

## \* Function :

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

### Function Definition (testing JS)

```
Function fun_name()
```

```
{
```

```
  // Do something
```

```
}
```

### Function Call (Using the Function)

```
fun_name();
```

1) e.g      function message()

```
{
```

```
  console.log("Rohit");
```

```
}
```

```
message();
```

OP : Rohit

## \* Function with Argument :

Value we pass to the F<sup>n</sup> 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

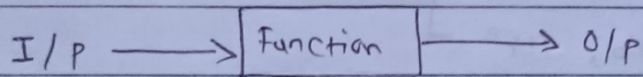
keypoints :- 1] Function mai order important hai.

2] Hum Function mai First argument don't skip but second argument may be skip.

3] if you don't pass 2<sup>nd</sup> argument Javascript return value as "undefined".

\* Return keyword:

return keyword is used to return some value from the function.



`function sum(a, b)`  
`{`

`return a+b;`

`}`

`let c = sum(2, 3);`

`console.log(c);`

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



\* What is scope?

Scope determines the accessibility of variable, object & fn from different <sup>part</sup> 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 fn.

```
let sum = 2; // Global Scope (Accessible anywhere)
```

```
Function sum() {
```

```
  let a = 2; // fn
```

```
  let b = 3; // fn
```

```
  let sum = a + b; // fn
```

```
  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 fn)

A variable defined outside a function can be accessible inside another fn defined after the variable declaration.

\* This concept is NOT true.

### \* Hoisting :-

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

### \* Function Expression :

- Function expression nothing but anonymous fn.
- Function expression is used to define function inside any expression.
- Function expression is used as an Immediately Invoked Function Expression, which runs as soon as it is defined.
- A fn 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, ...);
```



\* Higher Order Function:

A Function that does one or both:

- 1) Take one or multiple F<sup>n</sup> as argument.
- 2) return a Function.

1] e.g

```
function multiplegreet(greetFunc, 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("Invalid request");
}
}

let req = "add";
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]  $F^n$  declaration is a traditional way of declaring  $F^n$ .
- 2]  $F^n$  is named
- 3]  $F^n$  are invoked only using the  $F^n$  name
- 4] e.g.  

```
function message()
{
}
message();
```
- 5] no need to add semi-colon at the end of  $F^n$  declaration
- 6]  $F^n$  declaration is hoisted at the top of the program
- 1]  $F^n$  expression is same as  $F^n$  declaration, but with it, we can declare unnamed  $F^n$ .
- 2]  $F^n$  is nameless, as well as named
- 3]  $F^n$  are stored inside a variable & invoked only using the variable name.
- 4] e.g.  

```
let con = function ()
{
}
con();
```
- 5] need to add semi-colon at the end of  $F^n$  expression.
- 6]  $F^n$  expression is not hoisted at the top of the program & is only loaded when the interpreter reaches the line of code.