## Scope

In JavaScript, scope refers to the accessibility of variables and functions in your code. Understanding scope is crucial for writing clean and efficient code. There are three types of scopes: global scope, functional/local scope, and block scope.

### Global Scope
Variables declared outside of any function or block are in the global scope. They can be accessed from anywhere in the code, including inside functions or blocks.

### Functional/Local Scope
Variables declared inside a function or block are in the functional/local scope. They can only be accessed from within that function or block.

### Block Scope
Variables declared can only be accessed from within that block.

### var Varibales

```javascript
var globalVar = 10;
console.log(`I am global var ${globalVar} in global scope`);

if(true)
{
  var blockVar = 10;
  console.log(`I am block var ${blockVar} in block scope`);
  console.log(`I am global var ${globalVar} in block scope`);
}

function print()
{
  var localVar = 10;
  console.log(`I am local var ${localVar} in local or function scope`);
  console.log(`I am block var ${blockVar} in local or function scope`);
  console.log(`I am global var ${globalVar} in local or function scope`);
}

print();

console.log(`I am block var ${blockVar} in global scope`);
//console.log(`I am local var ${localVar} in global scope`); // error
```

let or const variables

```
let globalLet = 10;
const globalConst = 20;

console.log(`I am global let ${globalLet} in global scope`);
console.log(`I am global const ${globalConst} in global scope`);

if(true)
{
  let blockLet = 10;
  const blockConst = 20;

  console.log(`I am block let ${blockLet} in block scope`);
  console.log(`I am block const ${blockConst} in block scope`);

  console.log(`I am global let ${globalLet} in block scope`);
  console.log(`I am global const ${globalConst} in block scope`);
}

function print()
{
  let localLet = 10;
  const localConst = 20;

  console.log(`I am local let ${localLet} in local or function scope`);
  console.log(`I am local const ${localConst} in local or function scope`);

  //console.log(`I am block let ${blockLet} in local or function scope`);
//error
  //console.log(`I am block const ${blockConst} in local or function scope`);
//error

  console.log(`I am global let ${globalLet} in local or function scope`);
  console.log(`I am global const ${globalConst} in local or function scope`);
}

print();

//console.log(`I am block let ${blockLet} in global scope`); //error
//console.log(`I am block const ${blockConst} in global scope`); //error

//console.log(`I am local let ${localLet} in global scope`); // error
//console.log(`I am local const ${localConst} in global scope`); // error
```

By default, all variables are function or local scope. But let and const are block scope. var is liberal. var is not a block scope. var variables declared in block scope can be accessed in function scope and global scope too.

## Lexical environment

Lexical environment is a fundamental concept in JavaScript that refers to the specific context in which code is executed. It encompasses variables, functions, and objects that are accessible and in scope at a particular point during the execution of code. A fresh lexical environment is generated whenever a function is called or invoked in JavaScript. This lexical environment encompasses all the variables and functions that are in scope and can be accessed within that particular function call.

## Scope chaining or Lexical Scoping

Scope chaining, also known as lexical scoping, is a mechanism in JavaScript that allows a function to access variables from its outer (enclosing) lexical environment as well as from the global scope. This means that functions can access variables defined in their parent functions, grandparent functions, and so on, all the way up to the global scope.

```javascript
let a = 10;
const b = 20;
var c = 30;

function outer()
{
  let a = "outer 10";
  const b = "outer 20";

  console.log(a); // outer 10
  console.log(b); // outer 20
  console.log(c); // 30

  function inner()
  {
    let a = "inner 10";
    console.log(a); // inner 10
    console.log(b); // outer 20
    console.log(c); // 30
  }

  inner();

}

console.log(a); // 10
console.log(b); // 20
console.log(c); // 30

outer();
```