APR 2024

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

Recursion

'24 | 10th Week | 069-297

MARCH 09 SATURDAY

```
# def fun1():
    Print ('fun1() called')

def fun2():
    Print ('Before fun1()')
    fun1()
    Print ('After fun1()')
Print ('Before fun2()')
fun2()
Print ('After fun2()')
```

o/p —
Before fun2()
Before fun1()
fun1() called
After fun1()
After fun2()

Many algorithms technique are based on Recursion.

Application
→ Dynamic Programming eg fibonacci series, factorial
→ Back tracking
→ Divide & conquer (Binary search, Quick sort and merge sort)

Many problem inherently recursive —
→ tower of hanoii
→ DFS based traversal (DFS of Graph and Inorder/Preorder/Postorder traversal of tree)

factorial

```
def f(n):
    if (n==0):
        return 1
    return n*f(n-1)
```

fibonacci

```
def fib(n):
    if n in [0,1]
        return n
    else:
        return f(n-1) + f(n-2)
```

→ Recursion = Terminates when the base case becomes true.
↳ Every recursive call needs extra spaces in stack memory.
Iteration = Terminates when the conditions becomes false.
↳ Every iterations doesn't require any extra spaces.

MARCH
10
SUNDAY
'24
11th Week
070-296

↳ Function which calls itself again & again until a specific condition met.

eg. Inception movie example of recursion

→ Recursion works on the principal of mathematical Induction.
→ Performing the same operation multiple times with different inputs.
→ In every step we try smaller input to make the problem smaller.
→ Base conditions is needed to stop the recursion, otherwise infinite loop will occur.
→ Recursive thinking a really important in programming and it helps you break down ~~to~~ problem into smaller ones and easier to use.

	Iteration	Recursion
Time efficient	✓	x
Space efficient	✓	x
easy to code	x	✓

There are two types of recursion:-

① Finite Recursion
Recursive method is going to execute finite times, it stops automatically once if conditions is satisfied such type of recursion is called finite recursion.

② Infinite Recursion
Recursive method is going to execute infinite times, never ending process but Python terminates this kind of recursion automatically if stack got overflow.

Base Conditions
Inside a recursion to stop recursion process at a finite No. of steps, we can use base conditions. We have to take this base conditions as first statement inside recursion call, if the condition is satisfied it terminates else continue the execution flow.

eg
```
def fun(n):
    if n==0: base conditions
        return
    Print('Good morning')
    f(n-1)
```

APR 2024

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

→ Infinite recursion may leads to running out of stack memory. And result in stack overflow.

'24 | 11th Week 071-295 | MARCH 11 MONDAY

Based on function calls/ method calls we have two types of recursion.

① Direct recursion

```
def fun():
    =
    =
    fun()
```

② Indirect recursion.

```
def fun():          def f2():
    =                   =
    =                   =
    def f2()            def f1()
```

When we go for recursion?

→ If the problem is very big and we are unable to solve the Problem then we should go for recursion.

Based on Position of recursive statement again it is divided into two types.

① Tail recursive (Perfect recursion)
A recursive fnction is called as tail recursion if the function doesn't contain any statement after recursive function calls.

② Non tail recursion (back tracking)
A recursive function is called as Non-tail recursion if the function contain statement after recursive function calls.

eg

```
def fun(n):
    if n==0:
        return
    print(n)
    fun(n-1)
```

o/p - 4 3 2 1

```
def fun(n):
    if (n==0):
        return
    fun(n-1)
    print(n)
```

fun(4)
o/p - 1 2 3 4

eg Quick Sort
Post order tree traversal
Post order is faster than other two traversal.

# To print on same line use end operator.
print(n, end=" ");

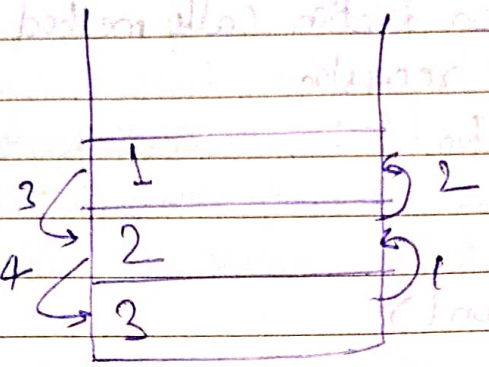
MARCH
12
'24
11th Week
072-294
TUESDAY

S	M	T	W	T	F	S
31					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

MAR 2024

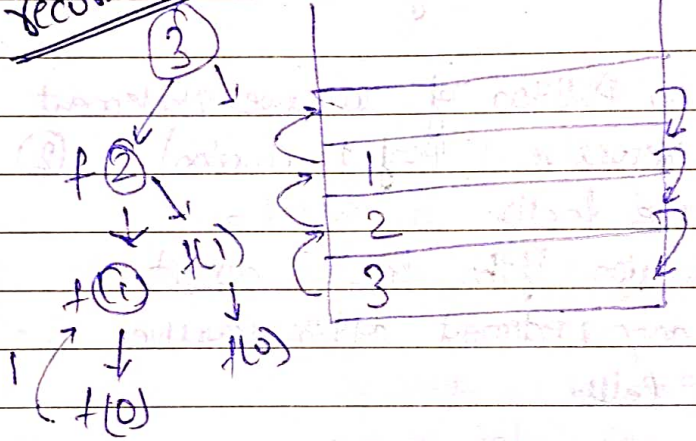
eg def fun(n):
08 if n==0:
return
09 print(n)
fun(n-1)
10 print(n)

fun(3)
11 o/p - 3,2,1,1,2,3



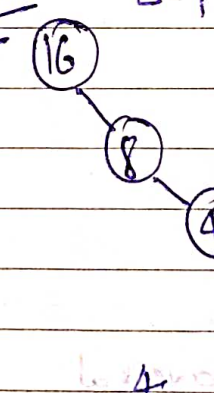
eg2 def fun(n):
if n==0:
01 return
fun(n-1)
02 print(n)
fun(n-1)
03 fun(3)
o/p - 1,2,1,3,1,2,1
04

Recursive Tree



eg def fun(n):
05 if n<=1:
return 0
06 else:
return 1+fun(n//2)
07 fun(16)
o/p - 4

Recursive Tree fun(16) 4
↳ 1+fun(8) 3
↳ 1+fun(4) 2
↳ 1+fun(2)
↓ 1
1+fun(1)
↓ 0

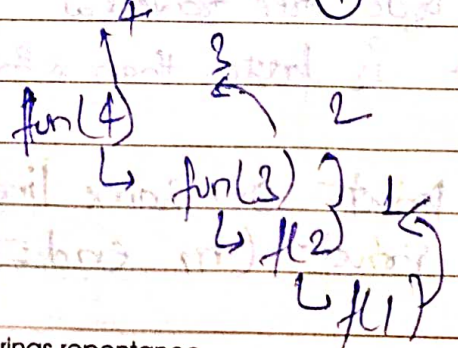


eg Print 1 to N using recursion.

def fun(n):
if n==0:
return
fun(n-1)
Print(n)

fun(4)

if n>0:
fun(n-1)
Print(n)



→ हमें Base case के बारे में सोचना है। बाकि के खुद Print हो जाएगी।

APR 2024 | S M T W T F S | 1 2 3 4 5 6 / 7 8 9 10 11 12 13 / 14 15 16 17 18 19 20 / 21 22 23 24 25 26 27 / 28 29 30

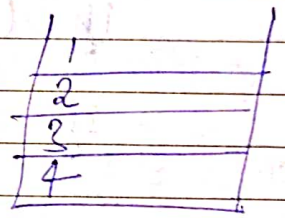
MARCH '24 | 11th Week 073-293 | 13 WEDNESDAY

eg
```
def fun(n):
    if n==0:
        return
    fun(n//2)
    Print(n%2)
fun(13)
o/p- 1101
```

fun(13)
- → fun(6)
  - → fun(3)
    - → fun(1)
      - → fun(0)
      - → Print(1%2)
    - → Print(3%2)
  - → Print(6%2)
- → Print(13%2)

eg Print N to 1 using recursion.
```
def fun(n):
    if n==0:
        return
    Print(n)
    fun(n-1)
fun(4)
```

f(4)
- → Pf(4)
- → f(2)
  - → Pf(3)
  - → f(2)
    - → Pf(2)
    - → f(1)
      - → Pf(1)
      - → f(1) ✓



Sum of digit using recursion
```
def fun(n):
    if n<10:
        return n
    return fun(n//10) + n%10
```
→ fun(253)
- → fun(25) + 3
- → fun(2) + 5

Palindrome.
```
def is_palindrome(str, start, end)
    if start >= end:
        return True
    return (str[start] = str[end] and is_palindrome(str, start+1, end-1))
```

→ Recursive Algorithms have two types of cases, recursive cases & base cases.

MARCH 14 THURSDAY

→ Every recursive function case must Terminate at base case.

→ Any problem that can be Solved recursively can also be Solved Iteratively.

11th Week 074-292

Sum of natural NOs using Recursion:—

```
def fun(n):
    if n <= 1:
        return n
    else: return fun(n-1) + n
```

Sum(1) = 1
Sum(2) = 2 + Sum(1)
Sum(3) = 3 + Sum(2)
Sum(4) = 4 + Sum(3)

Palindrome using Recursion:—

```
def isPal(st, s, e):
    if (s == e):
        return True
    if (st[s] != st[e]):
        return False
    if (s < e + 1):
        return isPal(st, s+1, e-1)
    return True
```

```
def isPali(st):
    n = len(st)
    if (n == 0):
        return True
    return isPali(st, 0, n-1)
```

fibonacci Sequence:—

```
def fib(n):
    if n in [0,1]:
        return n
    else:
        return fib(n-1) + fib(n-2)
```

Print Even Numbers:—

```
def Even(n):
    if n <= 0:
        return
    if n % 2 == 0:
        Even(n-2)
        Print(n)
    else:
        # even(n-1)
```

Even वाला 1 के लिए decrease कर रहे

factorial:

```
def fac(n):
    if n in [0,1]
        return 1
    return fac(n-1) * n
```

```
n = input()
a = 1
for i in range(2, n+1):
    a = a * i
return(a)
```

Sympathy is the key that fits the lock of any heart

APR 2024

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

'24 | 11th Week | 075-291

MARCH 15 FRIDAY

Tower of Hanoi

Tower of Hanoi is a mathematical puzzle where we have three rods (A, B, C) and N disks, initially, all the disks are stacked in decreasing value of diameter, i.e. smallest disk is placed on the top and they are on rod A. The objective of the puzzle is to move the entire stack of another rod (here considered C), obeying the following simple rules.

→ Only one disk can be moved at a time.

→ Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack. i.e a disk can only be moved if it is the uppermost disk on a stack.

→ No disk may be placed on top of a smaller disk.

Source to Auxiliary.

idea:
- shift 'N-1' disks from 'A' to 'B', using C.
- shift last disks from 'A' to 'C'. [Source to Destination]
- shift 'N-1' disks from 'B' to 'C', using A. [Aux to Dest]

A = Source
B = Auxiliary
C = Destination

Source destination helper

```
def TOH(n, from_rod, to_rod, aux_rod):
    if n==1:
        return # Print("Move 1 from", from_rod, "to", aux_rod)
    TOH(n-1, from_rod, aux_rod, to_rod)
    Print("Move disk", n, "from rod", from_rod, "to rod", to_rod)
    TOH(n-1, aux_rod, to_rod, from_rod).
```

↳ याद करने का तरीका

F	T	A
F	A	T
A	T	F

A	B	C
A	C	B
B	A	C

No. of Movement = $2^n - 1$

TC — $O(2^n)$
SC — $O(n)$

Rope cutting Problem.

Given a Rod of length N meters, and the rod can be cut in only 3 sizes A, B and C. the task is to maximize the no. of cuts in rod.

If it is impossible to make cut then print -1.

$0 < a, b, c \leq n$.

i/p = n = 5, a = 2, b = 5, c = 1

o/p = 5 {we make 5 piece of length 1 each}.

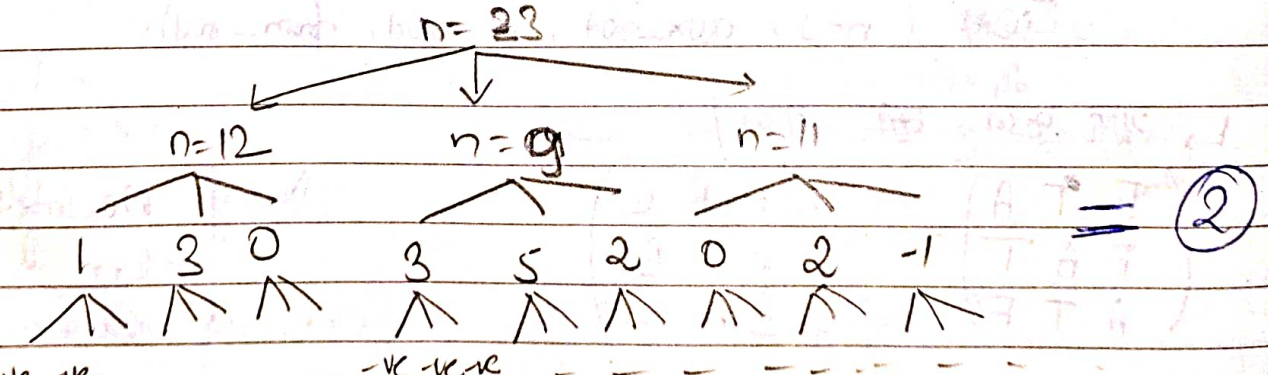
Length of every piece (after cuts) should be in set {a, b, c}

i/p = n = 23, a = 12, b = 9, c = 11

o/p = 2 { make 2 piece of length 11 & 12}

```
def maxpiece(n, a, b, c):
    if n == 0:
        return 0
    if n <= -1:
        return -1
    res = max(maxpiece(n-a, a, b, c),
              maxpiece(n-b, a, b, c),
              maxpiece(n-c, a, b, c))
    if res == -1:
        return -1
    return res + 1
```

TC = $\Theta(3^n)$



Subsequence.

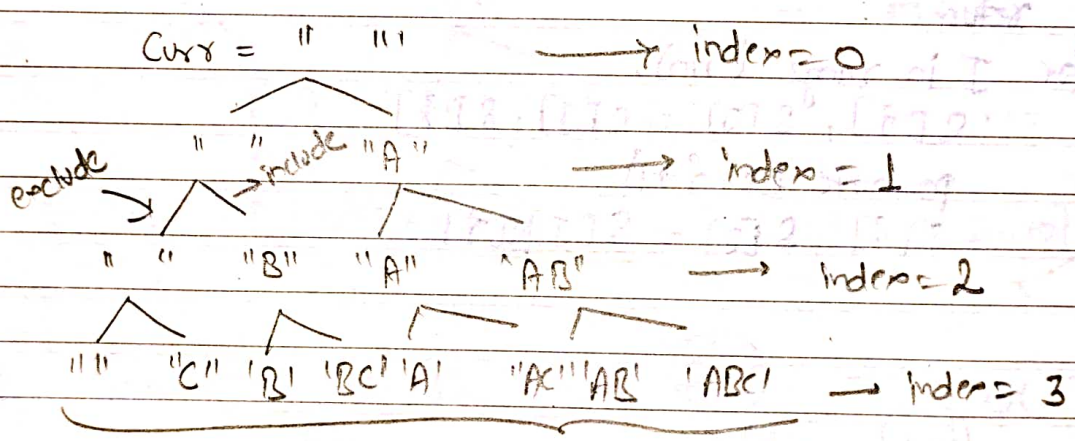
Subset of a given string (any order)

for string Subset ≈ Subsequence.

Given a string, we have to find out all subsequence of it. A string is a subsequence of a given string, that is generated by deleting some character of a given string w/o changing its order.

eg
i/p = "AB"
o/p = "", 'A', 'B', 'AB'.

i/p = 'ABC'
o/p = 'A', '', 'B', 'C', 'AB', 'AC', 'BC', 'ABC'.




```
def Sub (Str, curr, ind):
    if ind == len(Str):
        print (curr, end = ' ')
        return
    Sub ( Str, curr, ind + 1)
    Sub ( Str, curr + Str[ind], ind + 1)
```

TC = O(2^n)
SC = O(n)

MARCH
18
'24
12th Week
078-288
MONDAY

S	M	T	W	T	F	S
31					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

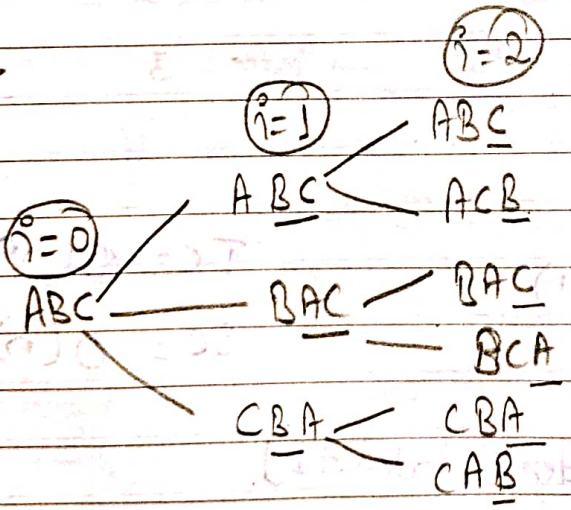
MAR 2024

Printing all permutations.

A permutation also called "arrangement number" or "order" is a re-arrangement of the elements of an ordered list.
S into a one-to-one correspondence with S itself.
A string of length n has n! permutations.

```
def permute(s, i):
    n = len(s)
    if (i == n-1):
        print(''.join(s))
        return
    for j in range(i, n):
        s[i], s[j] = s[j], s[i]
        permute(s, i+1)
        s[i], s[j] = s[j], s[i]
```

TC = O(n × n!)
SC = O(n − 1)

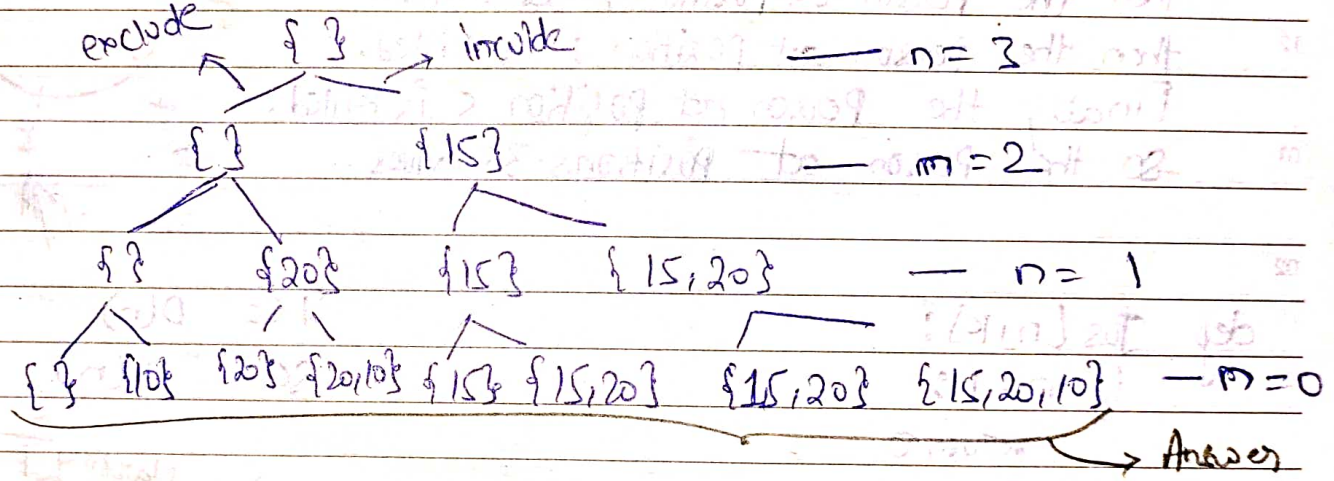


Subset Sum Problem

i/p = {10, 5, 2, 3, 6}
Sum = 8
o/p = 2.

i/p = {10, 20, 15}
Sum = 0
o/p = 1.

Subset of {1,2,3} are { }, {1}, {2}, {3}, {1,2}, {1,3}, {2,3}, {1,2,3}




```
def Subset(arr, n, sum):
    if (n==0)
        return 1 if sum==0 else 0
    return Subset(arr, n-1, sum) + Subset(arr, n-1, sum-arr[n])
```

TC = $\Theta(2^n) + \Theta(n) = \Theta(n)$

MARCH
20 '24
WEDNESDAY
12th Week
080-286

S	M	T	W	T	F	S
31					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

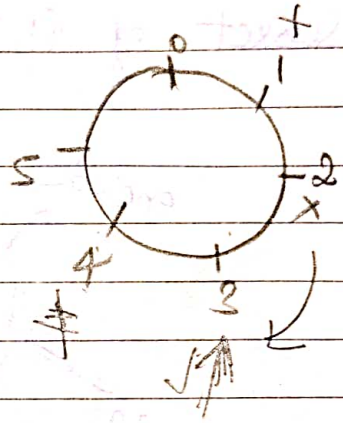
MAR 2024

Josephus Problem in Python.

Given the total number of Person N and a number K which indicates that K-1 persons are skipped and the Kth person is killed in a circle. The task is to choose the person in the initial circle that survives.

eg i/p = N = 5 and K = 2 o/p = 3

Firstly, the person at position 2 is killed.
then the person at position 4 is killed.
then the person at position 1 is killed.
Finally the Person at position 5 is killed.
So the Person at positions 3 survives.




```
def Jos(n,k):
    if n==1:
        return 0
    else:
        return (Jos(n-1,k)+k)%n
```

TC = O(n)
SC = O(n)

again starting से गिनने के लिए किया है।

```
def posBeginOne(n,k):
    return Jos(n,k)+1
```

eg Jos(5,3)
(Jos(4,3)+3)%5
((Jos(3,3)+3)%4+3)%5
(((Jos(2,3)+3)%3+3)%4+3)%5
((((Jos(1,3)+3)+3)%2+3)%3+3)%4+3)%5
0
= (3) ✓

If counting begins with (1)