Programming Assignment 2

Multidimensional Lists/Mutability and Dictionary

This assignment is an Individual Effort assignment.

Learning Objectives:

- Using multidimensional lists
- Mutability
- Exploring other sequences such as dictionaries.

General Assumptions

We have some assumptions to make the project easier. You may assume that:

- The types of the values that are sent to the functions are the proper ones and you don't have to validate them.
- Be sure to read the assumptions made in individual problems!

Restrictions

Any function that violates any of the following will lose significant points.

- You are **not** allowed to **import** anything
- You are **not** allowed to use slicing
- You are **not** allowed to use these functions: **dict()**, **sum()**, **zip()**, **enumerate()**
- You are **not** allowed to use list comprehension
- Any arithmetic/comparison/boolean operators are all fine to use, such as **+**, **-**, *, /, //, **%**, **and**, **or**, **>=**, **<**, etc.
- You **may use** **range()**, **len()**, and these type casting functions as needed: **int()**, **str()**, **float()**, **list()**
- You **may use** **del**, **in**, and **not in** operators
- From list methods, you are **only** allowed to use **.append()**, **.pop()**, **.remove()**, **.clear()**
- From dictionary methods, you are **only** allowed to use **.keys()**, **.values()**, **.items()**
- You are **not** allowed to use anything that hasn't been covered in class yet

- Pay close attention to specific restrictions/assumptions (if any) written in for individual functions
- There is **no user input or print** statements. Your functions should not ask for user input and should not print anything, just return a value if needed based on the specification of the tasks.
- Don't forget to comment your code!
- No hard coding!

Submitting Your Code

Submit your .ipynb file under PA2

Functions

A tech company has hired you to analyze data for their social media platform. The social media platform collects user engagement data (data for likes, comments, shares, etc.) for the users and the posts, and you are to develop these **five functions below** to organize and make sense of the data.

```
def filter_popular(reacts_2D, names, threshold):
```

Description: You are provided a two-dimensional list of engagement data counts for some users, a one-dimensional list of those user names, and an integer threshold to determine the popular users. A user is popular if the total engagement count for that user is equal or more than the threshold. Your function should return a one-dimensional list of users who are popular.

Parameters:

- `reacts_2D` - A 2D list. Each item is a list that contains integers representing engagement data counts for a user.
- `names` - A 1D list. Each item is a string representing a user name.
- `threshold` - An integer representing the threshold for determining the popular users.

Assumptions:

- `reacts_2D` and `names` lists are in the same order and of same length. This means first item of `reacts_2D` has engagement data counts for the first user in `names`, second item of `reacts_2D` has engagement data counts for the second user in `names`, and so on.
- Each name in the `names` list is unique.
- Items in `reacts_2D` can be empty lists.

Return value: A 1D list. Each item is a user name string from `names` indicating a popular user.

Examples:

```
filter_popular([[4, 9, 6, 5], [1, 2, 3, 5, 8], [17, 2, 9]],
               ["crazy_guy", "solid321", "amicoolyet"], 22)
→ ['crazy_guy', 'amicoolyet']
```

```
filter_popular([[4, 9, 6, 5], [1, 2, 3, 5, 8], [17, 2, 9]],
               ["crazy_guy", "solid321", "amicoolyet"], 15)
→ ['crazy_guy', 'solid321', 'amicoolyet']

filter_popular([[31], [22, 1, 1], [2, 2, 11, 65]],
               ["alien", "tomato2", "simon23"], 50)
→ ['simon23']
```

```
def gather_engagement(names, reacts, grouping):
```

Description: You are provided a one-dimensional list of user names, a one-dimensional list of ungrouped engagement data counts for those users, and a one-dimensional list with information about how to group the engagement data counts by user. Your function should gather the engagement data counts for each user to create the groups and return a two-dimensional list that contains the groups.

Parameters:

- `names` - A 1D list. Each item is a string representing a user name.
- `reacts` - A 1D list. Each item is an integer representing an engagement data count.
- `grouping` - A 1D list. Each item is an integer representing how many consecutive items in the `reacts` list corresponds to a particular user from the `names` list.

Assumptions:

- Each user's data comes in the `reacts` list in the order of the `names` list. This means the engagement data for the first user from the `names` list comes first in the `reacts` list, the engagement data for the second user from the `names` list comes after the first user's data in the `reacts` list, and so on.
- `names` and `grouping` lists are in the same order and of same length. This means first item of `grouping` list indicates how many consecutive items from the `reacts` list corresponds to the first user from the `names` list, second item of `grouping` list indicates how many consecutive items from the `reacts` list corresponds to the second user from the `names` list, and so on.
- Sum of the items in the `grouping` list is equal to the length of the `reacts` list.
- An item in the `grouping` list can be 0, which means the corresponding user have no data.
- Each name in the `names` list is unique.

Return value: A 2D list. Each item is a list that contains grouped user engagement data for a user where the first item is that user's name (string) from the `names` list and rest of it is that user's data (integers) from the `reacts` list in the same order as they appear in the `reacts` list.

Examples:

```
gather_engagement(["crazy_guy", "solid321", "amicoolyet"],
                  [4, 9, 6, 5, 1, 2, 3, 5, 8, 17, 2, 9],
                  [4, 5, 3])
→ [['crazy_guy', 4, 9, 6, 5], ['solid321', 1, 2, 3, 5, 8], ['amicoolyet', 17, 2, 9]]
```

```
gather_engagement(["crazy_guy", "solid321", "amicoolyet"],
                  [4, 9, 6, 5, 1, 2, 3, 5, 8, 17, 2, 9],
                  [2, 1, 9])
→ [['crazy_guy', 4, 9], ['solid321', 6], ['amicoolyet', 5, 1, 2, 3, 5, 8, 17, 2, 9]]

gather_engagement(["butter12", "helloworld"],
                  [40, 3, 35, 7],
                  [0, 4])
→ [['butter12'], ['helloworld', 40, 3, 35, 7]]
```

```
def clear_zeros(reacts_2D):
```

Description: You are provided a two-dimensional list of engagement data counts for some users and your job is to clean this data by getting rid of all zeros. After removing the zeros, if an item becomes an empty list, your function should remove that item as well. The non-zero values in the item should remain in the same order.

Parameters:

- `reacts_2D` - A 2D list. Each item is a list that contains integers representing engagement data counts for a user.

Assumptions:

- Items in `reacts_2D` are not empty lists initially.

Return value: A 2D list.

Examples:

```
user_reacts = [[4, 9, 0, 0], [1, 2, 0, 5, 8], [17, 2, 0]]
clear_zeros(user_reacts)
→ [[4, 9], [1, 2, 5, 8], [17, 2]]

user_reacts = [[1, 2, 0], [0, 0, 0], [17, 2, 4, 0]]
clear_zeros(user_reacts)
→ [[1, 2], [17, 2, 4]]

user_reacts = [[40, 3], [35, 7]]
clear_zeros(user_reacts)
→ [[40, 3], [35, 7]]
```

```
def form_reactions_list(react_dict1, react_dict2):
```

Description: You are provided two dictionaries that contain user reactions (like, comment, share, etc.) as keys and the count of such reactions as values for a post. You need to combine these dictionaries to return a two-dimensional list that contains every reaction from the dictionaries exactly once and their counts. If a reaction appears in both dictionaries, the returned list should have the total count for that reaction.

Parameters:

- `react_dict1` - A dictionary. Each item's key is a reaction (string) and value is the count (integer) for that reaction.
- `react_dict2` - A dictionary. Each item's key is a reaction (string) and value is the count (integer) for that reaction.

Assumptions:

- A string can appear as a key in both `react_dict1` and `react_dict2`

Return value: A 2D list. Each item is a list that has 2 items in this format: [a reaction (string), count (integer) for that reaction].

Examples:

```
react1 = {"like" : 5, "comment" : 10, "share" : 3}
react2 = {"love" : 10, "like" : 5, "wow" : 2}
form_reactions_list(react1, react2)
```

→ `[['like', 10], ['comment', 10], ['share', 3], ['love', 10], ['wow', 2]]`

```
react1 = {"wow" : 34, "angry" : 9, "comment" : 1, "sad" : 42}
react2 = {"wow" : 34, "angry" : 9, "comment" : 1, "sad" : 42}
form_reactions_list(react1, react2)
```

→ `[['wow', 68], ['angry', 18], ['comment', 2], ['sad', 84]]`

```
react1 = {"angry" : 54, "love" : 11}
react2 = {"share" : 21}
form_reactions_list(react1, react2)
```

→ `[['angry', 54], ['love', 11], ['share', 21]]`

```
def form_reactions_dict(reacts_2D):
```

Description: You are provided a two-dimensional list that contains user reactions (like, comment, share, etc.) and their counts for a post in the same format as the last function's (`form_reactions_list`) returned list. You need to organize this information in a dictionary that contains the reactions from the list as keys and their counts as values. Your returned dictionary should have an additional item at the end with the string key `"total"` and an integer value that represents the total count for all the reactions.

Parameters:

- `reacts_2D` - A 2D list. Each item is a list that has 2 items in this format: [a reaction (string), count (integer) for that reaction].

Assumptions:

- Each reaction string appears exactly once in `reacts_2D`

Return value: A dictionary. Each item's key is a reaction (string) from `reacts_2D` and value is the count (integer) from `reacts_2D` for that reaction. The additional last item's key is a string `"total"` and the value is an integer representing the sum of all reaction counts.

Examples:

```
form_reactions_dict([['like', 10], ['comment', 10], ['share', 3],
['love', 10],
['wow', 2]])
```

```
→ {'like': 10, 'comment': 10, 'share': 3, 'love': 10, 'wow': 2,
'total': 35}
```

```
form_reactions_dict([['comment', 21], ['share', 26], ['love', 10],
['like', 5],
['wow', 2]])
```

```
→ {'comment': 21, 'share': 26, 'love': 10, 'like': 5, 'wow': 2,
'total': 64}
```

```
form_reactions_dict([['wow', 34], ['angry', 9], ['comment', 1],
['sad', 43], ['haha',
11], ['love', 2]])
```

```
→ {'wow': 34, 'angry': 9, 'comment': 1, 'sad': 43, 'haha': 11,
'love': 2, 'total':
100}
```