Jessica Yun

November 20, 2023

ID FDN 110 A

Assignment 06

https://github.com/jssoyoung/IntoToProg-Python-Mod06

Classes and Functions

Introduction

This week, I learned how about classes and functions. 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. This program was created using two classes and multiple functions. 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 one import since I will be using json. I imported json so I could use all the features of json. 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". 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 one variable 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 variable menu_choice: = "". I also created one list typed variable. I set students: list equal to []. Figure 1 below shows all the inputs, constants, and variables written out.

```
import json

# 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.
-----------------------------------------
'''
FILE_NAME: str = 'Enrollments.json'
students: list = []  # a table of student data
menu_choice = ''
```

Figure 1. Imports and data constants code

File Processing

I had to create two classes for this program. I created one name called FileProcessor. This class will be used to store all of the functions for file processing. I started this block of code by writing class FileProcessor:. All classes must be written camel case and have the word “class” written in front of it. I then started my first function which would be the read_data_from_file function. This function will be used as soon as the program starts in order to open the Enrollments.json file. I started with the @staticmethod and def read_data_from_file. Def is needed in front of every function. Within the parenthesis, I put two parameters which is file_name and student_data. For each of my files, I wrote a little comment on what the function is doing, returning, and their parameters. Since they are multiple lines, I wrote them within three quotation marks. I began the function 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 student_data 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: IO.output_error_messages("Text file must exist before running this script!", e. The IO.output_error_messages will be another function described later on within my IO class that will hold all the error message code. The next exception is for any other exceptions. I wrote except Exception as e: IO.output_error_messages("There was a non-specific error!", e). I ended my error handling by stating finally: if file.closed == False: file.close() which will close any opened file. Then I returned student_data. Figure 2 shows the code for this function.

```
# Processing ------------------------------------ #
class FileProcessor:
    # When the program starts, read the file data into table
    # Extract the data from the file
    # Read from the Json file
    @staticmethod
    def read_data_from_file(file_name: str, student_data: list):
        '''
        The function reads JSON information from the specified file
        :param file_name: A string indicating the file name
        :return:The student data which is of type list[dict[str, str | float]]
        '''
        try:
            file = open(FILE_NAME, "r")
            student_data = json.load(file)
            file.close()
        except FileNotFoundError as e:
            IO.output_error_messages( message: "Text file must exist before running this script!", e)
        except Exception as e:
            IO.output_error_messages( message: "There was a non-specific error!", e)
        finally:
            if file.closed == False:
                file.close()
        return student_data
```

Figure 2. Function to read data from file

The next function for this class was the write_data_to_file function. I began by writing def write_data_to_file again to start my function. I had the same two parameters being file_name and student_data. I wrote in a comment about the function's purpose and parameters. I also started this function with a try loop. 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(FILE_NAME, "w").

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(student_data, file) and closed the file using the close() function. I printed a function for my user that will say “The following data was saved to file!”. I will also display the student’s name is enrolled in the student’s course. I did this by writing a for loop. For student in students: I printed(f’Student {student[“FirstName”]}’ f’{student[“LastName”]} is enrolled in {student[“CourseName”]}’). This used the f string to grab the student list. Within the list, it grabbed the dictionary key of their FirstName, LastName, and CourseName. For my error handling, I wrote except TypeError as e: IO.output_error_messages(“Please check that the data is a valid JSON format”, e). For all the other errors, I wrote except Exception as e: IO.output_error_messages(“There was a non-specific error!”, e). I ended this piece of code with finally: if file.closed == False: file.close() to close all open files again. Figure 3 shows the function to write data to file which will be used later for menu option 3.

```
@staticmethod
def write_data_to_file(file_name: str, student_data: list):
    '''
    The function writes JSON information into the specified file
    :param student_data: The roster to add to
    '''
    try:
        file = open(FILE_NAME, "w")
        json.dump(student_data, file)
        file.close()
        print("The following data was saved to file!")
        for student in students:
            print(f'Student {student["FirstName"]} '
                  f'{student["LastName"]} is enrolled in {student["CourseName"]}')
    except TypeError as e:
        IO.output_error_messages( message: "Please check that the data is a valid JSON format", e)
    except Exception as e:
        IO.output_error_messages( message: "There was a non-specific error!", e)
    finally:
        if file.closed == False:
            file.close()
```

Figure 3. Function to write data to file

IO Class

I then moved onto my second class which is class IO:. This class will hold all of the other functions in order to run this program. I wrote @static method def output_error_messages to have my error handling function. I set the parameters to message and error: Exception to None. I will print the error message by writing print(message, end="\n\n"). I wrote an if statement that if error is not None: it will print("—Technical Error Message –") and print(error, error.__doc__, type(error), sep='\n'). I will use this function whenever I need to output an error handling for any other part of my code. Figure 4 shows the code for error handling.

```
class IO:
    @staticmethod
    def output_error_messages(message: str, error: Exception = None):
        '''
        Prints an error message to the user
        :param message: The message to print
        :param exception: The exception for the error
        :return: The error message
        '''
        print(message, end="\n\n")
        if error is not None:
            print("-- Technical Error Message -- ")
            print(error, error.__doc__, type(error), sep='\n')
```

Figure 4. Function to output error messages

The next function will be used to display the menu. I started with my @staticmethod def output_menu():. The only parameter for this function is menu which will be a string. Within the function I wrote print(), print(menu), print(). The extra print functions were used just to add extra space to make the menu appear nicer for the user when displayed. Figure 5 displays the output_menu function.

```python
@staticmethod
def output_menu(menu: str):
    '''
    Displays the menu options for the user
    :param menu: The string with the menu options
    :return: The user's string choice from 1, 2, 3, 4
    '''
    print()
    print(menu)
    print()  # Adding extra space to make it look nicer.
```

Figure 5. Function to output menu

The third function is the input_menu_choice function that will get the menu_choice from the user. I started with @staticmethod def input_menu_choice(): like always. This function does not have a parameter so I left the inside of the parathesis empty. I set choice = “0”. I wrote a try loop and set choice equal to input(“Ener your menu choice number: “). I wrote an exception that if choice not it (“1”, “2”, “3”, “4”): raise Exception(“Please, choose only 1, 2, 3, or 4”). I wrote an Exception error as well that says except Exception as e: IO. Output_error_messages(e.__str__()). This will display any other exception errors. I ended this function with return choice that will allow the user to choose another menu choice. Figure 6 shows the function to input a menu choice.

```python
@staticmethod
def input_menu_choice():
    """ This function gets a menu choice from the user
    :return: string with the users choice
    """
    choice = "0"
    try:
        choice = input("Enter your menu choice number: ")
        if choice not in ("1","2","3","4"):  # Note these are strings
            raise Exception("Please, choose only 1, 2, 3, or 4")
    except Exception as e:
        IO.output_error_messages(e.__str__())  # Not passing the exception object to avoid the technical message

    return choice
```

Figure 6. Function to input menu choice

This next function will be used for the second menu choice that will output the student's data. I wrote @staticmethod def output_student_courses(): to start this function. I set the parameter to student_data. Then, I wrote a print function that prints nothing for an extra space up top. Then I printed("-"*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 printed f'Student {student_data["FirstName"]}' f'{student_data["LastName"]} is enrolled in {student_data["CourseName"]}'. In the f-string method, the name of variables are used as expressions inside curly-braces {} Since I was calling on different keys as well, I had to use [] square brackets. I then printed("-"*50) and another empty print function. Figure 7 shows the function to output student courses.

```
@staticmethod
def output_student_courses(student_data: list):
    '''Displays information regarding students
    :param student_data: The roster of students
    '''
    # Process the data to create and display a custom message
    print()
    print("-" * 50)
    for student_data in students:
        print(f'Student {student_data["FirstName"]} '
              f'{student_data["LastName"]} is enrolled in {student_data["CourseName"]}')
    print("-" * 50)
    print()
```

Figure 7. Function to output student courses

The last function will be used for the first menu choice to input a student and their data. In order to do this, I began by writing @staticmethod def input_student_data():. The parameter was set to student_data. 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 should not contain numbers.'). 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 = {‘FirstName’: student_first_name, ‘LastName’: student_last_name, ‘CourseName’: course_name}. I appended the student list into the student_data list by using append(). I ended this piece of code with another error handling that state except ValueError as e: IO.output_error_messages(“That value is not the correct type of data!”, e) and except Exception as e: IO.output_error_messages(“There was a non-specific error!”, e) before I returned student_data. Figure 8 shows the function to input student data.

```
@staticmethod
def input_student_data(student_data: list):
    '''
    Retrieves information required for each student including first name, last name, and course name
    :param student_data: The roster to add to
    :return: The roster
    '''
    try:
        # Input the data
        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 = {"FirstName": student_first_name,
                   "LastName": student_last_name,
                   "CourseName": course_name}
        student_data.append(student)

    except ValueError as e:
        IO.output_error_messages( message: "That value is not the correct type of data!", e)
    except Exception as e:
        IO.output_error_messages( message: "There was a non-specific error!", e)
    return student_data
```

Figure 8. Function to input student data

Main Code

Now that the classes and functions are done being written, I could write the main block of code that will call upon these different functions. I first created a variable called students and set it

equal to FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data= students). This is calling upon the read_data_from_file function I created early under the FileProcessor class. The parameters are file_name and student_data. Figure 9 shows the code written out. This block of code will begin as soon as the program funs and will create a read-only file called "Enrollments.json".

```
# Beginning of the main body of this script
students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)
```

Figure 9. Code to start reading data from file

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. In order to print out the menu, I called the output_menu function. Since it is stored within the IO class, I called IO.output_menu with the parameter of menu to present the menu choices. In order to input user data, I had to call on the input_menu_choice function stored within the IO class as well. I did this by setting menu_choice equal to IO.input_menu_choice(). Figure 10 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
    IO.output_menu(menu=MENU)
    # Input user data
    menu_choice = IO.input_menu_choice()
```

Figure 10. Code to present the menu and list of choices

Menu choice 1 should prompt the user for their first name, last name, and course name. Then it will store the inputs within the respective variables. I started with an if statement that says if the user chooses "1" for their menu choice. The next block of code will only happen if the user chooses menu option 1. I set students equal to IO.input_student_data() with the parameter of student_data. This is calling on the input_student_data function within the IO code. I also called IO.output_student_courses with the student_data parameter before I wrote continue. The code

will ask the student for the new data before displaying the data back to them. Figure 11 shows a picture of the full code for inputting data.

```
if menu_choice == "1":  # Get new data (and display the change)
    students = IO.input_student_data(student_data=students)
    IO.output_student_courses(student_data=students)
    continue
```

Figure 11. Code for inputting student data

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. I called on the output_student_course function from the IO class with student_data as the parameter before I wrote continue. This will run the output function to display the data to the user. Figure 12 shows what the full code will look like written together.

```
elif menu_choice == "2":  # Display current data
    IO.output_student_courses(student_data=students)  # Added this to improve user experience
    continue
```

Figure 12. Code for outputting data

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 block of code will only run if the user inputs 3 as their menu choice. I called on the write_data_to_file function from the FileProcessor class. I put two parameters of file_name and student_data before having the code continue. This will write all the students into the Enrollments.json file. Figure 13 portrays the code for file processing used for this program.

```
elif menu_choice == "3":  # Save data in a file
    FileProcessor.write_data_to_file(file_name=FILE_NAME, student_data=students)
    continue
```

Figure 13. Code for file 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 14 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 14. 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 15 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 table with all the students in their enrolled classes including my newly added student. Then 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.
----------------------------------------


Enter your menu choice number: 1
Enter the student's first name: Sam
Enter the student's last name: Cheese
Please enter the name of the course: Python100

------------------------------------------------
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Jessica Yun is enrolled in Python100
Student Sam Cheese is enrolled in Python100
------------------------------------------------
```

Figure 15. 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. It then displayed the menu again for me and redirected me to choose another option. Figure 16 shows option 2 being selected.

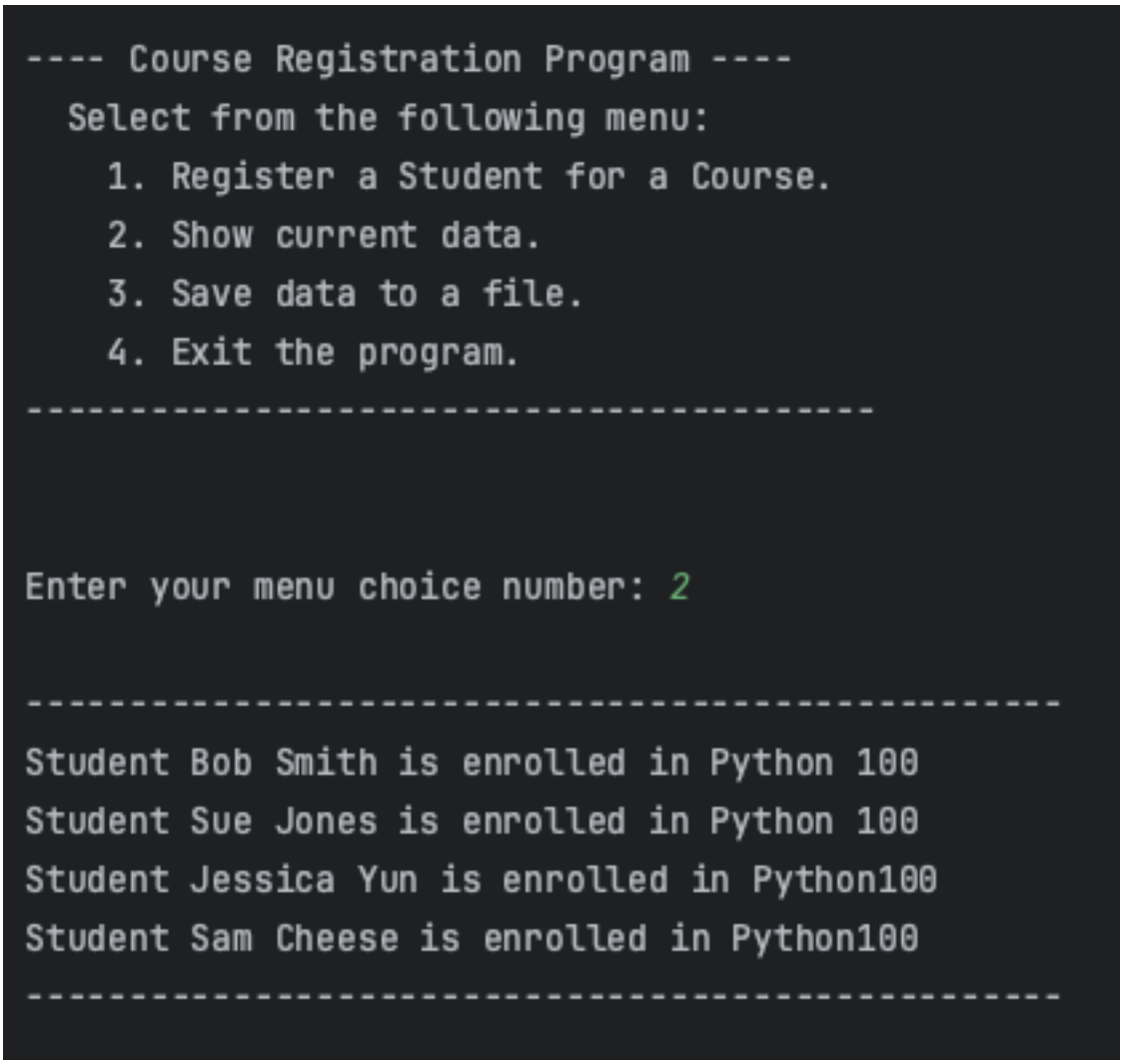

Figure 16. Option 2 selected on PyCharm

Below is option 3 being tested. When I input 3, all my inputted data got saved to the "Enrollments.json" file. Figure 17 shows option 3 being selected while figure 18 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.
----------------------------------------


Enter your menu choice number: 3
The following data was saved to file!
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Jessica Yun is enrolled in Python100
Student Sam Cheese is enrolled in Python100
```

Figure 17. Option 3 selected on PyCharm

Figure 18. 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 19 shows a picture of option 4 being chosen and the program ending.

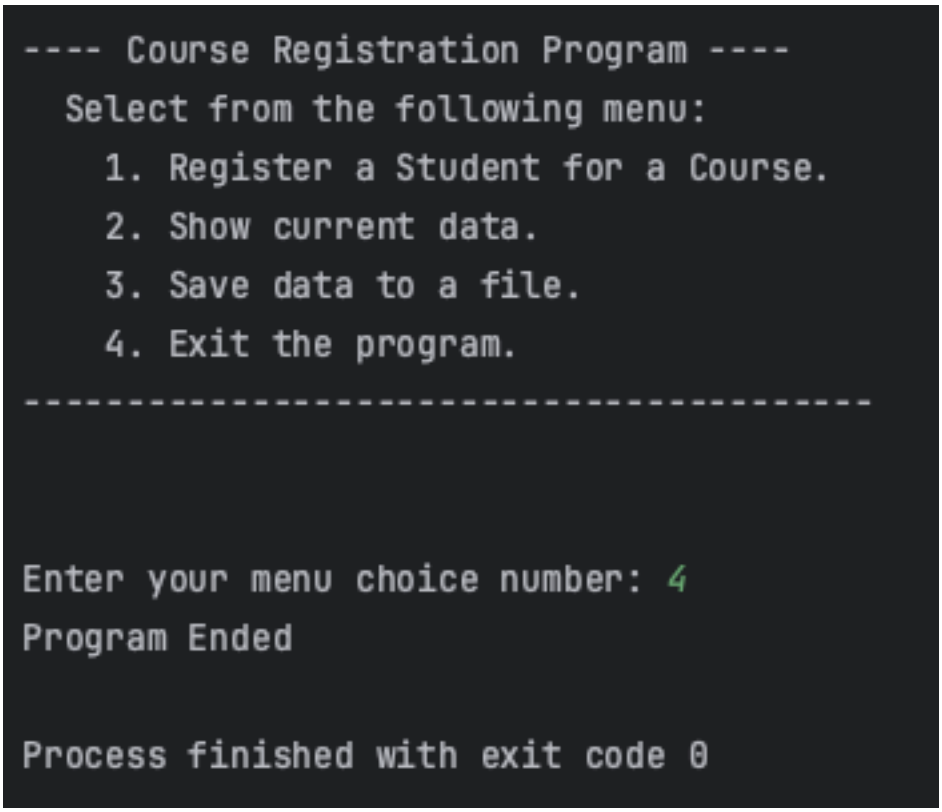

Figure 19. Option 4 selected on PyCharm

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 20 and 21 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. 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: Thu Nov  9 14:10:56 on ttys000
jessica@Jessicas-MacBook-Pro ~ % cd Desktop/A06
jessica@Jessicas-MacBook-Pro A06 % Python3 Assignment06.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.
----------------------------------------


Enter your menu choice number: 1
Enter the student's first name: Tim
Enter the student's last name: Smith
Please enter the name of the course: Python100

--------------------------------------------------
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Jessica Yun is enrolled in Python100
Student Sam Cheese is enrolled in Python100
Student Tim Smith 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.
----------------------------------------


Enter your menu choice number: 2

--------------------------------------------------
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Jessica Yun is enrolled in Python100
Student Sam Cheese is enrolled in Python100
Student Tim Smith 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.
----------------------------------------


Enter your menu choice number: 3
The following data was saved to file!
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Jessica Yun is enrolled in Python100
Student Sam Cheese is enrolled in Python100
Student Tim Smith 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.
----------------------------------------


Enter your menu choice number: 4
Program Ended
jessica@Jessicas-MacBook-Pro A06 %
```

Figure 20 and 21. 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. It was definitely harder than any of the past assignments however since creating classes and functions were completely new. Having to move all the code around into the correct places were especially challenging. This week's Lab Review videos were helpful in knowing how to begin 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. It was created using functions and classes.