Recursion

Recursion means function calling itself for different inputs or parameters.

Let's understand it step by step:

Example: Printing numbers from 1 to N recursively

```
var N=5;
var i=1;

function print1toN(i,N){
   if(i>N){
      return;
   }
   console.log(i);
   print1toN(i+1,N);
}
print1toN(i,N);
```

Output:

```
1
2
3
4
5
```

Visual Representation:

```
print1toN(1, 5) -> prints 1 -> calls print1toN(2, 5)
   print1toN(2, 5) -> prints 2 -> calls print1toN(3, 5)
      print1toN(3, 5) -> prints 3 -> calls print1toN(4, 5)
         print1toN(4, 5) -> prints 4 -> calls print1toN(5, 5)
            print1toN(5, 5) -> prints 5 -> calls print1toN(6, 5)
               print1toN(6, 5) -> i > N, so return
```

Steps involved:

1. **Initial Call:**
   - `print1toN(1, 5)` is called.
   - `i` (1) is not greater than `N` (5), so it proceeds, base case is not reached yet.
   - `console.log(1)` prints `1`.
   - The function calls itself recursively: `print1toN(2, 5)`.

2. **First Recursive Call:**
    - `print1toN(2, 5)` is called.
    - `i` (2) is not greater than `N` (5), so it proceeds, base case is not reached yet.
    - `console.log(2)` prints `2`.
    - The function calls itself recursively: `print1toN(3, 5)`.
3. **Second Recursive Call:**
    - `print1toN(3, 5)` is called.
    - `i` (3) is not greater than `N` (5), so it proceeds, base case is not reached yet.
    - `console.log(3)` prints `3`.
    - The function calls itself recursively: `print1toN(4, 5)`.
4. **Third Recursive Call:**
    - `print1toN(4, 5)` is called.
    - `i` (4) is not greater than `N` (5), so it proceeds, base case is not reached yet.
    - `console.log(4)` prints `4`.
    - The function calls itself recursively: `print1toN(5, 5)`.
5. **Fourth Recursive Call:**
    - `print1toN(5, 5)` is called.
    - `i` (5) is not greater than `N` (5), so it proceeds, base case is not reached yet.
    - `console.log(5)` prints `5`.
    - The function calls itself recursively: `print1toN(6, 5)`.
6. **Base Case (Termination):**
    - `print1toN(6, 5)` is called.
    - `i` (6) is greater than `N` (5), so the function hits the base case, base case is not reached yet.
    - The function returns without making further recursive calls, ending the recursion.

Another Example: Printing Numbers from N to 1

```
var N=5;
var i=N;

function print1toN(i,N){
   if(i<=0){
      return;
   }
   console.log(i);
   print1toN(i-1,N);
}
print1toN(i,N);
```

Output:

5
4
3
2
1

Types of Recursion:
1.Head Recursion
2.Tail recursion

Head recursion:
When recursion is made above all statements and logics.
 Example:
Suppose you are asked to print numbers from N to 1 but input i will start from 0
Simply, start from i=1 and print N to 1.

```
var N=5;
var i=1;

function print1toN(i,N){
    if(i>5){
        return;
    }

    print1toN(i+1,N);
    console.log(i);
}
print1toN(i,N);
```

Output:
5
4
3
2
1

In the above case as you can see the recursive call is made above the console.log statement.

Case-2:
Suppose you are asked to print numbers from 1 to N but input i will start from N
Simply, start from i=N and print 1 to N.

```
var N=5;
```

```
var i=N;

function print1toN(i,N){
    if(i<=0){
        return;
    }

    print1toN(i-1,N);
    console.log(i);
}
print1toN(i,N);
```

Output:
1
2
3
4
5

Tail recursion:
When recursion is made below all statements and logics.
Example:
Suppose you are asked to print numbers from 1 to N .

```
var N=5;
var i=1;

function print1toN(i,N){
    if(i>N){
        return;
    }
    console.log(i);
    print1toN(i+1,N);
}
print1toN(i,N);
```

Output:
1
2
3
4
5

In the above code,the recursive call is made below the console.log statement.

Practice Problems:

Factorial of a number:

```
var N=5;

function factorial(N){
   if(N==0){
      return 1;
   }

   return N*factorial(N-1);
}
console.log(factorial(N));
```

Output:
120

Nth Fibonacci Number:
Problem Link: https://leetcode.com/problems/fibonacci-number/

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

Nth Tribonacci Number:
Problem Link:https://leetcode.com/problems/n-th-tribonacci-number/

```
function tribonacciNumber(n){
    if(n==0){
        return 0;
    }
```

```
    if(n==1||n==2){
        return 1;
    }
    return tribonacciNumber(n-1)+tribonacciNumber(n-2)+tribonacciNumber(n-3);
}
var tribonacci = function(n) {
    return tribonacciNumber(n);
};
```

Calculate Power Linearly:
Problem: Given two variables x,n you have to calculate x^n.

```
var x=3;
var n=7;

function calculatePower(x,n){
   if(n==1){
      return x;
   }
   return x*calculatePower(x,n-1);
}
console.log(calculatePower(x,n));
```

Output:
2187

Calculate Power Logarithmically:
Problem: Given two variables x,n you have to calculate x^n.

```
var x=3;
var n=7;

function calculatePower(x,n){
   if(n==1){
      return x;
   }
   if(n==0){
      return 1;
   }
   var power=calculatePower(x,Math.floor(n/2));
   power*=power;
```

```
    if(n%2!=0){
        power*=x;
    }
    return power;
}
console.log(calculatePower(x,n));
```

Output:
2187

Generate all subsequences of a string:

```
var ansArray=[];
function getSubsequences(S,i,res){
    if(i==S.length){
        ansArray.push(res);
        return;
    }

    getSubsequences(S,i+1,res+S[i]);

    getSubsequences(S,i+1,res);

}
getSubsequences("abc",0,"");
console.log(ansArray);
```