Erin Leggett
November 23, 2021
IT FDN 110A
Module 06 - Assignment 06

DEFINING AND CALLING FUNCTIONS IN PYTHON

OBJECTIVE

Describe the steps taken to create the Python script attached to this assignment.

INTRODUCTION

This week our course content primarily focused on creating more complex scripts through defining and calling user-defined functions to process and write data back to a .txt file. Though the objective of this week's program is the same as last week's (i.e., creating and administering a simple To Do list file), employing functions not only required that we separate our code into smaller, more serviceable parts, but also required careful attention to arguments as well as local and global data types.

STEP 1: PROCESS DATA

To begin, it was necessary to give the program some basic parameters to work with. There were only three variables that required some sort of activation prior to the program running (as the rest were defined and populated within): `file_name_str`, `table_lst`, and `choice_str`. I also created a simple variable file_create to open (and, if nonexistent, create) the program's companion .txt file. This was accomplished by inserting the following line of code early in the script file:

```
file_create open(file_name_str, "a")
```

From there, the script moves on to define the four functions it will use to *process* data: read, add, remove, and write. Though read, remove, and write were simple enough, I added some basic limitations to the add function preventing the user from inputting an invalid task priority. This was accomplished by inserting a short conditional loop into the function confirming that the inputted priority matched the options provided (low, medium, or high). I made a few attempts to condense the more repetitive elements, but ultimately the following proved to be most consistently accurate:

```
if priority.lower() == "high":
    list_of_rows.append(row)
    print("Task successfully added to list!")
    return task, priority
elif priority.lower() == "medium":
    list_of_rows.append(row)
    print("Task successfully added to list!")
    return task, priority
elif priority.lower() == "low":
    list_of_rows.append(row)
    print("Task successfully added to list!")
    return task, priority
else:
    input("Please choose a priority between low, medium, and high!")
```

STEP 2: COLLECT INPUTS & DEFINE OUTPUTS

Next, the IO - or Input/Output - class was created to display program outputs to (in this case, the menu and to do list itself) as well as collect necessary inputs from (menu selection, tasks to be added/removed) the user. Compared to the processing functions, these were much simpler in syntax and purpose: of the five IO functions, only one passes through an argument and two of five do not indicate a specific `return` (i.e., output). As these functions are primarily tasked with collecting and displaying data, they also did not necessitate the use of conditionals or loops; instead, they largely employed `input` and `print` to engage with the user and collect inputs.

STEP 3: CALL FUNCTIONS

The final step in creating this week's script file was calling the Processor and IO functions created earlier in the script when needed - which is to say, when executing the task at hand. To do so, a more robust conditional loop similar to that in last week's assignment was deployed; however, as much of the "work" the program is doing had been defined earlier in the script, this week's loop was not populated with further conditionals, but rather functions and arguments being called. As evidenced below, many of these conditional statements consisted solely of functions:

```
if choice_str.strip() == "1":  # Choice 1: Add new task to list.
    (task, priority) = IO.input_new_task_and_priority()
    Processor.add_data_to_list(task, priority, table_lst)
    continue  # Returns user to main menu.
elif choice_str == "2":  # Choice 2: Remove an existing task from list.
    remove_task = IO.input_task_to_remove()
    Processor.remove_data_from_list(remove_task, table_lst)
    continue  # Returns user to main menu.
```

In the case of saving the user's to do list data back to the .txt file, however, I decided to add a warning message alerting the user that the file was in "write" mode, which would overwrite their current list data:

```
elif choice_str == "3":  # Choice 3: Save current To Do list data to
file.
    save_choice = input("Save current list to file? This can't be
undone! (y/n): ")  # Warns user that list data will be overwritten.
    if save_choice.lower() == "y":  # Saves data to files if user
inputs 'y'.
        Processor.write_data_to_file(file_name_str, table_lst)
        input("Data saved to file. Press Enter to return to program.")
    else:  # Returns user to main menu if 'y' is not inputted.
        input("Data not saved to file. Press Enter to return to
program.")
    continue  # Returns user to main menu.
```

STEP 4: VALIDATE CODE

As with last week, I validated my script file two-fold: once by adding a task and saving it back to ToDoFile.txt in the PyCharm IDE (in the process confirming the file was created in the correct location) and again in Terminal by reading, removing, and re-saving the file, essentially overwriting any addition I had just made.

Here are screenshots of that first sequence in PyCharm:

```
******* Here is your current To Do list: *******
************************************************


        Menu of Options
        1) Add New Task
        2) Remove Existing Task
        3) Save List to File
        4) Exit Program
```

```
Which option would you like to perform? [1 to 4] - 1

Enter a task to add to the list: do laundry
Assign a priority (high, medium, low): high
Task successfully added to list!

******* Here is your current To Do list: *******
do laundry (high)
************************************************
```

```
        4) Exit Program

Which option would you like to perform? [1 to 4] - 3

Save current list to file? This can't be undone! (y/n): y
Data saved to file. Press Enter to return to program.

******* Here is your current To Do list: *******
do laundry (high)
************************************************
```

```
Which option would you like to perform? [1 to 4] - 4

Exiting program. Goodbye!
```

As well as the companion ToDoFile.txt file after my first pass in PyCharm:

From there, I moved into Terminal to validate that my .txt file could be read and interacted with from the command line as well:

```
******* Here is your current To Do list: *******
do laundry (high)
************************************************


        Menu of Options
        1) Add New Task
        2) Remove Existing Task
        3) Save List to File
        4) Exit Program
```

```
Which option would you like to perform? [1 to 4] - 2

Enter a task to remove from your to do list: do laundry
Task successfully removed from list!

******* Here is your current To Do list: *******
************************************************
```

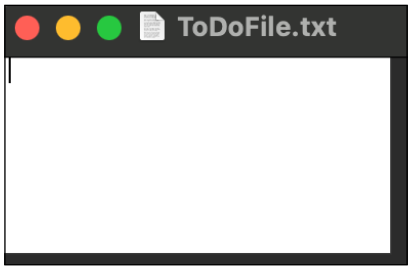
```
Save current list to file? This can't be undone! (y/n): y
Data saved to file. Press Enter to return to program.

******* Here is your current To Do list: *******
************************************************


        Menu of Options
        1) Add New Task
        2) Remove Existing Task
        3) Save List to File
        4) Exit Program

Which option would you like to perform? [1 to 4] - 4

Exiting program. Goodbye!
Pete-Wisdom:Assignment06 erinleggett$
```



LESSONS LEARNED

Though it took some time to get the final script file working (and formatted) as desired, this week I found myself debugging with greater ease and taking advantage of PyCharm's built in debugging tool. That said, when it came to arguments in particular I did run into a few hang-ups: in some cases, I passed an argument when I was ultimately seeking to collect one, which made debugging tricky in some places. Above all else, this assignment highlighted the logical structure of Python - when a function expected (or didn't expect) an argument, the entire program hiccuped in its absence or presence.

Furthermore, I had to make a lot of upfront decisions about how to organize my code. In my first attempt, I left a fair amount of the processing (for example, the Boolean iteration that confirms the task to be removed matches a task on the to do list) down in the main script area. However, in running my program a few times I noticed that I could simplify my main script by adding many of those elements into the functions themselves. In addition to being a hands-on introduction to defining functions in Python, this assignment was a gainful exercise in overall logic and structure.

GITHUB LINK

https://emleggett.github.io/IntroToProg-Python-Mod06/