**Variables, Data Types and Operators in JavaScript | Web Development Tutorials #48**

From the last tutorials, we are discussing JavaScript in detail. Moving on, here we will see what are variables in JavaScript and how to create them. The JavaScript variables are the containers for storing data values. Make a new file as *tut48.html*and add an instant boilerplate to get the basic HTML code.

To understand it, initially, we have to add some HTML code to get started as follows-

<body>

<div class="container">

<h1>This is a heading</h1>

<div class="content">

<p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Laborum atque laudantium vero tenetur repellendus consequuntur incidunt consectetur deleniti, reprehenderit dolores reiciendis aperiam ducimus aliquid, fugiat ipsum, corporis praesentium quibusdam exercitationem ex unde quae libero odio tempora. Voluptatibus odio molestiae esse unde quibusdam fuga accusantium facere quae, eum error asperiores itaque hic tempora temporibus illo vitae provident ad debitis libero dolorum dignissimos corporis dolore. Quia maxime quidem velit?</p>

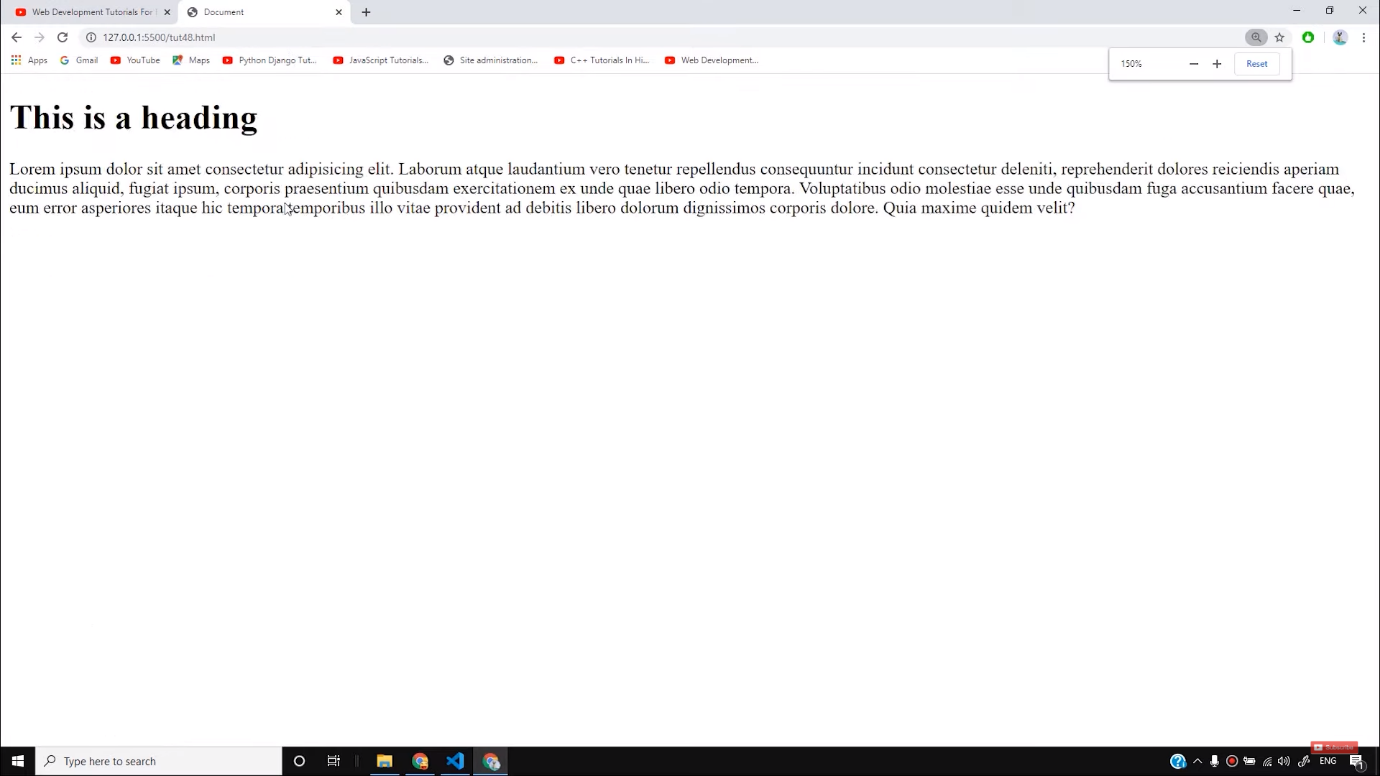
</div>

</div>

</body>

Copy

The result of the HTML code will look as follows-



As discussed earlier, variables are the data that stores value. Therefore, in the below example **a**and **b**are the variables that store integer and String values respectively.

var a = 78;

var b = "Harry";

Copy

To see the output, we have to write-

console.log(a);

console.log(b);

Copy

The result of both the variables will be displayed in the console tab of the browser. These types of variables are known as **dynamic typing** as you do not require to identify the data type.

If you are a web developer do not deep dive into the knowledge of core programming concepts because it can divert you from becoming a successful web developer. Therefore, it is recommended initially, do not get in the concepts of Data Structures or OOPS. Once you get the basic command over the JavaScript then you can move further.

Let us now understand the **operators in JavaScript.**There are two types of operators present in JavaScript- **Binary Operators**and **Unary Operators**. Unary Operators work only on 1 operand. For example, **3+4**. On the other hand, Binary operators work only on 2 operands. For example,**x= x+6**. Here ‘=’ and ‘+’ are two operands.

Let us now understand different types of operators with the help of examples-

var num1 = 2;

var num2 = 9;

// Arithmetic operators in action in JavaScript

console.log("The value of num1 + num2 is "+ (num1 + num2));

console.log("The value of num1 - num2 is "+ (num1 - num2));

console.log("The value of num1 \* num2 is "+ (num1 \* num2));

console.log("The value of num1 / num2 is "+ (num1 / num2));

console.log("The value of num1 \*\* num2 is "+ (num1 \*\* num2));

console.log("The value of num1++ is "+ (num1++));

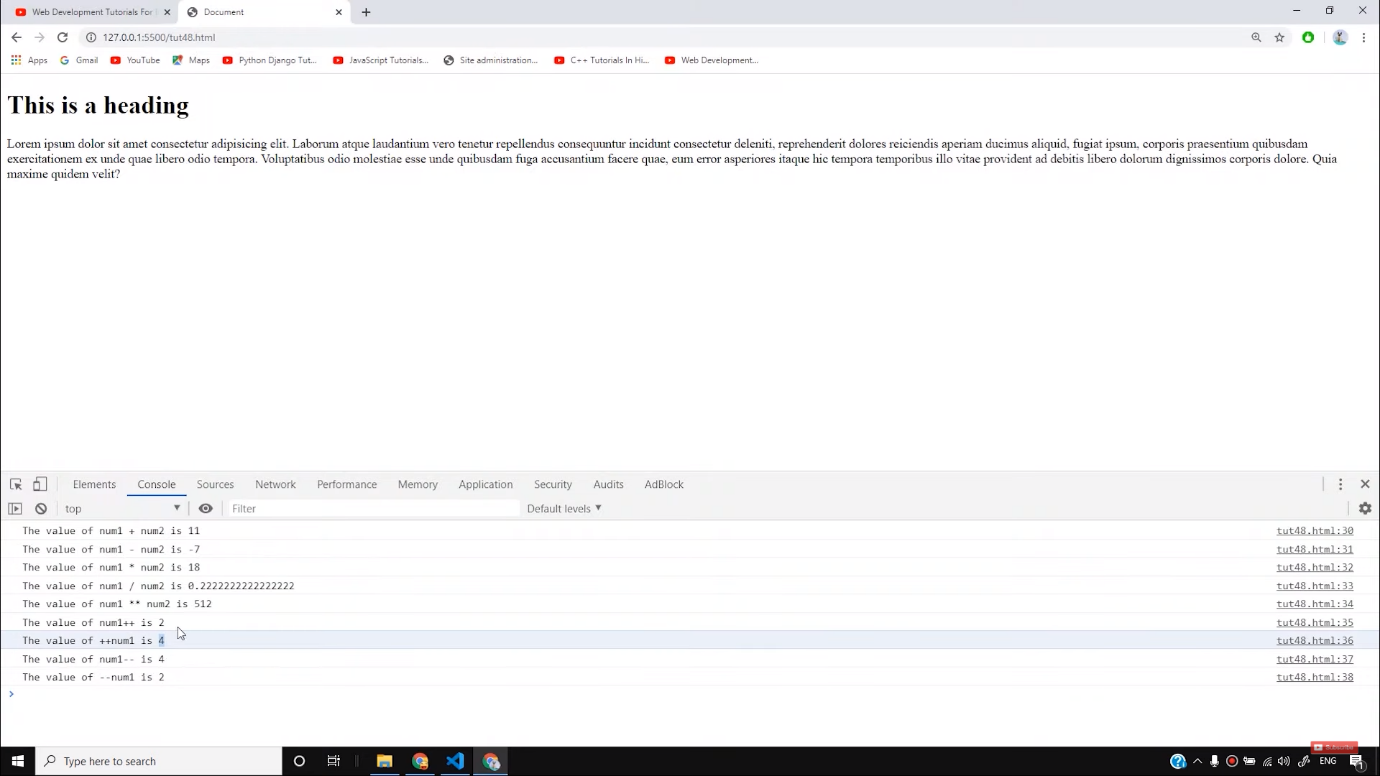
console.log("The value of ++num1 is "+ (++num1));

console.log("The value of num1-- is "+ (num1--));

console.log("The value of --num1 is "+ (--num1));

Copy

The output of all the above arithmetic expressions is shown here-



I hope you must understand the basic expressions of how to use variables in JavaScript. In the next upcoming tutorials, we will see these operations on String variables. Till then stay with the tutorials and keep practicing.

**Code as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

<script>

var a = 78;

var b = "Harry";

c = 34.55;

// console.log(c);

// Operators in JavaScript

// Operand - entities on which operators operate

// In 3 + 4 '+' is the operator and 3,4 are operands

// 1. unary operator - It has single operand (x = -x)

// Examples of unary operator

c = -c;

// console.log(c);

// 2. binary operator - It has two operand (x = x+6)

// Examples of binary operator

c = 456+8;

// console.log(c);

var num1 = 2;

var num2 = 9;

// Arithmetic operators in action in JavaScript

console.log("The value of num1 + num2 is "+ (num1 + num2));

console.log("The value of num1 - num2 is "+ (num1 - num2));

console.log("The value of num1 \* num2 is "+ (num1 \* num2));

console.log("The value of num1 / num2 is "+ (num1 / num2));

console.log("The value of num1 \*\* num2 is "+ (num1 \*\* num2));

console.log("The value of num1++ is "+ (num1++));

console.log("The value of ++num1 is "+ (++num1));

console.log("The value of num1-- is "+ (num1--));

console.log("The value of --num1 is "+ (--num1));

</script>

</head>

<body>

<div class="container">

<h1>This is a heading</h1>

<div class="content">

<p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Laborum atque laudantium vero tenetur repellendus consequuntur incidunt consectetur deleniti, reprehenderit dolores reiciendis aperiam ducimus aliquid, fugiat ipsum, corporis praesentium quibusdam exercitationem ex unde quae libero odio tempora. Voluptatibus odio molestiae esse unde quibusdam fuga accusantium facere quae, eum error asperiores itaque hic tempora temporibus illo vitae provident ad debitis libero dolorum dignissimos corporis dolore. Quia maxime quidem velit?</p>

</div>

</div>

</body>

</html>

JavaScript Syntax

[❮ Previous](https://www.w3schools.com/js/js_statements.asp)[Next ❯](https://www.w3schools.com/js/js_comments.asp)

JavaScript syntax is the set of rules, how JavaScript programs are constructed:

// How to create variables:  
var x;  
let y;  
  
// How to use variables:  
x = 5;  
y = 6;  
let z = x + y;

# JavaScript Variables

[❮ Previous](https://www.w3schools.com/js/js_comments.asp)[Next ❯](https://www.w3schools.com/js/js_let.asp)

### **4 Ways to Declare a JavaScript Variable:**

* Using var
* Using let
* Using const
* Using nothing

## **What are Variables?**

Variables are containers for storing data (storing data values).

In this example, x, y, and z, are variables, declared with the var keyword:

### **Example**

var x = 5;  
var y = 6;  
var z = x + y;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables)

In this example, x, y, and z, are variables, declared with the let keyword:

### **Example**

let x = 5;  
let y = 6;  
let z = x + y;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_let)

In this example, x, y, and z, are undeclared variables:

### **Example**

x = 5;  
y = 6;  
z = x + y;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_undeclared)

From all the examples above, you can guess:

* x stores the value 5
* y stores the value 6
* z stores the value 11

## **When to Use JavaScript var?**

Always declare JavaScript variables with var,let, orconst.

The var keyword is used in all JavaScript code from 1995 to 2015.

The let and const keywords were added to JavaScript in 2015.

If you want your code to run in older browser, you must use var.

## **When to Use JavaScript const?**

If you want a general rule: always declare variables with const.

If you think the value of the variable can change, use let.

In this example, price1, price2, and total, are variables:

### **Example**

const price1 = 5;  
const price2 = 6;  
let total = price1 + price2;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_total)

The two variables price1 and price2 are declared with the const keyword.

These are constant values and cannot be changed.

The variable total is declared with the let keyword.

This is a value that can be changed.

## **Just Like Algebra**

Just like in algebra, variables hold values:

let x = 5;  
let y = 6;

Just like in algebra, variables are used in expressions:

let z = x + y;

From the example above, you can guess that the total is calculated to be 11.

## **Note**

Variables are containers for storing values.

ADVERTISEMENT

## **JavaScript Identifiers**

All JavaScript **variables** must be **identified** with **unique names**.

These unique names are called **identifiers**.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

The general rules for constructing names for variables (unique identifiers) are:

* Names can contain letters, digits, underscores, and dollar signs.
* Names must begin with a letter
* Names can also begin with $ and \_ (but we will not use it in this tutorial)
* Names are case sensitive (y and Y are different variables)
* Reserved words (like JavaScript keywords) cannot be used as names

## **Note**

JavaScript identifiers are case-sensitive.

## **The Assignment Operator**

In JavaScript, the equal sign (=) is an "assignment" operator, not an "equal to" operator.

This is different from algebra. The following does not make sense in algebra:

x = x + 5

In JavaScript, however, it makes perfect sense: it assigns the value of x + 5 to x.

(It calculates the value of x + 5 and puts the result into x. The value of x is incremented by 5.)

## **Note**

The "equal to" operator is written like == in JavaScript.

## **JavaScript Data Types**

JavaScript variables can hold numbers like 100 and text values like "John Doe".

In programming, text values are called text strings.

JavaScript can handle many types of data, but for now, just think of numbers and strings.

Strings are written inside double or single quotes. Numbers are written without quotes.

If you put a number in quotes, it will be treated as a text string.

### **Example**

const pi = 3.14;  
let person = "John Doe";  
let answer = 'Yes I am!';

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_types)

## **Declaring a JavaScript Variable**

Creating a variable in JavaScript is called "declaring" a variable.

You declare a JavaScript variable with the var or the let keyword:

var carName;

or:

let carName;

After the declaration, the variable has no value (technically it is undefined).

To **assign** a value to the variable, use the equal sign:

carName = "Volvo";

You can also assign a value to the variable when you declare it:

let carName = "Volvo";

In the example below, we create a variable called carName and assign the value "Volvo" to it.

Then we "output" the value inside an HTML paragraph with id="demo":

### **Example**

<p id="demo"></p>  
  
<script>  
let carName = "Volvo";  
document.getElementById("demo").innerHTML = carName;  
</script>

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_create)

## **Note**

It's a good programming practice to declare all variables at the beginning of a script.

## **One Statement, Many Variables**

You can declare many variables in one statement.

Start the statement with let and separate the variables by **comma**:

### **Example**

let person = "John Doe", carName = "Volvo", price = 200;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_multi)

A declaration can span multiple lines:

### **Example**

let person = "John Doe",  
carName = "Volvo",  
price = 200;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_multiline)

## **Value = undefined**

In computer programs, variables are often declared without a value. The value can be something that has to be calculated, or something that will be provided later, like user input.

A variable declared without a value will have the value undefined.

The variable carName will have the value undefined after the execution of this statement:

### **Example**

let carName;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_undefined)

## **Re-Declaring JavaScript Variables**

If you re-declare a JavaScript variable declared with var, it will not lose its value.

The variable carName will still have the value "Volvo" after the execution of these statements:

### **Example**

var carName = "Volvo";  
var carName;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_redefine)

## **Note**

You cannot re-declare a variable declared with let or const.

This will not work:

let carName = "Volvo";  
let carName;

## **JavaScript Arithmetic**

As with algebra, you can do arithmetic with JavaScript variables, using operators like = and +:

### **Example**

let x = 5 + 2 + 3;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_add_numbers)

You can also add strings, but strings will be concatenated:

### **Example**

let x = "John" + " " + "Doe";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_add_strings)

Also try this:

### **Example**

let x = "5" + 2 + 3;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_add_string_number)

## **Note**

If you put a number in quotes, the rest of the numbers will be treated as strings, and concatenated.

Now try this:

### **Example**

let x = 2 + 3 + "5";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_add_number_string)

## **JavaScript Dollar Sign $**

Since JavaScript treats a dollar sign as a letter, identifiers containing $ are valid variable names:

### **Example**

let $ = "Hello World";  
let $$$ = 2;  
let $myMoney = 5;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_dollar)

Using the dollar sign is not very common in JavaScript, but professional programmers often use it as an alias for the main function in a JavaScript library.

In the JavaScript library jQuery, for instance, the main function $ is used to select HTML elements. In jQuery $("p"); means "select all p elements".

## **JavaScript Underscore (\_)**

Since JavaScript treats underscore as a letter, identifiers containing \_ are valid variable names:

### **Example**

let \_lastName = "Johnson";  
let \_x = 2;  
let \_100 = 5;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_underscore)

Using the underscore is not very common in JavaScript, but a convention among professional programmers is to use it as an alias for "private (hidden)" variables.

# JavaScript Let

[❮ Previous](https://www.w3schools.com/js/js_variables.asp)[Next ❯](https://www.w3schools.com/js/js_const.asp)

The let keyword was introduced in [ES6 (2015)](https://www.w3schools.com/js/js_es6.asp).

Variables defined with let cannot be Redeclared.

Variables defined with let must be Declared before use.

Variables defined with let have Block Scope.

## **Cannot be Redeclared**

Variables defined with let cannot be **redeclared**.

You cannot accidentally redeclare a variable.

With let you can not do this:

### **Example**

let x = "John Doe";  
  
let x = 0;  
  
// SyntaxError: 'x' has already been declared

With var you can:

### **Example**

var x = "John Doe";  
  
var x = 0;

## **Block Scope**

Before ES6 (2015), JavaScript had only **Global Scope** and **Function Scope**.

ES6 introduced two important new JavaScript keywords: let and const.

These two keywords provide **Block Scope** in JavaScript.

Variables declared inside a { } block cannot be accessed from outside the block:

### **Example**

{  
  let x = 2;  
}  
// x can NOT be used here

Variables declared with the var keyword can NOT have block scope.

Variables declared inside a { } block can be accessed from outside the block.

### **Example**

{  
  var x = 2;  
}  
// x CAN be used here

## **Redeclaring Variables**

Redeclaring a variable using the var keyword can impose problems.

Redeclaring a variable inside a block will also redeclare the variable outside the block:

### **Example**

var x = 10;  
// Here x is 10  
  
{  
var x = 2;  
// Here x is 2  
}  
  
// Here x is 2

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_es6_var)

Redeclaring a variable using the let keyword can solve this problem.

Redeclaring a variable inside a block will not redeclare the variable outside the block:

### **Example**

let x = 10;  
// Here x is 10  
  
{  
let x = 2;  
// Here x is 2  
}  
  
// Here x is 10

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_es6_let)

## **Browser Support**

The let keyword is not fully supported in Internet Explorer 11 or earlier.

The following table defines the first browser versions with full support for the let keyword:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| Chrome 49 | Edge 12 | Firefox 44 | Safari 11 | Opera 36 |
| Mar, 2016 | Jul, 2015 | Jan, 2015 | Sep, 2017 | Mar, 2016 |

ADVERTISEMENT

## **Redeclaring**

Redeclaring a JavaScript variable with var is allowed anywhere in a program:

### **Example**

var x = 2;  
// Now x is 2  
  
var x = 3;  
// Now x is 3

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_let_redeclare_var)

With let, redeclaring a variable in the same block is NOT allowed:

### **Example**

var x = 2;    // Allowed  
let x = 3;    // Not allowed  
  
{  
let x = 2;    // Allowed  
let x = 3     // Not allowed  
}  
  
{  
let x = 2;    // Allowed  
var x = 3     // Not allowed  
}

Redeclaring a variable with let, in another block, IS allowed:

### **Example**

let x = 2;    // Allowed  
  
{  
let x = 3;    // Allowed  
}  
  
{  
let x = 4;    // Allowed  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_let_redeclare)

## **Let Hoisting**

Variables defined with var are **hoisted** to the top and can be initialized at any time.

Meaning: You can use the variable before it is declared:

### **Example**

This is OK:

carName = "Volvo";  
var carName;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_let_hoisting_var)

If you want to learn more about hoisting, study the chapter [JavaScript Hoisting](https://www.w3schools.com/js/js_hoisting.asp).

Variables defined with let are also hoisted to the top of the block, but not initialized.

Meaning: Using a let variable before it is declared will result in a ReferenceError:

### **Example**

carName = "Saab";  
let carName = "Volvo";

# JavaScript Const

[❮ Previous](https://www.w3schools.com/js/js_let.asp)[Next ❯](https://www.w3schools.com/js/js_operators.asp)

The const keyword was introduced in [ES6 (2015)](https://www.w3schools.com/js/js_es6.asp).

Variables defined with const cannot be Redeclared.

Variables defined with const cannot be Reassigned.

Variables defined with const have Block Scope.

## **Cannot be Reassigned**

A const variable cannot be reassigned:

### **Example**

const PI = 3.141592653589793;  
PI = 3.14;      // This will give an error  
PI = PI + 10;   // This will also give an error

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_const_value)

## **Must be Assigned**

JavaScript const variables must be assigned a value when they are declared:

### **Correct**

const PI = 3.14159265359;

### **Incorrect**

const PI;  
PI = 3.14159265359;

## **When to use JavaScript const?**

As a general rule, always declare a variable with const unless you know that the value will change.

Use const when you declare:

* A new Array
* A new Object
* A new Function
* A new RegExp

## **Constant Objects and Arrays**

The keyword const is a little misleading.

It does not define a constant value. It defines a constant reference to a value.

Because of this you can NOT:

* Reassign a constant value
* Reassign a constant array
* Reassign a constant object

But you CAN:

* Change the elements of constant array
* Change the properties of constant object

## **Constant Arrays**

You can change the elements of a constant array:

### **Example**

// You can create a constant array:  
const cars = ["Saab", "Volvo", "BMW"];  
  
// You can change an element:  
cars[0] = "Toyota";  
  
// You can add an element:  
cars.push("Audi");

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_const_array)

But you can NOT reassign the array:

### **Example**

const cars = ["Saab", "Volvo", "BMW"];  
  
cars = ["Toyota", "Volvo", "Audi"];    // ERROR

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_const_array_assign)

## **Constant Objects**

You can change the properties of a constant object:

### **Example**

// You can create a const object:  
const car = {type:"Fiat", model:"500", color:"white"};  
  
// You can change a property:  
car.color = "red";  
  
// You can add a property:  
car.owner = "Johnson";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_const_object)

But you can NOT reassign the object:

### **Example**

const car = {type:"Fiat", model:"500", color:"white"};  
  
car = {type:"Volvo", model:"EX60", color:"red"};    // ERROR

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_const_object_assign)

## **Browser Support**

The const keyword is not supported in Internet Explorer 10 or earlier.

The following table defines the first browser versions with full support for the const keyword:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| Chrome 49 | IE 11 / Edge | Firefox 36 | Safari 10 | Opera 36 |
| Mar, 2016 | Oct, 2013 | Feb, 2015 | Sep, 2016 | Mar, 2016 |

ADVERTISEMENT

## **Block Scope**

Declaring a variable with const is similar to let when it comes to **Block Scope**.

The x declared in the block, in this example, is not the same as the x declared outside the block:

### **Example**

const x = 10;  
// Here x is 10  
  
{  
const x = 2;  
// Here x is 2  
}  
  
// Here x is 10

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_const)

You can learn more about block scope in the chapter [JavaScript Scope](https://www.w3schools.com/js/js_scope.asp).

## **Redeclaring**

Redeclaring a JavaScript var variable is allowed anywhere in a program:

### **Example**

var x = 2;     // Allowed  
var x = 3;     // Allowed  
x = 4;         // Allowed

Redeclaring an existing var or let variable to const, in the same scope, is not allowed:

### **Example**

var x = 2;     // Allowed  
const x = 2;   // Not allowed  
  
{  
let x = 2;     // Allowed  
const x = 2;   // Not allowed  
}  
  
{  
const x = 2;   // Allowed  
const x = 2;   // Not allowed  
}

Reassigning an existing const variable, in the same scope, is not allowed:

### **Example**

const x = 2;     // Allowed  
x = 2;           // Not allowed  
var x = 2;       // Not allowed  
let x = 2;       // Not allowed  
const x = 2;     // Not allowed  
  
{  
  const x = 2;   // Allowed  
  x = 2;         // Not allowed  
  var x = 2;     // Not allowed  
  let x = 2;     // Not allowed  
  const x = 2;   // Not allowed  
}

Redeclaring a variable with const, in another scope, or in another block, is allowed:

### **Example**

const x = 2;       // Allowed  
  
{  
  const x = 3;   // Allowed  
}  
  
{  
  const x = 4;   // Allowed  
}

## **Const Hoisting**

Variables defined with var are **hoisted** to the top and can be initialized at any time.

Meaning: You can use the variable before it is declared:

### **Example**

This is OK:

carName = "Volvo";  
var carName;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_let_hoisting_var)

If you want to learn more about hoisting, study the chapter [JavaScript Hoisting](https://www.w3schools.com/js/js_hoisting.asp).

Variables defined with const are also hoisted to the top, but not initialized.

Meaning: Using a const variable before it is declared will result in a ReferenceError:

### **Example**

alert (carName);  
const carName = "Volvo";

# JavaScript Operators

[❮ Previous](https://www.w3schools.com/js/js_const.asp)[Next ❯](https://www.w3schools.com/js/js_arithmetic.asp)

### **Example**

Assign values to variables and add them together:

let x = 5;         // assign the value 5 to x  
let y = 2;         // assign the value 2 to y  
let z = x + y;     // assign the value 7 to z (5 + 2)

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_oper)

The **assignment** operator (=) assigns a value to a variable.

### **Assignment**

let x = 10;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_equal)

The **addition** operator (+) adds numbers:

### **Adding**

let x = 5;  
let y = 2;  
let z = x + y;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_add)

The **multiplication** operator (\*) multiplies numbers.

### **Multiplying**

let x = 5;  
let y = 2;  
let z = x \* y;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_mult)

ADVERTISEMENT

## **JavaScript Arithmetic Operators**

Arithmetic operators are used to perform arithmetic on numbers:

|  |  |
| --- | --- |
| **Operator** | **Description** |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| \*\* | Exponentiation ([ES2016](https://www.w3schools.com/js/js_2016.asp)) |
| / | Division |
| % | Modulus (Division Remainder) |
| ++ | Increment |
| -- | Decrement |

Arithmetic operators are fully described in the [**JS Arithmetic**](https://www.w3schools.com/js/js_arithmetic.asp) chapter.

## **JavaScript Assignment Operators**

Assignment operators assign values to JavaScript variables.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example** | **Same As** |
| = | x = y | x = y |
| += | x += y | x = x + y |
| -= | x -= y | x = x - y |
| \*= | x \*= y | x = x \* y |
| /= | x /= y | x = x / y |
| %= | x %= y | x = x % y |
| \*\*= | x \*\*= y | x = x \*\* y |

The **addition assignment** operator (+=) adds a value to a variable.

### **Assignment**

let x = 10;  
x += 5;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_plusequal)

Assignment operators are fully described in the [**JS Assignment**](https://www.w3schools.com/js/js_assignment.asp) chapter.

## **JavaScript String Operators**

The + operator can also be used to add (concatenate) strings.

### **Example**

let text1 = "John";  
let text2 = "Doe";  
let text3 = text1 + " " + text2;

The result of text3 will be:

John Doe

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_concatenate)

The += assignment operator can also be used to add (concatenate) strings:

### **Example**

let text1 = "What a very ";  
text1 += "nice day";

The result of text1 will be:

What a very nice day

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_concat4)

When used on strings, the + operator is called the concatenation operator.

## **Adding Strings and Numbers**

Adding two numbers, will return the sum, but adding a number and a string will return a string:

### **Example**

let x = 5 + 5;  
let y = "5" + 5;  
let z = "Hello" + 5;

The result of x, y, and z will be:

10  
55  
Hello5

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_concat5)

If you add a number and a string, the result will be a string!

## **JavaScript Comparison Operators**

|  |  |
| --- | --- |
| **Operator** | **Description** |
| == | equal to |
| === | equal value and equal type |
| != | not equal |
| !== | not equal value or not equal type |
| > | greater than |
| < | less than |
| >= | greater than or equal to |
| <= | less than or equal to |
| ? | ternary operator |

Comparison operators are fully described in the [**JS Comparisons**](https://www.w3schools.com/js/js_comparisons.asp) chapter.

## **JavaScript Logical Operators**

|  |  |
| --- | --- |
| **Operator** | **Description** |
| && | logical and |
| || | logical or |
| ! | logical not |

Logical operators are fully described in the [**JS Comparisons**](https://www.w3schools.com/js/js_comparisons.asp) chapter.

## **JavaScript Type Operators**

|  |  |
| --- | --- |
| **Operator** | **Description** |
| typeof | Returns the type of a variable |
| instanceof | Returns true if an object is an instance of an object type |

Type operators are fully described in the [**JS Type Conversion**](https://www.w3schools.com/js/js_type_conversion.asp) chapter.

## **JavaScript Bitwise Operators**

Bit operators work on 32 bits numbers.

Any numeric operand in the operation is converted into a 32 bit number. The result is converted back to a JavaScript number.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Same as** | **Result** | **Decimal** |
| & | AND | 5 & 1 | 0101 & 0001 | 0001 | 1 |
| | | OR | 5 | 1 | 0101 | 0001 | 0101 | 5 |
| ~ | NOT | ~ 5 | ~0101 | 1010 | 10 |
| ^ | XOR | 5 ^ 1 | 0101 ^ 0001 | 0100 | 4 |
| << | left shift | 5 << 1 | 0101 << 1 | 1010 | 10 |
| >> | right shift | 5 >> 1 | 0101 >> 1 | 0010 | 2 |
| >>> | unsigned right shift | 5 >>> 1 | 0101 >>> 1 | 0010 | 2 |

## **Operator Precedence**

Operator precedence describes the order in which operations are performed in an arithmetic expression.

### **Example**

let x = 100 + 50 \* 3;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_arithmetic_precedence1)

Is the result of example above the same as 150 \* 3, or is it the same as 100 + 150?

Is the addition or the multiplication done first?

As in traditional school mathematics, the multiplication is done first.

Multiplication (\*) and division (/) have higher **precedence** than addition (+) and subtraction (-).

And (as in school mathematics) the precedence can be changed by using parentheses:

### **Example**

let x = (100 + 50) \* 3;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_arithmetic_precedence2)

When using parentheses, the operations inside the parentheses are computed first.

When many operations have the same precedence (like addition and subtraction), they are computed from left to right:

### **Example**

let x = 100 + 50 - 3;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_arithmetic_precedence3)

## **JavaScript Operator Precedence Values**

Pale red entries indicates ECMAScript 2015 (ES6) or higher.

|  |  |  |  |
| --- | --- | --- | --- |
| **Value** | **Operator** | **Description** | **Example** |
| 21 | ( ) | Expression grouping | (3 + 4) |
|  |  |  |  |
| 20 | . | Member | person.name |
| 20 | [] | Member | person["name"] |
| 20 | () | Function call | myFunction() |
| 20 | new | Create | new Date() |
|  |  |  |  |
| 18 | ++ | Postfix Increment | i++ |
| 18 | -- | Postfix Decrement | i-- |
|  |  |  |  |
| 17 | ++ | Prefix Increment | ++i |
| 17 | -- | Prefix Decrement | --i |
| 17 | ! | Logical not | !(x==y) |
| 17 | typeof | Type | typeof x |
|  |  |  |  |
| 16 | \*\* | Exponentiation (ES2016) | 10 \*\* 2 |
|  |  |  |  |
| 15 | \* | Multiplication | 10 \* 5 |
| 15 | / | Division | 10 / 5 |
| 15 | % | Division Remainder | 10 % 5 |
|  |  |  |  |
| 14 | + | Addition | 10 + 5 |
| 14 | - | Subtraction | 10 - 5 |
|  |  |  |  |
| 13 | << | Shift left | x << 2 |
| 13 | >> | Shift right | x >> 2 |
| 13 | >>> | Shift right (unsigned) | x >>> 2 |
|  |  |  |  |
| 12 | < | Less than | x < y |
| 12 | <= | Less than or equal | x <= y |
| 12 | > | Greater than | x > y |
| 12 | >= | Greater than or equal | x >= y |
| 12 | in | Property in Object | "PI" in Math |
| 12 | instanceof | Instance of Object | instanceof Array |
|  |  |  |  |
| 11 | == | Equal | x == y |
| 11 | === | Strict equal | x === y |
| 11 | != | Unequal | x != y |
| 11 | !== | Strict unequal | x !== y |
|  |  |  |  |
| 10 | & | Bitwise AND | x & y |
| 9 | ^ | Bitwise XOR | x ^ y |
| 8 | | | Bitwise OR | x | y |
| 7 | && | Logical AND | x && y |
| 6 | || | Logical OR | x || y |
| 5 | ?? | Nullish Coalescing | x ?? y |
| 4 | ? : | Condition | ? "Yes" : "No" |
|  |  |  |  |
| 3 | += | Assignment | x += y |
| 3 | /= | Assignment | x /= y |
| 3 | -= | Assignment | x -= y |
| 3 | \*= | Assignment | x \*= y |
| 3 | %= | Assignment | x %= y |
| 3 | <<= | Assignment | x <<= y |
| 3 | >>= | Assignment | x >>= y |
| 3 | >>>= | Assignment | x >>>= y |
| 3 | &= | Assignment | x &= y |
| 3 | ^= | Assignment | x ^= y |
| 3 | |= | Assignment | x |= y |
|  |  |  |  |
| 2 | yield | Pause Function | yield x |
| 1 | , | Comma | 5 , 6 |

## **JavaScript Assignment Operators**

Assignment operators assign values to JavaScript variables.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example** | **Same As** |
| = | x = y | x = y |
| += | x += y | x = x + y |
| -= | x -= y | x = x - y |
| \*= | x \*= y | x = x \* y |
| /= | x /= y | x = x / y |
| %= | x %= y | x = x % y |
| <<= | x <<= y | x = x << y |
| >>= | x >>= y | x = x >> y |
| >>>= | x >>>= y | x = x >>> y |
| &= | x &= y | x = x & y |
| ^= | x ^= y | x = x ^ y |
| |= | x |= y | x = x | y |
| \*\*= | x \*\*= y | x = x \*\* y |

# JavaScript Data Types

[❮ Previous](https://www.w3schools.com/js/js_assignment.asp)[Next ❯](https://www.w3schools.com/js/js_functions.asp)

JavaScript variables can hold different data types: numbers, strings, objects and more:

let length = 16;                               // Number  
let lastName = "Johnson";                      // String  
let x = {firstName:"John", lastName:"Doe"};    // Object

## **The Concept of Data Types**

In programming, data types is an important concept.

To be able to operate on variables, it is important to know something about the type.

Without data types, a computer cannot safely solve this:

let x = 16 + "Volvo";

Does it make any sense to add "Volvo" to sixteen? Will it produce an error or will it produce a result?

JavaScript will treat the example above as:

let x = "16" + "Volvo";

When adding a number and a string, JavaScript will treat the number as a string.

### **Example**

let x = 16 + "Volvo";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_addstring)

### **Example**

let x = "Volvo" + 16;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_addstring2)

JavaScript evaluates expressions from left to right. Different sequences can produce different results:

### **JavaScript:**

let x = 16 + 4 + "Volvo";

Result:

20Volvo

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_addstrings_1)

### **JavaScript:**

let x = "Volvo" + 16 + 4;

Result:

Volvo164

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_addstrings_2)

In the first example, JavaScript treats 16 and 4 as numbers, until it reaches "Volvo".

In the second example, since the first operand is a string, all operands are treated as strings.

ADVERTISEMENT

## **JavaScript Types are Dynamic**

JavaScript has dynamic types. This means that the same variable can be used to hold different data types:

### **Example**

let x;           // Now x is undefined  
x = 5;           // Now x is a Number  
x = "John";      // Now x is a String

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_dynamic)

## **JavaScript Strings**

A string (or a text string) is a series of characters like "John Doe".

Strings are written with quotes. You can use single or double quotes:

### **Example**

let carName1 = "Volvo XC60";   // Using double quotes  
let carName2 = 'Volvo XC60';   // Using single quotes

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_string_quotes)

You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

### **Example**

let answer1 = "It's alright";             // Single quote inside double quotes  
let answer2 = "He is called 'Johnny'";    // Single quotes inside double quotes  
let answer3 = 'He is called "Johnny"';    // Double quotes inside single quotes

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_string)

You will learn more about strings later in this tutorial.

## **JavaScript Numbers**

JavaScript has only one type of numbers.

Numbers can be written with, or without decimals:

### **Example**

let x1 = 34.00;     // Written with decimals  
let x2 = 34;        // Written without decimals

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_numbers)

Extra large or extra small numbers can be written with scientific (exponential) notation:

### **Example**

let y = 123e5;      // 12300000  
let z = 123e-5;     // 0.00123

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_numbers_large)

You will learn more about numbers later in this tutorial.

## **JavaScript Booleans**

Booleans can only have two values: true or false.

### **Example**

let x = 5;  
let y = 5;  
let z = 6;  
(x == y)       // Returns true  
(x == z)       // Returns false

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_bolean)

Booleans are often used in conditional testing.

You will learn more about conditional testing later in this tutorial.

## **JavaScript Arrays**

JavaScript arrays are written with square brackets.

Array items are separated by commas.

The following code declares (creates) an array called cars, containing three items (car names):

### **Example**

const cars = ["Saab", "Volvo", "BMW"];

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_array)

Array indexes are zero-based, which means the first item is [0], second is [1], and so on.

You will learn more about **arrays** later in this tutorial.

## **JavaScript Objects**

JavaScript objects are written with curly braces {}.

Object properties are written as name:value pairs, separated by commas.

### **Example**

const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_object)

The object (person) in the example above has 4 properties: firstName, lastName, age, and eyeColor.

You will learn more about **objects** later in this tutorial.

## **The typeof Operator**

You can use the JavaScript typeof operator to find the type of a JavaScript variable.

The typeof operator returns the type of a variable or an expression:

### **Example**

typeof ""             // Returns "string"  
typeof "John"         // Returns "string"  
typeof "John Doe"     // Returns "string"

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_typeof_string)

### **Example**

typeof 0              // Returns "number"  
typeof 314            // Returns "number"  
typeof 3.14           // Returns "number"  
typeof (3)            // Returns "number"  
typeof (3 + 4)        // Returns "number"

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_typeof_number)

You will learn more about **typeof** later in this tutorial.

## **Undefined**

In JavaScript, a variable without a value, has the value undefined. The type is also undefined.

### **Example**

let car;    // Value is undefined, type is undefined

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_undefined)

Any variable can be emptied, by setting the value to undefined. The type will also be undefined.

### **Example**

car = undefined;    // Value is undefined, type is undefined

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_undefined_2)

## **Empty Values**

An empty value has nothing to do with undefined.

An empty string has both a legal value and a type.

### **Example**

let car = "";    // The value is "", the typeof is "string"