Name: Lorena Sainz-Maza Lecanda
Date: 11/19/2024
Course: Introduction to Programing with Python
GitHub: https://github.com/lorenasml/lorenasml-IntroToProg-Python-Mod06

Assignment 06 – Functions

Introduction

In this assignment, I use PyCharm IDE to write a Python script that allows us to register multiple students, process data using dictionaries and save the data in a CSV file. In this program, I introduce `functions` and docstrings. In this essay, I review the steps I took to define each function and run the program successfully. It also includes short observations about my performance (i.e. where I got stuck and how I resolved it).

Steps & Observations

1. Defining functions

First, I started by adding my processing functions to read and write the file.

```
38      @staticmethod  1 usage
39      def read_data_from_file(file_name: str, student_data: list):
40          """ Reads student data from a JSON file and updates the student_data list.
41
42          :param file_name: Name of the file to read from.
43          :param student_data: List to hold the student data loaded from the file.
44          :return: None (modifies student_data in place).
45          """
46          try:
47              file = open(file_name, "r")
48              student_data = json.load(file)
49              file.close()
50          except FileNotFoundError as e:
51              IO.output_error_messages( message: "Text file must exist before running this script!", e)
52          except Exception as e:
53              IO.output_error_messages( message: "There was a non-specific error!", e)
54          finally:
55              if file.closed == False:
56                  file.close()
57          return student_data
```

`read_data_from_file` opens the `file_name` in "read" mode and uses the `json.load` built-in function. The function is defined with two paramenters, one that holds the name of the file to read from and another one for the list of student data. The functions ends with a return statement; if I didn't add this, the program failed with a `TypeError: 'NoneType' object is not iterable` in lines 153 and 172 when trying to output student courses. Instead of

`return`, I believe I could have used `extend` instead (i.e. `student_data.extend(json.load(file))`)

```python
    @staticmethod  1 usage
    def write_data_to_file(file_name: str, student_data: list):
        """ This method writes student data to a JSON file and displays the enrolled students.

        :param file_name: Name of the file where student data will be written.
        :param student_data: List of student data dictionaries to write to the file.
        :return: None
        """
        try:
            with open(file_name, "w") as file:
                json.dump(student_data, file, indent=4)
        except KeyError as e:
            IO.output_error_messages( message: "Please make sure your dictionary key exists!", e)
        except Exception as e:  # catch all
            IO.output_error_messages( message: "There was a non-specific error!", e)
        finally:
            if file.closed == False:
                file.close()
        print("The following students have been successfully enrolled:")
        with open(FILE_NAME, "r") as file:
            for row in file:
                print(row.strip())
```

`write_data_to_file` opens the file in "write" mode and uses the built-in function `json.dump`. Just like for the read function, it also takes 2 parameters (the file where the names are written and the list of student data dictionaries to write to the file). I used the with `open(…)` statement, which doesn't require you to manually close the file. In this step, I also ask this function to read over the file and print the list of student data dictionary in `json` format.

```
Enter your menu choice number: 3
The following students have been successfully enrolled:
[
{
"student_first_name": "Michael",
"student_last_name": "Hollis",
"course_name": "Python 200"
},
{
"student_first_name": "Lucas",
"student_last_name": "Hollis",
"course_name": "Python 200"
}
]
```

After defining my functions to process the data, I went ahead and defined the functions to present student input and output.

`def output_error_messages`

```
87      @staticmethod  6 usages
88      def output_error_messages(message: str, error: Exception = None):
89          """ This function displays a custom error message to the user and optionally includes t
90
91          :param message: Custom error message to display to the user.
92          :param error: Optional exception object containing error details.
93          :return: None
94          """
95          print(message, end="\n\n")
96          if error is not None:
97              print("-- Technical Error Message -- ")
98              print(error, error.__doc__, type(error), sep='\n')
```

The `output_error_messages` function prints a custom error message and adds a conditional statement that optionally prints a more technical error message.

def output_menu

```
101     @staticmethod  1 usage
102     def output_menu(menu: str):
103         """ This function displays a menu of choices to the user.
104
105         :param menu: A list of menu choices to display.
106         :return: None
107         """
108         print(MENU)
```

The `output_menu` function passes menu as an argument that makes the function accept a list of menu choices.

def input_menu_choice

```
107     @staticmethod  1 usage
108     def input_menu_choice():
109         """ This function gets a menu choice from the user
110
111         :return: string with the users choice
112         """
113         choice = input("Enter your menu choice number: ")
114         return choice
```

This function gets the menu choice from the user. Note, if I didn't wrap the function with a `return choice` statement, the program would continue to display the menu options rather than executing the next step in the while loop.

def input_student_data

```python
    @staticmethod  1 usage
    def input_student_data(student_data: list):
        """ This function gets first name, last name, and course name information from the user.

        :param student_data: A list to store student information
        :return: None
        """
        try:
            student_first_name = input("What is the student's first name? ")
            if not student_first_name.isalpha():
                raise ValueError("The first name should not contain numbers.")

            student_last_name = input("What is the student's last name? ")
            if not student_last_name.isalpha():
                raise ValueError("The last name should not contain numbers.")

            course_name = input("Which course are you enrolled in? ")
            # Create the student dictionary
            student_data = {"student_first_name": student_first_name,
                            "student_last_name": student_last_name,
                            "course_name": course_name}
            # Append the student data to the list
            students.append(student_data)
            print("Student data successfully added.")
        except ValueError as e:
            IO.output_error_messages( message: "That value is not the correct type of data!", e)
        except Exception as e:  # catch-all
            IO.output_error_messages( message: "There was a non-specific error!", e)
```

The code inside this function is very similar to the one I wrote for Assignment05. It takes in one parameter (`student_data`) to hold the list of student dictionaries. I also use the `output_error_messages` function, rather than printing the entire error message code twice.

def output_student_courses

```python
    @staticmethod  1 usage
    def output_student_courses(student_data: list):
        """ This function displays the output of student's information

        :param student_data: List of student data dictionaries.
        :return: None
        """
        global students

        print("-" * 50)
        print(student_data)
        print("-" * 50)
        for student in students:
            print(f"{student['student_first_name']}, {student['student_last_name']}, {student['course_name']}")
        print("-" * 50)
```

Lastly, I define the function that outputs the student's information. I provide the output in two formats, a list of student dictionaries and a comma-separated string.

2. The program script

```
# Beginning of the main body of this script
students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)

# Repeat the follow tasks
while True:
    IO.output_menu(menu=MENU)

    menu_choice = IO.input_menu_choice()

    if menu_choice == "1":
        IO.input_student_data(student_data=students)
        continue

    elif menu_choice == "2":
        IO.output_student_courses(student_data=students)
        continue

    elif menu_choice == "3":
        FileProcessor.write_data_to_file(file_name=FILE_NAME, student_data=students)
        continue

    elif menu_choice == "4":
        break  # out of the while loop
```

The first line `students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)` loads student data from "Enrollments.json" at the start of the program. The `students` variable is passed to `read_data_from_file`. Note that I had to add the class `FileProcessor` to the function in order to execute this line, otherwise I got `NameError: name 'read_data_from_file' is not defined` error message. After than, I introduce the `while` loop, which ensures that the program keeps running until the user chooses option 4 (Exit). The `IO.output_menu(menu=MENU)` function displays the menu, and `IO.input_menu_choice()` prompts the user for their selection. The program iterates through 4 different menu options:

- **Option 1**: Adds a new student to the students list and uses the `input_student_data` function.
- **Option 2**: Displays the current student courses and uses the `output_student_courses` function.
- **Option 3**: Saves the student data back to a file with the file processing function `write_data_to_file`.
- **Option 4**: Exits the loop and terminates the program.

The program runs successfully in PyCharm and Terminal.

Summary & Reflection

To summarize, Task 6 took me approximately 6 hours. I was not very sure what parameters I needed to add to each function, but I think this was a bit clearer once I started running the actual program script and debugging. At first, I had also defined several global variables, but I realized I could define them locally instead. To add docstrings, I leveraged ChatGPT. Module06-Lab3 was extremely helpful to figure out how to write each function and run the program.