Recursion

Example - WAP to calc. vol^m of cylinder.

Sol^n volume = area * height

```
main(){                                  // main calling volm funcn
    volume(radius, height);
    return 0;
}
```

main → volm → area (chaining)

```
float volume(float radius, float height) // volm calls area
{                                        // funcn.
    return height * area(radius);
}
float area(float radius){                // area funn returns
    return 3.14*radius*radius;           // area of circle to volm
}                                        // to calls volm.
```

⊙ function chaining going on here, but when a funcn calls itself, that particular process known as recursion.

Example

→ print n! using recursion

```
void printnum(n){                        if n=6 => 5 4 3 2 1.
if(n==0)
    return;          //this block won't
                     execute. as n=0.
print(n);            // here n=6, so 6 will be printed
printnum(n-1);
}                    // again call funcn with updated value as we
                     // need in using order
```

printnum(6) → printnum(5) → printnum(4) ... (1) (0)

wont print

here it'll check base condtn, & funcn terminat

And start returning back value

→ If there will be no base condit^{n}, func^{n} will keep on going with updated value. "Base condition" is use to terminate the recursive func^{n}. stack overflow

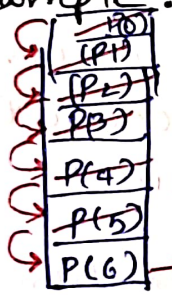
⊙ When to use recursion?

When we've to do same task on diff set of parameter, we use recursion

⊙ Recursion in Stack

In memory every time a method is called, a copy is created & this copy stored in stack.

Example: n! func^{n} (n=6)



when P(0), the execution finishes all values related to P(0) remove from stack. Similarly for all P(n).

⊙ When understanding recursive func^{n}, 2 ways (1) from stack memory (2) recursive tree.

Example: calculate factorial of a no. in recursion.

$n! = n*(n-1)*(n-2)*---(2)(1) \Rightarrow n*(n-1)!$

$5! = 5\times4\times3\times2\times1 \Rightarrow 5\times4!$

$4! = 4\times3\times2\times1 \Rightarrow 4\times3!$

$3! = 3\times2\times1 \Rightarrow 3\times2!$

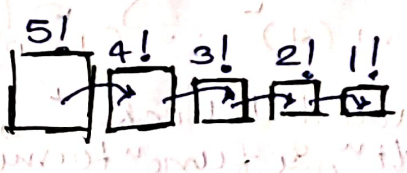
$2! = 2\times1! \Rightarrow 2\times1!$

$1! = 1$

$0! = 1$

Base condt^{n} ← where $n>1$, $n=0, n=1 \Rightarrow n!=1$

Recurrance Relat^{n}



Big problem solved with sub-problem... goes on, & only parameter is changing then we can be know, we should apply recursion

⊙ Few things to take care of

(1) Keep track of parameter. (eg. n in factorial func^{n})

(2) Keep track of what value is returned & to whom.
eg. (n x fact(n-1) → returning value to fact( )
and again used by fact(n-1).

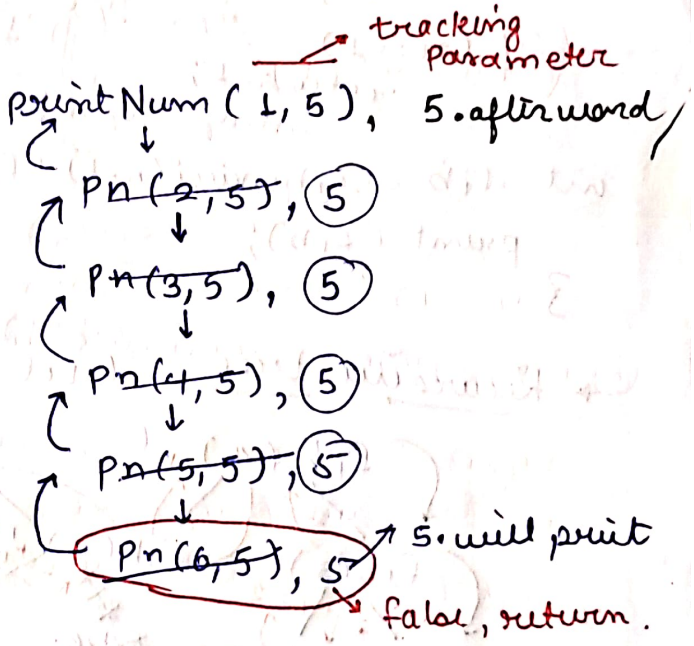
(3) from where the func^{n} will resume, after called func^{n} finishes execution.

Example :-

print pattern : 1, 2, 3, 4, 5, --- n, n, --- 5, 4, 3, 2, 1
increasing and decreasing

```
void printNum(n, num)
{
1.  if (n == num+1)
2.      return;
3.  print(n);
4.  printNum(n+1, num);
5.  print(n);
}
```

O/P → 1, 2, 3, 4, 5, 5, 4, 3, 2, 1.



Recursion And Iterator

→ every problem that can be solved with recur., can solved with iteration also, both have pros & cons.

→ for this you have to understand recursive tree approach

Example : fibonacci series using recursion

0 1 2 3 4 5 6
0 1 1 2 3 5 9 → any no. is sum of 2 prev. no.

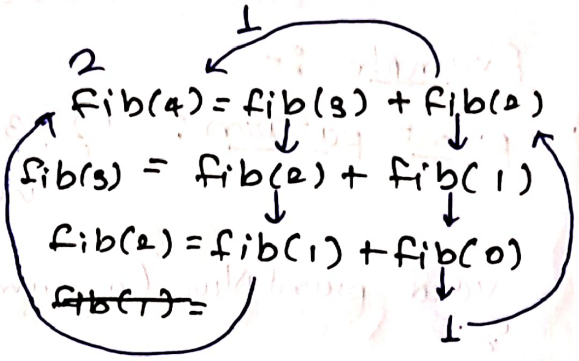
→ 1, 1+2, 2+3, 3+5, ---.

⇒ $f(n) = f(n-1) + f(n-2) \quad \forall\ n > 1$

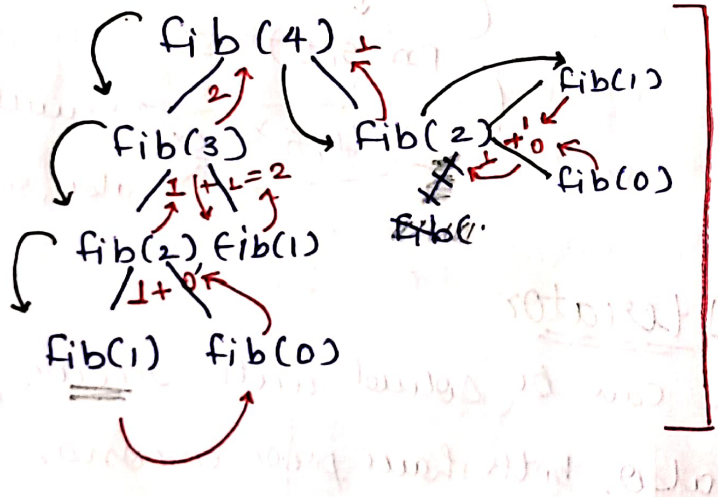
$f(n) = 2, \quad n = 1, 1$] base condt^n

$n = 0, 0$

```
int fibonacci(int n){
    if (n == 0)
        return 0;
    if (n == 1)
        return 1;
    int fib = fibonacci(n-1) + fibonacci(n-2);
    print (fib);
}
```



→ Recursive tree (let n = 4)



This is recursive approch for fibonacci series.

→ Iterative approch for fibonacci series

where n = 4.

```
int fibonacci(int n) {
  int first = 0;
  int second = 1;
  print(first); print(second);
for(int i=0+2; i<=n; i++)
{
   int next = first + second;
   Print(next);
    first = second;
    second = next;
  }
}
```

0 1 2 3 4 5 -

0 1 1 2 3 5 . . .

this is iterative approch

loop 2 to 4, con

$n_0 = 0$, $n_1 = 1$.

next = $n_0 + n_1$; next = 0+1;

then

$n_0 = n_1$; ⇒ first = 1, second = 1. ~~first = 1~~

n_1 = next;

Diff. b/w Iteration and recursion

→ each recur. call require extra space in memory.

→ Sol^n to some problem are easier to formulate recursively

→ test condt^n in loop and base condt^n in recursion.

→ if base condt^n absent, ~~infi~~ stack overflow error, and in iteration infinite loop.

How to solve any recursion problem :-

1. Recursion tree.
2. identify base condition
3. all variable, which you'll use
4. think how recursion will work on paper
5. practice recursion

Ques → 1 - Combination of factors

n is given, find out all factors of n, & print in lexographical order.

Example

n = 8

2	8
2	4
2	2
1	1
	0

→ 1, 2, 2, 2 (Ans)
1, 2, 4
1, 8

(print only non-repeated order).

⊛ Implementation (Recursive approch)

⊙ Most imp is to handle duplicate value. for this we can say that next value is smaller than current value we'll stop & recurr back.

1	12
2	12
3	6
4	2
	1

×

1	12
2	12
2	6
3	3
	1

✓

⊙ we'll take 3 variable n, str, p. :-

n = 12, str = "1 ", p = 2 (∵ 1 will always be in ans).

Base conditions :- (pseudocode)

```
void solve ( int n, int p , string str)
{
  if (n == 1)
      cout<<str<<endl;
  else {
    for( int i=p; i<=n; i++)
    {
      if (n%i == 0)
          solve (n/i , i, < prev. soln >);
    }
  }
}
```

Ques → Print the pattern (7)

Print a pattern using recursion. int N is given where $a_i, a_{i+1}, a_{i+2}, \dots N$, where if $a_i > 0$ then $a_{i+1} = a_i - 5$ else $a_{i+1} = a_i + 5$. till ↓se $a_i <= 0$ & ↑se till N.

✱ Example

N = 16 => 16, 11, 6, 1, -4, 1, 6, 11, 16.
(1, 11, 6, 1: -5; -4, 1, 6, 11, 16: +5)

✱ Intuition

- operations - print (n)
- base condt^n if (n <= 0)
- parameter (n, n-5)
- return type. for print.

✱ Pseudocode

```
void PrintNum( int n )
    if ( n <= 0 ) {
        return;
    }
    PrintNum (n-5);
    print (N);
    Print (n);
```

✱ Dry Run :-

n = 6

Print:

6, 1, -4, 1, 6

(1, 6: Happening because of return.)