DSA

28/8/25

Recursion

☆ **Recursion -**

A function calling itself again & again till it meets it's base condition.

- **Function call -**

```
def add(x,y):
    return x+y

def mul(x,y):
    return x*y

def sub(x,y):
    return x-y

def print_result(x):
    print(x)

def main():
    x=10
    y=20
    print_result(sub(mul(add(x,y),30),75))
```

print_result()
x=825
sub()
x=900 y=75
return 900-75
mul()
x=30 y=30
return 30*30
add()
x=10, y=20
return 10+20
main()
x=10, y=20
add 30
mul 900
sub 825
print_result

30

900

825

- **Recursion:**
  Function calling itself

Sum of first 5 natural numbers:

Sum(5) = 1+2+3+4+5
Sum(4) = 1+2+3+4
Sum(3) = 1+2+3
Sum(2) = 1+2
Sum(1) = 1

Sum(5) = Sum(4) + 5
↑ main problem
↑ just smaller sub problem

Sum(n) = 1+2+3+....+n-1+n

Relation:
Sum(n) = Sum(n-1)+n

- **Steps to Recursive Code:**

1. Assumption: Decide what your function does & assume it works.
2. Main logic: Solve the problem with the help of the sub problem.
3. Base condition: Condition with smallest problem (stop)

★ Problem: Sum of first n natural numbers.

1. Assumption: Assume "sum(n)" → sum of first 'n' natural numbers
2. Main logic: sum(n) = sum(n-1) + n
   ↑ problem ↑ just smaller sub problem
3. Base condition: smallest problem is n=1
   ∴ sum(1) = 1

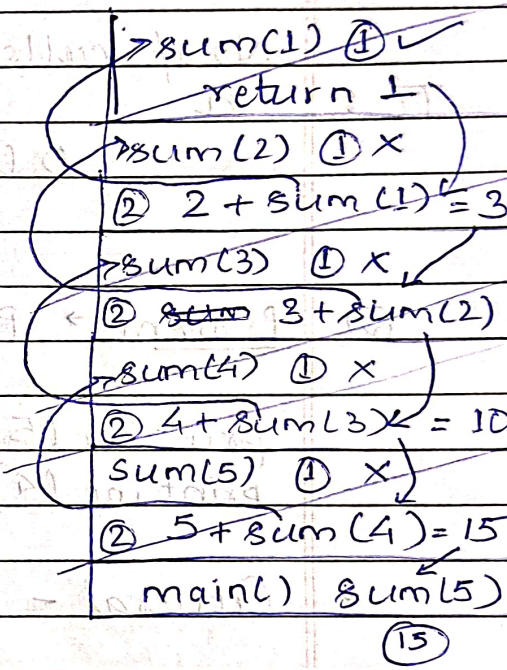
```
if n == 1:
    return 1
```

★ Code & Dry Run: sum(5)

```
def sum(N):
    # Base case
    if N == 1:
        return 1

    # Main logic
    return N + sum(N-1)
```

15 ← 5 + 4 + 3 + 2 + 1



- Base condition: Recursion will not stop without base condition.

1. TLE: No. of iterations allowed → 10^8 → 1 sec
2. MLE: (Memory Limit Exceeded):
   In recursion → stack overflow
   If we create approximately more than 10^5 levels, it will throw stack overflow.

Q) Problem → Given value of N, Calculate N!.

```
def fact(N):
    # Base condition
    if N==1 or n=0:
        return 1

    # Main Logic
    return N*fact(N-1)

print(fact(4))
```

No. of calls	Time taken
n	* O(1)

TC: O(N)

No. of calls ↓ Space ↙

SC: n + O(1)

O(N)

Q) Problem → Print numbers in increasing order

printinc(5) → 1 2 3 4 5
printinc(4) → 1 2 3 4

- General - Main problem: printinc(n)
  Smaller problem: printinc(n-1)

```
def printinc (N):
    # Base Cond"
    if N==1:
        print (N, end=" ")
        return

    # Main Logic
    printinc (N-1)
    print (N, end=" ")
```

o/p → 1 2 3 4 5

TC : O(N) SC : O(N)

Q) Problem → Print numbers in decreasing order.

printdec (5) → 5 4 3 2 1

General : Main problem printdec (n)
Smaller problem printdec (n-1)

```
def printdec (N):
    if N==1:
        print (N, end=" ")
        return

    print (N, end=" ")
    printdec (N-1)
```

TC : O(N) SC : O(N)

✱ Time & Space Complexity of Recursion :

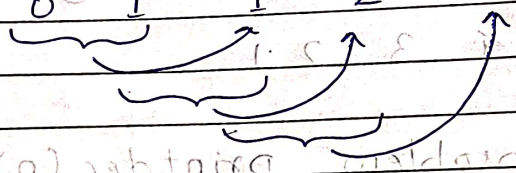
Total TC : No. of func^n call
$*$
Time taken by each func^n call

Total SC : Depth of the call stack
$*$
Space taken in each cell

✱ Nth number in the Fibonacci Series :

The fibonacci series is a series of numbers

N	0	1	2	3	4	5	6
	0	1	1	2	3	5	8



Fib(6) = Fib(5) + Fib(4)
= 5 + 3
= 8

Q) If N=7, find the Nth number in the fib series

1. Assume — Fib(N)
2. Main logic — Fib(N) = Fib(N-1) + Fib(N-2)
3. Base Cond^n — Fib(0) = 0, Fib(1) = 1
   if n == 0 or n == 1 :
   return n

```
def Fib(n):
    1.  if n==0 or n==1:
            return n

        return Fib(n-1) + Fib(n-2)
                  2         3
```

TC: level 1 : $1 \rightarrow 2^0$
2 : $2 \rightarrow 2^1$
3 : $4 \rightarrow 2^2$

GP: $a = 2^0$, $r = 2$, terms $= N$

Sum of GP $= \frac{a \times (r^n - 1)}{r - 1}$

$= \frac{1 \times (2^n - 1)}{1}$

$= 2^n$

2^n func^n calls
$O(1)$ in each call
TC: $O(2^n)$
SC: $O(N)$

Print the fibonacci from 1 to N

```
for i in range (1, n+1):
        print (Fib(i))
```