Ophir Amon

2/26/2024

Foundations of Programming: Python

Assignment07

GitHub link: https://github.com/OphirAmon13/IntroToProg-Python-Mod07

Assignment 07 Steps

Introduction

Our seventh assignment was very similar to the previous assignment, as it dealt with loops, conditional statements, lists, dictionaries, and error handling, functions, classes, and JSON files. However, this assignment added two new topics: objects and inheritance. The only difference in this assignment was that we were required to use two new classes: the Person and the Student class. These are class objects. The Student class inherits from the Person class. Before I presented any information to the user, I first imported the method "exit()" from the module "sys" and the module "json" so that I could use the exit() and JSONDecodeError functions later on, then I established my constant variables and the variables I will use later on in the code. I then defined all the classes and functions I was planning on using in the code, so that I can call them in my "main".

Loops

In general, loops are helpful for when you want to perform a task repeatedly. I used two different types of loops for this program: a 'while' loop and a 'for' loop. 'While' loops are helpful for when you want to iterate your program based on a condition. In this case, my condition was while the program was TRUE (Figure 1). 'For' loops are good for when we know exactly how many times we want to iterate and are therefore good when looping through lists as I did in this program (Figure 2).

```
while True:
```

Figure 1: Python 'While' Loop Example.

```
for student in list_of_students:
    print(f"You have registered {student["first_name"]} {student["last_name"]} for {student["course_name"]}.")
```

Figure 2: Python 'For' Loop Example

Conditional Statements

Much like a 'while' loop, conditional statements only run if a criterion is met. For this program, I used the match method. This method allows me to consider however many cases I want by simply labeling what the case should be. This program had 5 base cases, based on what menu option the user inputs (Figure 3). The first option allowed the user to register a student for a specific course. The user just needs the first name, last name, and the course name to complete the first task. The second task shows the current data that the user has input so far. This can include registration information for multiple students. The third option takes in the input data and creates a JSON file with the data, as well as informing the user that they have registered students for their courses. The fourth option ends the program by using the exit() function. The last option (case _) is for when

the user selects an option that is not on the menu. When this option runs, it simply prints the message “Please select a valid option”.

```
# Checks the user's menu choice against different cases
match menu_choice:
    case "1":
        IO.input_student_data(students)
    case "2":
        IO.output_student_courses(students)
    case "3":
        FileProcessor.write_data_to_file(FILE_NAME, students)
    case "4":
        IO.quit_program()
    case _:
        print("ERROR: Please select a valid option")
```

Figure 3: Python Conditional Statements Examples

JSON Files

As mentioned above, the third option of choices was to create a JSON text file with the information retrieved from the user. To do this, I used the ‘open’ function and set it to the ‘w’ (writing) mode. The open() function opens a new file with any desired name and type. In this case, I opened a JSON file named ‘Enrollments.json’ and began editing the contents of the file (Figure 4).

```
@staticmethod
def write_data_to_file(file_name: str, student_data: list): # Saves all information to JSON file
    try:
        with open(file_name, "w") as file:
            json.dump(student_data, file)
            print(student_data)
            for row in student_data:
                print(f"You have registered {row["first_name"]} {row["last_name"]} for {row["course_name"]}.")
    except Exception as error_message:
        FileProcessor.output_error_message(f"There was an error saving the data to the {file_name} file!", error_message)
```

Figure 4: Python Open() Function Example.

Lists

Lists are useful for storing multiple pieces of data into one collection. For this program I used lists so that the user can input registration information for multiple students at the same time. The list I used in this assignment stored Students in its contents (Figure 5).

```
student_first_name = input("Enter the student's first name: ")
student_last_name = input("Enter the student's last name: ")
course_name = input("Please enter the name of the course: ")
new_student = Student(student_first_name, student_last_name, course_name)
student_data.append(new_student) # Adds student to list of all student data
```

Figure 5: Python Lists Example.

Dictionaries

Dictionaries are useful for storing data in key-value pairs. For each key in a dictionary there is a corresponding value. For this assignment, the keys were the strings “first_name”, “last_name”, and “course_name”, while the values were the student.first_name, student.last_name, and student.course_name that the user inputted (Figure 6).

```
student_dict: dict = {
     "first_name": student.first_name,
     "last_name": student.last_name,
     "course_name": student.course_name,
}
```

Figure 6: Python Dictionaries Example.

Error Handling

Error handling is a very important aspect for the programmer to account for. Often, while writing code, the program can encounter many errors. In doing so, the program crashes, which is not very desirable. So, to ensure that does not happen, while developing the code, one should anticipate any potential errors that might come up and "handle" them. For example, the code for this assignment had a requirement that whenever the code was run, it read data from a file called Enrollments.json. However, if the computer running the program did not have such a file stored, the program would crash and a FileNotFoundError would appear. To avoid having the program crash, I used a try-except block. The computer will try to execute whatever is under the try block and if it encounters an error accounted for in the except block it will print an error message instead of crashing (Figure 7).

```
# Function to output error message whereever needed
@staticmethod
def output_error_message(message: str, error: Exception = None):
    print(message)
    print(f"Error detail: {error}")

# Function that reads and prints the data from a file
@staticmethod
def read_data_from_file(file_name: str, student_data: list):
    try:
        with open(file_name, "r") as file:
            print("First Name \tLast Name \tCourse Name")
            loaded_student_table = json.load(file)
            for item in loaded_student_table:
                    new_student = Student(item["first_name"], item["last_name"], item["course_name"])
                    student_data.append(new_student)
                    print(f"{new_student.first_name} \t\t{new_student.last_name} \t\t{new_student.course_name}")
    except FileNotFoundError as error_message:
        IO.output_error_messages(f"There was an error finding the {file_name} file!", error_message)
    except JSONDecodeError as error_message:
        IO.output_error_messages(f"There was an error reading the data from the {file_name} file!", error_message)
    except Exception as error_message:
        IO.output_error_messages(f"There was an error reading the data from the {file_name} file!", error_message)
```

Figure 7: Python Error Handling Example.

Functions

Functions are very useful as they can make code much neater. A function is a block of code that only runs when it is called. Functions are especially useful for when you want to run the same block of code multiple times, as instead of having the same block of code for each instance, you can just call the function each time. Functions can take in parameters as well. This means that it can use variables that are not explicitly written in the function itself. I defined all my functions before my "main" block and called any function I needed under the main block, making my code much nicer visually (Figure 8).

```python
if __name__ == "__main__":
    FileProcessor.read_data_from_file(FILE_NAME, students)
    while True:
        # Present the menu of choices
        IO.output_menu(MENU)
        menu_choice = IO.input_menu_choice()
        # Checks the user's menu choice against different cases
        match menu_choice:
            case "1":
                IO.input_student_data(students)
            case "2":
                IO.output_student_courses(students)
            case "3":
                FileProcessor.write_data_to_file(FILE_NAME, students)
            case "4":
                IO.quit_program()
            case _:
                print("ERROR: Please select a valid option")
```

Figure 8: Python Functions Example.

Classes

Classes are also helpful for making code neater. For this assignment I used two classes: one called FileProcessor and one called IO. In each class, I included the functions that had to do with those categories in their respective class (Figure 9).

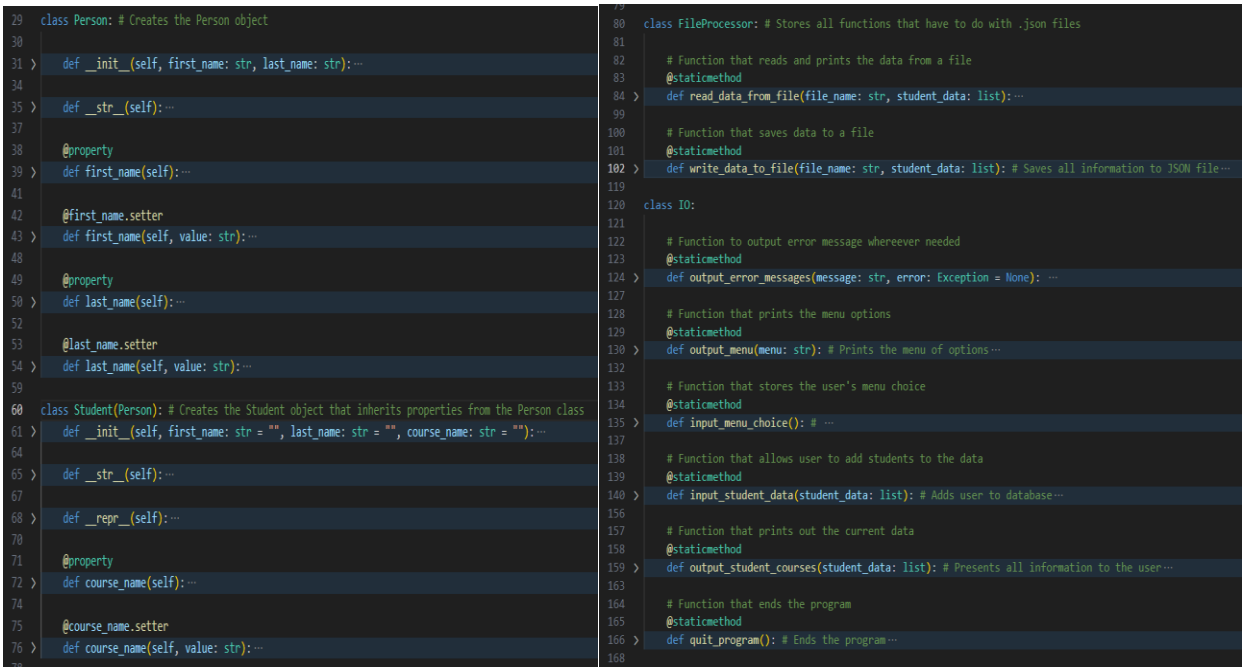

Figure 9: Python Classes Example.

Objects and Inheritance

In programming, objects are classifications of specific data. These objects can have specific parameters that each instance must have. For example, in this assignment I use two class objects:

Person and Student. Any instance of a person must contain the parameters first_name and last_name. Since any student is also a person, it inherits these required parameters as well as has one of its own: course_name (Figure 10). I utilized these classes when the user chooses option 1, which allows them to input a student into the data. Whatever information they input is then stored and used to create an instance of a Student and by extension an instance of a Person. In previous assignments, this information was just stored into variables.

```
class Person: # Creates the Person object

    def __init__(self, first_name: str, last_name: str):···

    def __str__(self):···

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

    @first_name.setter
    def first_name(self, value: str):
        if value.isalpha():
            self.__first_name = value
        else:
            raise ValueError("The first name should not contain numbers.")

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

    @last_name.setter
    def last_name(self, value: str):
        if value.isalpha():
            self.__last_name = value
        else:
            raise ValueError("The last name should not contain numbers.")

class Student(Person): # Creates the Student object that inherits properties from the Person class
    def __init__(self, first_name: str = "", last_name: str = "", course_name: str = ""):
        super().__init__(first_name, last_name)
        self.course_name = course_name

    def __str__(self):···

    def __repr__(self):···

    @property
    def course_name(self):
        return self.__course_name

    @course_name.setter
    def course_name(self, value: str):
        self.__course_name = value
```

Figure 10: Python Objects/Inheritance Example.

Summary

In conclusion, this assignment utilized six topics: loops, conditional statements, lists, dictionaries, error handling, JSON files, functions, classes, objects, and inheritance. I utilized a ‘while’ loop to continuously print the options from the menu to the user and I used a ‘for’ loop in order to relay to the user what task they completed in case three. I used conditional statements to complete a task based on the menu option the user chose. I used lists to store Students given by the user. I used error handling to ensure that my program would not crash. I used the information the user inputted to create a JSON file with the information in it. I used functions and classes to help organize and keep my program neat. Lastly, I used objects and inheritance to creates instances of Students as opposed to storing their information in variables.