Function scope

```
function abc() {
	var a = 10;
	console.log(a);
}

abc();
```

a : undefined

a: 10

Global Scope: when a variable is declared outside any function and not inside any parent function also then it becomes global scope.

```
var a = 10;
function abc() {
	console.log(a);
}

abc();
```

Window

Any variable or a function that is declared in javascript which is not in local scope would be part of the window object.

```
function gp() {
 var a = 20;
function parent(){
   function child(){
       console.log(a);
   }
   child()
}
    parent()
}

gp()
```

gp(gp-scope + window) -> parent(gp-scope + parent-scope + window) -> child(gp+parent+child-scope + window)

Lexical scope is the ability for a function scope to access variables from the parent scope. We call the child function to be lexically bound by that of the parent function

```
function gp() {
 var a = 20;
 console.log(a)

function parent(){
     var a = 30;

    console.log(a)

   function child(){
        var a = 40;

       console.log(a);
   }
     child()
     console.log(a)
}
    parent()
     console.log(a)

}
gp()
```

Hoisting

All the variables would be hoisted to the top of the scope and assigned a value of undefined when they are declared;

```
var a = 20;

function abc() {
    console.log(a);
    var a = 30;
    var b = 20;
    var c = 30;
    console.log(a);
}

abc();
```

Internally javascript converts the above function to this

```
var a = 20;
function abc() {
   var a;
   var b;
   var c;
    console.log(a);
    a = 30;
    b =20;
   c =30
```

```
    console.log(a);
}

abc();



function abc() {
    // var salary
     console.log("Original salary was " + salary);

     salary = "5000$";

     console.log("My New Salary " + salary);

}


var salary = "5000$";

function abc() {
     console.log("Original salary was " + salary);
     var salary = "5000$";
     Var salary1 =  2000;
     console.log("My New Salary " + salary);
}
```

Internally js is going to convert it

```
var salary = "5000$";
function abc() {
    var salary;
    var salary1;
     console.log("Original salary was " + salary);
     salary = "5000$";
     salary1 =2000;
     console.log("My New Salary " + salary);
}
```

```
function xyz() {
 console.log(x);
 var x = 10;
}

xyz();
```

Closure

Closure is retaining the scope of a variable event after a function has returned.

Closure is a side effect of a function returning another function.

parent's variables are transferred to closure scope of the returned function which is being held by the variable

```
function makeWorker() {
  var name = "Pete";

  return function() {
    console.log(name); // variable is trapped
  };
}

var name = "John";


var work = makeWorker();


work();
```

A closure scope is formed when a variable is utilised from the parent in a function which is getting returned.

```
////////

function parent(){
   var a = 10;
   return function child(){
        console.log(a) // trapped(closure scope)
      }
}

let functionReceived = parent()
functionReceived();

//////////////

function makeCounter() {
  var count = 0;

  return function() {
        return count++;
   };
}

var counter = makeCounter();

console.log(counter());
```