* Function :

Function is a block of code that perform a specific task.

Function Definition (telling JS)

```
Function fun_name()
{
    // Do something
}
```

Function Call (Using the Function)

```
 fun_name();
```

i] e.g

```
function message()
{
    Console.log(" Rohit ");
}
message();
```

O/P : Rohit

* Function with Argument :

Value we pass to the F^n nothing but Argument.

```
Function fun_name (arg1, arg2, ... arg n)
{
    // Do something
}
```

2] e.g

```
Function   message(name)
{
        Console.log(name);
}
    message(" Rohit");
```

O/P :   Rohit

keypoints :- 1] Function mai order important hai.
2] Hum function mai First argument don't skip but Second argument may be skip.
3] if you don't pass 2nd argument Javascript return value as "undefined".

* return keyword :
return keyword is used to return some value from the Function.

$$\begin{tikzcd} I/P \arrow[r] & \boxed{\text{Function}} \arrow[r] & O/P \end{tikzcd}$$

```
Function   sum(a, b)
{
    retun a+b;
}
let c = sum(~~4,4~~ 2, 3);
Consol.log(c);
```

Note : 1] if you add statement after return keyword that not executed because of return keyword instany return something /or exit from that f^n.

* What is scope?

Scope determine the accessibility of variable, object & f^n from different part of the code

1] Function
2] Block
3] Lexical
4] Global

1] Function Scope :-
Variables defined inside a function are not accessible (visible) from outside the f^n.

```
let sum = 2;          // Global scope   (Accessible anywhere)
Function sum() {
    let a = 2;        // f^n  ]
    let b = 3;        // f^n  ] scope
    let sum = a+b;    // f^n  ]
    Console.log(sum);
}
```

2] Block Scope:
Variables declared inside { } block cannot be accessed from outside the block.

let, const keyword scope is block

3) ~~Global~~ Lexical Scope :- (nested f^n)

A variable defined outside a function can be accessible inside another f^n defined after the variable declara

* This opposit is NOT true.

* Hoisting :-

Hoisting in javascript default behaviour of moving declaration to the top.

* Function Expression :

- Function expression nothing but anonyms f^n.
- Function expression is used to define function inside any expression.
- function expression is used as an Immediately Invoked Function Expression, which run as soon as it is defined.
- A f^n expression are stored in a variable & can be accessed using variable_name.

Syntax :

```
Const  var_name = Function (arg1, arg2, ....)
{
    // do or return something
}

Var_name (val1, val2, ...);
```

* Highor Order function:

A Function that does one or both:

1) Take one or multiple F^n as argument.
2) return a Function.

1] e.g

```
    Function multiplegreet( ~~greet~~ Func , n)
    {
        For (let i=0; i<=n; i++)
        {
            Func();
        }
    }
    let greet = function() {
            Console.log("hello");
    }
    multiplegreet (greet, 2);

    O/P : hello
          hello
```

2]

```
  Function OddEvenTest( req)
  {
    if( req == "odd")
    {
        return Function (n)
        {
            Console.log( !(n%2) = =0));
        }
    }
```

```
else if (req == "even")
 {
    return function(n)
     {
        Console.log((n%2)==0));
     }
 }
 else {
      console.log("wrong request");
    }
}

 let   req = "odd";
 let   res = oddEvenTest(req);
       res(5);
```

* Method :

```
 Const   Cal = {
         add : Function (a, b)
         {
             Console.log(a+b);
          },
         sub : function (a, b)
          {
             Console.log(a-b);
           }
   }
   Cal.add (1,2);
```

Function Declaration	Function Expression
1] Fⁿ declaration is a traditional way of declaring Fⁿ.	1] Fⁿ expression is same as Fⁿ declaration, but with it, we can declare unnamed Fⁿ.
2] Fⁿ is named	2] Fⁿ is nameless, as well as named
3] Fⁿ are invoked only using the Fⁿ name	Fⁿ are stored inside a variable & invoked only using the variable name.

4] e.g
Function message()
{
}
message();
	4] e.g
	let con = Function ()
	{
	};
	con();

5] no need to add semi-colon at the end of Fⁿ declaration	5] need to add semi-colon at the end of Fⁿ expression.
6] Fⁿ declaration is hoisted at the top of the program	6] Fⁿ expression is not hoisted at the top of the program & is only loaded when the interpreter reaches the line of code.