1. [JavaScript Basics](https://www.geeksforgeeks.org/javascript/" \l "javascript-basics)

9. [JavaScript First Program Hello world](https://www.geeksforgeeks.org/javascript-course-printing-hello-world-in-javascript/)

Involves displaying the phrase “Hello World” using simple methods like console.log(), document.write(), or alert(), introducing beginners to fundamental JavaScript syntax and output techniques.

**Using console.log() method**

The [console.log() method](https://www.geeksforgeeks.org/javascript-console-log-method/) prints the message to the browser console. This approach is mainly used for debugging purposes and checking outputs while developing code.

**Example:** In this example, we will print the legendary “Hello World” in the window console.

// Using console.log

console.log('Hello World');

**Output**

Hello World

**Note** To see the output in browser open file .html file -> right click in webpage -> inspect element or F12 -> go to console tab – here you will find the output “Hello World”.

**Using document.write() Method**

Using the [document.write() method](https://www.geeksforgeeks.org/html-dom-write-method/" \t "_blank) in JavaScript allows you to display “Hello World” directly on the webpage. This method is straightforward and places the text within the HTML document. However, it’s generally only suitable for simple demos or initial page load.

**Example:** In this example, we will print the “Hello World” in the HTML document.

// Using document.write

document.write('Hello World');

**Output:**

Hello world

**Using alert() Method**

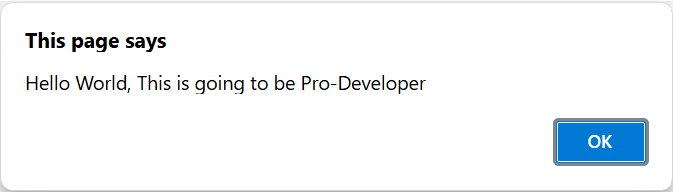
The [alert() method](https://www.geeksforgeeks.org/html-dom-window-alert-method/) displays a popup alert box with the message. It’s useful for simple notifications or warnings but should be used sparingly as it disrupts user interaction.

**Example: I**n this example, we will print the “Hello World” on the browser window with some message or warning.

// Using alert

alert("Hello World, This is going to be Pro-Developer");

**Output:**



Each of the above methods has different ways of outputting the content. Though ‘document.write()’ is used when we want to print the content onto the document which is the HTML Document. Also ‘console.log()’ is mainly used when we are debugging JavaScript code and the ‘alert()’ is used to show an alert box on the browser window with some message or warning.

**10. JavaScript Data Types:**

JavaScript supports multiple data types. JavaScript data types are broadly categorized into **primitive and non-primitive types. The primitive data types include Number, String, Boolean, Null, Undefined, and Symbol. Non-primitive types include Object, Array, and Function.**

The latest ECMAScript standard defines eight data types Out of which seven data types are **Primitive(predefined)** and one **complex or Non-Primitive**.

**Primitive Data Types**

The predefined data types provided by JavaScript language are known as primitive data types. Primitive data types are also known as in-built data types.

|  |  |
| --- | --- |
| **Type** | **Description** |
| **Number** | JavaScript numbers are always stored in double-precision 64-bit binary format IEEE 754. |
| **String** | JavaScript Strings are made up of a list of characters, essentially an array of characters. |
| **Boolean** | Represents a logical entity and can have two values: true or false. |
| **Null** | This type has only one value: null. |
| **Undefined** | A variable that has not been assigned a value is undefined. |
| **Symbol** | Symbols return unique identifiers that can be used as property keys in objects without colliding with other keys. |
| **BigInt** | BigInt is a built-in object providing a way to represent whole numbers larger than 253-1. |

This table summarizes the basic data types in JavaScript along with their descriptions.

**Non-Primitive Data Types:**

The data types that are derived from primitive data types of the JavaScript language are known as non-primitive data types. It is also known as derived data types or reference data types.

* [**Object:**](https://www.geeksforgeeks.org/objects-in-javascript) It is the most important data type and forms the building blocks for modern JavaScript.

**JavaScript Primitive Data Types Examples:**

**Number:**

The number type in JavaScript contains both integer and floating-point numbers. Besides these numbers, we also have some ‘special-numbers’ in javascript that are: **‘Infinity’, ‘-Infinity’, and ‘NaN’. Infinity basically represents the mathematical ‘?’. The ‘NaN’ denotes a computational error.**

let num = 2; **// Integer**

let num2 = 1.3; **// Floating point number**

let num3 = Infinity; **// Infinity**

let num4 = 'something here too'/2; **// NaN**

**String:**

**A String in JavaScript is basically a series of characters that are surrounded by quotes.** There are three types of quotes in JavaScript, which are:

let str = **"Hello There";**

let str2 = **'Single quotes works fine';**

let phrase = **`can embed ${str}`;**

There’s no difference between ‘single’ and “double” quotes in JavaScript.

Backticks provide extra functionality as with their help of them we can embed variables inside them.

let name = **"Mukul";**

// embed a variable

alert( **`Hello, ${name}!`** ); // Hello, Mukul!

**Boolean:**

**The Boolean type has only two values: true and false.** This data type is used to store yes/no type of values: True means “Yes, Correct”, and False means “No, Incorrect”.

let isCoding = True; // yes

let isOld = False; // no

**NULL:**

The special NULL value does not belong to any of the default data types. It forms a separate type of its own which contains only the null value:

let age = null;

The ‘null’ data type basically defines a special value that represents ‘nothing’, ’empty’, or ‘value unknown’.

**Undefined** Just like null, Undefined makes its own type. The meaning of undefined is ‘value is not assigned’.

let x;

console.log(x); // undefined

|  |  |
| --- | --- |
| **NULL** | **UNDEFINED** |
| Null is an assignment value, meaning that a variable has been declared and given the value of null. | Undefined means a variable has been declared but has not yet been assigned a value. |
| **let** y = null;  console.**log**(y); *// logs 'null'* | **let** x;  console.**log**(x); *// logs 'undefined'* |
| console.log(typeof a); *//****logs 'object'*** | console.log(typeof z);*//****logs 'undefined'*** |
| Explicitly we need to assign to null. | Automatically assigned to undefined. |
| null values are preserved during JSON serialization (e.g., {“key”: null} ). | undefined values are omitted during serialization. |

**Symbol:**

**Symbols are new primitive built-in object types introduced as part of ES6.** Symbols return unique identifiers that can be used to add unique property keys to an object that won’t collide with keys of any other code that might add to the object. They are used as object properties that cannot be recreated. It basically helps us to enable encapsulation or information hiding.

let symbol1 = Symbol("Geeks")

let symbol2 = Symbol("Geeks")

// Each time Symbol() method

// is used to create new global Symbol

console.log(symbol1 == symbol2); // False

**BigInt:**

BigInt is a built-in object in JavaScript that provides a way to represent whole numbers larger than 253-1. The largest number that JavaScript can reliably represent with the Number primitive is 253-1, which is represented by the MAX\_SAFE\_INTEGER constant.

let bigBin = BigInt("0b1010101001010101001111111111111111");

// 11430854655n

console.log(bigBin);

**JavaScript Non-Primitive Data Types Examples:**

**Object:**

JavaScript objects are fundamental data structures used to store collections of data. They consist of key-value pairs and can be created using curly braces {} or the new keyword. Understanding objects is crucial, as everything in JavaScript is essentially an object.

**Object creation:**

**Using the “object constructor” syntax:**

let person = new Object();

**Using the “object literal” syntax:**

let person = {}; //

Both these methods are correct, though it’s totally your call what to choose. We can also put properties inside an Object.

**11. JavaScript Variables**

**Variables**are used to store data in JavaScript. Variables are used to store reusable values. The values of the variables are allocated using the assignment operator(“=”).

JavaScript assignment operator is equal (=) which assigns the value of the right-hand operand to its left-hand operand.

y = "Hello"

**JavaScript Identifiers**

JavaScript variables must have unique names. These names are called Identifiers.

**Basic rules to declare a variable in JavaScript:**

* These are case-sensitive
* Can only begin with a letter, underscore(“\_”) or “$” symbol
* It can contain letters, numbers, underscore, or “$” symbol
* A variable name cannot be a reserved keyword.

**Operators in JavaScript with Example:**

**1. Arithmetic Operators**

* **Addition (+)**:

let a = 5;

let b = 10;

let sum = a + b; // 15

* **Subtraction (-)**:

let diff = b - a; // 5

* **Multiplication (\*)**:

let product = a \* b; // 50

* **Division (/)**:

let quotient = b / a; // 2

* **Modulus (%)**:

let remainder = b % a; // 0

* **Increment (++)**:

let x = 5;

x++; // x becomes 6

* **Decrement (--)**:

x--; // x becomes 5 again

* **Exponentiation (\*\*)**:

let power = 2 \*\* 3; // 8

**2. Assignment Operators**

* **Assign (=)**:

let c = 5;

* **Add and assign (+=)**:

c += 5; // c becomes 10

* **Subtract and assign (-=)**:

c -= 2; // c becomes 8

* **Multiply and assign (\*=)**:

c \*= 2; // c becomes 16

* **Divide and assign (/=)**:

c /= 4; // c becomes 4

* **Modulus and assign (%=)**:

c %= 3; // c becomes 1

* **Exponentiation and assign (\*\*=)**:

c \*\*= 3; // c becomes 1 (because 1 raised to any power is still 1)

**3. Comparison Operators**

* **Equal to (==)**:

let isEqual = (5 == "5"); // true (type coercion happens)

* **Strict equal to (===)**:

let isStrictEqual = (5 === "5"); // false (no type coercion)

* **Not equal to (!=)**:

let isNotEqual = (5 != "5"); // false

* **Strict not equal to (!==)**:

let isStrictNotEqual = (5 !== "5"); // true

* **Greater than (>)**:

let isGreater = (10 > 5); // true

* **Less than (<)**:

let isLess = (10 < 5); // false

* **Greater than or equal to (>=)**:

let isGreaterOrEqual = (10 >= 10); // true

* **Less than or equal to (<=)**:

let isLessOrEqual = (10 <= 5); // false

**4. Logical Operators**

* **Logical AND (&&)**:

let andResult = (true && false); // false

* **Logical OR (||)**:

let orResult = (true || false); // true

* **Logical NOT (!)**:

let notResult = !true; // false

**5. Bitwise Operators**

* **Bitwise AND (&)**:

let andBitwise = 5 & 1; // 1 (binary: 0101 & 0001 = 0001)

* **Bitwise OR (|)**:

let orBitwise = 5 | 1; // 5 (binary: 0101 | 0001 = 0101)

* **Bitwise XOR (^)**:

let xorBitwise = 5 ^ 1; // 4 (binary: 0101 ^ 0001 = 0100)

* **Bitwise NOT (~)**:

let notBitwise = ~5; // -6 (binary: ~0101 = 1010, which is -6 in two's complement)

* **Left shift (<<)**:

let leftShift = 5 << 1; // 10 (binary: 0101 << 1 = 1010)

* **Right shift (>>)**:

let rightShift = 5 >> 1; // 2 (binary: 0101 >> 1 = 0010)

* **Unsigned right shift (>>>)**:

let unsignedRightShift = -5 >>> 1; // 2147483645 (shifts the bits to the right and fills with zeros)

**6. String Operators**

* **Concatenation (+)**:

let str1 = "Hello";

let str2 = "World";

let greeting = str1 + " " + str2; // "Hello World"

**7. Ternary Operator**

* **Ternary (? :)**:

let age = 18;

let canVote = (age >= 18) ? "Yes" : "No"; // "Yes"

**8. Type Operators**

* **typeof**:

let type = typeof 123; // "number"

* **instanceof**:

let date = new Date();

let isDate = date instanceof Date; // true

**9. Unary Operators**

* **Unary Plus (+)**:

let num = +"123"; // Converts the string "123" to the number 123

* **Unary Negation (-)**:

let neg = -123; // -123

* **Logical NOT (!)**:

let isFalse = !true; // false

* **Bitwise NOT (~)**:

let complement = ~5; // -6

* **delete**:

let obj = {name: "Alice"};

delete obj.name; // Deletes the "name" property from obj

* **void**:

void 0; // Returns undefined

**10. Relational Operators**

* **in**:

let obj = {name: "Alice"};

let hasName = "name" in obj; // true

* **instanceof**:

let arr = [];

let isArray = arr instanceof Array; // true

**11. Comma Operator**

* **Comma (,)**:

let result = (5, 10, 15); // 15 (returns the last value)

These examples demonstrate the various operators in JavaScript and how they can be used in code.

**12. Nullish Coalescing Assignment (??=)**

This operator is represented by **x ??= y**and it is called Logical nullish assignment operator. Only if the value of **x** is **nullish** then the value of y will be assigned to **x**thatmeansif the value of x is **null**or**undefined**then the value of y will be assigned to x.

logical nullish assignment is represented as **x ??= y**, this is derived by two operators nullish coalescing operator and assignment operator we can also write it as **x ?? (x = y)**. Now javascript checks the **x**first, if it is **nullish** then the value of **y** will be assigned to **x**.

**Syntax :**

x ??= y // Means : x ?? (x = y)

**Example 1 :**

let x = 12;

let y = null;

let z = 13;

// The value of x will become

// unchanged because x is not nullish.

x ??= z;

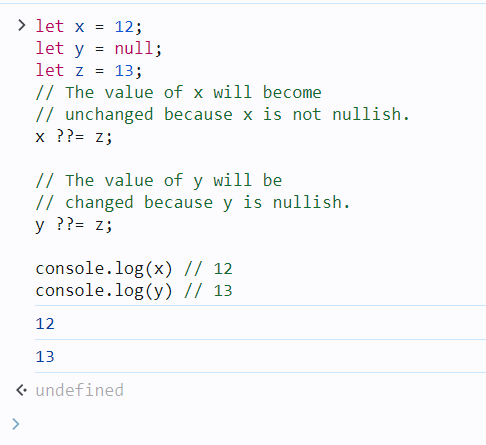
// The value of y will be

// changed because y is nullish.

y ??= z;

console.log(x) // 12

console.log(y) // 13



**Example 2:**

let x = {

name : "Ram"

}

// The value of name will remain

// unchanged because x.name is not nullish

x.name ??= "Shyam";

// There is no any property named age in object x .

// So the value of x.age will be

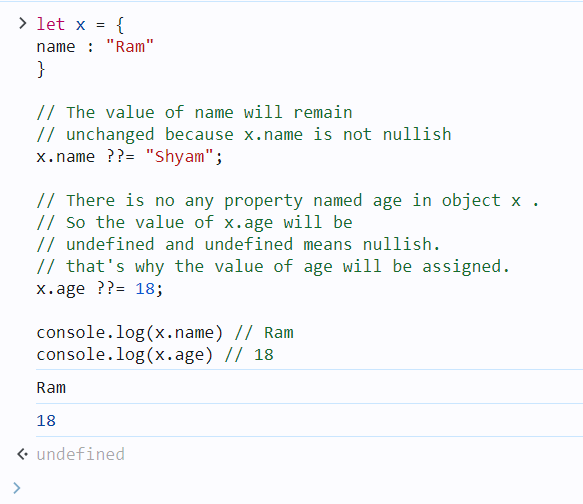
// undefined and undefined means nullish.

// that's why the value of age will be assigned.

x.age ??= 18;

console.log(x.name) // Ram

console.log(x.age) // 18

****

**Example 3:**

<h1>Hello Coalescing</h1>

<p id="print\_arr"></p>

<script>

    let arr = [1, 2, "apple", null, undefined, []]

    // Replace each nullish values with "RAM"

    arr.forEach((item, index)=>{

    arr[index] ??= "RAM"

        })

        document.getElementById("print\_arr")

    .innerText = arr.toString();

        //console.log(arr)

</script>



**Variable Declaration:**

**Var :**

The var keyword in JavaScript is used to declare a variable. It was the primary way to declare variables before the introduction of let and const in ES6. Variables declared with var have some unique characteristics that distinguish them from let and const.

**Characteristics of var:**

1. **Function Scope**: var is function-scoped, meaning that if a variable is declared inside a function, it is only accessible within that function. However, it is not block-scoped (i.e., var ignores block scopes such as loops and if statements).
2. **Hoisting**: Variables declared with var are hoisted to the top of their scope. This means the declaration is moved to the top of its scope, but the initialization stays in place.
3. **Redeclaration**: You can redeclare a variable with var without causing an error.
4. **Global Object Property**: In the global scope, var declarations create properties on the global object (e.g., window in browsers).

**Syntax:**

var variableName = value;

**Examples:**

**1. Basic Declaration and Assignment:**

var x = 10; // Declare and initialize a variable

console.log(x); // 10

**2. Function Scope:**

function testVar() {

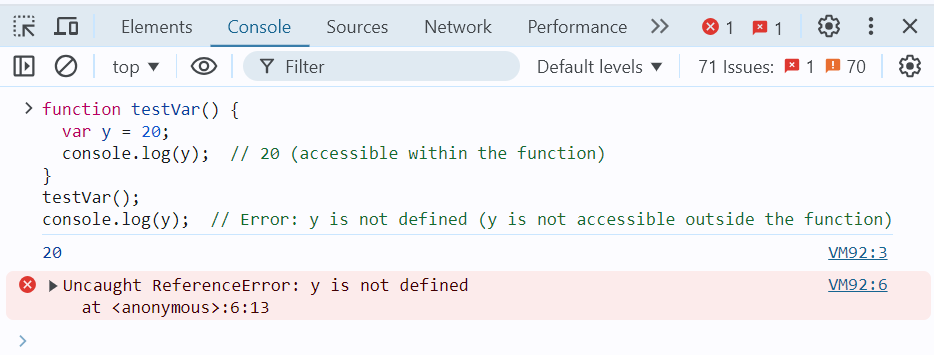
var y = 20;

console.log(y); // 20 (accessible within the function)

}

testVar();

console.log(y); // Error: y is not defined (y is not accessible outside the function)



**3. Hoisting:**

console.log(a); // undefined (the declaration is hoisted, but not the assignment)

var a = 5;

console.log(a); // 5

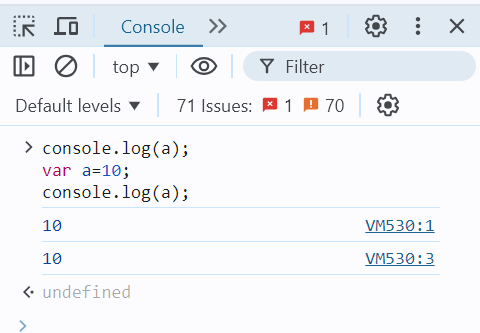
This code behaves as though it was written like this:

var a;

console.log(a); // undefined

a = 5;

console.log(a); // 5



**4. Ignoring Block Scope:**

if (true) {

var z = 30;

}

console.log(z); // 30 (accessible outside the block, because var is not block-scoped)

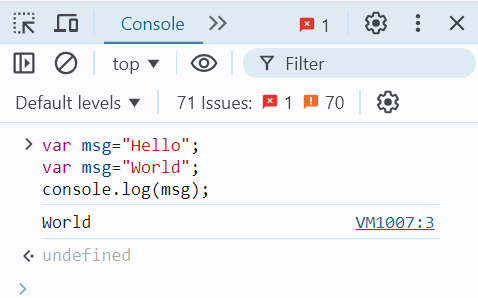


**5. Redeclaration:**

var message = "Hello";

var message = "World";

console.log(message); // "World" (no error on redeclaration)



**Let:**

The let keyword in JavaScript was introduced in ES6 (ECMAScript 2015) and is used to declare variables. Unlike var, let provides block-scoping, which makes it a more predictable and safer way to declare variables in modern JavaScript.

**Characteristics of let:**

1. **Block Scope**: Variables declared with let are confined to the block in which they are defined. A block is typically defined by {} (e.g., within an if, for, or function).
2. **No Hoisting with Initialization**: Although let variables are hoisted to the top of their block, they are not initialized until their declaration is encountered in the code. This leads to a **"Temporal Dead Zone (TDZ)"** where accessing the variable before its declaration results in an error.
3. **No Redeclaration**: Variables declared with let cannot be redeclared within the same scope. This helps prevent accidental overwriting of variables.

**Syntax:**

let variableName = value;

**Examples:**

**1. Basic Declaration and Assignment:**

let x = 10; // Declare and initialize a variable

console.log(x); // 10

**2. Block Scope:**

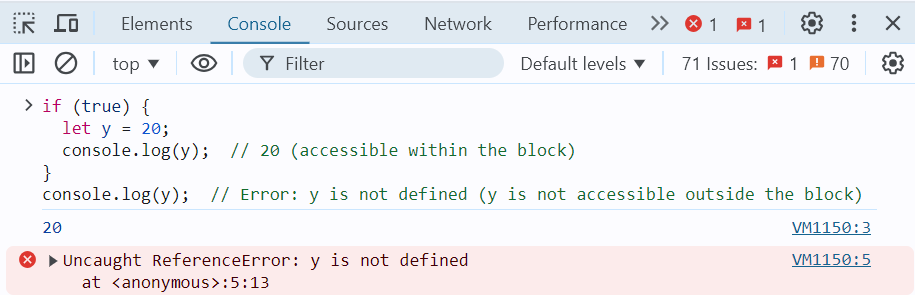
if (true) {

let y = 20;

console.log(y); // 20 (accessible within the block)

}

console.log(y); // Error: y is not defined (y is not accessible outside the block)



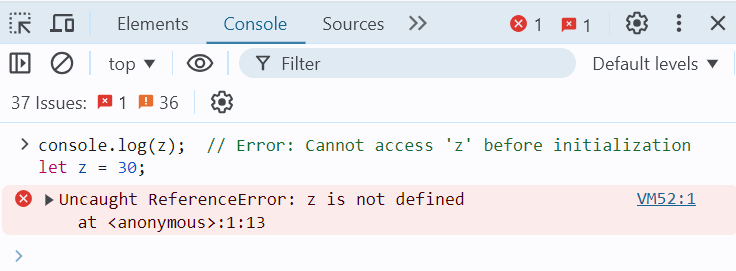
**3. No Hoisting with Initialization:**

Unlike var, let variables are hoisted but not initialized, leading to a temporal dead zone (TDZ).

console.log(z); // Error: Cannot access 'z' before initialization

let z = 30;

The above code throws an error because z is in the temporal dead zone until the let z = 30; line is executed.



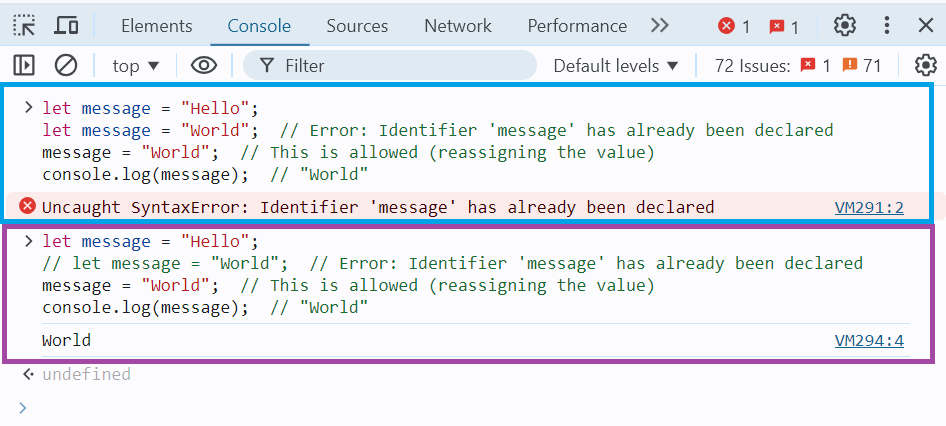
**4. No Redeclaration:**

let message = "Hello";

// let message = "World"; // Error: Identifier 'message' has already been declared

message = "World"; // This is allowed (reassigning the value)

console.log(message); // "World"



**5. Using let in Loops:**

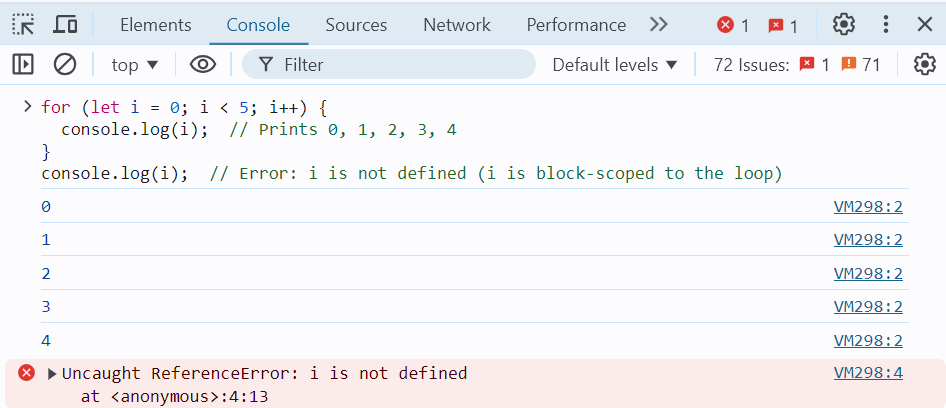
let is commonly used in loops, especially in scenarios where each iteration needs its own scope.

for (let i = 0; i < 5; i++) {

console.log(i); // Prints 0, 1, 2, 3, 4

}

console.log(i); // Error: i is not defined (i is block-scoped to the loop)



The let keyword is generally preferred over var in modern JavaScript because of its block-scoping behaviour, lack of redeclaration, and safer hoisting. It reduces the risk of errors in code by limiting the scope of variables and ensuring that they are not accidentally redeclared. This makes let a more predictable and reliable way to declare variables.

**Const:**

The const keyword in JavaScript, introduced in ES6 (ECMAScript 2015), is used to declare variables that are **constant**. Once a variable is assigned a value with const, it cannot be reassigned. const is typically used to declare variables that should not change throughout the program, such as constants or configuration values.

**Characteristics of const:**

1. **Block Scope**: Similar to let, const is block-scoped, meaning it is only accessible within the block in which it is declared.
2. **No Reassignment**: Variables declared with const cannot be reassigned a new value after they are initialized.
3. **Must be Initialized**: A const variable must be initialized at the time of declaration. Unlike let, you cannot declare a const variable without assigning a value.
4. **Mutable Objects**: While you cannot reassign a const variable, if the variable holds an object (including arrays), the contents of the object or array can still be modified.

**Syntax:**

const variableName = value;

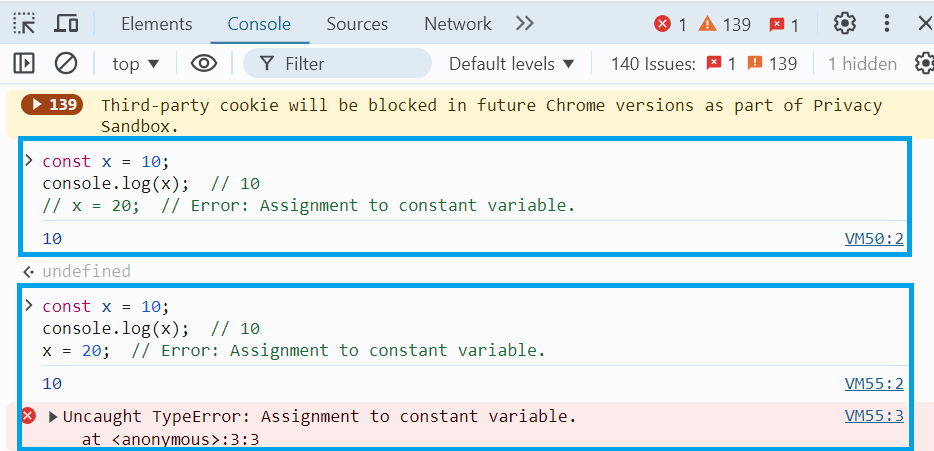
**Examples:**

**1. Basic Declaration and Assignment:**

const x = 10;

console.log(x); // 10

// x = 20; // Error: Assignment to constant variable.



**2. Block Scope:**

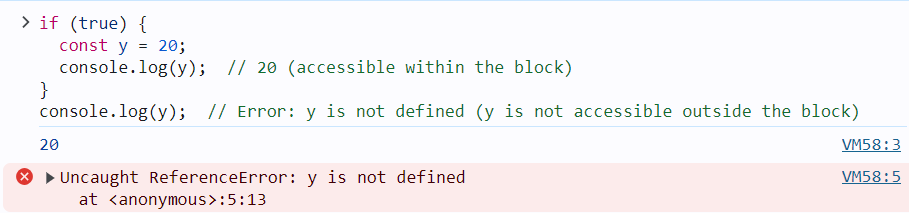
if (true) {

const y = 20;

console.log(y); // 20 (accessible within the block)

}

console.log(y); // Error: y is not defined (y is not accessible outside the block)



**3. Mutable Objects and Arrays:**

Even though const prevents reassignment, the contents of objects and arrays can still be modified:

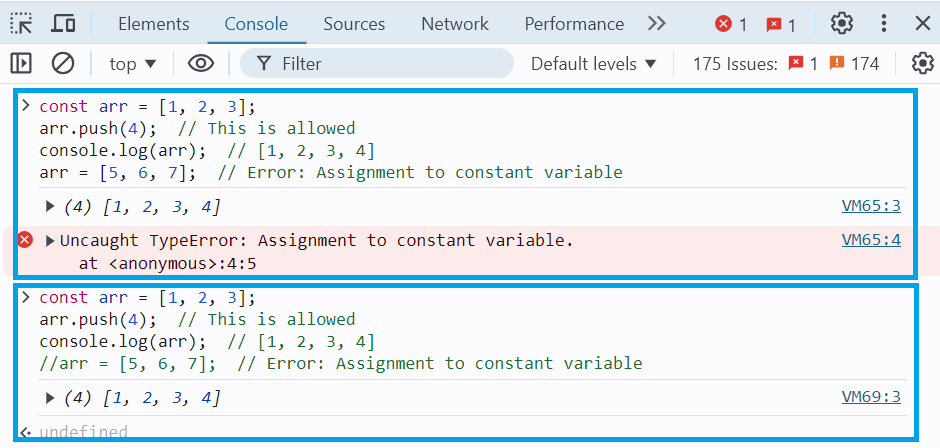
* **Array Example**:

const arr = [1, 2, 3];

arr.push(4); // This is allowed

console.log(arr); // [1, 2, 3, 4]

// arr = [5, 6, 7]; // Error: Assignment to constant variable



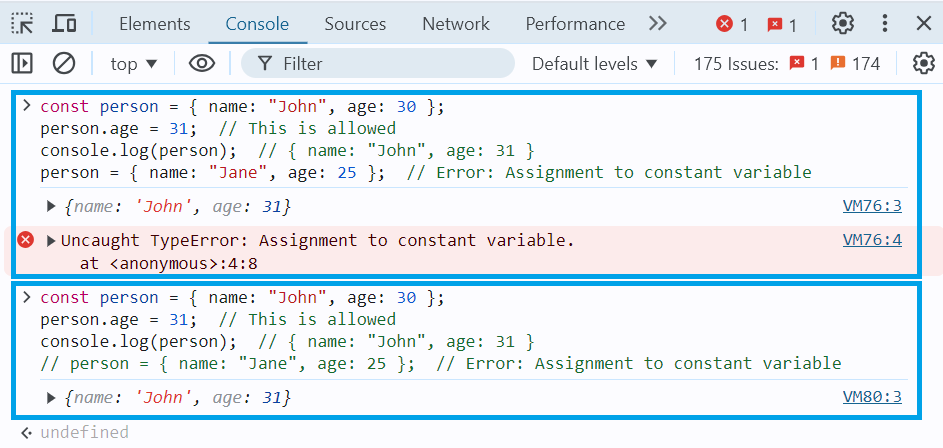
* **Object Example**:

const person = { name: "John", age: 30 };

person.age = 31; // This is allowed

console.log(person); // { name: "John", age: 31 }

// person = { name: "Jane", age: 25 }; // Error: Assignment to constant variable



**4. Const in Loops:**

You can use const in loops, but only if the variable does not need to be reassigned. For example, const can be used inside a loop for iteration variables in a for...of loop:

const arr = [10, 20, 30];

for (const num of arr) {

console.log(num); // Prints 10, 20, 30

}

However, const is not suitable for traditional for loops where the loop counter is expected to change:

// This will throw an error because i is being reassigned in every iteration

// for (const i = 0; i < 5; i++) {

// console.log(i);

// }



The const keyword is ideal for declaring variables that should not be reassigned. However, it does not make the variable itself immutable if it holds an object or an array — only the reference to the variable is constant. For values that need to remain constant throughout the program, const is the best choice, and it is generally preferred for readability and clarity in modern JavaScript.

1. JavaScript Control Flow Statements