

Search...

[DSA with JS - Self Paced](#)[JS Tutorial](#)[JS Exercise](#)[JS Interview Questions](#)[JS Array](#)[JS String](#)[JS](#)

Difference between var, let and const keywords in JavaScript

Last Updated : 21 Apr, 2025

JavaScript provides three ways to declare variables: var, let, and const, but they differ in scope, hoisting behaviour, and re-assignment rules. Understanding these differences helps write more predictable and maintainable code.

What is var, let and const in JavaScript?

- **var:** Declares variables with function or global scope and allows re-declaration and updates within the same scope.
- **let:** Declares variables with block scope, allowing updates but not re-declaration within the same block.
- **const:** Declares block-scoped variables that cannot be reassigned after their initial assignment.

1. Declaring Variables with var

[var](#) is the original keyword for declaring variables in JavaScript. It is [function-scoped](#) or globally scoped, depending on where it's declared.

```
function e() {  
    var n = "Janardhan";  
    console.log(n);  
}  
e();
```



Output

Janardhan



- The function e declares a variable n with the value “Janardhan” and logs it to the console.

- When the function e() is called, it outputs “Janardhan” to the console.

2. Block Scope with let

Introduced in [ES6](#), [let](#) provides [block-level scoping](#). This means the variable is only accessible within the block (like loops or conditionals) where it is declared.

```
if (true) {  
  let age = 30;  
  console.log(age);  
}  
console.log(age)
```

Output

```
ReferenceError: age is not defined  
    at Object.<anonymous> (/home/guest/sandbox/Solution.js:5:13)  
    at Module._compile (node:internal/modules/cjs/loader:1198:14)  
    at Object.Module._extensions..js (node:internal/modules/cjs/loader:  
    at Module.load (node:internal/modules/cjs/loader:1076:32)  
    at Function.Module._load (node:internal/modules/cjs/loader:911:12)  
    at Function.executeUserEntryPoint [...]
```

Block Scope with let

- **Block Scope:** The variable age is declared with let inside the if block, so it is only accessible within that block and cannot be accessed outside of it.
- **Reference Error:** The second console.log(age) will throw a Reference Error because age is not defined in the outer scope.

3. Immutability with const

[const](#) is used to declare variables that should not be reassigned after their initial assignment. This keyword is also block-scoped like let.

```
const country = "USA";  
console.log(country);
```

Output

USA

- The variable country is declared with const, meaning its value cannot be reassigned after initialization.
- The value of country (which is “USA”) is printed to the console.

4. Hoisting Behavior of var, let, and const

[Hoisting](#) is a JavaScript behavior where variable declarations are moved to the top of their containing scope. However, the way hoisting works differs for var, let, and const.

- **Hoisting with var:** The variable x is hoisted but only initialized after the console.log(x) call. It prints undefined because it is declared but not assigned a value yet.

```
console.log(x);  
var x =5;
```



Output

undefined

- **Hoisting with let:** The code logs x before it is declared with let, causing a Reference Error. This happens because let variables are hoisted but not initialized, so they remain in the TDZ until the declaration is executed.

```
console.log(x)  
let x=10
```



Output



```
Hangup (SIGHUP)
/home/guest/sandbox/Solution.js:1
console.log(x)
      ^
```

```
ReferenceError: Cannot access 'x' before initialization
    at Object.<anonymous> (/home/guest/sandbox/Solution.js:1:13)
    at Module._compile (node:internal/modules/cjs/loader:1198:14)
    at Object.Module._extensions..js (node:internal/modules/cjs/loader:1252:10)
    at Module.load (node:internal/modules/cjs/loader:1076:32)
    at Function.Module._load (node:internal/modules/cjs/loader:911:12)
    at Function.exec...
```

no Hoisting with let

- **Hoisting with const:** The variable x is declared with const, which is block-scoped and not hoisted. When console.log(x) is called before the declaration, it throws a Reference Error due to being in the Temporal Dead Zone (TDZ).

```
console.log(x)
const x=10
```



Output

```
ReferenceError: Cannot access 'x' before initialization
    at Object.<anonymous> (/home/guest/sandbox/Solution.js:1:13)
    at Module._compile (node:internal/modules/cjs/loader:1198:14)
    at Object.Module._extensions..js (node:internal/modules/cjs/loader:1252:10)
    at Module.load (node:internal/modules/cjs/loader:1076:32)
    at Function.Module._load (node:internal/modules/cjs/loader:911:12)
    at Function.exec...
```

no Hoisting with const

5. Re-declaring Variables with var, let, and const

- **declaring variables with var:** The variable name is declared twice using var, which allows re-declaration in the same scope. The final console.log(name) prints “Tanmay” as it overwrites the previous value.

```
var name = "Pranjal";
var name = "Tanmay";
console.log(name);
```



Output

Tanmay

- **declaring variables with let:** The variable name is declared with let, which allows reassignment but not re-declaration. The final `console.log(name)` prints “Tanmay” after the reassignment.

```
let name='Pranjal'  
name='Tanmay'  
console.log(name)
```



Output

Tanmay

- **declaring variables with const:** The variable city is declared with const, which does not allow reassignment. The code will throw a Type Error when trying to reassign “Los Angeles” to city.

```
const city = "New York";  
city = "Los Angeles";  
console.log(city)
```



Output

```
Hangup (SIGHUP)  
/home/guest/sandbox/Solution.js:2  
city = "Los Angeles";  
  ^
```

```
TypeError: Assignment to constant variable.  
    at Object.<anonymous> (/home/guest/sandbox/Solution.js:2:6)  
    at Module._compile (node:internal/modules/cjs/loader:1198:14)  
    at Object.Module._extensions..js (node:internal/modules/cjs/loader:1252:10)  
    at Module.load (node:internal/modules/cjs/loader:1076:32)  
    at Function.Module._load (node:internal/modules/cjs/loader:911:12)  
    at Function.executeUserEntryPoint...
```

declaring variables with const



6. Block-level Scope in Loops with let

When using let in a [loop](#), each iteration of the loop creates a new instance of the variable. This is different from var, which shares the

same variable across all iterations.

```
for (let i = 0; i < 3; i++) {  
  console.log(i);  
}  
  
console.log(i);
```



Output

```
Hangup (SIGHUP)  
/home/guest/sandbox/Solution.js:5  
console.log(i);  
      ^  
  
ReferenceError: i is not defined  
    at Object.<anonymous> (/home/guest/sandbox/Solution.js:5:13)  
    at Module._compile (node:internal/modules/cjs/loader:1198:14)  
    at Object.Module._extensions..js (node:internal/modules/cjs/loader:1271:10)  
    at Module.load (node:internal/modules/cjs/loader:1076:32)  
    at Function.Module._load (node:internal/modules/cjs/loader:911:12)  
    at Function.executeUserEntryPoint [a...]
```

Block-level Scope in Loops with let

- let creates a variable i that is only accessible within the loop block.
- Trying to access i outside the loop causes an error since it's not available outside the block where it was declared.

7. Constant Arrays and Objects with const

- **Arrays with const in JavaScript:** The const declaration makes the reference to the numbers array constant, but its contents can still be modified. The code will print [1, 2, 3, 4] before throwing a TypeError when trying to reassign the array.

```
const numbers = [1, 2, 3];  
numbers.push(4);  
console.log(a);  
numbers = [5, 6];
```



Output

```

Hangup (SIGHUP)
/home/guest/sandbox/Solution.js:4
numbers = [5, 6];
      ^

```

```

TypeError: Assignment to constant variable.
    at Object.<anonymous> (/home/guest/sandbox/Solution.js:4:9)
    at Module._compile (node:internal/modules/cjs/loader:1198:14)
    at Object.Module._extensions..js (node:internal/modules/cjs/loader:1252:10)
    at Module.load (node:internal/modules/cjs/loader:1076:32)
    at Function.Module._load (node:internal/modules/cjs/loader:911:12)
    at Function.executeUserEntryPo...

```

Arrays with const in JavaScript

- **Objects with const in JavaScript:** The const declaration makes the reference to the person object constant, but its properties can be modified. The code will print { name: "Pranjal", age: 31 } before throwing a Type Error when trying to reassign the object.

```

const person = { name: "Pranjal", age: 30 };
person.age = 31;
console.log(person);
person = { name: "Nanda" };

```



Output

```

Hangup (SIGHUP)
/home/guest/sandbox/Solution.js:4
person = { name: "Nanda" };
      ^

```

```

TypeError: Assignment to constant variable.
    at Object.<anonymous> (/home/guest/sandbox/Solution.js:4:8)
    at Module._compile (node:internal/modules/cjs/loader:1198:14)
    at Object.Module._extensions..js (node:internal/modules/cjs/loader:1252:10)
    at Module.load (node:internal/modules/cjs/loader:1076:32)
    at Function.Module._load (node:internal/modules/cjs/loader:911:12)
    at Function.execute...

```

Objects with const in JavaScript

Interesting Facts About let, var and const



- **let:** Block-scoped and can be reassigned.
- **var:** Function-scoped and can be reassigned.
- **const:** Block-scoped and cannot be reassigned.
- **let vs var:** let is safer because it's block-scoped, unlike var.

- **const:** Used for values that shouldn't change, but objects/arrays inside can still change.

Differences between var, let, and const

var	let	const
The scope of a <u>var</u> variable is functional or global scope.	The scope of a <u>let</u> variable is block scope.	The scope of a <u>const</u> variable is block scope.
It can be updated and re-declared in the same scope.	It can be updated but cannot be re-declared in the same scope.	It can neither be updated or re-declared in any scope.
It can be declared without initialization.	It can be declared without initialization.	It cannot be declared without initialization.
It can be accessed without initialization as its default value is "undefined".	It cannot be accessed without initialization otherwise it will give 'referenceError'.	It cannot be accessed without initialization, as it cannot be declared without initialization.
These variables are hoisted.	These variables are hoisted but stay in the temporal dead zone until the initialization.	These variables are hoisted but stay in the temporal dead zone until the initialization.

When to Use let and const

var can be tricky because its scope is either global or within a function, which can lead to bugs. To avoid these issues:



- Use **let** when you know a variable's value might change later in your code.

- Use **const** for variables that should never change once you set them.

Using let and const makes your code easier to understand and helps prevent errors caused by unexpected variable changes.

Conclusion

JavaScript statements are fundamental to creating a functional program. Understanding the different types of statements—such as declarations, expressions, conditionals, loops, and exception handling is crucial for writing efficient and effective code. With this knowledge, you can build dynamic web applications and handle various scenarios in a structured way.

[Comment](#)[More info](#)[Campus Training Program](#)

Next Article

How to add File Dropper in
Next.js ?

Similar Reads

Difference between var and let in JavaScript

In the early days of JavaScript, there was only one way of declaring variables and that was using the var keyword. A variable declared with...

14 min read

Difference Between Static and Const in JavaScript

Static variable: A static variable in JavaScript is basically a property of the class which is not used on the object of the class but is used in the class...

12 min read

JavaScript Coding Questions and Answers

JavaScript is the most commonly used interpreted, and scripted Programming language. It is used to make web pages, mobile...

