Josh Huang

12/14/2023

IT FDN 110 A

Assignment 6

Assignment 6 – Functions

Introduction

This assignment further expands on assignment 5. Additional refinements were made through the usage of classes and functions, as well as the separation of concerns. These practices help to break the program into more manageable blocks of code. Much of the logic is encapsulated within functions, which acts as a reusable, modular block of code.

Classes

Classes are used as an organizational tool that helps maintain the code as well as help other developers read the code. A class is typically followed by a docstring that has a brief description of what the class is, as well as a change log. The definition of a class is important as it gives context to the grouping for the functions within.

Separation of Concerns

The Separation of Concerns, or SoC, is a design principle that breaks down code into self-contained parts, each which play a role in the functionality of the program. A concern is a specific responsibility of a program's functionality. In this assignment, the concerns are broken out into input/output and file processing (FileProcessor class, and IO class).

Within each of these classes are a list of functions, broken out into groups, taken from the code requirements. The division and separation of these concerns was based on which functions were related to giving/requesting user information (Figure 1), and which functions were related to processing data and manipulating files (Figure 2). In reference to the function requirements, the separation is as follows.

Within IO: output_error_messages, output_menu, input_menu_choice, output_student_courses, input_student_data.

Within File Processor: read_data_from_file, write_data_to_file.

```
class IO:
    """
    Class IO consists of functions that manage user input and output
    ChangeLog: (Who, When, What)
    Joshh, 12-14-2023 Created Class
```

Figure 1: IO (Input/Output)

```
class FileProcessor:

    """
    class: FileProcessor is a collection of processing functions that work with Json files
    ChangeLog: (Who, When, What)
    Joshh, 12-14-2023, Created class
    """
```

Figure 2: FileProcessor

Reading Data from File

The addition of modular sections changes the way that data is read from the existing file into the program.

In the 'main' block of code, 'students' calls upon the class 'FileProcessor' for function 'read_data_from_file' to populate the list of dictionaries. This function is called with FILE_NAME and student as its arguments. Within this function, the existing data is loaded with the use of 'json.load' (Figure 3), and all data is appended. This is done with a for loop to add each new dictionary (or individual student information) to the list. This differs from the previous assignment in that we are now calling on a different block of code in the initialization process, before any user input is requested.

```
            list_of_dictionary_data = json.load(file)
131         for student in list_of_dictionary_data:
132             student_data.append(student)
133             file.close()
```

```
student_data = {list: 3} [{'CourseName': '399', 'FirstName': 'lol', 'LastName': 'kok'}, {'CourseName': 'P34', 'FirstName': 'Bob', 'LastName': 'Ross'}, {'CourseName': 'PY100', 'FirstName': 'Sam', 'LastName': 'Banks'}]
  0 = {dict: 3} {'CourseName': '399', 'FirstName': 'lol', 'LastName': 'kok'}
  1 = {dict: 3} {'CourseName': 'P34', 'FirstName': 'Bob', 'LastName': 'Ross'}
  2 = {dict: 3} {'CourseName': 'PY100', 'FirstName': 'Sam', 'LastName': 'Banks'}
  __len__ = {int} 3
```

Figure 3: Loading Existing data from the file

Error Exceptions

Throughout the program, upon raising an exception, the class 'IO' is called for the error message functions. As an example, for ValueErrors raised during the user input process, such as within the IO class (Figure 4). This modular structure would allow an editor to modify the error message for the entire program in a centralized manner and would help keep consistency throughout the program.

```
100         except ValueError as e:
101             IO.output_error_messages( message: "That value is not the correct type of data!",
102         except Exception as e:
103             IO.output_error_messages( message: "There was a non-specific error!", e)
104
```

Figure 4: Calling the 'output_error_messages' function

Static Classes

Throughout the program, the @staticmethod decorator is used (Figure 5) so that Python will allow the usage of the class functions directly, without the need to define an object first. This is useful because it allows the function to be 'static', in that it never changes.

```
@staticmethod
def write_data_to_file(file_name: str, student_data: list):
```

Figure 5: @staticmethod decorator that precedes each function

Source Control

The script file and the knowledge document are hosted on the following GitHub repository.

https://github.com/jjoshn/IntroToProg-Python-Mod06

Summary

Using the three practices: classes, functions, and separation of concerns results in a program that is more modular and reusable than before. This allows for more complex code to be more manageable, not only for the code writer, but also for code review and code testing. This method of organization also makes it easier to compare against the requirements, and I am excited to see how this would apply to even more complicated projects.