Ryan Lupinski

02.18.2022

Foundations of Programming: Python

Assignment 06

To Do List Improved

1 INTRODUCTION 1

2 FUNCTIONS 1

3 CLASSES 2

3.1 Organization 2

3.2 Separation of concerns 3

4 TO DO LIST PROGRAM IMPROVED 4

4.1 Loading existing tasks into Memory 4

4.2 Print current tasks and options menu 5

4.3 Add or remove a task from the task list in memory 5

4.4 Save or reload data to/from the file 6

5 SUMMARY 7

1 Introduction

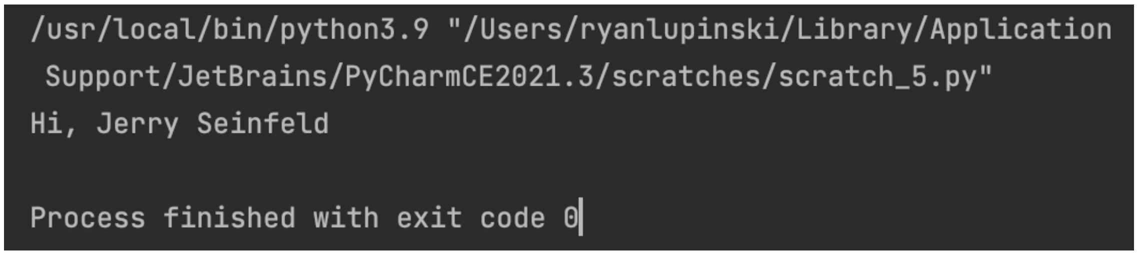
This write up will discuss how functions are used in Python, how to pass arguments to functions, the difference between arguments and parameters, and how organizing scripts by placing functions in large sections called classes will improve the previous assignment's To Do List script.

2 Functions

By definition, "A function is a block of code which only runs when it is called" (w3schools, https://www.w3schools.com/python/python_functions.asp, Feb,2022)(External Site). Functions are useful for allowing a script to perform a specific task the same way each time it is run. To run a function, it must be ‘called’. For a function to be called, it must be defined previously in the script. Listing 1 shows a simple function being defined and called with an argument.

```
def test_function(name):
    print("Hi, " + name)
    return
test_function("Jerry Seinfeld")
```

Listing 1 test_function defined and called with an argument



```
/usr/local/bin/python3.9 "/Users/ryanlupinski/Library/Application
 Support/JetBrains/PyCharmCE2021.3/scratches/scratch_5.py"
Hi, Jerry Seinfeld

Process finished with exit code 0
```

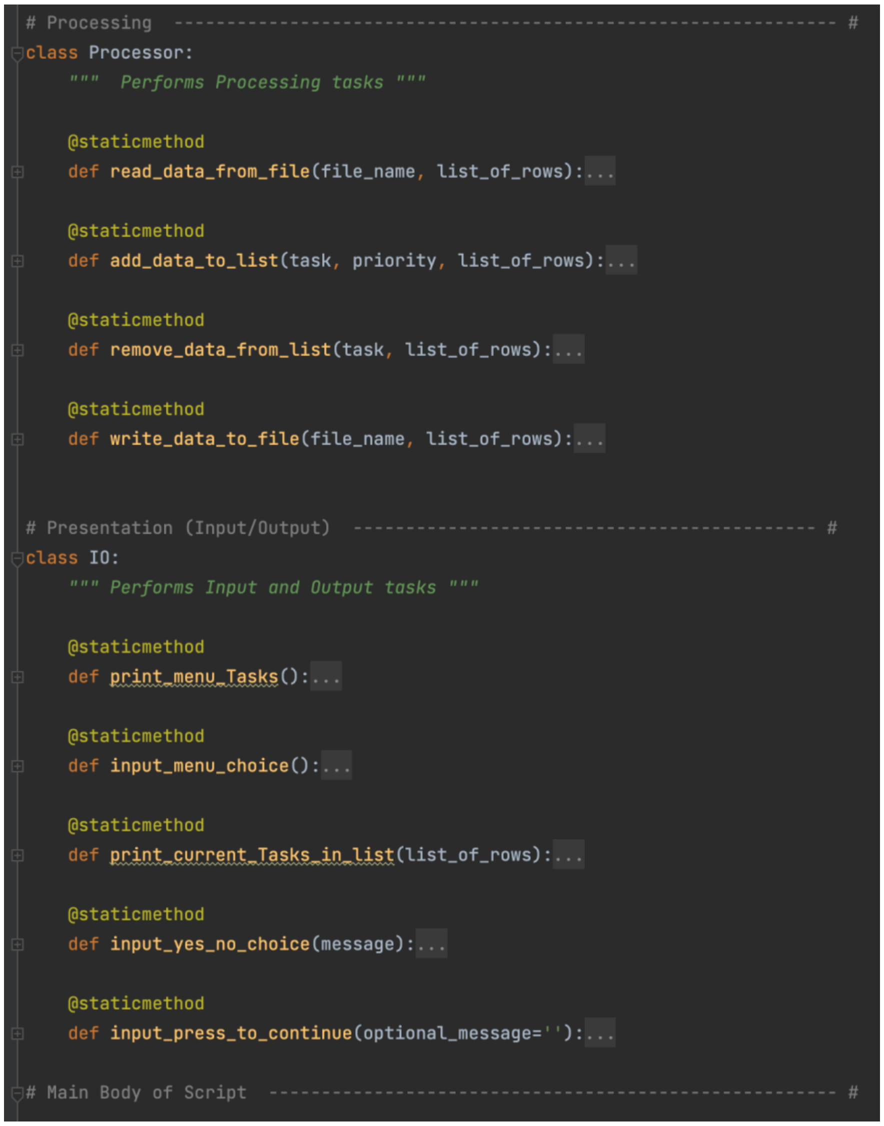
Figure 1 Output of script calling test_function

Figure 1 shows the output of the script. Functions are defined with the ‘def’ prefix before the function name. Arguments are any information that is passed into the function. Test_function is passed the argument “Jerry Seinfeld”. The parameter ‘name’ inside the brackets is the variable that operates in the body of the function. Arguments are passed to functions and the parameters may take on the value of the argument and used inside the function. Functions may return something upon completion, however test_function returns nothing. The To Do List program often takes in a list as an argument to a function, modifies the list, and then returns the modified list.

3 Classes

3.1 Organization

Organizing functions into meaningful sections of the code is very helpful. “A class is a way to take a grouping of functions and data and place them inside a container so you can access them with the . (dot) operator.” (Shaw, Zed A., Learn Python 3 The Hard way, Nov 2018). The To Do List has a large section of functions that perform processing tasks, and another section that performs input/output (IO) tasks, so it makes sense to create two classes called Processor and IO. Calling an IO function would look something like this IO.function(). See Figure 2.



```
# Processing  ------------------------------------------------------------- #
class Processor:
    """  Performs Processing tasks """

    @staticmethod
    def read_data_from_file(file_name, list_of_rows):...

    @staticmethod
    def add_data_to_list(task, priority, list_of_rows):...

    @staticmethod
    def remove_data_from_list(task, list_of_rows):...

    @staticmethod
    def write_data_to_file(file_name, list_of_rows):...


# Presentation (Input/Output)  ------------------------------------------- #
class IO:
    """ Performs Input and Output tasks """

    @staticmethod
    def print_menu_Tasks():...

    @staticmethod
    def input_menu_choice():...

    @staticmethod
    def print_current_Tasks_in_list(list_of_rows):...

    @staticmethod
    def input_yes_no_choice(message):...

    @staticmethod
    def input_press_to_continue(optional_message=''):...

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

Figure 2 Functions grouped inside classes

3.2 Separation of concerns

Separation of concerns is a technique that programmers use to organize their code in a meaningful, readable, and useful way. An example of this is creating a script with a section at the top for global variables, variable creation/initialization, and constants, then a section for processing which includes functions grouped inside classes, then a section for IO, then the main body of the program which prompts the user, calls functions, and operates until the code is complete. Organizing a script in this fashion is best practice, as it makes what is happening easier for other programmers to understand, makes faulty code easier to troubleshoot, and makes it easier to build the script up when beginning a project.

4 To Do List Program Improved

The previous version of the To Do List program used a while loop to jump to different sections of code to perform certain tasks but did not utilize any functions. The new version delegates these repetitive lines of code to functions and groups functions into classes. The program begins by declaring and initializing variables, then defines all the functions within the two classes, Processor and IO, loads data from a file, then the main body of the program utilizes a While loop to prompt a user for an option involving adding or removing tasks, saving to the file or ending the program. The full code can be viewed here (https://github.com/ryanlupinski/IntroToProg-Python-Mod06)(external site).

4.1 Loading existing tasks into Memory

The first step in the main body of the program calls function in the Processor class using Processor.read_data_from_file(strFileName, lstTable) and passes two arguments, a string of the File Name, and a list table. See Listing 2.

```
# Step 1 - When the program starts, Load data from ToDoList.txt.
# Processing  -------------------------------------------------------------- #
class Processor:
    """  Performs Processing tasks """

    @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
        try:  # check if ToDoList.txt already exists
            objFile = open(file_name, "r")  # open ToDoList.txt in read mode
            for line in objFile:  # for each line in the file
                task, priority = line.split(",")  # split elements in the
file line by a comma, save to variables
                row = {"Task": task.strip(), "Priority": priority.strip()}  #
create dicRow from previous variables
                list_of_rows.append(row)  # append each row to the lstTable
            objFile.close()  # close the file
        except:  # if ToDoList.txt doesn't exist create a new txt file and
populate with a header
            objFile = open(file_name, 'w')  # open the file
            objFile.write("Task" + ',' + "Priority" + '\n')  # write
task,priority header to file
            objFile.close()  # close the file
            print("ToDoList.txt not found. Creating a new ToDoList.txt file")
# print to console
        return list_of_rows, 'Success'  # return the lstTable
```

Listing 2 Processor.read_data_from_file(strFileName, lstTable) section

Arguments strFileName and lstTable are passed to the functions and assigned to the parameters file_name and list_of_rows, respectively. A header section in triple quotes gives a description of each parameter. The same try-except method from the previous assignment is utilized to see if a file named ToDoList.txt already exists. See the comments in Listing 2 that describe function read_data_from_file and what it does.

4.2 Print current tasks and options menu

Now that the data from the file has been loaded into memory with the return list_of_rows line from the code in Listing 2, the tasks can be printed to the screen and along with a menu using the following code.

```
# Step 2 - Display a menu of choices to the user
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
```

Listing 3 While loop, print tasks, print menu, and prompt user for option code

In the While loop, the task list table is passed to function print_current_Tasks_in_list which formats the tasks in a readable fashion. Next the print_menu_Tasks function displays a menu of options and returns nothing. The next line of code calls the function input_menu_choice and returns a string value choice which is saved to the variable strChoice. Now the if/elif/else statements will execute and perform the requested function.

4.3 Add or remove a task from the task list in memory

If the user types ‘1’ they are prompted to enter a task and a priority. If they enter ‘2’, they are prompted to type in the name of a task to be removed from the list which is stored in memory as variables. See listing 4.

```
if strChoice.strip() == '1':  # Add a new Task
    strTask = str(input("Add task: "))  # prompt user to add a task
    strPriority = str(input("Add priority[1(high) - 5(low): "))  # prompt
user to define priority
    Processor.add_data_to_list(strTask, strPriority, lstTable)  # pass task,
priority, and task table args to add data function
    IO.input_press_to_continue(strStatus)
    continue  # to show the menu

elif strChoice == '2':  # Remove an existing Task
    strTask = str(input("Remove a task: "))  # prompt user to add a task
    Processor.remove_data_from_list(strTask, lstTable) # pass task and task
table as args to remove data function
    IO.input_press_to_continue(strStatus)
    continue  # to show the menu
```

Listing 4 Add and remove option function call section

Listing 4 shows arguments being passed to various functions depending on the option being chosen. Listing 5 shows the function definition in the Processor class section.

```
@staticmethod
def add_data_to_list(task, priority, list_of_rows):
    dicRow = {'Task': task, 'Priority': priority}  # create a new dicRow with
new task
    list_of_rows.append(dicRow)  # append the table w/ new dicRow
    return list_of_rows, 'Success'  # return the updated list

@staticmethod
def remove_data_from_list(task, list_of_rows):
    for row in range(len(list_of_rows)):  # look in each row of the lstTable
        if list_of_rows[row]['Task'] == task:  # check if task key equals the
task to be removed
            del list_of_rows[row]  # delete the row
            break  # break the if loop
    return list_of_rows, 'Success'
```

Listing 5 Add and remove function definition

Adding a task involves passing the task and priority to the function as arguments, creating a new dictionary row, and appending the list table of to do task. Finally, the updated task list is returned. Removing a task applies the same idea. The function is called, and the task and task list are passed as arguments. Each row in the list is looped through using the For function, and if the parameter variable task matches the 'Task' key in the dictionary row, the row is removed.

4.4 Save or reload data to/from the file

```
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) # pass file name
and task table to write to file function
        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':
        Processor.read_data_from_file(strFileName, lstTable)  # calls read
file data function again
        IO.input_press_to_continue(strStatus)
    else:
        IO.input_press_to_continue("File Reload Cancelled!")
    continue  # to show the menu
```

Listing 5 Save data to file and load data from file

Listing 5 shows what functions are called and the arguments that are passed. It is important to note that each of these options gives the user the ability to abort saving or reloading data. For instance, if the user forgot to add a task, they can skip saving and return to the menu. If the user forgot to save all the updated tasks in the task list in memory, they may not want to reload the tasks from the file, which would overwrite the current task list table in memory. Each function passes the file name and list table to the function as arguments. Option 4 has been covered in section 4.1 and will not be shown in this section.

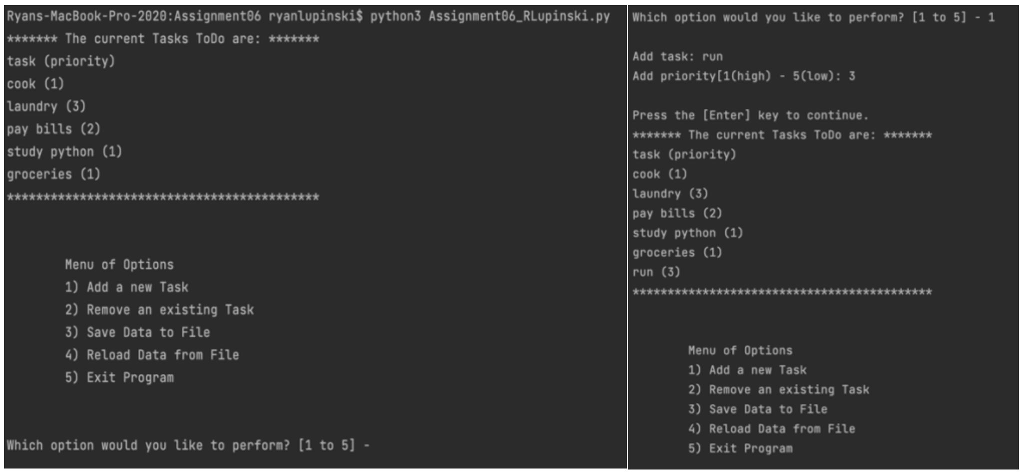
```
@staticmethod
def write_data_to_file(file_name, list_of_rows):
    objFile = open(file_name, 'w')  # open connection to file in write mode
    for row in list_of_rows:    # for each row lstTable
        strTask = row['Task'] # pull task value from lstTable
        strPriority = row['Priority']   # pull priority value from lstTable
        objFile.write(str(strTask) + "," + str(strPriority) + "\n")  # write
task/priority values to file
    objFile.close()  # close the file
    return list_of_rows, 'Success'
```

Listing 6 Save data to file function

When the user is satisfied with the added or removed tasks in the task list in memory, they can save the updated task list to the file. A connection to the file is opened and for each row of the task list, a task and a priority are written to the objFile. The connection is closed and the list of rows is once again returned. Option 5 prints “Goodbye!”, breaks the While loop and the program ends.

5 Summary

Functions are important in programming to keep code consistent and organized. When there are multiple functions that have different general purposes, they can be grouped into Classes. The improved To Do List program utilizes a Processor and IO Class to group functions that either process information or request input or print an output. Separating different sections of code to specific areas is part of the separation of concerns style of programming. Separating code into sections makes it easier to read, helps others who did not write the program to understand what is happening, and makes troubleshooting and organization simpler.



```
Ryans-MacBook-Pro-2020:Assignment06 ryanlupinski$ python3 Assignment06_RLupinski.py
******* The current Tasks ToDo are: *******
task (priority)
cook (1)
laundry (3)
pay bills (2)
study python (1)
groceries (1)
*******************************************



        Menu of Options
        1) Add a new Task
        2) Remove an existing Task
        3) Save Data to File
        4) Reload Data from File
        5) Exit Program



Which option would you like to perform? [1 to 5] -
```

```
Which option would you like to perform? [1 to 5] - 1

Add task: run
Add priority[1(high) - 5(low): 3

Press the [Enter] key to continue.
******* The current Tasks ToDo are: *******
task (priority)
cook (1)
laundry (3)
pay bills (2)
study python (1)
groceries (1)
run (3)
*******************************************



        Menu of Options
        1) Add a new Task
        2) Remove an existing Task
        3) Save Data to File
        4) Reload Data from File
        5) Exit Program
```

Figure 3 Assignment06_RLupinski.py running in a terminal session