JavaScript

JavaScript is the world's most popular programming language. JavaScript is the programming language of the Web. JavaScript is easy to learn. This tutorial will teach you JavaScript from basic to advanced.

JavaScript and Java are completely different languages, both in concept and design. JavaScript was invented by Brendan Eich in 1995, and became an ECMA standard in 1997.

JavaScript Can Change HTML Content

One of many JavaScript HTML methods is getElementById().

The example below "finds" an HTML element (with id="demo"), and changes the element content (innerHTML) to "Hello JavaScript":

Example

```
document.getElementById("demo").innerHTML = "Hello JavaScript";
```

JavaScript accepts both double and single quotes

JavaScript Can Change HTML Styles (CSS)

Changing the style of an HTML element, is a variant of changing an HTML attribute:

Example

```
document.getElementById("demo").style.fontSize = "35px";
```

JavaScript Can Hide HTML Elements

Hiding HTML elements can be done by changing the display style:

Example

```
document.getElementById("demo").style.display = "none";
```

JavaScript Can Show HTML Elements

Showing hidden HTML elements can also be done by changing the display style:

Example

```
document.getElementById("demo").style.display = "block";
```

The <script> Tag

In HTML, JavaScript code is inserted between <script> and </script> tags.

Example

```
<script>
document.getElementById("demo").innerHTML = "My First JavaScript"; </script>
```

JavaScript Functions and Events

A JavaScript function is a block of JavaScript code, that can be executed when "called" for.

For example, a function can be called when an event occurs, like when the user clicks a button.

JavaScript in <head> or <body>

You can place any number of scripts in an HTML document.

Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both.

JavaScript in <head>

In this example, a JavaScript function is placed in the <head> section of an HTML page.

The function is invoked (called) when a button is clicked:

Example

```
<!DOCTYPE html>
<html>
<head> <script>
function myFunction() {   document.getElementById("demo").innerHTML
= "Paragraph changed."; }
</script>
</head>
<body>

<h2>Demo JavaScript in Head</h2>

<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try it</button>

</body> </html>
```

JavaScript in <body>

In this example, a JavaScript function is placed in the <body> section of an HTML page.

The function is invoked (called) when a button is clicked:

Example

```
<!DOCTYPE html>
```

```
<html>
<body>

<h2>Demo JavaScript in Body</h2>

<p id="demo">A Paragraph</p>

<button type="button" onclick="myFunction()">Try it</button>

<script> function myFunction() {
document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>

</body>
</html>
```

External JavaScript

Scripts can also be placed in external files: External

file: myScript.js

```
function myFunction() {    document.getElementById("demo").innerHTML
= "Paragraph changed."; }
```

External scripts are practical when the same code is used in many different web pages.

JavaScript files have the file extension .js.

To use an external script, put the name of the script file in the src (source) attribute of a <script> tag:

Example

```
<script src="myScript.js"></script>
```

You can place an external script reference in <head> or <body> as you like.

The script will behave as if it was located exactly where the <script> tag is located.

External scripts cannot contain <script> tags.

External JavaScript Advantages

Placing scripts in external files has some advantages:

- It separates HTML and code
- It makes HTML and JavaScript easier to read and maintain
- Cached JavaScript files can speed up page loads

To add several script files to one page - use several script tags:

Example

```
<script src="myScript1.js"></script>
<script src="myScript2.js"></script>
```

External References

An external script can be referenced in 3 different ways:

- With a full URL (a full web address)
- With a file path (like /js/)
- Without any path

This example uses a full URL to link to myScript.js:

Example

```
<script src="https://www.w3schools.com/js/myScript.js"></script>
```

This example uses a file path to link to myScript.js:

Example

```
<script src="/js/myScript.js"></script>
```

This example uses no path to link to myScript.js:

Example

```
<script src="myScript.js"></script>
```

JavaScript Display Possibilities

JavaScript can "display" data in different ways:

- Writing into an HTML element, using innerHTML.
- Writing into the HTML output using document.write().
- Writing into an alert box, using window.alert().
- Writing into the browser console, using console.log().

Using inner HTML

To access an HTML element, JavaScript can use the document.getElementById(id) method.

The id attribute defines the HTML element. The innerHTML property defines the HTML content:

Example

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My First Paragraph</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>

</body>
</html>
```

Using document.write()

For testing purposes, it is convenient to use document.write():

Example

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script> document.write(5
+ 6);
</script>

</body> </html>
```

Example

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<button type="button" onclick="document.write(5 + 6)">Try it</button>

</body>
</html>
```

Using window.alert()

You can use an alert box to display data:

Example

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
window.alert(5 + 6);
</script>

</body> </html>
```

You can skip the window keyword.

In JavaScript, the window object is the global scope object. This means that variables, properties, and methods by default belong to the window object. This also means that specifying the window keyword is optional:

Example

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script> alert(5
+ 6);
</script>

</body> </html>
```

Using console.log()

For debugging purposes, you can call the console.log() method in the browser to display data.

Example

```
<!DOCTYPE html>
<html>
<body>

<script>
console.log(5 + 6);
</script>
```

```
</body>
</html>
```

JavaScript Print: JavaScript does not have any print object or print methods. You cannot access output devices from JavaScript. The only exception is that you can call the window.print() method in the browser to print the content of the current window. Example

```
<!DOCTYPE html>
<html>
<body>

<button onclick="window.print()">Print this page</button>

</body> </html>
```

JavaScript Statements

Example

```
let x, y, z;    // Statement 1
x = 5;          // Statement 2
y = 6;          // Statement 3 z
= x + y;      // Statement 4
```

JavaScript Programs

A computer program is a list of "instructions" to be "executed" by a computer.

In a programming language, these programming instructions are called statements.

A JavaScript program is a list of programming statements.

In HTML, JavaScript programs are executed by the web browser.

JavaScript Statements

JavaScript statements are composed of:

Values, Operators, Expressions, Keywords, and Comments.

This statement tells the browser to write "Hello Dolly." inside an HTML element with id="demo":

Example

```
document.getElementById("demo").innerHTML = "Hello Dolly.";
```

Semicolons ;

Semicolons separate JavaScript statements.

Add a semicolon at the end of each executable statement:

Examples

```
let a, b, c;  // Declare 3 variables a = 5;
// Assign the value 5 to a b = 6;       //
Assign the value 6 to b c = a + b;    //
Assign the sum of a and b to c
```

JavaScript White Space

JavaScript ignores multiple spaces. You can add white space to your script to make it more readable.

The following lines are equivalent:

```
let person = "Hege"; let
person="Hege";
```

JavaScript Line Length and Line Breaks

For best readability, programmers often like to avoid code lines longer than 80 characters.

If a JavaScript statement does not fit on one line, the best place to break it is after an operator:

Example

```
document.getElementById("demo").innerHTML = "Hello
Dolly!";
```

JavaScript Code Blocks

JavaScript statements can be grouped together in code blocks, inside curly brackets {...}.

The purpose of code blocks is to define statements to be executed together.

One place you will find statements grouped together in blocks, is in JavaScript functions:

Example

```
function myFunction() {
document.getElementById("demo1").innerHTML = "Hello Dolly!";
document.getElementById("demo2").innerHTML = "How are you?";
}
```

JavaScript Keywords

Keyword	Description
var	Declares a variable
let	Declares a block variable

const	Declares a block constant
if	Marks a block of statements to be executed on a condition
switch	Marks a block of statements to be executed in different cases
for	Marks a block of statements to be executed in a loop
function	Declares a function
return	Exits a function
try	Implements error handling to a block of statements

JavaScript Syntax

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 Values

The JavaScript syntax defines two types of values:

- Fixed values
- Variable values

Fixed values are called Literals. Variable values are called Variables.

JavaScript Literals

The two most important syntax rules for fixed values are:

1. Numbers are written with or without decimals:

```
10.50
```

```
1001
```

2. Strings are text, written within double or single quotes:

```
"John Doe"
```

```
'John Doe'
```

JavaScript Variables

In a programming language, variables are used to store data values. JavaScript uses the keywords var, let and const to declare variables.

An equal sign is used to assign values to variables. In this example, x is defined as a variable. Then, x is assigned (given) the value 6:

```
let x; x
= 6;
```

JavaScript Operators

JavaScript uses arithmetic operators (+ - * /) to compute values:

```
(5 + 6) * 10
```

JavaScript uses an assignment operator (=) to assign values to variables:

```
let x, y;
x = 5; y
= 6;
```

JavaScript Expressions

An expression is a combination of values, variables, and operators, which computes to a value. The computation is called an evaluation.

For example, 5 * 10 evaluates to 50:

```
5 * 10
```

The values can be of various types, such as numbers and strings.

For example, "John" + " " + "Doe", evaluates to "John Doe":

```
"John" + " " + "Doe"
```

JavaScript Keywords

JavaScript keywords are used to identify actions to be performed.

The let keyword tells the browser to create variables:

```
let x, y; x =
5 + 6; y =
x * 10;
```

The var keyword also tells the browser to create variables:

```
var x, y; x
= 5 + 6; y
= x * 10;
```

JavaScript Comments

Not all JavaScript statements are "executed".

Code after double slashes // or between /* and */ is treated as a comment.

Comments are ignored, and will not be executed:

```
let x = 5;   // I will be executed

// x = 6;   I will NOT be executed
```

JavaScript Identifiers / Names

Identifiers are JavaScript names. Identifiers are used to name variables and keywords, and functions. The rules for legal names are the same in most programming languages.

A JavaScript name must begin with:

- A letter (A-Z or a-z)
- A dollar sign ($)
- Or an underscore (_)

Subsequent characters may be letters, digits, underscores, or dollar signs.

JavaScript is Case Sensitive

All JavaScript identifiers are case sensitive.

The variables lastName and lastname, are two different variables:

```
let lastname, lastName;
lastName    =    "Doe";
lastname = "Peterson";
```

Hyphens are not allowed in JavaScript. They are reserved for subtractions.

JavaScript Character Set

JavaScript uses the Unicode character set. Unicode covers (almost) all the characters, punctuations, and symbols in the world.

JavaScript Comments

Single Line Comments

Single line comments start with //.

Any text between // and the end of the line will be ignored by JavaScript (will not be executed).

This example uses a single-line comment before each code line:

Example

```
// Change heading:
document.getElementById("myH").innerHTML = "My First Page";

// Change paragraph: document.getElementById("myP").innerHTML =
"My first paragraph.";
```

This example uses a single line comment at the end of each line to explain the code:

Example

```
let x = 5;      // Declare x, give it the value of 5 let y
= x + 2;  // Declare y, give it the value of x + 2
```

Multi-line Comments

Multi-line comments start with /* and end with */.

Any text between /* and */ will be ignored by JavaScript.

This example uses a multi-line comment (a comment block) to explain the code:

Example

```
/*
The code below will change the
heading with id = "myH" and the
paragraph with id = "myP" in my
web page:
*/
document.getElementById("myH").innerHTML = "My First Page"; document.getElementById("myP").innerHTML
= "My first paragraph.";
```

JavaScript Variables

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;
```

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;
```

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

Example

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

When to Use JavaScript var?

Always declare JavaScript variables with var, let, or const. 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 browsers, 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;
```

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.

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.

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;
```

JavaScript Let

The let keyword was introduced in 2015.

Variables defined with let cannot be Re-declared. Variables defined with let must be Declared before use. Variables defined with let have Block Scope. Variables defined with let cannot be redeclared.

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;
```

JavaScript Const

Variables defined with const cannot be Re-declared. Variables defined with const cannot be Re-assigned. 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
```

When to use JavaScript const?

Always declare a variable with const when you know that the value should not be changed.

Use const when you declare:

- A new Array
- A new Object
- A new Function
- A new RegExp

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");
```

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";
```

JavaScript Operators Assignment

Examples

```
let x = 10;

// Assign the value 5 to x let
x = 5;
// Assign the value 2 to y let
y = 2;
// Assign the value x + y to z:
let z = x + y;
```

Adding

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

The Multiplication Operator (*) multiplies numbers. Multiplying

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

Types of JavaScript Operators

There are different types of JavaScript operators:

- Arithmetic Operators
- Assignment Operators
- Comparison Operators
- Logical Operators
- Conditional Operators
- Type Operators

JavaScript Arithmetic Operators

Arithmetic Operators are used to perform arithmetic on numbers:

Arthmetic Operators Example

```
let a = 3;
let x = (100 + 50) * a;
```

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

Adding JavaScript Strings

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

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

JavaScript Logical Operators

Operator	Description
&&	logical and

||	logical or
!	logical not

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

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

JavaScript Arithmetic Operators Example

```
let x = 100 + 50; Example
let x = (100 + 50) * a; Adding
```

The addition operator (+) adds numbers: Example

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

Subtracting

The subtraction operator (-) subtracts numbers. Example

```
let x = 5; let
y = 2; let z
= x - y;
```

Multiplying

The multiplication operator (*) multiplies numbers. Example

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

Dividing

The division operator (/) divides numbers. Example

```
let x = 5; let
y = 2; let z
= x / y;
```

Remainder

The modulus operator (%) returns the division remainder.

Example

```
    x = 5; let
y = 2; let z =
x % y;
```

Note: In arithmetic, the division of two integers produces a quotient and a remainder. In mathematics, the result of a modulo operation is the remainder of an arithmetic division.

Incrementing

The increment operator (++) increments numbers. Example

```
let x = 5; x++;
let z = x;
```

Decrementing

The decrement operator (--) decrements numbers. Example

```
let x = 5;
x--; let z
= x;
```

Exponentiation

The exponentiation operator (**) raises the first operand to the power of the second operand. Example

let x = 5; let z = x ** 2; x ** y produces the same

result as Math.pow(x,y): Example

```
let x = 5; let z =
Math.pow(x,2);
```

The += Operator

The Addition Assignment Operator adds a value to a variable. Addition

Assignment Examples

```
let x = 10; x
+= 5; The -
= Operator
```

The Subtraction Assignment Operator subtracts a value from a variable. Subtraction

Assignment Example

```
let x = 10; x
-= 5;
```

The *= Operator

```
let
```

The Multiplication Assignment Operator multiplies a variable. Multiplication

Assignment Example

```
let x = 10; x
*= 5;
```

The **= Operator

The Exponentiation Assignment Operator raises a variable to the power of the operand. Exponentiation

Assignment Example

```
let x = 10; x
**= 5;
```

The /= Operator

The Division Assignment Operator divides a variable. Division

Assignment Example

```
let x = 10; x
/= 5;
```

The %= Operator

The Remainder Assignment Operator assigns a remainder to a variable. Remainder

Assignment Example

```
let x = 10; x
%= 5;
```

The <<= Operator

The Left Shift Assignment Operator left shifts a variable.

Left Shift Assignment Example

```
    x = -100;
x <<= 5;
```

The >>= Operator

The Right Shift Assignment Operator right shifts a variable (signed). Right

Shift Assignment Example

```
let x = -100; x
>>= 5;
```

The >>>= Operator

The Unsigned Right Shift Assignment Operator right shifts a variable (unsigned). Unsigned

Right Shift Assignment Example

```
let x = -100; x
>>>= 5;
```

The &= Operator

The Bitwise AND Assignment Operator does a bitwise AND operation on two operands and assigns the result to the the variable.

Bitwise AND Assignment Example

```
let x = 10; x
&= 5;
```

The |= Operator

The Bitwise OR Assignment Operator does a bitwise OR operation on two operands and assigns the result to the variable.

Bitwise OR Assignment Example

```
let x = 10; x
|= 5;
```

The ^= Operator

The Bitwise XOR Assignment Operator does a bitwise XOR operation on two operands and assigns the result to the variable.

Bitwise XOR Assignment Example

```
let x = 10; x
^= 5;
```

The &&= Operator

The Logical AND assignment operator is used between two values.

If the first value is true, the second value is assigned.

```
let
```

Logical AND Assignment Example

```
let x = 10; x
&&= 5;
```

The ||= Operator

The Logical OR assignment operator is used between two values.

If the first value is false, the second value is assigned.

Logical OR Assignment Example

```
let x = 10; x
||= 5;
```

The ??= Operator

The Nullish coalescing assignment operator is used between two values.

If the first value is undefined or null, the second value is assigned.

Nullish Coalescing Assignment Example

```
let x = 10; x
??= 5;
```

JavaScript has 8 Datatypes

1. String
2. Number
3. Bigint
4. Boolean
5. Undefined
6. Null
7. Symbol
8. Object

The Object Datatype

The object data type can contain:

1. An object
2. An array
3. A date

Examples

```
// Numbers: let
length = 16;
weight = 7.5;

// Strings:
```

```
let color = "Yellow";
let lastName = "Johnson";

// Booleans let
x = true;
let y = false;

// Object:
const person = {firstName:"John", lastName:"Doe"};

// Array object:
const cars = ["Saab", "Volvo", "BMW"];

// Date object:
const date = new Date("2022-03-25"); let
x = "16" + "Volvo";
```

Note

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

```
let x = 16 + "Volvo";
```

Example let x =

```
"Volvo" + 16;
```

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

Example:

```
let x = 16 + 4 + "Volvo";
```

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
```

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

```
// Using double quotes:
let carName1 = "Volvo XC60";
```

```
let
```

```
// Using single quotes: let
carName2 = 'Volvo XC60';
```

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

Example

```
// Single quote inside double quotes:
let answer1 = "It's alright";

// Single quotes inside double quotes: let
answer2 = "He is called 'Johnny'";

// Double quotes inside single quotes: let
answer3 = 'He is called "Johnny"';
```

JavaScript Numbers

All JavaScript numbers are stored as decimal numbers (floating point). Numbers can be written with, or without decimals:

Example

```
// With decimals: let
x1 = 34.00;

// Without decimals: let
x2 = 34;
```

Exponential Notation

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

Example

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

Javascript are always one type: double
(64-bit floating point).

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"];
```

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"};
```

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"
```

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
```

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
```