Natalie Ferri
August 05, 2024
Foundations Of Programming: Python
Assignment 06
https://github.com/nferri/IntroToProg-Python-Mod06

Creating Functions for the Python Program

Introduction

This document provides insights into how I created Python code to process and save data to a JSON file using user input, lists, and a dictionary grouped by classes and functions.

Writing A Registration Program

The course builds upon the previous week's code to introduce new techniques. This week, we continued writing to a JSON file and using error handling but introduced classes and functions to execute these operations. The assignment is to create a Python program that uses classes and functions for the input, open, write/dump, close, lists, dictionaries, print, and error handling functions to capture data from a user and display messages to a user about course registration. The documentation will be broken into three sections: assigning variables, executables, and testing.

Assigning Variables

Before defining constants, we must import the JSON module to work with the JSON file. Using the outline from the assignment deliverables, I assigned the constant variables for FILE_NAME and MENU. FILE_NAME defines the name of the JSON to be written to by the program from user input. MENU is an open string and will have text written for the output within the logic statement.

```
import json

# Define the Data Constants
FILE_NAME: str = "Enrollments.json"

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.

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

'''
```

Instead of using predefined variables in the header of the script, we introduced global variables in

the header script and local variables within the functions. This week, we have fewer global variables because we incorporate previously used variables in the specific functions later in the script.

```
# Define the Data Variable
menu_choice: str = "" # choice made by the user.
students: list = [] # a table of student data
```

Executable

Previously, the first line of the executable, “try”, established our first function. This week, we begin with a class to process the file. Within the class FileProcessor are two defined functions, read/write data to/from file. Each class and function uses a comment block to detail what the function does, its local variables, and when it was last updated. Within the defined function, we indicate the variables to use in the initial line (our argument) 3and then tell the code what to run in a try statement. If errors are encountered, the code for the function will report the errors to the user. The function will end with a finally statement to close the file if it is still open. Then it returns the list information within the dictionary information. A selection of the code can be viewed in Figure 1.

```
class FileProcessor:

    '''
    This class reads and writes to JSON files.
    '''

    # Open the JSON
    @staticmethod
    def read_data_from_file (file_name: str, student_data: list):

        '''
        This function loads data from a JSON to a dictionary. Error prints if file
            does not exist or issue reading.

        :param file_name: name of the JSON file
        :param student_data: list of student data
        :return: list of student data

        Change Log:
            Natalie Ferri, 08/03/2024, incorporate IO error
        '''

        try:
            file = open (file_name, "r")
            student_data = json.load (file)
            file.close()

        except FileNotFoundError as e:
            IO.output_error_messages(message="File does not exist!", error=e)

        except Exception as e:
            IO.output_error_messages(message="Error reading the file.", error=e)

        finally:
            if not file.closed:
                file.close()

        return student_data
```

Figure 1: class FileProcessor

After class FileProcessor is class IO, which has multiple functions that help interact with the user. The functions are presenting the menu, allowing user input, displaying the data, and presenting errors. A selection of the class IO code can be viewed in Figure 2. In input_student_data we defined

that for inputting student data, we need to use the student_data as a list to process and append to the student dictionary. We still use the input function and have error handling established from the previous week. This will be executed in an if/else loop established for the menu choices. Instead of writing the entire code out again in the elif statement, we can simply call the input_student_data function. As a continuation of the error handling theme used last week, I kept the .isupper() AttributeError. When working with CSV data, I had the script convert the name and course input by the user into all caps. I could do the same within the script but kept this AttributeError for fun.

```
    # User input of student data
    @staticmethod
    def input_student_data (student_data: list):
        '''
        This function allows user to input student data.

        :param student_data: List of student data
        :return: updated list of student data

        ChangeLog:
            Natalie Ferri, 08/04/2024, created function for user input
        '''

        try:
            student_first_name = input ("Enter the student's first name in all caps: ").strip(
            if not student_first_name.isupper():
                raise AttributeError ("Error: The first name should be in all caps.\n")

            student_last_name = input ("Enter the student's last name in all caps: ").strip()
            if not student_last_name.isupper():
                raise AttributeError ("Error: The last name should be in all caps.\n")

            course_name = input ("Enter course name: ")

            student = {"FirstName": student_first_name,
                       "LastName": student_last_name,
                       "CourseName": course_name}

            student_data.append(student)

            print (f"You have registered {student_first_name} {student_last_name} for {course_

        except AttributeError as e:
            IO.output_error_messages (message="\nERROR", error=e)

        except Exception as e:
            IO.output_error_messages (message="\nError: There was a problem with your entered

        return student_data
```

Figure 2: Class IO

The script has two classes that have been explained. Now we need to create the if/else statements to call the functions within the classes. We are still using the same menu choices, but the if/else statements are now looking at single-line function calls to execute functions previously defined. As you can see (Figure 3) in choice 1: the if statement is the same for if a user types “1,” then we want to register them. The student = IO.input_student_data will run asking for the user's first and last names and course name. It will then save this data to the local computer log in case the user selects Menu Choice 3 to save the data to the JSON file. This function can be seen in Figure 2.

```
# Present and Process the data
'''
This code defines the menu choices to choose from
    Change log:
        Natalie Ferri, 08/04/2024, defining menu_choice items
'''
while (True):

    # Present the menu of choices
    IO.output_menu (menu=MENU)
    menu_choice = IO.input_menu_choice()

    # 1. Register a student for a course.
    if menu_choice == "1":
        students = IO.input_student_data (students)
        continue

    # 2. Show current data.
    elif menu_choice == "2":
        IO.output_student_courses (students)
        continue

    # 3. Save data to a file.
    elif menu_choice == "3":
        FileProcessor.write_data_to_file (FILE_NAME, students)
        continue


    # 4. Exit the program.
    elif menu_choice == "4":
        break  # Out of the loop

    else:
        print ("Please only choose option 1, 2, 3, or 4.")
```

Figure 3: Menu Choices

Testing

I struggled through Assignment06.py because using the functions made it hard for me to follow the variables and logic. I was able to get the script formatted, but when I would run menu_choice 2, the function IO.output_student_courses would have a “TypeError: string indices must be integers, not 'str'.” I tried changing the print statement using an index to call the list, which was unsuccessful. The index would print, but as single letters not full names. I tried changing the variables used within the function and would get None errors. I was focused on changing the code within the class that I didn’t think the issue could be with the else statement. I previously had for the elif statement for menu_choice 2, “students = IO.output_student_courses (student_data = students).” When I removed “students” from the beginning, the script ran properly as a printing a string from a dictionary. It took me three days to find the answer to my issue and that is why I do not like functions. I tried debugging the script and it didn’t lead me to find an answer. The correction was found after asking for help. Once menu_choice 2 ran correctly, the remainder of the script executed without issues (Figure 4). That was a relief!

```
---- 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.
----------------------------------------


Enter menu item: 1
Enter the student's first name in all caps: NATALIE
Enter the student's last name in all caps: FERRI
Enter course name: SWIMMING
You have registered NATALIE FERRI for SWIMMING.


---- 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.
----------------------------------------


Enter menu item: 1
Enter the student's first name in all caps: NAT
Enter the student's last name in all caps: ATAT
Enter course name: CRYING
You have registered NAT ATAT for CRYING.


---- 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.
----------------------------------------


Enter menu item: 2
Student NATALIE FERRI is enrolled in PYTHON2
Student N N is enrolled in V
Student NATALIE FERRI is enrolled in SWIMMING
Student NAT ATAT is enrolled in CRYING

---- 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.
----------------------------------------


Enter menu item: 3
Student NATALIE FERRI is enrolled in PYTHON2
Student N N is enrolled in V
Student NATALIE FERRI is enrolled in SWIMMING
Student NAT ATAT is enrolled in CRYING
{'FirstName': 'NATALIE', 'LastName': 'FERRI', 'CourseName': 'PYTHON2'}
{'FirstName': 'N', 'LastName': 'N', 'CourseName': 'V'}
{'FirstName': 'NATALIE', 'LastName': 'FERRI', 'CourseName': 'SWIMMING'}
{'FirstName': 'NAT', 'LastName': 'ATAT', 'CourseName': 'CRYING'}

---- Course Registration Program ----
  Select from the following menu:
    1. Register a student for a course.
```

Figure 4: Success!

Summary

In conclusion, I detailed my process for writing a Python script that uses defined functions within classes to allow the user to interact with the program and how to incorporate lists to store and call data. The script uses the user's input to save data to a JSON file and presents it to them in print statements. Multiple users can submit information to be saved since the loop allows the variables to be reassigned as the user inputs names. The JSON uses the dump function to write each new input saved. Finally, the script concludes when the user selects the break function for the code.

exit() # end writeup