

Kyle Hayward

11/14/2023

Foundations of Programming: Python

Assignment 05

<https://github.com/HaywardUW/Python110-Fall2023>

# Dictionaries, JSON and Exception Handling

## Introduction

Module 05 introduced dictionaries, JSON files and the concepts of exception handling. I learned how to import the 'json' module and how to implement its code to simplify reading from and writing to JSON files. I also learned how to use exceptions with 'try', 'except', and 'finally' blocks to handle potential errors that a program might experience during execution.

In this Assignment 05 documentation, I will explain what a JSON file is, how to import the 'json' module, and a few common functions to use with JSON files.

I will then demonstrate how to create a Python program that extracts data from a JSON file, takes input from a user, displays the data back to the user, writes to the JSON file, and implements exception handling before exiting.

The last section of the documentation will demonstrate a few scenarios where the exception handling can be triggered.

The program will be run through both PyCharm and the Command Prompt to validate functionality in two separate environments.

## JSON Files and Functions

JSON stands for 'JavaScript Object Notation'. It is a commonly used file format in programming languages due to it being easy for humans to read and for computers to parse. JSON is most often used for communication between servers and clients.

Very similar to Python dictionaries, JSON files are constructed of key-value pairs. The Keys are always enclosed in double quotes, while the Values can be strings, integers, floats, Booleans, arrays, objects, or null. JSON data is formatted using curly braces {} for the objects and square brackets [] for the arrays. The following is an example of basic JSON file structure:

```
{  
  "Name": "Kyle Hayward",  
  "Age": "28",
```

```
"State": "Washington",  
  
"isPHD": False,  
  
"Courses": ["Python", "SQL", "VMWare", "C#"]  
  
}
```

The Python language has a library of pre-built code that can be imported into the program called modules. One such module is the 'json' module. The 'json' module can be imported into the program by simply entering 'import json' into the top of the code under the script header. (Figure 1.1)

```
# Import the JSON module  
import json
```

**Figure 1.1: Importing the 'json' module.**

After importing the 'json' module, a host of pre-built code will be available to utilize with JSON files. Two such code samples are the 'json.load()' and 'json.dump()' functions.

The 'json.load()' function takes the contents of the JSON file and loads it into whatever the programs calls for. In this example, the function loads the contents into a list called 'student\_list'. (Figure 1.2)

```
# Read the contents of the JSON file  
file = open(FILE_NAME, 'r')  
# Load contents of the JSON file into student_list  
student_list = json.load(file)  
# Close the file  
file.close()
```

**Figure 1.2: Utilizing the 'json.load()' function to load JSON data into a list.**

The 'json.dump()' function writes the list data into the JSON file as shown in Figure 1.3.

```
# Open "Enrollments.json" and writes the student list data to it  
file = open(FILE_NAME, "w")  
json.dump(student_list, file)  
file.close()
```

**Figure 1.3: Utilizing 'json.dump()' to write data to a JSON file.**

While a developer can certainly build a program without the 'json' module, it does however save several lines of complex code. For example, a 'for' loop with the 'readlines()', 'split()', 'strip()', and 'append()' functions are no longer needed when utilizing the 'json' module. I do recommend being familiar with the 'for' loop method of extracting data from files, but as one progresses in the programming journey, it's nice to know that such modules exist to simplify the process.

## Creating the Program

As with all the programs created in this course, I start with a script header. (Figure 2.1)

```
# ----- #
# Title: Assignment05
# Desc: This assignment demonstrates using dictionaries, files, and exception handling
# Change Log: (Who, When, What)
#   RRoot,1/1/2030,Created Script
#   Kyle Hayward,11/14/2023,Completed Script
# ----- #
```

**Figure 2.1: Starting the program with a script header.**

Since I am working with a JSON file and want to work with some pre-built code, I import the 'json' module into the program. (Figure 2.2)

```
# Import the JSON module
import json
```

**Figure 2.2: Importing the 'json' module.**

Next, I define the two Data Constants which are the Course Registration Menu and the FILE\_NAME holding the "Enrollments.json" file. (Figure 2.3)

```
# Defining the Data Constants
MENU: str = '''
-----Course Registration Menu-----
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
-----
'''
FILE_NAME: str = "Enrollments.json"
```

**Figure 2.3: Defining the Data Constant variables.**

I then define the seven data variables. Four of which are strings, one dictionary, one list and the 'file' variable set to None. (Figure 2.4)

```
# Defining the Data Variables
student_first_name: str = ""
student_last_name: str = ""
course_name: str = ""
file = None
menu_choice: str = ""
student_data: dict = {}
student_list: list = []
```

**Figure 2.4: Declaring the seven data variables.**

To start the program, I begin with an exception handling block to try opening the “Enrollments.json” file. If the file exists, the ‘load()’ function will load all the data contained in the JSON file into a list before closing. (Figure 2.5)

```
try:
    # Read the contents of the JSON file
    file = open(FILE_NAME, 'r')
    # Load contents of the JSON file into student_list
    student_list = json.load(file)
    # Close the file
    file.close()
```

**Figure 2.5: Beginning of the exception handling block to open and read from the JSON file.**

If the file does not exist, I add an exception to notify the user that the file does not exist and the program will now create the required JSON file. This is accomplished by utilizing the ‘open()’ function in ‘w’ mode. I also add a catch-all Exception to catch any unforeseen errors that might occur during program execution. (Figure 2.6)

```
# Present an error if "Enrollments.json" does not exist
except FileNotFoundError as e:
    print("This file doesn't exist! "
          "\nCreating the file and opening!")
    print("The file has been created!")
    # Creates "Enrollments.json" if the file does not exist
    file = open(FILE_NAME, 'w')
# Catch all exception for unforeseen errors
except Exception as e:
    print("There was a generic error opening the document!")
    print(e, e.__doc__,
          e.__str__())
```

**Figure 2.6: Continuing the exception handling block with FileNotFoundError and Exception errors.**

Lastly, I complete the exception handling block by including the ‘finally’ clause. The ‘finally’ clause checks if the file is open and closes it if ‘file.closed == False’. (Figure 2.7)

```
# Closes the file if it is not already closed
finally:
    if file.closed == False:
        print("Closing the file!")
        file.close()
```

**Figure 2.7: Utilizing the ‘finally’ clause to close out the exception handling block.**

Now that the JSON file has been created and read from, I start the ‘while’ loop by printing out the Course Registration Menu and prompting the user to make a menu selection. (Figure 2.8)

```

while True:
    # Print the Course Registration Menu
    print(MENU)
    # Use 'menu_choice' variable to store user's selection
    menu_choice = input("Please make a menu selection: ")
    print()

```

**Figure 2.8: Starting the 'while' loop with printing the menu and prompting for user selection.**

If the user chooses menu choice "1", I configure another exception handling block by first asking the user for the student's first name. The 'if not' statement checks to ensure that the first name only contains letters and does not contain numbers. This is accomplished with the 'isalpha()' function. I do the same for the student's last name. A ValueError will be raised if either name has anything other than letters contained in the input. (Figure 2.9)

```

if menu_choice == "1":
    try:
        print('*' * 50)
        # Prompt the user for input
        student_first_name = input("Please enter the student's first name: ")
        # Validate first name contains letters only
        if not student_first_name.isalpha():
            raise ValueError("\nThe first name should only contain letters!")
        student_last_name = input("Please enter the student's last name: ")
        # Validate last name contains letters only
        if not student_last_name.isalpha():
            raise ValueError("\nThe last name should only contain letters!")

```

**Figure 2.9: Prompting for user input while implementing ValueError exception handling.**

As the course name can contain numbers, I chose to use an 'if not' statement to ensure the user enters at least one character. If the course name input is blank, the program raises a ValueError exception informing the user that they did not enter any information. (Figure 2.10)

```

course_name = input("Please enter the course name: ")
# Validate user does not leave course name blank
if not course_name:
    raise ValueError("\nYou did not enter any course information!")

```

**Figure 2.10: Raising an exception if the user does not enter any course name information.**

The program then loads the user input data into a dictionary named 'student\_data'. The dictionary data is then appended to the student list. A summary of the user's input data is then displayed notifying the user of the First Name, Last Name and the Course in which was registered. (Figure 2.11)

```

# Adding data to the dictionary
student_data = {"FirstName": student_first_name,
                "LastName": student_last_name,
                "CourseName": course_name}
# Appending dictionary data to the student list
student_list.append(student_data)
print(f'\nYou have registered '
      f'{student_data["FirstName"]} '
      f'{student_data["LastName"]}, for '
      f'{student_data["CourseName"]}')
```

**Figure 2.11: Adding data to the dictionary and appending that data to the student list.**

I then complete the exception handling block by defining both the Value Error and Exception errors and assign them to a variable name 'e'. (Figure 2.12)

```

# Formatting the ValueError to the 'e' variable
except ValueError as e:
    print(e)
    print("---Technical Error Message---")
    print(e.__doc__)
    print(e.__str__())
# Formatting the Exception error to the 'e' variable
except Exception as e:
    print("A non-specific error has occurred!\n")
    print("---Technical Error Message---")
    print(e, e.__doc__,
          e.__str__(),
          type(e),
          sep='\n')
continue
```

**Figure 2.12: Formatting the ValueError and Exception errors to complete the exception handling.**

Next in the program I complete the code necessary to execute what happens when the user selects menu option "2". I begin this portion with an 'elif' statement and proceed by printing out all the data contained in the student list. List is accomplished with a 'for' loop that loops through all the items in the 'student\_list' variable and prints all the data. (Figure 2.13)



```

elif menu_choice == "2":
    print('*' * 50)
    print("The current data is: \n")
    # Loop through and print all the data in the student list
    for student in student_list:
        print(f'{student["FirstName"]}, '
              f'{student["LastName"]}, '
              f'{student["CourseName"]}!')
    print('*' * 50)
    continue

```

**Figure 2.13: Printing all data that is contained in the student list.**

I continue the program with another 'elif' statement to cover what happens when the user chooses menu option "3". This portion of the code also contains an exception block. I begin the 'try' block by opening the JSON file, dump the list data into the file using the 'json.dump()' function, close the file, and prompt the user regarding what data was saved. (Figure 2.14)

```

elif menu_choice == "3":
    try:
        # Open "Enrollments.json" and writes the student list data to it
        file = open(FILE_NAME, "w")
        json.dump(student_list, file)
        file.close()
        print('*' * 50)
        print("The following data is saved: \n")
        for student in student_list:
            print(f'{student["FirstName"]}, '
                  f'{student["LastName"]}, '
                  f'{student["CourseName"]}!')
    except:
        pass

```

**Figure 2.14: Dumping list data to the JSON file and notifying the user of what data was saved.**

As with the menu choice "1" block of code, I complete this block of code by including a TypeError and Exception error. The TypeError exception informs the user to validate that the file is indeed in JSON format. The Exception error is another catch-all exception for unforeseen errors during program execution. (Figure 2.15)

```

# Raising an exception to validate the JSON format
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")
# Catch-all exception
except Exception as e:
    print("---Technical Error Message---")
    print("Built-In Python Error Info: ")
    print(e, e.__doc__,
          type(e),
          sep='\n')

```

**Figure 2.15: Completing the error handling block with a TypeError and Exception error.**

The final portions of the code close the program if menu option “4” is chosen, prompts the user to enter a valid option if 1, 2, 3, or 4 is not selected, and notifies the user that the program has ended. (Figure 2.16)

```

# Close the program
elif menu_choice == "4":
    break

# Prompt the user to choose a valid menu option
else:
    print("Please choose option 1, 2, 3, or 4!")

print('*' * 50)
# Inform the user that the program has ended
print("The program has ended!")
print('*' * 50)

```

**2.16: Closing the ‘if’ loop with an ‘elif’ and ‘else’ statement and notifying the user of program end.**

This completes all the required code to create the program that extracts data from a JSON file, takes input from a user, displays the data back to the user, writes to the JSON file, and implements exception handling before exiting.

The next sections will demonstrate launching the program within the PyCharm IDE, running the program to completion in the Command Prompt, and purposefully introduces user input error to show how the program executes the exception handling.



## Running the Program – PyCharm

To avoid this documentation becoming too long winded, I will be demonstrating that the program launches successfully in the PyCharm IDE but I will not run the program to completion. I will cover full execution in the next section within the Command Prompt.

Launch the PyCharm IDE and open “Assignment05.py”. Right-click anywhere within the program code window. Click **Run ‘Assignment05’**. (Figure 3.1)

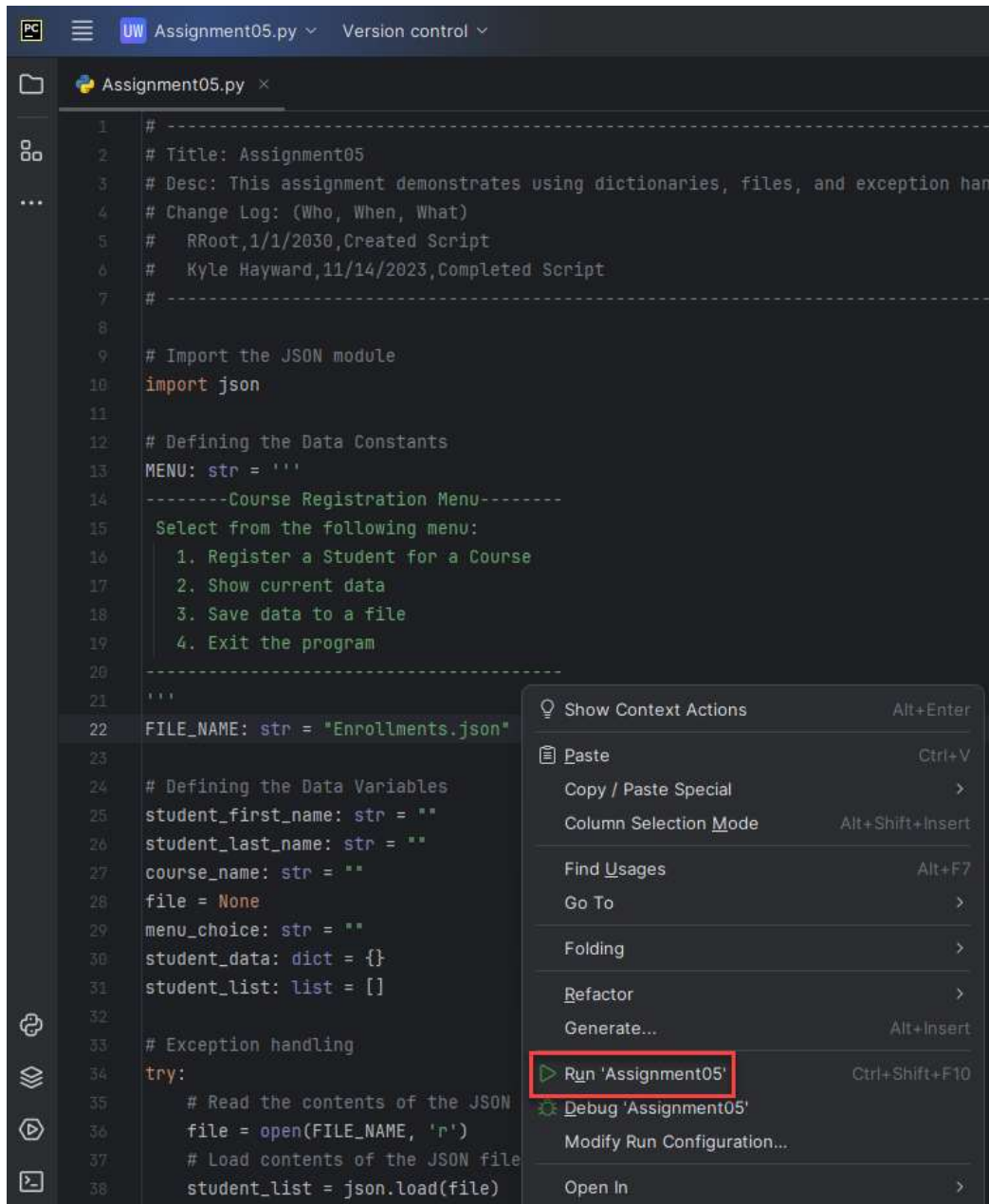
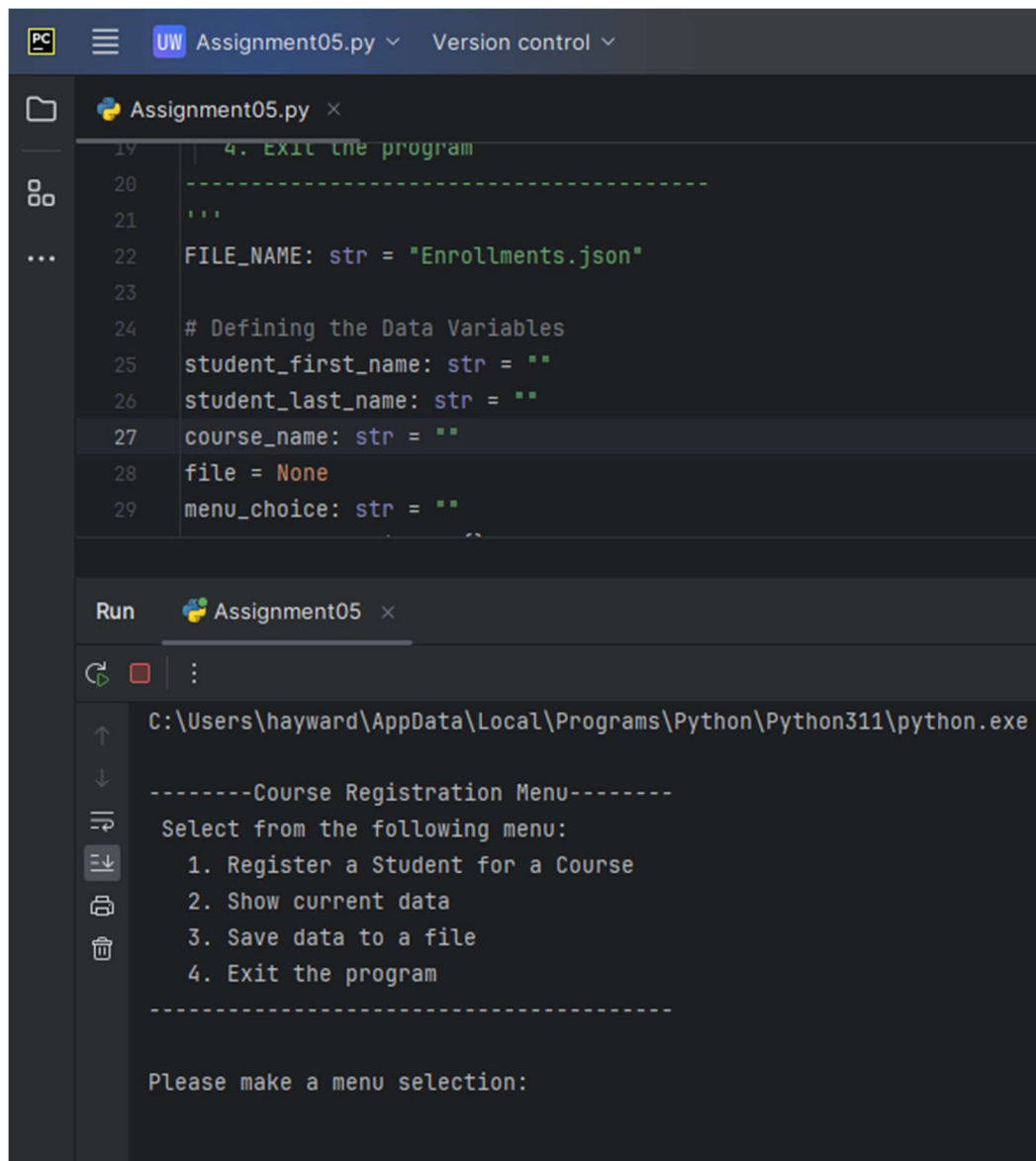


Figure 3.1: Running ‘Assignment05.py’ in the PyCharm IDE.

The program successfully runs within the PyCharm IDE as shown in Figure 3.2.



The image shows the PyCharm IDE interface. The top bar displays 'UW Assignment05.py' and 'Version control'. The left sidebar shows the project structure with 'Assignment05.py' selected. The main editor window displays the following Python code:

```
19 | 4. Exit the program
20 | -----
21 | '''
22 | FILE_NAME: str = "Enrollments.json"
23 |
24 | # Defining the Data Variables
25 | student_first_name: str = ""
26 | student_last_name: str = ""
27 | course_name: str = ""
28 | file = None
29 | menu_choice: str = ""
```

Below the editor, the 'Run' tab is active, showing the command prompt output:

```
C:\Users\hayward\AppData\Local\Programs\Python\Python311\python.exe
-----Course Registration Menu-----
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
-----

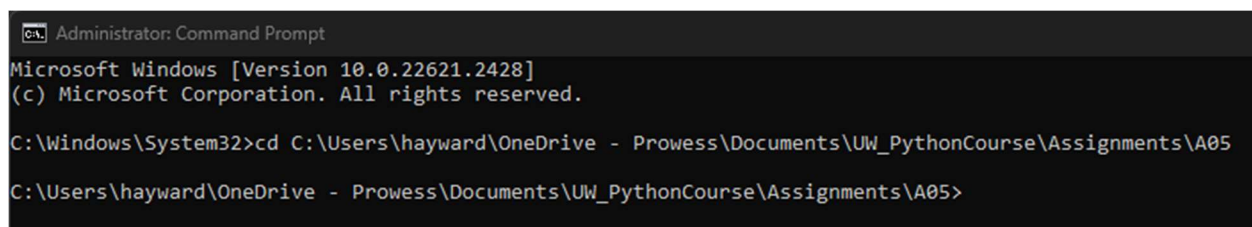
Please make a menu selection:
```

**Figure 3.2: Starting the program within the PyCharm IDE.**

The next section will demonstrate running the program to completion within the Command Prompt.

## Running the Program – Command Prompt

Launch the Command Prompt and navigate to the directory where the 'Assignment05.py' program resides. (Figure 4.1)



The image shows an Administrator Command Prompt window. The text displayed is:

```
C:\Windows\System32>cd C:\Users\hayward\OneDrive - Prowess\Documents\UW_PythonCourse\Assignments\A05
C:\Users\hayward\OneDrive - Prowess\Documents\UW_PythonCourse\Assignments\A05>
```

**Figure 4.1: Navigating to the location of the assignment within the command prompt.**

Type 'python Assignment.05.py' and press Enter on the keyboard to start the program. (Figure 4.2)

```
C:\Windows\System32>cd C:\Users\hayward\OneDrive - Prowess\Documents\UW_PythonCourse\Assignments\A05
C:\Users\hayward\OneDrive - Prowess\Documents\UW_PythonCourse\Assignments\A05>python Assignment05.py

-----Course Registration Menu-----
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
-----

Please make a menu selection:
```

**Figure 4.2: Program being launched in the command prompt.**

Figure 4.3 to 4.7 will demonstrate the program being run to completion including the message that appears when an invalid menu selection is input.

```
Please make a menu selection: 1

*****
Please enter the student's first name: Chong
Please enter the student's last name: Kim
Please enter the course name: SQL 101

You have registered Chong Kim for SQL 101
*****
```

**Figure 4.3: Registering a student with menu option 1 and displaying data back to the user.**

```
Please make a menu selection: 2

*****
The current data is:

Kyle, Hayward, Python 101!
Chong, Kim, SQL 101!
*****
```

**Figure 4.4: Displaying all data contained in the student list.**

```

Please make a menu selection: 3

*****
The following data has been saved:

Chong, Kim, SQL 101
*****

```

**Figure 4.5: Saving the recently entered data to the JSON file.**

```

Please make a menu selection: 5

Please choose option 1, 2, 3, or 4!

-----Course Registration Menu-----
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
-----

```

**Figure 4.6: Program prompts the user when an invalid menu choice has been selected.**

```

Please make a menu selection: 4

*****
The program has ended!
*****

```

**Figure 4.7: The program ends when menu choice 4 has been selected.**

The program has now been run to completion. The next section will cover what happens when errors are encountered. This is where exception handling comes into play.

## Validate Exception Handling

To demonstrate how one portion of the exception handling code functions in this program, I have deleted the JSON file. What happens if I run the program without an existing file? The program catches that the file does not exist with the `FileNotFoundError` exception. It then creates the file to prevent the error from occurring in future program runs. (Figure 5.1)

```
C:\Users\hayward\OneDrive - Prowess\Documents\UW_Python
This file doesn't exist!
Creating the file and opening!
The file has been created!
Closing the file!

-----Course Registration Menu-----
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
-----

Please make a menu selection:
```

**Figure 5.1: The program creates the file since it doesn't already exist and continues to the menu.**

Moving to the menu choices; the program expects the student's first name and last name to contain only letters. It is not expecting any numbers. If numbers are discovered, the program will throw a ValueError and inform the user to enter a name that contains letters only. (Figure 5.2)

```
Please make a menu selection: 1

*****
Please enter the student's first name: 1Ning

The first name should only contain letters!
---Technical Error Message---
Inappropriate argument value (of correct type).

The first name should only contain letters!

-----Course Registration Menu-----
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
-----
```

**Figure 5.2: Program throwing a value error since the name contained characters other than letters.**

Another ValueError exception is implemented when the program prompts the user to enter a course name. However, in this case, the program validates that the input is not left blank. If left blank, a message is displayed to the user that they did not enter any course information. (Figure 5.3)



```
Please make a menu selection: 1
*****
Please enter the student's first name: Kyle
Please enter the student's last name: Hayward
Please enter the course name:

You did not enter any course information!
---Technical Error Message---
Inappropriate argument value (of correct type).
You did not enter any course information!
```

**Figure 5.3: User fails to enter course information and the program responds with a message.**

The exception handling caught all the errors that were thrown at it and handled them in a graceful manner. This is the sign of a complex and well written program that anticipates potential issues and corrects them in a way that does not affect the user experience negatively nor break the program itself.

## Summary

Module 05 covered dictionaries, JSON, files, exception handling and an introduction to importing Python modules. The addition of these learnings allowed me to write a more complete and professional program that implements error checking with 'try', 'except', and 'finally' blocks. These blocks of code are imperative in anticipating potential errors and addressing them gracefully.

I found the 'json' module particularly intriguing. Implementing the 'json.load()' function saves several lines of confusing code. While that code is important to learn, having a module that bypasses that is quite helpful.

As requested, this program's code has also been uploaded to my GitHub repo:

<https://github.com/HaywardUW/Python110-Fall2023>

Direct link to the code: <https://github.com/HaywardUW/Python110-Fall2023/blob/main/Assignment05.py>