

Jessica Yun

November 9, 2023

ID FDN 110 A

Assignment 05

<https://github.com/jssoyoung/IntroToProg-Python-Mod05>.

Dictionaries and Errors

Introduction

This week, I learned how about dictionaries and error handling. Using my new findings, I created a program that will display a menu with options that the user could choose from. Once the program starts, the contents of the file are automatically read into a two-dimensional list table. Using the menu, the user has the options to store their first name, last name, and course name using inputs, have the collected data displayed for them, open and write in a file, or end the program. Once the user clicks on an option, the user could do that option and once they are finished, will automatically be taken back to the menu options again until they choose the program end option. The information the user inputs is also stored into a two-dimensional list table. Below is a step-by-step guide on how I created my program:

Creating Constants and Variables

To start my program, I began by opening the pycharm application within my computer. I created a header script that included who created the program, when it was worked on, and what changes were made. Once my program was created, I had to import a few imports since I will be using json. I imported json so I could use all the features of json. From json, I also imported JSONDecodeError that I will use later for my error handling and from typing, I imported TextIO that will be used for my file variable. Figure 1 shows the list of imports used for this program.

```
import json
from json import JSONDecodeError
from typing import TextIO
```

Figure 1. Imports used

Then I had to give the program some constants and variables. A big difference between declaring constants and variables are the way they are written. I wrote the constants in all caps locks while the variables were all lower-case letters. I created the constant MENU: str and set it equal to the value “---- 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

”. Since this is a multi-lined string, I put the string within three quotation marks to keep the string in multiple lines. Another constant that I created was: FILE_NAME: str = “Enrollments.json”.

Figure 2 below shows all the constants written out.

```
# Define the Data Constants
MENU: str = '''
---- 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.
-----
'''

# Define the Data Constants
FILE_NAME: str = "Enrollments.json"
```

Figure 2. Data Constants Code

After I defined all the data constants, I moved on to the variables. Just like I did for the constants, I included the data type next to the name of the variable for this program. I created multiple variables and set them equal to an empty string. I did this by setting them equal to `""`. By setting these variables equal to empty strings, this allows for data to be inputted into the variables in the future based on user input. I created the variables: `student_first_name: str = ""`, `student_last_name: str = ""`, `course_name: str = ""`, `menu_choice: str = ""`, and `csv_data : str = ""`. I also created a variable called `file` but set it equal to `TextIO`. I created one list typed variable. I set `students: list` equal to `[]`. The empty parenthesis stands for an empty list. I also created a dictionary variable. `Student_data: dict` was set equal to `{}`. The curly brackets stands for an empty dictionary. Figure 3 shows all the data variables created for this program.

```
# Define the Data Variables and constants
student_first_name: str = '' # Holds the first name of a student entered by the user.
student_last_name: str = '' # Holds the last name of a student entered by the user.
course_name: str = '' # Holds the name of a course entered by the user.
student_data: dict = {} # one row of student data
students: list = [] # a table of student data
csv_data: str = '' # Holds combined string data separated by a comma.
file = TextIO # Holds a reference to an opened file.
menu_choice: str = '' # Hold the choice made by the user.
```

Figure 3. Data Variables Code

Using Files

When the program starts, the contents of the “Enrollments.json” should automatically read into a two-dimensional list table. In order to make this occur, I began with a try statement. I set `file` equal to `open(FILE_NAME, “r”)`. This will open the constant `FILE_NAME` which has our `enrollments.json` file. The “r” stands for a read-only version of this file to be opened. I set `students` equal to `json.load(file)` to load the file. I then ended this portion of the code with a `file.close()`. I included some error handling within this portion in case there is no file with the name or if there was any error in opening the file. The first error handling states except `FileNotFoundError` as `e`: `print(‘Text file not found’)`, `print(‘---Technical Information---’)`, `print(e, e.__doc__, type(e), sep=‘\n’)`, `print(“Creating file since it doesn’t exist”)`, `file = open(FILE_NAME, ‘w’)`. This means if there is not file with that name, to display to the user all of the technical information with the type of error and then to create a file with

“Enrollments.json” as its name. The next error is for the JSONDecodeError that I imported. I wrote except JSONDecodeError: print(‘Data in file isn’t valid. Resetting it...’), file = open(FILE_NAME, ‘w’), json.dump(students, file). This resets the file if the data within the file is not valid. The last exception is for any other exceptions. I wrote except Exception as e: print(‘Unhandled exception’), print(‘---Technical Information---’), print(e, e.__doc__, type(e), sep=’\n’). I ended my error handling by stating finally: if not file.closed: file.close() which will close any opened file. Figure 4 shows the code for using file including all of the error handling code.

```
# When the program starts, read the file data into a list of lists (table)
# Extract the data from the file
try:
    file = open(FILE_NAME, "r")
    students = json.load(file)
    file.close()
except FileNotFoundError as e:
    print('Text file not found')
    print('---Technical Information---')
    print(e, e.__doc__, type(e), sep='\n')
    print("Creating file since it doesn't exist")
    file = open(FILE_NAME, 'w')
except JSONDecodeError:
    print('Data in file isn\'t valid. Resetting it...')
    file = open(FILE_NAME, 'w')
    json.dump(students, file)
except Exception as e:
    print('Unhandled exception')
    print('---Technical Information---')
    print(e, e.__doc__, type(e), sep='\n')
finally:
    if not file.closed:
        file.close()
```

Figure 4. Code for using files

Input and Output

Now that the variables and constants are defined, the program needs to ask the user to choose a menu option. I began with a while loop that said `while (True):`. The `while True:` allows the menu options to continuously loop and run until the user chooses option 4 which will contain a `break` and end the loop. Since the text of the menu is all stored within the `MENU` variable, I just wrote `print(MENU)` to have the text appear for the user. Then, I set `menu_choice = input("What would you like to do: ")`. This will print the statement for the user so they could input a number. The number they choose will then get stored within the `menu_choice` variable. Figure 5 shows the code used to present the menu and the list of choices.

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

    # Present the menu of choices
    print(MENU)
    menu_choice = input("What would you like to do: ")
```

Figure 5. Code to present the menu and list of choices

Menu choice 1 will prompt the user for their first name, last name, and course name. Then it will store the inputs within the respective variables. In order to do this, I began by writing an `if` statement. This option should only appear if the user inputs “1” and their menu choice. I wrote `if menu_choice == “1”:` to begin my `if` statement. Then I wrote another `try` statement and began with `try:`. Then I created a variable called `student_first_name`. I set this variable equal to `input(“Enter the student’s first name: ”)`. This `input()` function asks the user any code is recording within the parenthesis. The question also needs to be stored within quotes as it is a form of a string. I wrote in some error handling that said `if not student_first_name.isalpha(): raise ValueError(‘The first name must be alphabetic’)`. This error will run if the user tries to enter a name that is not alphabetic. On the next line, I created wrote the same piece of code but this time asking “Enter the student’s last name?” and stored it to a variable called `student_last_name`. I put the same error handling for this input as well so only alphabetic values could be entered for the user’s last name. The last input was `course_name = input(“Please enter the name of the course:`

“). Since we want the data collected from menu choice 1 to be added to a two-dimensional list table, I continued by setting `student_data = {'first_name': student_first_name, 'last_name': student_last_name, 'course_name': course_name}`. I appended the `student_data` list into the `students` list by using `append()`. I created a print function to tell the user that they were successfully registered by printing `(f'You have registered {student_first_name} {student_last_name} for {course_name}.')` I ended this piece of code with another error handling that state `except ValueError as e: print(e), print ('---Technical Information---'), print (e, e.__doc__, type(e), sep='\n')`. Figure 6 shows a picture of the full code using inputs.

```
# 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 must be alphabetic')
        student_last_name = input("Enter the student's last name: ")
        if not student_last_name.isalpha():
            raise ValueError('The last name must be alphabetic')
        course_name = input("Please enter the name of the course: ")
        student_data = {'first_name': student_first_name, 'last_name': student_last_name, 'course_name': 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)
        print('---Technical Information---')
        print(e, e.__doc__, type(e), sep='\n')
```

Figure 6. Code for input

Menu choice 2 will present a coma-separated string by formatting the collected data until the `print()` function. I started by adding to my `if` statement and created an `elif` (else if) statement. I wrote `elif menu_choice == "2":`. This block of code will only run if the user inputs 2 as their menu choice. Then, I wrote a print function that states `“---*50)` followed by a for loop. The for loop is for `student_data` in `students`:. This is calling upon all the `student_data` that was stored in the `students` list. I set `student_first_name` variable equal to the `student_data['first_name']`. This is the `first_name` key within the `student_data` dictionary. I did the same setting `student_last_name` equal to `student_data['last_name']` which will be the `last_name` key in the dictionary and `course_name` equal to `student_data['course_name']`. There are multiple ways to print this next block of code; however, I decided to write it using the f-string method. I set message equal to `f'Student {student_first_name} {student_last_name} is enrolled in {course_name}'`. In the f-string method, the name of variables are used as expressions inside curly-braces `{}`. I then printed(`“---*50)`. Figure 7 shows what the full code will look like written together.

```

# Present the current data
elif menu_choice == "2":

    # Process the data to create and display a custom message
    print("-"*50)
    for student_data in students:
        student_first_name = student_data['first_name']
        student_last_name = student_data['last_name']
        course_name = student_data['course_name']
        print(f"Student {student_first_name} {student_last_name} is enrolled in {course_name}")
    print("-"*50)

```

Figure 7. Code for output

Processing

The third menu choice opens a file named “Enrollments.json”. I began this portion of the code with another elif statement and wrote `elif menu_choice == “3”:`. This means if the user chooses the third menu choice, the next block of code will occur. I started with a try statement that I will use for the error handling later within the code. In order to get the program to open a file in write mode, I used file processing. I began by creating a variable called `file` and set it equal to `open(“Enrollments.json”, “w”)`. “Enrollments.json” is what the file will be saved as. I used “w” which stands for opening the writing mode. This was because the file was already created when the program first started but in read-only so now it can be written on. I wrote `json.dump(students, file)` and closed the file using the `close()` function. For my error handling, I wrote `except TypeError as e: print(‘JSON data was malformed’, print (‘---Technical Information---’), print (e, e.__doc__, type(e), sep=’\n’)`. This error handling will happen if there was malformed data. For all the other errors, I wrote `except Exception as e: print (‘---Technical Information---’), print (e, e.__doc__, type(e), sep=’\n’)`. I ended this piece of code with finally: `if not file.closed: file.close()` to close all open files again. Figure 8 portrays the code for file processing used for this program with the error handling.

```

# Save the data to a file
elif menu_choice == "3":
    try:
        file = open(FILE_NAME, "w")
        json.dump(students, file)
        file.close()
    except TypeError as e:
        print('JSON data was malformed')
        print('---Technical Information---')
        print(e, e.__doc__, type(e), sep='\n')
    except Exception as e:
        print('---Technical Information---')
        print(e, e.__doc__, type(e), sep='\n')
    finally:
        if not file.closed:
            file.close()

```

Figure 8. Code for processing

The last menu choice will end the program. For option 4, I wrote `elif menu_choice == : break`. This will break the loop of the menu options re-running and will end the whole program. I ended with an else statement and wrote `else: print("Please only choose option 1, 2, or 3")`. This states that if the menu option is not 1, 2, or 3, this block of code will print. I printed "Program Ended" to notify the user that the program is ending. Figure 9 shows the code to stop the loop.


```
# Stop the loop
elif menu_choice == "4":
    break # out of the loop

else:
    print("Please only choose option 1, 2, or 3")

print("Program Ended")
```

Figure 9. Code to stop the loop

Testing

To make sure that the code is running smoothly and without any errors, a test should be run.

Before any testing could happen, the program must be saved prior to every test. Once the program is saved, the test could be run using PyCharm or terminal.

I first ran the program using PyCharm. Since this program has multiple different options for the user to choose, I tested the program four different times, once for each option. Figure 10 shows the program running through PyCharm and selecting option 1. The program asked me for my first name, last name, and course name. I was shown a print that the student was enrolled for the course before I was redirected to the menu again and ask to select another option. I could continuously select option one and add as many students as I wish.

```
---- 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: 1  
Enter the student's first name: Bob  
Enter the student's last name: Smith  
Please enter the name of the course: Python100  
You have registered Bob Smith for Python100.
```

Figure 10. Option 1 selected on PyCharm

I proceeded to test option 2. When I input 2, the program displayed all the first names, last names, and course names all separated by commas. It then displayed the menu again for me and redirected me to choose another option. Figure 11 shows option 2 being selected.

```
---- 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 Jessica Yun is enrolled in Python100  
Student Bob Smith is enrolled in Python100  
-----
```

Figure 11. Option 2 selected on PyCharm

Below is option 3 being tested. When I input 3, all my inputted data got saved to the “Enrollments.json” file. Figure 12 shows option 3 being selected while figure 13 shows all the data being written into the Enrollments.json file. The option then displays the menu again and asks the user to select another option.

```
---- 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
```

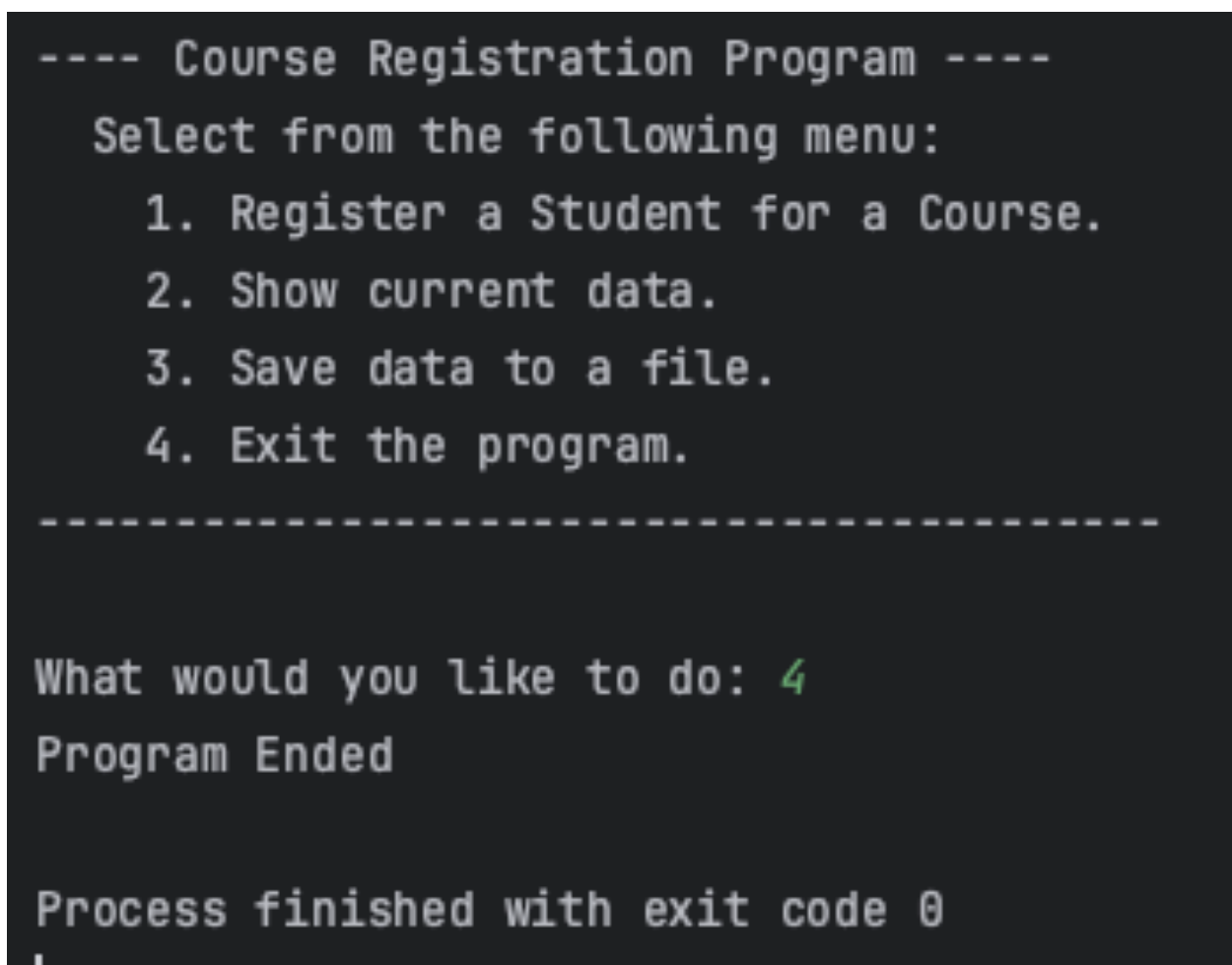
Figure 12. Option 3 selected on PyCharm



```
Assignment05.py  Enrollments.json x
1 [{"first_name": "Jessica", "last_name": "Yun", "course_name": "Python100"}, {"first_name": "Bob", "last_name": "Smith", "course_name": "Python100"}]
```

Figure 13. Data written into the Enrollments.csv file

Finally, I tested option 4. When I input the number 4, I got a printed message that said the program ended and a message that the process was finished. The menu option did not appear for this option which means the loop stopped. Figure 14 shows a picture of option 4 being chosen and the program ending.



```
---- 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: 4
Program Ended

Process finished with exit code 0
```

Figure 14. Option 4 selected on PyCharm

Then I ran the test using terminal. I opened terminal and located my code. When the code has been located, I wrote “Python3” along with the name of the program. Figure 15 displays the test being run on terminal with all four of the options being selected. When I selected option 1, I was

asked for a first name, last name, and course name. Option 2 displayed this information back to me in a coma-separated string. Option 3 also displayed this information for me and wrote my data to the Enrollments.json file on my computer. Option 4 ended the program and stopped the loop.

```
Last login: Wed Nov  8 11:15:42 on ttys000
[jessica@Jessicas-MacBook-Pro ~ % cd Desktop/A05
[jessica@Jessicas-MacBook-Pro A05 % Python3 Assignment05.py
```

```
---- 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: 1
```

```
Enter the student's first name: Sam
```

```
Enter the student's last name: Jones
```

```
Please enter the name of the course: Python100
```

```
You have registered Sam Jones 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 Jessica Yun is enrolled in Python100
```

```
Student Bob Smith is enrolled in Python100
```

```
Student Sam Jones is enrolled in 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: 3
```

```
---- 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.
-

Figure 15. Testing using terminal

Summary

These were the steps taken to create this program. While creating this program, I thought I would run into a lot of bugs like I did in my previous programs. Surprisingly, with all the error handling that was included while making this program, it made it really easy to debug any issues I came across while creating this program. This week's Lab Review videos were especially helpful in knowing how to being on this program. I was able to successfully create a program where the user could choose a menu option and based on their input, it would ask them to input a name/course, output information stored in constants and variables, open and update a writing file or stop the program altogether. This program also had a lot of error handling that will pop up if the user uses the program incorrectly.