

Alexander Clark

20Nov2023

IT FDN 110 B

Assignment06

<https://github.com/arclark360/IntroToProg-Python-Mod06/>

MOD 6 Functions

Introduction

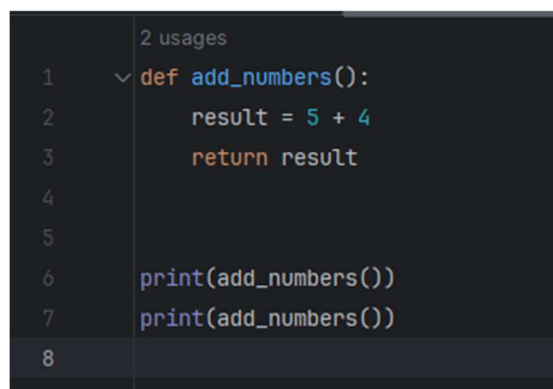
In Module 6 of our Introduction to Python course, we delved into the art of crafting code with a increased sense of modularity. This process included the exploration of functions and classes. Functions are used to define specific tasks that a program will perform, while classes are the blueprints for the creation of objects.

Armed with this new understanding of code modularity, we applied these newfound principles to enhance the student registration application.

Functions

In Python, a function is a block of code that executes a single task or multiple tasks. A function can be invoked multiple times within a code, allowing the repetition of the specified task. Previously, it was necessary to rewrite the code each time we wanted to perform the task. However, now you can simply call the function, saving space and streamlining the code, especially for tasks that are frequently used.

Figure 1 provides an example of a function that adds two values together and returns the result. The function is invoked twice, meaning the code block for the function runs twice without the need to duplicate the code.



```
2 usages
1  def add_numbers():
2      result = 5 + 4
3      return result
4
5
6  print(add_numbers())
7  print(add_numbers())
8
```

Figure 1. Function that will print out the number 9

Arguments

One benefit of writing functions is the ability to pass them parameters or arguments, allowing them to interact with your program. This is particularly helpful in terms of scope since functions operate locally. If you attempt to reference a variable outside the function, known as a **"global variable,"** Python may not recognize it. To enable a function to work with a global variable, you need to specify it within the function using the **'global'** keyword.

To overcome scope issues, you can use arguments to pass information from the **"outside"** using keyword parameters. **Figure 2** provides an example of how arguments are employed. In this figure, a global variable named `result` holds a list of integers. The **'add_numbers'** function, which was previously used, now includes a parameter that allows you to pass the function the results list. The local variable **'stored_int'** within the function is utilized to modify results and pass the updated list back out. This approach **eliminates the need to reference global variables** directly within the function.

```
result = [0,9]

1 usage
def add_numbers(stored_int: list[int,int]) -> list[int,int]:
    stored_int.append(5)
    return stored_int

print(result)
add_numbers(result)
print(result)
```

Figure 2. Function that takes the results list and adds 5 to the list. The first print would show [0,9] while the second print after the function is used shows [0,9,5]

Classes

Classes in Python offer yet another way to organize and encapsulate data into objects, consisting of multiple functions that define their behavior. This capability in Python enables the modeling of real-world entities in code. **Figure 3** provides an example of a class called **'Dog'**, which represents a Dog object that can be instantiated with a name and age. It includes an inner function to make the Dog object print out a bark statement when initialized. This example illustrates how functions can be grouped together within a class to create an object.

```
2 usages
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age

1 usage
    def bark(self):
        print(f"{self.name} barks!")

mydog = Dog(name="Alex", age=12)
Dog.bark(mydog)
```

Figure 3. Dog class which creates a Dog named Alex age 12 which when the bark command is used will say the dog's name and barks.

Assignment

After gaining a basic understanding of the workings of functions and classes, the next step toward improving code organization in this new modular form was to implement it in the student registration program. Most of the coding had been completed in the prior weeks, and this week primarily focused on condensing it into classes and creating defined functions.

Constants And Variables

By incorporating much of the app's functionality into specific functions, the scope of variables came into question this week. This involved reviewing all our previous variables and determining whether they could be called locally within the functions or needed to be declared as global. I opted to retain only four of the original variables as global variables. I retained the constants, the **'students'** list (as it is utilized by multiple functions), and the **'menu_choice'** string. Everything else was moved to their respective functions and utilized locally (see Figure 4).

```
# Define the Data Constants
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.
-----
"""

FILE_NAME: str = "Enrollments.json"

# Define the Data Variables
menu_choice: str = "" # Hold the choice made by the user.
students: list[dict[str, str, str]] = [] # Holds the list of dictionary's for the JSON
```

Figure 4. Global Variables and Constants for student registration program

Classes & Functions

When creating the classes and functions for the student registration program, I worked backward in the sense that I defined the functions before I created the classes. Once all the functions were created, I then organized them into the appropriate classes based on their functionality. The two classes for this program were the **IO class**, which housed any function dealing with user input/output. It's five functions were as follows:

Output_error_messages: this function housed all the print statements for any code that had an except during the Exception handling. This helped eliminate some of the repetitive code from the error handling.

Output_menu: function to print out the menu of choices

Input_menu_choice: function to take and store the user menu choice

Output_student_courses: this function housed all the code block that was menu choice 2 now as a single function to output the current list of students

Input_student_data: this function housed all the code block that was menu choice 1 now as a single function to take in another dictionary of student data

```
class IO:
    """
    Handles all functions that deal with program Input and Output
    """

    4 usages
    @staticmethod
    def output_error_messages(message: str, error: Exception = None):...

    1 usage
    @staticmethod
    def output_menu(menu: str):...

    1 usage
    @staticmethod
    def input_menu_choice():...

    1 usage
    @staticmethod
    def output_student_courses(student_matrix: list[dict[str, str, str]]):...

    1 usage
    @staticmethod
    def input_student_data(students_matrix: list[dict[str, str, str]]) -> list[dict[str, str, str]]:...
```

Figure 5. Condensed version of the IO Class.

The second Class that was decided on was the **FileProcessor class**. In this class I housed all functions that handled any processing of the JSON File. There ended up being two functions for this class:

Read_data_from_file: this function took the code block that we created from last week to read a JSON file and all the inner exception handling

Write_data_to_file: this function replace the menu choice 3 of writing to the JSON file and outputting a message to the user of which students registered for which classes.

```

class FileProcessor:
    """
    Handles all functions that deal with processing and working with the JSON File
    """

    1 usage
    @staticmethod
    def read_data_from_file(file_name: str, students_matrix: list[dict[str, str, str]]) -> list[dict[str, str, str]]: ...

    1 usage
    @staticmethod
    def write_data_to_file(file_name: str, students_matrix: list[dict[str, str, str]]): ...

```

Figure 6. Condensed version of the FileProcessor Class.

The trickiest part this week, while modifying the code to fit into the functions, was understanding the core concept behind each function. For example, not all the functions created needed to have a return statement since they didn't need to provide information to the user. Functions that ended up modifying the students list did need to include a return statement because they store local versions of the data only, and the information needs to be returned back outside to the main program for it to take effect.

Another modification needed was that all the variables called within the function and had a keyword parameter needed to be modified to use the keyword parameter. This adjustment was necessary because Python would become confused if the keyword was the same as an outside variable. For instance, the global variable **'students'** needed to be referenced in the function as **'students_matrix'**.

Overall, these changes significantly reduced the code needed for the actual program, as you'll see in the next section.

New Condensed Student Registration

As seen in **Figure 7**, by implementing functions and classes to the program it greatly reduces the code required in the working code. While a lot of the functions are only utilized once in this specific program if this program in the future required the functions more than once it would require a lot more repetitive code. By establishing functions for tasks it eliminates repetitive code writing.

```

173 # Begins the while loop for the program and won't close till option 4 is chosen
174 while True:
175     # Present the menu of choices
176     IO.output_menu(MENU)
177     # Input menu selection
178     IO.input_menu_choice()
179     # Option 1 Input user data
180     if menu_choice == "1": # This will not work if it is an integer!
181         IO.input_student_data(students_matrix=students)
182     # Option 2 Present the current data of both the list and a string version of the list
183     elif menu_choice == "2":
184         IO.output_student_courses(students_matrix=students)
185     # Option 3 Save the data to a file
186     elif menu_choice == "3":
187         # Error handling setup to catch any errors that might occur when writing to the file that will notify the user
188         # an error occurred and the data wasn't saved
189         FileProcessor.write_data_to_file(file_name=FILE_NAME, students_matrix=students)
190     # Option 4 Stop the loop
191     elif menu_choice == "4":
192         break
193     # Option 5 Catch-all
194     else:
195         print("Please only choose option 1, 2, or 3")
196 print("Program Ended")
197

```

Figure 7. Student Registration program utilizing functions and classes

Test Output

Figure 8. PyCharm

```
C:\Python\Python3.12\python.exe "C:\Users\arcla\OneDrive\Desktop\Python Mods\_Module06\Ass

---- 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: 1
Enter the student's first name: Feon
Enter the student's last name: Shu
Please enter the name of the course: Python 3

---- 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: 2

The current data is:
List Data: [{'first_name': 'Feon', 'last_name': 'Shu', 'course_name': 'Python 3'}]
String Format:
Feon,Shu,Python 3

---- 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
You have registered Feon Shu for Python 3.

---- Course Registration Program ----

What would you like to do: 3
You have registered Feon Shu for Python 3.

---- 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

Process finished with exit code 0
```

Figure 9. Command Prompt

```
PS C:\Users\arcla\OneDrive\Desktop\Python Mods\_Module06\Assignment> python.exe Assignment06.py

---- 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: 1
Enter the student's first name: Jeff
Enter the student's last name: Shu
Please enter the name of the course: Python 2

---- 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: 2

The current data is:
List Data: [{'first_name': 'Feon', 'last_name': 'Shu', 'course_name': 'Python 3'}, {'first_name': 'J
String Format:
Feon,Shu,Python 3
Jeff,Shu,Python 2

---- 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
You have registered Feon Shu for Python 3.
You have registered Jeff Shu for Python 2.

---- 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
PS C:\Users\arcla\OneDrive\Desktop\Python Mods\_Module06\Assignment>
```

Summary

Throughout this week, we solidified our understanding of functions and classes by seamlessly integrating them into our student registration application. This shift towards object-oriented programming not only enhanced the current project but also lays a strong foundation for future weeks. As our coding endeavors evolve into more intricate tasks, this newfound approach will prove invaluable.