Tom Lowder

11/19/2024

IT FDN 110 A

Assignment06

GitHub Repository: https://github.com/tlowder24/Python110-Fall2024

Functions

Introduction

Module06 explored the building blocks of programming, functions and classes. The module delved into the distinction between global vs local variables, parameters, return values, and separation of concerns pattern.

Creating the program

The objective of Assignment06 was to modularize the Assignment06-Starter program to include classes and functions. The remainder of the paper describes modfications made to the program to achieve the task.

First off, the program header was modified for the Assginment06-Starter program.

Then the buidling blocks of the new program, the class names, functions names, and parameters used in each function, were added. The following structure was added to the program after the defined variables.

```
Class FileProcessor:
        read_data_from_file(file_name: str, student_data: list):
        write_data_to_file(file_name: str, student_data: list):
Class IO:
        output_error_messages(message: str, error: Exception = None):
        output_menu(menu: str):
        input_menu_choice():
        output_student_courses(student_data: list):
        input_student_data(student_data: list):
```

Since it is common practice to include a docstring (header) at the begginning of each class or function, these were included in the program. Information in the docstring inlcuded a description of the class or function, changelog, parameter description (for functions) and return value (for functions). An example of a class and function docstring used in the program are shown in Figure 1 and figure 2.

```
26  # File Processing Class ----------------------------
27  class FileProcessor:  2 usages
28      """
29      Class of functions that process JSON file data
30
31      ChangeLog: (Who, When, What)
32      Tom Lowder,11.19.2024,Created Class
33      """
```

Figure 1. Class docstring

```
    @staticmethod  1 usage
    def read_data_from_file(file_name: str, student_data: list):
        """ Function reads data from a JSON file and loads it into a list of list table

        ChangeLog: (Who, When, What)
        Tom Lowder,11.19.2024,Created function

        :param file_name: string data = file name to read from
        :param student_data: list of dictionary rows to be filled with file data

        :return: student_data list
        """
```

Figure 2. Function docstring and @staticmethod

Additionally, all function code in this program never change or are “static”, so the @staticmethod decorator is used and placed above each function. This allows the use of class functions directly without having to create an operator. This is shown in Figure 2.

After that, the following starter code was cut and pasted into the respective functions.

```
try:
    file = open(FILE_NAME, "r")
    students = json.load(file_name)

    file.close()
except Exception as e:
    print("Error: There was a problem with reading the file.")
    print("Please check that the file exists and that it is in a json format.")
    print("-- Technical Error Message -- ")
    print(e.__doc__)
    print(e.__str__())
finally:
    if file.closed == False:
        file.close()
```

Figure 3. For read_data_from_file(file_name: str, student_data: list):

```
        try:
            file = open(FILE_NAME, "w")
            json.dump(students, file)

            file.close()
            print("The following data was saved to file!")
            for student in students:
                print(f'Student {student["FirstName"]} '
                      f'{student["LastName"]} is enrolled in {student["CourseName"]}')
        except Exception as e:
            if file.closed == False:2

                file.close()
            print("Error: There was a problem with writing to the file.")
            print("Please check that the file is not open by another program.")
            print("-- Technical Error Message -- ")
            print(e.__doc__)
            print(e.__str__())
        continue
```

Figure 4. For write_data_to_file(file_name: str, student_data: list):

```
56      # Present the menu of choices
57      print(MENU)
```

Figure 5. For output_menu(menu: str):

```
58      menu_choice = input("What would you like to do: ")
```

Figure 6. For input_menu_choice():

```
92          print("-" * 50)
93          for student in students:
94              print(f'Student {student["FirstName"]} '
95                    f'{student["LastName"]} is enrolled in {student["CourseName"]}')
96          print("-" * 50)
97          continue
```

Figure 7. For output_student_courses(student_data: list):

```
63          try:
64              student_first_name = input("Enter the student's first name: ")
65              if not student_first_name.isalpha():
66                  raise ValueError("The last name should not contain numbers.")
67              student_last_name = input("Enter the student's last name: ")
68              if not student_last_name.isalpha():
69                  raise ValueError("The last name should not contain numbers.")
70              course_name = input("Please enter the name of the course: ")
71              student_data = {"FirstName": student_first_name,
72                              "LastName": student_last_name,
73                              "CourseName": course_name}
74              students.append(student_data)
75              print(f"You have registered {student_first_name} {student_last_name} for {course_name}.")
76          except ValueError as e:
77              print(e)  # Prints the custom message
78              print("-- Technical Error Message -- ")
79              print(e.__doc__)
80              print(e.__str__())
81          except Exception as e:
82              print("Error: There was a problem with your entered data.")
83              print("-- Technical Error Message -- ")
84              print(e.__doc__)
85              print(e.__str__())
86          continue
```

Figure 8. For input_student_data(student_data: list):

Output_error_messages(message: str, error: Exception = None): function was then written, Figure 9, to be used to present exception error messages to user.

```
108         print(message, end="\n\n")
109         if error is not None:
110             print("-- Technical Error Message -- ")
111             print(error, error.__doc__, type(error), sep='\n')
112
```

Figure 9. Output_error_messages function code

Next, the error handling code in each function was changed to use the output_error_messages function.

```
51      except Exception as e:
52          IO.output_error_messages(message="Error: There was a problem with reading the file.", error=e)
53
```

Figure 10. For read_data_from_file(file_name: str, student_data: list):

```
78      except Exception as e:
79          message = "Error: There was a problem with writing to the file.\n"
80          message += "Please check that the file is not open by another program."
81          IO.output_error_messages(message=message,error=e)
```

Figure 11. For write_data_to_file(file_name: str, student_data: list):

```
189     except ValueError as e:
190         IO.output_error_messages(message="One of the values was the incorrect type of data!", error=e)
191     except Exception as e:
192         IO.output_error_messages(message="Error: There was a problem with your entered data.", error=e)
193     return student_data
```

Figure 12. For input_student_data(student_data: list):

For input_menu_choice(): function, the else state from the starter file was incorporated as a raise exception in the function. Additionally, e.__string__() was used in the ouput_error_message function to not pass e and avoid a technical error message.

```
135     choice = "0"
136     try:
137         choice = input("Enter your menu choice number: ")
138         if choice not in ("1","2","3","4"):  # Note these are strings
139             raise Exception("Please, choose only 1, 2, 3, or 4")
140     except Exception as e:
141         IO.output_error_messages(e.__str__())  # Not passing e to avoid the technical message
142
143     return choice
```

Figure 13. Input_menu_choice(): function

After that, the for loop in the For write_data_to_file(file_name: str, student_data: list): function was replaced with the output_student_courses function as shown in Figure 14.

```
72          try:
73              file_name = open(FILE_NAME, "w")
74              json.dump(student_data, file_name)
75              file_name.close()
76              print("The following data has been saved:")
77              IO.output_student_and_course_names(student_data=students)
```

Figure 14. Output_student_courses function used in write_data_to_file function.

The next step was to clean up the formatting of the function data and variables. This included changing the old variables pasted in the function to the new parameters, removing variables not used, and adding extra space to make the program output neatly.

Lastly the main body of the code was simplified by including the functions.

```
196 # Main Body
197 # When the program starts, read the file data into a list of lists (table)
198 # Extract the data from the file
199 students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)
200
201 # Present and Process the data
202 while (True):
203
204     # Present the menu of choices
205     IO.output_menu(menu=MENU)
206
207     menu_choice = IO.input_menu_choice()
208
209     # Input user data
210     if menu_choice == "1":  # This will not work if it is an integer!
211         students = IO.input_student_data(students)
212         continue
213
214     # Present the current data
215     elif menu_choice == "2":
216         IO.output_student_courses(students)
217         continue
218
219     # Save the data to a file
220     elif menu_choice == "3":
221         FileProcessor.write_data_to_file(file_name=FILE_NAME, student_data=students)
222         "Data Saved"
223         continue
224
225     # Stop the loop
226     elif menu_choice == "4":
227         break  # out of the loop
228
229 print("Program Ended")
```

Figure 15. Main body of program with functions.

Testing the program

The program was tested in PyCharm and the command prompt. Tests included:

- The program takes the user's input for a student's first, last name, and course name.
- The program displays the user's input for a student's first, last name, and course name.
- The program saves the user's input for a student's first, last name, and course name to a coma-separated string file.
- The program allows users to enter multiple registrations (first name, last name, course name).
- The program allows users to display multiple registrations (first name, last name, course name).
- The program allows users to save multiple registrations to a file (first name, last name, course name).

The program successfully ran and saved to Enrollments.json in both PyCharm and the command prompt.

I saved the program and this assignment to my GitHub repository so others can review my work.

Summary

With the resources provided in module 06 I was able to create the program. The program demonstrates my new understanding of how to use classes and functions to modularize code.