Miyles Clark
8/7/2024
Foundations of Programming: Python
Assignment 06
https://github.com/macxuw/IntroToProg-Python-Mod06/blob/main/Assignment06.py

Classes, Functions, and Separation of Concerns

Introduction

Assignment 06 uses a while loop to present course registration options, process the user's menu choices, and write a students table to a JSON file. When the program begins, it reads and appends existing JSON data into a list of student_data dictionary values. The program will accept multiple course registrations and present all data in the students list.

The user is prompted to choose from a menu:

- Enter new data.
- Display student data.
- Write student data to a JSON file.
- Exit the program.

The main difference from the last assignment is the use of classes and functions to compartmentalize the code into separate areas of concern. Error handling in Assignment 06 employs the same Try-Except code seen in the previous assignment and stores it in a separate function.

Adding Code

Since the base program code for this assignment is identical to Assignment 5, the Module 6 materials included a starter script. Classes and functions are implemented to add modularity and separation of concerns to a program; in this assignment, the main classes are FileProcessing and IO (Input/Output). The FileProcessing class includes functions to read and write to a JSON file, while all the functions that take or display user data are under the IO class.

Lists and Collections

The program requires an existing JSON file, and upon initialization, the file data is loaded into the student_data dictionary (Figure 1).

```
file = open(file_name, "r")
```

```
student_data = json.load(file)
file.close()
```

Figure 1: Code to load existing data into the student_data dictionary

To input a new student registration, enter the number 1 and follow the prompts for specified student data. The program uses an f-string to display verification after every registration (Figure 2).

```
print(f"{student_first_name} {student_last_name} is enrolled in
        {course_name}")
```

Figure 2: Registration verification code

Another line of code associates the variables to the student_data dictionary keys (Figure 3).

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

Figure 3: Variables and associated dictionary keys

Menu choices two and three display all data in the students table and write – dump – all student data back to the specified file (Figure 4).

```
        file = open(file_name, "w")
json.dump(students, file, indent=1)
```

Figure 4: Save the students list to JSON file

Dumping data into a JSON file results in an unformatted file with enclosed “rows” of student_data on a single line. Setting the indent level to 1 positions each key and associated value on a separate line (Figure 5).

```
{
 "FirstName": "Chandler",
 "LastName": "Bing",
 "CourseName": "Python 202"
}
```

Figure 5: Formatted JSON data

Classes and Functions

As scripts increase in complexity, the importance of classes, functions, and the separation of functions becomes clear. By adding the modularity of classes and functions, coders lower the chances of an errant edit breaking a program. Classes and functions also facilitate collaboration. Coding teams can work on specific classes or related functions before rejoining a larger group to test and finalize program code.

Program modularity adds another level of documentation to guide the individual or team responsible for program edits, especially when the program creator or creators are no longer involved with a project. Where classes establish a function's layer of concern, functions create a self-contained code that can be used in multiple places in the body of a program's code.

For Assignment 06, the FileProcessor and IO classes organize functions into those responsible for JSON file processing and those accepting user inputs or displaying data to the user. Instead of defining global data into one or more functions, parameters create a bridge between global data and the data required for each encapsulated function. The MENU constant is passed into the output_menu function as the local variable menu (Figure 6), which is then called as an argument when implanting the function in the program body (Figure 7).

```
def output_menu(menu: str)
```

Figure 6: Setting the menu parameter

```
IO.output_menu(menu=MENU)
```

Figure 7: Calling the menu argument

Setting parameters and calling the argument decreases the likelihood of errors caused by changes to global data. If, for example, I want to add menu options, every function calling the MENU argument will automatically change. Let's say the programmer wants to add options for a student menu and a faculty menu; the menu parameter remains, and depending on the organization's needs, an IT department can set the argument to MENU1 or MENU2.

Error Handling

The Try-Except error code in Assignment 06 mimics that of the previous assignment, with the main difference being a separate function for general errors. Error messages related to specific instances — requiring input values to be only alphabetic or over a certain length, for example — reside within the associated function (Figure 8).

```
if not student_first_name.isalpha() or len(student_first_name) < 2:
…
```

```
except ValueError as e:
        IO.output_error_messages(e)
```

Figure 8: Error message handling within a function

As error handling occurs in every function, there is also an output_error_messages function within the IO class, as shown in Figure 8.

Disability Accessibility

Python code can improve ease of use for disability users in several ways. Like previous assignments, all user inputs have the .strip() string method attached to strip extra whitespace on either side of the input. Custom error messages display the specific parameters for acceptable inputs (Figure 9).

```
        menu_choice = input("Enter your menu choice number: ").strip()
if menu_choice not in ("1", "2", "3", "4"):
  raise Exception("Please choose an option from the menu.")
```

Figure 9: Custom message for valid input

The sole addition from the previous assignment adds formatting code to bold requested inputs in menu option 1 (Figure 10).[1]

```
course_name = input("Please enter the \033[1mcourse name\033[0m:
        ").strip()
```

Figure 10: Highlighting requested output

The sequence to activate the ANSI escape code for bold text is \033[1m, while \033[0m turns bold text off.[2] Note that the code uses a binary base: the first series ends in 1m and the latter with 0m.

Results

Testing my program in PyCharm (Figure 11) and Terminal (Figure 12) returned the correct outcome, whether a prompt for data, a print statement, or an end to the program.

---

[1] https://www.kodeclik.com/how-to-bold-text-in-python/
[2] Ibid.

```
---- 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 your menu choice number: 1
Enter the student's first name: Chandler
Enter the student's last name: Bing
Please enter the course name: Python 202

What would you like to do: 2
------------------------------------------------
Bob Smith is enrolled in Python 100
Sue Jones is enrolled in Python 100
Chandler Bing is enrolled in Python 202
------------------------------------------------

Enter your menu choice number: 3
The following data has been saved:

------------------------------------------------
Bob Smith is enrolled in Python 100
Sue Jones is enrolled in Python 100
Chandler Bing is enrolled in Python 202
------------------------------------------------

What would you like to do: 4

Program Ended

What would you like to do: w
Please choose a number from the menu.
```

Figure 11: Program output in PyCharm

```
---- 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 your menu choice number: 1
Enter the student's first name: Chandler
Enter the student's last name: Bing
Please enter the course name: Python 202

Enter your menu choice number: 2
------------------------------------------------
Bob Smith is enrolled in Python 100
Sue Jones is enrolled in Python 100
Chandler Bing is enrolled in Python 202
------------------------------------------------
```

```
Enter your menu choice number: 3
The following data is saved:

-----------------------------------------
Bob Smith is enrolled in Python 100
Sue Jones is enrolled in Python 100
Chandler Bing is enrolled in Python 202
-----------------------------------------

Enter your menu choice number: 4
Program Ended
Enter your menu choice number: s
Please choose an option from the menu.
```

Figure 12: Program output in Terminal

The JSON file for this assignment returns a student's first and last name and a course name. The data is listed next to the corresponding dictionary key (Figure 13).


```
[
 {
  "FirstName": "Bob",
  "LastName": "Smith",
  "CourseName": "Python 100"
 },
 {
  "FirstName": "Sue",
  "LastName": "Jones",
  "CourseName": "Python 100"
 },
 {
  "FirstName": "Chandler",
  "LastName": "Bing",
  "CourseName": "Python 202"
 }
]
```


Figure 13: JSON file data

Summary

Encapsulating program code within functions and separating functions into classes according to the layer of concern creates a modular script less prone to errors. Adding to this separation of concerns is the use of parameters and arguments; parameters set necessary local data for each function, while arguments will make the association between local and global data. An individual or organization can complete a more extensive program in clear phases without the errors that can occur from relying only on global data definitions.