What are the differences between var, let and const?

**Scope of var**

**Scope** essentially means where these variables are available for use. var declarations are globally scoped or function/locally scoped.

The scope is global 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";

}

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. So if we do this:

var tester = "hey hi";

function newFunction() {

var hello = "hello";

}

console.log(hello); // error: hello is not defined

We'll get an error which is as 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.

var greeter = "hey hi";

var greeter = "say Hello instead";

and this also

var greeter = "hey hi";

greeter = "say Hello instead";

### Hoisting of var

Hoisting is a JavaScript mechanism where variables and function declarations are moved to the top of their scope before code execution. This means that if we do this:

console.log (greeter);

var greeter = "say hello"

it is interpreted as this:

var greeter;

console.log(greeter); // greeter is undefined

greeter = "say hello"

So var variables are hoisted to the top of their scope and initialized with a value of undefined.

### Problem with var

There's a weakness that comes with  var. I'll use the example below to explain:

var greeter = "hey hi";

var times = 4;

if (times > 3) {

var greeter = "say Hello instead";

}

console.log(greeter) // "say Hello instead"

So, since times > 3 returns true, greeter is redefined  to "say Hello instead". While this is not a problem if you knowingly want greeter to be redefined, it becomes a problem when you do not realize that a variable greeter has already been defined before.

If you have used greeter in other parts of your code, you might be surprised at the output you might get. This will likely cause a lot of bugs in your code. This is why let and const are necessary.

### let is block scoped

A block is a chunk of code bounded by {}. A block lives in curly braces. Anything within curly braces is a block.

So a variable declared in a block with let  is only available for use within that block. Let me explain this with an example:

let greeting = "say Hi";

let times = 4;

if (times > 3) {

let hello = "say Hello instead";

console.log(hello);// "say Hello instead"

}

console.log(hello) // hello is not defined

We see that using hello outside its block (the curly braces where it was defined) returns an error. This is because let variables are block scoped .

### let can be updated but not re-declared.

Just like var,  a variable declared with let can be updated within its scope. Unlike var, a let variable cannot be re-declared within its scope. So while this will work:

let greeting = "say Hi";

greeting = "say Hello instead";

this will return an error:

let greeting = "say Hi";

let greeting = "say Hello instead"; // error: Identifier 'greeting' has already been declared

However, if the same variable is defined in different scopes, there will be no error:

let greeting = "say Hi";

if (true) {

let greeting = "say Hello instead";

console.log(greeting); // "say Hello instead"

}

console.log(greeting); // "say Hi"

Why is there no error? This is because both instances are treated as different variables since they have different scopes.

This fact makes let a better choice than var. When using let, you don't have to bother if you have used a name for a variable before as a variable exists only within its scope.

Also, since a variable cannot be declared more than once within a scope, then the problem discussed earlier that occurs with var does not happen.

### Hoisting of let

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 declarations are block scoped**

Like let declarations, const declarations can only be accessed within the block they were 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. So if we declare a variable with const, we can neither do this:

const greeting = "say Hi";

greeting = "say Hello instead";// error: Assignment to constant variable.

nor this:

const greeting = "say Hi";

const greeting = "say Hello instead";// error: Identifier 'greeting' has already been declared

Every const declaration, therefore, must be initialized at the time of declaration.

This behavior is somehow different when it comes to objects declared with const. While a const object cannot be updated, the properties of this objects can be updated. Therefore, if we declare a const object as this:

const greeting = {

message: "say Hi",

times: 4

}

while we cannot do this:

greeting = {

words: "Hello",

number: "five"

} // error: Assignment to constant variable.

we can do this:

greeting.message = "say Hello instead";

This will update the value of greeting.message without returning errors.

**Hoisting of const**

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

So just in case you missed the differences, here they are:

* var declarations are globally scoped or function scoped while let and const are block scoped.
* var variables can be updated and re-declared within its scope; let variables can be updated but not re-declared; const variables can neither be updated nor re-declared.
* They are all hoisted to the top of their scope. But while var variables are initialized with undefined, let and const variables are not initialized.
* While var and let can be declared without being initialized, const must be initialized during declaration.

Difference between undefined vs not defined vs NaN

**undefined:**It is a JavaScript keyword that has a special meaning. Everything which gets a space in memory will contain undefined until we assign a value to that memory space.

Let’s understand how the JavaScript code is being executed to see a more clear picture. Everything in JavaScript happens inside the execution context. Execution context is the little separate section where code is being executed and variables get their memory space.  
The JavaScript code is being executed in [two-phase](https://www.geeksforgeeks.org/javascript-code-execution/),

1. The first one is the memory allocation phase during this all the variables and function definitions get stored inside the memory heap. The JavaScript assigns undefined to each variable in this phase.
2. The second one is a thread of the execution phase, during this the code written inside the JavaScript file is being executed.  
   Each variable holds the value *undefined*till the program reaches the line where we have assigned that variable. After that line, the variable’s undefined value gets replaced by the original value.

<script>

console.log(a);

var a = 5;

console.log(a);

</script>

**Output (In console):**

undefined

5

**Additional Points:**

1. If you are assigning a function call to a variable, and that function doesn’t return anything, the variable will become *undefined*.
2. You can explicitly assign *undefined* to any variable but it is not good practice to use language keywords in the way it is not expected.

**not defined:**In JavaScript, it is one of the reference errors that JavaScript will throw when someone accesses the variable which is not inside the memory heap.

<script>

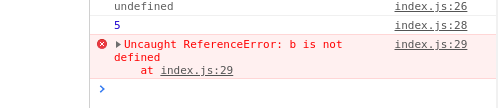
console.log(a);

var a = 5;

console.log(a);

console.log(b);

</script>



Nan

In JavaScript, NaN is short for "Not-a-Number".

In JavaScript, NaN is a number that is not a legal number.

The Global NaN property is the same as the Number.Nan property.

What is difference between null and undefined and where to use what?

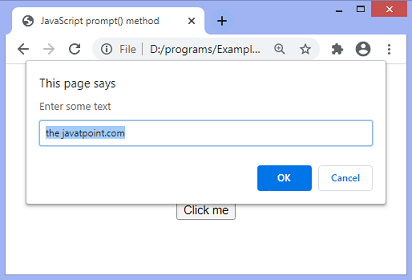
|  |  |
| --- | --- |
|  |  |
| **1.** | Undefined means a variable has been declared but has yet not been assigned a value. | Null is an assignment value. It can be assigned to a variable as a representation of no value. |
| **2.** | It is an ECMAScript1 (ES1) feature. | It is a Primitive value in Javascript |
| **3.** | Its supported browsers are -:  Chrome , Microsoft Edge , Internet Explorer , Firefox , Safari , Opera Mini | Its syntax is -:  **null** |
| **4.** | It is a global property. | Its supported browsers are -:  Chrome , Microsoft Edge , Internet Explorer , Firefox , Safari , Opera Mini |

What is execution context

When the JavaScript engine scans a script file, it makes an environment called the Execution Context that handles the entire transformation and execution of the code. During the context runtime, the parser parses the source code and allocates memory for the variables and functions.

What is use of prompt in JavaScript?

Javascript Window prompt() Method. The Javascript Window prompt() method is used to display a dialog box with an optional message prompting the user to input some text. It is often used if the user wants to input a value before entering a page. It returns a string containing the text entered by the user, or null.



Why JavaScript is Dyanmic Language?

Being able to change the underlying type of a variable while the program is running without causing an error is what makes JavaScript a dynamic language.

For example

Let a=90;

Console.log(a);

Output 90

a=7;

console.log(a);

output 7