

To Do List - Custom Functions

Introduction

During my 6th assignment, I was given another starting template by Professor Root for us to modify and to add in the missing code. It's very similar to last week's assignment in which we either add or remove input data from a list that contains two columns of data which are representatives of a "task" and "priority". The only difference is utilizing the concept around separation of concerns through the use of classes and functions. Throughout this paper, I will show you how I was able to fill in the missing code with separate classes and its functions.

Processing functions

The very first function (read_data_from_file) reads data from a file which requires two separate parameters which are the name of the file that contains the tasks and priorities and the other being the list that stores data into a list. The For Loop was thrown in to scan every row and the comma used as a split function separates them. Also, the use of append function enables the function to return the list of row datas.

Figure 1:

```
@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, file_name
```

This next function adds data to a list of dictionary rows. It has three separate parameters such as task, priority, and the list to which new data is appended. New data elements such as task and priority are added to a list of dictionary rows as key/value pairs. The function then returns the modified list.

Figure 2:

```
@staticmethod
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 filled with file data:
    :return: (list) of dictionary rows
    """
    row = {"Task": str(task).strip(), "Priority": str(priority).strip()}
    # TODO: Add Code Here!
    list_of_rows.append(row)
    return list_of_rows
```

The next function removes data from a list of dictionary rows. Basically, it's removing an existing task from the list if the string input is found within a list of dictionary rows. Once found, the supplied task is removed while returning the rest of the list.

Figure 3:

```
@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!
    for row in list_of_rows:
        if row["Task"].lower() == task.lower():
            list_of_rows.remove(row)
            print("You have successfully removed the task from the list")
    return list_of_rows
```

Finally, this final function writes data from a list of dictionary rows to a file. The parameters used for this function are the file name and the list of the file. Also, this function will shape how each dictionary row will look like in the file. Task and Priority will be separated by a comma separated formatting and a newline character to indicate the end of the dictionary row.

Figure 4:

```
@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(row["Task"] + ',' + row["Priority"] + '\n')
    file.close()
    return list_of_rows
```

Processing functions

The first function is a display menu of four separate options. Once called, the menu is printed to prompt the user. This is done in a repetitive manner. There are no parameters used for this function.

Figure 5:

```
@staticmethod
def output_menu_tasks():
    """ Display a menu of choices to the user

    :return: nothing
    """

    print('''
Menu of Options
1) Add a new Task
2) Remove an existing Task
3) Save Data to File
4) Exit Program
''')
    print() # Add an extra line for looks
```

The `input_menu_choice` function will then incorporate the user input of choice 1, 2, 3, or 4 to be stored into the choice parameter. This will return back to the main body .

Figure 6:

```
@staticmethod
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
```

This function displays current tasks and priorities that are currently stored in the list of dictionary rows. The list is used as a parameter. This function loops through the parameter and then prints out the current tasks back to the User. There isn't a return parameter for this type of function.

Figure 7:

```
@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
```

This function is used to prompt User into entering in a new task and priority. Once User provides these datas, both task and priority are returned to the main body.

Figure 8:

```
@staticmethod
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 = str(input("What is the task? - ")).strip()
    priority = str(input("What is the priority? - ")).strip()
    return task, priority
```

The final function asks the user which task should be removed from the current list. Once User input is entered, it is returned to the main body.

Figure 9:

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

    :return: (string) with task
    """
    task = str(input("What is the name of the task you wish to remove? - ")).strip()
    print()
    return task
```

Main Body of Script

There are two parts of the main body of the script, one loading the data from the text file and the rest of the main body being wrapped in an infinite while loop. There are three I/O functions at the start of the loop which are showing the current data in the list, displaying the menu, and getting the user's choice input which is stored in strChoice.

If option one is selected, IO.input_new_task_and_priority will grab user inputs to be stored into variables of task and priority. The Processor.add_data_to_list is supplied by these two along with table_1st. This returns the modified list that is currently stored in table_1st.

If option two is selected, IO.input_task_to_remove will grab User input for a particular task to be stored into task. The Processor.remove_data_from_list is supplied by the task and the current list.

If option three is selected, the current list and file data will be assigned Processor.write_data_to_file the written data to the file data. This function returns the list_of_rows stored into the data file.

If option four is selected, 'Adios' is printed to the User and ends the loop by breaking the infinite loop.

Figure 10:

```
#!/usr/bin/env python3
# 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):
    # 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
    strChoice = IO.input_menu_choice() # Get menu option

    # Step 4 - Process user's menu choice
    if strChoice.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 strChoice == '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 strChoice == '3': # Save Data to File
        table_lst = Processor.write_data_to_file(file_name=file_name_str, list_of_rows=table_lst)
        print("Data is saved to File!")
        continue # to show the menu

    elif strChoice == '4': # Exit Program
        print("Adios!")
        break # by exiting loop
    else:
        print("Can you please enter 1, 2, 3, or 4?")
```

Figure 11:

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

What is the name of the task you wish to remove? - Dishes

You have successfully removed the task from the list
***** The current tasks ToDo are: *****
Mowing (High)
Homework (High)
Laundry (Medium)
*****
```

Figure 12:

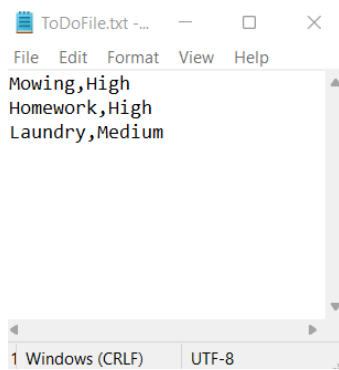
```
Which option would you like to perform? [1 to 4] - 3

Data is saved to File!
***** The current tasks ToDo are: *****
Mowing (High)
Homework (High)
Laundry (Medium)
*****
```

Figure 13:

```
Which option would you like to perform? [1 to 4] - 4  
  
Adios!  
  
Process finished with exit code 0
```

Figure 14:



ToDoFile.txt - ...

File Edit Format View Help

Mowing,High
Homework,High
Laundry,Medium

1 Windows (CRLF) UTF-8

Figure 15:

Next Page

```
***** The current tasks ToDo are: *****
Mowing (High)
Homework (High)
Laundry (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] - 1

What is the task? - Vacuum
What is the priority? - Low
***** The current tasks ToDo are: *****
Mowing (High)
Homework (High)
Laundry (Medium)
Vacuum (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] - 2

What is the name of the task you wish to remove? - Vacuum

You have successfully removed the task from the list
***** The current tasks ToDo are: *****
Mowing (High)
Homework (High)
Laundry (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] - 3

Data is saved to File!
***** The current tasks ToDo are: *****
Mowing (High)
Homework (High)
Laundry (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] -
```


Summary

Throughout this assignment, I learned to complete a starter program by creating custom functions. By using concepts around separation of concerns, I was able to define classes and functions then utilize them into the program. I understand now how useful these concepts can be to rework existing codes. I've learned a lot on creating custom functions in a starter program. Now, I'm looking forward to learning more about exception handling and how to use classes in a more efficient way.