Nick Greco
Nov 27, 2024
Foundations of Programming: Python
Assignment07
https://github.com/NBGreco/IntroToProg-Python-Mod07

Classes and Inheritance within Python

Introduction

This script implements a course registration system, allowing users to register students, view current student-course data, and save the information to a JSON file. It defines several classes, including “Person” and “Student”, to manage individual and student-specific data, with data handling done via JSON files using the “FileProcessor” class. In this script, Python inheritance is used by the “Student” class, which inherits from the “Person” class, allowing it to reuse the functionality and attributes of “Person” while adding its own property (“course_name”) and customizing the string representation method. The script runs a menu-driven interface that prompts the user to either register a new student, view current data, save the data, or exit the program, which is very similar to previous assignments.

Creating the Script

Docstrings have been removed from all script excerpts in this document for conciseness. These are included in the full ‘Assignment07’ script.

“Person” Class

The “Person” class is a blueprint for creating person objects that store a first name and a last name. The “Person” class is shown below in Figure 1. It includes:

- **Attributes**: It has two properties, “first_name” and “last_name”, which are validated to ensure that they only contain alphabetic characters (or can be empty). These are set using setter methods that enforce this rule.
- **Methods**:
    - The “__init__” method initializes the person’s first and last names.
    - The “first_name” and “last_name” properties provide access to the name attributes, applying a title case format when retrieving them.

- The “__str__” method returns a string representation of the person in the format "FirstName, LastName".

```
class Person:

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

    @property
    def first_name(self) -> str:
        return self.__first_name.title()

    @first_name.setter
    def first_name(self, value: str) -> None:
        if value.isalpha() or value == "":
            self.__first_name = value
        else:
            raise ValueError("The first name should only contain letters!")

    @property
    def last_name(self) -> str:
        return self.__last_name.title()

    @last_name.setter
    def last_name(self, value: str) -> None:
        if value.isalpha() or value == "":
            self.__last_name = value
        else:
            raise ValueError("The last name should only contain letters!")

    def __str__(self) -> str:
        return f"{self.first_name}, {self.last_name}"
```

Figure 1: “Person” Class

“Student” Class

The “Student” class is a subclass of the “Person” class that represents a student, who is a person with additional properties related to their education. It extends the functionality of the “Person” class by adding a “course_name” property, which represents the name of the course the student is enrolled in. The “Student” class is shown below in Figure 2. The class includes:

- **Constructor (“__init__”)**: Initializes a Student object with a first name, last name, and course name. It uses “super()” to call the constructor of the Person class to initialize the “first_name” and “last_name” attributes.
- **Property “course_name”**: A property with a getter and setter for managing the course name. The getter ensures the course name is capitalized (“title()”), while the setter simply assigns the value.

- **“__str__” method**: Overrides the string representation method to return a string that includes the student's full name (inherited from the “Person” class) and their enrolled course name.

```
class Student(Person):

    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

    @property
    def course_name(self) -> str:
        return self.__course_name.title()

    @course_name.setter
    def course_name(self, value: str) -> None:
        self.__course_name = value

    def __str__(self) -> str:
        return f'{super().__str__()}, {self.course_name}'
```

Figure 2: “Student” Class

“FileProcessor” Class

The “FileProcessor” class is designed to handle reading from and writing to JSON files, specifically for storing and retrieving student data. The “FileProcessor” class is shown below in Figure 3. It provides two key methods:

- **“read_data_from_file:**
  - This method reads student data from a JSON file and loads it into a list of Student objects.
  - It opens the specified file, and then creates Student instances using the values from the JSON file (such as “FirstName”, “LastName”, and “CourseName”).
  - The data is added to the “student_data” list, which is then returned.
  - If an error occurs while reading the file (e.g., file not found, invalid JSON), it outputs an error message using the “IO.output_error_messages()” method.
- **“write_data_to_file”:**
  - This method writes a list of Student objects to a JSON file.
  - It converts each Student object in the “student_data” list to a dictionary containing the student's “first_name”, “last_name”, and “course_name”.
  - The dictionary data is then saved into a JSON file using “json.dump()”.
  - If an error occurs during the write operation, an error message is displayed.
  - After successfully writing the data, the “IO.output_student_courses()” method is called to output the current student data.

```python
class FileProcessor:

  @staticmethod
   def read_data_from_file(file_name: str, student_data: list) -> list:
       try:
           file = open(file_name, "r")
           list_of_dictionary_data = json.load(file)
           for student in list_of_dictionary_data:
               st_temp = Student(first_name = student["FirstName"], \
                                 last_name = student["LastName"], \
                                 course_name = student["CourseName"])
               student_data.append(st_temp)
           file.close()
       except Exception as e:
           IO.output_error_messages(message = "Error: There was a problem" \
                   "with reading the file.", error = e)
       finally:
           if not file.close:
               file.close()
       return student_data

   @staticmethod
   def write_data_to_file(file_name: str, student_data: list) -> None:
       try:
           list_of_dictionary_data: list = []
           for student in student_data:
               st_temp: dict = {"FirstName": student.first_name, \
                                "LastName": student.last_name, \
                                "CourseName": student.course_name}
               list_of_dictionary_data.append(st_temp)
           file = open(file_name, "w")
           json.dump(list_of_dictionary_data, file)
           file.close()
           IO.output_student_courses(student_data = student_data)
       except Exception as e:
           message = "Error: There was a problem with writing to the file.\n"
           message += "Please check that the file is not open by another" \
                   " program."
           IO.output_error_messages(message = message, error = e)
       finally:
           if not file.close:
               file.close()
```

Figure 3: "FileProcessor" Class

“IO” Class

The “IO” class is responsible for managing the user interface aspects of the program, focusing on input and output. It provides several static methods to handle various user interactions with the program, including displaying messages, getting user input, and managing errors. The “IO” class is shown below in Figure 4. Here's what each method does:

- **“output_error_messages”**:
  - Displays custom error messages to the user.
  - Takes a message (string) and an optional error (exception object) as arguments.
  - If an error is provided, it prints detailed technical information about the error, including its type and documentation.
- **“output_menu”**:
  - Displays the menu of choices to the user, given a menu string.
  - Typically used to show the available options for the user to choose from.
- **“input_menu_choice”**:
  - Prompts the user to select an option from the menu.
  - The user is asked for input, and the method ensures that the input is valid (i.e., one of the choices: "1", "2", "3", or "4").
  - If the user enters an invalid choice, an error message is displayed and the user is asked again.
- **“output_student_courses”**:
  - Displays a list of students and their enrolled courses.
  - Accepts a list of “student_data” (which is a list of Student objects) and prints each student's first name, last name, and course name.
  - The student data is formatted in a clean, readable way with lines separating the information.
- **“input_student_data”**:
  - Prompts the user to input data for a new student (first name, last name, and course name).
  - Creates a new Student object with the provided data and adds it to the “student_data” list.
  - If the input data is invalid (e.g., if the user enters non-alphabetical characters for a name), an error message is displayed and the program asks for the data again.

```
class IO:

    @staticmethod
    def output_error_messages(message: str, error: Exception = None)-> None:
        print(message, end="\n")
        if error is not None:
            print("\n----------- Technical Error Message -----------")
            print(error, error.__doc__, type(error), sep='\n')

    @staticmethod
    def output_menu(menu: str) -> None:
        print(menu)
```

```python
    @staticmethod
    def input_menu_choice() -> str:
        choice: str = "0"
        try:
            choice = input("What would you like to do? ")
            if choice not in ("1", "2", "3", "4"):
                raise Exception("\n!!! Please choose a menu option " \
                                "(1, 2, 3, or 4). !!!")
        except Exception as e:
            IO.output_error_messages(e.__str__())
        return choice

    @staticmethod
    def output_student_courses(student_data: list) -> None:
        print("-" * 60)
        for student in student_data:
            print(f"\t{student.first_name} {student.last_name} "\
                  f"is enrolled in {student.course_name}")
        print("-" * 60)

    @staticmethod
    def input_student_data(student_data: list) -> list:
        try:
            student = Student()
            student.first_name = input("Enter the student's first name: ")
            student.last_name = input("Enter the student's last name: ")
            student.course_name = input("Please enter the course name: ")
            student_data.append(student)
            print()
            print(f"You have registered {student.first_name} " \
                    f"{student.last_name} for {student.course_name}.")
        except ValueError as e:
            IO.output_error_messages(message = "Incorrect type of data!", \
                                     error = e)
        except Exception as e:
            IO.output_error_messages(message = "There was a problem with " \
                                               "your entered data.", \
                                     error = e)
        return student_data
```

Figure 4: "IO" Class

The main code body is largely the same as the previous assignment, calling the various methods for each menu choice. For this reason, it is not examined in this knowledge document.

Key Functionalities & Concepts Demonstrated

Object-Oriented Programming (OOP)

Classes and Inheritance:

- The script defines classes “Person”, “Student”, “FileProcessor”, and “IO”, demonstrating object-oriented programming principles like encapsulation and inheritance.
- Student inherits from Person, making use of inheritance to reuse code (e.g., the “first_name” and “last_name” properties).

Encapsulation:

- The first_name and last_name properties are encapsulated with private attributes (“__first_name”, “__last_name”) and are accessed via getter and setter methods, ensuring that the values are validated before being set (e.g., only alphabetic characters are allowed).

Properties and Data Validation

Getters and Setters (Properties):

- The “first_name”, “last_name”, and “course_name” attributes use properties to define getter and setter methods. This ensures that data is accessed and modified in a controlled way.
- The setters enforce validation, such as ensuring that “first_name” and “last_name” only contain alphabetic characters (with “.isalpha()”).

Running the Script

Using PyCharm IDE

The ‘Assignment07’ script was run using PyCharm IDE using a starter ‘Enrollments.json’ file. Figure 5 (left) demonstrates selecting menu option “1” and entering multiple new students. Example user input is shown in green text. Figure 5 (right) demonstrates the output when menu options “2” and “3” are chosen.

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

What would you like to do? 1
Enter the student's first name: Albert
Enter the student's last name: Einstein
Please enter the course name: Advanced Physics

You have registered Albert Einstein for Advanced Physics.

----- 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
--------------------------------------

What would you like to do? 1
Enter the student's first name: Marie
Enter the student's last name: Curie
Please enter the course name: Advanced Chemistry

You have registered Marie Curie for Advanced Chemistry.
```

```
What would you like to do? 2
-------------------------------------------------------
    Nick Greco is enrolled in Python 110
    Steve Howe is enrolled in Advanced Guitar
    James Cameron is enrolled in Filmmaking
    Albert Einstein is enrolled in Advanced Physics
    Marie Curie is enrolled in Advanced Chemistry
-------------------------------------------------------

----- 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
--------------------------------------

What would you like to do? 3
-------------------------------------------------------
    Nick Greco is enrolled in Python 110
    Steve Howe is enrolled in Advanced Guitar
    James Cameron is enrolled in Filmmaking
    Albert Einstein is enrolled in Advanced Physics
    Marie Curie is enrolled in Advanced Chemistry
-------------------------------------------------------
```

Figure 5: 'Assignment07' Menu Selections "1" thru "3".

Figure 6 shows the program output when menu option "4" is chosen and the program is exited.

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

What would you like to do? 4

-----------------------------------
*** Exiting Program. Thank you! ***
-----------------------------------
```

Figure 6: Menu Option "4" Output

Figure 7 shows the contents of the 'Enrollments.json' file after running the 'Assignment07' script and providing user input to register two additional students.

Assignment07.py Enrollments.json

```
[
  {"FirstName": "Nick", "LastName": "Greco", "CourseName": "Python 110"},
  {"FirstName": "Steve", "LastName": "Howe", "CourseName": "Advanced Guitar"},
  {"FirstName": "James", "LastName": "Cameron", "CourseName": "Filmmaking"},
  {"FirstName": "Albert", "LastName": "Einstein", "CourseName": "Advanced Physics"},
  {"FirstName": "Marie", "LastName": "Curie", "CourseName": "Advanced Chemistry"}
]
```

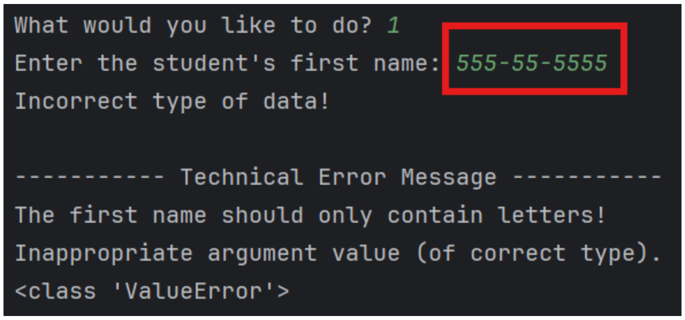
Figure 7: 'Enrollments.json' Data Contents after running Script

Figure 8 demonstrates error handling when incorrect menu option selections are provided. In the first scenario, a blank input was provided. In the second scenario, an alphabetic phrase with a special character was provided.

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

What would you like to do?

!!! Please choose a 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
--------------------------------------

What would you like to do? Thanksgiving Turkey!

!!! Please choose a 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 8: Menu Selection User Input Error Handling

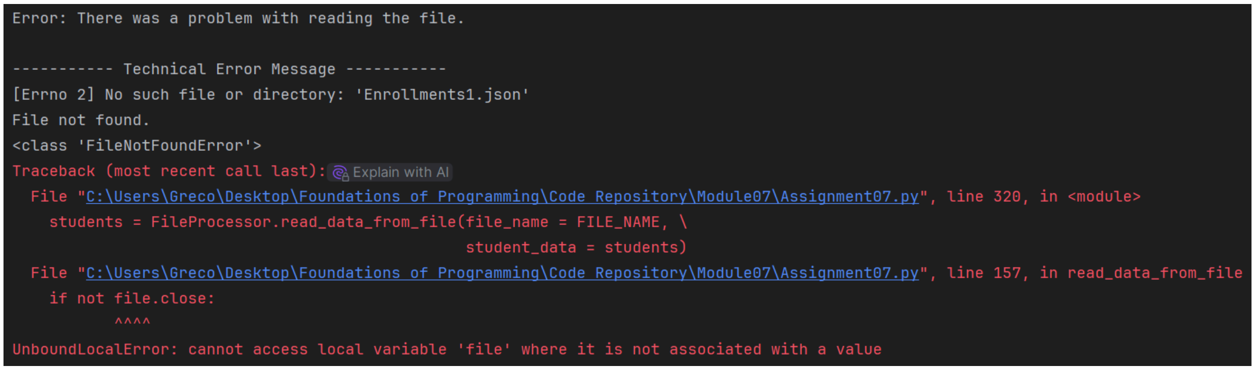
Figure 9 demonstrates error handling when incorrect user input is provided for a new student data entry. In this example, a social security number was provided instead of a first name.



```
What would you like to do? 1
Enter the student's first name: 555-55-5555
Incorrect type of data!

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

Figure 9: Menu Option “1” User Input Error Handling

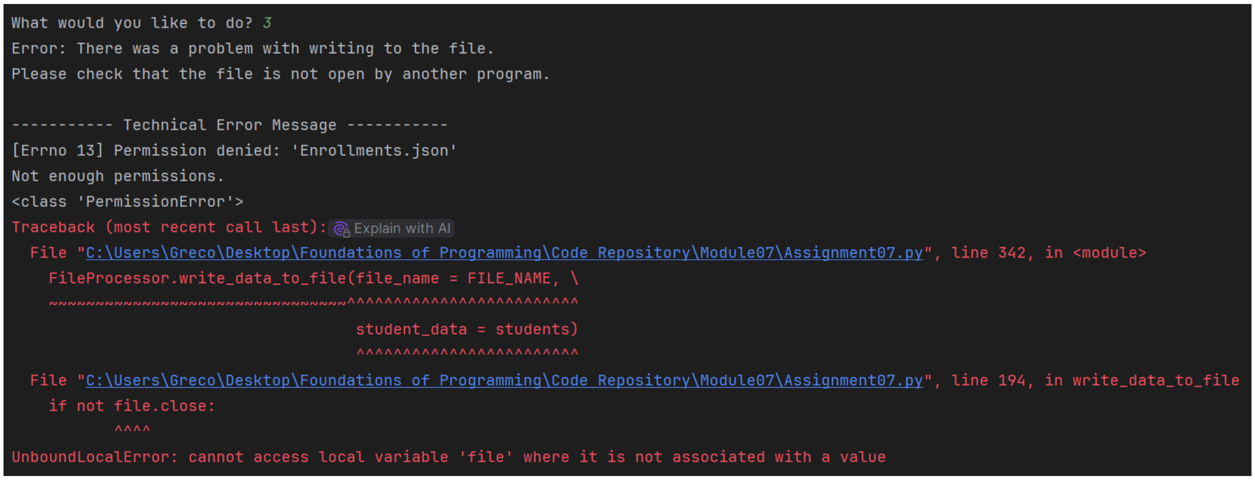
Figure 10 demonstrates error handling when the ‘Enrollments.json’ file is not found when initially read. To simulate this error, the file was temporarily renamed to ‘Enrollments1.json’.



```
Error: There was a problem with reading the file.

----------- Technical Error Message -----------
[Errno 2] No such file or directory: 'Enrollments1.json'
File not found.
<class 'FileNotFoundError'>
Traceback (most recent call last):
  File "C:\Users\Greco\Desktop\Foundations of Programming\Code Repository\Module07\Assignment07.py", line 320, in <module>
    students = FileProcessor.read_data_from_file(file_name = FILE_NAME, \
                                                 student_data = students)
  File "C:\Users\Greco\Desktop\Foundations of Programming\Code Repository\Module07\Assignment07.py", line 157, in read_data_from_file
    if not file.close:
           ^^^^
UnboundLocalError: cannot access local variable 'file' where it is not associated with a value
```

Figure 10: ‘Enrollments.json’ Read Error Handling

Figure 11 demonstrates error handling when the ‘Enrollments.json’ file is unable to be written to. To simulate this error, the file was set to read-only mode.



```
What would you like to do? 3
Error: There was a problem with writing to the file.
Please check that the file is not open by another program.

----------- Technical Error Message -----------
[Errno 13] Permission denied: 'Enrollments.json'
Not enough permissions.
<class 'PermissionError'>
Traceback (most recent call last):
  File "C:\Users\Greco\Desktop\Foundations of Programming\Code Repository\Module07\Assignment07.py", line 342, in <module>
    FileProcessor.write_data_to_file(file_name = FILE_NAME, \
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^
                                     student_data = students)
                                     ^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\Greco\Desktop\Foundations of Programming\Code Repository\Module07\Assignment07.py", line 194, in write_data_to_file
    if not file.close:
           ^^^^
UnboundLocalError: cannot access local variable 'file' where it is not associated with a value
```

Figure 11: ‘Enrollments.json’ Write Error Handling

Using Windows Command Prompt

Figure 12 shows running the same script using Windows Command Prompt.

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

What would you like to do? 1
Enter the student's first name: Sammy
Enter the student's last name: Sosa
Please enter the course name: How to Hit a Homerun

You have registered Sammy Sosa for How To Hit A Homerun.

----- 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
--------------------------------------

What would you like to do? 2
------------------------------------------------------------
        Nick Greco is enrolled in Python 110
        Steve Howe is enrolled in Advanced Guitar
        James Cameron is enrolled in Filmmaking
        Albert Einstein is enrolled in Advanced Physics
        Marie Curie is enrolled in Advanced Chemistry
        Sammy Sosa is enrolled in How To Hit A Homerun
------------------------------------------------------------
```

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

What would you like to do? 3
------------------------------------------------------------
        Nick Greco is enrolled in Python 110
        Steve Howe is enrolled in Advanced Guitar
        James Cameron is enrolled in Filmmaking
        Albert Einstein is enrolled in Advanced Physics
        Marie Curie is enrolled in Advanced Chemistry
        Sammy Sosa is enrolled in How To Hit A Homerun
------------------------------------------------------------

----- 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
--------------------------------------

What would you like to do? 4

-----------------------------------
*** Exiting Program. Thank you! ***
-----------------------------------
```

Figure 11: Using Windows Command Prompt to run 'Assignment06.py'

Figure 13 shows the contents of the JSON file after executing the script using Command Prompt.

Enrollments.json

File Edit View


```
[
 {"FirstName": "Nick", "LastName": "Greco", "CourseName": "Python 110"},
 {"FirstName": "Steve", "LastName": "Howe", "CourseName": "Advanced Guitar"},
 {"FirstName": "James", "LastName": "Cameron", "CourseName": "Filmmaking"},
 {"FirstName": "Albert", "LastName": "Einstein", "CourseName": "Advanced Physics"},
 {"FirstName": "Marie", "LastName": "Curie", "CourseName": "Advanced Chemistry"},
 {"FirstName": "Sammy", "LastName": "Sosa", "CourseName": "How To Hit A Homerun"}
]
```


Figure 13: JSON File Contents after running Script in Windows Command Prompt

Summary

This document describes the implementation of a course registration system using Python, which allows users to register students, view current data, and save it to a JSON file. It outlines the use of object-oriented programming (OOP), particularly inheritance and encapsulation, by defining classes such as "Person" and "Student", with the latter inheriting from the former. Additionally, the "FileProcessor" class handles reading and writing to the JSON file, and the "IO" class manages user input and output, with robust error handling throughout the program to ensure smooth user interaction and data processing.

Citations

OpenAI ChatGPT. (November 2024). https://chatgpt.com/: Aspects of this assignment were informed and created by queries I submitted to ChatGPT.