Instructions for students:

1. Complete the following problem using concepts of **RECURSION**
2. You may use any language to complete the tasks.
3. You need to submit one single file containing all the methods/functions. You will get one week to complete your lab. **NO LATE SUBMISSIONS WILL BE CONSIDERED**.
4. The submission format MUST be maintained. You need to copy paste all your codes in ONE SINGLE .ipynb file and upload that. If format is not maintained, whole lab submission will be canceled.
5. If you are using JAVA, you must include the main method as well which should test your other methods and print the outputs according to the tasks.
6. If you are using PYTHON, make sure your code has the methods invoked and proper printing statements according to the tasks.
7. NO USE OF LOOP OR ANY OTHER BUILT IN FUNCTION UNTIL EXPLICITLY MENTIONED
8. You may use more than one method to solve these problems.
9. You may use default parameters if required.

Advantage of recursion than loop:

- time complexity same

Space complexity disadvantage for recursion
So why do we still use recursion? To make more manageable subproblems, breaking down large portions of code

If 1 loop then 1 recursive function, If 2 loop then 2 recursive function

Assignment Tasks: [CO4]

1. [Very Easy] [5 Marks each]

a) Implement a recursive algorithm that takes an array and **adds all the even positive elements** of the array.

b) Implement a recursive algorithm to **multiply all the odd positive elements** of a non-dummy headed singly linked linear list. Only the head of the list will be given as a parameter where you may assume every node can contain only integers as its element.
Note: you'll need a Singly Node class for this code.

c) The total number of combinations of size r from a set of size n [where r is less than or equal to n], we use the combination nCr where the formula is:

C(n,r)=n!/r!(n-r)!

Given n and r [r<=n], write a recursive function to find out nCr.

Apart from finding the factorial **recursively**, there is another recursive formula:

C(n,r) = C(n – 1, r – 1) + C(n – 1, r)

Where if r == 0 or n == r, C(n,r) = 1

You can solve the task using any formula you want.

d) Count the number of digits in a given number.

e) Given a number, check whether the number is prime or not.

f) Implement a recursive algorithm which will print all the elements of a non-dummy headed singly linked linear list in reversed order.
Example: if the linked list contains 10, 20, 30 and 40, the method will print
40
30
20
10

Note: you'll need a Singly Node class for this code.

2. [Easy] [5 Marks each]

a) Implement a recursive algorithm that takes a decimal number *n* as an input and converts *n* to its corresponding (you may return as a string) binary number.

b) Implement a recursive algorithm that takes a decimal number *n* as an input and converts *n* to its corresponding (you may return as a string) hexadecimal number.

c) Write a Recursive function that takes a **singly linked list head** as a parameter and prints all the vowels in the given string in the **same** order as they appear and then all the consonants in the **reverse** order. You can assume that the string will only consist of alphabets (a-z, A-Z), i.e. no space, special character, or numeric character will be in the string.

d) Adam wants to calculate the **alternating harmonic sum** of any positive integer by using recursion. But he does not know how to write recursive code. He just knows that alternating harmonic sum is the alternating sum of reciprocals of the positive integers. The formula is given below:

$$\text{sum(n)} = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \ldots\ldots + \frac{(-1)^{n+1}}{n}$$

So, you have a task to help Adam by writing a recursive function to calculate the alternating harmonic sum of any positive integer, n.

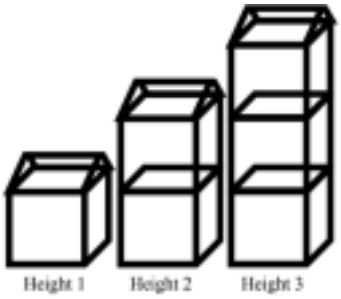
Sample Input and Output:

sum(3) = 1-(1⁄2)+(1⁄3) = 0.8333333333333333

sum(4) = 1-(1⁄2)+(1⁄3)-(1⁄4) = 0.5833333333333333

3. [Medium] [8 Marks Each]

a) Suppose, you have been given a non-negative integer which is the height of a ‘house of cards’. To build such a house you at-least require 8 cards. To increase the level (or height) of that house, you would require four sides and a base for each level. Therefore, for the top level, you would require 8 cards and for each of the rest of the levels below you would require 5 extra cards. If you were asked to build level one only, you would require just 8 cards. Of course, the input can be zero; in that case, you do not build a house at all. Implement a recursive algorithm to calculate the number of cards required to build a ‘house of cards’ of specific height given by the parameter.



b) Suppose, you are playing a game of cards where at first, you will be given **n** number of cards.

The goal of the game is to end up with EXACTLY 1 card. You must follow the rules below(where n is number of cards you have):
1. If n is even, then you may give back half of the cards you have.
2. If n is odd, your card number will be multiplied by 3. But, with that an extra card will also be added.
Write a **recursive function to find out the number of steps** you need to reach the goal of the game where you will be given n number of cards as an input.

For example, suppose you start with 21 cards.
– 21 is odd, so it will be multiplied by 3, and 1 card will be added, so n becomes 64.
– 64 is even, so you may return half of the cards, leaving you with 32 cards.
– 32 is even, so you may again return half of the cards, leaving you with 16 cards.
– 16 is even, so you may again return half of the cards, leaving you with 8 cards.
– 8 is even, you can return half of the cards, leaving you with 4 cards.
– 4 is even, again return half of the cards, you will now have 2 cards.
– 2 is even, you can return half of the cards, leaving you with 1 card.

So You have reached the goal of the game in 7 steps.

4. [Hard] [10 Marks Each]

a. Print the following pattern for the given input (you must use **recursion**):
We basically first reduce 5 one by one until we reach a negative or 0.
After we reach 0 or negative, we then add 5 until we reach n.

Sample Input-1: **16**

Sample Output-1:

16	11	6	1	-4	1	6	11	16
	11	6	1	-4	1	6	11	
		6	1	-4	1	6		
			1	-4	1			

Sample Input-2: **10**

Sample Output-2:

10	5	0	5	10
	5	0	5	

b. Your mom wants to see your midterm and final exam scripts. You are planning to show her the scripts where you got better marks first then gradually to the ones where you couldn't perform well. **Since you have the scripts of both midterm and final separately arranged in ascending order, you want to merge them in descending order**. Here, the lists represent the marks of the midterm and final exams respectively.

You must solve the problem using RECURSION.

You can not use any loop in the function. You **can use** built-in functions like append or concatenate the list to make a merged list.

Example 1:

Input Lists:

Midterm exam marks list: [5, 7, 14, 20, 24]

Final exam marks list: [10, 12, 25]

Output List:

Merged List: [25, 24, 20, 14, 12, 10, 7, 5]

Example 2:

Input Lists:

Midterm exam marks list: [11, 20, 24, 28]

Final exam marks list: [10, 12]

Output List:

Merged List:[28, 24, 20, 12, 11, 10]

5. [Very Hard] [12 Marks]

Complete the recursive function **flattenList()** which will take a nested python list and an empty list as parameters. The function should convert the nested list into a new flat list sequentially. Check the input-output to understand the sequence pattern.

```
def flattenList(given_list, output_list):
        # To Do

given_list = [1, [2, [3, [4], 5], 6], 7, 8, [9, [[10, 11], 12], 13], 14, [15, [16, [17]]]]
output_list = flattenList(given_list, []) # Initial empty list is sent for update
print(output_list)
#output_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]
```

Your code must work for any depth of the nested array. You can not use any loop in the function. You **can use** built-in functions like append, type, etc.

6. Bonus Task:

You went on a trip to see the great wall of china with your family. You want to take **one, two, or three steps at a time** as you go up the staircase. Out of curiosity you want to know the number of ways you have to reach the top of the great wall of china. Let's assume you start this game at any point in the staircase of the Great wall of china. However to make your life easier you know the number of steps that are remaining to reach the top which are marked on each stairs.

Write a recursive algorithm which can help you solve the mystery of finding the number of ways to reach to the top.

Input: The number of steps remaining to reach the top

Output: The number of ways you can climb the stairs

Example 1:

Input: 3

Output: 4

Explanation: There are n=3 steps left so it can be achieved in the following 4 ways:

1 -> 1 -> 1

1 -> 2

2 -> 1

3

Therefore, you can climb the remaining 3 steps using 4 possible ways.

Example 2:

Input: 5

Output: 13

Explanation: For n=5 steps remaining there are the following 13 ways for you to climb to the top.

1 -> 1 -> 1 -> 1 -> 1

1 -> 1 -> 1 -> 2

1 -> 1 -> 2 -> 1

1 -> 2 -> 1 -> 1

2 -> 1 -> 1 -> 1

1 -> 2 -> 2

2 -> 2 -> 1

2 -> 1 -> 2

1 -> 1 -> 3

1 -> 3 -> 1

3 -> 1 -> 1

2 -> 3

3 -> 2