OOP

1. Read Chapter 9 (Classes) fully and thoroughly from the book 'PYTHON CRASH COURSE' by Eric Matthes. Retrieve the primary information about classes and articulate it in your own words in 2 pages of your Word document **(10 points)**

---

*Easy level

2. You are climbing a staircase. It takes n steps to reach the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top? **(5 points)**

Example 1:
Input: n = 2
Output: 2
Explanation: There are two ways to climb to the top.
1. 1 step + 1 step
2. 2 steps

Example 2:
Input: n = 3
Output: 3
Explanation: There are three ways to climb to the top.
1. 1 step + 1 step + 1 step
2. 1 step + 2 steps
3. 2 steps + 1 step

class Solution(object):
def climbStairs(self, n):

---

*Medium level

3. In English, we have a concept called root, which can be followed by some other word to form another longer word - let's call this word successor. For example, when the root "an" is followed by the successor word "other", we can form a new word "another".

Given a *dictionary* consisting of many roots and a *sentence* consisting of words separated by spaces, replace all the successors in the sentence with the root forming it. If a successor can be replaced by more than one root, replace it with the root that has the shortest length.

Return the *sentence* after the replacement. **(10 points)**

Example 1:

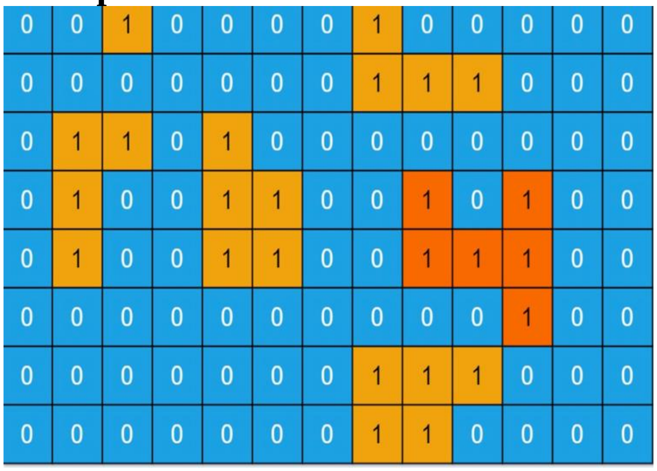
Input: dictionary = ["cat","bat","rat"], sentence = "the cattle was rattled by the battery"
Output: "the cat was rat by the bat"

Example 2:
Input: dictionary = ["a","b","c"], sentence = "aadsfasf absbs bbab cadsfafs"
Output: "a a b c"

```
class Solution(object):
def replaceWords(self, dictionary, sentence):
```

4. You are given an *m x n* binary matrix *grid*. An island is a group of 1's (representing land) connected 4-directionally (horizontal or vertical.) You may assume all four edges of the grid are surrounded by water.
The area of an island is the number of cells with a value 1 in the island.

Return the maximum area of an island in *grid*. If there is no island, return 0. **(10 points)**

Example 1:


0	0	1	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1	0	0	0
0	1	1	0	1	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	0	1	0	1	0	0
0	1	0	0	1	1	0	0	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	1	1	0	0	0	0

Input:
grid=
[[0,0,1,0,0,0,0,1,0,0,0,0,0],[0,0,0,0,0,0,0,1,1,1,0,0,0],[0,1,1,0,1,0,0,0,0,0,0,0,0],
[0,1,0,0,1,1,0,0,1,0,1,0,0],[0,1,0,0,1,1,0,0,1,1,1,0,0],[0,0,0,0,0,0,0,0,0,0,1,0,0],
[0,0,0,0,0,0,0,1,1,1,0,0,0],[0,0,0,0,0,0,0,1,1,0,0,0,0]]
Output: 6
Explanation: The answer is not 11, because the island must be connected 4-directionally.

Example 2:
Input: grid = [[0,0,0,0,0,0,0,0]]
Output: 0

```
class Solution(object):
def maxAreaOfIsland(self, grid):
```

5. There is a bookstore owner that has a store open for *n* minutes. Every minute, some number of customers enter the store. You are given an integer array *customers* of length *n* where *customers[i]* is the number of the customer that enters the store at the start of the *ith* minute and all those customers leave after the end of that minute.
On some minutes, the bookstore owner is grumpy. You are given a binary array grumpy where *grumpy[i]* is *1* if the bookstore owner is grumpy during the *ith* minute, and is *0* otherwise.
When the bookstore owner is grumpy, the customers of that minute are not satisfied, otherwise, they are satisfied.
The bookstore owner knows a secret technique to keep themselves not grumpy for *minutes* consecutive minutes, but can only use it once.

Return the maximum number of customers that can be satisfied throughout the day. **(10 points)**

Example 1:
Input: customers = [1,0,1,2,1,1,7,5], grumpy = [0,1,0,1,0,1,0,1], minutes = 3
Output: 16
Explanation: The bookstore owner keeps themselves not grumpy for the last 3 minutes.
The maximum number of customers that can be satisfied = 1 + 1 + 1 + 1 + 7 + 5 = 16.

Example 2:
Input: customers = [1], grumpy = [0], minutes = 1
Output: 1

```
class Solution(object):
def maxSatisfied(self, customers, grumpy, minutes):
```

---

6. Given an integer array *nums*, return the maximum possible sum of elements of the array such that it is divisible by three. **(10 points)**

Example 1:
Input: nums = [3,6,5,1,8]
Output: 18
Explanation: Pick numbers 3, 6, 1 and 8 their sum is 18 (maximum sum divisible by 3).
Example 2:
Input: nums = [4]
Output: 0
Explanation: Since 4 is not divisible by 3, do not pick any number.
Example 3:
Input: nums = [1,2,3,4,4]

Output: 12
Explanation: Pick numbers 1, 3, 4 and 4 their sum is 12 (maximum sum divisible by 3).

```
class Solution(object):
def maxSumDivThree(self, nums):
```

---

7. You would like to make dessert and are preparing to buy the ingredients. You have *n* ice cream base flavors and *m* types of toppings to choose from. You must follow these rules when making your dessert:

- There must be exactly one ice cream base.
- You can add one or more types of topping or have no toppings at all.
- There are at most two of each type of topping.

You are given three inputs:

- *baseCosts*, an integer array of length *n*, where each *baseCosts[i]* represents the price of the *ith* ice cream base flavor.
- *toppingCosts*, an integer array of length *m*, where each *toppingCosts[i]* is the price of oneof the *ith* topping.
- *target*, an integer representing your target price for dessert.

You want to make a dessert with a total cost as close to *target* as possible.

Return the closest possible cost of the dessert to *target*. If there are multiple, return the lower one. **(10 points)**

Example 1:
Input: baseCosts = [1,7], toppingCosts = [3,4], target = 10
Output: 10
Explanation: Consider the following combination (all 0-indexed):
- Choose base 1: cost 7
- Take 1 of topping 0: cost 1 x 3 = 3
- Take 0 of topping 1: cost 0 x 4 = 0

Total: 7 + 3 + 0 = 10.

Example 2:
Input: baseCosts = [2,3], toppingCosts = [4,5,100], target = 18
Output: 17
Explanation: Consider the following combination (all 0-indexed):
- Choose base 1: cost 3
- Take 1 of topping 0: cost 1 x 4 = 4
- Take 2 of topping 1: cost 2 x 5 = 10
- Take 0 of topping 2: cost 0 x 100 = 0

Total: 3 + 4 + 10 + 0 = 17. You cannot make a dessert with a total cost of 18.

```
class Solution(object):
def closestCost(self, baseCosts, toppingCosts, target):
```

---

8. You are given an integer *total* indicating the amount of money you have. You are also given two integers *cost1* and *cost2* indicating the price of a pen and pencil respectively. You can spend part or all of your money to buy multiple quantities (or none) of each kind of writing utensil.

Return the number of distinct ways you can buy some number of pens and pencils. **(10 points)**

Example 1:
Input: total = 20, cost1 = 10, cost2 = 5
Output: 9
Explanation: The price of a pen is 10 and the price of a pencil is 5.
- If you buy 0 pens, you can buy 0, 1, 2, 3, or 4 pencils.
- If you buy 1 pen, you can buy 0, 1, or 2 pencils.
- If you buy 2 pens, you cannot buy any pencils.

The total number of ways to buy pens and pencils is 5 + 3 + 1 = 9.

Example 2:
Input: total = 5, cost1 = 10, cost2 = 10
Output: 1
Explanation: The price of both pens and pencils are 10, which cost more than total, so you cannot buy any writing utensils. Therefore, there is only 1 way: buy 0 pens and 0 pencils.

class Solution(object):
def waysToBuyPensPencils(self, total, cost1, cost2):

---

9. You are given a 0-indexed array of positive integers *tasks*, representing tasks that need to be completed in order, where *tasks[i]* represents the type of the *ith* task.

You are also given a positive integer *space*, which represents the minimum number of days that must pass after the completion of a task before another task of the same type can be performed.

Each day, until all tasks have been completed, you must either:
- Complete the next task from *tasks*, or
- Take a break.

Return the minimum number of days needed to complete all tasks. **(10 points)**

Example 1:
Input: tasks = [1,2,1,2,3,1], space = 3
Output: 9
Explanation:
One way to complete all tasks in 9 days is as follows:
Day 1: Complete the 0th task.

Day 2: Complete the 1st task.
Day 3: Take a break.
Day 4: Take a break.
Day 5: Complete the 2nd task.
Day 6: Complete the 3rd task.
Day 7: Take a break.
Day 8: Complete the 4th task.
Day 9: Complete the 5th task.
It can be shown that the tasks cannot be completed in less than 9 days.

Example 2:
Input: tasks = [5,8,8,5], space = 2
Output: 6
Explanation:
One way to complete all tasks in 6 days is as follows:
Day 1: Complete the 0th task.
Day 2: Complete the 1st task.
Day 3: Take a break.
Day 4: Take a break.
Day 5: Complete the 2nd task.
Day 6: Complete the 3rd task.
It can be shown that the tasks cannot be completed in less than 6 days.

class Solution(object):
def taskSchedulerII(self, tasks, space):

---

*Hard level

10. There are *n* children standing in a line. Each child is assigned a rating value given in the integer array *ratings*.
You are giving candies to these children subjected to the following requirements:
- Each child must have at least one candy.
- Children with a higher rating get more candies than their neighbors.

Return the minimum number of candies you need to have to distribute the candies to the children. **(15 points)**

Example 1:
Input: ratings = [1,0,2]
Output: 5
Explanation: You can allocate to the first, second and third child with 2, 1, 2 candies respectively.

Example 2:
Input: ratings = [1,2,2]

Output: 4
Explanation: You can allocate to the first, second and third child with 1, 2, 1 candies respectively.
The third child gets 1 candy because it satisfies the above two conditions.

```
class Solution(object):
def candy(self, ratings):
```