Chanel M. Leung

May 24, 2020

Foundations of Programming: Python

Assignment 06

https://cml-python.github.io/ITFDN100-Mod06/

# Functions & Modules

## Introduction

In Module 6, I was able to learn about functions with parameters, return values and classes of functions. Using these concepts, the Assignments goal was to complete the code of a script given using the classes of functions with parameters. I thought that last week's module was difficult using an outlined code, but I could tell through this week's why it is said that sometimes using outlines can be harder as I was going through the process of adding my own code to the partially pre coded script. Overall the assignment would still, like last week, create a to do list that could add, remove, save and read data that was added by a user.

## Functions with Parameters

Working with functions previously, I vaguely remember talking more on parameters and what they were used for, but through this module I became to have a better understanding of what they are, which is basically allowing a coder to pass values into the functions. You would be able to use the values that the parameters held to use within the function, through this you would first have to define the function, then you would put in the parameters you wanted to and manipulate the values within the function. In this modules assignment, the outline code already defined functions and gave parameters as well, however I also was able to learn that after the function, I was able to add docstring to help define what I meant to do with the parameters which I thought was interesting (Figure 1). I

learned this helped tremendously when I was looking at the Main Body of the Script, determining what functions to put in.

```
@staticmethod
def add_data_to_list(task, priority, list_of_rows):
    """Adds User's Task and Priority to Dictionary Row

    :param task: (string) user's task input:
    :param priority: (string) user's priority input:
    :return: list_of_rows: (list) contains file data:
    """
    strTask = input("Task: ")
```

*Figure 1*

**Return Values**

I was not aware what return values were prior to this module, but now I know that it is taking returning values of a function in variables as an expression to use again without having to call the function. I also learned that this doesn't only mean you can return one value but can be multiple when in a collection put usually with commas. In this assignment, I was not able to learn hands on how and when to code them, but it seems pretty straight forward on where to put and utilize them. In the script I saw it was after each defined function, it would return to "list_of_rows" (Figure 2). This would allow to use the rows within the table to retain the information to be manipulated through the different functions after going through that certain one.

```
        loadFile.close()
        print("Data Saved")
        return list_of_rows, 'Success'
```

*Figure 2*

**Classes of Functions**

When learning about classes, I was excited because it organized the functions so it would be easier to call/define them in the script later. I learned that it is able to group functions, variables and constants. In this week's assignment, we were able to use basically 2 different classes, input/output and processing. Within the classes we were able to add to the list, remove from the list and save the data using functions that would make the script

look more organized when written out completely (Figure 3). I really liked how utilizing classes, we were able to further show that there were clearly sections to the code from the process, users input and output then finally displaying the utilization of the processes within the functions that were written above in the code. I think I really enjoyed the simple look at the bottom, as well as the effectiveness of the layout to help find areas that were not coded correctly.

```
while (True):
    # Step 3 Show current data
    IO.print_current_Tasks_in_list(lstTable)  # Show current data in the list/table
    IO.print_menu_Tasks()  # Shows menu
    strChoice = IO.input_menu_choice()  # Get menu option

    # Step 4 - Process user's menu choice
    if strChoice.strip() == '1':  # Add a new Task
        IO.input_new_task_and_priority()
        Processor.add_data_to_list(strTask,strPriority,lstTable)
        IO.input_press_to_continue(strStatus)
        continue  # to show the menu

    elif strChoice == '2':  # Remove an existing Task
        IO.input_task_to_remove()
        Processor.remove_data_from_list(strTask,lstTable)
        IO.input_press_to_continue(strStatus)
        continue  # to show the menu

    elif strChoice == '3':  # Save Data to File
        strChoice = IO.input_yes_no_choice("Save this data to file? (y/n) - ")
        if strChoice.lower() == "y":
            Processor.write_data_to_file(strFileName,lstTable)
            IO.input_press_to_continue(strStatus)
        else:
            IO.input_press_to_continue("Save Cancelled!")
        continue  # to show the menu

    elif strChoice == '4':  # Reload Data from File
        print("Warning: Unsaved Data Will Be Lost!")
        strChoice = IO.input_yes_no_choice("Are you sure you want to reload data from file? (y/n) -  ")
        if strChoice.lower() == 'y':
            IO.print_current_Tasks_in_list(lstTable)
            IO.input_press_to_continue(strStatus)
        else:
            IO.input_press_to_continue("File Reload  Cancelled!")
```

*Figure 3*

**Summary**

Module 6's assignment took a lot more time than last weeks, which I thought was funny because I thought last week was time consuming. But I think with the outline of this script, it really helped me more visualize how this would process the to do list in a more

organized way when it comes to processing data, input/output of data and displaying it to the user. I think that I was able to complete my objective of the assignment which organized and added code to the premade script. I also wanted to add that I thought that the debugging section of the video/notes was helpful because when I tried to play around, I was wondering how I would be able to just try part of the script without having to comment out everything I did not want to test in the first place.