

Bryan Pannell

5/22/2024

IT FDN 110 A

Assignment 06

<https://github.com/bpannell12/IntroToProg-Python-Mod06>

Classes and Functions

Introduction

Just as things are getting more complex, they are also getting more streamlined and more organized. This week introduced classes and functions. These allow for easier organization, less redundancy in the code, and more flexibility for using the same code over and over. By using classes and functions, organizing the code becomes easier. Classes give you the ability to bundle a group of functions together that are inter-related. Functions allow us to develop a set of instructions that can be called whenever needed by the script. Essentially, by defining these upfront, the main body of the script can be organized as a set of calls to these classes and functions. This week, we took the previous work and re-organized it using these classes and functions. The code becomes much more modular and manageable this way. For example, if the code to perform a repeated task in the script needs to be updated, if it is in a function that is called to perform that task, then it only needs to be updated once instead of multiple times. The script may be reorganized and look very different, but it still produces the same results.

Setting the Stage to Work with JSON files

To be able to work with JSON files, the script must import Python's built-in JSON capabilities. Being able to do this greatly simplifies working with this file type and allows for a wide variety of operations to be performed using the data and files.

```
7  
8  import json  
9
```

Figure 1: Simple code to import Python's built-in JSON functionality

Proper Script Structure and Comments

Proper documentation is even more important as more complex is added. In addition to the comments and pseudo code we previously recorded, specific entries for classes and functions are essential. These elements require adequate explanations for others to be able to follow the logic and breadcrumbs to understand the code and how it operates. Functions are almost like scripts within scripts and making sure that we understand how they work helps to ensure the code performs as intended.

```

19 FILE_NAME: str = "Enrollments.json"
20
21 # Define the Data Variables and constants
22 students: list = [] # a table of student data
23 menu_choice: str = '' # Holds the choice made by the user.
24
25 # Define classes
26 2 usages
27 class FileProcessor:
28     1 usage
29     @staticmethod
30     def read_data_from_file(file_name: str, student_data: list):
31         """This function reads the file data and extracts it into memory
32         Parameters are:
33             file_name: string for the file_name where the data is stored
34             student_data: list of the student data to extract
35             Return: student_data: list of the student data
36         """
37         try:
38             file = open(file_name, "r")
39             student_data = json.load(file)
40             file.close()
41         except Exception as e:
42             IO.output_error_messages(message="There was a problem with reading the file.", error=e)
43         finally:
44             if file.closed is False:
45                 file.close()

```

Figure 2: Example of pseudo code and explanatory notes

Defining Constants and Variables

Because of organizing the code into classes and functions, the definitions of constants and variables upfront are streamlined to only a few. The figure below shows the constants and variables for this version of the script.

```

9 # Define the Data Constants
10 MENU: str = ''
11 ---- Course Registration Program ----
12 Select from the following menu:
13     1. Register a Student for a Course.
14     2. Show current data.
15     3. Save data to a file.
16     4. Exit the program.
17 -----
18 '''
19 FILE_NAME: str = "Enrollments.json"
20
21 # Define the Data Variables and constants
22 students: list = [] # a table of student data
23 menu_choice: str = '' # Holds the choice made by the user.
24

```

Figure 3: Defining constants and variables

Defining the Classes and Functions

Two classes are established for this script to use to bundle functions in: FileProcessor and IO. FileProcessor is the bucket for functions that will handle read, extracting, and writing to the specified file for the data. IO is the class for the functions that will process the menu operations including option selection, user input, output, etc. Error handling is also built into functions for both classes. As an example, the code in the figure below sets up the FileProcessor class and shows the development of one of the functions stored in that class.

```
26 class FileProcessor:
27     @staticmethod
28     def read_data_from_file(file_name: str, student_data: list):
29         """This function reads the file data and extracts it into memory
30         Parameters are:
31         file_name: string for the file_name where the data is stored
32         student_data: list of the student data to extract
33         Return: student_data: list of the student data
34         """
35         try:
36             file = open(file_name, "r")
37             student_data = json.load(file)
38             file.close()
39         except Exception as e:
40             IO.output_error_messages(message="There was a problem with reading the file.", error=e)
41         finally:
42             if file.closed is False:
43                 file.close()
44         return student_data
```

Figure 3: Example of code establishing a class and function

Multiple functions can be grouped together in a class. To help the computer to interpret the function correctly and perform the needed task, parameters can be passed along with the function calls. These parameters help to define the instructions for specific functionality. For this script, here is a breakdown:

Function	Purpose	Parameter 1	Parameter 2	Return
class: FileProcessor				
read_data_from_file	Reads and extracts data from the file	file_name: str	student_data: list	student_data
write_data_to_file	Writes data to the file	file_name: str	student_data: list	None
class: IO				
output_error_messages	Process and display error messages when issues are encountered	message: str	error: Exception = None	None
output_menu	Displays the menu of choices	menu: str		None
input_menu_choice	Collects the user's menu choice	None		choice
output_student_and_course_names	Displays the current data set	student_data: list		None
input_student_data	Collects the user's data inputs	student_data: list		student_data

Looking at the table above, it becomes clear that the main modules of the script have been placed into functions that are grouped in the two classes.

The Main Body

Once all the classes and functions are set up, the main body of the script is needed to create the linkage to make everything work. The main body calls the functions in each class at the appropriate point to perform each of their assigned tasks.

The first thing that needs to be done is to read and extract the data from a specified file:

```
150 # Main body
151 # Read and extract the data from the file
152 students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)
```

Figure 4: Example code to call the function to read and extract data from a file

Just as in the previous assignment, we use a while loop to be able to account for multiple entries:

```
155 # Present and Process the data
156 while True:
157
```

Figure 5: Python code example to start a “while” loop

While the While loop continues to execute, the menu system needs to be functional. To accomplish this, the functions related to displaying and collecting the user’s menu choice are called on each loop iteration:


```
157 # Display the menu and gather the user's menu selection
158  IO.output_menu(menu=MENU)
159 |
160 menu_choice = IO.input_menu_choice()
```

Figure 6: Python code to call the menu call the menu display function and get user selection

```
----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 selection for the options above:
```

Figure 7: Menu display

Depending on the menu option selected by the user, the “if” and “elif” statement for that option triggers the function call for that specific task. The user can enter a menu option on each iteration of the loop. There is also an “else” statement to catch any invalid selections. The figure below shows this structure:

```
159 |
160 |     menu_choice = IO.input_menu_choice()
161 |
162 |     # Collect user input - menu option 1
163 |     if menu_choice == "1": # This will not work if it is an integer!
164 |         students = IO.input_student_data(student_data=students)
165 |         continue
166 |
167 |     # Display current data - menu option 2
168 |     elif menu_choice == "2":
169 |         IO.output_student_and_course_names(students)
170 |         continue
171 |
172 |     # Save the data to the JSON file
173 |     elif menu_choice == "3":
174 |         FileProcessor.write_data_to_file(file_name=FILE_NAME, student_data=students)
175 |         continue
176 |
177 |     # Stop the loop and end the script
178 |     elif menu_choice == "4":
179 |         break # out of the loop
180 |     else:
181 |         print("Please only enter a valid menu option.")
182 |
```

Figure 8: Python code example for menu processing using function calls

Running the script produces the same results as last week’s assignment, but the code is more efficiently structured. Below are samples of the output:

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

What would you like to do: 3
The following data was saved to file!
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Steve Rogers is enrolled in Ah 310
Student Tony Stark is enrolled in QP 3000
```

Figure 9: Script report for saving data to a JSON file

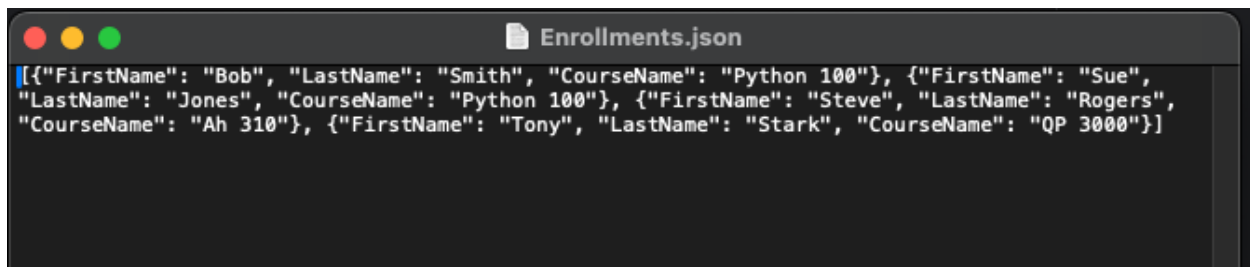


Figure 10: JSON data file sample

Once the loop is exited, adding a simple `print()` line with a message such as “Program Ended” lets the user know the session is over.

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

What would you like to do: 4
Program Ended
```

Figure 11: Sample exit message

Making sure that everything runs correctly

Error handling is built into several of the functions to handle issues that might arise. The following functions have error handling:

- `read_data_from_file()`
- `write_data_to_file()`
- `input_menu_choice()`
- `input_student_data()`

This error handling identifies issues that occur with input, output, or file handling. For error handling, the above functions call the `IO.output_error_messages()` function. Below is an example of error handling:

```

73 class I0:
123     @staticmethod
124     def input_student_data(student_data: list):
125         """ This function gets user input for the student's first name and last name, with a course name
126             Parameters are:
127                 student_data: list of dictionary data made up of the student data entered to display
128             Return: list
129         """
130         try:
131             student_first_name = input("Enter the student's first name: ")
132             if not student_first_name.isalpha():
133                 raise ValueError("The last name should not contain numbers.")
134             student_last_name = input("Enter the student's last name: ")
135             if not student_last_name.isalpha():
136                 raise ValueError("The last name should not contain numbers.")
137             course_name = input("Please enter the name of the course: ")
138             student = {"FirstName": student_first_name,
139                       "LastName": student_last_name,
140                       "CourseName": course_name}
141             student_data.append(student)
142             print()
143             print(f"You have registered {student_first_name} {student_last_name} for {course_name}.")
144         except ValueError as e:
145             I0.output_error_messages(message="An error occurred because of an invalid entry data type.", error=e)
146         except Exception as e:
147             I0.output_error_messages(message="There was a technical error.", error=e)
148         return student_data

```

Figure 12: Sample of structured error handling message built into a function

Summary

This week's assignment took the existing script to a new level. Adding in classes and functions gives a lot more organization and flexibility to the code. Using these powerful new tools enables a more modular approach to the structure of the script. It allows for the reuse of the code in multiple areas of the script without having to replicate the same code over and over. While the overall functionality and performance of the script are the same as last week, the code looks and feels different.