Kyle Banfill

2/24/2022

Foundations of Programming: Python

Assignment 06

https://blue-blazes.github.io/IntroToProg-Python-Mod06/

Creating a ToDo List with Functions

Introduction

In this module, we learned how to use functions to clean up our code. We took the Concern of Separations concept from last week and put it into practice. Additionally, we learned how to better fix bugs in our code and present our code via GitHub.

Questions

1. What is a function?

A function is an easy to way to group statements. Functions must be defined before they are able to be called.

2. What are parameters?

Parameters are the values that you pass through a function for processing.

3. What are arguments?

Arguments are the name for the values that are passed into parameters.

4. What is the difference between parameters and arguments?

Parameter is a description of what the value is performing, and argument is the actual value passed through the parameter.

5. What are return values?

Return values are the capture of the results of a function.

6. What is the difference between a global and a local variable?

Global variables are variables declared outside a function. They can be recognized throughout the body of the code. Local variables are declared within the function and are only recognized within the function.

7. How do you use functions to organize your code?

Functions allow you to clean up code by separating out the processing sections and then just referencing functions later in presentation segments of code.

8. What is the difference between a function and a class?

A Class doesn't perform any processing on its own, but it groups functions with their variables and constants.

9. How do functions help you program using the "Separations of Concerns" pattern?

Functions allow you to separate your code between processing code and presentation code.

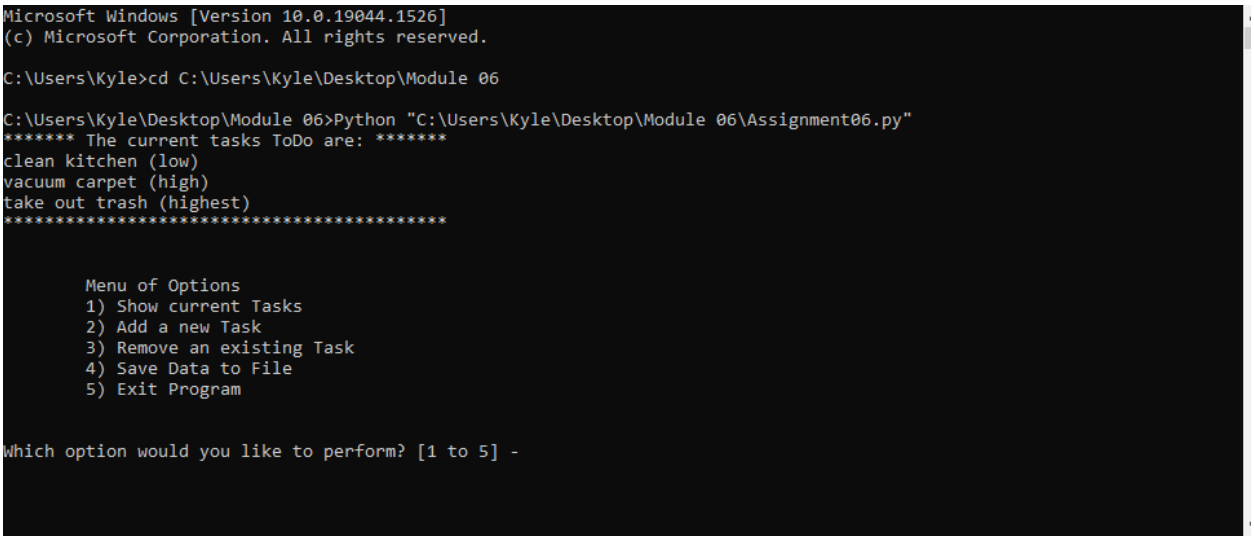
10. How are the debugging tools use in PyCharm?

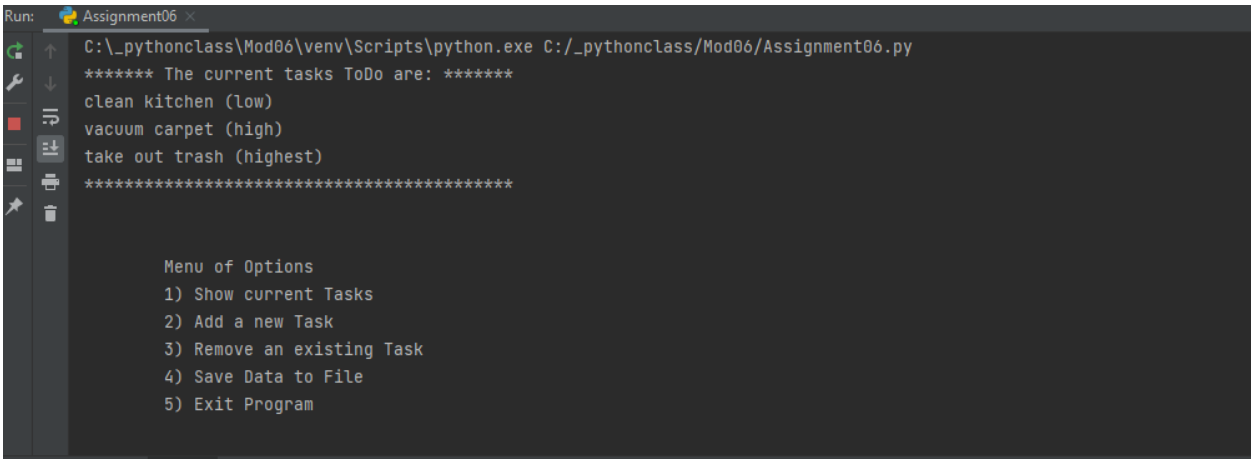
Debugging tools are used to find and fix bugs much faster. You can also set up breakpoints where the code stops and allows you to examine data related to your code while its running like variables or functions.

11. What is a GitHub webpage?

A GitHub webpage is a simple website that enhances how GitHub repositories are displayed.

Our ToDo List with Functions





Our ToDo List looks a lot like it did last week, but we are building off pre-existing script and using functions.

```
12  # Data ---------------------------------------------------------------------- #
13  # Declare variables and constants
14  file_name_str = "ToDoFile.txt"  # The name of the data file
15  file_obj = None  # An object that represents a file
16  row_dic = {}  # A row of data separated into elements of a dictionary {Task,Priority}
17  table_lst = []  # A list that acts as a 'table' of rows
18  choice_str = ""  # Captures the user option selection
19
20
```

Our first step is to declare our variables and constants.

```
21  # Processing  ---------------------------------------------------------- #
22  class Processor:
23      """  Performs Processing tasks """
```

Next, in the processing section, a class called “Processor” is created for our processing functions. If we need to call our functions later, we can refer to the Processor class when calling the function to make sure our variables are defined correctly.

```python
    @staticmethod
    def read_data_from_file(file_name, list_of_rows):
        """ Reads data from a file into a list of dictionary rows

        :param file_name: (string) with name of file:
        :param list_of_rows: (list) you want filled with file data:
        :return: (list) of dictionary rows
        """
        list_of_rows.clear()  # clear current data
        file = open(file_name, "r")
        for line in file:
            task, priority = line.split(",")
            row = {"Task": task.strip(), "Priority": priority.strip()}
            list_of_rows.append(row)
        file.close()
        return list_of_rows

```

For our first function we're defining, "read_data_from_file", we're creating the code for reading from a pre-existing file, defined above as ToDoFile.txt. We also set our parameters and our return value. A lot of the code in the processing section looks exactly like the code used last week in our ToDoList, but modified slightly to conform to the pre-written variables and constants.

```python

    @staticmethod
    def add_data_to_list(task, priority, list_of_rows):
        """ Adds data to a list of dictionary rows

        :param task: (string) with name of task:
        :param priority: (string) with name of priority:
        :param list_of_rows: (list) you want filled with file data:
        :return: (list) of dictionary rows
        """
        row = {"Task": str(task).strip(), "Priority": str(priority).strip()}
        list_of_rows.append(row)
        return list_of_rows

```

Our next function is basically identical to our code from last week, but we've made it a defined function with a return value instead of including it in the main body of the script.

```python
    @staticmethod
    def remove_data_from_list(task, list_of_rows):
        """ Removes data from a list of dictionary rows

        :param task: (string) with name of task:
        :param list_of_rows: (list) you want filled with file data:
        :return: (list) of dictionary rows
        """
        for row in list_of_rows:
            if row["Task"].lower() == task.lower():
                list_of_rows.remove(row)
        return list_of_rows

```

Same for this section as well.

```python
    @staticmethod
    def write_data_to_file(file_name, list_of_rows):
        """ Writes data from a list of dictionary rows to a File

        :param file_name: (string) with name of file:
        :param list_of_rows: (list) you want filled with file data:
        :return: (list) of dictionary rows
        """
        file_obj = open(file_name, "w")
        for row in list_of_rows:
            file_obj.write(row["Task"] + "," + row["Priority"] + "\n")
        file_obj.close()
        return list_of_rows

```

And this as well.

```python
# Presentation (Input/Output)  -------------------------------------------- #


class IO:
    """ Performs Input and Output tasks """

```

Our next section is our presentation section. All of the instances of us using the print() function or input() function are going to be included here.

```python
    @staticmethod
    def output_menu_tasks():
        """  Display a menu of choices to the user

        :return: nothing
        """
        print('''
        Menu of Options
        1) Show current Tasks
        2) Add a new Task
        3) Remove an existing Task
        4) Save Data to File
        5) Exit Program
        ''')
        print()  # Add an extra line for looks
```

Here's our menu that looks just like it did last week. The pre-written menu didn't include an option to show current tasks, so we've included one as a menu option.

```python
    @staticmethod
    def input_menu_choice():
        """ Gets the menu choice from a user

        :return: string
        """
        choice = str(input("Which option would you like to perform? [1 to 5] - ")).strip()
        print()  # Add an extra line for looks
        return choice

```

Here's our function for gathering user input for menu choices. The strip() function has been included so that user input is normalized.

```python
    @staticmethod
    def output_current_tasks_in_list(list_of_rows):
        """ Shows the current Tasks in the list of dictionaries rows

        :param list_of_rows: (list) of rows you want to display
        :return: nothing
        """
        print("******* The current tasks ToDo are: *******")
        for row in list_of_rows:
            print(row["Task"] + " (" + row["Priority"] + ")")
        print("*********************************************")
        print()  # Add an extra line for looks
```

This code was pre-written, but looks just like our code from last week because it's almost only printing functions.

```python
    @staticmethod
    def input_new_task_and_priority():
        """  Gets task and priority values to be added to the list

        :return: (string, string) with task and priority
        """
        task = str(input("Please enter a task: ")).strip()
        priority = str(input("Please enter its priority: ")).strip()
        return task, priority

```

Next, we define an input function for adding new tasks.

```python
    @staticmethod
    def input_task_to_remove():
        """  Gets the task name to be removed from the list

        :return: (string) with task
        """
        task = str(input("Which task do you want to remove?: "))
        return task

```

Then we define an input function for removing tasks.

```
# Main Body of Script  ------------------------------------------------------ #

# Step 1 - When the program starts, Load data from ToDoFile.txt.
Processor.read_data_from_file(file_name=file_name_str, list_of_rows=table_lst)  # read file data

```

Our last section is the main body of the script where we'll write the script as it will actually process. We'll take all our defined functions and variables/constants and create the actual body of code. Thanks to our work in defining functions and grouping them with classes, our code will be super condense compared to last week.

```
# Step 2 - Display a menu of choices to the user
while (True):
    # Step 3 Show current data
    IO.output_current_tasks_in_list(list_of_rows=table_lst)  # Show current data in the list/table
    IO.output_menu_tasks()  # Shows menu
    choice_str = IO.input_menu_choice()  # Get menu option

```

Our while loop was already created, but notice that we're just calling our previously defined functions using our class pre-fixes in order to ensure everything is defined correctly. We are also sure to use our correct arguments in our functions.

```
    # Step 4 - Process user's menu choice
    if choice_str.strip() == '1':   # Show current task
        IO.output_current_tasks_in_list(list_of_rows=table_lst)
        continue

    elif choice_str.strip() == '2':  # Add a new Task
        task, priority = IO.input_new_task_and_priority()
        table_lst = Processor.add_data_to_list(task=task, priority=priority, list_of_rows=table_lst)
        continue  # to show the menu

    elif choice_str == '3':  # Remove an existing Task
        task = IO.input_task_to_remove()
        table_lst = Processor.remove_data_from_list(task=task, list_of_rows=table_lst)
        continue  # to show the menu

    elif choice_str == '4':  # Save Data to File
        table_lst = Processor .write_data_to_file(file_name=file_name_str, list_of_rows=table_lst)
        print("Data Saved!")
        continue  # to show the menu

    elif choice_str == '5':  # Exit Program
        print("Goodbye!")
        break  # by exiting loop
```

All of this was created for us already and it just flows because of the time we took to set up our functions in the processing and presentation sections of code. There was an option for displaying current tasks missing, so we created one of those quickly and set that to occur if our menu choice string is set to 1.

Conclusion

In this module, we explored functions and classes and how they can really organize our code. We took the concept of Separation of Concerns and implemented it into our code. Segmenting our code in this way does wonders for code hygiene, as we can easily locate the different parts of our code, but we also learned about debugging functionality to streamline fixing our code further.