_choice == "3":
116             try:
117                 file = open(FILE_NAME, "w")
118                 json.dump(students, file)
119                 file.close()
120                 print("The following data was saved to file!")
121                 for student in students:
122                     print(f"Student {student["FirstName"]} {student["LastName"]} is enrolled in {student["CourseName"]}")
123                 continue
124             except TypeError as e:
125                 print("Please check that the data is a valid JSON format\n")
126                 print("-- Technical Error Message -- ")
127                 print(e, e.__doc__, type(e), sep='\n')
128             except Exception as e:
129                 print("-- Technical Error Message -- ")
130                 print("Built-In Python error info: ")
131                 print(e, e.__doc__, type(e), sep='\n')
132             finally:
133                 if file.closed == False:
134                     file.close()
```

Fig. 29

Testing the changes

With seemingly all the changes I need to make to the code completed, I moved on to begin testing my new program. I started by entering student names into my input fields, then by making sure it displayed, and finally by saving it to the json file.

I also wanted to test my error messages, so I opted to include some inputs with numbers (Fig. 30). This sent up an error message that was visible to the user, so it seemed like it was ready to go.

```
What would you like to do: 1
Enter the student's first name: 12233
The first name should not contain numbers.
 -- Technical Error Message --
Inappropriate argument value (of correct type).
The first name should not contain numbers.
```

Fig. 30

I then ran the program through the command terminal, and packaged it up to submit for review by adding it to a GitHub repository.

Summary

Overall, this project went smoothly – much smoother than our last assignment. The program worked with the new json file and was displaying error messages when I wanted it to. To reflect on this experience, I would like to improve my workflow by reducing my dependence on directly referencing the labs and notes to complete individual steps. However, sticking closer to these suggestions did keep me from having to attempt to write the program twice, so that was certainly a better result.