Adam Eichholz
August 13, 2025
Foundations of Programming: Python
Assignment 5

Advanced Collections and Error Handling

Introduction

In this paper, I look at the steps I took to complete the update to our student enrollment project. I will look at the hurdles I had when getting started, the steps I took to overcome them, and what my takeaways were for this week.

Topic

To start off the assignment for this week, I got started by looking at the assessment criteria for the project. I knew that one of the key objectives would be to use JSON files instead of CSVs, and I remembered from the video lessons that using JSON files would mean significant changes to the code blocks to import and save file data. Knowing that I would be needing that first step of loading saved data to run smoothly as I tested my iterations, I decided to start my changes there.

I quickly ran into a bit of an issue as I got started on this. I knew where I would need to make changes to my code, but I did not know exactly what the JSON-specific commands would actually be. While the notes helped a little bit with the command, there was a lot of focus on what we would write to create some initial data for Python to load for JSON, something that was already done for us, and was not needed for my script.

To get the commands I needed, I referred to the answers for our JSON lab. These proved to be exactly what I needed, and testing the code worked fine. I had already changed the file name variable, and because the constant uses the same “FILE_NAME”, I did not need to make any modifications. I was able to do the same thing for the save command as well, and when I tested my code, everything worked smoothly.

Figure 1 - JSON Loading and Saving

```
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()

# Present and Process the data
while (True):
```

Things progressed smoothly from there. I decided that the Exception Handling component would be the last thing I would tackle, since that would be simply adding modifications to the code as it existed, though I was concerned about how I might force the Menu Option 1 to only accept letters. Nonetheless, I decided to tackle the dictionary component next.

I made the appropriate change to my variables – I was very grateful to have the “TODO” comment explicitly made in the appropriate line! – and then started assessing where I needed to establish my key-values. It felt a bit strange to me to establish these in the block of my code that writes to a dictionary (Menu Option 1). I would have thought that these would need to be established earlier, similar to how we establish our variables in the beginning of the code. I did notice as well that, had we not been using a JSON file, we would have established these in the file loading step as well, which did make more sense to me. However, using the JSON command essentially makes this “invisible”. This still feels a bit strange to me, and I do not know that I have a great understanding as to why this is the way it is.

With all that said, with using the appropriate lab as a reference, I found making the dictionary updates to the code fairly straightforward. Testing went smoothly as well. I appreciated that using a dictionary meant not using the indexing method that a list does. That felt smoother and easier to read.

Figure 2 - Writing a Dictionary

```
# Input user data
if menu_choice == "1":  # This will not work if it is an integer!
    try:
        student_first_name = input("Enter the student's first name: ")
        if not student_first_name.isalpha():
            raise ValueError("The first name should not contain numbers.")

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

        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)
        print(f"You have registered {student_first_name} {student_last_name} for {course_name}.")
    except ValueError as e:
        print(e)  # Prints the custom message
        print("-- Technical Error Message -- ")
        print(e.__doc__)
        print(e.__str__())
```

Lastly, I started inserting the Exception Handling for the code. Again, I referenced the lab answers and made appropriate changes to fit my code. Testing the Exception Handling necessitated some creativity on my part - I needed to remove the Enrollments file from its folder to make sure the Exception worked properly, and I needed a couple of iterations on the Menu Option 1 Exceptions to make sure that it would not accept numbers as a value. I also found myself wishing I could indent multiple lines of code at once, since I needed to do this for the Exceptions to work properly. There might be a command for this in PyCharm, but I was not able to locate it.

Summary

Things went a lot more smoothly for me this week compared to last. I came into this week's assignment feeling like I had a very good understanding of the concepts we would be working on, but not the best understanding of how I would actually write the code. Fortunately, using the labs as a reference point, I was able to quickly and easily assemble my code and get the program working smoothly. I enjoyed using dictionaries as a way to cleanly and neatly sort and store information in multiple potential formats. I found JSON a bit strange to work with; despite its simplicity, it felt like I was relying on “invisible” code.