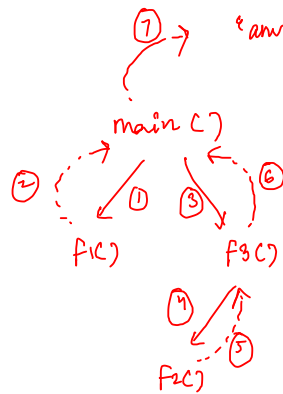
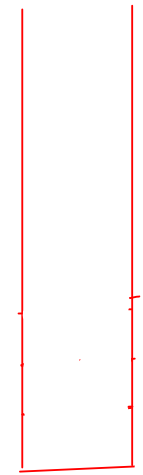
```
function f1() {
  1. console.log("Hello");
}
function f2() {
  1. console.log("world");
}
function f3() {
  1. f2();
  2. console.log("amrog");
}
function main() {
  1. f1();
  2. f3();
}
main();
```

op: "hello"
"world"
"amrog"



★ How function calling works? (Recap)



1. main is waiting for f1() to complete

stack

★ always function which is on top of stack will be running.

✱ Recursion : (vvvImp)

→ It is function calling itself until a condition is met.

```
function f() {
  1. console.log("Hello");
  2. f();
}

function main() {
  1. f();
}
main();
```

```
function f() {
  1. console.log("Hello");
  2. f();
}
function f() {
  1. console.log("Hello");
  2. f();
}
function f() {
  1. console.log("Hello");
  2. f();
}
function f() {
  1. console.log("Hello");
  2. f();
}
```

f·f

f·F·f

f·f·f·f

f·f·f·f·f

so on until infinite times

op: "hello"
"hello"
"hello"
"hello"
"hello"

✱ stackoverflow
(no space left in stack memory)

keep on growing

```
function F(cnt) {          // cnt = 0   (f)
  1. if (cnt == 3) {
  2.   return;
     }
  3. console.log("Hello");
  4. F(cnt + 1);
}

F(0);
```

```
function F(cnt) {          // cnt = 1   (f•f)
  1. if (cnt == 3) {
  2.   return;
     }
  3. console.log("Hello");
  4. F(cnt + 1);
}
```

```
function F(cnt) {          // cnt = 2   (f•f•f)
  1. if (cnt == 3) {
  2.   return;
     }
  3. console.log("Hello");
  4. F(cnt + 1);
}
```

```
function F(cnt) {          // cnt = 3   (f•f•f•f)
  1. if (cnt == 3) {
  2.   return;
     }
  3. console.log("Hello");
  4. F(cnt + 1);
}
```

op: "Hello"
"Hello"
"Hello"

Stack

★ Base Case,

⇒ termination condition in recursion

* How to solve problems using recursion ?

1. faith (vishwas , believe)
2. Logic (do some work, give rem. work to recursion) (* cannot give entire work to recursion)
3. Basecase
4. Recursive tree (function calls tree)

Q1 : print "ACCIOJOB" n times

```
function print (n) {
    if (n == 0) {
        return;
    }
    print (n-1) ;
    console.log ("ACCIOJOB");
}
```

1. print(n) → "ACCIOJOB" will be printed n times

* there is no code, nothing is there, but put a blind faith.

2. print(5) → "ACCIOJOB"
"ACCIOJOB"
"ACCIOJOB"
"ACCIOJOB"
"ACCIOJOB"

print(4) → "ACCIOJOB"
"ACCIOJOB"
"ACCIOJOB"
"ACCIOJOB"

3. think about smallest input possible
(n)
draw the recursive tree and decide

4. recursive tree

op: "AJ"
"AJ"
"AJ"
"AJ"
"AJ"

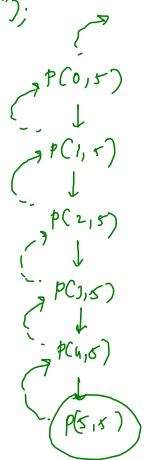
(You all printing "AJ after coming back)

another way :

$n \rightarrow n-1 \rightarrow n-2 \rightarrow \cdots \rightarrow 0$

```
function print (cnt, n) {
    if (cnt == n) {
        return;
    }
    console.log ("ACCIOJOB");
    print (cnt + 1, n);
}
```

op: "AJ"
"AJ"
"AJ"
"AJ"
"AJ"



cnt = 0
n = 5

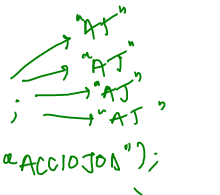
1. print (cnt, n)
   → It will print "n - cnt" times
   ⇒ It will print cnt → n times

2. print (0, 5)
   ⇒ "AJ"
   "AJ"
   "AJ"
   "AJ"
   "AJ"

print (1, 5) ⇒ "AJ"
"AJ"
"AJ"
"AJ"

★ pre, in, post Areas :

```
function print (n) {
    if (n == 0) {
        return;
    }
    print (n-1);
    console.log ("ACCIOJOB");
}
```



★ Area After rec. call is called post area

(post-order way of solving)

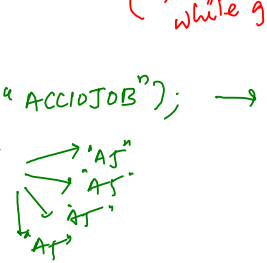
[Dynamic programming]

(getting the Answer)

★ Area in between recursion call is call in Area

```
function print (n) {
    if (n == 0) {
        return;
    }
    console.log ("ACCIOJOB");
    print (n-1);
}
```

("you are printing "AJ" while going down")



★ Area before recursion call is called pre Area

(pre-order way of solving)

(forming the answer)

(answer will be achieved at the Basecase)

* Why Recursion, why not Iteration :

① "a b c d e f" 6

⇒ n! ⇒ 6! ⇒ 720

6 * 5 * 4 * 3 * 2 * 1

print all permutations -

a b c d f e

a b c f d e

② solving sudoku (explore possibilities)

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

9:35 - 9:50 pm

Break

* Recursively print numbers : (1 to N)

ip: n = 5

op: 1 2 3 4 5

1. print(n) ⇒ will print all numbers from 1 to N increasingly.

2.
print(5) ⇒ 1 2 3 4 (5)

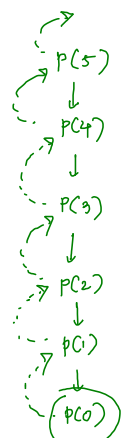
print(4) ⇒ 1 2 3 4 (4)

rec | you

3.
```
function print (n) {
    if(n == 0) {
        return;
    }
    print (n-1) // 1 2 3 4 (rec) 5 (you)
    process.stdout.write (n + " ");
}
```

(post-way)

4.



op: 1_2_3_4_5_

≠ another way : (pre-way)

1. print (curr_num, n)

⇒ It will print all numbers from curr_num → n

2. print (1, 5) ⇒ 1 2 3 4 5

print (2, 5) ⇒ 2 3 4 5

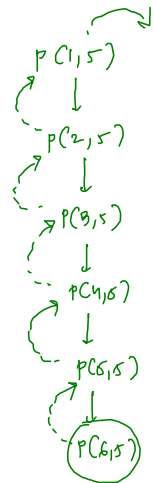
3. function print (curr_num, n) {

```
if ( curr_num > n ) {
    return;
}

process.stdout.write ( curr_num + " " );
print (curr_num + 1, n);   // 1 | 2 3 4 5
                           //   you  rec
}
```

op: 1 2 3 4 5

4.



* Reverse print Recursive :

ip: n = 5

op: 5 4 3 2 1

3.
```
function print(n) {
    if (n == 0) {
        return;
    }
    process.stdout.write(n + " ");
    print(n-1); //
}
```
5 | 4 3 2 1

you | rec

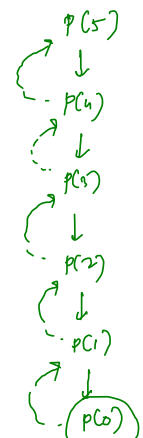
(pre-way)

1. print(n) ⇒ print all the numbers from n to 1 decreasingly.

2. print(5) ⇒ 5 4 3 2 1

print(4) ⇒ 4 3 2 1

4.



op: 5 4 3 2 1

another way (post-way):

```
function print(curr_num, n) {
    if (curr_num > n) {
        return;
    }
    print(curr_num + 1, n)
    process.stdout.write(curr_num);
}
```

rec: 5 4 3 2 | you: 1

1. print(curr_num, n)
   → will print all numbers from n → curr_num

2. print(1,5) ⇒ 5 4 3 2 1
   print(2,5) ⇒ 5 4 3 2

4. p(1,5) up: 5 4 3 2 1