

John Lewandowski

08/6/24

Foundations of Python

Assignment 06

<https://github.com/lewandj1/Assignment06.git>

Module 06- Functions

Introduction

This report documents some of the features I learned to use within Python regarding the sixth assignment in the Foundations of Python course at the University of Washington. Key features included were as follows: The difference between classes and functions, Global vs Local Variables, Using Parameters and Arguments, and Separation of concerns. In each section I will briefly explain what the feature is and how I implemented the feature.

The Difference between Classes and Functions

Within the uses of Python Classes and Functions have very different uses and functions. “Classes often are just a defined set of attributes and behaviors for an object” (Sep, P 2024). By defining these attributes and behaviors we then compact our code to be more efficient and organized. They often are used for defining parameters or arguments. Functions are the building blocks of code, they accomplish actions and are reusable. Typically they are designed to take the created parameters or arguments and use them as input. In this assignment Classes were used to separate the User input (IO) and the processing of the files (File Processor). These can be seen in the **Figure 1** below.

```
class FileProcessor:
    """A collection of processing layer functions that work with Json files

    ChangeLog: (Who, When, What)
    John Lewandowski, 08/05/24, Created class
    """
```

```
class IO:
    """A collection of presentation layer functions that manage user input and output

    ChangeLog: (Who, When, What)
    John Lewandowski, 08/05/24, Created class
    """
```

Figure 1: Example of Classes used in Assignment 06 code

Within the Classes used are the functions. For this assignment the functions were pre-specified on what was necessary, see the following list:

- output_error_messages
- output_menu
- input_menu_choice
- output_student_courses
- input_student_data
- read_data_from_file

We can see that each of these functions performs a designated action. **Figure 2** is an example of the input_menu_choice function and I will briefly discuss the functionality of that portion of the code.

```
@staticmethod
def input_menu_choice():
    """ This function gets a menu choice from the user
    :return: string with the users choice
    """
    choice = "0"
    try:
        choice = input("Enter your menu choice number: ")
        if choice not in ("1", "2", "3", "4"): # Note these are strings
            raise Exception("Only 1, 2, 3, or 4 are valid options")
    except Exception as e:
        IO.output_error_messages(e.__str__()) # Not passing the exception object to avoid the technical message

    return choice
```

Figure 2: Input_menu_choice function

Starting at the top the function is called out to be a “Static method”, in other words it doesn’t affect the class and thereby does not change the class. From there we begin a try-except so that we can handle any errors that come without the code crashing. Then the user is prompted to make a choice of 1, 2, 3 or 4. These correspond to the defined choices of ‘menu’, if the choice is not any of those options then the code will prompt an error message regarding the chosen option.

As you can see the function is very basic, now that it is defined instead of having to write all the code seen, we can simply call the function and have it process what we call inside of it, for example `x=input_menu_choice(4)`. In this case i am setting the value of the choice to be `x=4`.

```

# Present and Process the data
while (True):
    # Present the menu of choices
    IO.output_menu(menu=MENU)
    # Prompt user to make a choice
    menu_choice = IO.input_menu_choice()
    # Input user data
    if menu_choice == "1": # This will not work if it is an integer!
        students = IO.input_student_data(student_data=students)
        continue
    # Present the current data
    elif menu_choice == "2":
        IO.output_student_courses(students)
        continue
    # Save the data to a file
    elif menu_choice == "3":
        FileProcessor.write_data_to_file(file_name=FILE_NAME, student_data=students)

```

Figure 3: Example of code usage for input_menu_choice function

As you can see in **Figure 3** the implementation of the input_menu_choice function is classed as a IO and is being defined as a variable 'menu_choice', the code then splits the choice by 1-4 and calls different functions based on the value assigned. In this case it is inputting student data, showing student data, saving student data, and closing the program.

Global Vs. Local Variables

Global and local variables are one and the same, except that Global variables can be called throughout the entire code and Local variables are specific to sections of the code. In particular local variables are used 'temporarily' by a function. That is to say that local variables do not get stored permanently (GeeksforGeeks, 2024b). Whereas, a global variable once used is stored and can then be continually used throughout the program. For this assignment the global variables were students and menu_choice. We can see the use of students in the body of the code as shown below:

```

# When the program starts, read the file data into a list of lists (table)
# Extract data from file
students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)

```

Figure 4: Example of Global variable 'students' being fed into a function.

As you can see the Fileprocessor class is calling the 'read_data_from_file' function and then calling the file and inputting the students values in the 'student_data' table. For user inputs 2 and 3:

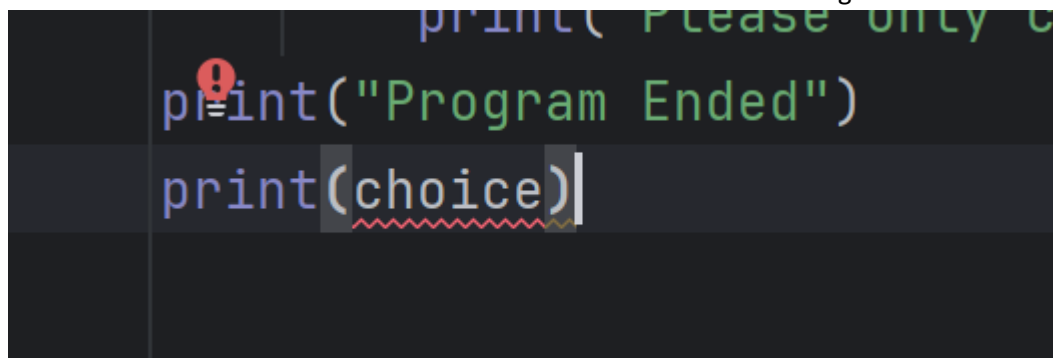
```

elif menu_choice == "2":
    IO.output_student_courses(students)
    continue
# Save the data to a file
elif menu_choice == "3":
    FileProcessor.write_data_to_file(file_name=FILE_NAME, student_data=students)
    continue
# Stop the loop

```

Figure 5: Image of inputs 2 & 3.

The global variable 'students' is being used 2: output the student course info and 3: write the data to the file. Notably inside the functions being called the value 'students' is not being called so as to prevent confusion of whether we are calling the global or local variable. Hence, we use the 'student_data' variable as our local variable. This is important to note because when we use 'student_data' in each function we are forcing the data into the function and then processing it via our code. The values called within the function will not get stored and therefore when running the code, the user would not be able to call them. For example in the 'input_menu_choice' function we assign the local variable 'choice'. Once the code has been run if I called for 'choice' we would see the following:



```

print('Please only C')
print('Program Ended')
print(choice)

```

```

Enter your menu choice number: 4
Program Ended
Traceback (most recent call last):
  File "C:\Users\Lewan\Documents\PythonCourse\Module06\Assignment\Assignment06.py", line 200, in <module>
    print(choice)
    ^^^^^^^
NameError: name 'choice' is not defined

```

Figure 6: Showcase of local variables not being able to be called outside of the function.

We can see that the name of 'choice' is not valid because by the time we called it out, it was essentially forgotten by the code, i.e. it was not stored as a variable (GeekforGeeks, 2024b).

Using Parameters and Arguments

Parameters and Arguments are both concepts used to pass information to functions. Parameters are simply placeholders for a value within a function. Whereas, Arguments are values that are given to the function. Notably this was discussed in the prior section in detail. So how do we use Parameters?

Typically they are the values listed within parentheses when a function is defined. In the previous example of **Figures 5 and 6** 'choice' was the parameter that was used within the 'input_choice' function. Arguments then are used for the action of accomplishing the function, thereby taking the actual values and being run through the function where the parameters were defined. "Arguments can be more than just values they can be constants, variables, or expressions" (W3schools.com).

```
1 usage
def read_data_from_file(file_name: str, student_data: list):
    """ This function reads from a JSON file and loads the data into a dictionary

# Extract data from file
students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)

Assignment.Assignment06
```

Figure 7: Example of Parameters vs Arguments

In **Figure 7** the function being defined is calling 'file_name' and 'student_data', these are the Parameters of the function, or what the function is using to evaluate. Then in the second image, the FileProcessor is then taking the function and calling the Arguments 'file_name=FILE_NAME' and 'student_data=students'. For this case, we are calling the FILE_NAME and having the students data being read from the data in the file.

Separation of Concerns

Separation of Concerns is a methodology of creating an easier to read and understand code by taking certain sections of the code and keeping them self-contained and separating functionality. In other words we take our 'complex' code and break it down into chunks. In this assignment we accomplished this by creating our Classes and Functions. In doing this we used the Decorators for each function. The benefits of using these are clear identifiable points where we can distinguish the beginning and end of the function. Python also brings a really neat feature of allowing us to minimize the functions for better readability, shown below in **Figure 8**:

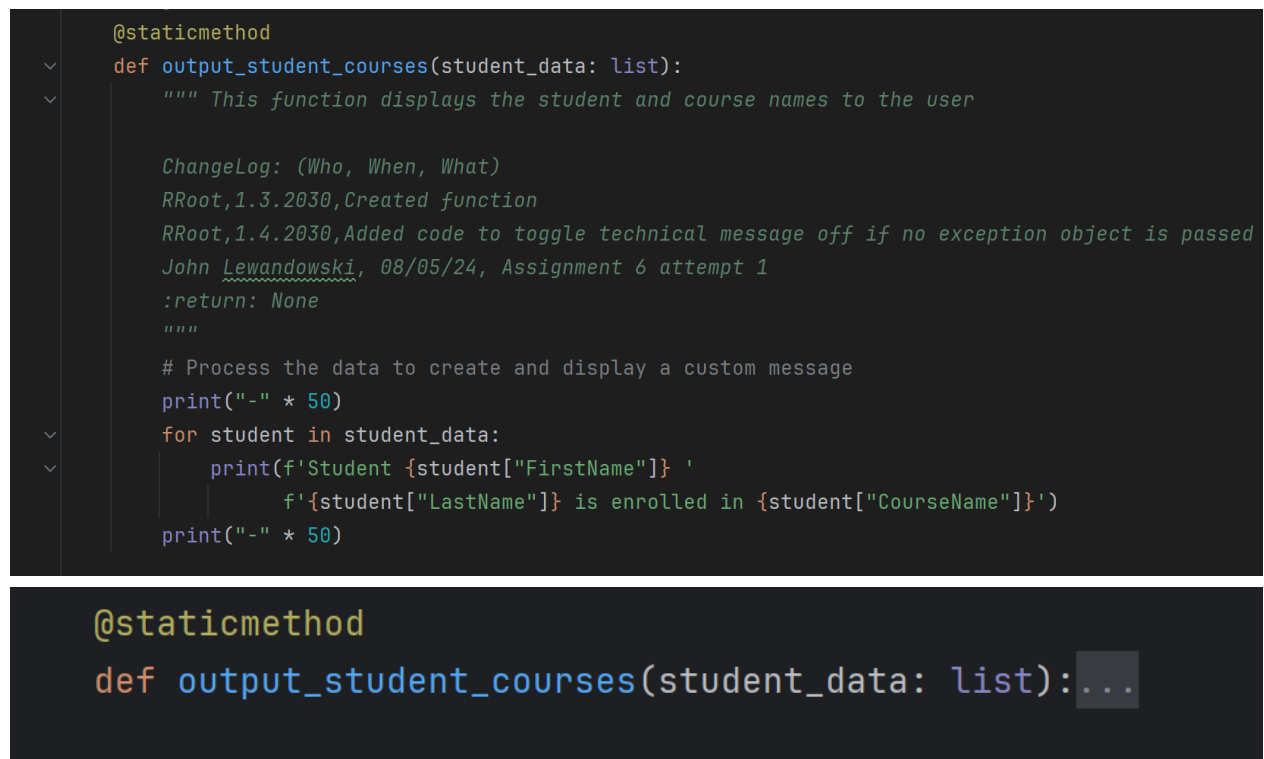


Figure 8: Image showcasing the arrows Python allows for minimizing portions of code (before and after).

Clicking the carrot directly to the left of the definition of the function, we see the function is then minimized, this greatly makes it easier to separate the function from other parts of the code thereby utilizing SOC. Another aspect of utilizing the SOC method is by keeping everything pertaining to the functions within the functions themselves. This then prevents us from having to run around the code to find things. For example, now that 'output_student_courses' function is defined if i had another program where i needed to read a list of students or another variable I could quickly copy and paste the code and easily modify the variables used to then adjust the code as needed. This greatly increases the modularity of the code and allows a more simple way to add to a code that would otherwise be very complicated.

Summary

Following the Module 06 videos, reading/watching the additional videos and modules in tasks 2 and 3 of the assignment file and referencing helpful websites on Python usage, I was able to successfully complete the assignment. Learning along the way how to utilize classes and function, and particularly avoiding the mistakes of indentation and capitalization when calling classes and functions. Also, I learned that Global variables are accessible throughout the code whereas local variables are only temporarily accessible within a function. When to use Parameters and when to use Arguments and being able to differentiate between the two. Finally, I learned how to better utilize the Separation of Concerns methodology allowing me to not only code more quickly but avoid difficulties that would otherwise be very hard to catch without the compartmentalization of the code. I was able to take code that otherwise would have to be written slowly and tested for function before being used, quickly adapting its functionality without having to start from the very beginning. These skills greatly increase my ability to develop as a coder and

progress through my journey as more advanced techniques are brought to my attention via the next assignment.

References

- Seb, P. (2024, February 23). Class vs Function Python: Master Key Differences & Uses. Codingdeeply. <https://www.codingdeeply.com/class-vs-function-python/>
- GeeksforGeeks. (2024b, July 25). Global and local variables in Python. GeeksforGeeks. <https://www.geeksforgeeks.org/global-local-variables-python/>
- W3Schools.com. (n.d.-b). https://www.w3schools.com/python/gloss_python_function_arguments.asp
- MyExamCloud. (2024, May 29). The Power of Decorators: Separation of Concerns (SOC) in Python Code. DEV Community. <https://dev.to/myexamcloud/the-power-of-decorators-separation-of-concerns-soc-in-python-code-5ci9>
-