Homework 6

Total Points: 100 (Correctness and Style) + 3 EC (Checkpoint)

Due Dates (SD time):

- **Checkpoint (read below): Sunday, February 18th, 11:59pm**
- **Entire Assignment: Tuesday, February 20th, 11:59pm.**

Starter Files

Download hw06.zip. Inside the archive, you will find starter files for the questions of this homework. You cannot import anything to solve the problems.

IMPORTANT: Coding and Docstring Style

This is a reminder that your code style will be graded. Here are a few useful links:

Style Guide Document

Style Guide on Course Website

Style Guide Examples

Testing

At any point of the homework, use one of the following command to test your work:

```
>>> python3 -m doctest hw06.py
>>> py -m doctest hw06.py
>>> python -m doctest hw06.py
```

Checkpoint Submission

Due Date: Tuesday, February 20th, 11:59pm.

You can earn up to **3 points extra credit** by submitting the checkpoint by the due date above. In the checkpoint submission, you should complete **Question 1, 2, 3, 4** and submit the hw06.py file to gradescope.

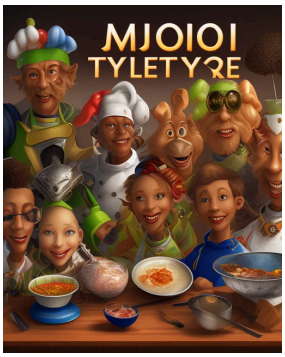
Checkpoint submission is graded by completion, which means you can get full points if your code can pass some simple sanity check (no tests against edge

cases). Note that in your final submission, you should still submit these questions, and you may modify your implementation if you noticed any errors.

General Notes and Requirements

1. **DO NOT IMPORT ANY PACKAGES.**
2. Please add your own doctests (**at least three**) as the given doctests are not sufficient. **You will be graded on the doctests**.
3. When a question **(ALL QUESTIONS)** requires **recursion**: No for/while loops, no list comprehension, no map/filter, no helper functions. Make calls to the function itself. You will **NOT** get credit for the question if you don't follow this rule.
4. For this homework only, **no assert statements needed.** Assume all inputs are valid.

Required Questions



You decide to host a cooking class for your friends, providing them with a nice break from their busy lives. Let's get ready!

Question 1:

First, you want to print out invitations, but realize that your printer prints everything in reverse. You decide to write a **recursive** function that would reverse the strings so they print out properly.

Input: A string.

Output: Same string reversed.

Requirement: Recursion Only. No loops/comprehension/map/filter/helper or built-in functions.

```
def reverse_str(name):
    """
    >>> reverse_str('Lina')
    'aniL'
    >>> reverse_str('anna')
    'anna'
    >>> reverse_str('Hello World')
    'dlroW olleH'
    """
```

Question 2:

You decided that it is far too time consuming to reverse each string one by one, wouldn't it be nice if you could just handle a batch of invitations all in one go. Write a function that **recursively** reverses each string in the list.

Input: a list of strings
Output: a list of strings, where each string is reversed

Requirement: Recursion. No loops/comprehension/map/filter/helper functions, *except* you can call the function from question 1 to solve this problem.

```
def reverse_all(names_lst):
    """
    >>> names_lst = ["Emma", "Liam", "Olivia", "Noah", "Ava"]
    >>> reverse_all(names_lst)
    ['ammE', 'maiL', 'aivilO', 'haoN', 'avA']
    >>> names_lst = ["William"]
    >>> reverse_all(names_lst)
    ['mailliW']
    >>> names_lst = ["Isabella", "Oliver", "Mia"]
    >>> reverse_all(names_lst)
    ['allebasI', 'revilO', 'aiM']
    """
```

Question 3:

You want to make sure that every person will be able to fit at one of the cooking stations. You know how many people are arriving and the size of each station (by default 4 but can change upon specification), so you want to specify how many tables you would like to use.

Inputs:

- `total_ppl` (non-negative integer): total number of people coming
- `table_size` (positive integer): table size, by default set at 4

Output: number of tables needed

Requirement: **Recursion.** No loops/comprehension/map/filter/helper functions.

```
def num_tables(total_ppl, table_size=4):
    """
    >>> num_tables(35)
    9
    >>> num_tables(20, 2)
    10
    >>> num_tables(2, 20)
    1
    """
```

Question 4:

You need to order supplies for the cooking class, however your vendor only accepts orders of size no more than three. This means you will have to place several orders, and you want to know what is the total cost of each order you place.

Inputs:

- `price_lst` (`list of positive integers`): the prices of each item that you want to purchase
- `order_size (positive integer)`: size of each order, be default set to 3

Output: A list that contains the total cost of each order.

Requirement: **Recursion.** No loops/comprehension/map/filter/helper functions.

- You are allowed to use built-in function `sum`, but your solution still needs to be recursive.

Input	Output	Explanation
`[2, 2, 2, 4, 4, 4]`	`[6, 12]`	The cost of the first order will be the cost of the first three items combined: 2 + 2 + 2 = 6 The cost of the second order will be the cost of the second three items combined: 4 + 4 + 4 = 12 The result is [6, 12]

```
def orders(price_lst, order_size=3):
    """
    >>> orders([2, 2, 2, 4, 4, 4])
    [6, 12]
    >>> orders([])
    []
    >>> orders([2, 2, 2, 2])
    [6, 2]
    """
```

Question 5:

Some guests still haven't responded to your invite and you want to get their attention. You decided to send them a reminder and confuse them a little bit to get their attention. So you send them a reminder where every time a word has two of the same characters in a row you decide to change the second character to an *.

Input: a string
Output: string with the second duplicate character in a row replaced by a *

Requirement: Recursion. No loops/comprehension/map/filter/helper functions. **Can not use .index() or .find(), or replace()**, you must look for characters recursively.

Example:

Input	Output	Explanation
‘book’	‘bo*k’	Letter ‘o’ is repeated twice in a row, so we replace the second ‘o’ with a ‘*’ symbol
‘boook’	'bo*ok'	Letter ‘o’ is repeated twice in a row, so we replace the second ‘o’ with a ‘*’ symbol
'booook'	'bo*o*k'	Letter ‘o’ is repeated twice in a row, so we replace the second ‘o’ with a ‘*’ symbol, same for the other two letters.

```
def confuse(reminder_str):
    """
    >>> confuse('book')
    'bo*k'
    >>> confuse('successful')
    'suc*es*ful'
    >>> confuse('')
    ''
    """
```

Question 6:

After some consideration, you've decided that it's best to have multiple tables of various sizes for each decorating station, as some of your friends will be coming in groups. You've assigned seating for each group, but due to tiredness, you're unsure if all seat assignments are correct. Write a **recursive** function that double checks whether each group will fit at their designated table (group matches the table based on index), and return the number of groups that need to be reassigned to larger tables. (Note that the lists may have different lengths.)

Inputs:

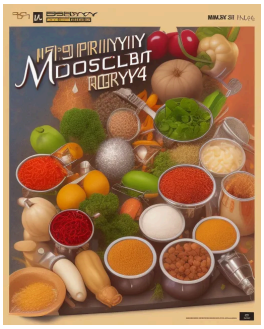
- `group_sizes` : a list of integers, where every integer represents the size of a group.
- `table_sizes` : a list of integers, where every integer represents the total number of seats at the tables.

Output: the number of groups that are assigned to a table that does not have enough seats for everyone at the table.

Input	Output	Explanation
`[2, 3, 4, 5], [3, 3, 3]`	2	We are given: [2, 3, 4, 5] [3, 3, 3] If we compare the lists, then 2 <= 3 → enough seats, we do nothing 3 <= 3 → enough seats, we do nothing 4 **>** 3 → not enough seats, we count as 1 5 doesn't have a matching table, so 5 and everything that would come after it would need a table assigned to it, so we count this as 1 more table adjustment Total: 1 + 1 = 2 tables adjustments

```
def seating_chart(group_sizes, table_sizes):
    """
    >>> seating_chart([3, 3, 3], [4, 4, 4])
    0
    >>> seating_chart([2, 3, 4, 5], [3, 3, 3])
    2
    >>> seating_chart([2, 2], [])
    2
    """
```

Question 7:



Now that people have arrived, it is time to cook! You have a good collection of ingredients but some of them are common allergens so they have to be removed from the box.

Write a **recursive** function that takes two parameters:

- `ingredients`: A list of strings
- `item_to_remove`: a string that needs to be removed from the list `ingredients`

It should return a new list where all occurrences of `items_to_remove` are removed **(case insensitive)** and the list is in the *reversed* order.

Requirement: Recursion. No loops/comprehension/map/filter/helper functions

```
def allergens(ingredients, item_to_remove):
    """
    >>> allergens(["Sugar", "Jam", "Honey", "Honey"], "Honey")
    ['Jam', 'Sugar']
    >>> allergens(["Kefir", "", "Juice", "Water", "Milk"], "milk")
    ['Water', 'Juice', '', 'Kefir']
    >>> allergens(["Pecans", "PECANS", "PeCANS"], "Pecans")
    []
    """
```

Submission

Please submit the homework via Gradescope. You may submit more than once before the deadline, and only the final submission will be graded. Please refer to the Gradescope FAQ if you have any issues with your submission.