Jessica Yun

November 27, 2023

ID FDN 110 A

Assignment 07

https://github.com/jssoyoung/IntoToProg-Python-Mod07

Classes

Introduction

This week, I dove deeper into learning about classes. 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 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:

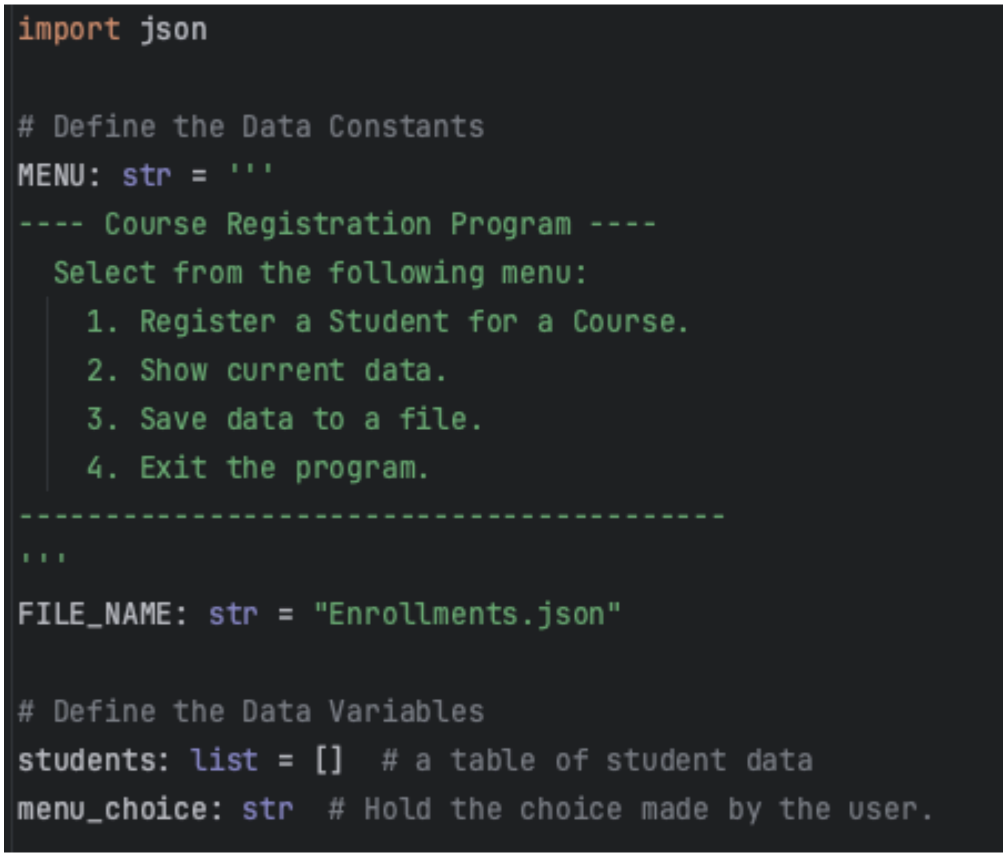
1. Register a Student for a Course

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: str = "". 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.

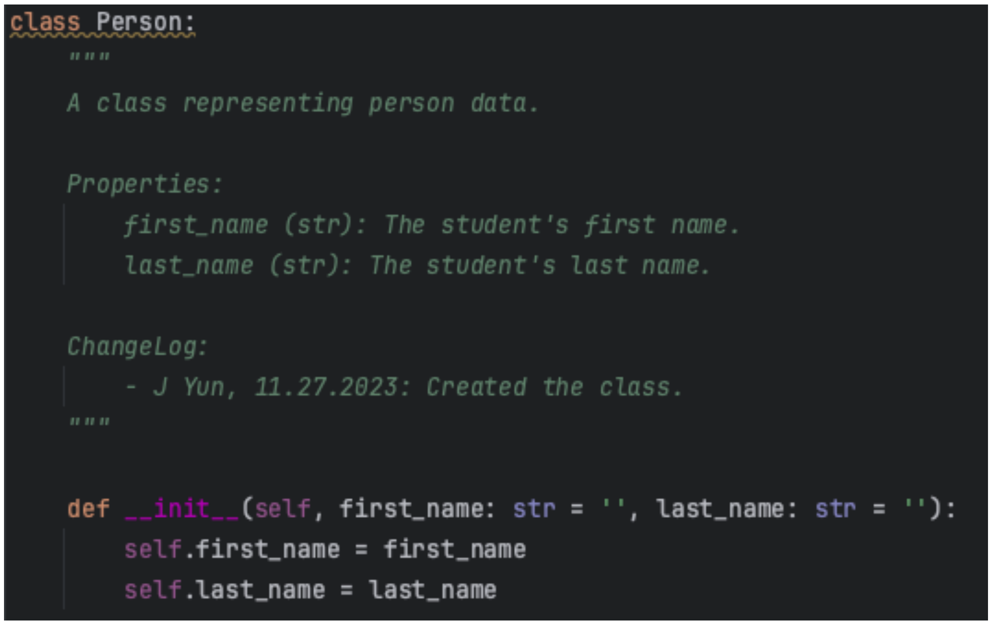


```python
import json

# 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"

# Define the Data Variables
students: list = []  # a table of student data
menu_choice: str  # Hold the choice made by the user.
```

Figure 1. Imports and data constants code

Classes

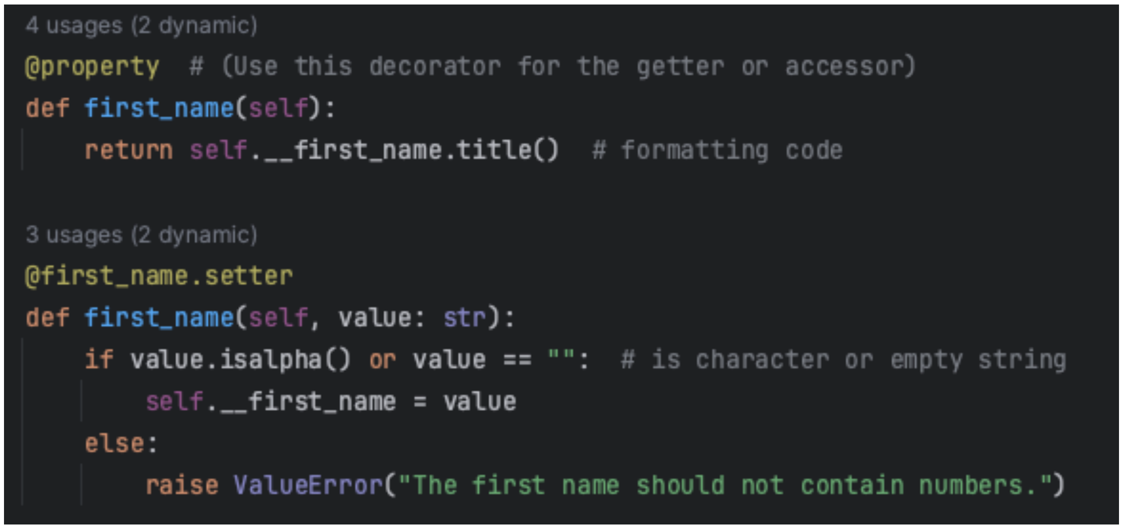
I created two classes for this program, one being a Person class and another being the student class. I created the Person class first by writing class Person:. For all of my different classes and functions, I wrote a comment which held information on what the class or function was doing, some of their properties and a change log on who edited the function/class and how it was changed. All the comments are written within three quotation marks. I started with a function called def __init__ within the person class. The parameters for this function were self, first_name and last_name. Within the function, I set self.first_name = first_name and self.last_name = last_name. Figure 2 shows the person class and the def__init__ function.



```python
class Person:
    """
    A class representing person data.

    Properties:
        first_name (str): The student's first name.
        last_name (str): The student's last name.

    ChangeLog:
        - J Yun, 11.27.2023: Created the class.
    """

    def __init__(self, first_name: str = '', last_name: str = ''):
        self.first_name = first_name
        self.last_name = last_name
```

Figure 2. Person Class

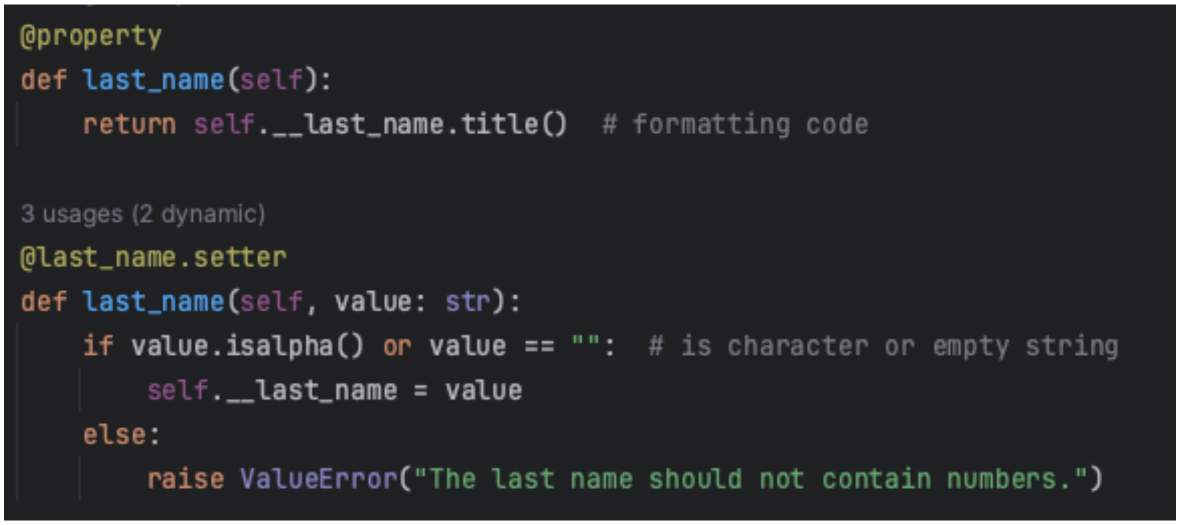
Next, I was made the getter and setter for first_name. For the getter function, I wrote @property def first_name with the parameter of self:. I returned self.__first_name.title(). For the setter function, I wrote @first_name.setter def first_name with the parameters of self and value. I put an if statement that if the value.isalpha or value == “” which is an empty string, then self.__first_name = value. Else: it would raise ValueError and pop a message saying “The first name should not contain numbers.” Figure 3 shows the first_name getter and setter code.



```
4 usages (2 dynamic)
@property  # (Use this decorator for the getter or accessor)
def first_name(self):
    return self.__first_name.title()  # formatting code

3 usages (2 dynamic)
@first_name.setter
def first_name(self, value: str):
    if value.isalpha() or value == "":  # is character or empty string
        self.__first_name = value
    else:
        raise ValueError("The first name should not contain numbers.")
```

Figure 3. first_name getter and setter code

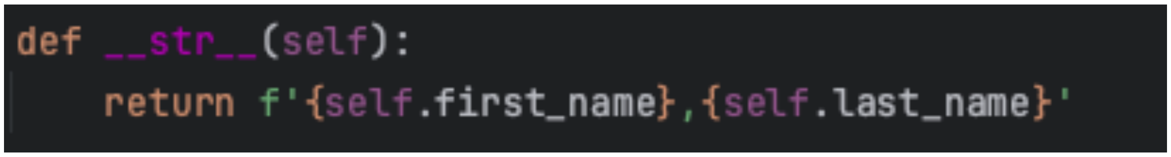
Now, I had to do the same thing but for last_name. I again, wrote the setter function by stating @property def last_name with the parameter of self. I returned self.__last_name.title(). For the setter function, I wrote @last_name.setter def last_name with the parameter of self and value. I wrote another if statement that if value.isalpha() or value == "" (an empty string), then self.last_name = value. Else: raise ValueError with the message "The last name should not contain numbers." Figure 4 shows the getter and setter code for last_name.



```
@property
def last_name(self):
    return self.__last_name.title()  # formatting code

3 usages (2 dynamic)
@last_name.setter
def last_name(self, value: str):
    if value.isalpha() or value == "":  # is character or empty string
        self.__last_name = value
    else:
        raise ValueError("The last name should not contain numbers.")
```

Figure 4. last_name getter and setter code

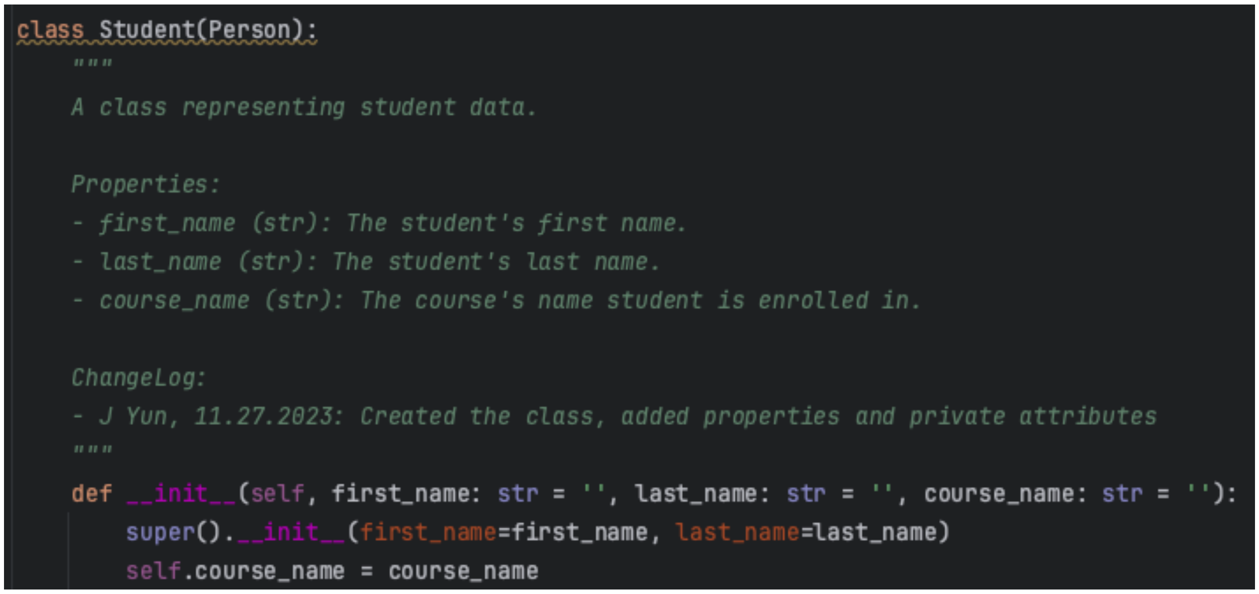
To override the __str__() method to return the Person data, I added another function called def __str__ with the parameter of self. I had it return a f-string that said f'{self.first_name}, {self.last}. Figure 5 shows the def __str__ code.



```python
def __str__(self):
    return f'{self.first_name},{self.last_name}'
```

Figure 5. def __str__ code

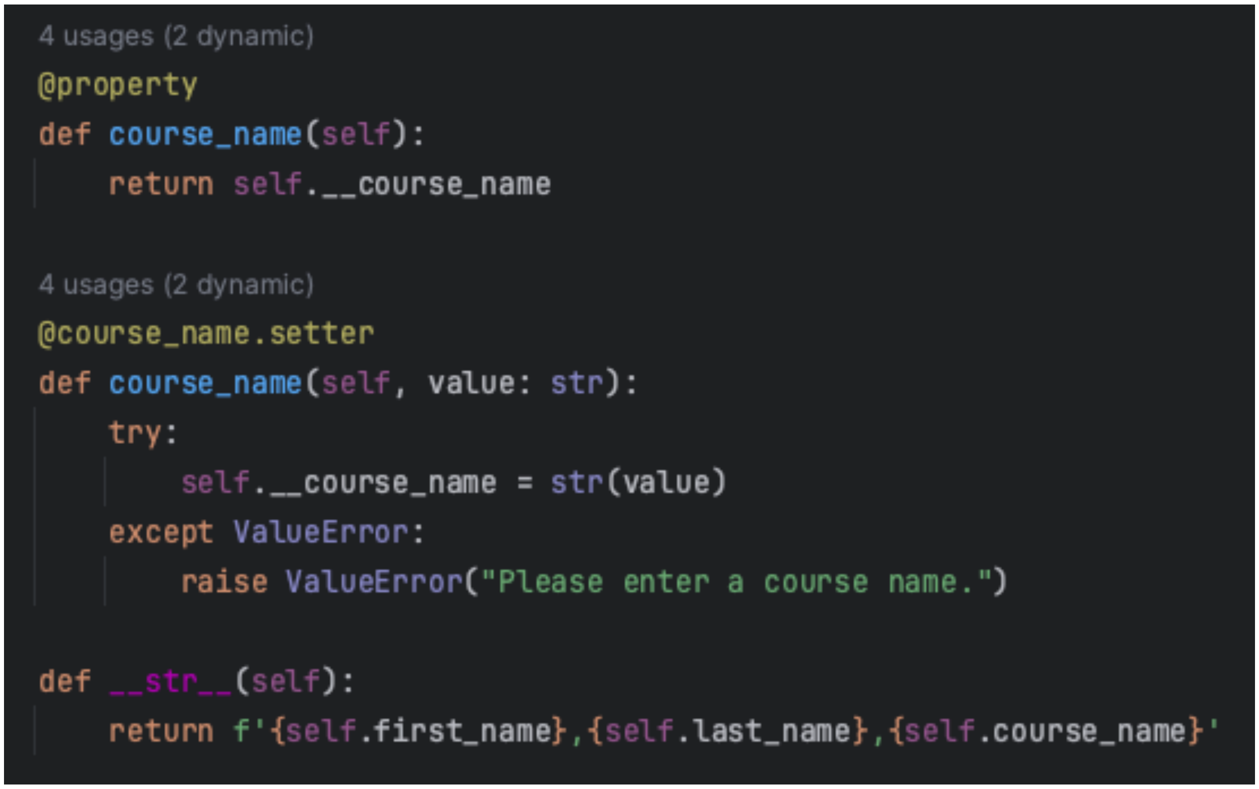
I then moved onto the Student class by writing class Student. I wrote Person within the paranthesis as the Student class has to inherit from the Person class. I wrote def __init__ with the parameters of self, first_name, last_name, and course_name. I used super() to call to the Person constructor and pass the first_name and last_name data. In order to do this, I wrote super().__init__(first_name=first_name, last_name=last_name). Then I set self.course_name equal to course_name as this was not a part of the Person class. Figure 6 shows the Student class being made with the def__init__ function.



```python
class Student(Person):
    """
    A class representing student data.

    Properties:
    - first_name (str): The student's first name.
    - last_name (str): The student's last name.
    - course_name (str): The course's name student is enrolled in.

    ChangeLog:
    - J Yun, 11.27.2023: Created the class, added properties and private attributes
    """
    def __init__(self, first_name: str = '', last_name: str = '', course_name: str = ''):
        super().__init__(first_name=first_name, last_name=last_name)
        self.course_name = course_name
```

Figure 6. Student Class

I had to create the getter and setter function for course_name just like I did for first_name and last_name. I did this by writing @property def course_name with the parameter of self. I returned self.__course_name to finish my getter function. For my setter function, I wrote @course_name.setter def course_name with the parameters of self and value. I wrote a try statement this time instead of an if statement. I wrote try: self.__course_name = str(value) except Value Error: raise ValueError. This states that when course_name isn't a string value, the error message will run. I wrote another def __str__ function to override the __str__ method to return the Student data with the parameter of self. I returned the f-string of {self.first_name}, {self.last_name}, and {self.course_name}. Figure 7 shows the code for the getter, setter and __str__ function.

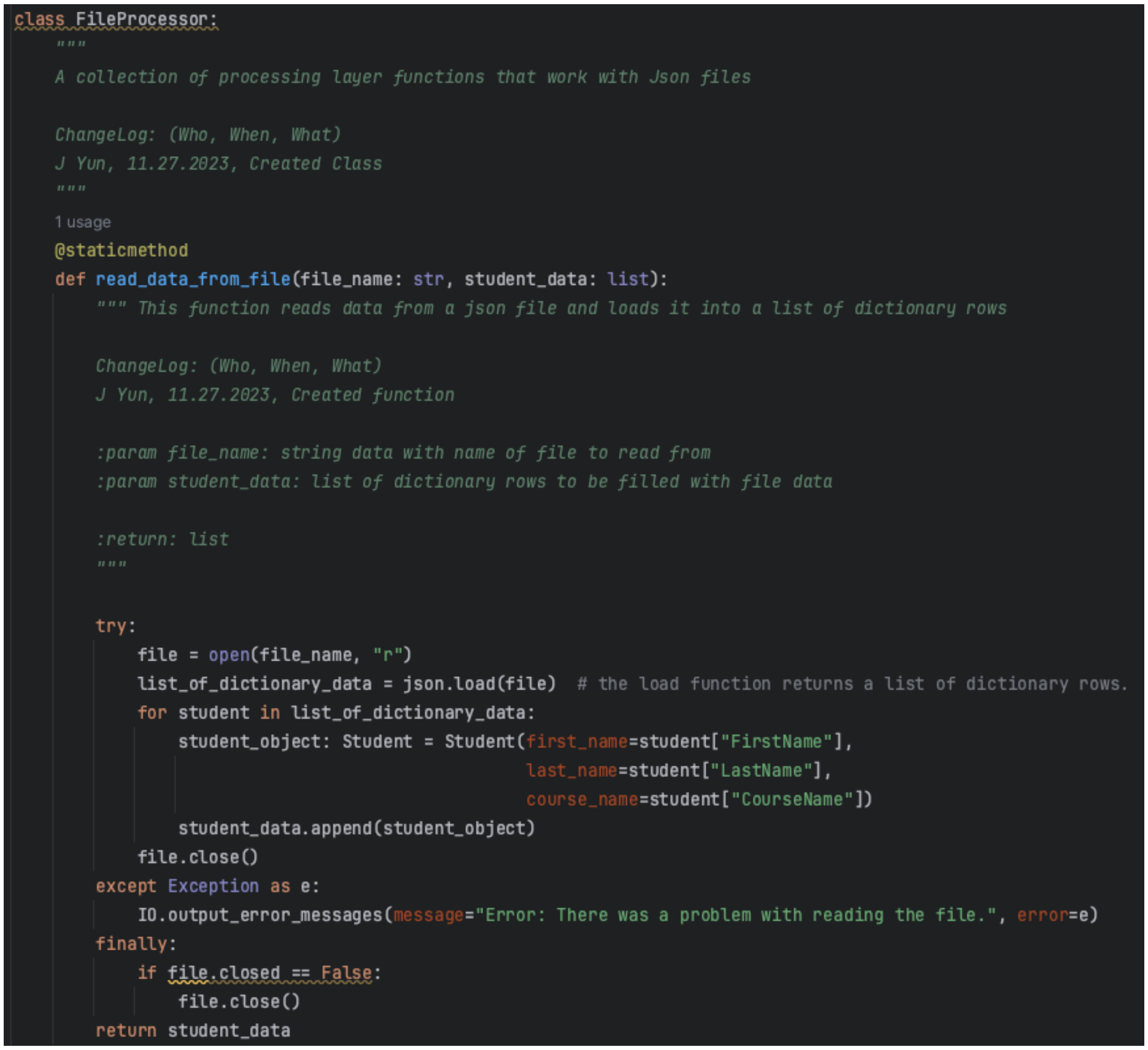


```
4 usages (2 dynamic)
@property
def course_name(self):
    return self.__course_name

4 usages (2 dynamic)
@course_name.setter
def course_name(self, value: str):
    try:
        self.__course_name = str(value)
    except ValueError:
        raise ValueError("Please enter a course name.")

def __str__(self):
    return f'{self.first_name},{self.last_name},{self.course_name}'
```

Figure 7. course_name getter and setter code

Processing

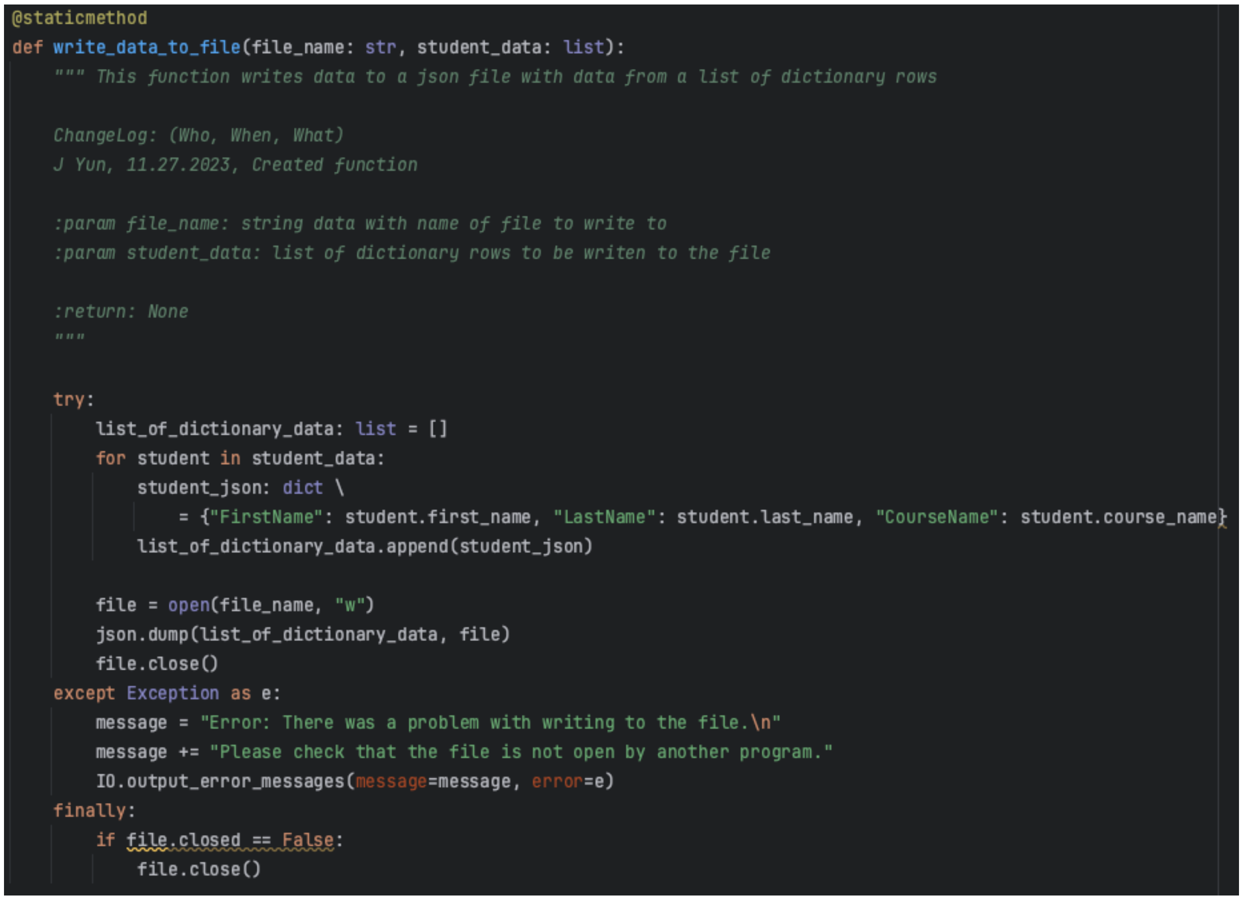
I had to create two more classes for this program that will store functions this time. I created one name called FileProcessor. This class will be used to store all of the functions for file processing. I started this block of code by writing class FileProcessor:. All classes must be written camel case 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. I wrote a little comment on what the function is doing, returning, their parameters, and a change log. 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"). The "r" stands for a read-only version of this file to be opened. I set list_of_dictionary_data equal to json.load(file) which makes the load function to return a list of dictionary rows. I wrote a for loop for student in list_of_dictionary_data: and set student_object: Student equal to Student(first_name = student['FirstName'], last_name = ['LastName'], and course_name = student['CourseName']). I appended the student_object into student_data. I then ended this portion of the code with a file.close(). I included some error handling within this portion in case there was any error in opening the file. The error handling states except Exception as e: IO.output_error_messages("Error: There was a problem with reading the file", 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 8 shows the code for this function.



```
class FileProcessor:
    """
    A collection of processing layer functions that work with Json files

    ChangeLog: (Who, When, What)
    J Yun, 11.27.2023, Created Class
    """
    1 usage
    @staticmethod
    def read_data_from_file(file_name: str, student_data: list):
        """ This function reads data from a json file and loads it into a list of dictionary rows

        ChangeLog: (Who, When, What)
        J Yun, 11.27.2023, Created function

        :param file_name: string data with name of file to read from
        :param student_data: list of dictionary rows to be filled with file data

        :return: list
        """

        try:
            file = open(file_name, "r")
            list_of_dictionary_data = json.load(file)  # the load function returns a list of dictionary rows.
            for student in list_of_dictionary_data:
                student_object: Student = Student(first_name=student["FirstName"],
                                                  last_name=student["LastName"],
                                                  course_name=student["CourseName"])
                student_data.append(student_object)
            file.close()
        except Exception as e:
            IO.output_error_messages(message="Error: There was a problem with reading the file.", error=e)
        finally:
            if file.closed == False:
                file.close()
        return student_data
```

Figure 8. 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. I began by writing list_of_dictionary_data: list = [] to set it open to an empty list. I wrote a for loop again for student in student_data:. Within the loop, I set student_json: dict \ equal to {'FirstName': student.first_name, 'LastName': student.last_name, 'CourseName': student.course_name. I appended student_json into the list_of_dictionary_data list. In order to get the program to open a file in write mode, I used file processing. I created 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(list_of_dictionary_data, file) and closed the file using the close() function. For my error handling, I wrote except Exception as e: IO.output_error_messages("Error: There was a problem with writing to the file") and ("Please check that the file is not open by another program"). I ended this piece of code with finally: if file.closed == False: file.close() to close all open files again. Figure 9 shows the function to write data to file which will be used later for menu option 3.

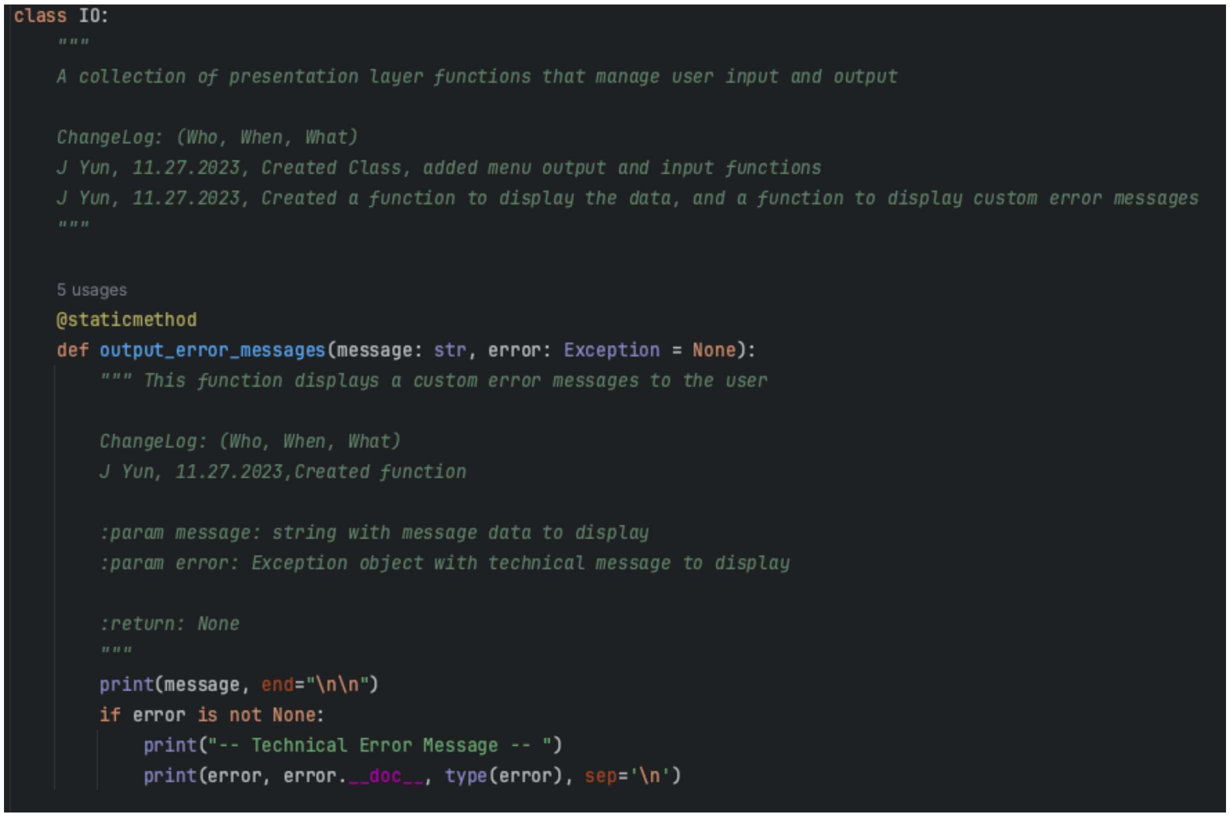


```python
@staticmethod
def write_data_to_file(file_name: str, student_data: list):
    """ This function writes data to a json file with data from a list of dictionary rows

    ChangeLog: (Who, When, What)
    J Yun, 11.27.2023, Created function

    :param file_name: string data with name of file to write to
    :param student_data: list of dictionary rows to be writen to the file

    :return: None
    """

    try:
        list_of_dictionary_data: list = []
        for student in student_data:
            student_json: dict \
                = {"FirstName": student.first_name, "LastName": student.last_name, "CourseName": student.course_name}
            list_of_dictionary_data.append(student_json)

        file = open(file_name, "w")
        json.dump(list_of_dictionary_data, file)
        file.close()
    except Exception as e:
        message = "Error: There was a problem with writing to the file.\n"
        message += "Please check that the file is not open by another program."
        IO.output_error_messages(message=message, error=e)
    finally:
        if file.closed == False:
            file.close()
```

Figure 9. Function to write data to file

Presentation

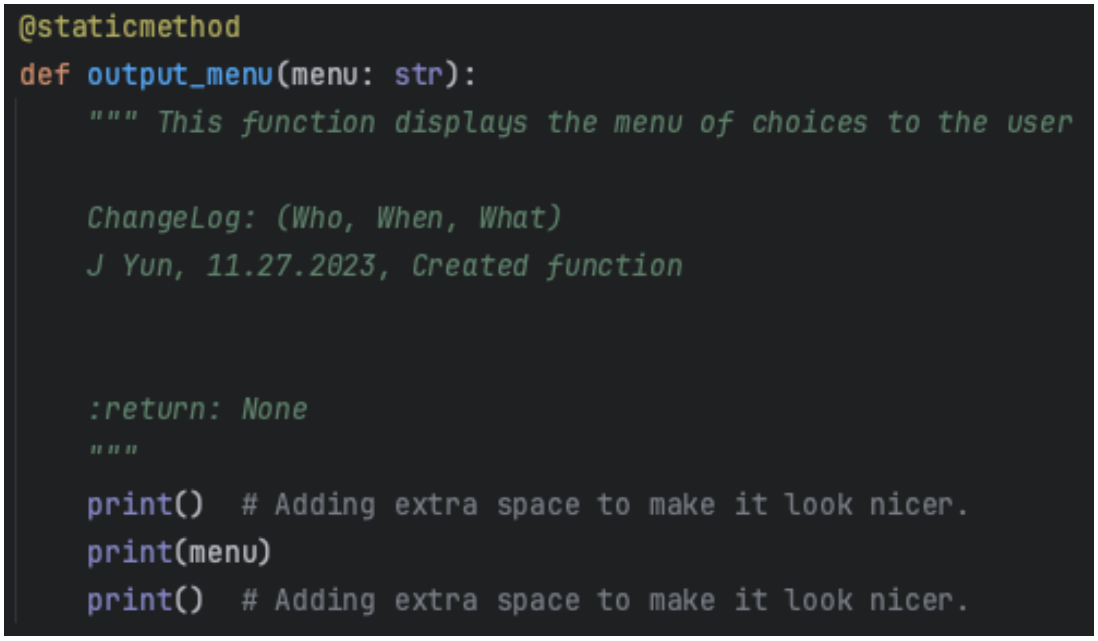
I then moved onto my second class which is class IO:. This class will hold all of the other functions in order to run this program. I wrote @static method def output_error_messages to have my error handling function. I set the parameters to message and error: Exception to None. I will print the error 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 10 shows the code for error handling.



```python
class IO:
    """
    A collection of presentation layer functions that manage user input and output

    ChangeLog: (Who, When, What)
    J Yun, 11.27.2023, Created Class, added menu output and input functions
    J Yun, 11.27.2023, Created a function to display the data, and a function to display custom error messages
    """

    5 usages
    @staticmethod
    def output_error_messages(message: str, error: Exception = None):
        """ This function displays a custom error messages to the user

        ChangeLog: (Who, When, What)
        J Yun, 11.27.2023,Created function

        :param message: string with message data to display
        :param error: Exception object with technical message to display

        :return: None
        """
        print(message, end="\n\n")
        if error is not None:
            print("-- Technical Error Message -- ")
            print(error, error.__doc__, type(error), sep='\n')
```

Figure 10. Function to output error messages

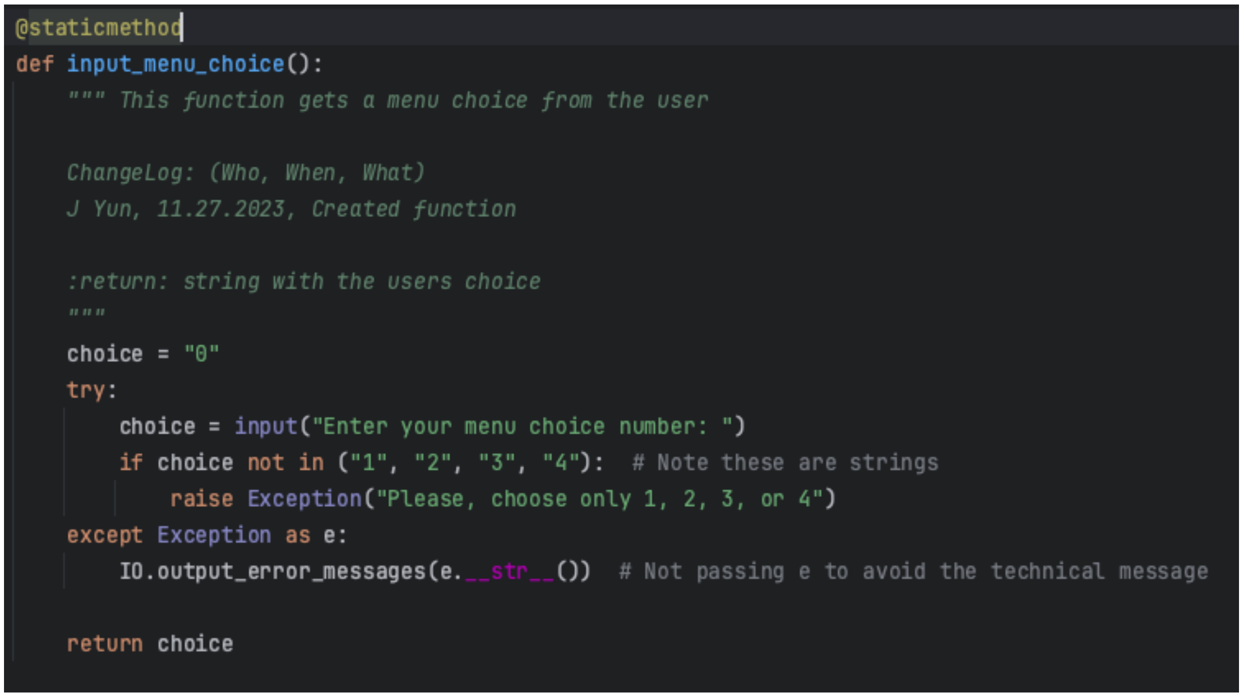
The next function will be used to display the menu. I started with my @staticmethod def output_menu():. The only parameter for this function is menu which will be a string. Within the function I wrote print(), print(menu), print(). The extra print functions were used just to add extra space to make the menu appear nicer for the user when displayed. Figure 11 displays the output_menu function.



```
@staticmethod
def output_menu(menu: str):
    """ This function displays the menu of choices to the user

    ChangeLog: (Who, When, What)
    J Yun, 11.27.2023, Created function


    :return: None
    """
    print()  # Adding extra space to make it look nicer.
    print(menu)
    print()  # Adding extra space to make it look nicer.
```

Figure 11. Function to output menu

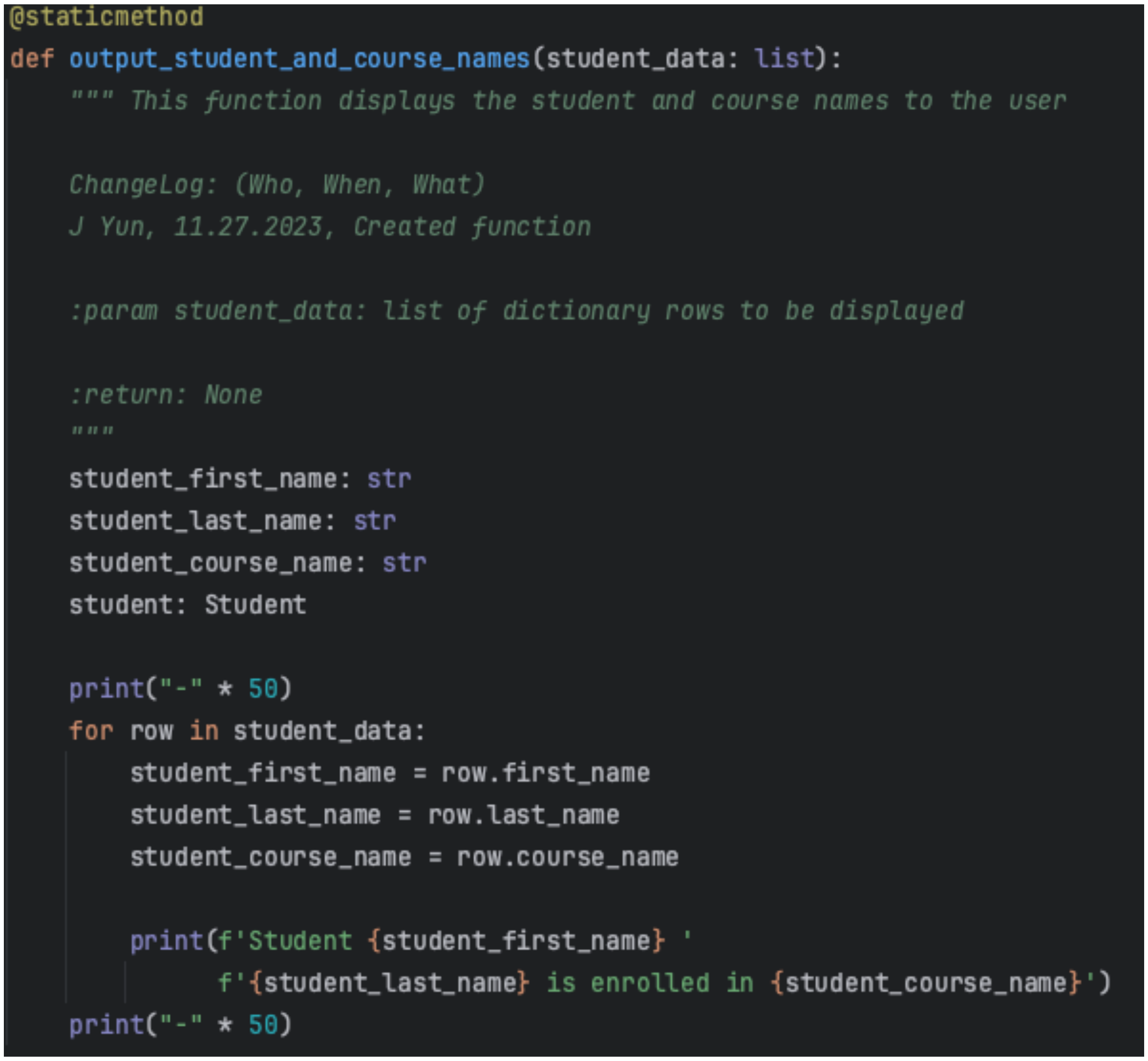
The third function is the input_menu_choice function that will get the menu_choice from the user. I started with @staticmethod def input_menu_choice(): like always. This function does not have a parameter so I left the inside of the parathesis empty. I set choice = “0”. I wrote a try loop and set choice equal to input(“Enter your menu choice number: “). I wrote an exception that if choice not in (“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 12 shows the function to input a menu choice.



```python
@staticmethod
def input_menu_choice():
    """ This function gets a menu choice from the user

    ChangeLog: (Who, When, What)
    J Yun, 11.27.2023, Created function

    :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 e to avoid the technical message

    return choice
```

Figure 12. Function to input menu choice

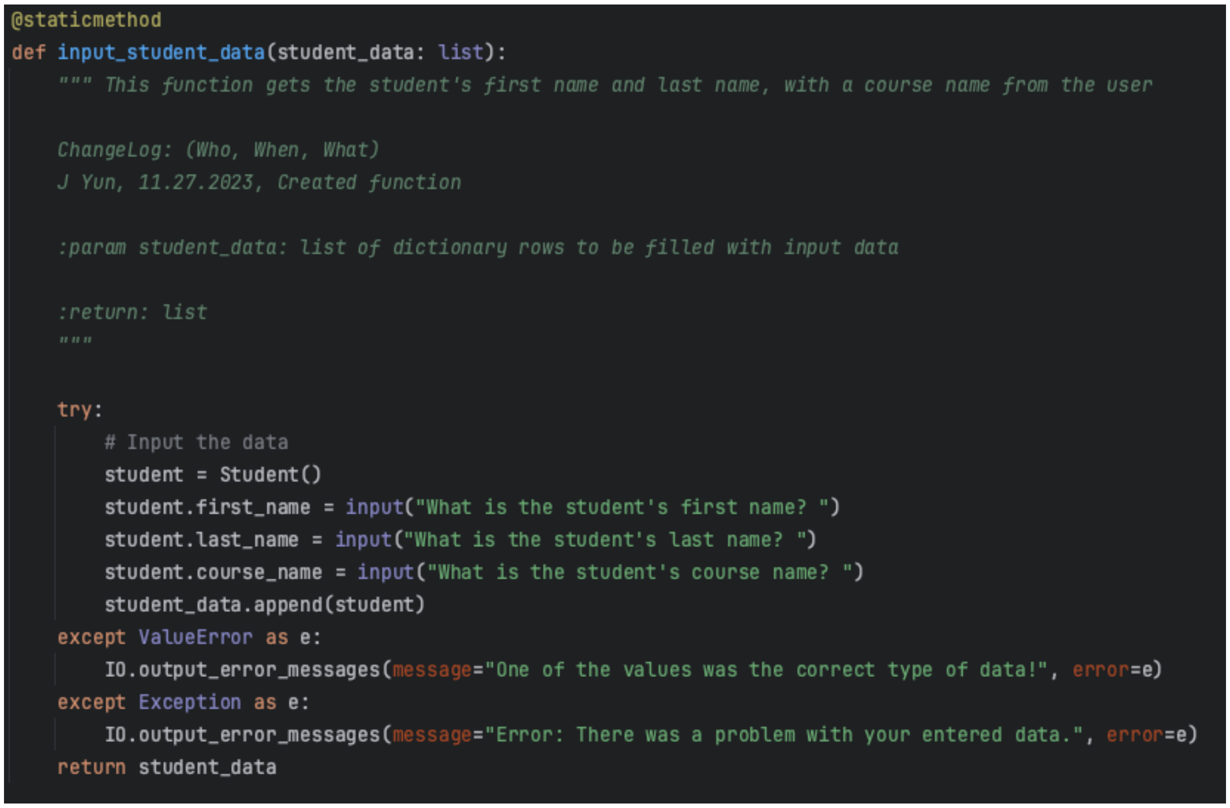
This next function will be used for the second menu choice that will output the student's data. I wrote @staticmethod def output_student_and_course_names(): to start this function. I began this function by setting a few variables to str. I wrote student_first_name: str, student_last_name: str, student_course_name: str, and student: Student. Then I printed("-"*50) followed by a for loop. The for loop is for row in student_data:. I set student_first_name equal to row.first_name, student_last_name equal to row.last_name, and student_course_name equal to row.course_name. I printed f'Student {student_first_name}' f'{student_last_name} is enrolled in {student_course_name}'. In the f-string method, the name of variables are used as expressions inside curly-braces {}. I then printed("-"*50) again. Figure 13 shows the function to output student courses.



```python
@staticmethod
def output_student_and_course_names(student_data: list):
    """ This function displays the student and course names to the user

    ChangeLog: (Who, When, What)
    J Yun, 11.27.2023, Created function

    :param student_data: list of dictionary rows to be displayed

    :return: None
    """
    student_first_name: str
    student_last_name: str
    student_course_name: str
    student: Student

    print("-" * 50)
    for row in student_data:
        student_first_name = row.first_name
        student_last_name = row.last_name
        student_course_name = row.course_name

        print(f'Student {student_first_name} '
              f'{student_last_name} is enrolled in {student_course_name}')
    print("-" * 50)
```

Figure 13. Function to output student and 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 and set it equal to Student(). I created another variable called student.first_name. I set this variable equal to input("What is 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. On the next line, I created wrote the same piece of code but this time asking "What is the student's last name?" and stored it to a variable called student.last_name. The last input was student.course_name = input("What is the student's 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 states except ValueError as e: IO.output_error_messages("One of the values was the correct type of data!", e) and except Exception as e: IO.output_error_messages("Error: There was a problem with your entered data.", e) before I returned student_data. Figure 14 shows the function to input student data.

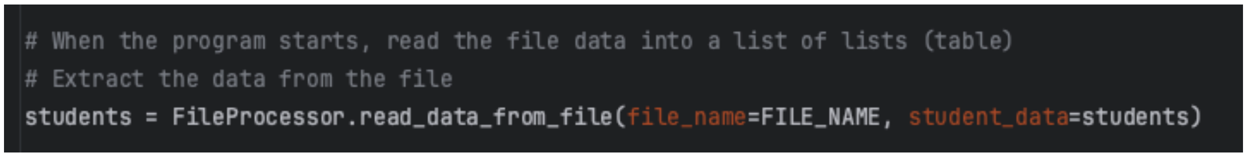


```python
@staticmethod
def input_student_data(student_data: list):
    """ This function gets the student's first name and last name, with a course name from the user

    ChangeLog: (Who, When, What)
    J Yun, 11.27.2023, Created function

    :param student_data: list of dictionary rows to be filled with input data

    :return: list
    """

    try:
        # Input the data
        student = Student()
        student.first_name = input("What is the student's first name? ")
        student.last_name = input("What is the student's last name? ")
        student.course_name = input("What is the student's course name? ")
        student_data.append(student)
    except ValueError as e:
        IO.output_error_messages(message="One of the values was the correct type of data!", error=e)
    except Exception as e:
        IO.output_error_messages(message="Error: There was a problem with your entered data.", error=e)
    return student_data
```

Figure 14. Function to input student data

Main Code

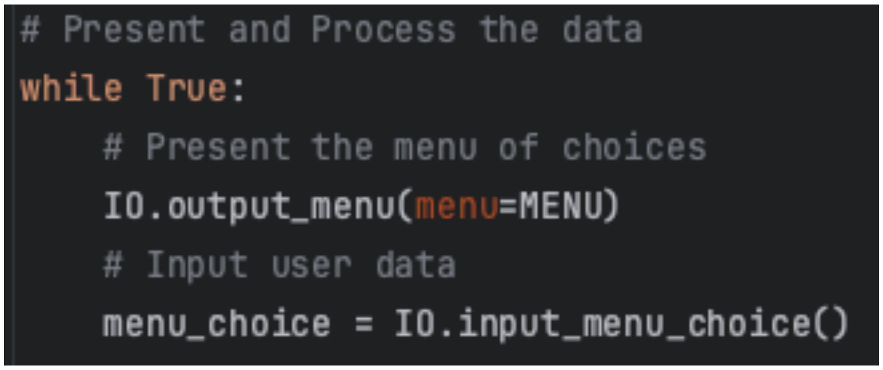
Now that the classes and functions are done being written, I could write the main block of code that will call upon these different functions. I first created a variable called students and set it equal to FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data= students). This is calling upon the read_data_from_file function I created early under the FileProcessor class. The parameters are file_name and student_data. Figure 15 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".



```
# When the program starts, read the file data into a list of lists (table)
# Extract the data from the file
students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)
```

Figure 15. Code to start reading data from file

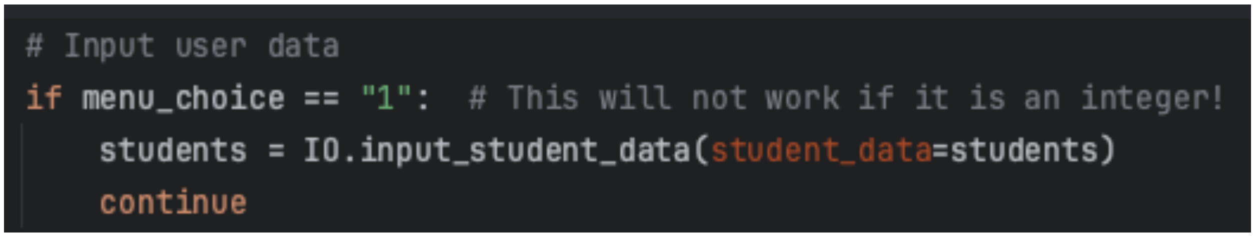
The program needs to ask the user to choose a menu option. I began with a while loop that said while (TRUE):. The while True: allows the menu options to continuously loop and run until the user chooses option 4 which will contain a break and end the loop. In order to print out the menu, I called the 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 16 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 16. Code to present the menu and list of choices

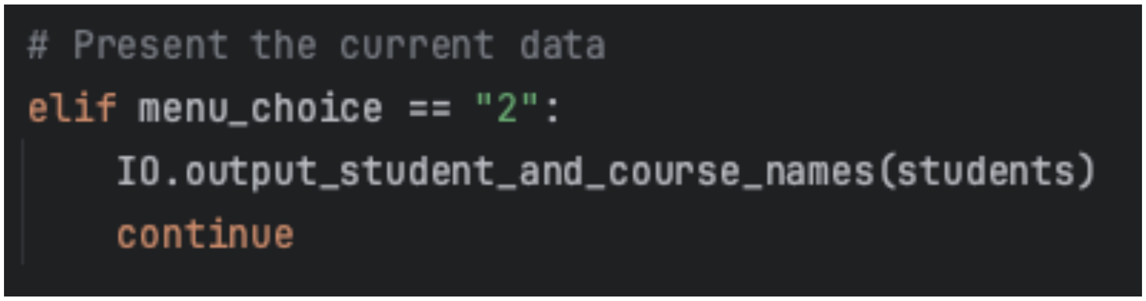
Menu choice 1 should prompt the user for their first name, last name, and course name. Then it will store the inputs within the respective variables. I started with an if statement that says if the user chooses “1” for their menu choice. The next block of code will only happen if the user chooses menu 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. Figure 17 shows a picture of the full code for inputting data.



```
# Input user data
if menu_choice == "1":  # This will not work if it is an integer!
    students = IO.input_student_data(student_data=students)
    continue
```

Figure 17. Code for inputting student data

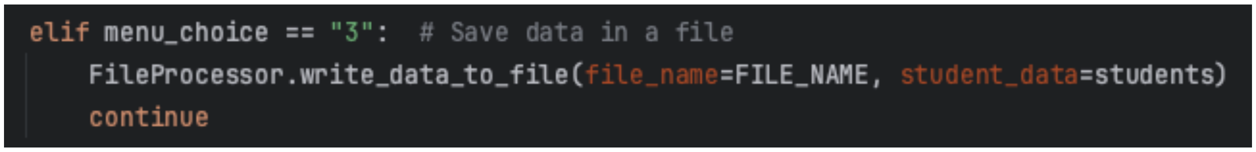
Menu choice 2 will present a coma-separated string by formatting the collected data until the print() function. I started by adding to my if statement and created an elif (else if) statement. I wrote elif menu_choice == “2”:. This block of code will only run if the user inputs 2 as their menu choice. I called on the output_student_and_course_names 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 18 shows what the full code will look like written together.



```
# Present the current data
elif menu_choice == "2":
    IO.output_student_and_course_names(students)
    continue
```

Figure 18. Code for outputting data

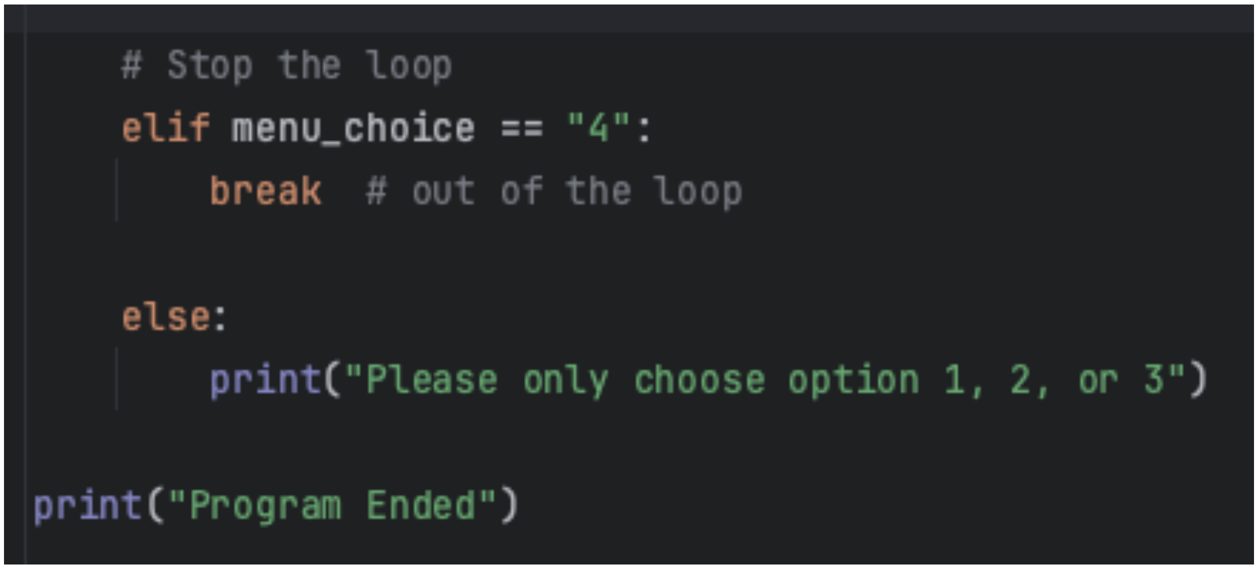
The third menu choice opens a file named “Enrollments.json”. I began this portion of the code with another elif statement and wrote elif menu_choice == “3”:. This block of code will only run if the user inputs 3 as their menu choice. I called on the write_data_to_file function from the FileProcessor 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 19 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 19. 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 20 shows the code to stop the loop.



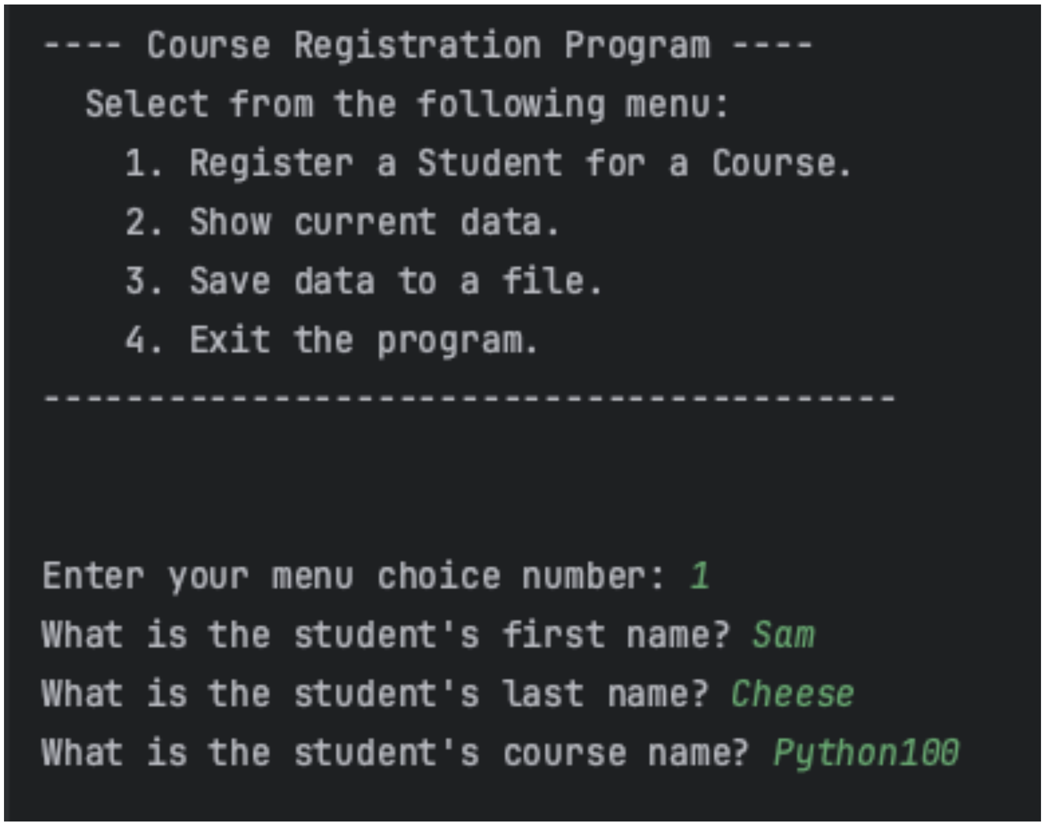
```
    # Stop the loop
    elif menu_choice == "4":
        break  # out of the loop

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

print("Program Ended")
```

Figure 20. 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.

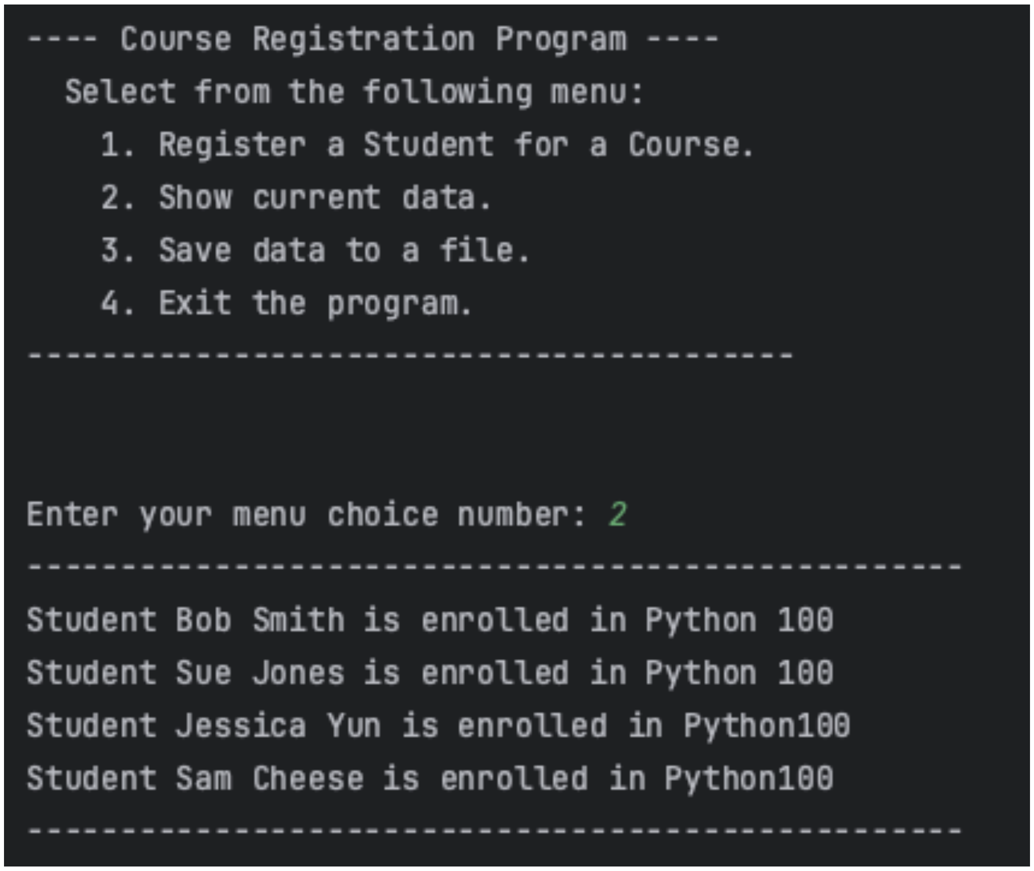
I first ran the program using PyCharm. Since this program has multiple different options for the user to choose, I tested the program four different times, once for each option. Figure 21 shows the program running through PyCharm and selecting option 1. The program asked me for my first name, last name, and course name. 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
What is the student's first name? Sam
What is the student's last name? Cheese
What is the student's course name? Python100
```

Figure 21. Option 1 selected on PyCharm

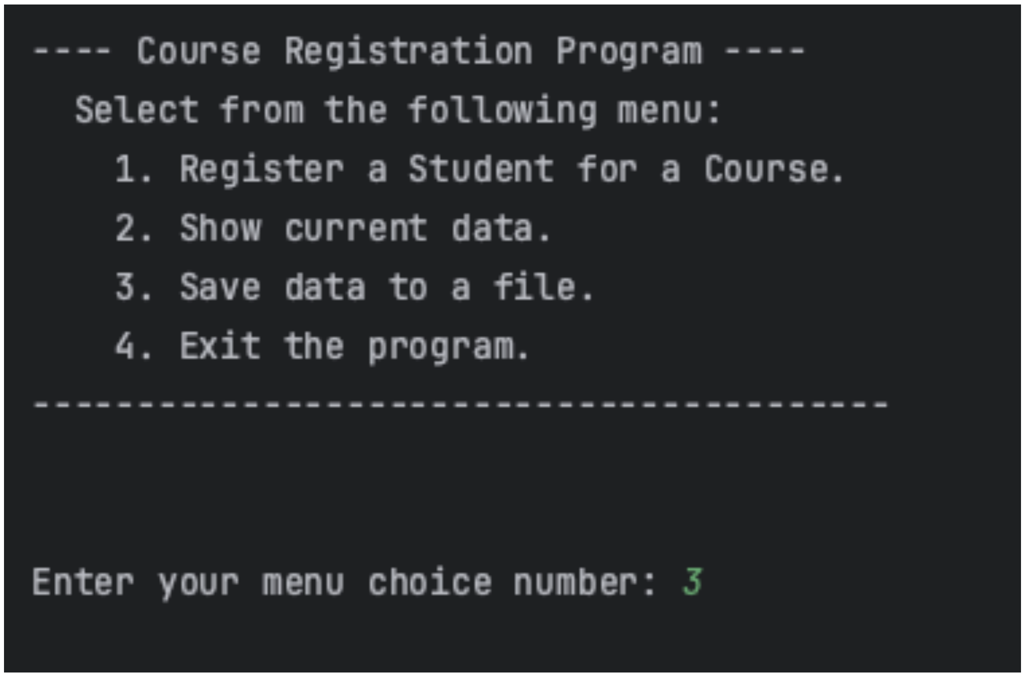
I proceeded to test option 2. When I input 2, the program displayed all the first names, last names, and course names. It then displayed the menu again for me and redirected me to choose another option. Figure 22 shows option 2 being selected.



```
---- 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.
-----------------------------------------



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
--------------------------------------------------
```

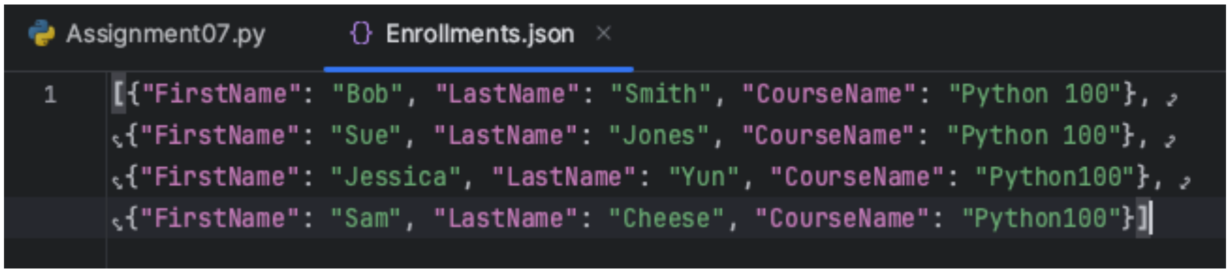
Figure 22. Option 2 selected on PyCharm

Below is option 3 being tested. When I input 3, all my inputted data got saved to the "Enrollments.json" file. Figure 23 shows option 3 being selected while figure 24 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
```

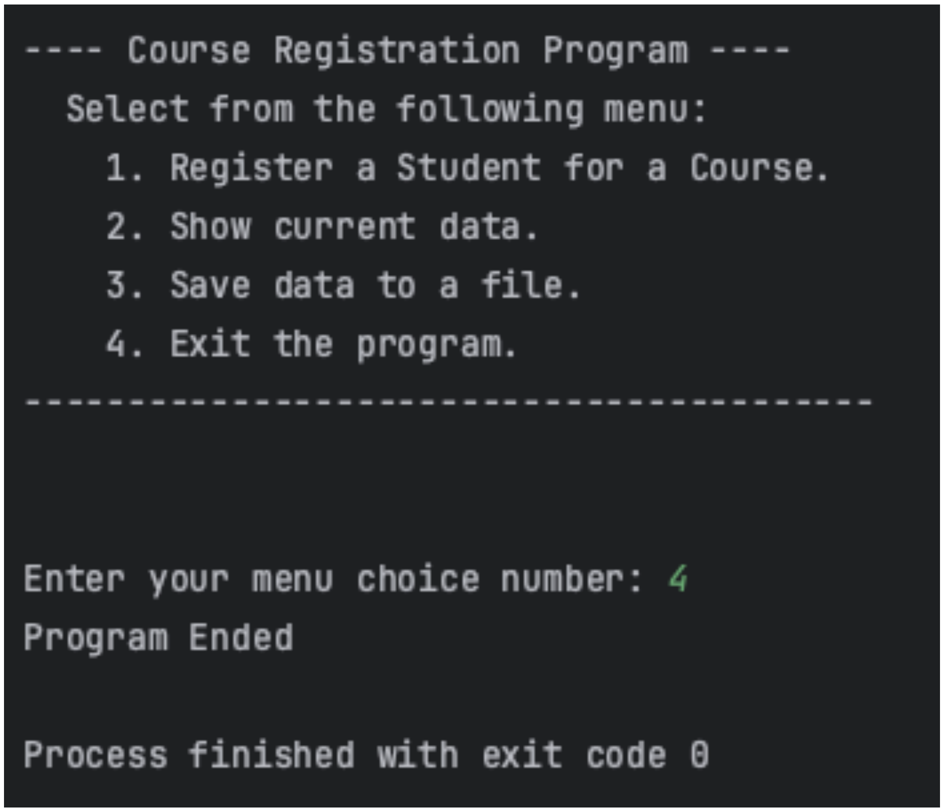
Figure 23. Option 3 selected on PyCharm



```
[{"FirstName": "Bob", "LastName": "Smith", "CourseName": "Python 100"}, 
{"FirstName": "Sue", "LastName": "Jones", "CourseName": "Python 100"}, 
{"FirstName": "Jessica", "LastName": "Yun", "CourseName": "Python100"}, 
{"FirstName": "Sam", "LastName": "Cheese", "CourseName": "Python100"}]
```

Figure 24. Data written into the Enrollments.csv file

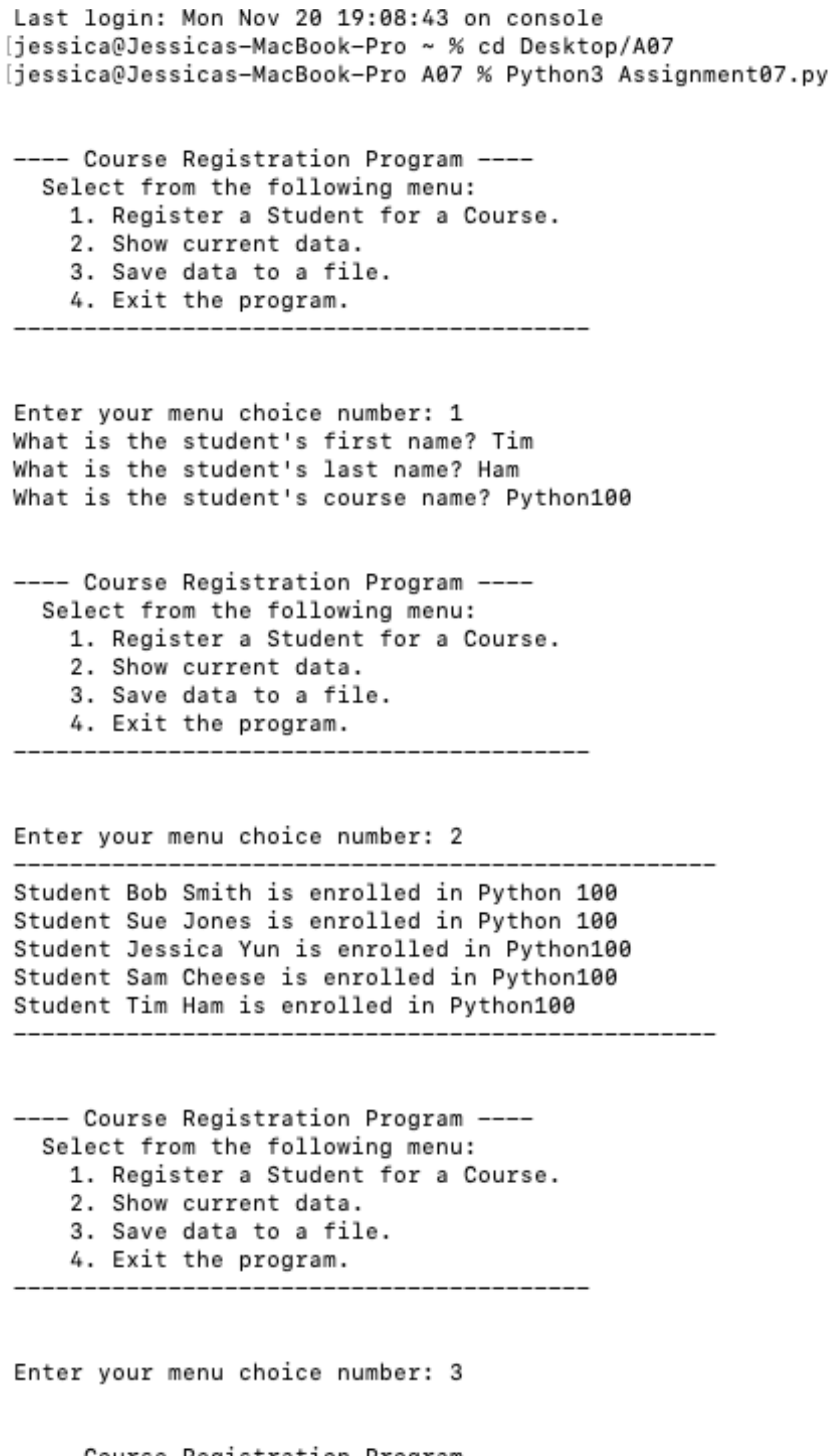
Finally, I tested option 4. When I input the number 4, I got a printed message that said the program ended and a message that the process was finished. The menu option did not appear for this option which means the loop stopped. Figure 25 shows a picture of option 4 being chosen and the program ending.



```
---- 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.
-----------------------------------------



Enter your menu choice number: 4
Program Ended

Process finished with exit code 0
```

Figure 25. 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 26 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 wrote my data to the Enrollments.json file on my computer. Option 4 ended the program and stopped the loop.



```
Last login: Mon Nov 20 19:08:43 on console
[jessica@Jessicas-MacBook-Pro ~ % cd Desktop/A07
[jessica@Jessicas-MacBook-Pro A07 % Python3 Assignment07.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
What is the student's first name? Tim
What is the student's last name? Ham
What is the student's course name? 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 Ham 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


     Course Registration Program
```

Figure 26. Testing using terminal

Summary

These were the steps taken to create this program. For this program, I had to create lots of classes and functions. It got a little confusing with all the different functions and classes and knowing where to put all the different pieces of information. 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.