**VAR**

var declarations ruled as King for long, but there are issues associated with variables declared with var though. So, it was necessary for new ways to declare variables to emerge.

**Scope of var**

**Scope** essentially means where these variables are available for use. var declarations are globally scoped or function/locally scoped. It is globally scoped when a var variable is declared outside a function. This means that any variable that is declared with var outside a function block is available for use in the whole window. var is function scoped when it is declared within a function. This means that it is available and can be accessed only within that function.  
To understand further, look at the example below.

var greeter = "hey hi";

function newFunction()

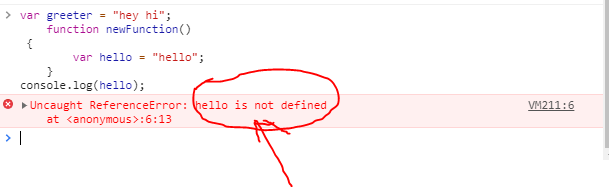
{

var hello = "hello";

}

console.log(hello);

Here, greeter is globally scoped because it exists outside a function while hello is function scoped. So we cannot access the variable hello outside of a function. If we try to access, we will get an error as shown in fig. below which is a result of hello not being available outside the function.



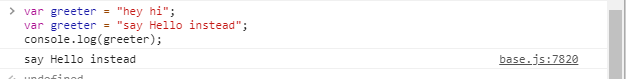
Var variables can be re-declared and updated. This means that we can do this within the same scope and won't get an error. The example has been shown in below code:

var greeter = "hey hi";

var greeter = "say Hello instead";

console.log(greeter);

Here, the greeter has been assigned redeclared with new value, so the output will be the new value. The output is displayed in the image below.



The code for updating the value is as shown below along with the output displayed in image.

var greeter = "hey hi";

greeter = "say Hello instead";

console.log(greeter);

Output:

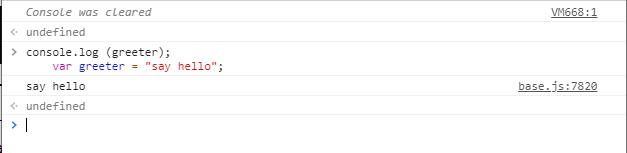


**Hoisting of VAR:**

Hoisting is a JavaScript mechanism where variables and function declarations are moved to the top of their scope before code execution.

Eg: console.log (greeter);

var greeter = "say hello";



**LET**

Let is preferred for variable declaration now. It's no surprise as it comes as an improvement to the var declarations.

Let is block scooped. Block is a part of code bounded by {}. A block is the code inside the curly braces. So, if the variable is declared within the block with let function, then it is available for use within that block only. This can be seen in the following example.

Eg: let greeting = "say Hi";

let times = 4;

if (times > 3)

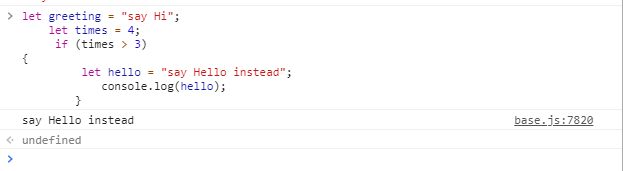
{

let hello = "say Hello instead";

console.log(hello);

}

Output:



Here, the hello function has been defined and called inside the block, so no error is displayed and the output is displayed.

Now, we will try to call hello function outside the block without defining it outside the block. This will give us an error of not defined function.

We will take same code as above , just by adding the line to call hello function outside the block. This will clear our concepts.

Eg: let greeting = "say Hi";

let times = 4;

if (times > 3)

{

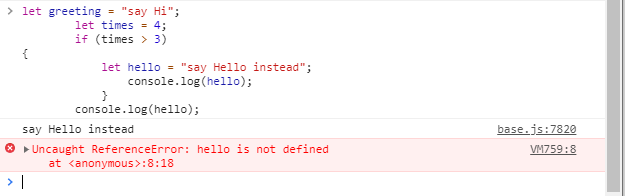
let hello = "say Hello instead";

console.log(hello);

}

console.log(hello);

Output:



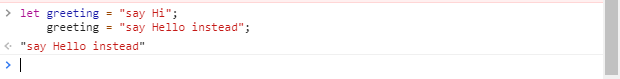
Let variable can be updated just like var variable. But, it cannot be re-declared.

Below images show the example of the same. First image shows updation of the value, second shows the re-declaration code.

Code:

let greeting = "say Hi";

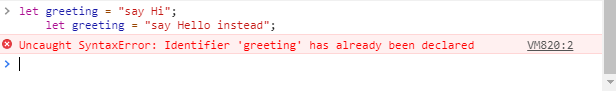
greeting = "say Hello instead";



Code:

let greeting = "say Hi";

let greeting = "say Hello instead";



The second image shows the error of given variable has already been declared.

Now, we have seen that the same variable cannot be defined two times. This is the case for same block. But, if we define a same variable in two different scopes, it can be defined without any error. The example of the same is shown in the image below:

Code:

let greeting = "say Hi";

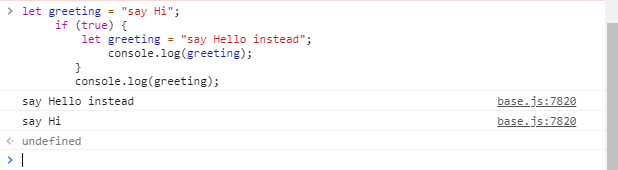
if (true) {

let greeting = "say Hello instead";

console.log(greeting);

}

console.log(greeting);



In the above image, we can see that there is no error. This is because both the instances are treated as different variables since they have different scopes or block.

Just like var, let declarations are hoisted to the top. Unlike var which is initialized as undefined, the let keyword is not initialized. So if you try to use a let variable before declaration, you'll get a Reference Error.

**CONST**

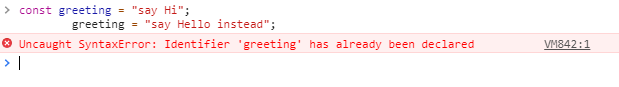
Variables declared with the const maintain constant values. const declarations share some similarities with let declarations.

const declarations are block scoped. Like let declarations, const declarations can only be accessed within the block it was declared.

const cannot be updated or re-declared. This means that the value of a variable declared with const remains the same within its scope. It cannot be updated or re-declared.

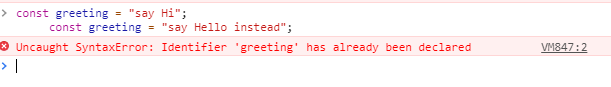
Eg.1: const greeting = "say Hi";

greeting = "say Hello instead";



Eg.2: const greeting = "say Hi";

const greeting = "say Hello instead";



Just like let, const declarations are hoisted to the top but are not initialized.

|  |  |  |  |
| --- | --- | --- | --- |
|  | VAR | LET | CONST |
| Scope | Global and function | Block Scoped | Block Scoped |
| Can be updated? | Yes | Yes | No |
| Can be re-declared? | Yes | No | No |
| Hoisting | At the top initialized with undefined | At the top without initialization | At the top without initialization |
| Declaration | Can be declared without initialization | Can be declared without initialization | Must be initialized during declaration |