Andrea Folmer

May 23rd 2023

IT FDN 110 A SP 23: Foundations of Programming: Python

Assignment 06

Functions

Introduction

The goal of this assignment is to learn how to use functions. This assignment started with reviewing someone else's code and then mapping out the relationship between functions, understanding shadow variables, and correctly setting up parameters, arguments, and returns to ensure that my data is formatted correctly throughout the program. This was very important because data in the program is organized as dictionary rows within a list, so you have to know in each step if you're processing data using a key or an index.

Topic 1: Understanding Decorators

My first step in creating this program was to read through the template and try to understand the flow of variables and constants through the functions. At first it was very confusing seeing the function names written as class.function in the code execution block, but now I understand that this is a way to categorize and label your functions to provide quick context to other programmers. The other new item was @staticmethod, which allows you to use just the function name later in a program without the class prefix.

Using decorators within the code enabled a new feature in PyCharm. When you hover your mouse over the code block you get a formatted pop defining parameters and returns based on what you type as a comment.

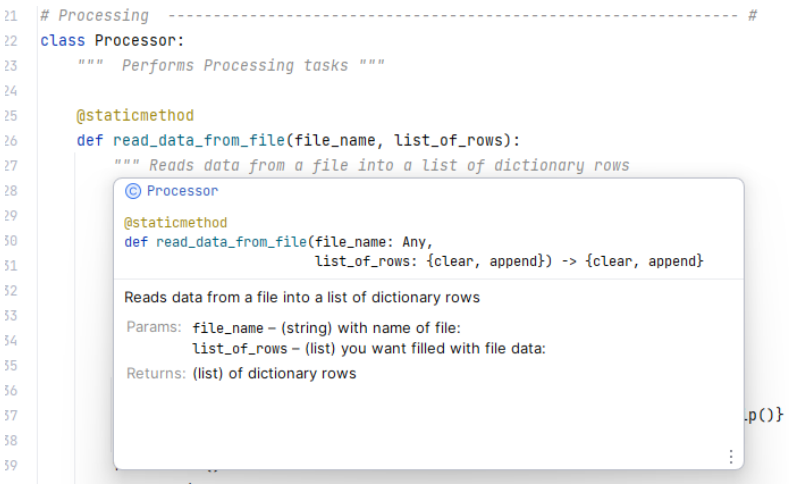

Figure 1: Example of decorators

Topic 2: Processors

The first set of functions in the template are labeled as processors and execute the same processes used in the previous module: import data from a file into a table of dictionary rows, add data to list, remove data from list, and save data to file. As you can see in Figure 2, setting up the new functions required defining parameters and return. The parameters are inputs to the function and their variable names within the function. Returns are the outputs and variable names within the function.

In the add and remove functions the inputs are list_of_rows, task, and priority. The most challenging part of the assignment was tracing the shadow variables in this program. Shadows are variables used in multiple code blocks that may be independent of each other if they're not passed in and out. I understand why input and output tasks were separated to illustrate the use of classes, but I think the program would be easier if user input collection and processing were one function.

```
42      @staticmethod
43      def add_data_to_list(task, priority, list_of_rows):
44          """ Adds data to a list of dictionary rows
45
46          :param task: (string) with name of task:
47          :param priority: (string) with name of priority:
48          :param list_of_rows: (list) you want to add more data to:
49          :return: (list) of dictionary rows
50          """
51          # format user input into dictionary row
52          row = {"Task": str(task).strip(), "Priority": str(priority).strip()}
53          # append row to the task list
54          list_of_rows.append(row)
55          print(priority + " priority task " + task + " added.")
56          return list_of_rows
57
58      @staticmethod
59      def remove_data_from_list(task, list_of_rows):
60          """ Removes data from a list of dictionary rows
61
62          :param task: (string) with name of task:
63          :param list_of_rows: (list) you want filled with file data:
64          :return: (list) of dictionary rows
65          """
66          # check to see if there are any tasks in list to remove
67          if len(list_of_rows) == 0:
68              print("Your list is empty.")
69          else:
70              # variable used to note that task was/not in list to remove
71              remove = 0
72              # loop to check each row and remove task
73              for row in list_of_rows:
74                  if row["Task"] == task:
75                      list_of_rows.remove(row)
76                      remove = 1
77                      print(task + " removed.")
78              # message to let user know if task not found in list
79              if remove == 0:
80                  print("Task not found.")
81          return list_of_rows
```

Figure 2: Add and remove tasks from list

Topic 3: Input and Output tasks

The next class of functions is inputs and outputs. This was a little bit tricky because these functions layer with the processing functions above and the returns here need to be in the same format as the parameters in the processors. On line 152 you can see where I renamed the function variable task and priority to new_task and new_priority to keep track of where the variables are being used. I changed the names back to the generic to stay inline with the template, but left this in to show that the names don't matter as long as they're ordered correctly within the () to input function parameters. This goes back to the earlier point on shadow variables that look like they're the same thing, but may not be as they are meant to exist inside a function block vs. the main body of the code.

```
121
122     @staticmethod
123     def input_menu_choice():
124         """ Gets the menu choice from a user
125         :return: string
126         """
127         choice = str(input("Which option would you like to perform? [1 to 4] - ")).strip()
128         print()  # Add an extra line for looks
129         return choice
130
131     @staticmethod
132     def output_current_tasks_in_list(list_of_rows):
133         """ Shows the current Tasks in the list of dictionaries rows
134         :param list_of_rows: (list) of rows you want to display
135         :return: nothing
136         """
137         print("******* The current tasks ToDo are: *******")
138         for row in list_of_rows:
139             print(row["Task"] + " (" + row["Priority"] + ")")
140         print("*******************************************")
141         print()  # Add an extra line for looks
142
143     @staticmethod
144     def input_new_task_and_priority():
145         """  Gets task and priority values to be added to the list
146         :return: (string, string) with task and priority
147         """
148         # obtain user input for task and priority
149         # convert answer to lower case
150         new_task = input("What is your quest? ").lower()
151         new_priority = input("What is the priority? ").lower()
152         return new_task, new_priority
153
```

Figure 3: Input and output functions

Topic 4: Main Body of Script

The last step in the program is building the main body of the script where functions are blocked out to execute in the correct order with the right data passed back and forth as parameters/returns. This is done by using the class prefix to make it clear what each function is supposed to do e.g. lines 180 and 181 in Figure 4. First the code collects the user input variables task and priority as the returns of IO.input_new_task_and_priority(): the function class is IO (input output) and the function name is input_new_task_and_priority. Next the program inputs the variables we just defined into Processor.add_data_to_list(task = task, priority=priority,list_of_rows=table_list)

- Function class = Processor
- Function = add_data_to_list

- task = value returned from function input_new_task_and_priority on line 180
- priority = value returned from function input_new_task_and_priority on line 180
- table_list = table created at the beginning of the program outside the function blocks
- list_of_rows = variable name used within the function add_data_to_list

```
# 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

    # Step 4 - Process user's menu choice
    if choice_str.strip() == '1':  # 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 == '2':  # 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 == '3':  # 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 == '4':  # Exit Program
        print("Goodbye!")
        break  # by exiting loop
```

Figure 4: Code execution with nested functions

Summary

In this program I used the input, output, and processing code I learned in module 5 and formatted it using classes and functions to make it easier to read and reuse. I then posted my code to GitHub here: CaseNightmareGreen/IntroToProg-Python-Mod06 (github.com). In the next module I will continue to use classes and functions to keep my code organized and easy to follow.

```
C:\Users\andre\AppData\Local\Microsoft\WindowsApps\python3.10.e
******* The current tasks ToDo are: *******
mow lawn (high)
wash dishes (medium)
wash hair (low)
*****************************************


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


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

What is your quest? make cake
What is the priority? immediate
immediate priority task make cake added.
******* The current tasks ToDo are: *******
mow lawn (high)
wash dishes (medium)
wash hair (low)
make cake (immediate)
*****************************************


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

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

Which task should be removed? mow lawn
mow lawn removed.
******* The current tasks ToDo are: *******
wash dishes (medium)
wash hair (low)
make cake (immediate)
*****************************************


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

Figure 5: Code executed in PyCharm

References

1. Dawson, Michael, Python Programming for the Absolute Beginner, 3rd ed. Boston, MA: Cengage Learning, Inc, 2010.

2. Stackoverflow, “Why do we use staticmethod.” [Online]. Available python - Why do we use @staticmethod? - Stack Overflow [Accessed May 22nd 2023]