Ogonna Anunoby

09/23/2024

IT FDN 110 B Su 24: Foundations of Programming: Python

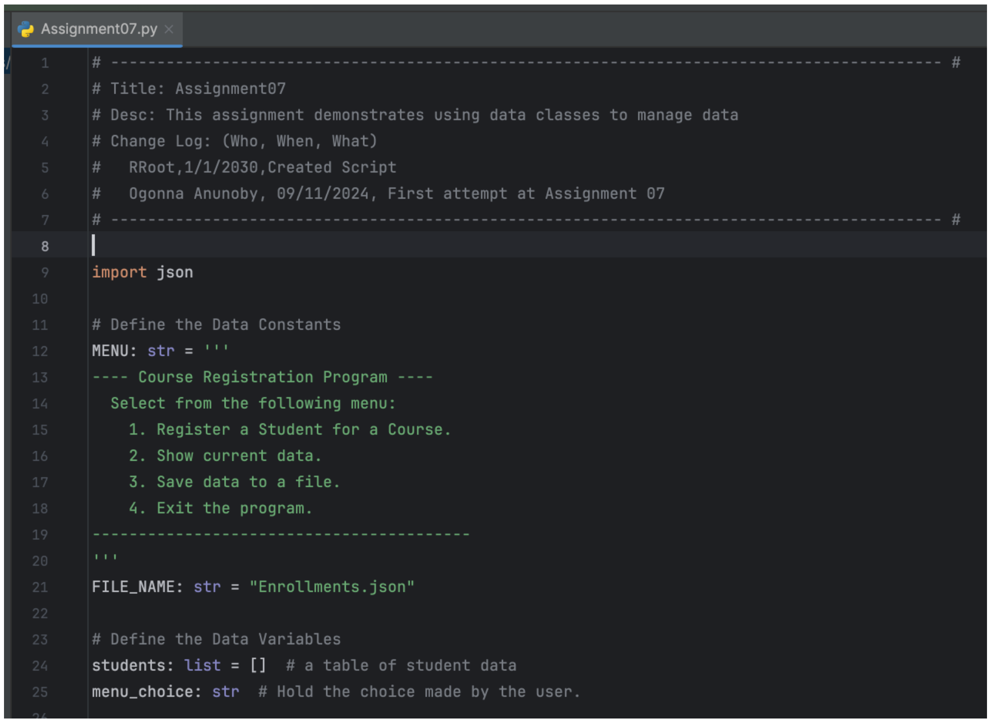
Assignment 07

Assignment 07 Report

Introduction

In Assignment 07, I wrote a script that registers students for various courses. This assignment allowed me to use principles of object-oriented programming such as constructors, properties, inheritance, and method overriding. This assignment allowed me to continue developing my Python programming skills.

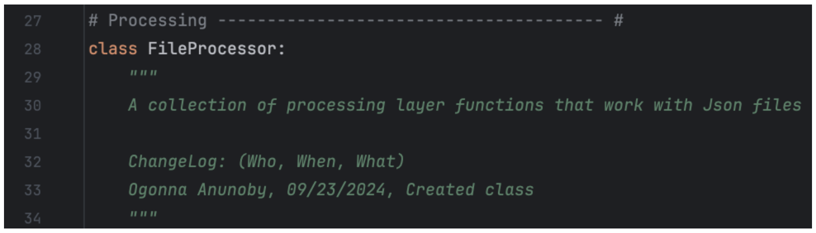
Script Testing



```python
# ------------------------------------------------------------------------------------------ #
# Title: Assignment07
# Desc: This assignment demonstrates using data classes to manage data
# Change Log: (Who, When, What)
#   RRoot,1/1/2030,Created Script
#   Ogonna Anunoby, 09/11/2024, First attempt at Assignment 07
# ------------------------------------------------------------------------------------------ #

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: Script Header, Constants, and Variables for Assignment 07 Script Code

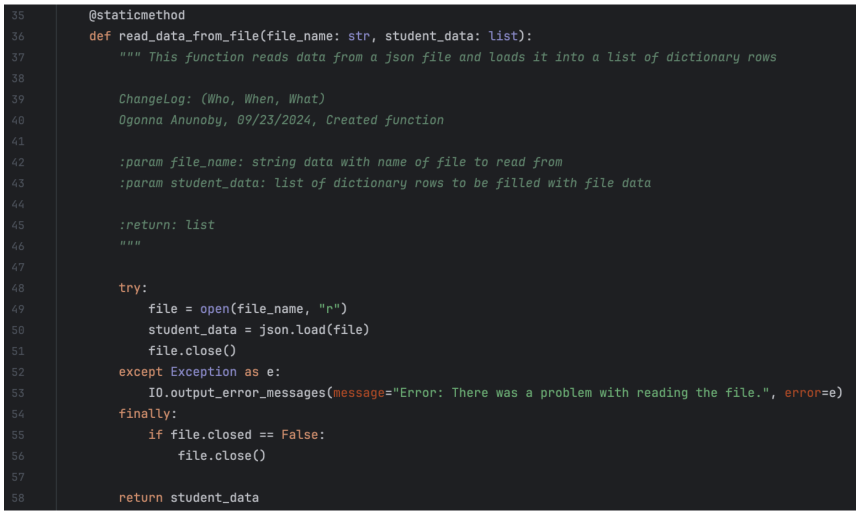
Figure 1 shows the header and data constants for the Assignment 07 script code. Lines 1 through 8 consist of the script header. The change log has one entry, showing that I made my first attempt of Assignment 07 on 09/11/2024. Line 9 shows that I imported the Json library to allow for use of its library function when processing Json data. Lines 11 through 21 are where I have defined my data constants. In this script, there are two data constants. MENU is a string that displays the registration menu to the user. FILE_NAME is a string that holds the name of the file. Enrollments.json is the name of this file, which will save registration data. Lines 23 through 25 show the data variables for this assignment. The variable students is initialized to an empty list. The variable menu_choice is initialized to an empty string.



```
27  # Processing -------------------------------------- #
28  class FileProcessor:
29      """
30      A collection of processing layer functions that work with Json files
31
32      ChangeLog: (Who, When, What)
33      Ogonna Anunoby, 09/23/2024, Created class
34      """
```

Figure 2: FileProcessor class and its docstring

Figure 2 shows the FileProcessor class and its docstring. A class is blueprint that allows for the grouping of functions, variables, and constants. Instances of classes are called objects, which can be created and used to handle data. This class deals with the data storage concern in the separation of concerns design principle. This class has methods that deal the management, processing, and storage of data in the Json file. The FileProcessor class runs from lines 28 through 84. Lines 29 through 34 show the docstring for this class. This code serves as developer notes and is enclosed in three quotation marks. Here the docstring explains that this class has methods to handle Json file processing. It also shows the change log for this class.

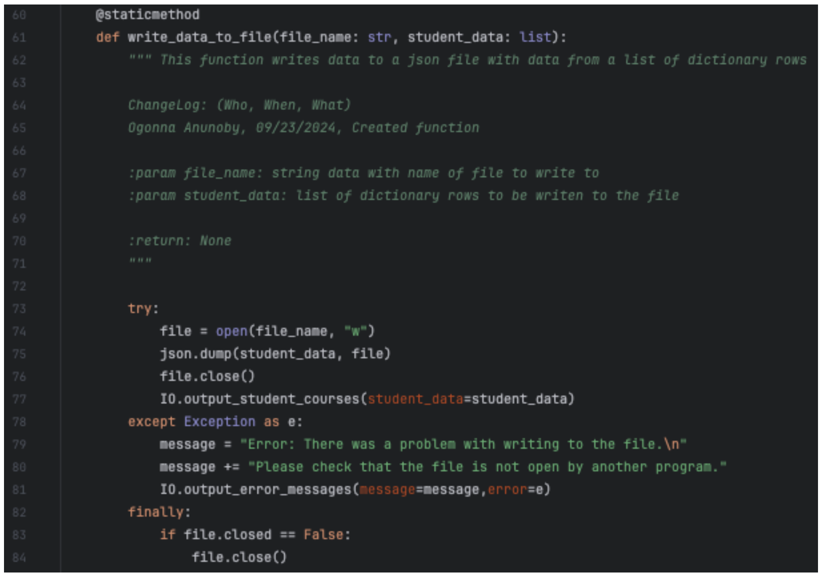


```
35      @staticmethod
36      def read_data_from_file(file_name: str, student_data: list):
37          """ This function reads data from a json file and loads it into a list of dictionary rows
38
39          ChangeLog: (Who, When, What)
40          Ogonna Anunoby, 09/23/2024, Created function
41
42          :param file_name: string data with name of file to read from
43          :param student_data: list of dictionary rows to be filled with file data
44
45          :return: list
46          """
47
48          try:
49              file = open(file_name, "r")
50              student_data = json.load(file)
51              file.close()
52          except Exception as e:
53              IO.output_error_messages(message="Error: There was a problem with reading the file.", error=e)
54          finally:
55              if file.closed == False:
56                  file.close()
57
58          return student_data
```

Figure 3: The read_data_from_file() method

Figure 3 shows the read_data_from_file() method. A method is a function that belongs to a class. Meanwhile, a function is a reusable block of code that performs a specific task or a set of tasks. Line 35 uses the @staticmethod decorator to indicate that this method belongs to the class. So, a FileProcessor object does not need to be created in to use the method. Line 36 has the method definition. The “def” keyword is used to define the method. Inside the parentheses are the arguments that the method requires. Here we have a string which will replace the file_name variable when passed to the method. We also have student_data as a parameter of type list. This will replace student_data in this method. Everything indented after line 36 is included in this method. So, the method runs from lines 35 through 58 (including the @staticmethod decorator). Lines 37 through 46 shows the docstring for this method. Here, there is a general description of the method and a change log. There is also a description of the parameters and return value. This method reads data from the Json file into the student_data list.

Lines 48 through 56 show the try-except-except-finally block that is used to process reading data from the file. In the try block, the code attempts to open the file in read mode, load the Json data into the student_data dictionary, then close the file. If the try block fails, a general error exception will be thrown, and the code will go to line 58. Here, the output_error_messages() method belonging to the IO class will be called, with an error message stating that reading the file failed and also the error object are passed to it. In the finally block, which will always be called, if the file is still open, it will be closed. On line 58, the student_data list is returned.

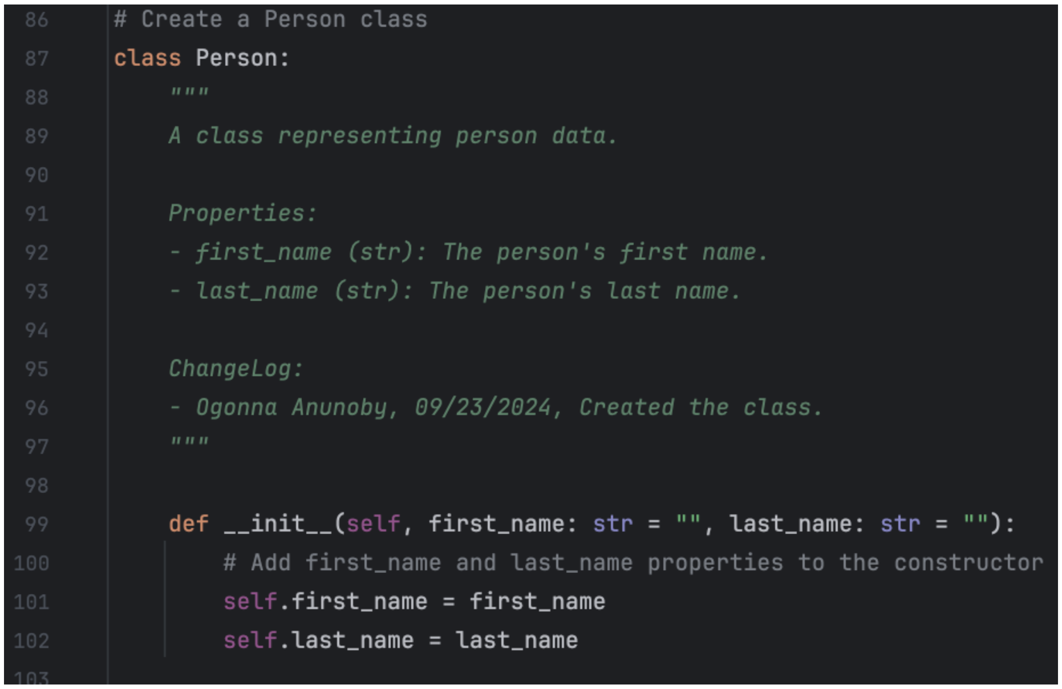


```
60      @staticmethod
61      def write_data_to_file(file_name: str, student_data: list):
62          """ This function writes data to a json file with data from a list of dictionary rows
63
64          ChangeLog: (Who, When, What)
65          Ogonna Anunoby, 09/23/2024, Created function
66
67          :param file_name: string data with name of file to write to
68          :param student_data: list of dictionary rows to be writen to the file
69
70          :return: None
71          """
72
73          try:
74              file = open(file_name, "w")
75              json.dump(student_data, file)
76              file.close()
77              IO.output_student_courses(student_data=student_data)
78          except Exception as e:
79              message = "Error: There was a problem with writing to the file.\n"
80              message += "Please check that the file is not open by another program."
81              IO.output_error_messages(message=message,error=e)
82          finally:
83              if file.closed == False:
84                  file.close()
```

Figure 4: The write_data_to_file() method

Figure 4 shows the write_data_to_file() method. This also belongs to the FileProcessor class and runs from lines 60 through 84 (including the @static method decorator). The decorator on line 60 indicates that this method is static. Lines 62 through 71 show the docstring giving a general description of this method, the change log, and a description of the parameters and return value for this method. This method writes data from the student_data list to the Json file. This method takes in a sting for the file_name and list for student_data. Since there is no return statement in this method, the method will return a None object. In other words, this method returns nothing.

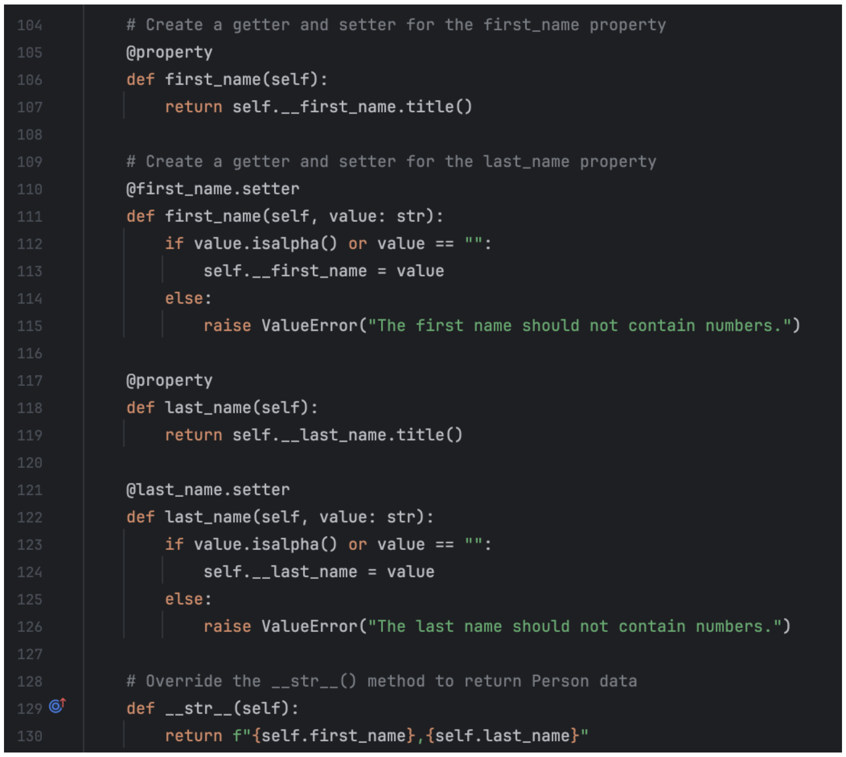
Lines 73 through 84 show the try-except-except-finally block that is used to process writing data to the file. In the try block, the code attempts to open the file in write mode, put the data in the student_data dictionary into the Json file, then close the file. If the type of data being put into the Json file is incompatible with it, a general exception will be thrown, and the code will go to line 78. The output_error_messages() method belonging to the IO class will be called, with a specified error message and the error object passed to it. In the finally block, which will always be called, if the file is still open, it will be closed.



```
86   # Create a Person class
87   class Person:
88       """
89       A class representing person data.
90
91       Properties:
92       - first_name (str): The person's first name.
93       - last_name (str): The person's last name.
94
95       ChangeLog:
96       - Ogonna Anunoby, 09/23/2024, Created the class.
97       """
98
99       def __init__(self, first_name: str = "", last_name: str = ""):
100          # Add first_name and last_name properties to the constructor
101          self.first_name = first_name
102          self.last_name = last_name
103
```

Figure 5: Person class docstring and constructor

Figure 5 shows the docstring and constructor for the Person class. This class runs from lines 89 through 130. Lines 89 through 97 show the docstring which gives a general description of this method, the change log, and a description of the parameters and return value for this method. This class contains the string properties first_name and last_name. Lines 99 through 102 have the constructor (__init__) for the class. The user can pass the first name and last name of the user, which are used to set the properties first_name and last_name. The constructor uses the "self" keyword, which is a reference to the instance of the object being used. So, on lines 99, the constructor is passed in the current instance of Person object being used, the sets the first_name and last_name properties of that instance. Also, as shown on lines 99, the constructor has default values that will set first_name and last_name to empty strings if the user omits any of those arguments.

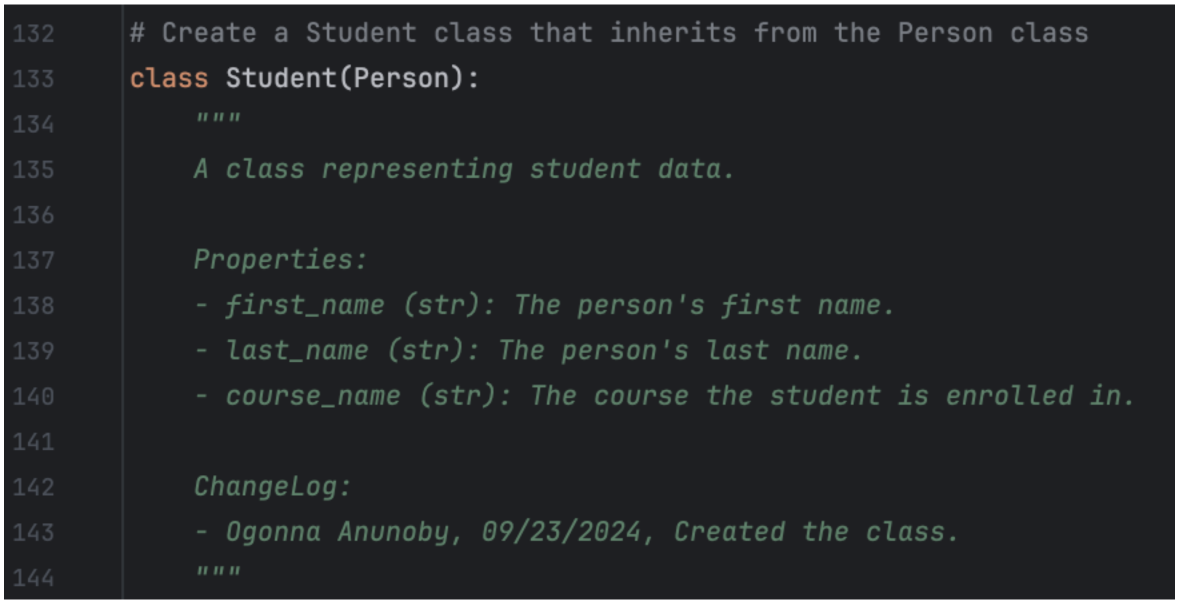


```
104        # Create a getter and setter for the first_name property
105        @property
106        def first_name(self):
107            return self.__first_name.title()
108
109        # Create a getter and setter for the last_name property
110        @first_name.setter
111        def first_name(self, value: str):
112            if value.isalpha() or value == "":
113                self.__first_name = value
114            else:
115                raise ValueError("The first name should not contain numbers.")
116
117        @property
118        def last_name(self):
119            return self.__last_name.title()
120
121        @last_name.setter
122        def last_name(self, value: str):
123            if value.isalpha() or value == "":
124                self.__last_name = value
125            else:
126                raise ValueError("The last name should not contain numbers.")
127
128        # Override the __str__() method to return Person data
129        def __str__(self):
130            return f"{self.first_name},{self.last_name}"
```

Figure 6: Person class getters, setters, and overridden str() function

Figure 6 shows the getters, setters, and overridden str() function. Getters and setters are functions that are used to access and mutate class data for the purposes of encapsulating the data and protecting them from being accessed or mutated. This allows the code follow principles of object-oriented programming. The getter for first_name is on lines 105 through 107. The decorator @property is used to indicate a getter on line 105. On line 106, the property is defined using the "def" keyword then the property name first_name as the name of a function with the "self" keyword passed as a parameter. The private variable __first_name belonging to the current instance of the Person class using title case is returned. Lines 109 through 115 show the setter for first_name. Line 110 has the decorator @first_name.setter to indicate that the following function is defined as a setter for the first_name property. On line 111, first_name takes in "self" and a string value as parameters. Lines 112 through 115 use an if-else statement to validate the value that was passed in. If the string is either an empty string or all alphabetical characters, the values passed will be used to update the first_name property. If this is not true, The ValueError expectation will be raised, and an error message will print to the screen.

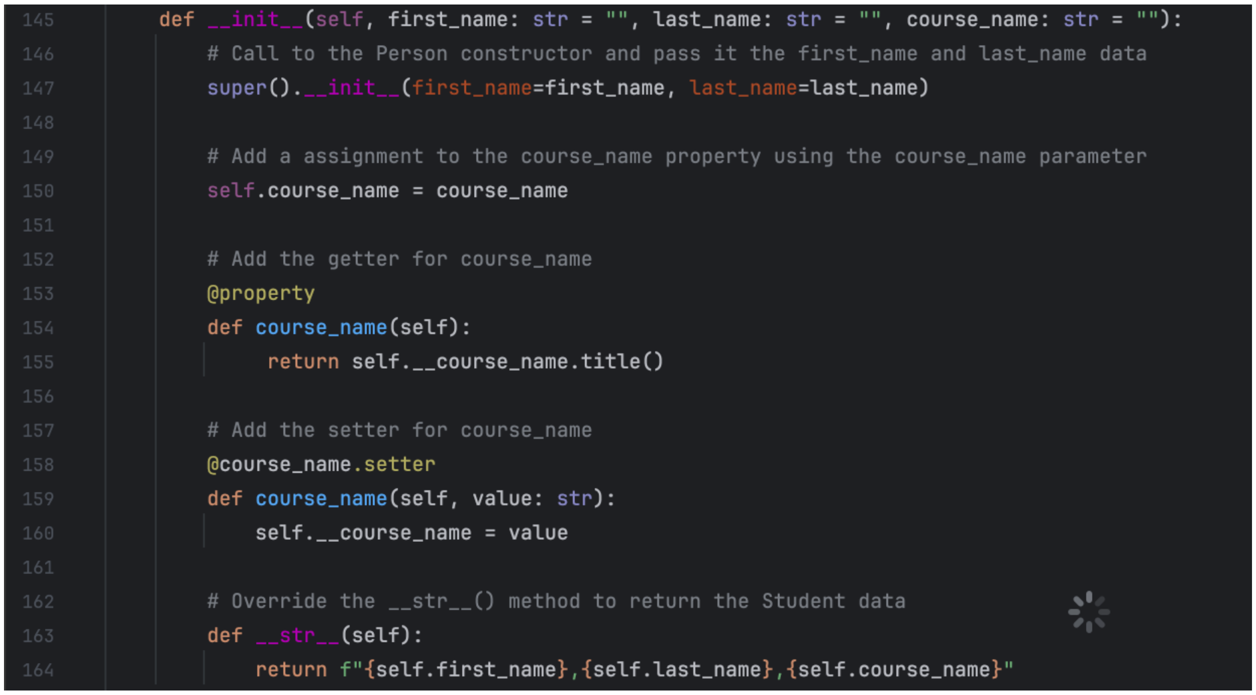
The getter for last_name is on lines 117 through 119. The decorator @property is used to indicate a getter on line 117. On line 118, the property is defined using the "def" keyword then the property name last_name as the name of a function with the "self" keyword passed as a parameter. The private variable __last_name belonging to the current instance of the Person class using title case is returned. Lines 121 through 126 show the setter for last_name. The decorator @property is used to indicate a getter on line 121. On line 122, the property is defined using the "def" keyword then the property name last_name as the name of a function with the “self” keyword passed as a parameter. Lines 121 through 126 show the setter for first_name. Line 121 has the decorator @last_name.setter to indicate that the following function is defined as a setter for the last_name property. On line 122, first_name takes in “self” and a string value as parameters. Lines 123 through 126 use an if-else statement to validate the value that was passed in. If the string is either an empty string or all alphabetical characters, the values passed will be used to update the last_name property. If this is not true, The ValueError expectation will be raised, and an error message will print to the screen. Furthermore, lines 128 through 130 show the overridden str() function, which prints a customized string representation of the Person object. Since the Person object is inherited from the object class, so it the str() object.



```
132  # Create a Student class that inherits from the Person class
133  class Student(Person):
134      """
135      A class representing student data.
136
137      Properties:
138      - first_name (str): The person's first name.
139      - last_name (str): The person's last name.
140      - course_name (str): The course the student is enrolled in.
141
142      ChangeLog:
143      - Ogonna Anunoby, 09/23/2024, Created the class.
144      """
```

Figure 7: Person class docstring

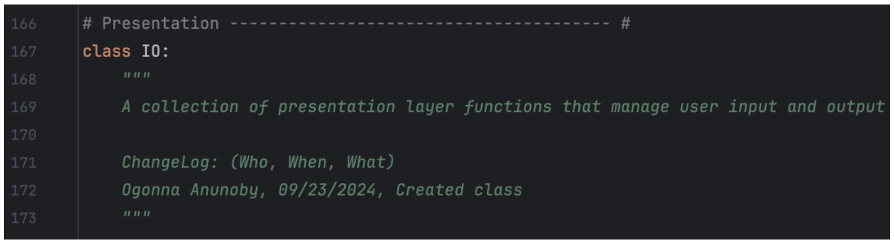
Figure 7 shows the docstring and constructor for the Person class. This class runs from lines 132 through 164. Lines 134 through 144 show the docstring which gives a general description of this method, the change log, and a description of the parameters and return value for this method. This class contains the string properties first_name, last_name, and course_name.



```
145     def __init__(self, first_name: str = "", last_name: str = "", course_name: str = ""):
146         # Call to the Person constructor and pass it the first_name and last_name data
147         super().__init__(first_name=first_name, last_name=last_name)
148 
149         # Add a assignment to the course_name property using the course_name parameter
150         self.course_name = course_name
151 
152         # Add the getter for course_name
153         @property
154         def course_name(self):
155              return self.__course_name.title()
156 
157         # Add the setter for course_name
158         @course_name.setter
159         def course_name(self, value: str):
160             self.__course_name = value
161 
162         # Override the __str__() method to return the Student data
163         def __str__(self):
164             return f"{self.first_name},{self.last_name},{self.course_name}"
```

Figure 8: Student class constructor

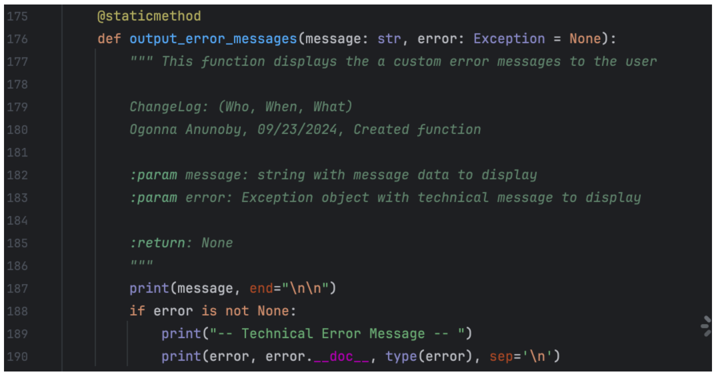
Figure 8 shows the Student class constructor. Lines 145 through 164 have the constructor (__init__) for the class. The user can pass the first name, last name, and course name of the user, which are used to set the properties first_name, last_name, and course_name. The constructor uses the "self" keyword, which is a reference to the instance of the object being used. So, on lines 145 the constructor is passed in the current instance of Student object being used, the sets the first_name, last_name and course_name properties of that instance. Also, as shown on lines 145, the constructor has default values that will set first_name, last_name, and course_name to empty strings if the user omits any of those arguments. On line 147, super().__init__() is used to refer to the Person constructor, which the Student class inherits from the Person class. The properties first_name and last_name are passed into it. On line 150, the course_name property is set using the value passed it to the constructor. Lines 152 through 155 show the getter for course_name and lines 157 through 160 show the setter for course_name. Lines 162 show the overridden str() function inherited from the Person class, which returns a customized string representation of the Student class.



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

    ChangeLog: (Who, When, What)
    Ogonna Anunoby, 09/23/2024, Created class
    """
```

Figure 9: IO class and its docstring

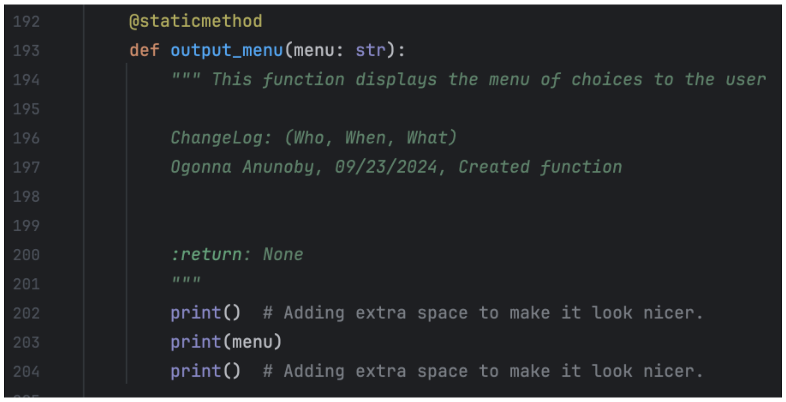
Figure 9 shows the IO class and its docstring. The IO class runs from lines 166 through 275 Lines 168 through 173 show the docstring for this class, which explains that this class has methods to deal with input and output. It also lists the change log for this class.



```
175        @staticmethod
176        def output_error_messages(message: str, error: Exception = None):
177            """ This function displays the a custom error messages to the user
178
179            ChangeLog: (Who, When, What)
180            Ogonna Anunoby, 09/23/2024, Created function
181
182            :param message: string with message data to display
183            :param error: Exception object with technical message to display
184
185            :return: None
186            """
187            print(message, end="\n\n")
188            if error is not None:
189                print("-- Technical Error Message -- ")
190                print(error, error.__doc__, type(error), sep='\n')
```

Figure 10: The output_error_messages() method

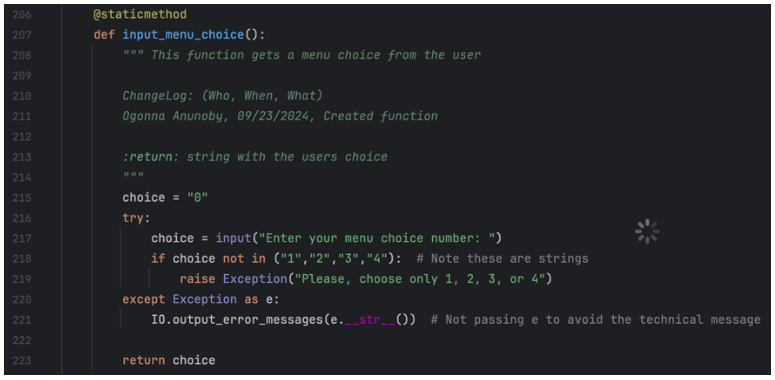
Figure 10 shows the output_error_messages() method. The method runs from lines 175 through 190 (including the @staticmethod decorator). The docstring runs from lines 177 through lines 186. The docstring explains that the method displays custom error messages to the user. It also has a change log and explains the method takes a string for message and exception object for error. The method returns nothing. It prints the message string that was passed to is, followed by two new lines as indicated by the end parameter in the print statement on line 187. Is should be noted that on line 176, the error exception object has a default value of None. So, this parameter is optional. The if statement on line 168 only runs if an exception object was passed to it. If this is the case, then an error message is printed, as well as the error object, error docstring, error type, each separated by a new line.



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

        ChangeLog: (Who, When, What)
        Ogonna Anunoby, 09/23/2024, 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: The output_menu() method

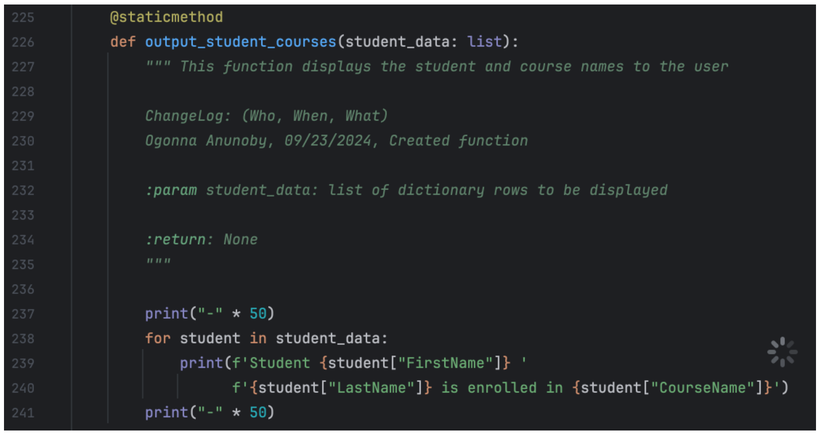
Figure 11 shows the output_menu() method. The method runs from lines 192 through 204 (including the @staticmethod decorator). The docstring runs from lines 194 through lines 201. The docstring explains that the method displays the menu to the user. It also has a change log and explains the method takes no parameters and returns nothing.



```
206     @staticmethod
207     def input_menu_choice():
208         """ This function gets a menu choice from the user
209 
210         ChangeLog: (Who, When, What)
211         Ogonna Anunoby, 09/23/2024, Created function
212 
213         :return: string with the users choice
214         """
215         choice = "0"
216         try:
217             choice = input("Enter your menu choice number: ")
218             if choice not in ("1","2","3","4"):  # Note these are strings
219                 raise Exception("Please, choose only 1, 2, 3, or 4")
220         except Exception as e:
221             IO.output_error_messages(e.__str__())  # Not passing e to avoid the technical message
222 
223         return choice
```

Figure 12: The input_menu_choice() method

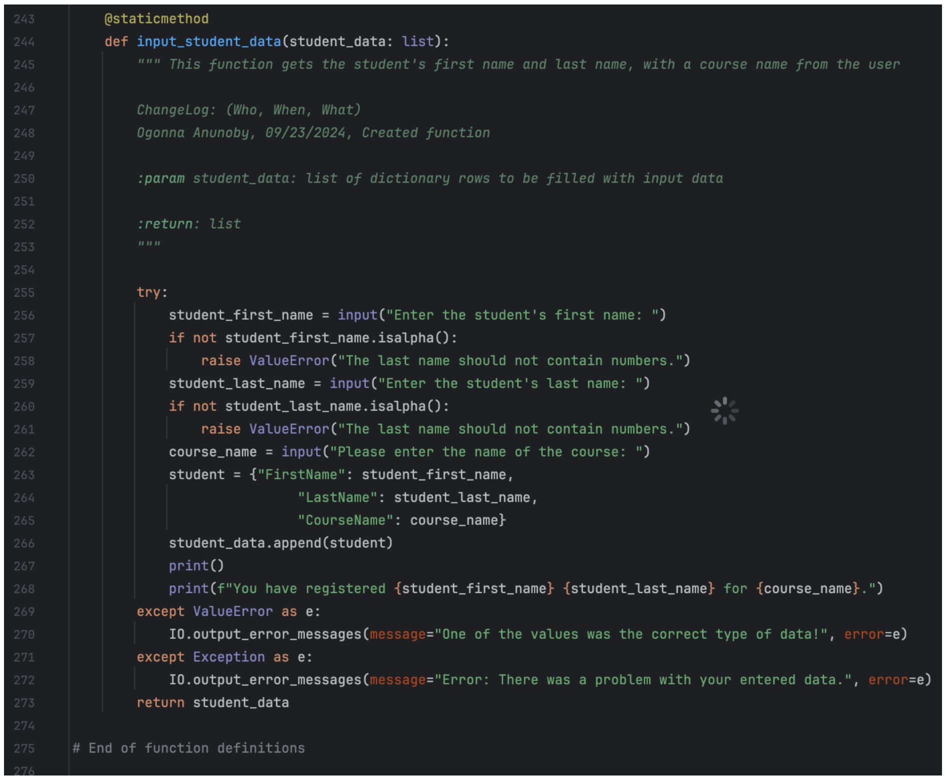
Figure 12 shows the input_menu_choice() method. The method runs from lines 206 through 223 (including the @staticmethod decorator). The docstring runs from lines 208 through lines 214. The docstring explains that the method gets the menu choice from the user. It also has a change log and explains the method takes no parameters and returns the string choice. Furthermore, this method uses a try-except block to ensure that only valid choices “1”, “2”, “3”, and “4” are entered. If an invalid choice is entered, an exception is raised on line 146, then caught on line 220, Then the output_error_messages() method belonging to the IO class is called, passing in the error’s string object. If a valid choice is entered, the choice is returned.



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

        ChangeLog: (Who, When, What)
        Ogonna Anunoby, 09/23/2024, Created function

        :param student_data: list of dictionary rows to be displayed

        :return: None
        """

        print("-" * 50)
        for student in student_data:
            print(f'Student {student["FirstName"]} '
                  f'{student["LastName"]} is enrolled in {student["CourseName"]}')
        print("-" * 50)
```

Figure 13: The output_student_courses() method

Figure 13 shows the output_student_courses() method. This method runs from lines 225 through 241 (including the @staticmethod decorator). The docstring runs from lines 227 through lines 235. The docstring explains that the method displays the registration data to the user. It also has a change log and explains the method takes a list for student_data and returns nothing. This method prints a boarder to surround the registration data on lines 237 and 241 printing. It does this by printing “-” fifty times to the screen. Lines 239 and 240 iterate through each student in the student_data dictionary in the students list and prints out each student’s information using the student dictionary and the FirstName, LastName, and CourseName keys.



```
243     @staticmethod
244     def input_student_data(student_data: list):
245         """ This function gets the student's first name and last name, with a course name from the user
246 
247         ChangeLog: (Who, When, What)
248         Ogonna Anunoby, 09/23/2024, Created function
249 
250         :param student_data: list of dictionary rows to be filled with input data
251 
252         :return: list
253         """
254 
255         try:
256             student_first_name = input("Enter the student's first name: ")
257             if not student_first_name.isalpha():
258                 raise ValueError("The last name should not contain numbers.")
259             student_last_name = input("Enter the student's last name: ")
260             if not student_last_name.isalpha():
261                 raise ValueError("The last name should not contain numbers.")
262             course_name = input("Please enter the name of the course: ")
263             student = {"FirstName": student_first_name,
264                        "LastName": student_last_name,
265                        "CourseName": course_name}
266             student_data.append(student)
267             print()
268             print(f"You have registered {student_first_name} {student_last_name} for {course_name}.")
269         except ValueError as e:
270             IO.output_error_messages(message="One of the values was the correct type of data!", error=e)
271         except Exception as e:
272             IO.output_error_messages(message="Error: There was a problem with your entered data.", error=e)
273         return student_data
274 
275 # End of function definitions
```

Figure 14: The input_student_data() method

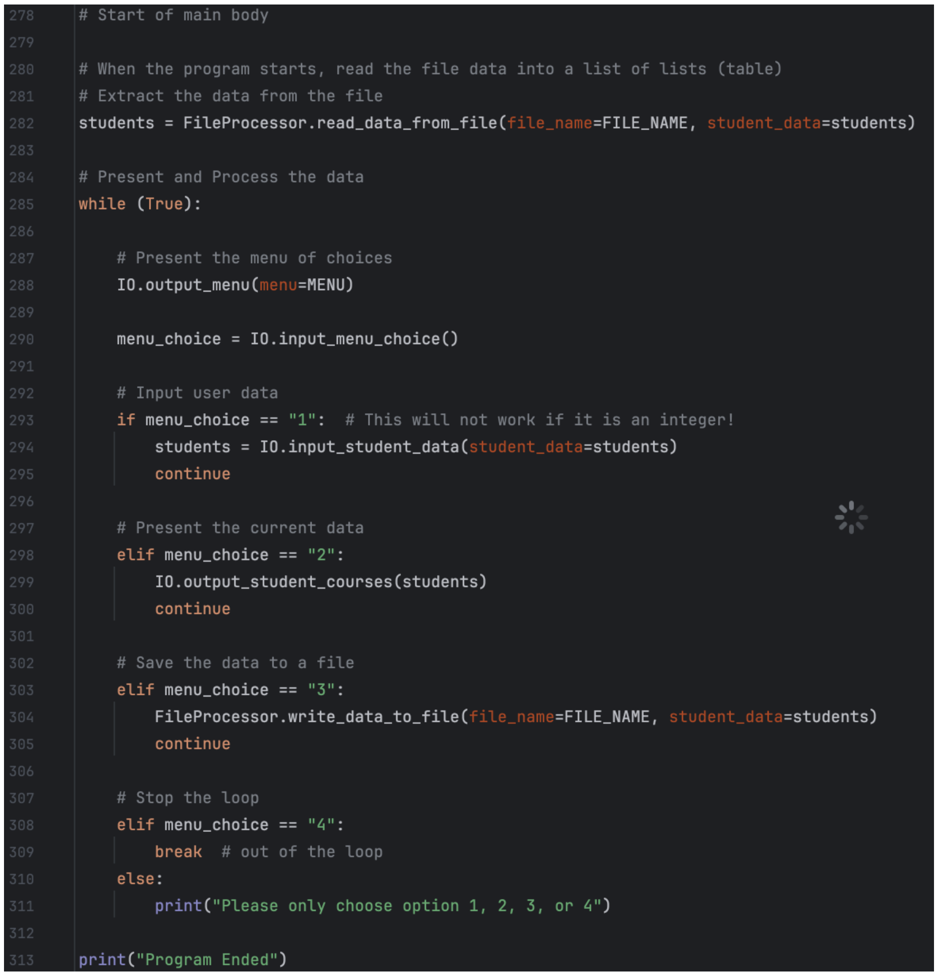
Figure 14 shows the input_student_data() method. This method runs from lines 243 through 273 (including the @staticmethod decorator). The docstring runs from lines 245 through lines 253. The docstring explains that this method gets the student’s first name, last name, and course choice from the user. It also has a change log and explains the method takes a list for student_data and returns the student_data list.

In the try block from lines 255 through 268, the user is asked to enter a first name on line 256. Line 257 checks if the user entered all alphabetical characters for the student's first name. If they did, the code asks the user to enter the student's last name on line 259. If they did not enter all alphabetical characters for the student's first name, a ValueError exception will be thrown, and the code will jump to the ValueError exception on line 269. Then, the output_error_messages() method belonging to the IO class will be called with a specified error message and the error message passed as parameters.

Line 260 checks if the user entered all alphabetical characters for the student's last name. If they did, the code would move to line 262 and asks the user to input the course name. If they did not enter all alphabetical characters for the student's first name, a ValueError exception will be thrown, and the code will jump to the ValueError exception on line 269. Then, the output_error_messages() method belonging to the IO class will be called with a specified error message and the error message passed as parameters.

If the user successfully enters a first and last name, the user enters the course name on line 262. Then, a dictionary named student is constructed from the entered first name, last name, and course name. The dictionary is appended to student_data, which is returned by the input_student_data() method.

The else block on lines 199 through 203 will be run if no exceptions are thrown. Here, the user is asked to enter the name of the course. Then, on line 201 the dictionary for the student is created with FirstName, LastName, and CourseName as keys and student_first_name, student_last_name, and course_name as values. The student dictionary is appended to the students list on line 202. Then a message letting the user know that the student was registered is printed to the screen. Line 207 has a comment indicating that the script is at the end of all of the function definitions.



```
278  # Start of main body
279
280  # When the program starts, read the file data into a list of lists (table)
281  # Extract the data from the file
282  students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)
283
284  # Present and Process the data
285  while (True):
286
287      # Present the menu of choices
288      IO.output_menu(menu=MENU)
289
290      menu_choice = IO.input_menu_choice()
291
292      # Input user data
293      if menu_choice == "1":  # This will not work if it is an integer!
294          students = IO.input_student_data(student_data=students)
295          continue
296
297      # Present the current data
298      elif menu_choice == "2":
299          IO.output_student_courses(students)
300          continue
301
302      # Save the data to a file
303      elif menu_choice == "3":
304          FileProcessor.write_data_to_file(file_name=FILE_NAME, student_data=students)
305          continue
306
307      # Stop the loop
308      elif menu_choice == "4":
309          break  # out of the loop
310      else:
311          print("Please only choose option 1, 2, 3, or 4")
312
313  print("Program Ended")
```

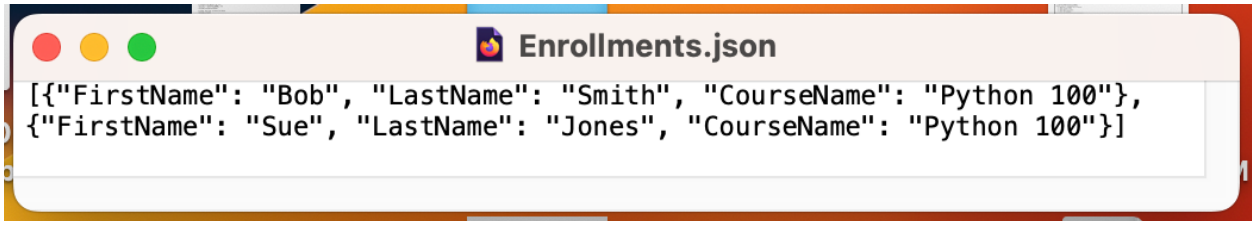
Figure 15: Main body while loop

Figure 15 shows the main body while loop. Line 278 has a comment indicating that the script is at the main body of the script. Line 282 calls the read_data_from_file() method belonging to the FileProcessor class. It passes in the file_name string and student list. The list returned from this function is stored in the students variable. The infinite while loop running from lines 285 through 311 allows the user to interact with the application. This is shown in Figure 15. It continuously prints the menu choice one line 288 using the output_menu() function from the IO class, with MENU passed to it. On line 290, the input_menu_choice() method from the IO class is called to get a valid menu choice from the user. The value returned from the function is stored in menu_choice. If menu_choice is “1”, it calls the input_student_data() function from the IO class, passing the students list as a parameter. It stores the returned list in students. If menu_choice is “2”, it calls the output_student_courses() function from the IO class, passing the students list as a parameter. It stores the returned list in students. If menu_choice is “3”, it calls the write_data_to_file() function from the FileProcessor class, passing the file_name string and students list as parameters. It stores the returned list in students. If the menu_choice is

“4”, the code breaks out of the loop. Then on line 313, “Program Ended” is printed onto the screen, and the program ends.

Script Testing

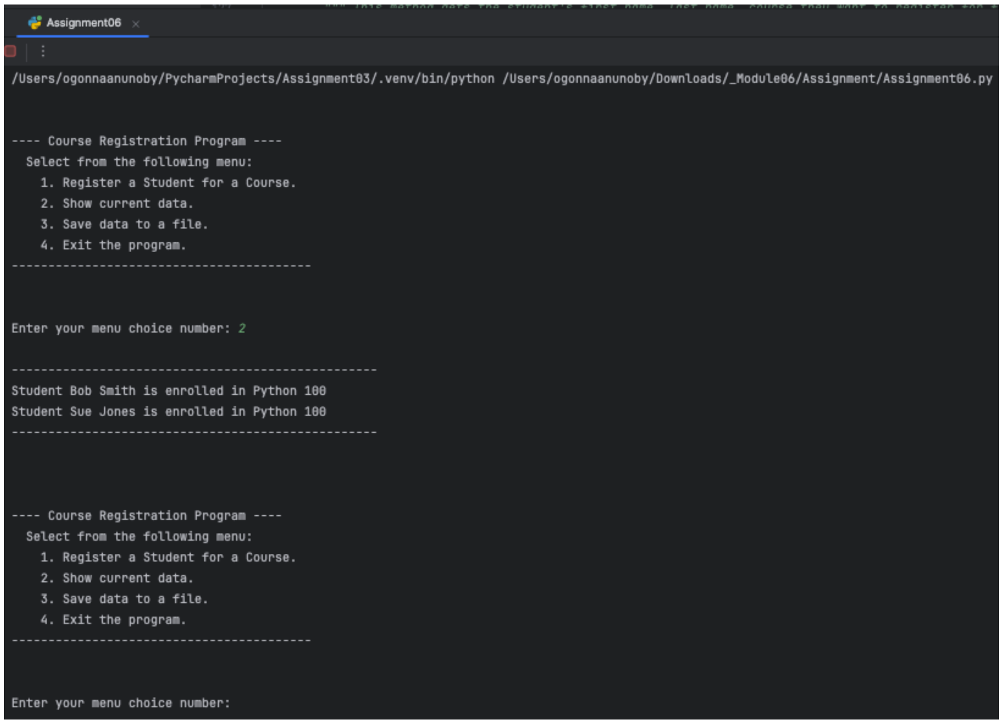
After writing the code for the script, I needed to test it. First, I saved the script using the name Assignment07.py to my _Module06/Assignments folder in my downloads folder. I ran the script and tested all of the options to ensure that my script worked as I expected.



```
[{"FirstName": "Bob", "LastName": "Smith", "CourseName": "Python 100"},
{"FirstName": "Sue", "LastName": "Jones", "CourseName": "Python 100"}]
```

Figure 16: Enrollments.json Before Running the Script

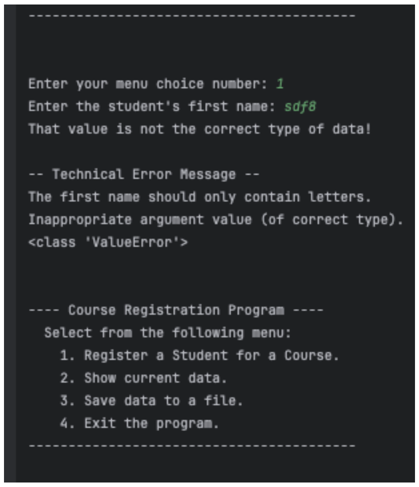
Figure 16 shows the contents of Enrollments.json before running the script. I ran the script in PyCharm and chose choice “2” to display each registered student. The contents of the Json file were printed to the screen, as expected, as shown in Figure 13.



```
/Users/ogonnaanunoby/PycharmProjects/Assignment03/.venv/bin/python /Users/ogonnaanunoby/Downloads/_Module06/Assignment/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: 2

--------------------------------------------------
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
--------------------------------------------------



---- 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:
```

Figure 17: Students Enrolled Before Running the Script

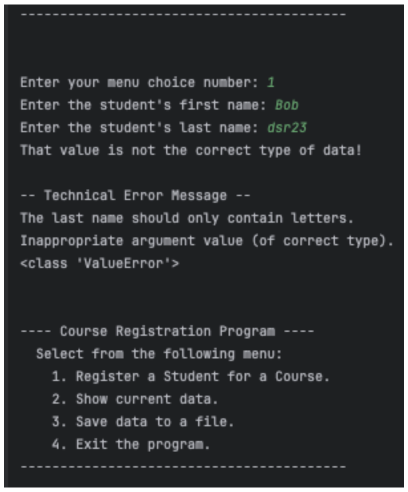
Next, I made choice “1” to register a student for the course. I entered a name with numbers in it and the ValueError exception was thrown.



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


Enter your menu choice number: 1
Enter the student's first name: sdf8
That value is not the correct type of data!

-- Technical Error Message --
The first name should only contain letters.
Inappropriate argument value (of correct type).
<class 'ValueError'>


---- 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 18: ValueError Exception for the Student’s First Name

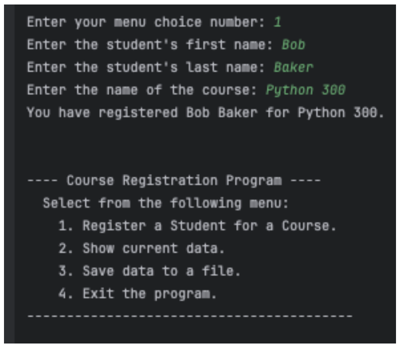
Then, I made choice “1” again and entered all alphabetic characters for the student’s first name. The code then prompted me to enter the student’s last name. I entered a name with numbers in it, and the ValueError exception was thrown again.



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


Enter your menu choice number: 1
Enter the student's first name: Bob
Enter the student's last name: dsr23
That value is not the correct type of data!

-- Technical Error Message --
The last name should only contain letters.
Inappropriate argument value (of correct type).
<class 'ValueError'>


---- 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 19: ValueError Exception for the Student’s Last Name

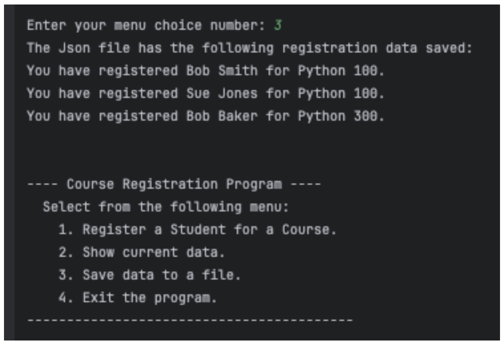
I then entered the name of the course, and the student was registered for the course.



```
Enter your menu choice number: 1
Enter the student's first name: Bob
Enter the student's last name: Baker
Enter the name of the course: Python 300
You have registered Bob Baker for Python 300.


---- 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 20: Successfully Registering a Student

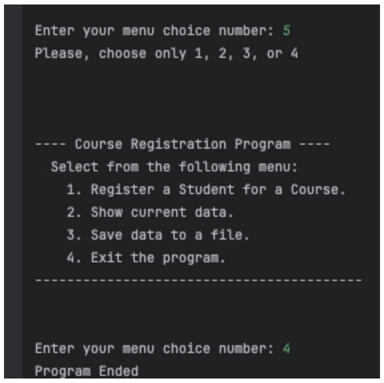
Figure 21 shows me making choice "3", displaying the data and saving the data to Enrollments.csv.



```
Enter your menu choice number: 3
The Json file has the following registration data saved:
You have registered Bob Smith for Python 100.
You have registered Sue Jones for Python 100.
You have registered Bob Baker for Python 300.


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

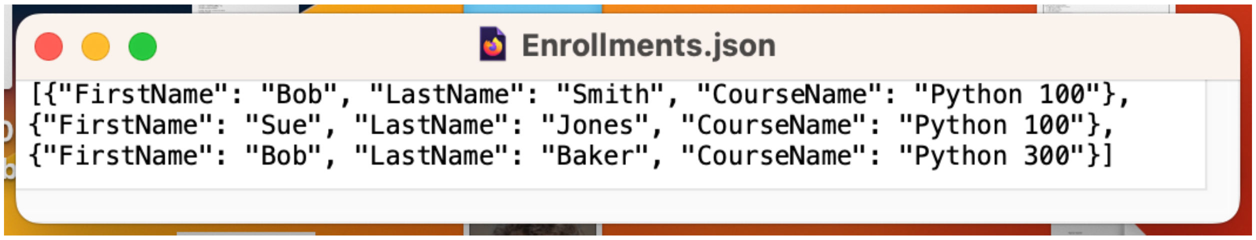
Figure 21: Saving All of the Registered Students

In Figure 22, I enter an invalid choice, and I get an error message and am prompted to try again. I then choose "4" and exit the program.



```
Enter your menu choice number: 5
Please, choose only 1, 2, 3, or 4



---- 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
```

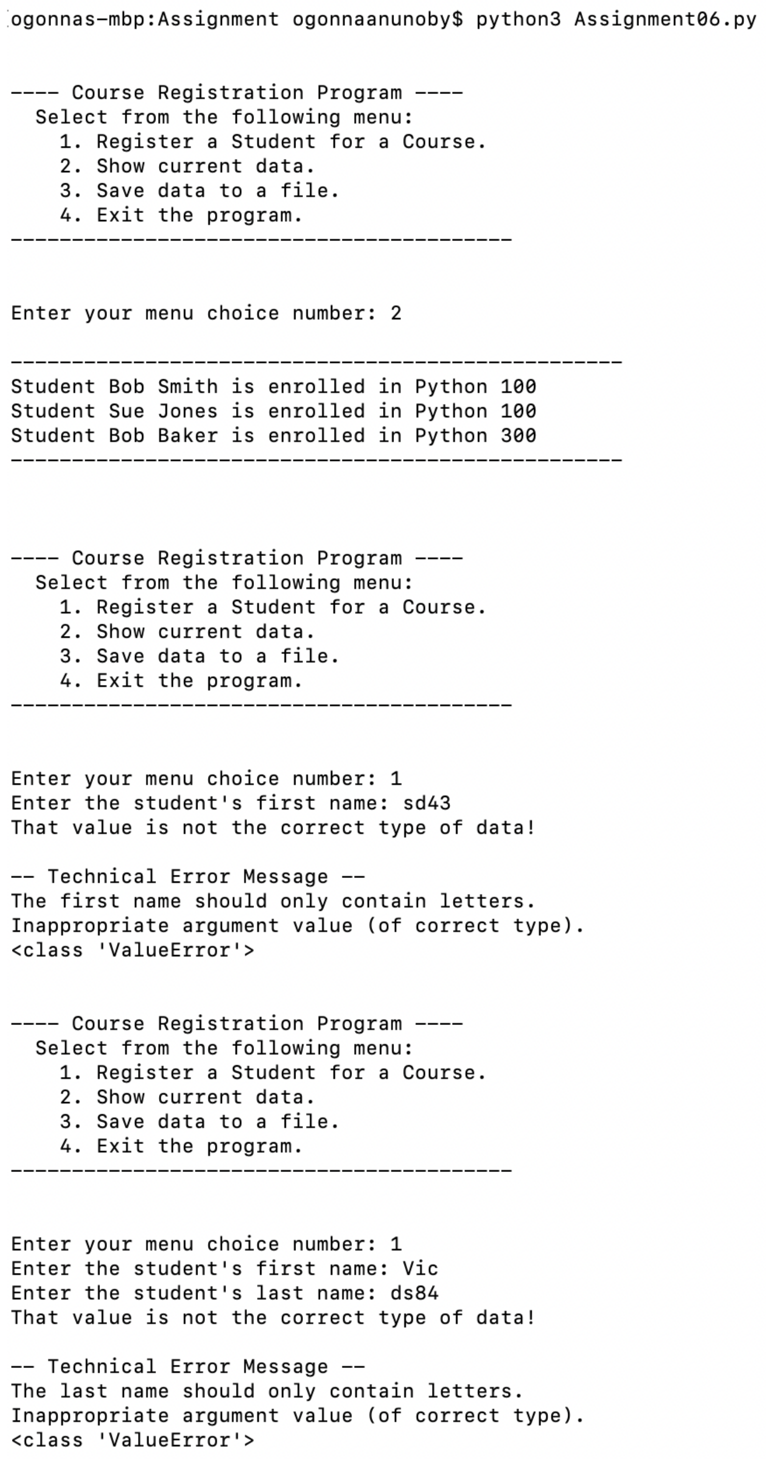
Figure 22: Entering an invalid Option and Exiting the Program



```
[{"FirstName": "Bob", "LastName": "Smith", "CourseName": "Python 100"},
{"FirstName": "Sue", "LastName": "Jones", "CourseName": "Python 100"},
{"FirstName": "Bob", "LastName": "Baker", "CourseName": "Python 300"}]
```

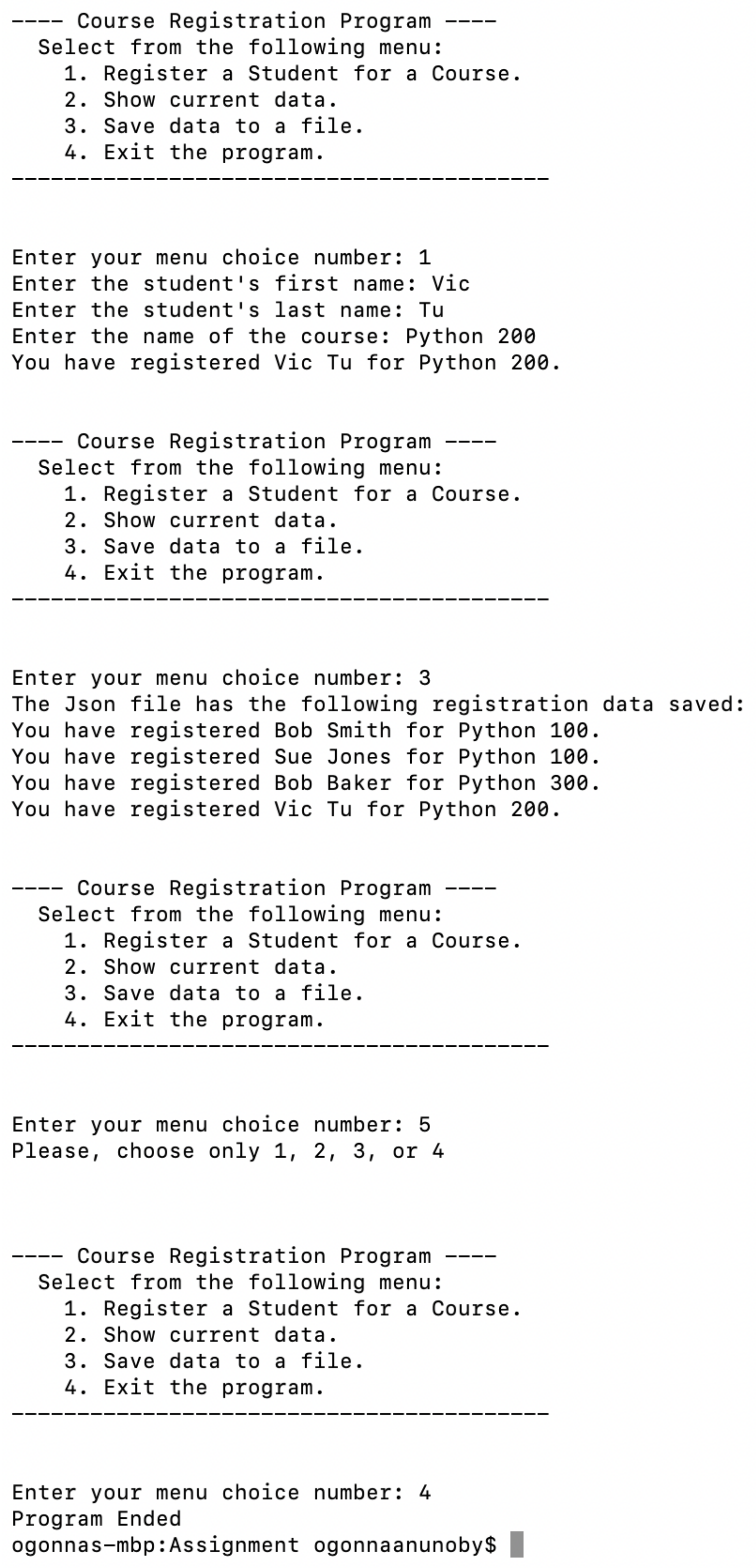
Figure 23: Showing All Registered Students in the Json File

Figure 23 shows that each registered student has been saved to Enrollments.json. I also ran the script in the terminal using the same commands as I did in PyCharm. I started with the Enrollments.json file shown in Figure 11. Then, I ran the script in the terminal with the same results. The script output in the terminal is shown in Figures 24 and 25. Figure 26 shows the final Enrollments.json file I got.



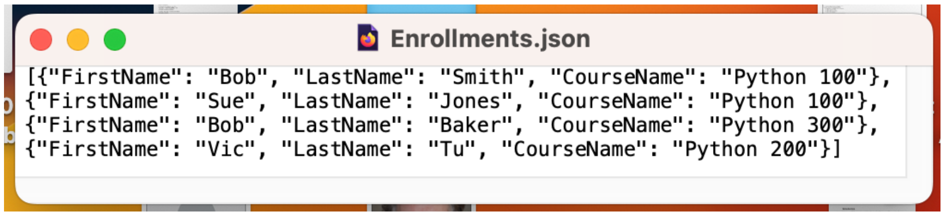
```
ogonnas-mbp:Assignment ogonnaanunoby$ python3 Assignment06.py


---- 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: 2

--------------------------------------------------
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Bob Baker is enrolled in Python 300
--------------------------------------------------



---- 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: sd43
That value is not the correct type of data!

-- Technical Error Message --
The first name should only contain letters.
Inappropriate argument value (of correct type).
<class 'ValueError'>


---- 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: Vic
Enter the student's last name: ds84
That value is not the correct type of data!

-- Technical Error Message --
The last name should only contain letters.
Inappropriate argument value (of correct type).
<class 'ValueError'>
```

Figure 24: Terminal Script Results, Part 1



```
---- 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: Vic
Enter the student's last name: Tu
Enter the name of the course: Python 200
You have registered Vic Tu for Python 200.


---- 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 Json file has the following registration data saved:
You have registered Bob Smith for Python 100.
You have registered Sue Jones for Python 100.
You have registered Bob Baker for Python 300.
You have registered Vic Tu for Python 200.


---- 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: 5
Please, choose only 1, 2, 3, or 4



---- 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
ogonnas-mbp:Assignment ogonnaanunoby$
```

Figure 25: Terminal Script Results, Part 2


Enrollments.json

```
[{"FirstName": "Bob", "LastName": "Smith", "CourseName": "Python 100"},
{"FirstName": "Sue", "LastName": "Jones", "CourseName": "Python 100"},
{"FirstName": "Bob", "LastName": "Baker", "CourseName": "Python 300"},
{"FirstName": "Vic", "LastName": "Tu", "CourseName": "Python 200"}]
```

Figure 26: Again, Showing All Registered Students in the Json File

Finally, I uploaded my report and python script to the GitHub repository I created for this assignment. This can be found at https://github.com/944695/IntroToProg-Python-Mod07.

Conclusion

In conclusion, Assignment 07 allowed me to practice the skills I gained from lesson 07. It allowed me to practice using principles of object-oriented programing such as classes, methods, setters, getters, and properties. Overall, completing Assignment 07 has helped me continue building my Python knowledge.

Citations

Mod07-Notes.docx
Assignment07.py