Kyle Hayward

11/28/2023

Foundations of Programming: Python

Assignment 07

GitHub Link: https://github.com/HaywardUW/IntroToProg-Python-Mod07

Constructors, Properties, and Inheritance

Introduction

Module 07 goes into detail about constructors, properties, and the concepts behind inheritance. This module also covers statements, functions, and classes in more depth. Not only did we learn about several core elements in OOP (object-oriented programming), but also about streamlining version control with Git and GitHub Desktop.

In this Assignment 07 documentation, I will summarize the difference between Objects and Classes, what Constructors and Properties are, and how to implement Class Inheritance into a Python program.

I will also demonstrate how to build a program using constants, variables, classes, class properties, inheritance, and class methods to extract coma-separated data from each data class.

Objects and Classes

Now that we are building much more complex programs, we need to be able to distinguish between all the terminology used when creating Python code.

New programmers can often be confused between Objects and Classes. So, what is the difference between an Object and a Class?

You can think of a Class as a cookie cutter or blueprint for creating Objects. For instance, a class named People could be created. The class could then create multiple objects that would in turn be separate people. Each person created from the People class would be a new instance, each with its own attributes such as first name, last name, hair color, etc.

In short, classes are used to create objects.

Constructors and Properties

A Constructor is a method (or function) that is called automatically when a class object is created. Constructors initialize data and are used to set the attributes of each object. The default Python constructor name is __init__ and never has a return type as the attributes are set globally or by special functions.

Properties are functions used to access and modify attributes. For each attribute being defined, a ‘getter’ and a ‘setter’ function are created. This allows access to the data as well as applying program formatting. When using properties, ensure that the setter and the getter function names are the same. This will create a ‘getter and setter’ pair. While this practice is a bit odd in OOP, it is required in Python programming.

Inheritance

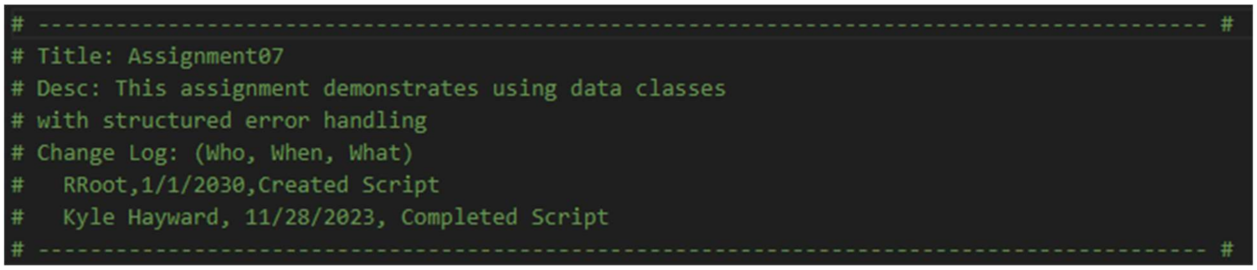
Inheriting code is a pivotal and quite powerful concept in OOP. Inheritance is when a child class uses code from the parent class so that the program code is easier to read, more manageable, and much more dynamic.

The developer does not need to re-write code for multiple classes. They can simply access code from the parent class and implement it to multiple child classes. If code is changed in the parent class, it does not need to be changed in the child class.

Creating the Program

Now that we understand more powerful Python concepts such as classes, objects, constructors, and properties; we can build a much more manageable and dynamic program. This section will demonstrate how to accomplish just that.

As with all programs within this course, I begin the program with a script header. (Figure 1.1)

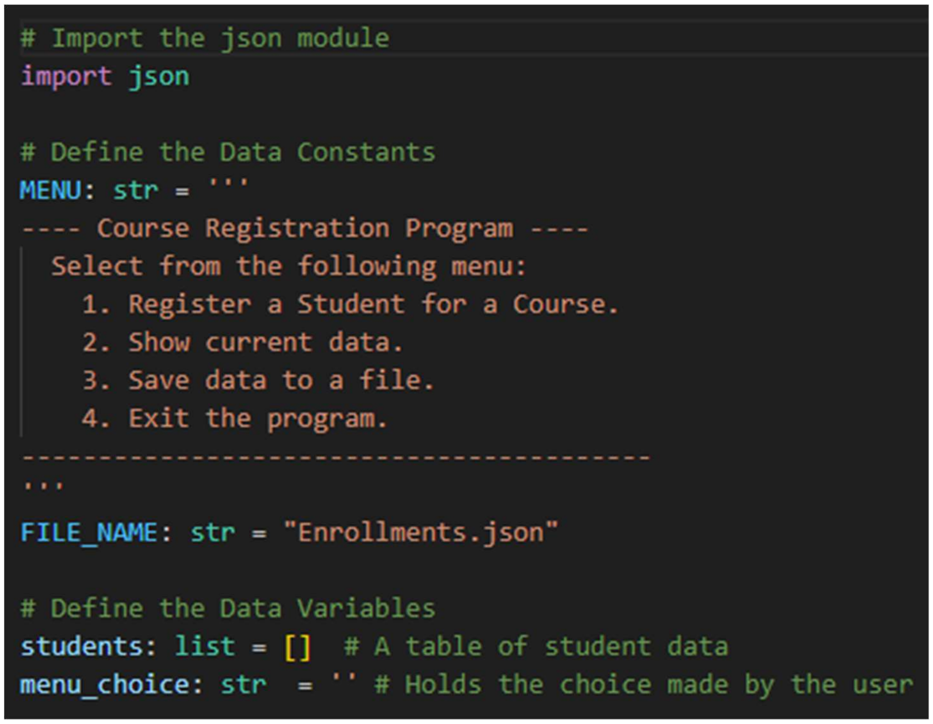


```
# ------------------------------------------------------------------------------------------ #
# Title: Assignment07
# Desc: This assignment demonstrates using data classes
# with structured error handling
# Change Log: (Who, When, What)
#   RRoot,1/1/2030,Created Script
#   Kyle Hayward, 11/28/2023, Completed Script
# ------------------------------------------------------------------------------------------ #
```

Figure 1.1: Starting the program with a script header.

I then import the json module, define the two data constants, and the two data variables.

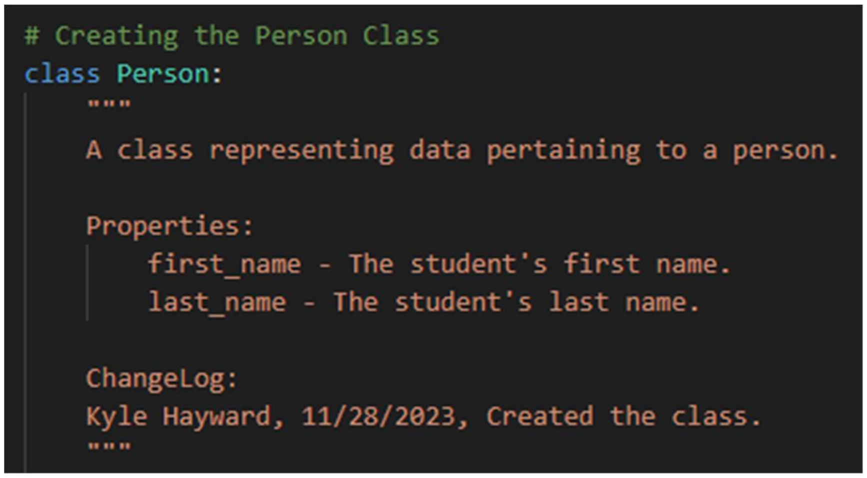
The data constants consist of the variables that hold both Course Registration Menu and the “Enrollments.json” file. The data variables consist of an empty student list and an empty string which will hold the menu choice input by the user of the program. (Figure 1.2)



```python
# Import the json module
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  = '' # Holds the choice made by the user
```

Figure 1.2: Importing the json module and defining the data constants and variables.

I created the first class of the program and named it Person. After creating the class, I included a docstring which summarizes what the class is designed to do. (Figure 1.3)



```python
# Creating the Person Class
class Person:
    """
    A class representing data pertaining to a person.

    Properties:
        first_name - The student's first name.
        last_name - The student's last name.

    ChangeLog:
    Kyle Hayward, 11/28/2023, Created the class.
    """
```

Figure 1.3: Creating the first class in the program (Person).

This Person class consists of the __init__ constructor that sets private attributes.

It also creates a property getter and setter for both first_name and last_name.

Lastly, I override the default __str__ method to return a coma-separated string of data with self.first_name and self.last_name wrapped in an f-string. (Figure 1.4)

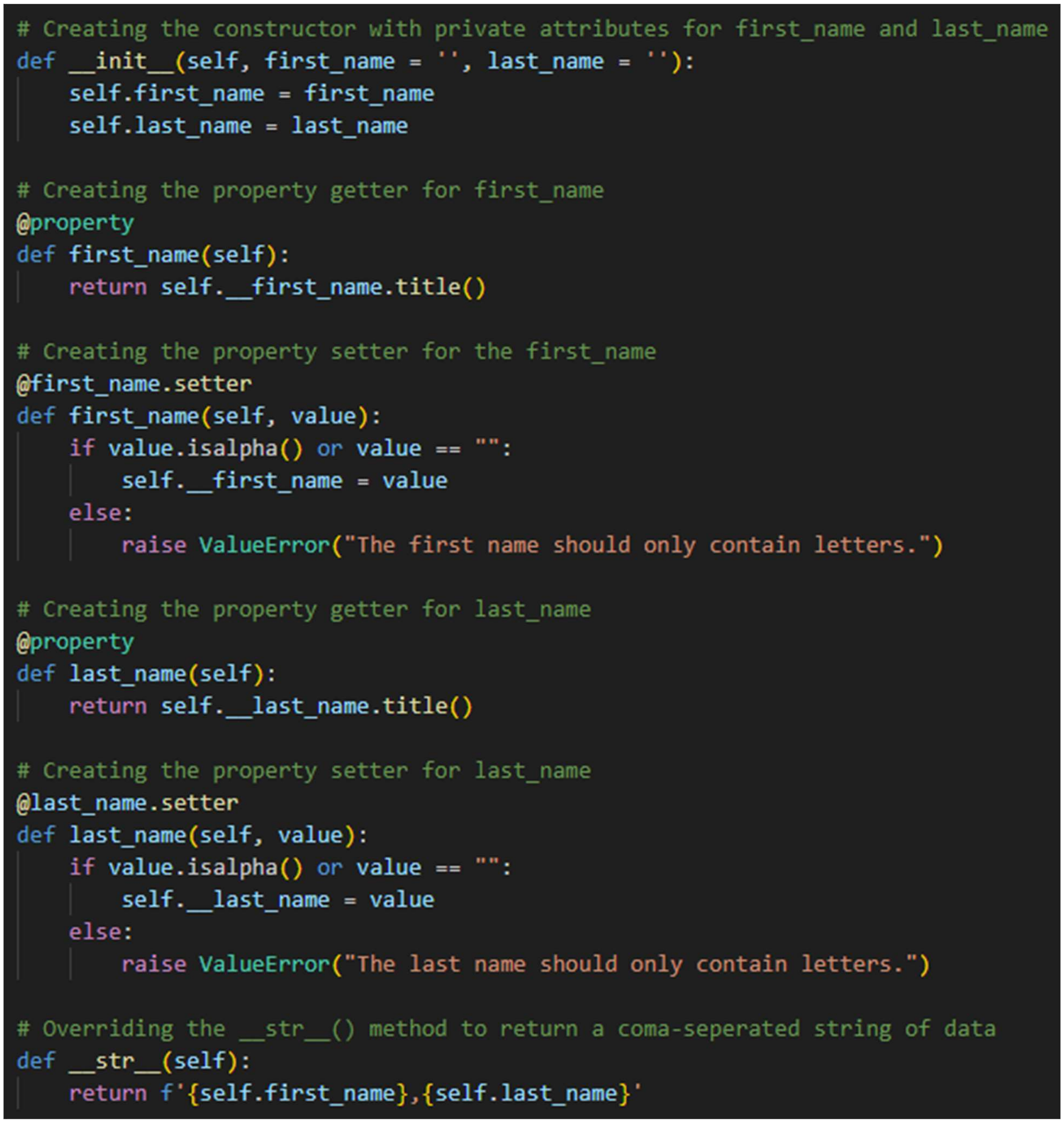
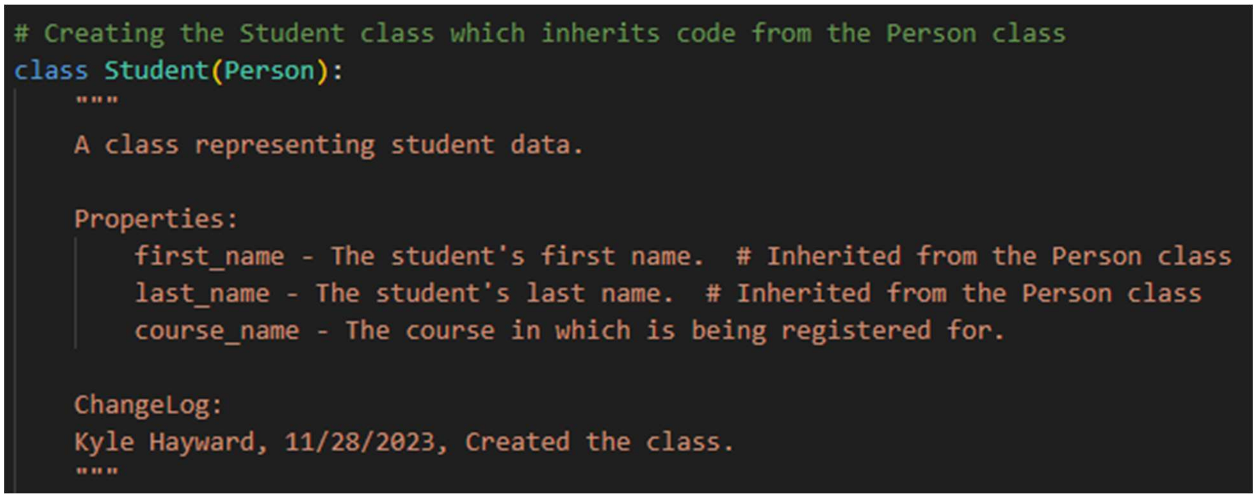

```python
# Creating the constructor with private attributes for first_name and last_name
def __init__(self, first_name = '', last_name = ''):
    self.first_name = first_name
    self.last_name = last_name

# Creating the property getter for first_name
@property
def first_name(self):
    return self.__first_name.title()

# Creating the property setter for the first_name
@first_name.setter
def first_name(self, value):
    if value.isalpha() or value == "":
        self.__first_name = value
    else:
        raise ValueError("The first name should only contain letters.")

# Creating the property getter for last_name
@property
def last_name(self):
    return self.__last_name.title()

# Creating the property setter for last_name
@last_name.setter
def last_name(self, value):
    if value.isalpha() or value == "":
        self.__last_name = value
    else:
        raise ValueError("The last name should only contain letters.")

# Overriding the __str__() method to return a coma-seperated string of data
def __str__(self):
    return f'{self.first_name},{self.last_name}'
```

Figure 1.4: Completing the Person class with a constructor, properties and __str__ method.

I then created the second class, named it Student, and configured it to inherit code from the parent class. This class inherits both the first_name and last_name properties from the Person class.

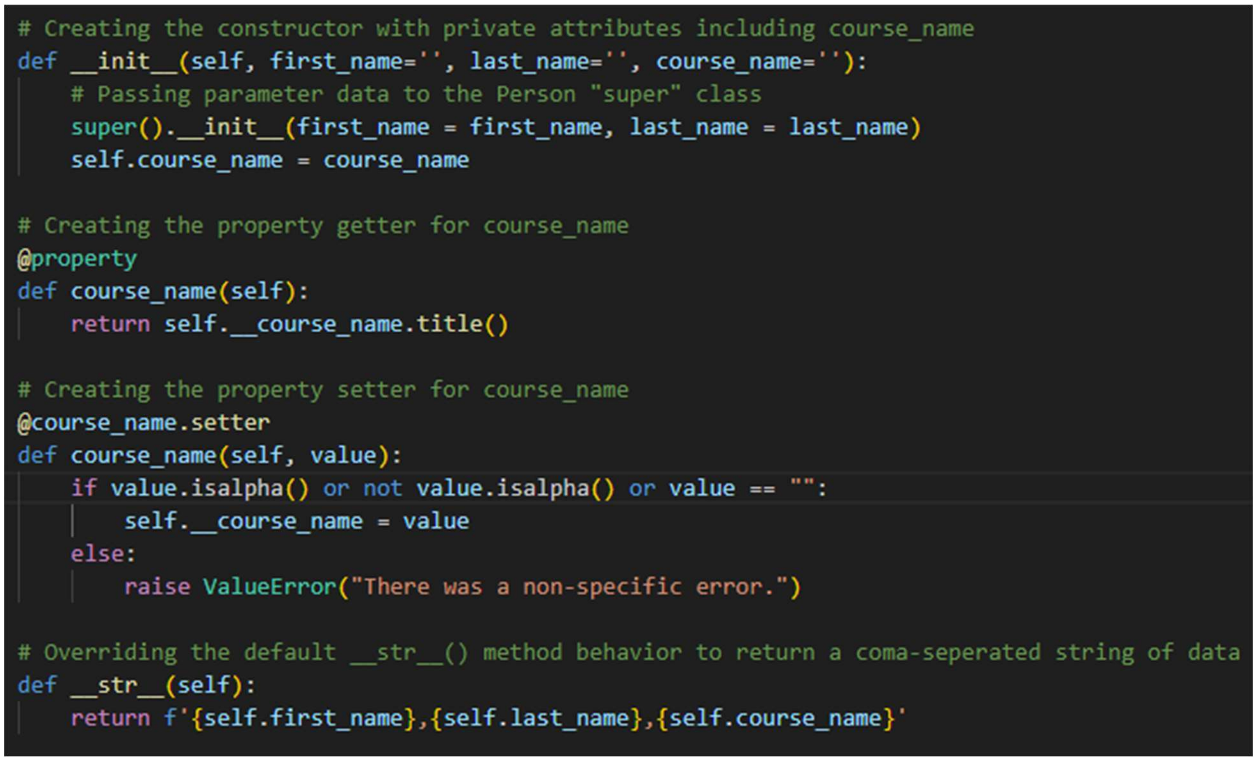
As with the Person class, I include a docstring summarizing what the class does and the properties within it. (Figure 1.5)



```
# Creating the Student class which inherits code from the Person class
class Student(Person):
    """
    A class representing student data.

    Properties:
        first_name - The student's first name.  # Inherited from the Person class
        last_name - The student's last name.  # Inherited from the Person class
        course_name - The course in which is being registered for.

    ChangeLog:
    Kyle Hayward, 11/28/2023, Created the class.
    """
```

Figure 1.5: Creating the Student class which inherits code from the parent class (Person)

The Student class is similar in construction to the Person class in that it has a constructor with private attributes, a property setter, a property getter, and a __str__ method that returns a comma-separated string of data**.** The only difference here is that the Student class is a sub-class of the parent Person class. (Figure 1.6)

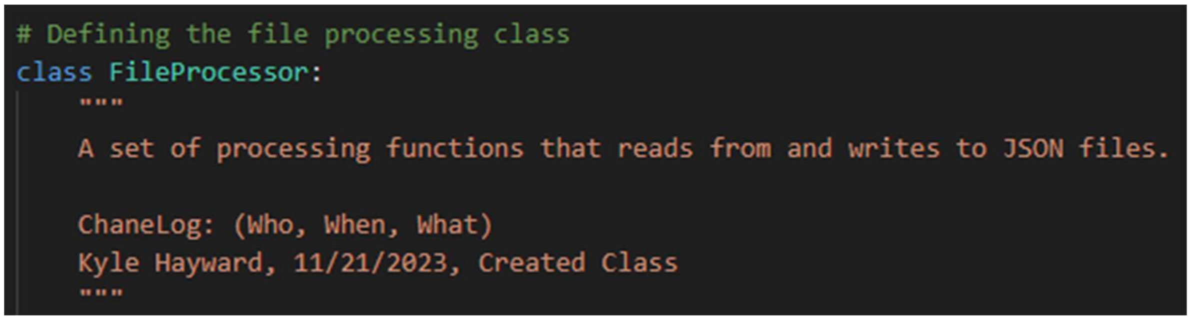


```
# Creating the constructor with private attributes including course_name
def __init__(self, first_name='', last_name='', course_name=''):
    # Passing parameter data to the Person "super" class
    super().__init__(first_name = first_name, last_name = last_name)
    self.course_name = course_name

# Creating the property getter for course_name
@property
def course_name(self):
    return self.__course_name.title()

# Creating the property setter for course_name
@course_name.setter
def course_name(self, value):
    if value.isalpha() or not value.isalpha() or value == "":
        self.__course_name = value
    else:
        raise ValueError("There was a non-specific error.")

# Overriding the default __str__() method behavior to return a coma-seperated string of data
def __str__(self):
    return f'{self.first_name},{self.last_name},{self.course_name}'
```

Figure 1.6:

Please note that I included both 'value.isalpha' and 'not value.isalpha' functions as a way to let the user enter both alphabetic and numeric data for the course name. I am certain there is a better way to accomplish this, however, I was challenged greatly in this assignment and was happy with getting this portion of the program working. I experienced with the ‘isalnum’ method as well, but since a user can enter a string with a space, this will always be false and raise an exception that I do not want.

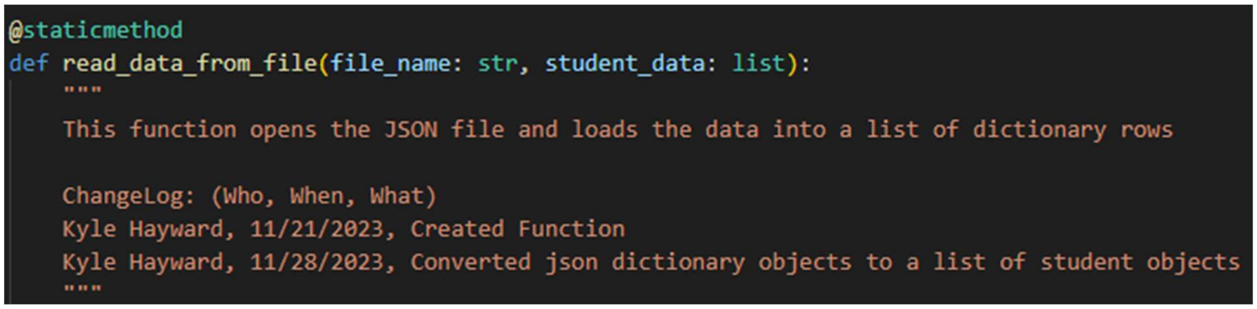
We now enter the Processing phase of the program. I start the processing phase by defining a file processor class named FileProcessor followed by a docstring summarizing what the class accomplishes. The class itself is unchanged from Assignment 06, however, the functions within do indeed change which I will cover in the upcoming documentation. (Figure 1.7)



```
# Defining the file processing class
class FileProcessor:
    """
    A set of processing functions that reads from and writes to JSON files.

    ChaneLog: (Who, When, What)
    Kyle Hayward, 11/21/2023, Created Class
    """
```

Figure 1.7: Creating the FileProcessor class with a docstring.

Using the @staticmethod, I define the file reading method with file_name and student_data as parameters. (Figure 1.8)

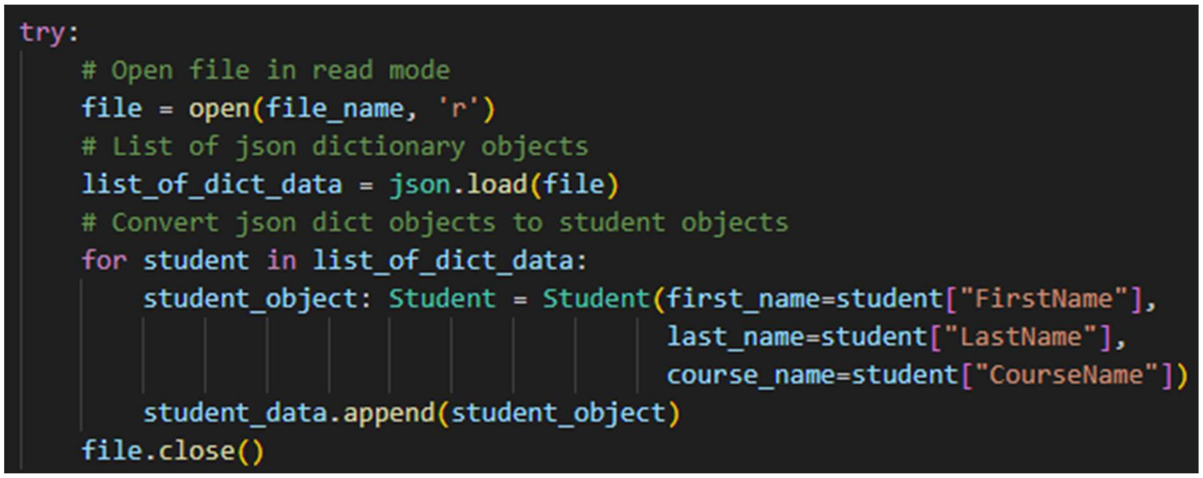


```
@staticmethod
def read_data_from_file(file_name: str, student_data: list):
    """
    This function opens the JSON file and loads the data into a list of dictionary rows

    ChangeLog: (Who, When, What)
    Kyle Hayward, 11/21/2023, Created Function
    Kyle Hayward, 11/28/2023, Converted json dictionary objects to a list of student objects
    """
```

Figure 1.8: Defining the read_data_from_file method using @staticmethod.

A few things have changed in this method from the previous assignment. First, I renamed the variable to make it clear I am using a list of json dictionary objects and also so I can use student_data with the append function.

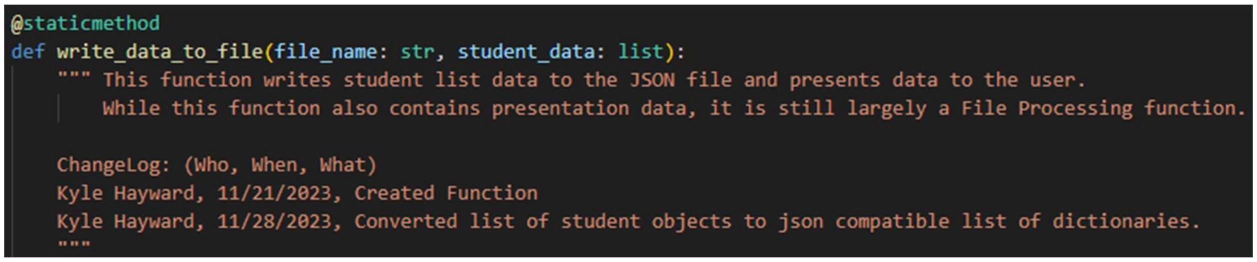
I then convert the json dictionary objects to student objects before appending that to the student list. All exception handling in this method remains the same as the previous assignment so those details will be left out in this documentation. (Figure 1.9)



```python
try:
    # Open file in read mode
    file = open(file_name, 'r')
    # List of json dictionary objects
    list_of_dict_data = json.load(file)
    # Convert json dict objects to student objects
    for student in list_of_dict_data:
        student_object: Student = Student(first_name=student["FirstName"],
                                          last_name=student["LastName"],
                                          course_name=student["CourseName"])
        student_data.append(student_object)
    file.close()
```

Figure 1.9: Using a list of json dictionary objects and converting them to student objects.

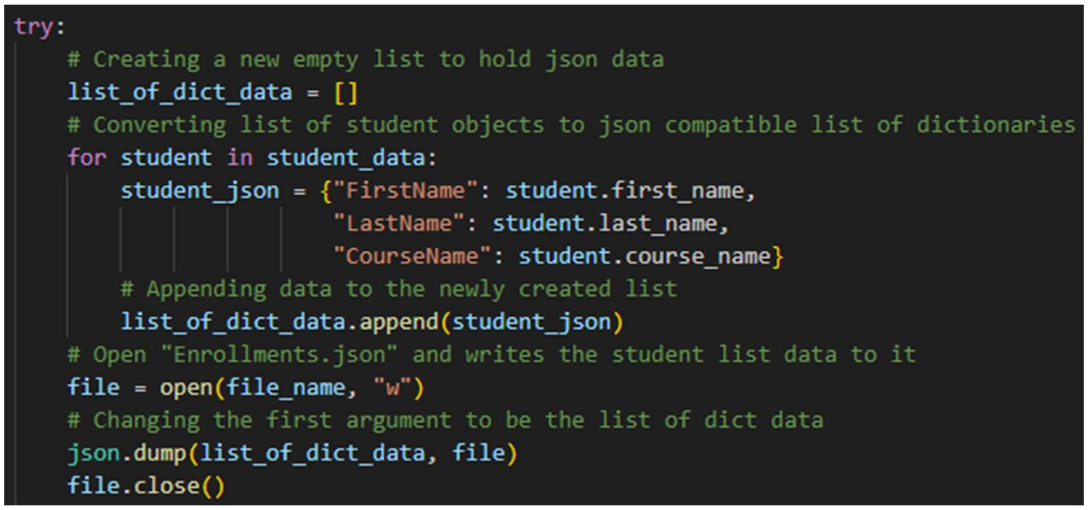
The write to file method is declared next. As with the read from file method, I use @staticmethod, two parameters, followed by a docstring. (Figure 1.10)



```python
@staticmethod
def write_data_to_file(file_name: str, student_data: list):
    """ This function writes student list data to the JSON file and presents data to the user.
        While this function also contains presentation data, it is still largely a File Processing function.

    ChangeLog: (Who, When, What)
    Kyle Hayward, 11/21/2023, Created Function
    Kyle Hayward, 11/28/2023, Converted list of student objects to json compatible list of dictionaries.
    """
```

Figure 1.10: Defining the write_data_to_file method with two parameters.

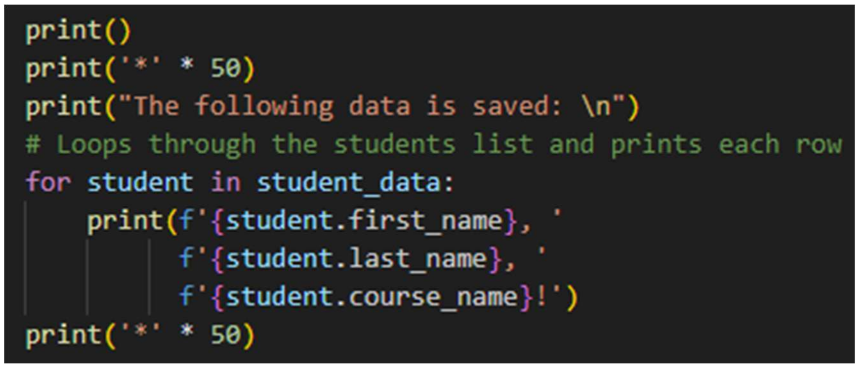
The write to file method starts with an empty list that will hold json data. I then convert the list of student objects to a json compatible list of dictionaries before appending that data to the newly created list. The “Enrollments.json” file is then opened or created in write mode, the list data is dumped into it and then closed. (Figure 1.11)



```python
try:
    # Creating a new empty list to hold json data
    list_of_dict_data = []
    # Converting list of student objects to json compatible list of dictionaries
    for student in student_data:
        student_json = {"FirstName": student.first_name,
                        "LastName": student.last_name,
                        "CourseName": student.course_name}
        # Appending data to the newly created list
        list_of_dict_data.append(student_json)
    # Open "Enrollments.json" and writes the student list data to it
    file = open(file_name, "w")
    # Changing the first argument to be the list of dict data
    json.dump(list_of_dict_data, file)
    file.close()
```

Figure 1.11: Converting student objects to json list of dictionaries and dumping data to the list.

I now loop through the students list and print out the data that has been saved. (Figure 1.12)

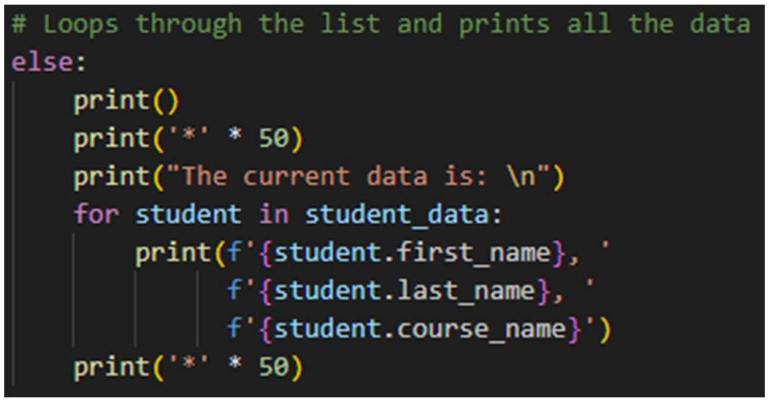


```
print()
print('*' * 50)
print("The following data is saved: \n")
# Loops through the students list and prints each row
for student in student_data:
    print(f'{student.first_name}, '
          f'{student.last_name}, '
          f'{student.course_name}!')
print('*' * 50)
```

Figure 1.12: Looping through the list and displaying the saved data.

We have now come to the presentation portion of the program. The IO class, error messages method, output menu method, and menu choice have not been edited since that last assignment. I will leave those portions of the program out of this documentation.

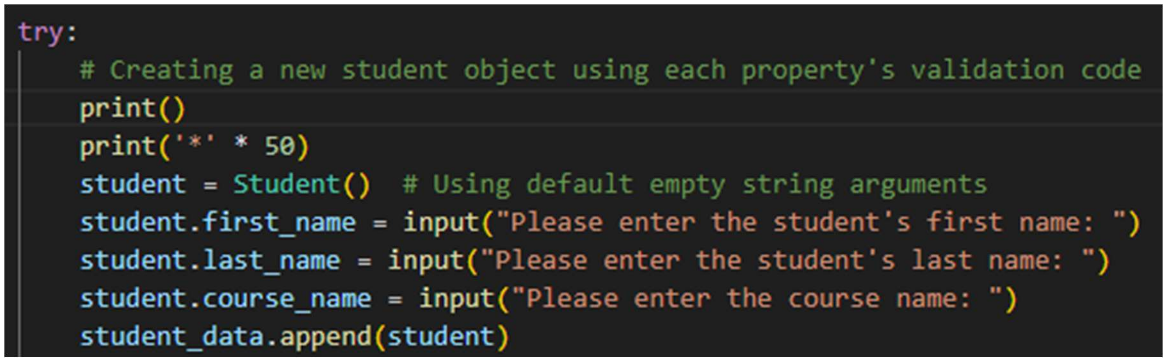
The only code that has changed in the output_student_data method is how we loop through the list. We are now accessing the student objects and do so with student.first_name, student.last_name, and student.course_name. (Figure 1.13)



```
# Loops through the list and prints all the data
else:
    print()
    print('*' * 50)
    print("The current data is: \n")
    for student in student_data:
        print(f'{student.first_name}, '
              f'{student.last_name}, '
              f'{student.course_name}')
    print('*' * 50)
```

Figure 1.13: Looping through the student objects to print data contained in "Enrollments.json".

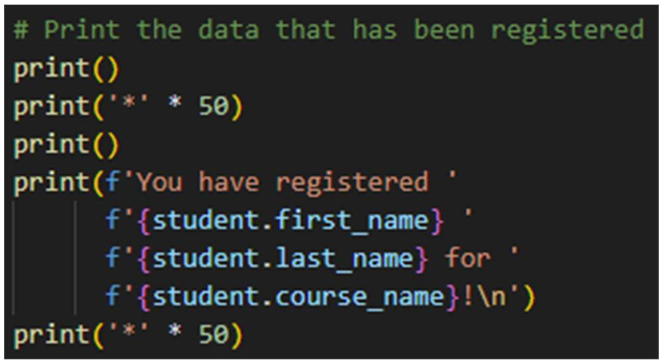
I have updated the input_student_data method to use student objects rather than dictionaries. This is accomplished by calling the Student class using empty string arguments. I then create new student objects by using each property's validation code before appending the data. (Figure 1.14)



```
try:
    # Creating a new student object using each property's validation code
    print()
    print('*' * 50)
    student = Student()  # Using default empty string arguments
    student.first_name = input("Please enter the student's first name: ")
    student.last_name = input("Please enter the student's last name: ")
    student.course_name = input("Please enter the course name: ")
    student_data.append(student)
```

Figure 1.14: Updating the input data method to use student objects rather than dictionaries.

The student object data is then accessed and displayed to the user with an f-string formatted print function. (Figure 1.15)

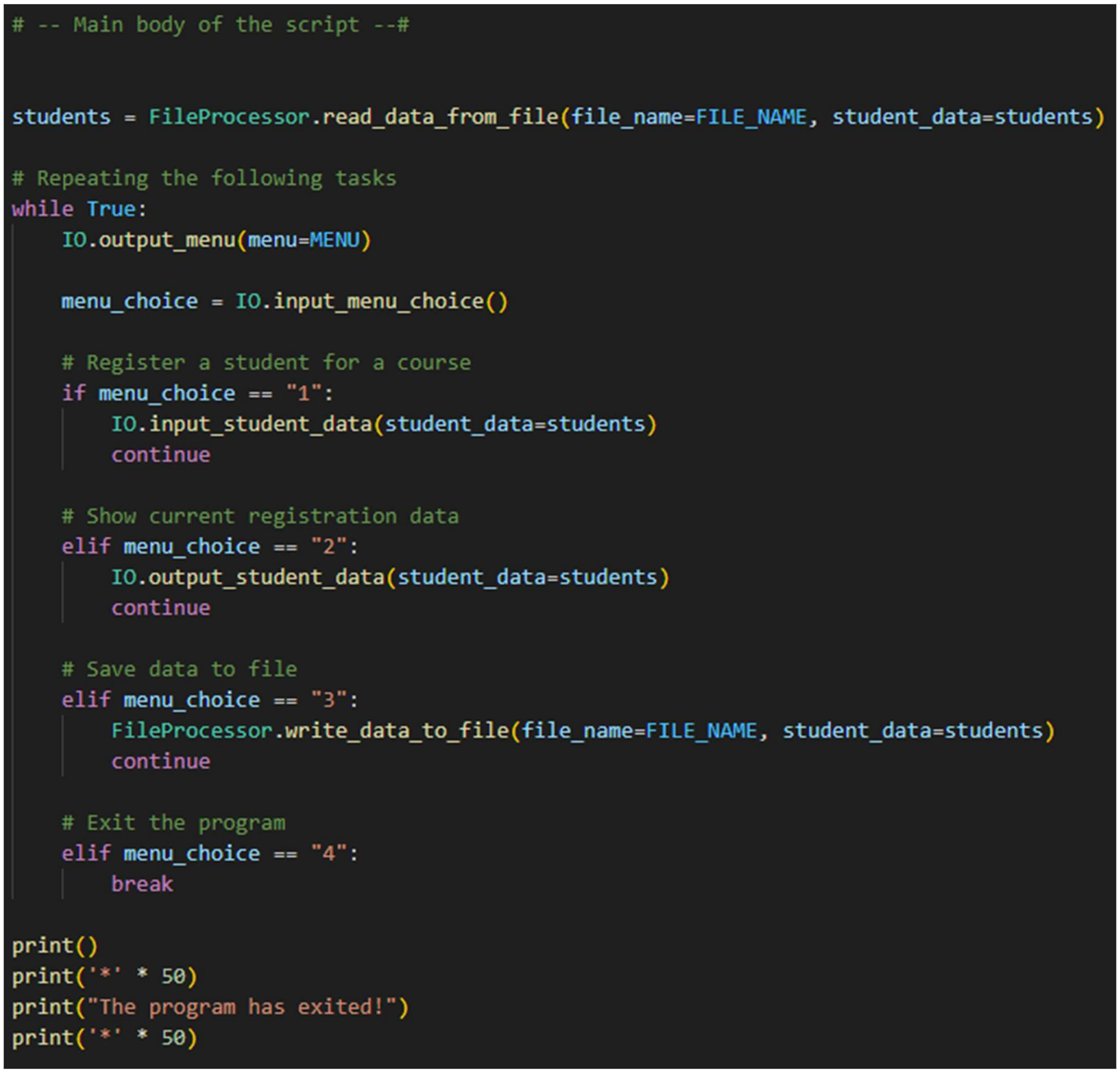


```
# Print the data that has been registered
print()
print('*' * 50)
print()
print(f'You have registered '
        f'{student.first_name} '
        f'{student.last_name} for '
        f'{student.course_name}!\n')
print('*' * 50)
```

Figure 1.15: Displaying currently registered data back to the user.

This marks the end of class and function/method definitions.

The main body of the script has not changed from that of Assignment 06. Figure 1.16 is for review purposes only as the structure of the while loop remains the same in this assignment.



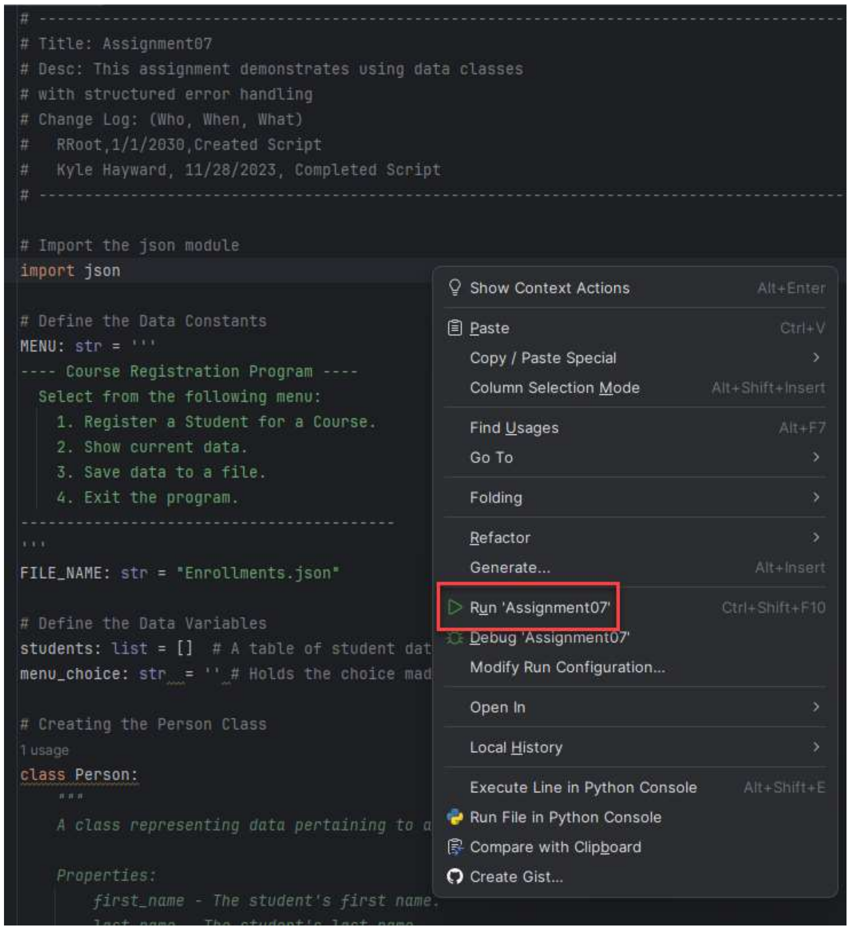
```python
# -- Main body of the script --#


students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)

# Repeating the following tasks
while True:
    IO.output_menu(menu=MENU)

    menu_choice = IO.input_menu_choice()

    # Register a student for a course
    if menu_choice == "1":
        IO.input_student_data(student_data=students)
        continue

    # Show current registration data
    elif menu_choice == "2":
        IO.output_student_data(student_data=students)
        continue

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

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

print()
print('*' * 50)
print("The program has exited!")
print('*' * 50)
```

Figure 1.16: Reviewing the main body of the script which has not changed since the last assignment.

The next sections will cover launching the program within the PyCharm IDE and running the program to completion within the Windows OS Command Prompt.

Launching the Program – PyCharm

Launch the PyCharm IDE and open ‘Assignment07.py’. Right-click anywhere within the program code and click **Run ‘Assignment07’.** (Figure 2.1)

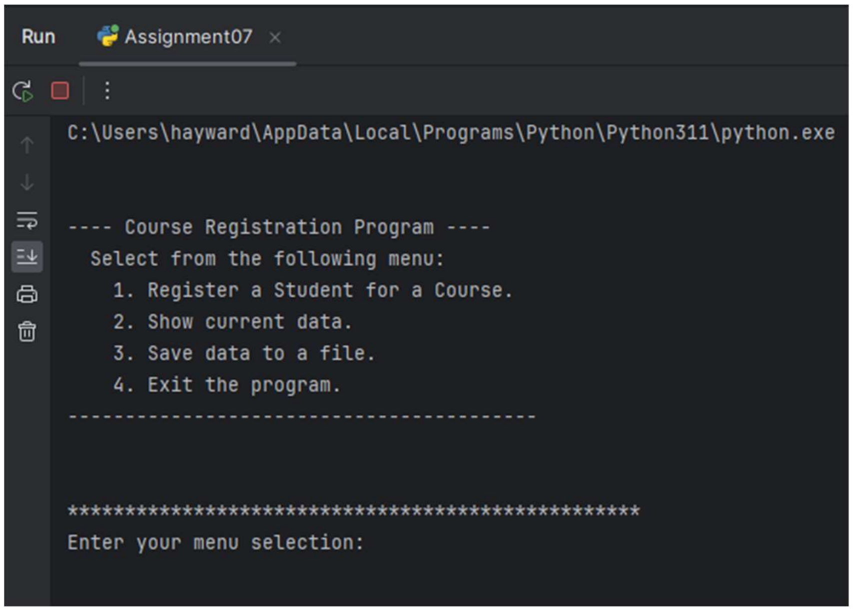


```
# ---------------------------------------------------------------------------
# Title: Assignment07
# Desc: This assignment demonstrates using data classes
# with structured error handling
# Change Log: (Who, When, What)
#   RRoot,1/1/2030,Created Script
#   Kyle Hayward, 11/28/2023, Completed Script
# ---------------------------------------------------------------------------

# Import the json module
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 dat
menu_choice: str  = '' # Holds the choice mad

# Creating the Person Class
1 usage
class Person:
    """
    A class representing data pertaining to a

    Properties:
        first_name - The student's first name.
```

Show Context Actions Alt+Enter
Paste Ctrl+V
Copy / Paste Special
Column Selection Mode Alt+Shift+Insert
Find Usages Alt+F7
Go To
Folding
Refactor
Generate... Alt+Insert
Run 'Assignment07' Ctrl+Shift+F10
Debug 'Assignment07'
Modify Run Configuration...
Open In
Local History
Execute Line in Python Console Alt+Shift+E
Run File in Python Console
Compare with Clipboard
Create Gist...

Figure 2.1: Run 'Assignment07' within the PyCharm IDE.

Assignment07.py successfully launches in PyCharm. (Figure 2.2)



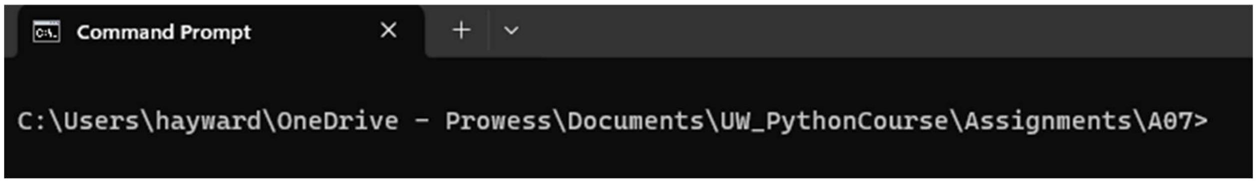
```
C:\Users\hayward\AppData\Local\Programs\Python\Python311\python.exe


---- 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 selection:
```

Figure 2.2: Assignment07.py program running in the PyCharm IDE.

Running the Program – Command Prompt

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

Launch the Command Prompt and navigate to the location of the Assignment07.py program. (Figure 3.1)

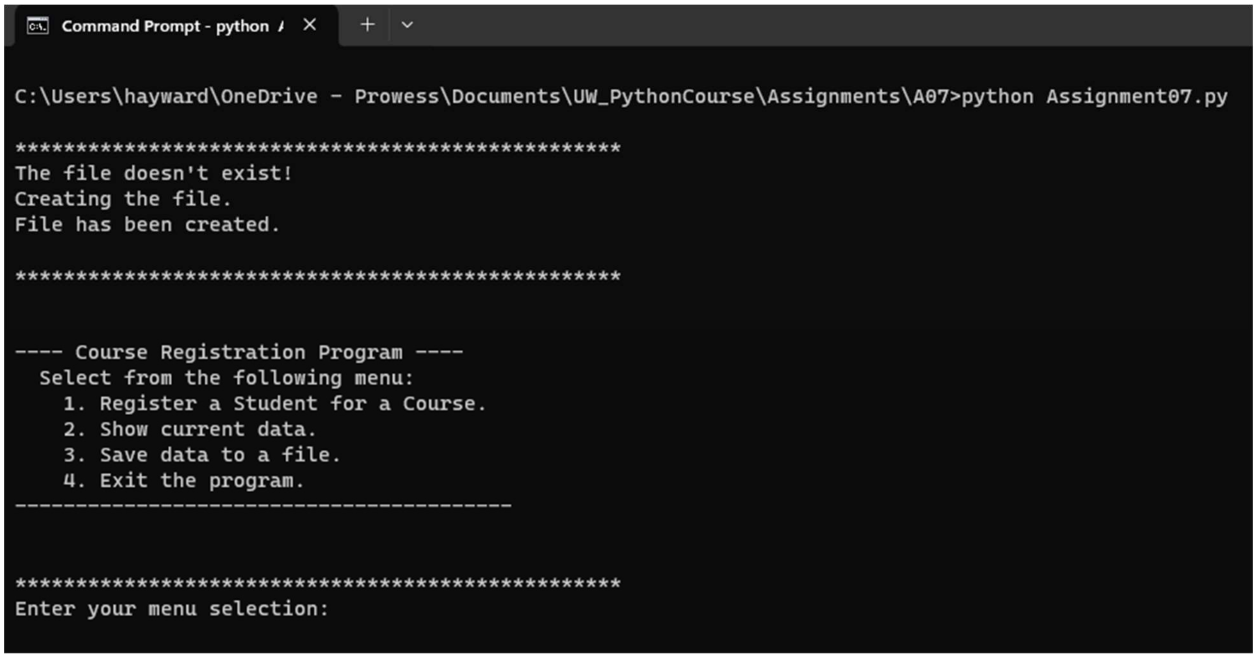


```
C:\Users\hayward\OneDrive - Prowess\Documents\UW_PythonCourse\Assignments\A07>
```

Figure 3.1: Navigating to the program location within the Command Prompt.

Type ‘python Assignment07.py’ and hit Enter on the keyboard.

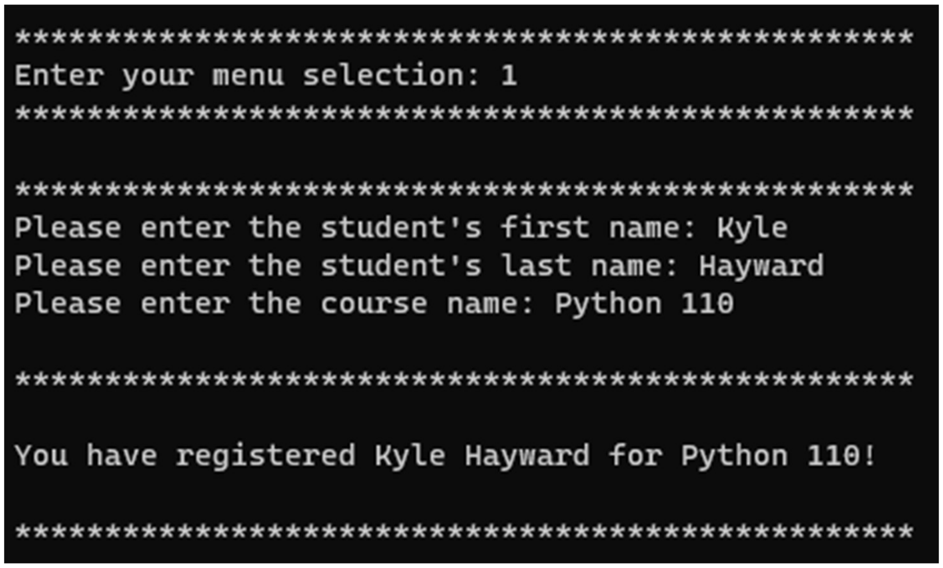
I intentionally started the program knowing the “Enrollments.json” file doesn’t exist. The program’s exception handling catches this, creates the file, and notifies the user. (Figure 3.2)



```
C:\Users\hayward\OneDrive - Prowess\Documents\UW_PythonCourse\Assignments\A07>python Assignment07.py

**************************************************
The file doesn't exist!
Creating the file.
File has been created.

**************************************************


---- 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 selection:
```

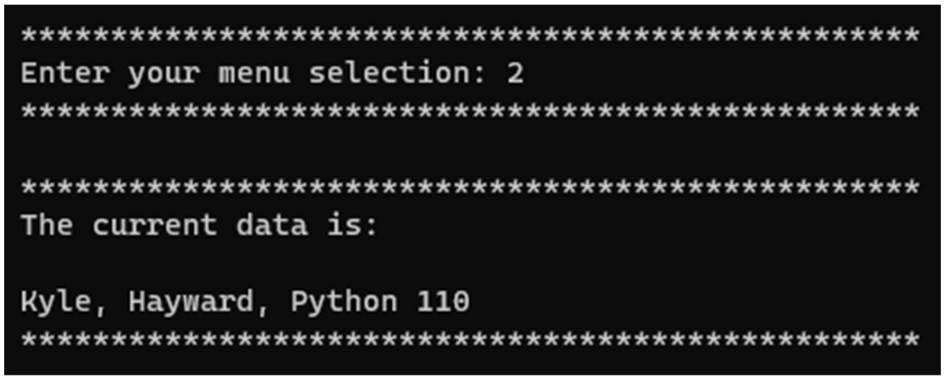
Figure 3.2: Running the program within the Command Prompt and demonstrating file creation.

Figures 3.3 to 3.6 will show the program running to completion after selecting all menu options in succession.



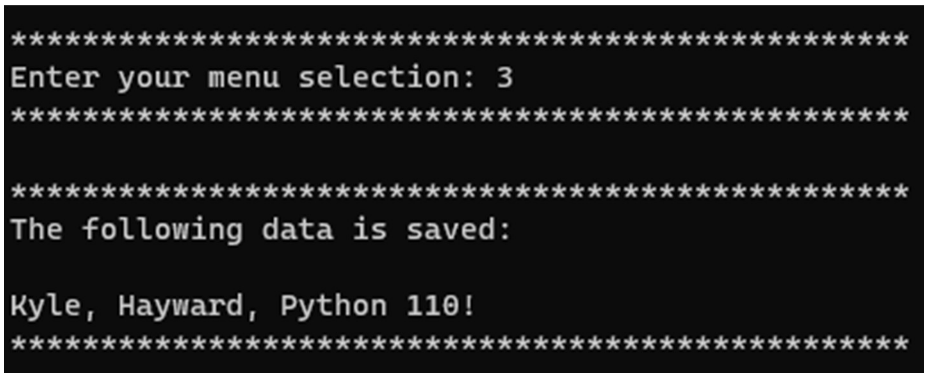
```
**************************************************
Enter your menu selection: 1
**************************************************

**************************************************
Please enter the student's first name: Kyle
Please enter the student's last name: Hayward
Please enter the course name: Python 110

**************************************************

You have registered Kyle Hayward for Python 110!

**************************************************
```

Figure 3.3: Program execution after user selects menu option 1.



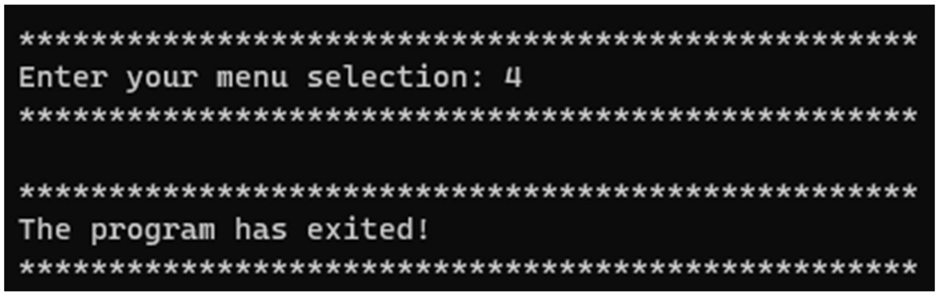
```
**************************************************
Enter your menu selection: 2
**************************************************

**************************************************
The current data is:

Kyle, Hayward, Python 110
**************************************************
```

Figure 3.4: Program execution after user selects menu option 2.



```
**************************************************
Enter your menu selection: 3
**************************************************

**************************************************
The following data is saved:

Kyle, Hayward, Python 110!
**************************************************
```

Figure 3.5: Program execution after user selects menu option 3.



```
*************************************************
Enter your menu selection: 4
*************************************************

*************************************************
The program has exited!
*************************************************
```

Figure 3.6: Program execution after user selects menu option 4.

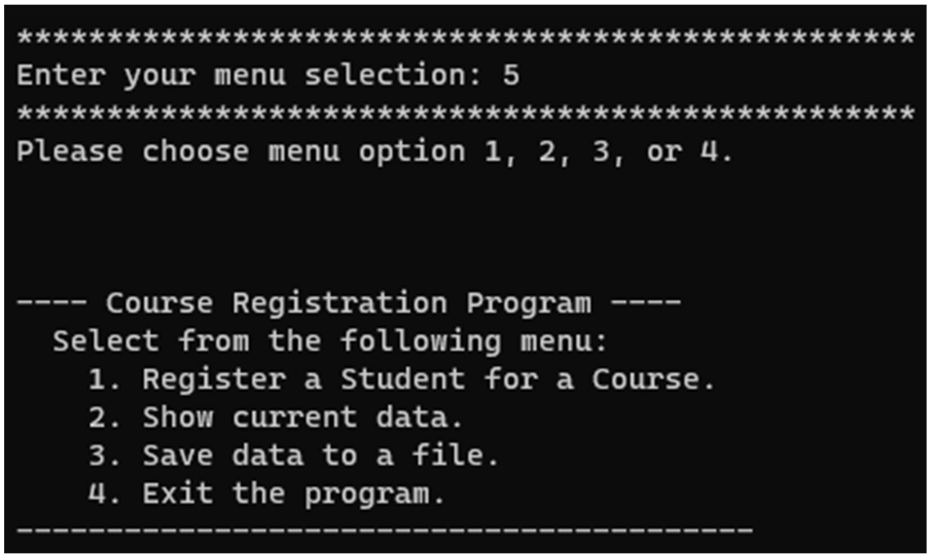
The program has run to completion within the Command Prompt.

The next section will demonstrate some exception handling techniques that have been implemented to catch potential issues introduced during user input.

Exception Handling

What happens when the user inputs data that is not valid or the program does not understand? For instance, if the user enters a menu option that is not either 1, 2, 3, or 4.

Figure 4.1 demonstrates the exception handling that addressing just such an issue.

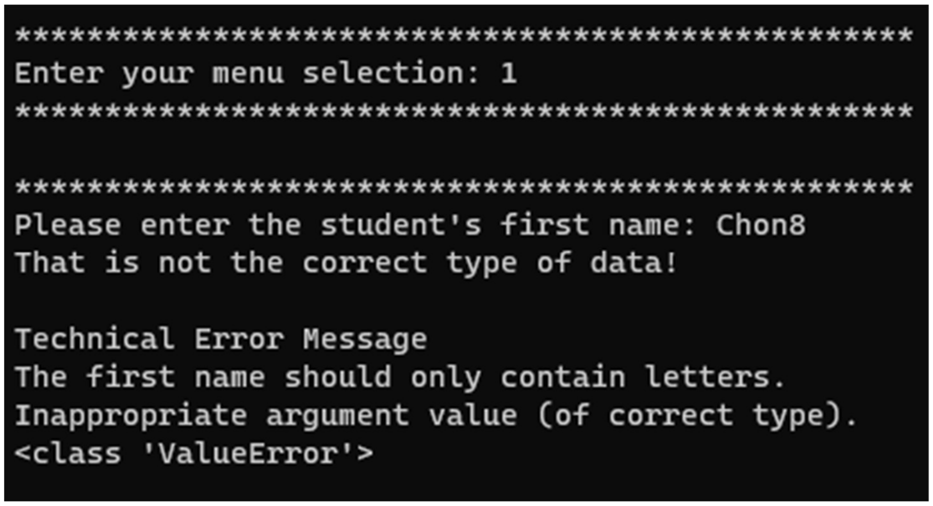


```
*************************************************
Enter your menu selection: 5
*************************************************
Please choose menu option 1, 2, 3, or 4.



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

Figure 4.1: Exception handling when an invalid menu option is selected by the user.

As you can see in Figure 4.1, the program recognizes that an invalid menu option has been chosen. The program notifies the user to choose menu option 1, 2, 3, or 4 and displays the Course Registration Menu again.

Suppose a user enters a typo when entering a student's first or last name. For instance, a student's name is Chong, however, the user accidentally enters Chon8. Figure 4.2 demonstrates how the program handles this type of exception.



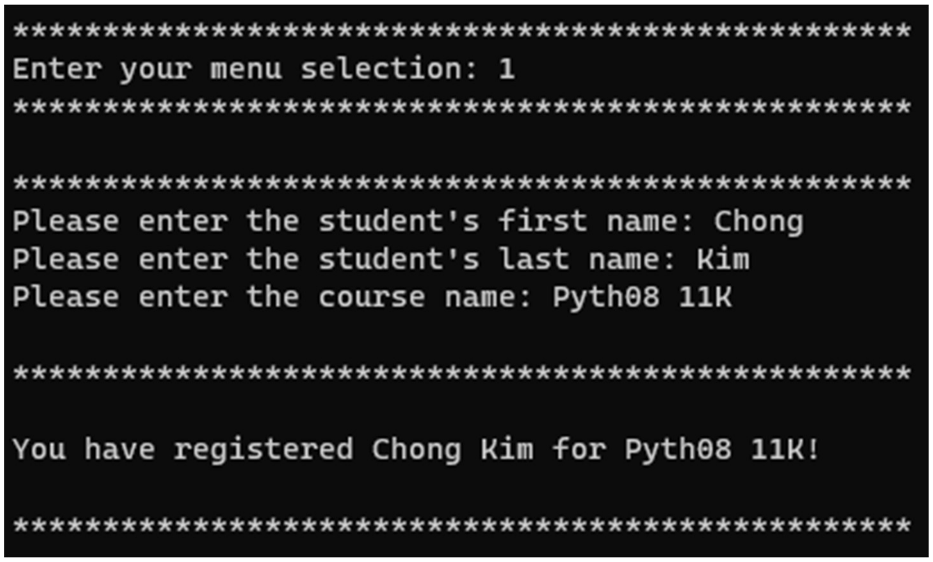
```
*************************************************
Enter your menu selection: 1
*************************************************

*************************************************
Please enter the student's first name: Chon8
That 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'>
```

Figure 4.2: The program throws a ValueError because the first name includes a numeric value.

Within the program, behind the scenes so to speak, the isalpha() function is implemented to catch incorrectly entered data. Since the first name entered contains characters other than alphabetic characters, the program displays the error and notifies the user.

It is important to note how this program handles the course name input. In the previous assignment I was able to utilize the 'if not course_name' loop to ensure the user entered at least one character. However, since I updated the classes and methods to include error handling within the property defining, that exception handling is no longer working.

Until I can determine a more dynamic way of handling course name user input, I have essentially allowed the user to enter any combination of characters without erroring out. This will allow the user to enter both alphabetic, numeric, and blank characters such as spaces when input course data. (Figure 4.3)



```
*************************************************
Enter your menu selection: 1
*************************************************

*************************************************
Please enter the student's first name: Chong
Please enter the student's last name: Kim
Please enter the course name: Pyth08 11K

*************************************************

You have registered Chong Kim for Pyth08 11K!

*************************************************
```

Figure 4.3: The program allows for numeric data as well as alphabetical data.

The isalpha() function does not work for me here because I want the user to be able to enter multiple types of data. If the isalpha() function is implemented, the program would throw an exception if Python 110 was entered. Not only because 110 was added, but because isalpha() would return as False as soon as it recognizes the blank space after Python. While the function is sufficient for first and last name, it is not how I want to program to behave for course name.

Summary

This assignment was significantly more challenging than the previous six assignments. I need a lot more work with constructors and how they differ from variable defining. The 'self' attribute is also quite confusing to me.

Along with constructors, the getter and setter properties are a bit perplexing. Without the exceptional course documentation, I would be completely lost with these concepts.

The conversion of json dictionaries to objects is also proving challenging for me to comprehend. I'm hoping as the course continues, I will grasp the concepts with further understanding.

As requested, this program's code has been uploaded to my GitHub repo:
https://github.com/HaywardUW/IntroToProg-Python-Mod07