Vanessa Rubin

21 November 2022

Foundations of Programming: Python

Assignment 06

https://github.com/VRubin123/IntrotoProg-Python-Mod6

Using Functions

Introduction

Functions, another fundamental concept in programming, allow the programmer to run a block at several different locations in the script without having to repeat the initial codes. One may call the code via the function identifier and integrate it with another script.

To Do list using Functions

This week's task is to repeat the function of last week's "To Do List" assignment but using functions. The provided starter script is organized into tasked sections of the program, including using "class" to denote the different functional blocks, Figure 1. What is also different, is the use of "return" within a function. This allows objects within the different "class" elements to be returned to the caller of the function embedded within the "Main Body of Script" to run the program.

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


# Processing  --------------------------------------------------------------- #
class Processor:...


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


class IO:...

# 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

# Step 2 - Display a menu of choices to the user
while (True):...
```

Figure 1, Display of program organization

The "Main Body of Script" portion of the code has been completed, and we were tasked to complete the "processing" and "input/output" code to make the program functional. I tackled this assignment by first

starting with the addition of a new task to the list “list_of_rows”. This was performed defining the variable “row” as a dictionary, and then using it to append the input data from the “task” and “priority” variables, Figure 2. Because the processing code needed user input, the accompanying input/output functions needed to be completed, Figure 3.

Figure 2, append statement

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

    :return: (string, string) with task and priority
    """
    pass  # TODO: Add Code Here!
    task = input("Please input task:")
    priority = input("Please assign priority (low, medium, high):")
    return task, priority
```

Figure 3, input objects to add to list.

The next task was to remove data from the list. This was performed by using a “for” statement with a nested “if” statement to identify the task to be removed, Figure 4. To print “task not found”, I used a Boolean switch where a variable is “False” at the beginning of the function, and if an item is removed successfully, the variable is switched to “True”, and prints out Item removed. However, if the switch is not activated, the function prints out “Task not found”. This resolved the issue I ran into with the last statement with having the phrase repeat over each row where the task was not found, Figure 4

```
@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
    """
    # TODO: Add Code Here!
    row = {"Task": str(task).strip()}
    na_task = False
    for row in list_of_rows:
        if row["Task"] == task:
            list_of_rows.remove(row)
            na_task = True
            print("Task removed.")
    if na_task == False:
        print("Task not found.")
    return list_of_rows
```

Figure 4, Remove item code and “Task not found.” printed statement.

To complete the “remove” function, I added an input statement in the “Class IO” that will be called in the “Main Body of Script”, Figure 5.

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

    :return: (string) with task
    """
    pass  # TODO: Add Code Here!
    task = input("Please input task to be removed:")
    return task
```

Figure 5, Defining “task” with a user input statement.

To finish the program, we wanted to save the to do list as a text file and inform the user that this function was performed, Figure 6. The function opens the existing file and indicates that it will be overwritten. Since the earlier functions store the data as a list of dictionary rows, the file is written using dictionary keys so that the format is more palatable to the user, Figure 6.

```
@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
    """
    # TODO: Add Code Here!
    file = open(file_name, "w")
    for row in list_of_rows:
        file.write(str(row["Task"] + ", " + row["Priority"] + "\n"))
    file.close()
    return list_of_rows
```

Figure 6, Saving list to text file.

Conclusion

Functions allow the programmer to create more organized and succinct code. Instead of having to repeat full blocks within a program, the programmer may call that block in subsequent code. Additionally, categorizing the code into functional sections prevents chaos during the creation of a program.