Jorden Thomas

March 12, 2025

IT FDN 110

Assignment 07

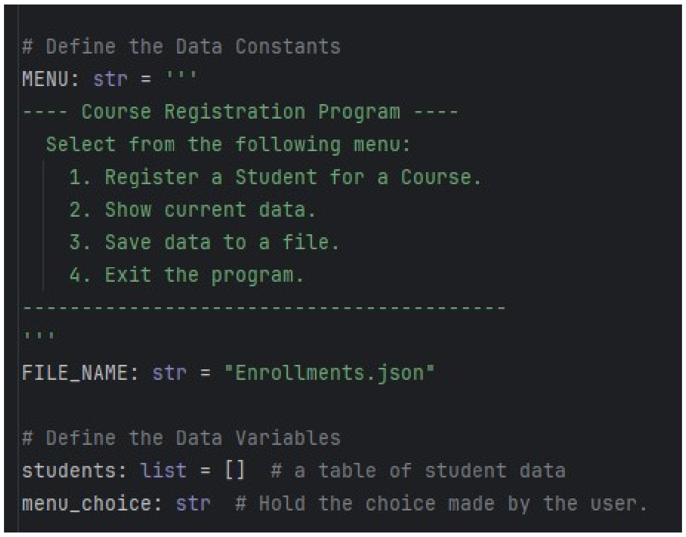
https://github.com/Jorden09T/IntroToProg-Python-Mod07

Assignment 07

Introduction

For this assignment, the focus was on expanding Assignment06 by adding a set of data classes. The classes added this time around are Person and student are added in.

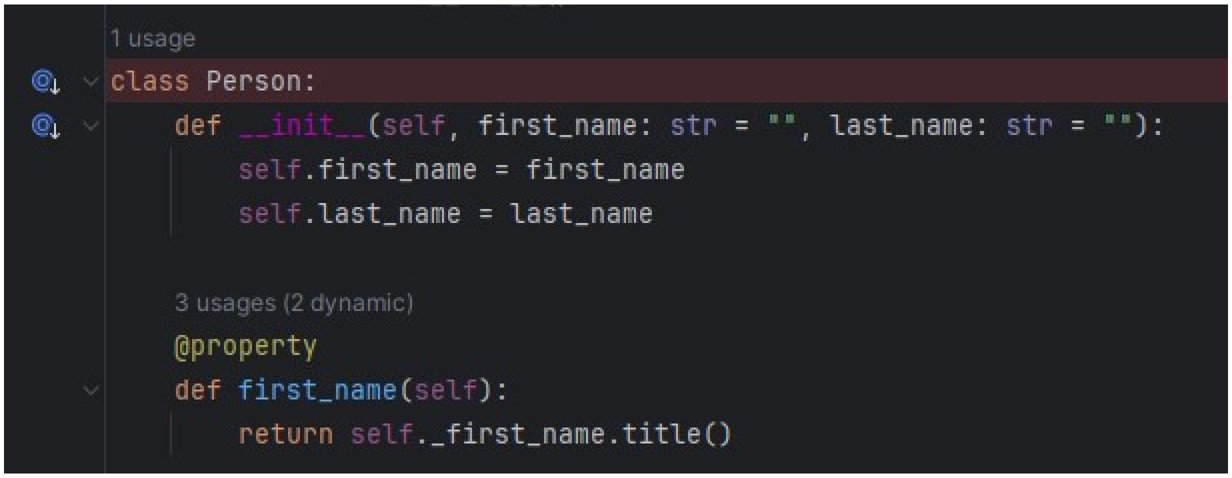
Constants & Variables



```
# 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.
```

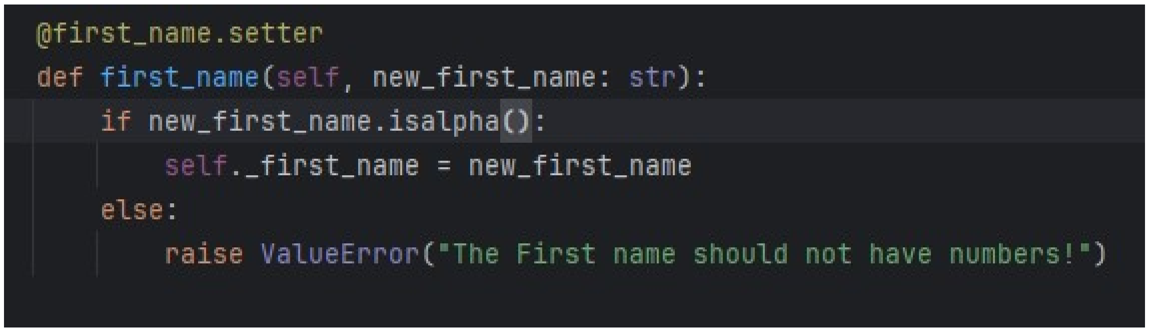
Nothing changed for the constants, they remained as they were. As for, the variable section, it was downsized to just the students: list and menu_choice: str.

Class Person



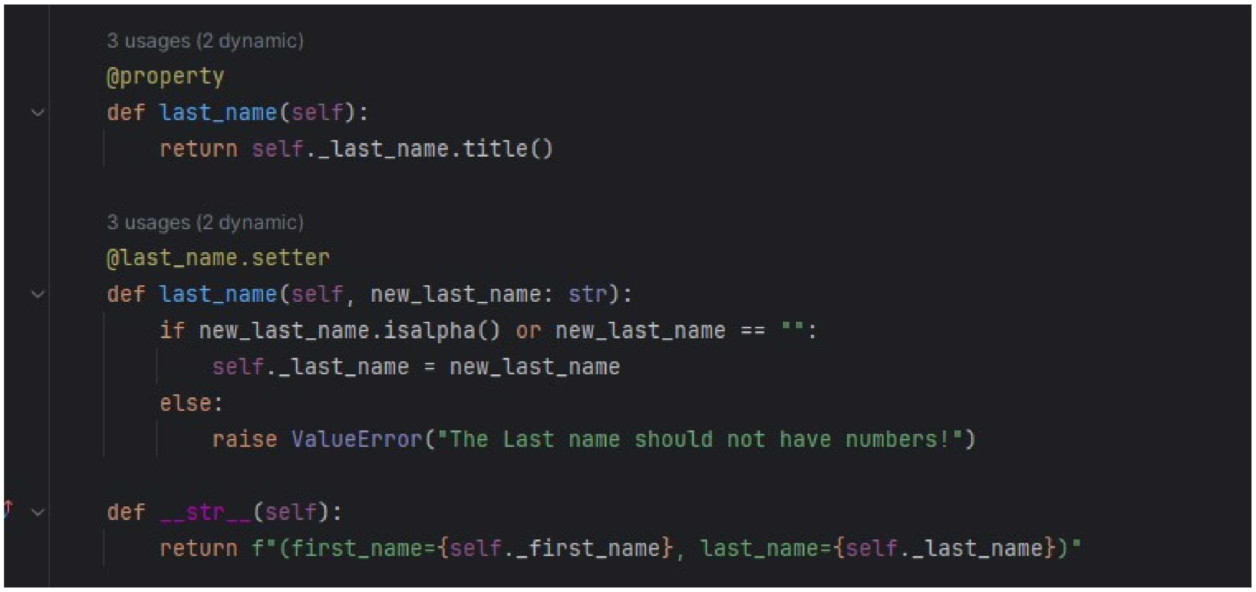
```
1 usage
class Person:
    def __init__(self, first_name: str = "", last_name: str = ""):
        self.first_name = first_name
        self.last_name = last_name

    3 usages (2 dynamic)
    @property
    def first_name(self):
        return self._first_name.title()
```

For the class person, the focus was to create the person class and develop constructors and properties for the first name and the last name. In the photo above we have the __initi__ being defined with the parameters of our variables that will be used in this class. Self is being used to reference the current instance of the person class. Below it we have the property for the first name. Within the property, we have a return of the first name attached to the title, which we are suing to capitalize the first name of the student.



```
@first_name.setter
def first_name(self, new_first_name: str):
    if new_first_name.isalpha():
        self._first_name = new_first_name
    else:
        raise ValueError("The First name should not have numbers!")
```

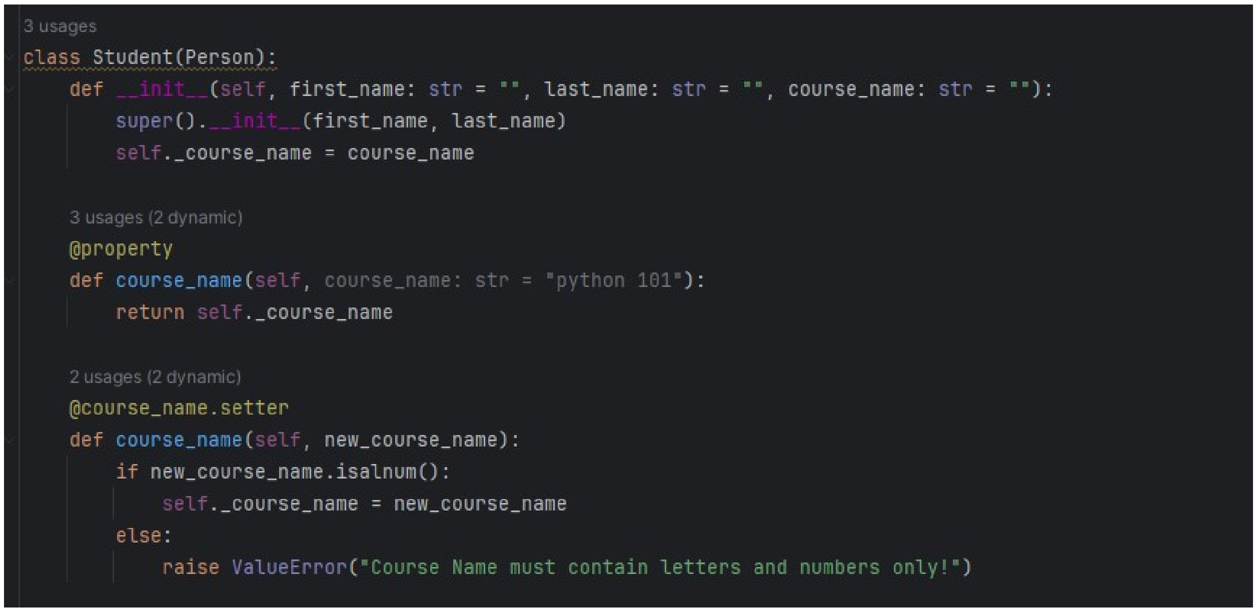
In the setter portion we are checking to see if the entered name is all from the alphabet if so it is registered as the first name, if it contains any numbers it is rejected and a custom exception is raised to alert the user.



```
3 usages (2 dynamic)
@property
def last_name(self):
    return self._last_name.title()

3 usages (2 dynamic)
@last_name.setter
def last_name(self, new_last_name: str):
    if new_last_name.isalpha() or new_last_name == "":
        self._last_name = new_last_name
    else:
        raise ValueError("The Last name should not have numbers!")

def __str__(self):
    return f"(first_name={self._first_name}, last_name={self._last_name})"
```

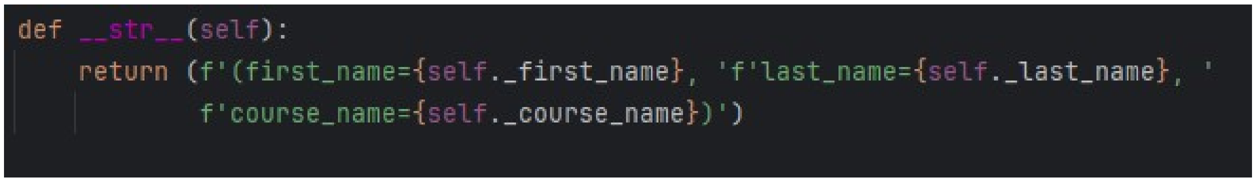
Same thing with the last name section for its property and setter. Lastly we have a have a string that is being defined. It will return the users first and last name.

Class Student



```
3 usages
class Student(Person):
    def __init__(self, first_name: str = "", last_name: str = "", course_name: str = ""):
        super().__init__(first_name, last_name)
        self._course_name = course_name

    3 usages (2 dynamic)
    @property
    def course_name(self, course_name: str = "python 101"):
        return self._course_name

    2 usages (2 dynamic)
    @course_name.setter
    def course_name(self, new_course_name):
        if new_course_name.isalnum():
            self._course_name = new_course_name
        else:
            raise ValueError("Course Name must contain letters and numbers only!")
```

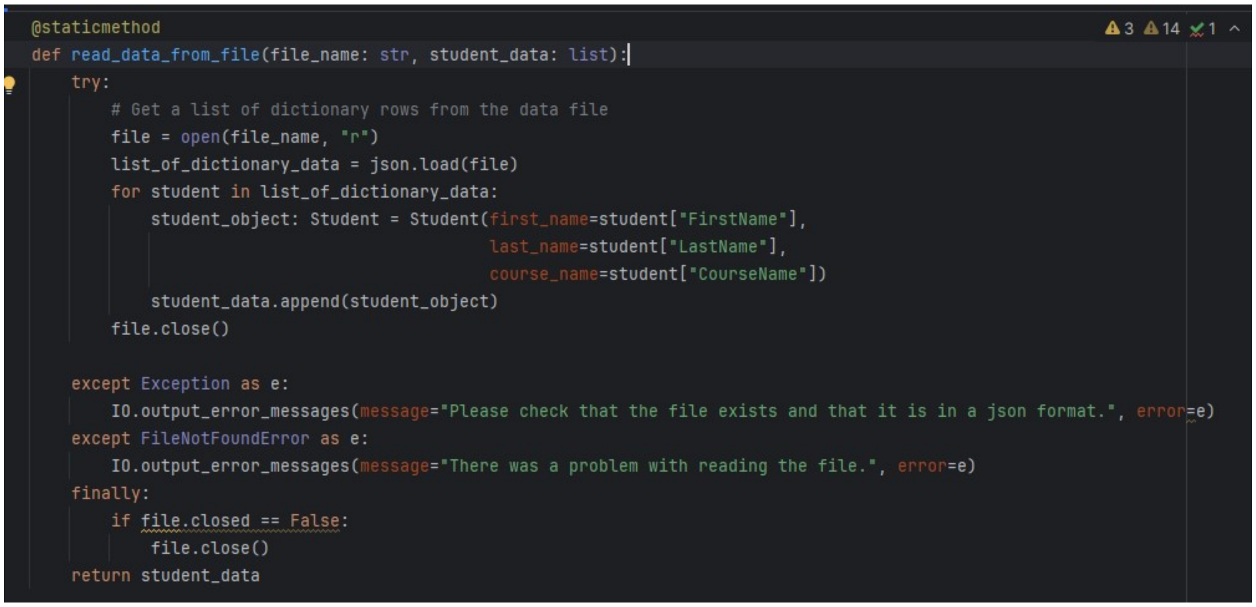
For the student class, we are extending it to the person class as we are pulling from it, in terms of the first and last name. these are defined in the parameters, along with the course name. Below is the supper() which is used to call methods from the parent class in this case is the Person class. Like above, the property and setter are created but for the course name. For the setter, I placed in a conditional for it to check for the course name to contain both numbers and characters. If it anything else is entered then a ValueError is raised and advises the user to enter only letters and numbers.



```python
def __str__(self):
    return (f'(first_name={self._first_name}, 'f'last_name={self._last_name}, '
            f'course_name={self._course_name})')
```

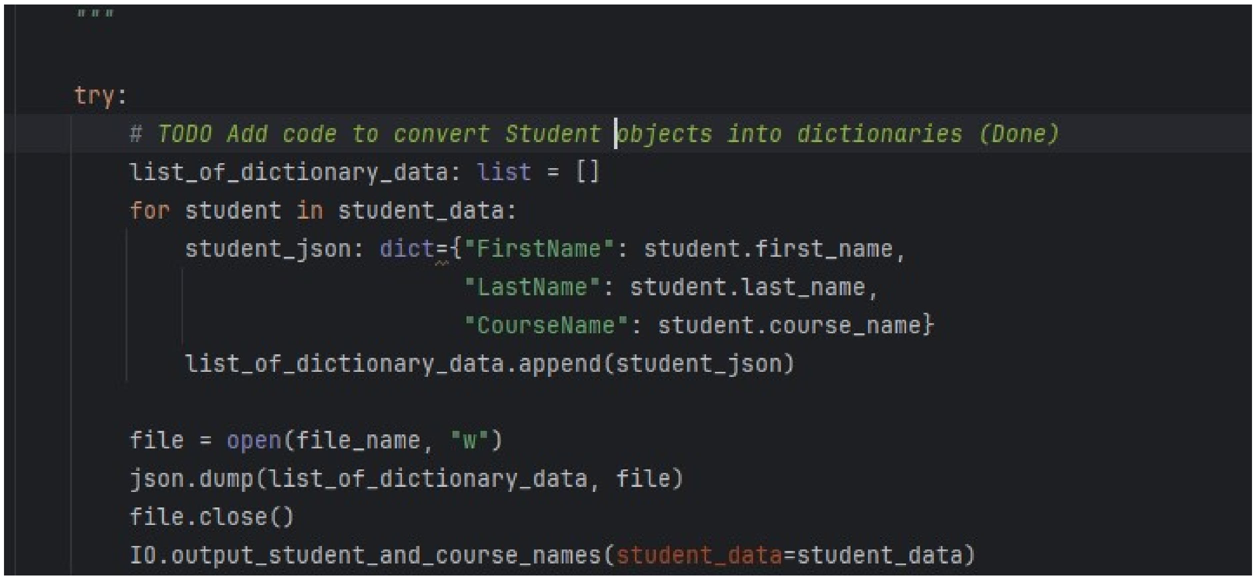
Once again we have the string where the first and last name are printed along with the course name.

Class FileProcessor



```python
@staticmethod
def read_data_from_file(file_name: str, student_data: list):
    try:
        # Get a list of dictionary rows from the data file
        file = open(file_name, "r")
        list_of_dictionary_data = json.load(file)
        for student in list_of_dictionary_data:
            student_object: Student = Student(first_name=student["FirstName"],
                                              last_name=student["LastName"],
                                              course_name=student["CourseName"])
            student_data.append(student_object)
        file.close()

    except Exception as e:
        IO.output_error_messages(message="Please check that the file exists and that it is in a json format.", error=e)
    except FileNotFoundError as e:
        IO.output_error_messages(message="There was a problem with reading the file.", error=e)
    finally:
        if file.closed == False:
            file.close()
    return student_data
```

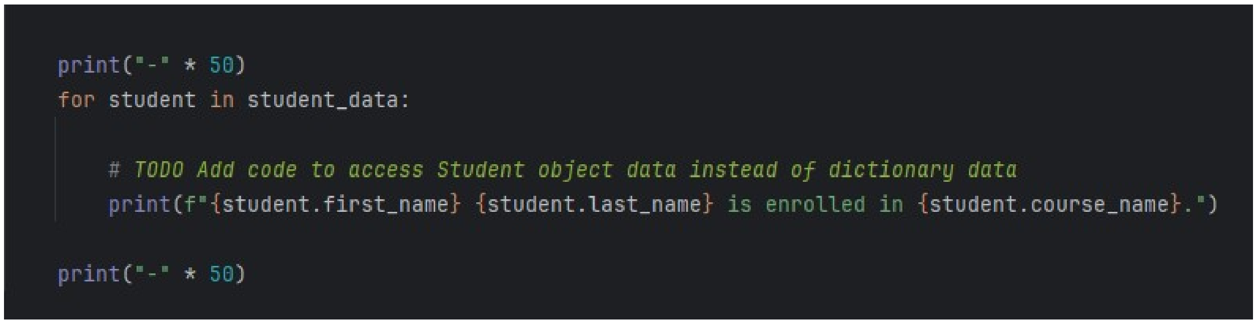
Within FileProcessor there were some changes made to the previous code. The first change was read_data_from_file(file_name: str, student_data: list), within this function, we changed what the json.load(file) was set to, so it went from student_data to list_of_dictionary_data. A statement was also developed for the list_of_dictionary_data where the student_object would store the student information within the json file. Along with the student_data.append(student_object), so that new entries would be added into the student_object.



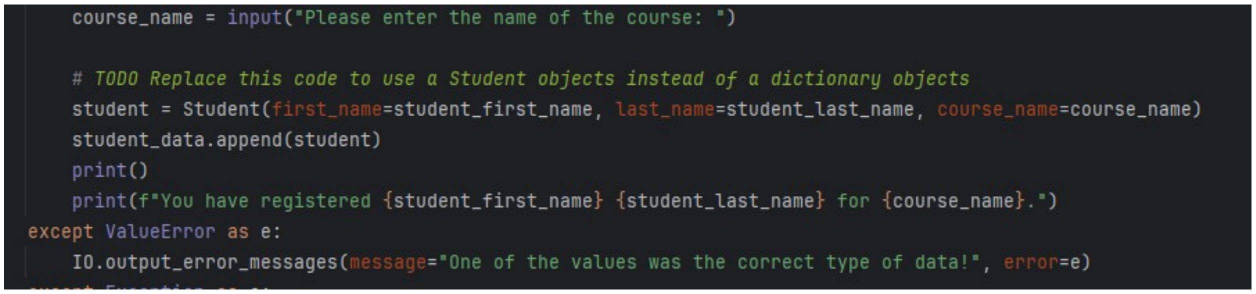
```python
    """

    try:
        # TODO Add code to convert Student objects into dictionaries (Done)
        list_of_dictionary_data: list = []
        for student in student_data:
            student_json: dict={"FirstName": student.first_name,
                                "LastName": student.last_name,
                                "CourseName": student.course_name}
            list_of_dictionary_data.append(student_json)

        file = open(file_name, "w")
        json.dump(list_of_dictionary_data, file)
        file.close()
        IO.output_student_and_course_names(student_data=student_data)
```

Within write_data_to_file(file_name: str, student_data: list) function, I converted the student objects into dictionaries as listed in the green. To do this we set list_of_dictionary_data to be an empty list. From there a For statement was used to structure how the data would be stored within the json file.

Class IO



```python
print("-" * 50)
for student in student_data:

    # TODO Add code to access Student object data instead of dictionary data
    print(f"{student.first_name} {student.last_name} is enrolled in {student.course_name}.")

print("-" * 50)
```



```python
    course_name = input("Please enter the name of the course: ")

    # TODO Replace this code to use a Student objects instead of a dictionary objects
    student = Student(first_name=student_first_name, last_name=student_last_name, course_name=course_name)
    student_data.append(student)
    print()
    print(f"You have registered {student_first_name} {student_last_name} for {course_name}.")
except ValueError as e:
    IO.output_error_messages(message="One of the values was the correct type of data!", error=e)
```

Like the FileProcessor class the changes done within the IO class. The changes made were two replacements of the dictionary objects and placing them to use a student object instead.