Scopes: Where the things (var/func) are visible

- visibility
  - variables
  - function

Every thing inside your code, will be used in one of the following two ways

1) Either it will be getting some value assigned to it
2) Some value will be retrived from it.

Do you think JS is compiled or interpreted?

Compiler vs Interpreted

↳ C/C++ ↳ Shell

Javascript is mix of both.

Javascript is a mix of both.

Javascript executes in two phases:

- Passing phase : Scope Resolution
- Execution phase

Types of Scopes (Scope Resolution):

1) Global
2) Function
3) Block

Global Scope example:

```
console.log(name);
let name = "Rizon";
function fun() {
  console.log(name);
}
fun();
console.log(name);
```

```
console.log(name);
            ^

ReferenceError: Cannot access 'name'
```

```
console.log(name);
var name = "Rizon";
function fun() {
  console.log(name);
}
fun();
console.log(name);
```

```
undefined
Rizon
Rizon
```

Block Scope :

```
{
  let x = 10;
}

console.log(x); // Error
```

```
{
  var x = 10;
}

console.log(x); // 10
```

using 'var' in block scope doesn't make sense if only support in function scope and global scope

'x' is throwing error because 'x'. is only accessible in the block scope NOT outside the block.

```
{
    // Block
}
```

Function Scope :

```
if (false) {
  var x = 10;
}
console.log(x); // undefined
```

x → is visible outside since it's a global scope

```
function fun() {
  var y = 10;
}
console.log(y); // Error
```

Now 'var' is enclosed in a f() so scope of 'y' is fun().

Examples:

04.functionscope.js U

04.functionscope.js > gun

```javascript
if (false) {
  var x = 10;
}

console.log(x); // undefined

function fun() {
  var y = 10; // not accessible outside
}

function gun() {
  console.log(y); // undefined
  var y = 10;
}

gun();
```

```javascript
if (false) {
  var x = 10;
}

console.log(x);

function fun() {
  var y = 10;
}

function gun() {
  console.log(y); // Reference Error
  let y = 10;
}

gun();
```

Cannot access 'y' before initialization

Hoisting Concept.

'var' creates a function scope and 'Let' creates a block scope

'y' is accessible any where in a fC)

'y' is accessible below it.

Module Scope: HOLD FOR NOW (Used in Node JS)

Parsing Phase : J.S reads the code line by line and start allocating the variables NOT the value to the corresponding scope.

```
var teacher = "Sanket";
function fun() {
  var teacher = "Pulkit";
  console.log(teacher);
}

function gun() {
  var student = "Sarthak";
  console.log(student);
}

fun();
gun();
console.log(teacher);
```

GS

G.S

fun scope

G.S

Scope of gun

Line no	Scope Manager
#1	Global Scope
#2 FD	Global Scope
#3	Function Scope of fun
#7 FD	Global Scope
#8	Function scope of gun

Every time we see a formal declaration (FD) we think about the scope in parsing phase

Executing Phase :

#1 [Sanket] → Global

teacher

#11

↳ #3 [Pulkit] → Scope of fun

teacher

#9 console.log(teacher) // Pulkit

#12

↳ #7 [Sarthak] → scope of gun

student

#8 console.log(student) // Sarthak

#13 [Sanket] console.log(teacher) // Sanket

teacher → G.S.

o/p Pulkit

Sarthak

Sanket

05.scope.js U

05.scope.js > ...

```js
var teacher = "Sanket";
function fun() {
  var teacher = "Pulkit";
  content = "JS";
  console.log(teacher);
}

function gun() {
  var student = "Sarthak";
  console.log(student);
}

fun();
gun();
console.log(teacher);
console.log(content);
```

G.S

scope of fun

G.S

scope of gun

```
Pulkit
Sarthak
Sanket
JS
```

#1 [Sanket] → G.S
teacher

#13
↳ #3 [Pulkit] → Scope of fun
teacher

#4 we never formally assigned variable content to the scope of fun, when this happen during execution phase you go one scope OUT, you go to global scope now you ask global scope do you have variable content in G.S. answer is NO, here you try to assign a value to the variable and it's not present in

any of the enclosing scope it will automatically become GLOBAL VARIABLE.

$\boxed{\text{JS}}$ → GLOBAL

Content

#5 Pulkit

#14

↳ #9 $\boxed{\text{Sarthak}}$ → scope of gun

Student

#10 Sarthak

#15 Sanket

#16 JS

```
Pulkit
Sarthak
Sanket
JS
```

The mechanism of AUTO GLOBAL only works when you trying to assign the value to the variable.

```
var teacher = "Sanket";
function fun() {
  var teacher = "Pulkit";
  content = "JS";
  console.log(teacher);
}
function gun() {
  var student = "Sarthak";
  console.log(student);
}
console.log(content);

fun();
gun();
console.log(teacher);
```

G.S

Scope of fun

G.S

Scope of gun

#1 $\boxed{\text{Sanket}}$ → G.S

teacher

#1 there is no content variable in G.S, You are not assigning the value, you are using the value so you get error.

Reference Error

Scopes in Strict Mode :

Keyword → "use strict"

Auto Global might be issue in production level so we use "use strict"

```js
"use strict";
var teacher = "Sanket";
function fun() {
  var teacher = "Pulkit";
  content = "JS";
  console.log(teacher);
}

function gun() {
  var student = "Sarthak";
  console.log(student);
}

fun();
gun();
console.log(teacher);
console.log(content);
```

Error

while using "use strict" auto global doesn't work.

08.nestedscope.js U ×

08.nestedscope.js > fun

```js
// fun -> GS
function fun() {
  var x = 10; // x -> fun scope
  function gun() {
    // gun -> fun scope
    var y = 20; // y -> gun scope
    console.log(x); // 10 (one scope outside)
    console.log(y); // 20
  }
  gun();
  console.log(x); // 10
  console.log(y); // error
}

fun();
```

Two type of scoping:

1) Lexical Scoping:

Scope are determined during compile time.

For e.g. JS

2) Dynamic Scoping:

Scope are determined during run time.

For e.g. Bash

09.whichscoping.js U

09.whichscoping.js > ask

```javascript
var teacher = "Sanket"; // teacher -> global -> sanket
function ask(question) {
  // ask -> global, question -> ask
  console.log(teacher, question); // sanket why?
}

function fun() {
  // fun -> global
  var teacher = "Pulkit"; // teacher -> fun -> pulkit
  ask("why?");
}
fun();
```

output: Sanket Why?

if JS work like dynamic scoping then o/p → Pulkit why?

Tempoary Dead Zone : The region before the declaration of a variable which is having a block scope made by let is actually called as TDZ.

```
12.let_block_scoping.js U ×
12.let_block_scoping.js > fun
var teacher = "Sanket"; // GS
// GS
function fun() {
  console.log(teacher); // no error will be given (undefined)
  console.log(content); // error (temporary dead zone)
  var teacher = "Pulkit"; // scope of fun
  let content = "JS"; //  content will be access only post declaration
  if (content == "JS") {
    let hours = "120+";
    console.log(hours); //120+
    console.log(content); // JS
  }
  console.log(teacher); // Pulkit
  console.log(content); // JS
  //   console.log(hours); // error
}
fun();
console.log(teacher); // Sanket
console.log(content); // error
```

var : function scope
let : block scope

var in a block scope :

If you have var inside a block it doesn't care about the block, so you will have access to the variable "x" outside the while loop also.

```
function fun() {
  var i = 5;
  while (i < 20) {
    var x = i;
    i++;
  }
  console.log(x); // 19
}
fun();
```

use case of var :

```
function fun(x) {
  var i; // let i;  // both same
  if (x % 2 == 0) {
    i = 0;
  } else {
    i = 1;
  }
}
```

```
function fun(x) {
  let i; // var i;  // both same
  if (x % 2 == 0) {
    i = 0;
  } else {
    i = 1;
  }
}

// OR

function gun(x) {
  if (x % 2 == 0) {
    var i = 0;
  } else {
    var i = 1;
  }
  console.log(i); // 0
}

gun(10);
```

use case of let :

```
function fun() {
  for (var i = 0; i < 10; i++) {
    // do something
  }
  console.log(i); // you can still access "i"
}

// Doesn't make sense to access "i" after end of for loop
// So we use "let" her

function gun() {
  for (let i = 0; i < 10; i++) {
    // do something
  }
  console.log(i); // you cannot  access "i"
}

fun();
gun();
```

Depends on the project we use var & let.

var : It allow redeclaration of variable.

```
var x = 10;
var x = 20;
```

let : It doesn't allow redeclaration

```
let x = 10;
let x = 20;   // Error
```

const : It's same as let but the only difference is const can't be changed through reassignment and it can't be redeclared. However, if a const is an object / array it's properties / items can be updated / removed.

```
Const x = 10
      x = 20;   // Error

Const obj = {x : 10}
      obj.x = 9   // No error
```

Task

Does temporary dead zone exist for const?

Answer is NO,

- TDZ concept is related to block scope variable (let & const)

- But in const variable unlike let variable, are always initialized to undefined at the beginning of their scope, make them accessible even before their declaration line

```
if (true) {
    console.log(x); // Reference Error
    console.log(y); // undefined
    let x = 10;
    const y = 20;
}
```

Function Expression :

```
16.functionExpression.js U ×
16.functionExpression.js > y
function fun() {
  // code
}

let f = function gun() {
  // code
};

let a = function () {
  // code
};

(function x() {
  // code
});

(function () {
  // code
});

let y = () => {
  // code
};
```

function declaration

function expression

expression

expression

expression

expression

When the 1st valid expression of the line where we are defining the function is the keyword function then it's a function declaration

IIFE : Immediate Invoked Function Expression

↳ the moment we define it, then only we call it

```
18.iife.js U ×
18.iife.js > ...
(function x(y) {
  console.log("hi", y);
})("Rizon");
```

↳ invoke the f().

o/p ⇒ hi Rizon

Scope of function expression :

```
const f = function fun() {
  console.log("How much fun ??");
};

f(); // How much fun
fun(); // Error, not accessible outside f.
```

Function expression is of two types :

anonymous function

```
let a = function (){

}
```

named function

```
let f = function gun () {

}
```

Named function is a better way of writing.

```
function fun(fn) {
  console.log("Welcome to fun");
  fn();
}

fun(function () {
  console.log("Wow so much fun");
});
```

This code is readable

But NOT this code.

Better way of writing it :

20.usecaseofNamedFunction.js > askingAboutFun

```javascript
function fun(fn) {
  console.log("Welcome to fun");
  fn();
}

fun(function askingAboutFun() {
  console.log("Wow so much fun");
});
```