
JavaScript const – Detailed Notes with document.write()

What is const?

- The const keyword is used to declare **constants** — variables whose value **cannot be reassigned**.
- It must be initialized at the time of declaration.
- It is **block-scoped**, just like let.

Basic Syntax

```
const PI = 3.14;
```

Example 1: Basic Number Constant

```
<script>
  const pi = 3.14159;
  document.write("<b>Example 1:</b><br>");
  document.write("Value of pi: " + pi + "<br><br>");
</script>
```

- const must be initialized.
- Reassignment like pi = 3.15; will cause an error.

Example 2: Missing Initialization (Error)

```
<script>
  // const radius; // Uncommenting this will throw an error
   document.write("<b>Example 2:</b><br>");
   document.write("const radius; // Error: Missing
initializer<br><br>");
</script>
```

A const must have a value when declared.

Example 3: const is Block Scoped

```
<script>
  document.write("<b>Example 3:</b><br>");
  {
    const city = "Pune";
    document.write("Inside block: " + city + "<br>");
  }
  document.write("Outside block: city is not accessible<br><br>");
</script>
```

- Accessible inside {}
- Not accessible outside block — throws an error if accessed

Example 4: const with Arrays

```
<script>
  const colors = ["red", "green"];
  colors.push("blue");
  colors[0] = "yellow";

  document.write("<b>Example 4:</b><br>");
  document.write("Modified Array: " + colors.join(", ") + "<br>");

  // colors = ["black"]; // Error
  document.write("Cannot reassign the array itself<br><br>");
</script>
```

- Elements can be changed
- Whole array cannot be reassigned

Example 5: const with Objects

```
<script>
  const user = { name: "Alice", age: 22 };
  user.age = 23; // Modifying a property is allowed

  document.write("<b>Example 5:</b><br>");
  document.write("Updated User: " + user.name + ", Age: " + user.age
+ "<br>");

  // user = { name: "Bob" }; // Error
  document.write("Cannot reassign the object itself<br><br>");
</script>
```

- You can modify the properties
- You cannot assign a new object

Example 6: Comparison Table

```
<script>
  document.write("<b>Example 6: Summary Table</b><br>");
  document.write("<table border='1' cellpadding='5'>");
  document.write("<tr><th>Feature</th><th>const</th></tr>");
  document.write("<tr><td>Must initialize</td><td>Yes</td></tr>");
  document.write("<tr><td>Reassign value</td><td>No</td></tr>");
  document.write("<tr><td>Block-scoped</td><td>Yes</td></tr>");
  document.write("<tr><td>Can modify
array/object</td><td>Yes</td></tr>");
  document.write("</table><br>");
</script>
```

- Use const when the value should not change.
- Use it for:
  - Configurations
  - Constants
  - Arrays and objects that should not be reassigned

- const ensures **immutability of the binding**, not the content.
- Great for reliable and bug-free code where reassignment should not happen.

Here are **detailed, side-by-side runnable examples of let and var in JavaScript** using document.write() — written clearly without emojis and with practical examples.

JavaScript let and var – Detailed Notes with Examples

---

1. Introduction

Keyword	Scope	Re-declaration	Hoisting	Must Initialize
var	Function	Yes	Yes	No
let	Block	No	No	No

---

2. Basic Declaration and Reassignment

```
<script>
  document.write("<b>Example 1:</b><br>");

  var a = 10;
  let b = 20;

  a = 15;
  b = 25;

  document.write("var a = 10 → a = 15 ➞ a = " + a + "<br>");
  document.write("let b = 20 → b = 25 ➞ b = " + b + "<br><br>");
</script>
```

---

3. Scope: var (Function) vs let (Block)

```
<script>
  document.write("<b>Example 2:</b><br>");

  function scopeTest() {
    if (true) {
      var x = 5;
      let y = 10;
    }
    document.write("var x is accessible: x = " + x + "<br>");
    document.write("let y is not accessible outside block<br><br>");
  }

  scopeTest();
</script>
```

x works because it's function-scoped;
y causes an error if accessed outside {} block.

---

4. Hoisting: var is hoisted, let is not

```
<script>
  document.write("<b>Example 3:</b><br>");

  document.write("var z = " + z + "<br>"); // undefined due to hoisting
  var z = 100;

  // document.write("let w = " + w + "<br>"); // Uncommenting causes
error
  let w = 200;
  document.write("let w is declared after use: Error if accessed
earlier<br><br>");
</script>
```

---

5. Re-declaration in Same Scope

```
<script>
  document.write("<b>Example 4:</b><br>");
  var a = 10;
  var a = 20; // Allowed

  document.write("var can be re-declared: a = " + a + "<br>");

  let b = 30;
  // let b = 40; // Uncommenting causes error

  document.write("let cannot be re-declared in same scope<br><br>");
</script>
```

---

6. Loop Scope Example

```
<script>
  document.write("<b>Example 5:</b><br>");

  for (var i = 0; i < 3; i++) {
    document.write("Inside loop (var): i = " + i + "<br>");
  }
  document.write("Outside loop (var): i = " + i + "<br>");

  for (let j = 0; j < 3; j++) {
    document.write("Inside loop (let): j = " + j + "<br>");
  }
  document.write("Outside loop (let): j is not accessible<br><br>");
</script>
```

---

7. Comparison Summary

```
<script>
  document.write("<b>Example 6: Summary Table</b><br>");
  document.write("<table border='1' cellpadding='5'>");

document.write("<tr><th>Feature</th><th>var</th><th>let</th></tr
>");

document.write("<tr><td>Scope</td><td>Function</td><td>Block</
td></tr>");
```

```
  document.write("<tr><td>Hoisting</td><td>Yes
(undefined)</td><td>No</td></tr>");
  document.write("<tr><td>Re-
declaration</td><td>Allowed</td><td>Not allowed</td></tr>");
  document.write("<tr><td>Global
Leakage</td><td>Possible</td><td>Safer</td></tr>");
  document.write("</table><br>");
</script>
```

- Use let for block-level variables to avoid scope issues.
- Avoid var unless needed for legacy support.
- let is preferred for safety, predictability, and modern code style.
