Hannah Brown
5/17/2025
IT FDN 110 A
Assignment 05
https://github.com/hkbrown42/IntroToProg-Python-Mod05

Assignment 05 - Dictionaries and Exception Handling

Introduction

This week we learned about dictionaries and how they can be used to handle more complex data in a more human-friendly format than lists, as well as using JSON files and how to set up structured error handling. This document describes the process of how I modified last week's program to use JSON files and dictionaries, and added structured error handling for each step of the program.

Initial Setup

The first thing I did to set up the program was to import the json module, since I'll be using that later to load and save data from and to the "Enrollments.json" file. Next, I verified that the MENU constant was defined correctly, and changed the FILE_NAME constant from a CSV file to a JSON file. Next, I removed the csv_data variable, since I won't be needing that anymore, redefined the student_data variable as an empty dictionary, and verified that all other existing variables were initialized correctly as either empty strings, a None type object, or an empty list.

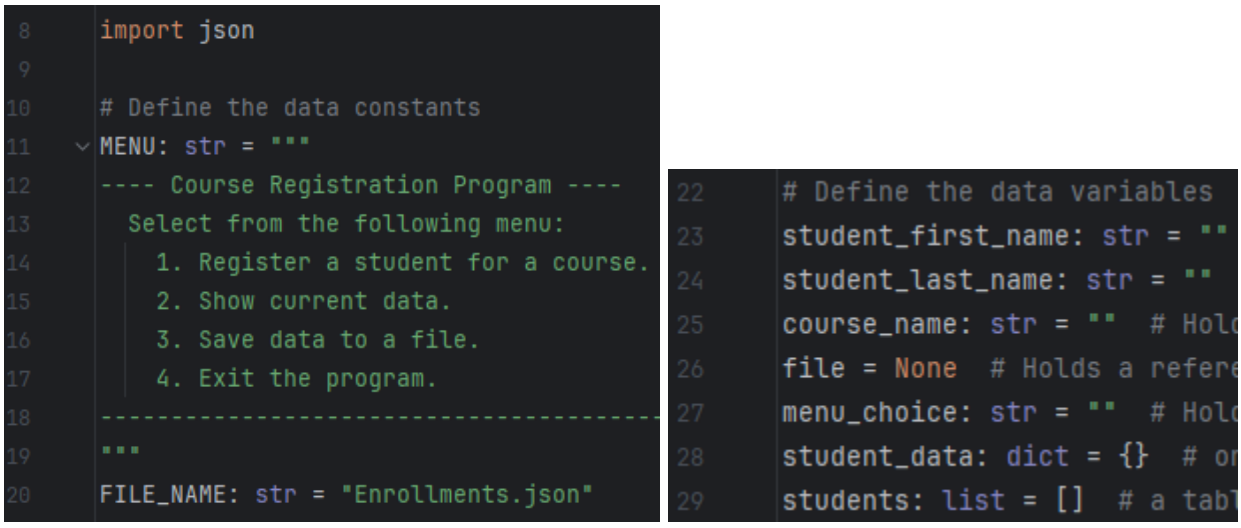


Figure 1-2: Module importing and definition of constants and variables.

I also copied the provided "Enrollments.json" file into the A05 project, so that there will be starting data to work with.

Writing the Program

Once the variables and constants are defined and the necessary module imported, the first step in the program is to read the contents of the “Enrollments.json” file and store it into a dictionary. However, I want to include structured error handling for this part, so I used a try-except construct here. First the program will try to perform the desired commands - opening the file, using the json.load() function to store the existing data into the previously-empty students list, and closing the file.

I then set up two possible error conditions using except statements. If the file the user is trying to open does not exist, the program will print a statement explaining what the error is, as well as the type of error (in this case, a FileNotFoundError). If trying to open, read, or load the file causes a different kind of error, the program will print a different statement, as well as the error type. I also added a finally statement after the try-except construct, so that if either of these exception states is met, the file will still be closed.

```
31  # When the program starts, read the file data into a list of lists (table)
32  # Extract the data from the file
33  try:
34      file = open(FILE_NAME, 'r')
35      students = json.load(file)
36      file.close()
37  except FileNotFoundError as e:
38      print("Please make sure the file you are trying to open actually exists!\n")
39      print(type(e))
40  except Exception as e:
41      print("There was a non-specific error!\n")
42      print(type(e))
43  finally:
44      if file.closed == False:
45          file.close()
```

Figure 3: Reading and storing the contents of the existing JSON file, with structured error handling.

Now that the existing data has been stored, I can move on to setting up each of the four menu options. For menu option 1, the basic code prompting the user’s input does not need to be modified, and after this, I set up the student_data dictionary with “FirstName”, “LastName”, and “CourseName” as keys, and the three user input variables as their respective values. I then used the append() function to add this dictionary to the students list.

However, I want to add structured error handling here as well, to prevent invalid input for the first name, last name, and course name. For this section specifically, if the user entered an invalid input, I wanted the program to return the user to the first user input prompt, so I placed this whole block inside a while loop and defined two possible options for invalid input - one where

the user enters a name that includes a number, and one where the user does not enter anything at all. If one of these inputs is entered, the program returns a statement telling the user why their input was invalid, and returns them to the first prompt. I also placed this whole segment inside a try-except construct, so that if there is an unexpected exception, the program will print the type of that error.

```
53      # Input user data
54      if menu_choice == "1":
55          while True:
56              try:
57                  student_first_name = input("Enter the student's first name: ")
58                  if not student_first_name.isalpha() and student_first_name != "":
59                      print("The first name cannot contain numbers.")
60                      continue
61                  if student_first_name == "":
62                      print("This field cannot be blank.")
63                      continue
64                  student_last_name = input("Enter the student's last name: ")
65                  if not student_last_name.isalpha() and student_last_name != "":
66                      print("The last name cannot contain numbers.")
67                      continue
68                  if student_last_name == "":
69                      print("This field cannot be blank.")
70                      continue
71                  course_name = input("Please enter the name of the course: ")
72                  if course_name == "":
73                      print("This field cannot be blank.")
74                      continue
75                  student_data = {"FirstName": student_first_name,
76                                  "LastName": student_last_name,
77                                  "CourseName": course_name}
78                  students.append(student_data)
79                  print(f"You have registered {student_first_name} "
80                        f"{student_last_name} for {course_name}.")
81                  break
82              except Exception as e:
83                  print("There was a non-specific error!\n")
84                  print(type(e))
```

Figure 4: Menu option 1, with structured error handling.

For menu option 2, the existing code does not need to be changed much; I just changed the print string to call the keys from the students list rather than the index, since the students list now contains dictionaries instead of lists.

```
86      # Present the current data
87      elif menu_choice == "2":
88          # Process the data to create and display a custom message
89          print("-" * 50)
90          for student in students:
91              print(f"Student {student["FirstName"]} {student["LastName"]} "
92                    f"is enrolled in {student["CourseName"]}")
93          print("-" * 50)
94          continue
```

Figure 5: Menu option 2

For menu option 3, as in the existing code, the program opens the file in write mode. However, I'll now use the json.dump() function to write the students list to the file. The program then prints out what data is currently saved to the file. I want to include structured error handling for this portion as well, so I placed this block of code inside another try-except construct. If the data to be saved is somehow not formatted correctly, the program will return a statement explaining the error, as well as the type of error. If there is any other kind of error, the program will print a different statement, as well as the error type. I also added a finally statement after the try-except construct, so that if either of these exception states is met, the file will still be closed.

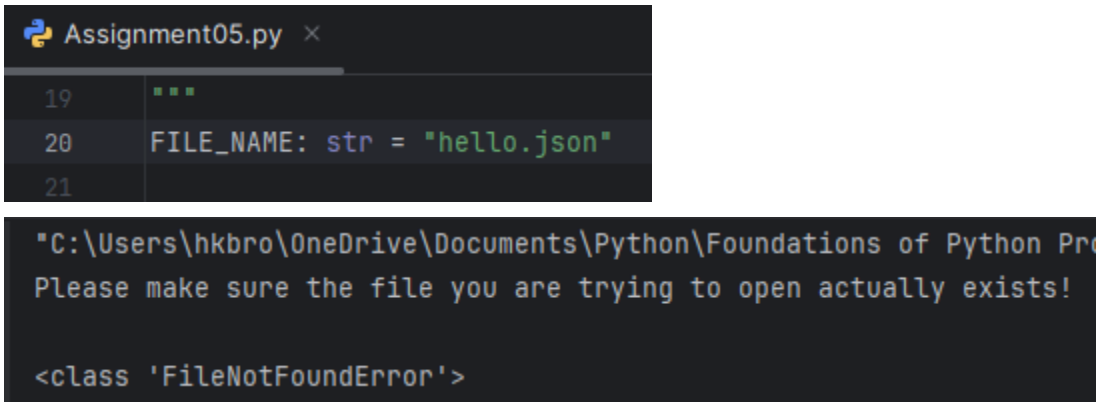
```
96      # Save the data to a file
97      elif menu_choice == "3":
98          try:
99              file = open(FILE_NAME, 'w')
00              json.dump(students, file, indent=2)
01              file.close()
02              print("-" * 50)
03              print("Here are the current student registrations: ")
04              for student in students:
05                  print(f"Student {student["FirstName"]} {student["LastName"]} "
06                        f"is enrolled in {student["CourseName"]}")
07              print("-" * 50)
08              continue
09          except TypeError as e:
10              print("Please make sure the data is a valid JSON format!\n")
11              print(type(e))
12          except Exception as e:
13              print("There was a non-specific error!\n")
14              print(type(e))
15          finally:
16              if file.closed == False:
17                  file.close()
```

Figure 6: Menu option 3, with structured error handling

Finally, for menu option 4, the original code needs no modification, as it simply needs to exit out of the while loop. Additionally, the final else statement of the loop defines what should happen if the user enters in an invalid menu option number. Once the while loop has been exited, the program displays a message to the user indicating that the program is completed, and then the program ends.

Testing the Program

The first test I tried was changing the FILE_NAME constant to the name of a file that doesn't exist, to make sure the program handled this error correctly - once I verified that it did, I changed the FILE_NAME constant back to the correct name.



Figures 7-8: Testing the opening/reading/loading error handling.

I first ran the program in PyCharm, and went through each of the menu options one by one. I selected menu option 1, where the program asked for the input as expected, then returned to the main menu. I went ahead and did this option twice, to enter multiple registrations. I also tried entering in several invalid inputs to make sure that the program handled these errors correctly.

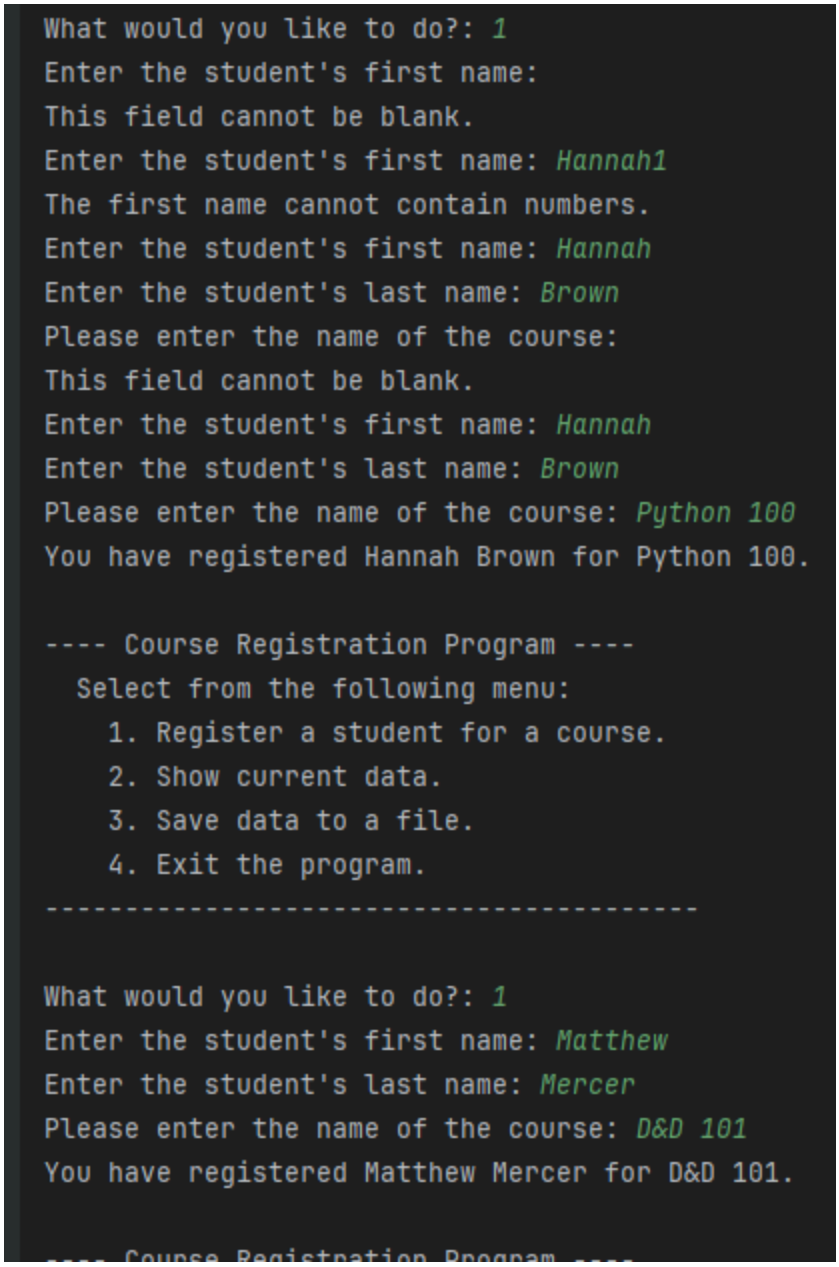


Figure 9: Testing menu option 1 with invalid inputs and multiple registrations.

I then tested menu option 2, and the program correctly displayed both the data from the original file, as well as the two registrations I just entered.

```
What would you like to do?: 2
-------------------------------------------------
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Hannah Brown is enrolled in Python 100
Student Matthew Mercer is enrolled in D&D 101
-------------------------------------------------
```

Figure 10: Testing menu option 2.

I then tested menu option 3, and the program displayed the current registrations again. I also verified that the “Enrollments.json” file now contained both the original and the new registrations.

```
What would you like to do?: 3
-------------------------------------------------
Here are the current student registrations:
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Hannah Brown is enrolled in Python 100
Student Matthew Mercer is enrolled in D&D 101
-------------------------------------------------
```

Enrollments - Notepad

File Edit Format View Help


```
[
  {
    "FirstName": "Bob",
    "LastName": "Smith",
    "CourseName": "Python 100"
  },
  {
    "FirstName": "Sue",
    "LastName": "Jones",
    "CourseName": "Python 100"
  },
  {
    "FirstName": "Hannah",
    "LastName": "Brown",
    "CourseName": "Python 100"
  },
  {
    "FirstName": "Matthew",
    "LastName": "Mercer",
    "CourseName": "D&D 101"
  }
```

Figures 11-12: Testing menu option 3, and JSON file with new data saved to it.
Finally, when I entered menu option 4, the program ended. I then ran the program in the command prompt, and ran through all the menu options again, again trying to enter invalid inputs to make sure the program handled the errors as expected. After verifying that each menu option worked here as well, I once again verified that the “Enrollments.json” file now contained both the original and new registrations.

```
    4. Exit the program.
----------------------------------------

What would you like to do?: 1
Enter the student's first name:
This field cannot be blank.
Enter the student's first name: 123
The first name cannot contain numbers.
Enter the student's first name: Maeve
Enter the student's last name: McCluskey
Please enter the name of the course:
This field cannot be blank.
Enter the student's first name: Maeve
Enter the student's last name: McCluskey
Please enter the name of the course: Python 100
You have registered Maeve McCluskey for Python 100.

---- 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 Hannah Brown is enrolled in Python 100
Student Matthew Mercer is enrolled in D&D 101
Student Maeve McCluskey is enrolled in Python 100
-------------------------------------------------

---- 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?: 3
-------------------------------------------------
Here are the current student registrations:
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Hannah Brown is enrolled in Python 100
Student Matthew Mercer is enrolled in D&D 101
Student Maeve McCluskey is enrolled in Python 100
-------------------------------------------------
```

Figure 13: Program running in command prompt.

Enrollments - Notepad

File Edit Format View Help

```
[
  {
    "FirstName": "Bob",
    "LastName": "Smith",
    "CourseName": "Python 100"
  },
  {
    "FirstName": "Sue",
    "LastName": "Jones",
    "CourseName": "Python 100"
  },
  {
    "FirstName": "Hannah",
    "LastName": "Brown",
    "CourseName": "Python 100"
  },
  {
    "FirstName": "Matthew",
    "LastName": "Mercer",
    "CourseName": "D&D 101"
  },
  {
    "FirstName": "Maeve",
    "LastName": "McCluskey",
    "CourseName": "Python 100"
  }
]
```

Figure 14: JSON file with new data saved to it.

Summary

While working on this program I came to discover that there were multiple ways to lay out structured error handling - the way I did it for menu option 2 wasn't exactly like how it was demonstrated in the content, but it worked in a way that I found satisfactory. I'm finding it interesting figuring out that there are often several different ways to achieve the same outcome of a program.