Recursion Concept for Data structure Algorithm

When func calls itself untill specified condition met

or we can say calling func itself with different parameters

psuedo code to understand the recursion logic

```
def func():
    print(1)
    func()
```

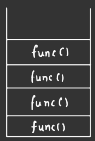
→ Concept of stack overflow

Maximum depth error → reached limit

No condition → given to stop execution of programm

Note

stack → used to store func calling for recursion and stack works on lifo concept



Execution of recursion code

Recursion requires base conditions

print name for n times

```
def func(n):
    if n==0:
        return []          # Base condition for termination
    result = func(n-1)     # Recursion on calling function with diff parameters
    result.append('Animuddha')
    return result
```

print linearly from 1 to n

```
func(x)
    if x==0:               # Base Condition
        return []
    result = []
    result.append(x)
    result.extend(func(x-1))   # Recursion
    return result
```

Sum of n numbers

```
def func :                          ⟶ function calling  func(5)

    if x==0                             if x==0 ⟶ false

        return 0                        result = func(5-1)
                                                  4
    result = func(x-1)                  1+2+3+4+5

    result += x                         Atlast ⟶ return summation of

    return result                       (15) ⟶ Output
```

factorial of n

Example ⟶ 5

5! x 4! x 3! x 2! x 1! ⟶ solve using recursion

120

Thing will remains same as above instead of addition ⟶ use multiplication

Reverse array

Array ⟶ 4 3 2 1 5

5 1 2 3 4 ⟶ Reverse of array

Approaching using for loop

```
↓          ↓↓          ↓
4    3     2     1     5

0    1     2     3     4   ⟶ indexing
```

swapping ⟶ 5 4

1 3

now both pointers at same index no more swapping

Code

```
def func(l,r)
    if l>=r                 # Base condition ⟶ l & r both overlap
        return
    arr[l], arr[r] = arr[r], arr[l]   # swapping
    func(l+1, r-1)
    return arr
result = func(0, n-1, arr)
return result
```

Check given str is pallindrome

Reverse of string ⟶ Exact same as original one

Example ⟶ Madam = madam

Both are same

Madam

left and right part should be identical for pallindrome string

Fibonacci Numbers Series

0 1 1 2 3 5 8 13

Addition of previous two element

Example ⟶ i need fibonacci of 5

f(5) = f(4) + f(3)

↳ previous two numbers

Basic for Approach

```
fib = [0,1]
for i in range(2, n):
    fib.append( fib(i-1) + fib(i-2))
return fib
```

Recursive approach

```
func(n)
    if n<=1:
        return n        # Base condition
    last = func(n-1)
    second = func(n-2)
    return last + second
```

```
func(4)
    if 4<=1   X
    last = func(4-1) = 3      2
    second = func(4-2) = 2    1
    return 2+1
                              3
0 1 1 2 (3)  ↳ output
```

Time Complexity ⟶ $O(2^n)$ Exponential

$\frac{2}{n}\ \frac{2}{n-1}\ \frac{2}{n-2}$ _ _ _