Chiyo Miyake

2023.02.15

IT FDN 110 A

Assignment 06

https://github.com/bread-buddies/IntroToProg-Python-Mod06

Assignment 06: Functions and Classes

Introduction

In this assignment, we utilize using functions to better organize our scripts and improve the to do task list by utilizing functions.

To Do List

From the last module, we made a script to keep record of tasks. This time we improve and organize the script to practice using functions. For this assignment, we are given a template to work with and figure out how to fit the right code into each function. Just like the last assignment, we want to be able to store the table and the information in it to a file, we first declare some data at the beginning of the script (Figure 1).

```
12  # Data ---------------------------------------------------------------------- #
13  # Declare variables and constants
14  file_name_str = "ToDoFile.txt"  # The name of the data file
15  file_obj = None  # An object that represents a file
16  row_dic = {}  # A row of data separated into elements of a dictionary {Task,Priority}
17  table_lst = []  # A list that acts as a 'table' of rows
18  choice_str = ""  # Captures the user option selection
19
```

Figure 1: Declaring variables and constants.

Just as we learned, we can organize our script into different parts: Data, Process, and Presentation. We set up the data section with the declaration of the necessary variables and constants. Now we want to process data. A class is created which holds the different functions that will process the data we want. The first function is to read any data that is stored on the text file.

```python
# 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
        file = open(file_name, "r")
        for line in file:
            task, priority = line.split(",")
            row = {"Task": task.strip(), "Priority": priority.strip()}
            list_of_rows.append(row)
        file.close()
        return list_of_rows
```

Figure 2: Adding a class to hold processing functions and the function to read data from the file.

The next function we have is to add the data to the list (Figure 3). Not to be confused with adding input data to the list. This function is a separate function to add it to the list. Recording the data onto the text file is on a separate function where we record user inputs and add them to the file. The row gets added to the end of the table, that is the list of rows.

```python
def add_data_to_list(task, priority, list_of_rows):
    """ Adds data to a list of dictionary rows

    :param task: (string) with name of task:
    :param priority: (string) with name of priority:
    :param list_of_rows: (list) you want to add more data to:
    :return: (list) of dictionary rows
    """
    row = {"Task": str(task).strip(), "Priority": str(priority).strip()}
    # TODO: Add Code Here!
    list_of_rows.append(row) #add new data to the list of rows

    return list_of_rows
```

Figure 3: Function to add data to the list of rows.

The next function is to delete a task from the list. We simply utilize the built in function to remove a row from the table (Figure 4).

```
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!
    for row in list_of_rows: # loop through the rows
        if row["Task"].lower() == task.lower(): # look for the task that the user wants to delete
            list_of_rows.remove(row) # removes the row
    return list_of_rows
```

Figure 4: Function to remove data from the list.

The last function that will be processing the data is writing the data to the file. The row itself will be recorded on the text file (Figure 5).

```
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_str, "w")
    for row in list_of_rows:
        file.write(row["Task"] + "," + row["Priority"] + "\n") # records data
    file.close() #closes file to save the record

    return list_of_rows
```

Figure 5: Function to write data to the file.

Now to 'present' our data by adding a user interface and utilizing the functions we created to process the data. We first want to inform the user of what options they have by printing the menu, which is done by calling the function we created for it (Figure 6). Figures 7 and 8 show the program running on PyCharm and the command prompt, respectively.

```
91      class IO:
92          """ Performs Input and Output tasks """
93
94          @staticmethod
95          def output_menu_tasks():
96              """  Display a menu of choices to the user
97
98              :return: nothing
99              """
100             print('''
101             Menu of Options
102             1) Add a new Task
103             2) Remove an existing Task
104             3) Save Data to File
105             4) Exit Program
106             ''')
107             print()  # Add an extra line for looks
```

Figure 6: New class for the interfacing and function to print the user menu of options.

```
        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] -
```

Figure 7: Menu display on PyCharm.

```
        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] - 3
```

Figure 8: Menu display on Command Prompt.

The next few functions are to be able to call user input and display the updated menu (Figure 9). The input function asks what option the user wants to choose and processes it based on the next function that will go through if statements. The output function shows the current list and prints each task within the table.

```
    def input_menu_choice():
        """ Gets the menu choice from a user

        :return: string
        """
        choice = str(input("Which option would you like to perform? [1 to 4] - ")).strip()
        print()  # Add an extra line for looks
        return choice

    @staticmethod
    def output_current_tasks_in_list(list_of_rows):
        """ Shows the current Tasks in the list of dictionaries rows

        :param list_of_rows: (list) of rows you want to display
        :return: nothing
        """
        print("******* The current tasks ToDo are: *******")
        for row in list_of_rows:
            print(row["Task"] + " (" + row["Priority"] + ")")
        print("*******************************************")
        print()  # Add an extra line for looks
```

Figure 9: Input function and output the current tasks function.

Finally, the last two functions are for using user inputs to add or remove data (Figure 10). This is different from the processing functions mentioned earlier because we ask for a user input, essentially presenting the user options to add tasks.

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

        :return: (string, string) with task and priority
        """
        task = input("What is the task called?: ").strip()
        priority = input("What is the priority level?: ").strip()
        print()
        return task, priority

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

        :return: (string) with task
        """
        task = input("What task do you want to remove?: ").strip()
        print()
        return task
```

Figure 10: Functions for user input to add or remove tasks.

Now that we have established the needed functions, we can write the main program (Figure 11). We first process the text file and then go into a while loop to allow the user to choose different options. Based on the options chosen, the if statements will call the different functions to process the data and store, or remove, the data on the text file.

```
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 11: Main body of the program.

The following figures shows the programs run on PyCharm and the command prompt.

```
******* The current tasks ToDo are: *******
Clean (Medium)
*****************************************


        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] -
What is the task called?: What is the priority level?:
******* The current tasks ToDo are: *******
Clean (Medium)
Laundry (Low)
*****************************************
```

Figure 12: Program run on PyCharm to add a task.

```
******* The current tasks ToDo are: *******
Laundry (Low)
Feed cats (High)
Load dishwasher (Medium)
*****************************************


        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

What task do you want to remove?: Feed cats

******* The current tasks ToDo are: *******
Laundry (Low)
Load dishwasher (Medium)
*****************************************
```

Figure 13: Program run on Command Prompt to remove a task.

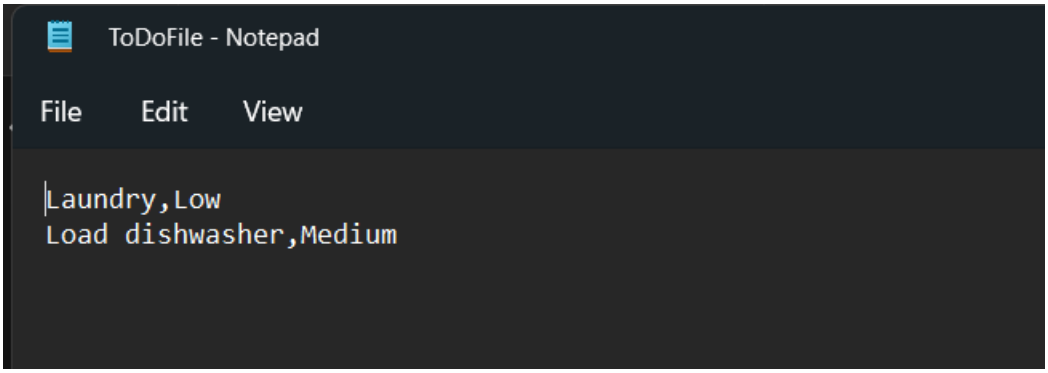

Figure 14: Text file with the updated records.

Summary

In summary, functions and classes are useful when you're organizing your program and streamlining the process. With the different functions, it reduces the amount of time you spend debugging because you call on individual functions and it is organized to be intuitive.