Derek Larson

11/26/2023

IT FDN 110: Foundations of Programming-Python

Assignment 07

github.com/derkrylar99/IntroToProg-Python

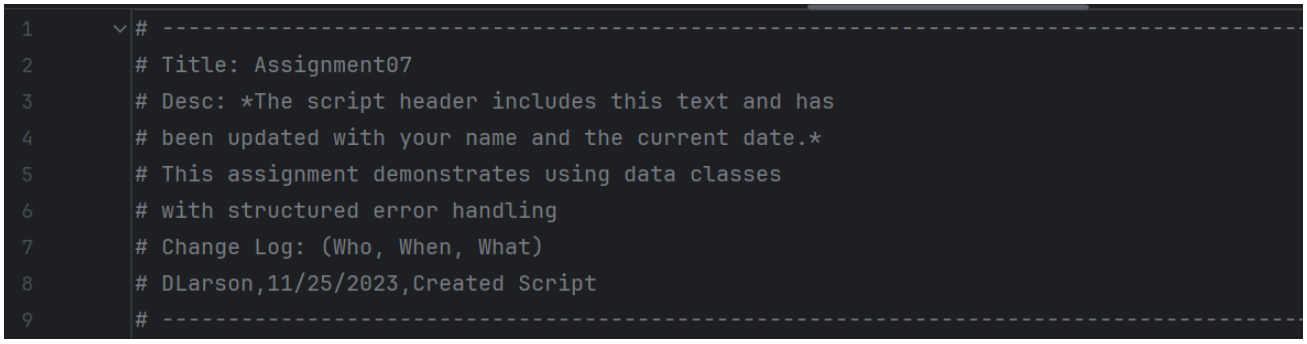
Class Types, Objects, and Inheritance

Introduction

In this module we were introduced to the concept of Inheritance in addition to working with Objects versus table cell data. Larger concepts such as Encapsulation and Abstraction are put into practice through the use of Classes and sub-Classes, where inheritance allows for code to be contained or "hidden" in a higher level for re-use and related concepts can be grouped together for easy modification.

Creating the Program

- Opened and reviewed starter file "Assignment07-Starter.py"
    - Diff'd against previous submission "Assignment06.py"
    - Saved previous submission "Assignment06.py" as new file "Assignment07.py"
    - Altered "Assignment07.py" to incorporate TODO list and verify functional code setup will work for the assignment tasks
    - Updated Comment Header script

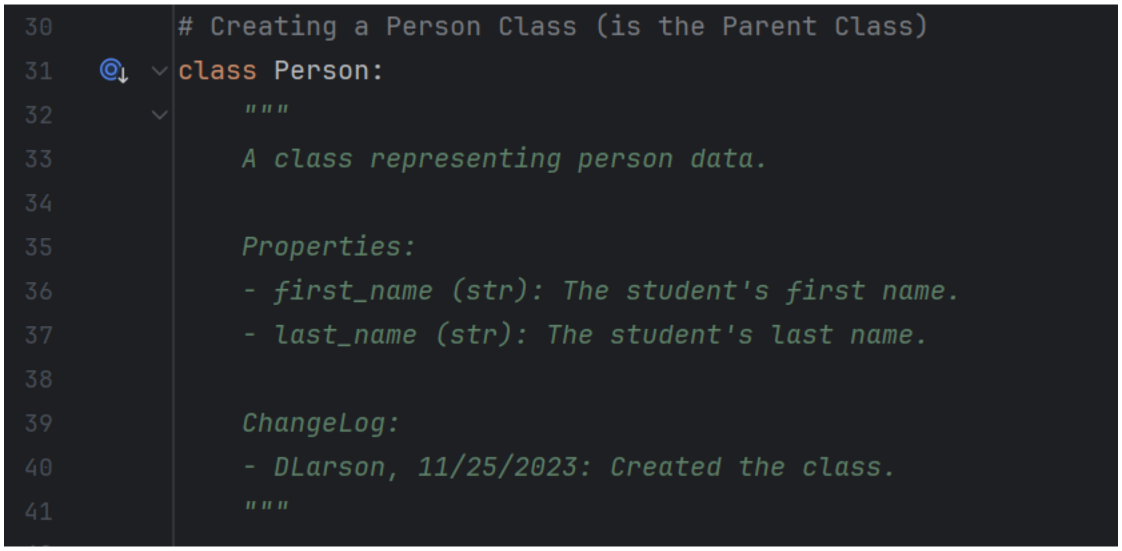


```
# ------------------------------------------------------------------------------
# Title: Assignment07
# Desc: *The script header includes this text and has
# been updated with your name and the current date.*
# This assignment demonstrates using data classes
# with structured error handling
# Change Log: (Who, When, What)
# DLarson,11/25/2023,Created Script
# ------------------------------------------------------------------------------
```

Figure 1.1: Header information updated when Assignment06 is edited to become the basis for Assignment07.

Parent Class : Person

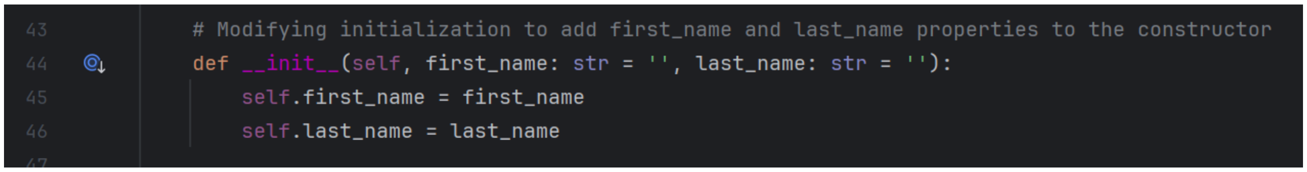
- Created a new Person parent class and Student child (sub) class:



```python
# Creating a Person Class (is the Parent Class)
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:
    - DLarson, 11/25/2023: Created the class.
    """
```

Figure 1.2: New Parent Class "Person" is defined with a Docstring description.

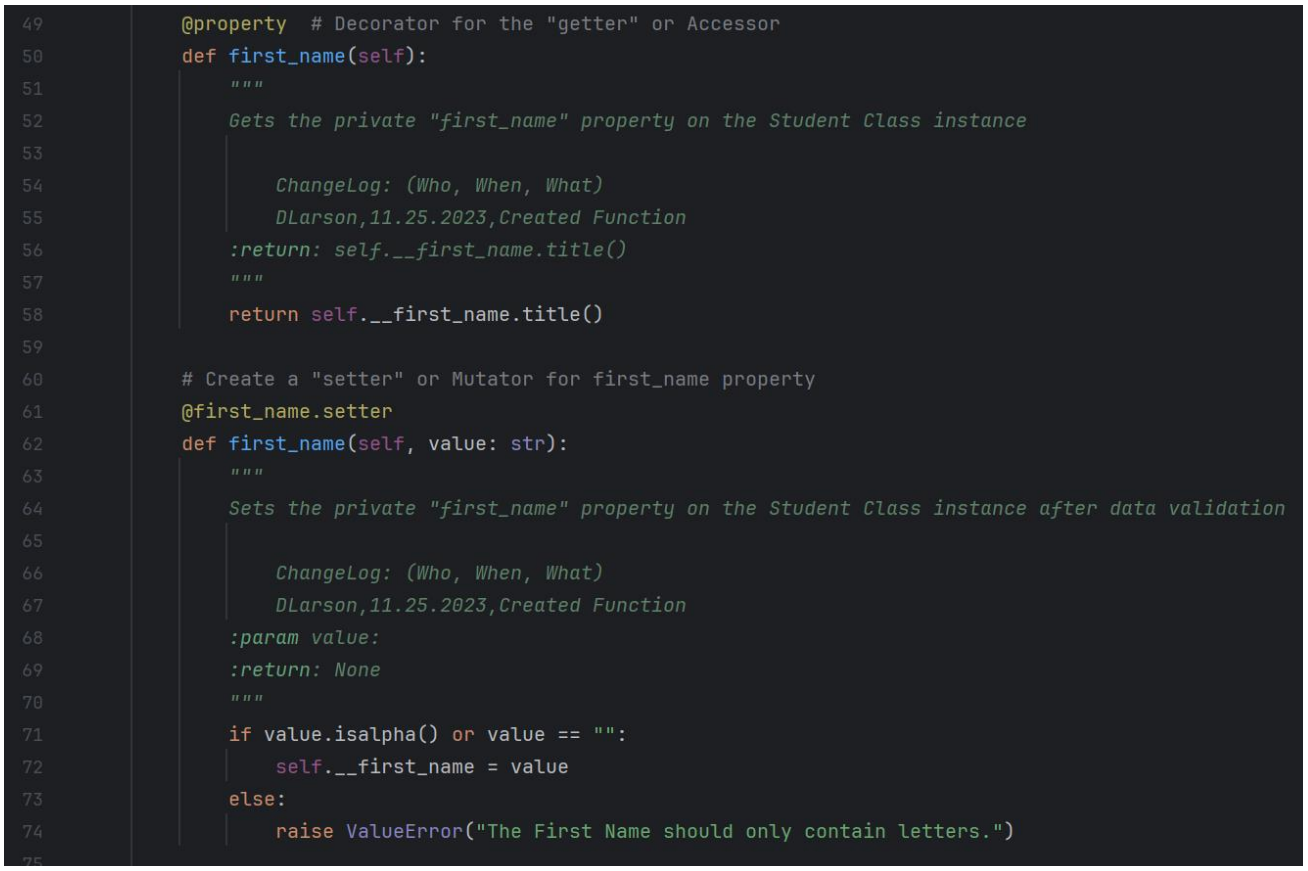
- Person class has properties first_name and last_name
  - Modified default initialization to add the new properties



```python
# Modifying initialization to add first_name and last_name properties to the constructor
def __init__(self, first_name: str = '', last_name: str = ''):
    self.first_name = first_name
    self.last_name = last_name
```

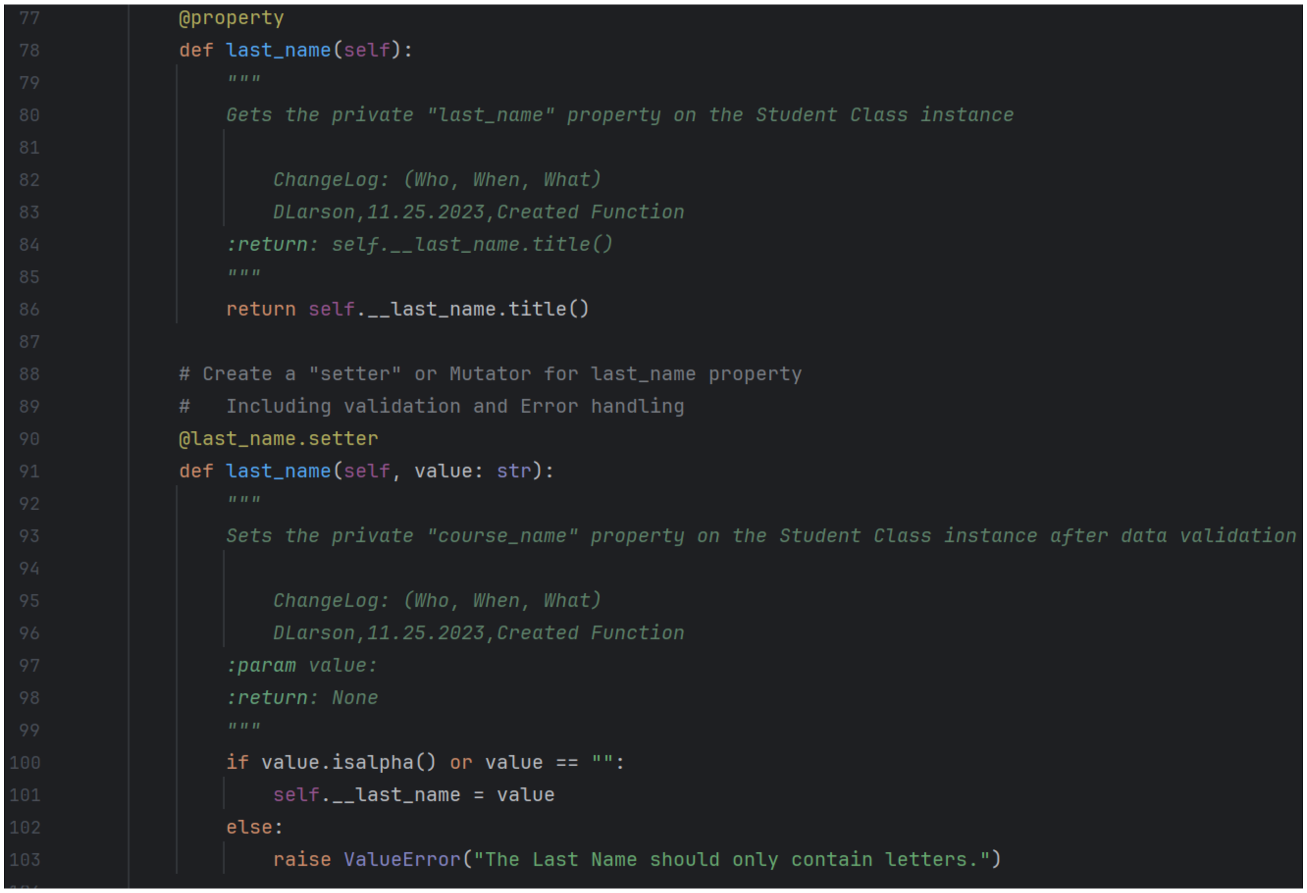
Figure 1.3: Modifying the default "initialization" to include the new parameters.

  - Each variable first_name and last_name has a dedicated "getter" or Accessor and "setter" or Mutator method



```python
    @property  # Decorator for the "getter" or Accessor
    def first_name(self):
        """
        Gets the private "first_name" property on the Student Class instance

            ChangeLog: (Who, When, What)
            DLarson,11.25.2023,Created Function
        :return: self.__first_name.title()
        """
        return self.__first_name.title()

    # Create a "setter" or Mutator for first_name property
    @first_name.setter
    def first_name(self, value: str):
        """
        Sets the private "first_name" property on the Student Class instance after data validation

            ChangeLog: (Who, When, What)
            DLarson,11.25.2023,Created Function
        :param value:
        :return: None
        """
        if value.isalpha() or value == "":
            self.__first_name = value
        else:
            raise ValueError("The First Name should only contain letters.")
```

Figure 1.4: Defining the "getter" and "setter" for the protected first_name variable.

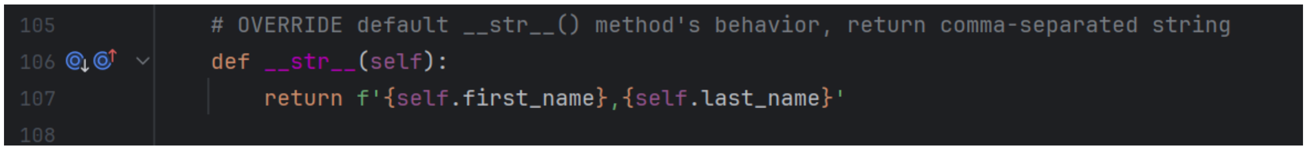


```python
    @property
    def last_name(self):
        """
        Gets the private "last_name" property on the Student Class instance

            ChangeLog: (Who, When, What)
            DLarson,11.25.2023,Created Function
        :return: self.__last_name.title()
        """
        return self.__last_name.title()

    # Create a "setter" or Mutator for last_name property
    #   Including validation and Error handling
    @last_name.setter
    def last_name(self, value: str):
        """
        Sets the private "course_name" property on the Student Class instance after data validation

            ChangeLog: (Who, When, What)
            DLarson,11.25.2023,Created Function
        :param value:
        :return: None
        """
        if value.isalpha() or value == "":
            self.__last_name = value
        else:
            raise ValueError("The Last Name should only contain letters.")
```

Figure 1.5: The "getter" and "setter" functions for last_name, which all include data validation and error handling.

- Accessor returns the instanced name in title-casing
- Mutator contains validation to check if it's valid input, and if so will assign the instanced property
    - "isalpha()" will only return True if the input string contains ONLY letters
    - This will filter out any numbers/symbols/invalid characters for data integrity

- Reset __str__() method default behavior to return a readable, comma-separated string
    - This helps with readability when requesting object information/identification

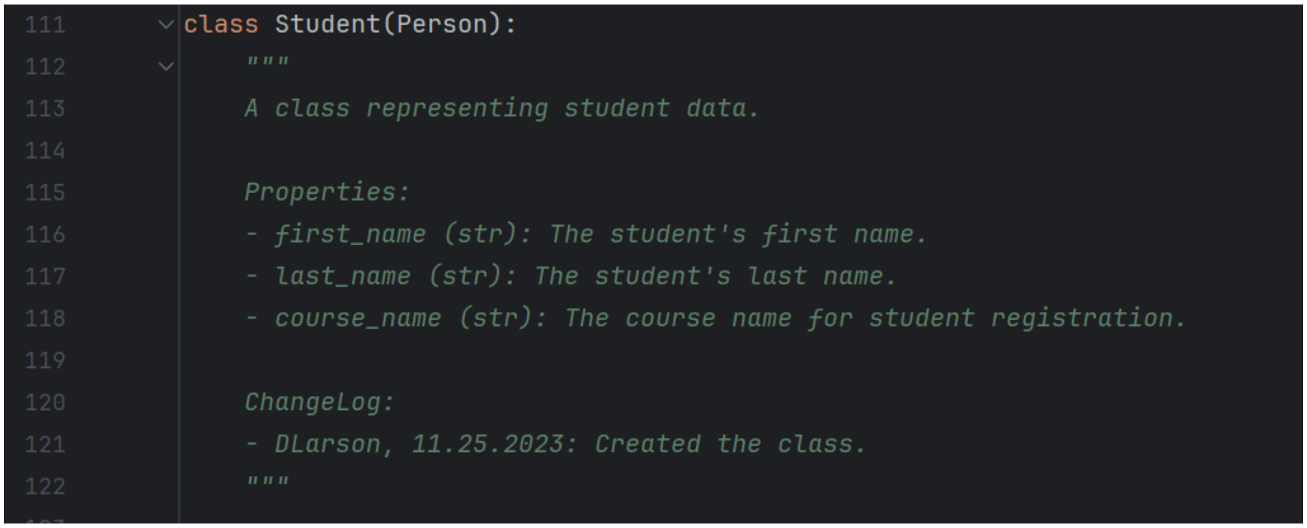


```python
    # OVERRIDE default __str__() method's behavior, return comma-separated string
    def __str__(self):
        return f'{self.first_name},{self.last_name}'
```

Figure 1.6: Overriding the default initialization of the Parent Class to add the new parameter.

        - F-string formatting using self.first_name and self.last_name

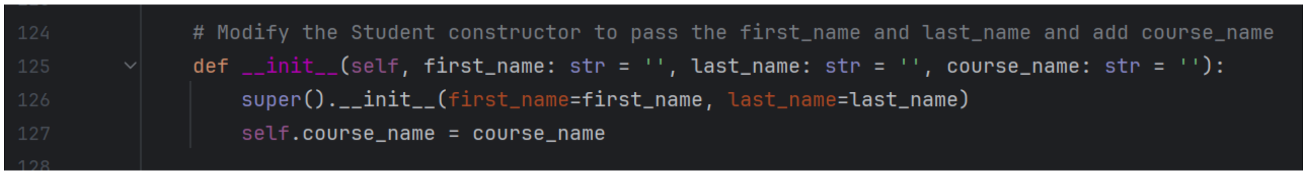
Child Class : Student (Derived from Parent Class "Person")



```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 name for student registration.

    ChangeLog:
    - DLarson, 11.25.2023: Created the class.
    """
```

Figure 1.7: Defining "Student" Class, the Sub-Class (Child) of Person.

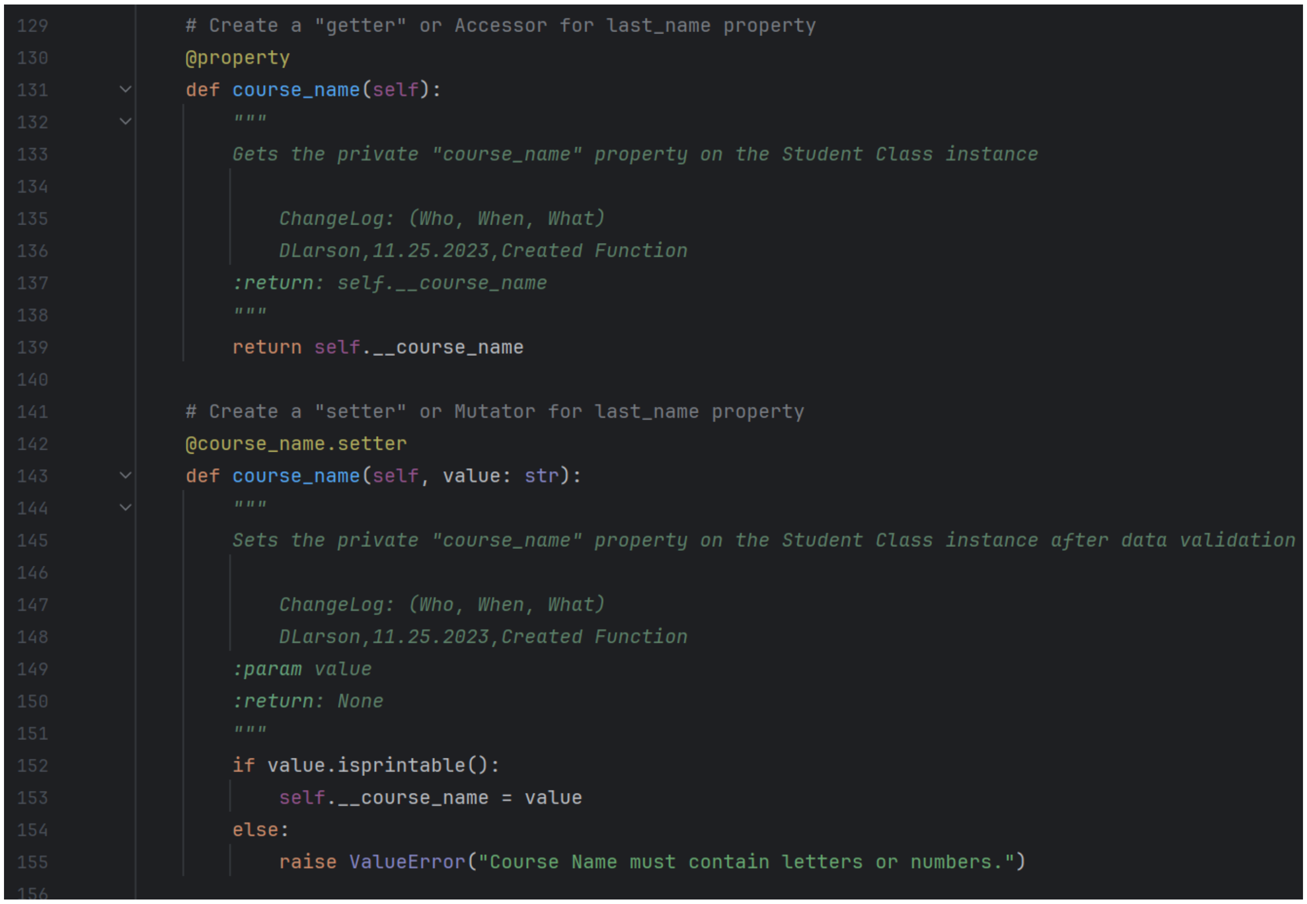
- Student class inherits first_name and last_name from parent Person class
    - Modify the default initialization and reference the parent class properties
    - Adds new property "course_name"



```python
    # Modify the Student constructor to pass the first_name and last_name and add course_name
    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 1.8: Adding the course_name variable to the initialization.

        - course_name has a dedicated "getter" or Accessor and "setter" or Mutator method

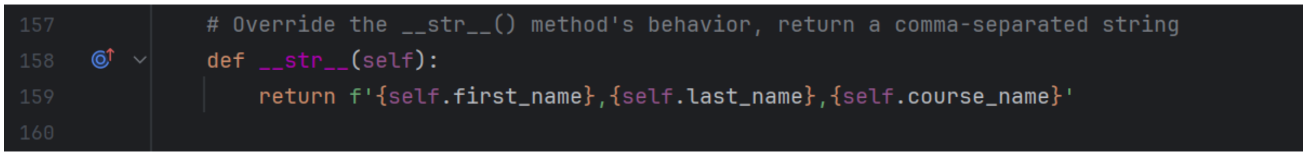


```
    # Create a "getter" or Accessor for last_name property
    @property
    def course_name(self):
        """
        Gets the private "course_name" property on the Student Class instance

            ChangeLog: (Who, When, What)
            DLarson,11.25.2023,Created Function
        :return: self.__course_name
        """
        return self.__course_name

    # Create a "setter" or Mutator for last_name property
    @course_name.setter
    def course_name(self, value: str):
        """
        Sets the private "course_name" property on the Student Class instance after data validation

            ChangeLog: (Who, When, What)
            DLarson,11.25.2023,Created Function
        :param value
        :return: None
        """
        if value.isprintable():
            self.__course_name = value
        else:
            raise ValueError("Course Name must contain letters or numbers.")
```

Figure 1.9: Adding "getter" and "setter" for the course_name variable including validation/error handling.

- Accessor returns the instanced string
- Mutator contains validation to check if it's valid input, and if so will assign the instanced property
    - "isprintable()" will return True if it is a letter, number, character/symbol, empty space, or any other valid printable character

○ Reset Student __str__() method default behavior to return a readable, comma-separated string adding new property course_name



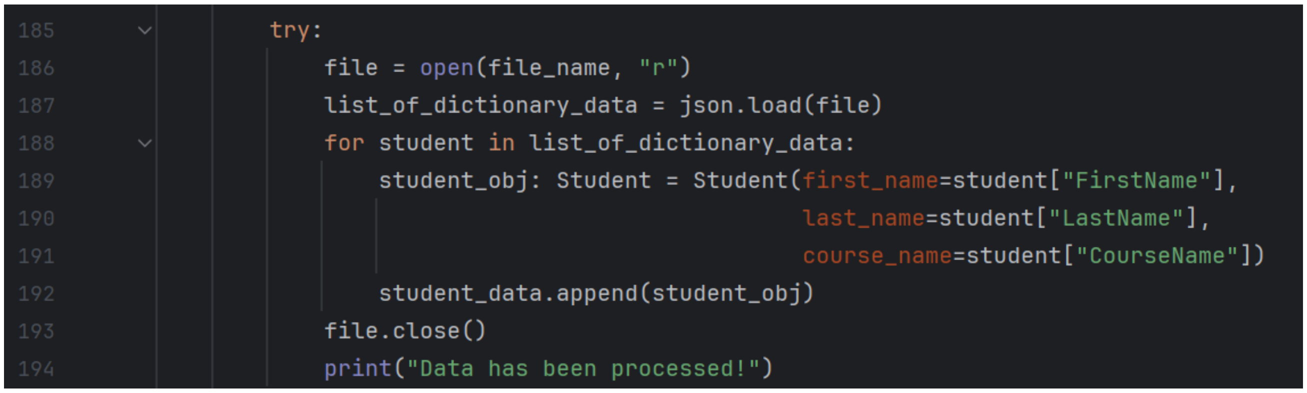
```
    # Override the __str__() method's behavior, return a comma-separated string
    def __str__(self):
        return f'{self.first_name},{self.last_name},{self.course_name}'
```

Figure 1.10: Improved formatting for the user by overriding the default String method to return an f-string formatted message.

        - F-string formatting using self.first_name , self.last_name, and self.course_name

Converting Data to use Student Object Instances

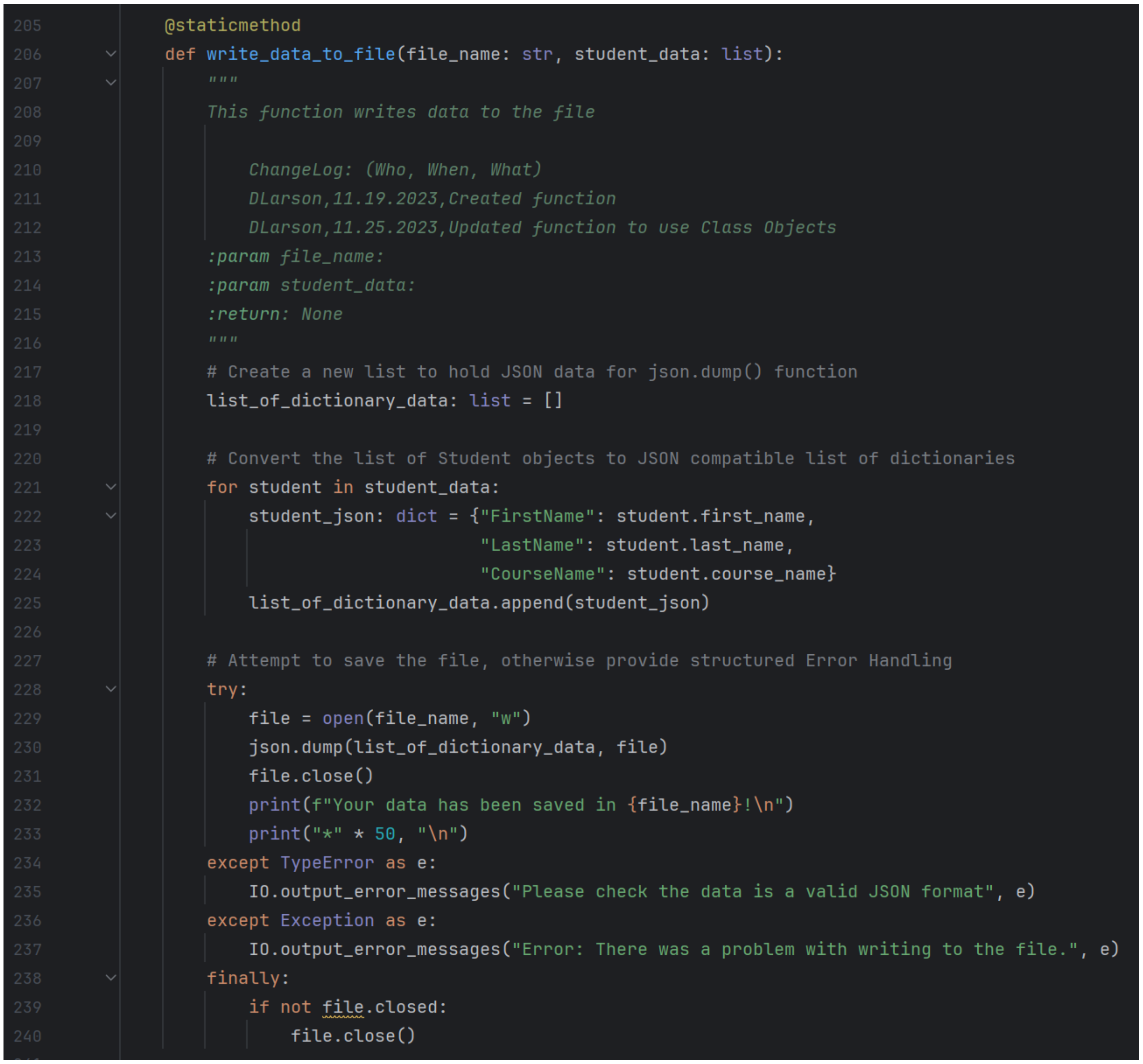
- student_data is now a list of objects NOT a dict, json needs to be converted from dictionary to list of objects
    - update method read_data_from_file
        - comment out student_data = json.load(file)
        - instead load json into new variable list_of_dictionary_data to convert
        - for loop to iterate through list_of_dictionary_data and convert each dictionary row entry into an object instance of the Student Class
            - use dictionary keys to set class properties
            - append each object to the student_data list



```
        try:
            file = open(file_name, "r")
            list_of_dictionary_data = json.load(file)
            for student in list_of_dictionary_data:
                student_obj: Student = Student(first_name=student["FirstName"],
                                               last_name=student["LastName"],
                                               course_name=student["CourseName"])
                student_data.append(student_obj)
            file.close()
            print("Data has been processed!")
```

Figure 1.11: Final code with changes to load the JSON into a dictionary list before converting to Student Object instances and appending to a new list of Objects.

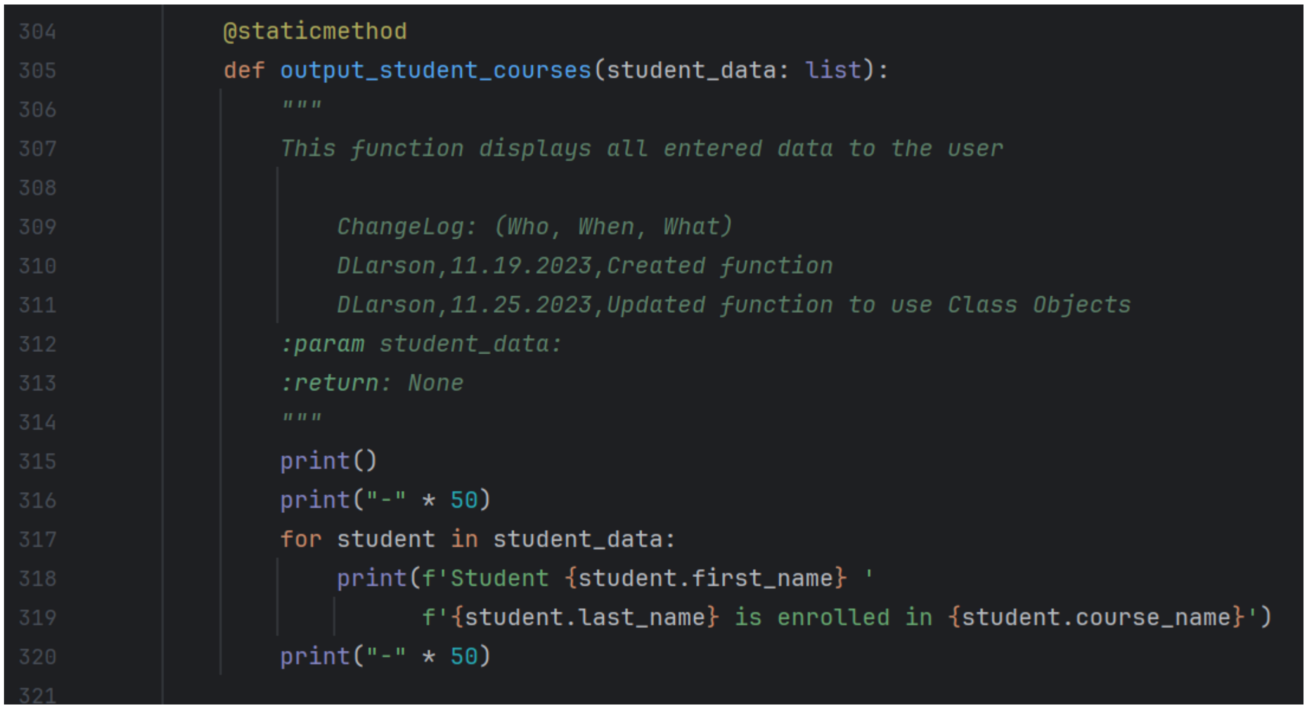
- to save json, the list of objects needs to be converted back to a dictionary
    - update method write_data_to_file
        - comment out json.dump(student_data, file)
            - this is because the student_data list contains objects, not dictionaries
        - create new variable as empty list to hold converted data
            - list_of_dictionary_data: list = []
        - convert the student_data list of Student instance objects into dictionaries with a “for” loop
            - each object “student” will use the student class properties to set dictionary Keys defined in the new dictionary variable student_json
            - each converted dictionary will be added to the new list of dictionaries list_of_dictionary_data
        - use the list_of_dictionary_data to write to the JSON using the dump function
            - json.dump(list_of_dictionary_data, file)



```python
    @staticmethod
    def write_data_to_file(file_name: str, student_data: list):
        """
        This function writes data to the file

            ChangeLog: (Who, When, What)
            DLarson,11.19.2023,Created function
            DLarson,11.25.2023,Updated function to use Class Objects
        :param file_name:
        :param student_data:
        :return: None
        """
        # Create a new list to hold JSON data for json.dump() function
        list_of_dictionary_data: list = []

        # Convert the list of Student objects to JSON compatible list of dictionaries
        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)

        # Attempt to save the file, otherwise provide structured Error Handling
        try:
            file = open(file_name, "w")
            json.dump(list_of_dictionary_data, file)
            file.close()
            print(f"Your data has been saved in {file_name}!\n")
            print("*" * 50, "\n")
        except TypeError as e:
            IO.output_error_messages("Please check the data is a valid JSON format", e)
        except Exception as e:
            IO.output_error_messages("Error: There was a problem with writing to the file.", e)
        finally:
            if not file.closed:
                file.close()
```

Figure 1.12: Converting the Student Object List back into a dictionary for compatibility with JSON file format.

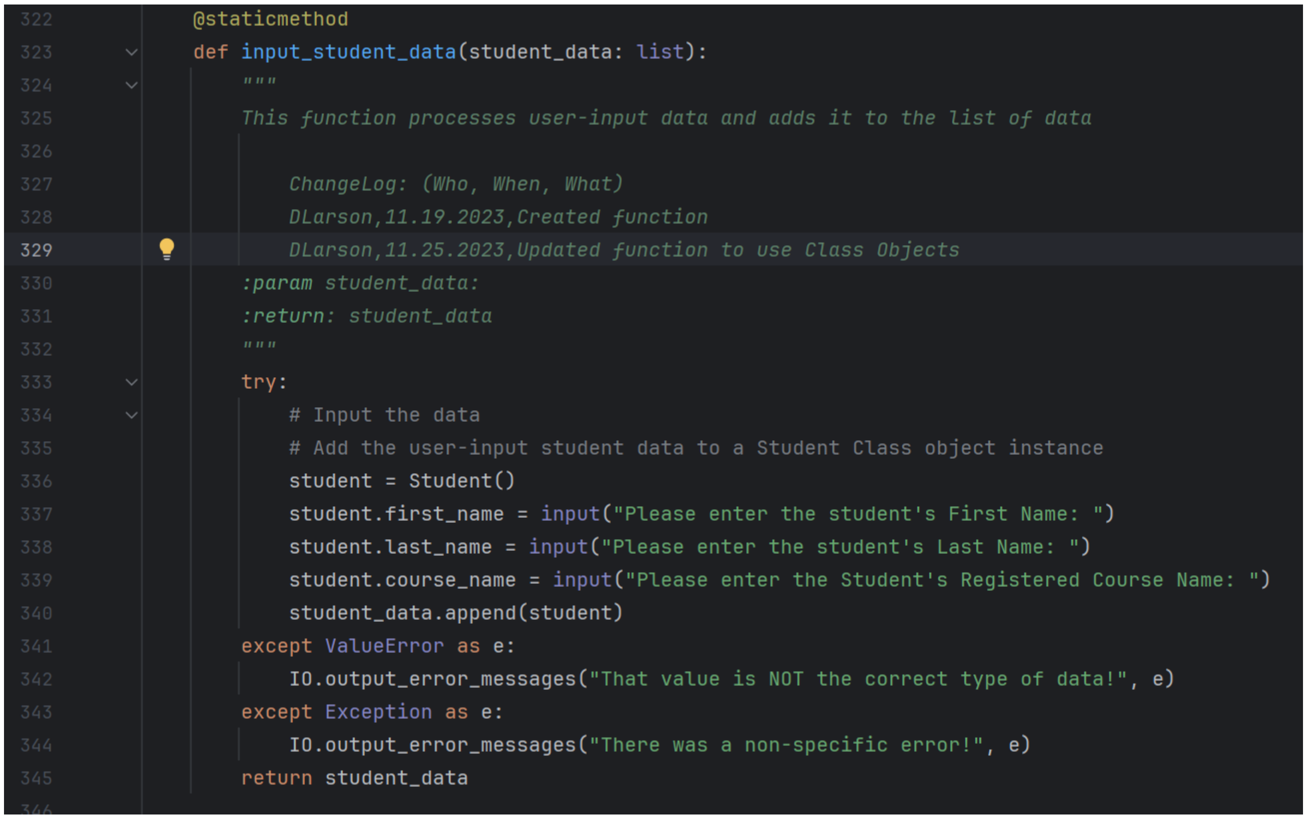
- display the student course data using the list of Student class instance objects instead of the dictionary list
    - update method output_student_courses to read from the list of objects instead of dictionary
        - replace using dictionary Keys to object attributes



```
    @staticmethod
    def output_student_courses(student_data: list):
        """
        This function displays all entered data to the user

            ChangeLog: (Who, When, What)
            DLarson,11.19.2023,Created function
            DLarson,11.25.2023,Updated function to use Class Objects
        :param student_data:
        :return: None
        """
        print()
        print("-" * 50)
        for student in student_data:
            print(f'Student {student.first_name} '
                  f'{student.last_name} is enrolled in {student.course_name}')
        print("-" * 50)
```

Figure 1.13: Updated code for the output message to use student Object attributes instead of Dictionary Keys.

- convert from using the list of dictionary data to use the list of Student object instances when inputting user data
    - update method IO.input_student_data
        - disable code assigning student variable as a dictionary with the dict keys
        - replace with code to convert the student objects into dictionaries matching properties to key values
            - "student" entry is assigned to an instance of the Student Class
            - Input values are assigned to class properties
                - Because the validation occurs within the Class level, it can be removed here



```
@staticmethod
def input_student_data(student_data: list):
    """
    This function processes user-input data and adds it to the list of data

        ChangeLog: (Who, When, What)
        DLarson,11.19.2023,Created function
        DLarson,11.25.2023,Updated function to use Class Objects
    :param student_data:
    :return: student_data
    """
    try:
        # Input the data
        # Add the user-input student data to a Student Class object instance
        student = Student()
        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 Student's Registered Course Name: ")
        student_data.append(student)
    except ValueError as e:
        IO.output_error_messages("That value is NOT the correct type of data!", e)
    except Exception as e:
        IO.output_error_messages("There was a non-specific error!", e)
    return student_data
```

Figure 1.14: Final updated code to create new Student Object Instances and assign the attributes with user input prompts.

Testing the Program

Now that the code runs properly when testing within PyCharm, it needs to be verified as functional outside of the IDE. To achieve this, the script is run in command shell by navigating to the directory where the file is stored and using the "python" command followed by the file name "Assignment07.py".

- tested and confirmed the following:
    - error handling when the file is read into the list of student Object instances
        - deleted the file from the directory, to mimic the file not existing
    - error handling for First and Last name
        - detects the presence of number or symbol characters
        - reports error to inform user to enter only letters
    - error handling for when student Object list is converted to dictionary rows and written to the file
        - to test: changed FILE_NAME constant to use .csv instead of .json
        - invalid file format error triggered
    - user can input student information: first name, last name, course name

        - input is saved to a Student Class instance object and assigned the correct property values
        - student object instances are appended to the students List of objects
    - user can input multiple student registrations
    - user can display and save multiple student registrations
    - program runs correctly in IDE and console

Summary

This module focused on a shift to Object oriented programming, with data (like Dictionary lists) being converted to allow for processing as Objects. This came with the introduction to Inheritance through the example of Person -> Student class instances, which are then used as Object instances to process and save Student Registration data in the updated program.

- Updating code to use Objects / properties instead of dictionaries / keys etc.
- Using protected private variables and “Accessor/Mutator” or “getter/setter” functions to validate / assign the instanced parameters
- Inheriting core properties through Parent Classes allows for changes to automatically propagate through every Child class, increasing efficiency / reuse
- Code efficiency and increased stability with validation checks happening in the Class level