Recursion

- Is method to solve complex problem by breaking them down into simpler ones. Recursion is process by which a function calls itself directly or indirectly by applying same subroutines on parameters by keeping an extra space overhead (stack)

Basic eg factorial

$f(n) = n * f(n-1)$ → subroutine on parameter

eg $5! = 5 \times 4!$

fn call itself

$f(3)$ → $3 \times f(2)$ → $2 \times f(1)$

break problem in small parts.

$f(1) = 1! = 1$

- This recursion is related to principal of mathematical Induction.

Steps for recursion

1) Find out smallest subproblem for which we know the ans (base case)
2) Assume for given problem recursion will work correctly & calculate a subproblem.
3) Find how smaller subproblem contribute to final ans. (self work)
   like $f(n) = n \times f(n-1)$

eg 1 Find factorial

```
def fact(n):
    // base case
    if (n==1): return 1
    // recursive assumption
    sub = fact(n-1)
    // self work
    return n * sub
```

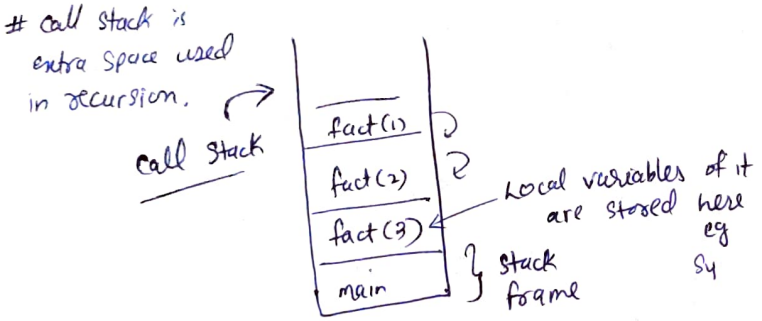
← can compress it

```
def fact(n):
    if (n==1): return 1
    else:
        return n * fact(n-1)
```

Q How memory is distributed for process ?

Stack (Linear)

call stack is extra space used in recursion.



Heap (big pool of data)

is not heap data structure

slower than stack

size is quite large

eg int *ptr = new int [10];

Whenever a function is called, its variables get memory allocated on stack. And whenever function call is over, its variables & memory is deallocated from stack.

A programmer does not have to worry about memory allocation & de-allocation of stack variables.

eg int a, int b[10], int c=10, int d[c] are stored in stack.

Whenever an object is created it's always stored in heap space & stack memory contains reference to it.

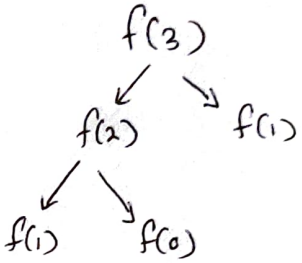
Basic Questions

Q1. fibonacci series , return nth value

0, 1, 1, 2, 3, 5 - - - - -
0th 1st 2nd term

Code

```
def fib(n):
    if (n==0 or n==1):        // base case
        return n
    else:
        return fib(n-1) + fib(n-2)

print (fib(6))
```



note
relations like fib(n) = fib(n-1) + fib(n-2)
this are called recurrence relation.

Q2 Print 1 to n natural number in normal & reverse order.

```
def sub(n):
    if (n==0):          # base case
        return
    else:
        sub(n-1)        # recursive assumption
        print(n)        # self work
```

sub(5) → 1 2 3 4 5

reverse order

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

sub(5) → 5 4 3 2 1

make call stack to see how its working.

Q3) Count number of ways to make binary string of length n where there is no adjacent 1.

For such questions there is usually a pattern.

Eg

n = 1 → 0, 1 (2)

n = 2 → 00, 01, 10 (3)

n = 3 → 000, 001, 010, 100, 101 (5)

n = 4 → 0000, 0001, 0010, 0100, 1000, 1001, 1010, 0101 (8)

n = 5 → 00000, 00001, 00010, 00100, 01000, 10000, 10100, 10101, 10010, 10001, 01010, 01001, 00101 (13)

So pattern here is 2, 3, 5, 8, 13 _ _

is fibonacci series

Soln

```
def fib (n):
    if (n==1):
        return 2
    if (n==2):
        return 3
    else:
        return fib(n-1) + fib(n-2)
```

Q4 There are N persons, who want to go to party. There is a constraint that any person can either go alone or can go in pair. Calculate no. of ways in which N persons will go to party.

Sol like ABC → A, B, C | AB, C | A, BC | AC, B

for N=3 4 ways

here concept of making recursive relation.

f(n) depends on 2 things

→ Let if A goes alone then f(n) = 1 * f(n-1) ways.

→ if A decides to go it pair so f(n) = (no. of ways A can make pair) * f(n-2)

↑ since n-2 persons are left

So f(n) = f(n-1) + (n-1) * f(n-2)

Base case

n = 1  ways = 1

n = 2  = 2

pair , alone

Code

```
def sub(n):
    if (n == 1 or n == 2):
        return n
    else:
        return sub(n-1) + (n-1)*sub(n-2)

print(sub(4))    # 10
```

Q5) Print pattern N = 4

Concept (handling & variables)

Code

```
def sub(n, x):
    if n<=0 :                    // base case
        return

    if x==n :
        print (" ")              // self work
        sub(n-1, 0)              // recursive call

    else :
        print ("*", end= " ")     // self work
        sub( n, x+1)             // recursive call.

sub(4,0)
```

Q6) Reverse a string

```
class Solution :
    def reverseString (self , s: List [str]) -> None :
        n = len(s)
        def sub (s , l, r):
            if ( l >=r):
                return
            else :
                s[l] , s[r] = s[r] , s[l]
                sub(s, l+1 , r-1)
                return s

        s= sub(s, 0, n-1)
```

Intermediate

1) Concept of output so far.
& take or not take cases.

Q1) Print all subsets of array.

eg ['a', 'b', 'c'] → [a, b, c], [a, b] [a, c], [a] [b, c] [b], [c], []

Code

```
def sub(a, i, n, s):                    # s = output so far.
    if i == n:                          // base case
        print(list(s))
    else:
        sub(a, i+1, n, s+a[i])          // self work
        sub(a, i+1, n, s)               // recursive assumption

# main
a = ['a', 'b', 'c']
sub(a, 0, len(a), "")
```

Q2) Print n length binary string not having any consecutive 1's.

eg n = 2 → 10, 01, 00

Code

```
def sub(i, n, s, t):
    if (i == n):
        print(s)
    else:
        if (t == 1):
            sub(i+1, n, '0'+s, 0)
        else:
            sub(i+1, n, '0'+s, 0)
            sub(i+1, n, '1'+s, 1)

sub(0, 3, "", 0)
```

s → output so far

t → keeps track of last digit

eg if last digit added is 0, we can add both 0 & 1 (t = 0)

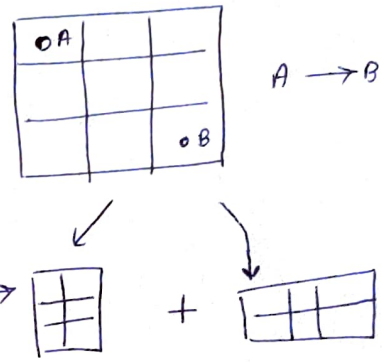
Concept

Recursion on Grid

(Q3) You are given a 2D grid, you are at top left point and you need to reach bottom right point. (can go only right or down)

1st) find total number of paths possible.

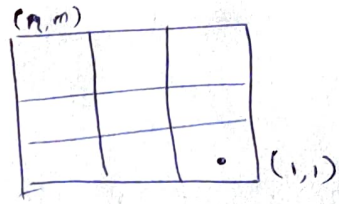
n = rows
m = columns




```
def sub(n, m):
    if (n==1 or m==1):
        return 1
    else:
        return sub(n,m-1) + sub(n-1, m)

print (sub(3,3))
```

2nd) Print all paths also

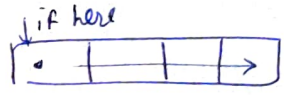
```
def sub(n, m, s, l):
    if (n==1):
        print (s + "R"*(l - len(s)))
        return(1)
    elif (m==1):
        print (s + "D"*(l - len(s)))
        return 1
    else:
        return sub(n, m-1, s+"R", l) +
               sub(n-1, m, s+"D", l)

n=4
m=5
print (sub(n, m, "", n-1+m-1)
```



can go Right & Down.

also see
Size of all paths
= (n-1) + (m-1)

also



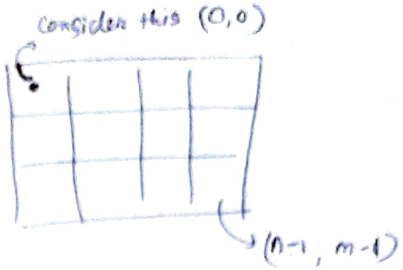
So only "RRR"

if

only "DDD"

3rd) Lets even allow Diagonal move

Using another method



if  So if i >=n return

Same for j >=m return.

```
def sub(i, j, n, m, s):
    if (i == n-1 and j == m-1):
        print(s)
        return 1
    elif (i >= n or j >= m):
        return 0
    else:
        return sub(i, j+1, n, m, s+"R") + sub(i+1, j, n, m, s+"B")
                + sub(i+1, j+1, n, m, s+"D")
```

// R → Right
// B → Down (bottom)
// D → Diagonal

```
n = 3
m = 3
print(sub(0, 0, n, m, ""))
```

// total 13 paths