**JavaScript Tutorial**

* 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.

**Why Study JavaScript?**

JavaScript is one of the 3 languages all web developers must learn:

  1. [HTML](https://www.w3schools.com/html/default.asp) to define the content of web pages.

  2. [CSS](https://www.w3schools.com/css/default.asp) to specify the layout of web pages.

  3. JavaScript to program the behaviour of web pages.

**JavaScript Introduction**

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":

**Program Example:**

<!DOCTYPE html>

<html>

<body>

<h1 id="demo">JavaScript can change HTML content.</h1>

<input type="button" value="Click Me!" onclick='document.getElementById("demo").innerHTML = "Hello JavaScript!"'/>

</body>

</html>

**JavaScript Can Change HTML Styles (CSS)**

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

**Program Example:**

<!DOCTYPE html>

<html>

<body>

<h3 id="demo">JavaScript can change the style of an HTML element.</h3>

<input type="button" value="Click Me for change fontSize" onclick="document.getElementById('demo').style.fontSize='35px'"/>

<input type="button" value="Click Me for change fontFamily" onclick="document.getElementById('demo').style.fontFamily='Arial'"/>

</body>

</html>

**JavaScript Can Hide HTML Elements**

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

**Program Example:**

<!DOCTYPE html>

<html>

<body>

<p id="demo">JavaScript can hide HTML elements.</p>

<input type="button" value="Click me" onclick="document.getElementById('demo').style.display='none'"/>

</body>

</html>

**JavaScript Can Show HTML Elements**

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

**Program Example:**

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript can show hidden HTML elements.</h1>

<h2 id="demo" style="display:none">Hellow World</h2>

<input type="button" value="Click me" onclick="document.getElementById('demo').style.display='block'"/>

</body>

</html>

**JavaScript Where To**

**JavaScript Functions and Events**

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

Example

function myFunction(a1, a2) {

  return a1 \* a2; // The function returns the product of a1 and a2

}

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

**Function Return**

When JavaScript reaches a return statement, the function will stop executing.

**Why Functions?**

You can reuse code: Define the code once, and use it many times.

You can use the same code many times with different arguments, to produce different results.

**External JavaScript**

Scripts can also be placed in external files:

External file: f.js

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:

**Program Example:**

**f.js**

function myFunction() {

document.getElementById("demo").innerHTML = "Paragraph changed.";

}

**external.html**

<!DOCTYPE html>

<html>

<body>

<h2>Demo External JavaScript</h2>

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

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

<p>This example links to "myScript.js".</p>

<p>(myFunction is stored in "myScript.js")</p>

<script src="f.js"></script>

</body>

</html>

**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:

**JavaScript Output**

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 innerHTML**

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.

**Program Example:**

<!DOCTYPE html>

<html>

<body>

<h2 id="demo">Example of innerHTML</h2>

<script>

document.getElementById("demo").innerHTML = 5 + 20;

</script>

</body>

</html>

**Using document.write()**

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

You can skip the window keyword.

**Program Example:**

<!DOCTYPE html>

<html>

<body>

<h2>Example of document.write</h2>

<script>

document.write(15 - 5);

</script>

</body>

</html>

**Using window.alert()**

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

**Program Example:**

<!DOCTYPE html>

<html>

<body>

<h2>Example of window.alert</h2>

<script>

window.alert(5 + 25);

</script>

</body>

</html>

**Using console.log()**

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

console.log(15 + 5);

**JavaScript Statements**

JavaScript statements are composed of: Values, Operators, Expressions, Keywords, and Comments.

**Semicolons ;**

Semicolons separate JavaScript statements.

Add a semicolon at the end of each executable statement.

**JavaScript White Space**

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

**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.

**Program Example:**

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Statements</h2>

<p>JavaScript statements are separated by semicolons.</p>

<p id="demo1"></p>

<p>You can break a code line after an operator an comma</p>

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

<script>

let a, b, c;

a = 5;

b = 6;

c = a + b;

document.getElementById("demo1").innerHTML = c;

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

</script>

</body>

</html>

**JavaScript Keywords**

JavaScript keywords are used to identify actions to be performed.

**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.

**JavaScript Values**

The JavaScript syntax defines two types of values:

1. Fixed values
2. 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:

55.45

1010

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

"hello world"

'hello world'

**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, a is defined as a variable. Then, a is assigned the value 10:

let a;

a = 10;

**Program Example:**

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript Variables</h1>

<p>In this example, a, b, and c are variables.</p>

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

<script>

var a = 5;

var b = 10;

var c = a + b;

document.getElementById("demo").innerHTML =

"The value of c is: " + c;

</script>

</body>

</html>

**JavaScript Operators**

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

(4 + 8) \* 11

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

let a, b;

a = 7;

b = 3;

**Program Example:**

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Arithmetic Operation</h2>

<h3>The + Operator</h3>

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

<h3>The - Operator</h3>

<p id="demo1"></p>

<h3>The / Operator</h3>

<p id="demo2"></p>

<h3>The % Operator</h3>

<p id="demo3"></p>

<h3>The \* Operator</h3>

<p id="demo4"></p>

<script>

let x = 5;

let y = 2;

let z = x + y;

let z1 = x - y;

let z2 = x / y;

let z3 = x % y;

let z4 = x \* y;

document.getElementById("demo").innerHTML = z;

document.getElementById("demo1").innerHTML = z1;

document.getElementById("demo2").innerHTML = z2;

document.getElementById("demo3").innerHTML = z3;

document.getElementById("demo4").innerHTML = z4;

</script>

</body>

</html>

**JavaScript Expressions**

An expression is a combination of values, variables, and operators, which computes a value.

The computation is called an evaluation.

For example, 8 + 2 evaluates to 10:

8 + 2

Expressions can also contain variable values:

a + 7

**JavaScript Comments**

JavaScript comments can be used to explain JavaScript code, and to make it more readable.

Not all JavaScript statements are "executed".

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

**Single-Line Comments**

Single line comments start with //.

Comments are ignored, and will not be executed:

let a = 8; // I will be executed

// a = 8; I will NOT be executed

**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:

**Program Example:**

<!DOCTYPE html>

<html>

<body>

<h1 id="myH"></h1>

<p id="myW"></p>

<script>

// Change heading:

document.getElementById("myH").innerHTML = "JavaScript Comments";

// Change paragraph:

document.getElementById("myP").innerHTML = "My first paragraph.";

/\*

The code below will change

the heading with id = "myH"

and the paragraph with id = "myW"

\*/

</script>

</body>

</html>

**JavaScript Arithmetic**

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

Example

let z = 8 + 2 + 5;

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

Example

let z = "hello" + " " + "world";

**JavaScript Arithmetic Operators**

Arithmetic operators are used to performing arithmetic on numbers:

**Operator**   **Description**

+ Addition

- Subtraction

\* Multiplication

/ Division

% Modulus (Division Remainder)

++ Increment

* Decrement

**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

**Program Example:**

<!DOCTYPE html>

<html>

<body>

<h2>The += Operator</h2>

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

<h2>The -= Operator</h2>

<p id="demo1"></p>

<h2>The \*= Operator</h2>

<p id="demo2"></p>

<h2>The /= Operator</h2>

<p id="demo3"></p>

<h2>The %= Operator</h2>

<p id="demo4"></p>

<script>

var x = 10;

var x1=10;

var x2=10;

var x3=10;

var x4=10;

x += 5;

x1 -= 5;

x2 \*= 5;

x3 /= 5;

x4 %= 5;

document.getElementById("demo").innerHTML = x;

document.getElementById("demo1").innerHTML = x1;

document.getElementById("demo2").innerHTML = x2;

document.getElementById("demo3").innerHTML = x3;

document.getElementById("demo4").innerHTML = x4;

</script>

</body>

</html>

**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 Data Types**

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

let length = 10; // Number

let lastName = "javascript"; // String

let x = {first:"hello", last:"world"}; // Object

**JavaScript Types are Dynamic**

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

Example

let a; // Now a is undefined

a = 5; // Now a is a Number

a = "hello"; // Now x is a String

**Program Example:**

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Strings</h2>

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

<p>Numbers can be written with, or without decimals:</p>

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

<script>

let answer1 = "It's alright";

let answer2 = "He is called 'javascript'";

let answer3 = 'He is called "javascript"';

let x2 = 34;

let x3 = 3.14;

document.getElementById("demo").innerHTML =

answer1 + "<br>" +

answer2 + "<br>" +

answer3 + "<br>" + x2 + "<br>" + x3;

</script>

</body>

</html>

**JavaScript Booleans**

Booleans can only have two values: true or false.

Example

let a = 5;

let b = 5;

let c = 6;

(a == b) // Returns true

(a == c) // Returns false

**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 = ["Swift", "dzire", "nano"];

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

**JavaScript Array Methods**

**Converting Arrays to Strings**

The JavaScript method toString() converts an array to a string of (comma separated) array values.

Example

const fruits = ["Banana", "Orange", "Apple", "Mango"];

document.getElementById("demo").innerHTML = fruits.toString();

Result:

Banana,Orange,Apple,Mango

**Popping and Pushing**

When you work with arrays, it is easy to remove elements and add new elements.

This is what popping and pushing are:

Popping items out of an array, or pushing items into an array.

**JavaScript Sorting Arrays**

Sorting an Array

The sort() method sorts an array alphabetically:

Example

const fruits = ["Banana", "Orange", "Apple", "Mango"];

fruits.sort();

**Reversing an Array**

The reverse() method reverses the elements in an array.

You can use it to sort an array in descending order:

Example

const fruits = ["Banana", "Orange", "Apple", "Mango"];

fruits.sort();

fruits.reverse();

**JavaScript Objects**

Real-Life Objects, Properties, and Methods

In real life, a car is an object.

A car has properties like weight and color, and methods like start and stop:

**Object Properties Methods**

car.name = Fiat car.start()

car.model = 500 car.drive()

car.weight = 850kg car.brake()

car.color = white car.stop()

All cars have the same properties, but the property values differ from car to car.

All cars have the same methods, but the methods are performed at different times.

**Program Example:**

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Objects</h2>

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

<script>

// Create an object:

const car = {type:"Fiat", model:"500", color:"white"};

// Display some data from the object:

document.getElementById("demo").innerHTML = "The car type is " + car.color;

</script>

</body>

</html>

**Object Definition**

You define (and create) a JavaScript object with an object literal:

Example

const person = {firstName:"hello", lastName:"world", age:50, eyeColor:"blue"};

**JavaScript Strings**

JavaScript strings are for storing and manipulating text.

**String Length**

To find the length of a string, use the built-in length property:

Example

let str = "helloworld";

let length = str.length;

**Converting to Upper and Lower Case**

* A string is converted to upper case with toUpperCase().
* A string is converted to lower case with toLowerCase().

**Program Example:**

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript String Methods</h2>

<p>Convert string to lower case:</p>

<p id="demo">Hello World!</p>

<p>Convert string to upper case:</p>

<p id="demo1">Hello World!</p>

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

<script>

function myFunction() {

let text = document.getElementById("demo").innerHTML;

document.getElementById("demo").innerHTML = text.toLowerCase();

let text1 = document.getElementById("demo1").innerHTML;

document.getElementById("demo1").innerHTML = text1.toUpperCase();

}

</script>

</body>

</html>

**Property Access**

ECMAScript 5 (2009) allows property access [ ] on strings:

Example

let str = "HELLO WORLD";

let char = str[0];

**JavaScript String Search**

JavaScript Search Methods

* String indexOf()
* String lastIndexOf()
* String startsWith()
* String endsWith()

**JavaScript String indexOf()**

The indexOf() method returns the index of (the position of) the first occurrence of a specified text in a string:

Example

let text = "Please locate where 'locate' occurs!";

text.indexOf("locate");

**JavaScript String lastIndexOf()**

The lastIndexOf() method returns the index of the last occurrence of a specified text in a string:

Example

let text = "Please locate where 'locate' occurs!";

text.lastIndexOf("locate");

**JavaScript String startsWith()**

The startsWith() method returns true if a string begins with a specified value, otherwise false:

Example

let str = "Hello world, welcome to the universe.";

str.startsWith("Hello");

**Program Example:**

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript String Methods</h2>

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

<script>

let str = "Please locate where 'locate' occurs!";

document.getElementById("demo").innerHTML = str.indexOf("locate")+"<br>"+str.startsWith("locate")+"<br>"+str.endsWith("locate");

</script>

</body>

</html>

**JavaScript Numbers**

JavaScript has only one type of number. Numbers can be written with or without decimals.

Example

let a = 3.37; // A number with decimals

let b = 9; // A number without decimals

By default, JavaScript displays numbers as base 10 decimals.

But you can use the toString() method to output numbers from base 2 to base 36.

Hexadecimal is base 16. The decimal is base 10. Octal is base 8. Binary is base 2.

Example

* let myNumber = 32;
* myNumber.toString(10);
* myNumber.toString(32);
* myNumber.toString(16);
* myNumber.toString(8);
* myNumber.toString(2);

**Program Example:**

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Numbers</h2>

<p>The toString() method can output numbers from base 2 to 36:</p>

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

<script>

let myNumber = 15;

document.getElementById("demo").innerHTML =

"15 = " + "<br>" +

" Decimal " + myNumber.toString(10) + "<br>" +

" Hexadecimal " + myNumber.toString(16) + "<br>" +

" Octal " + myNumber.toString(8) + "<br>" +

" Binary " + myNumber.toString(2);

</script>

</body>

</html>

**JavaScript Date Objects**

Example

const d = new Date();

By default, JavaScript will use the browser's time zone and display a date as a full-text string.

**Creating Date Objects**

Date objects are created with the new Date() constructor. There are 4 ways to create a new date object:

1. new Date()
2. new Date(year, month, day, hours, minutes, seconds, milliseconds)
3. new Date(milliseconds)
4. new Date(date string)

Note: JavaScript counts months from 0 to 11:

January = 0.

December = 11.

Specifying a month higher than 11, will not result in an error but add the overflow to the next year

**Using 6, 4, 3, or 2 Numbers**

* 6 numbers specify year, month, day, hour, minute, second:

Example

const d = new Date(2018, 11, 24, 10, 33, 30);

* 5 numbers specify year, month, day, hour, and minute:

Example

const d = new Date(2018, 11, 24, 10, 33);

* 4 numbers specify year, month, day, and hour:

Example

const d = new Date(2018, 11, 24, 10);

* 3 numbers specify year, month, and day:

Example

const d = new Date(2018, 11, 24);

* 2 numbers specify year and month:

Examp le

const d = new Date(2018, 11);

**JavaScript Stores Dates as Milliseconds**

JavaScript stores date as several milliseconds since January 01, 1970, 00:00:00 UTC (Universal Time Coordinated).

Zero time is January 01, 1970, 00:00:00 UTC.

Now the time is: 1643997997806 milliseconds past January 01, 1970

**Date Methods**

When a Date object is created, several methods allow you to operate on it.

Date methods allow you to get and set the year, month, day, hour, minute, second, and millisecond of date objects, using either local time or UTC (universal, or GMT) time.

When you display a date object in HTML, it is automatically converted to a string, with the toString() method.

Example

const d = new Date();

d.toString();

The toUTCString() method converts a date to a UTC string (a date display standard).

Example

const d = new Date();

d.toUTCString();

The toDateString() method converts a date to a more readable format:

Example

const d = new Date();

d.toDateString();

The toISOString() method converts a Date object to a string, using the ISO standard format:

Example

const d = new Date();

d.toISOString();

**JavaScript Date Formats**

JavaScript Date Input

There are generally 3 types of JavaScript date input formats:

TypeExample

1. ISO Date "2015-03-25" (The International Standard)
2. Short Date "03/25/2015"
3. Long Date "Mar 25 2015" or "25 Mar 2015"

The ISO format follows a strict standard in JavaScript. The other formats are not so well defined and might be browser-specific.

* Date and time are separated with a capital T.
* UTC time is defined with the capital letter Z.

If you want to modify the time relative to UTC, remove the Z and add +HH:MM or -HH:MM instead:

Example

const d = new Date("2015-03-25T12:00:00-06:30");

UTC (Universal Time Coordinated) is the same as GMT (Greenwich Mean Time).

Omitting T or Z in a date-time string can give different results in different browsers.

**Time Zones**

When setting a date, without specifying the time zone, JavaScript will use the browser's time zone.

When getting a date, without specifying the time zone, the result is converted to the browser's time zone.

**WARNINGS!**

In some browsers, months or days with no leading zeroes may produce an error: const d = new Date("2015-3-25");

The behavior of "DD-MM-YYYY" is also undefined.

Some browsers will try to guess the format. Some will return NaN.

Date Input - Parsing Dates

If you have a valid date string, you can use the Date.parse() method to convert it to milliseconds.

Date.parse() returns the number of milliseconds between the date and January 1, 1970:

Example

let msec = Date.parse("March 21, 2012");

**JavaScript Get-Date Methods**

These methods can be used for getting information from a date object:

**Method Description**

getFullYear() Get the year as a four digit number (yyyy)

getMonth() Get the month as a number (0-11)

getDate() Get the day as a number (1-31)

getHours() Get the hour (0-23)

getMinutes() Get the minute (0-59)

getSeconds() Get the second (0-59)

getMilliseconds() Get the millisecond (0-999)

getTime() Get the time (milliseconds since January 1, 1970)

getDay() Get the weekday as a number (0-6)

Date.now() Get the time. ECMAScript 5.

**UTC Date Methods**

UTC date methods are used for working with UTC dates (Universal Time Zone dates):

**Method Description**

getUTCDate() Same as getDate(), but returns the UTC date

getUTCDay() Same as getDay(), but returns the UTC day

getUTCFullYear() Same as getFullYear(), but returns the UTC year

getUTCHours() Same as getHours(), but returns the UTC hour

getUTCMilliseconds() Same as getMilliseconds(), but returns the UTC milliseconds

getUTCMinutes() Same as getMinutes(), but returns the UTC minutes

getUTCMonth() Same as getMonth(), but returns the UTC month

getUTCSeconds() Same as getSeconds(), but returns the UTC seconds

**JavaScript Set Date Methods**

**Set Date Methods**

Set Date methods are used for setting a part of a date:

Method Description

setDate() Set the day as a number (1-31)

setFullYear() Set the year (optionally month and day)

setHours() Set the hour (0-23)

setMilliseconds() Set the milliseconds (0-999)

setMinutes() Set the minutes (0-59)

setMonth() Set the month (0-11)

setSeconds() Set the seconds (0-59)

setTime() Set the time (milliseconds since January 1, 1970)

**Compare Dates**

Dates can easily be compared.

**JavaScript Math Object**

The JavaScript Math object allows you to perform mathematical tasks on numbers.

Example

Math.PI;

**The Math Object**

Unlike other objects, the Math object has no constructor. The Math object is static. All methods and properties can be used without creating a Math object first.

Math Properties (Constants)

The syntax for any Math property is Math.property.

JavaScript provides 8 mathematical constants that can be accessed as Math properties:

Example

1. Math.E // returns Euler's number
2. Math.PI // returns PI
3. Math.SQRT2 // returns the square root of 2
4. Math.SQRT1\_2 // returns the square root of 1/2
5. Math.LN2 // returns the natural logarithm of 2
6. Math.LN10 // returns the natural logarithm of 10
7. Math.LOG2E // returns base 2 logarithms of E
8. Math.LOG10E // returns base 10 logarithms of E

**Math Methods**

The syntax for Math any methods is: Math.method(number)

**Number to Integer**

There are 4 common methods to round a number to an integer:

1. Math.round(x) Returns x rounded to its nearest integer
2. Math.ceil(x) Returns x rounded up to its nearest integer
3. Math.floor(x) Returns x rounded down to its nearest integer
4. Math.trunc(x) Returns the integer part of x ([new in ES6](https://www.w3schools.com/js/js_es6.asp))

**JavaScript Math Methods**

**Method Description**

[abs(x)](https://www.w3schools.com/jsref/jsref_abs.asp) Returns the absolute value of x

[acos(x)](https://www.w3schools.com/jsref/jsref_acos.asp) Returns the arccosine of x, in radians

[acosh(x)](https://www.w3schools.com/jsref/jsref_acosh.asp) Returns the hyperbolic arccosine of x

[asin(x)](https://www.w3schools.com/jsref/jsref_asin.asp) Returns the arcsine of x, in radians

[asinh(x)](https://www.w3schools.com/jsref/jsref_asinh.asp) Returns the hyperbolic arcsine of x

[atan(x)](https://www.w3schools.com/jsref/jsref_atan.asp) Returns the arctangent of x as a numeric value between -PI/2 and PI/2 radians

[atan2(y, x)](https://www.w3schools.com/jsref/jsref_atan2.asp) Returns the arctangent of the quotient of its arguments

[atanh(x)](https://www.w3schools.com/jsref/jsref_atanh.asp) Returns the hyperbolic arctangent of x

[cbrt(x)](https://www.w3schools.com/jsref/jsref_cbrt.asp) Returns the cubic root of x

[ceil(x)](https://www.w3schools.com/jsref/jsref_ceil.asp) Returns x, rounded upwards to the nearest integer

[cos(x)](https://www.w3schools.com/jsref/jsref_cos.asp) Returns the cosine of x (x is in radians)

[cosh(x)](https://www.w3schools.com/jsref/jsref_cosh.asp) Returns the hyperbolic cosine of x

[exp(x)](https://www.w3schools.com/jsref/jsref_exp.asp) Returns the value of Ex

[floor(x)](https://www.w3schools.com/jsref/jsref_floor.asp) Returns x, rounded downwards to the nearest integer

[log(x)](https://www.w3schools.com/jsref/jsref_log.asp) Returns the natural logarithm (base E) of x

[max(x, y, z, .., n)](https://www.w3schools.com/jsref/jsref_max.asp) Returns the number with the highest value

[min(x, y, z, .., n)](https://www.w3schools.com/jsref/jsref_min.asp) Returns the number with the lowest value

[pow(x, y)](https://www.w3schools.com/jsref/jsref_pow.asp) Returns the value of x to the power of y

[random()](https://www.w3schools.com/jsref/jsref_random.asp) Returns a random number between 0 and 1

[round(x)](https://www.w3schools.com/jsref/jsref_round.asp) Rounds x to the nearest integer

[sign(x)](https://www.w3schools.com/jsref/jsref_sign.asp) Returns if x is negative, null or positive (-1, 0, 1)

[sin(x)](https://www.w3schools.com/jsref/jsref_sin.asp) Returns the sine of x (x is in radians)

[sinh(x)](https://www.w3schools.com/jsref/jsref_sinh.asp) Returns the hyperbolic sine of x

[sqrt(x)](https://www.w3schools.com/jsref/jsref_sqrt.asp) Returns the square root of x

[tan(x)](https://www.w3schools.com/jsref/jsref_tan.asp) Returns the tangent of an angle

[tanh(x)](https://www.w3schools.com/jsref/jsref_tanh.asp) Returns the hyperbolic tangent of a number

[trunc(x)](https://www.w3schools.com/jsref/jsref_trunc.asp) Returns the integer part of a number (x)

**JavaScript Random**

**Math.random()**

Math.random() returns a random number between 0 (inclusive), and 1 (exclusive):

Example

// Returns a random number:

Math.random();

**A Proper Random Function**

As you can see from the examples above, it might be a good idea to create a proper random function to use for all random integer purposes.

This JavaScript function always returns a random number between min (included) and max (excluded):

Example

function getRndInteger(min, max) {

  return Math.floor(Math.random() \* (max - min) ) + min;

}

**JavaScript Booleans**

A JavaScript Boolean represents one of two values: true or false.

The Boolean value of -0 (minus zero) is false:

let x = -0;

Boolean(x);

The Boolean value of "" (empty string) is false:

let x = "";

Boolean(x);

The Boolean value of undefined is false:

let x;

Boolean(x);

The Boolean value of null is false:

let x = null;

Boolean(x);

**JavaScript Booleans as Objects**

Normally JavaScript booleans are primitive values created from literal:

let x = false;

But booleans can also be defined as objects with the keyword new:

let y = new Boolean(false);

Comparing two JavaScript objects always return false.

**JavaScript For In**

The JavaScript for in statement loops through the properties of an Object:

Syntax

for (key in object) {

  // code block to be executed

}

**For In Over Arrays**

The JavaScript for in statement can also loop over the properties of an Array:

Syntax

for (variable in array) {

  code

}

**Array.forEach()**

The forEach() method calls a function (a callback function) once for each array element.

Example

const numbers = [45, 4, 9, 16, 25];

let txt = "";

numbers.forEach(myFunction);

function myFunction(value, index, array) {

 txt += value;}

**JavaScript For Of**

The JavaScript for statement loops through the values of an iterable object.

It lets you loop over iterable data structures such as Arrays, Strings, Maps, novelists, and more:

Syntax

for (variable of iterable) {

  // code block to be executed

}

**JavaScript Break and Continue**

The break statement can also be used to jump out of a loop:

Example

for (let i = 0; i < 10; i++) {

  if (i === 3) { break; }

 text += "The number is " + i + "<br>";

}

**The Continue Statement**

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

This example skips the value of 3:

Example

for (let i = 0; i < 10; i++) {

  if (i === 3) { continue; }

 text += "The number is " + i + "<br>";

}

**JavaScript Labels**

To label JavaScript statements you precede the statements with a label name and a colon:

label:

statements

The break and the continue statements are the only JavaScript statements that can "jump out of" a code block.

Syntax:

break labelname;

continue labelname;

The continue statement (with or without a label reference) can only be used to skip one loop iteration.

The break statement, without a label reference, can only be used to jump out of a loop or a switch.

With a label reference, the break statement can be used to jump out of any code block:

**JavaScript Sets**

A JavaScript Set is a collection of unique values. Each value can only occur once in a Set.

**Essential Set Methods**

**Method Description**

new Set() Creates a new Set

add() Adds a new element to the Set

delete() Removes an element from a Set

has() Returns true if a value exists in the Set

forEach() Invokes a callback for each element in the Set

values() Returns an iterator with all the values in a Set

**Essential Map Methods**

**Method Description**

new Map() Creates a new Map

set() Sets the value for a key in a Map

get() Gets the value for a key in a Map

delete() Removes a Map element specified by the key

has() Returns true if a key exists in a Map

forEach() Calls a function for each key/value pair in a Map

entries() Returns an iterator with the [key, value] pairs in a Map

**JavaScript Objects vs Maps**

Differences between JavaScript Objects and Maps:

**ObjectMap**

* IterableNot directly iterable Directly iterable
* size does not have a size property Have a size property
* Key TypesKeys must be Strings (or Symbols) Keys can be any datatype
* Key OrderKeys are not well-ordered Keys are ordered by insertion
* DefaultsHave default keys Do not have default keys

**JavaScript Type Conversion**

* Converting Strings to Numbers
* Converting Numbers to Strings
* Converting Dates to Numbers
* Converting Numbers to Dates
* Converting Booleans to Numbers
* Converting Numbers to Booleans

**Number Methods**

In the chapter [Number Methods](https://www.w3schools.com/js/js_number_methods.asp), you will find more methods that can be used to convert strings to numbers:

**Method Description**

Number() Returns a number, converted from its argument

parseFloat() Parses a string and returns a floating-point number

parseInt() Parses a string and returns an integer

**More Methods** In the chapter [Number Methods](https://www.w3schools.com/js/js_number_methods.asp), you will find more methods that can be used to convert numbers to strings:

**Method Description**

toExponential() Returns a string, with a number rounded and written using exponential notation.

toFixed() Returns a string, with a number rounded and written with a specified number of decimals.

toPrecision() Returns a string, with a number written with a specified length

In the chapter [Date Methods](https://www.w3schools.com/js/js_date_methods.asp), you will find more methods that can be used to convert dates to strings:

**Method Description**

getDate() Get the day as a number (1-31)

getDay() Get the weekday a number (0-6)

getFullYear() Get the four-digit year (yyyy)

getHours() Get the hour (0-23)

getMilliseconds() Get the milliseconds (0-999)

getMinutes() Get the minutes (0-59)

getMonth() Get the month (0-11)

getSeconds() Get the seconds (0-59)

getTime() Get the time (milliseconds since January 1, 1970)

**JavaScript Regular Expressions**

A regular expression is a sequence of characters that forms a search pattern.

The search pattern can be used for text search and text replacement operations.

Syntax

/pattern/modifiers;

Example

/w3schools/i;

Example explained:

/w3schools/i is a regular expression.

w3schools is a pattern (to be used in a search).

i is a modifier (modifies the search to be case-insensitive).

**Using String Methods**

In JavaScript, regular expressions are often used with the two string methods: search() and replace().

The search() method uses an expression to search for a match and returns the position of the match.

The replace() method returns a modified string where the pattern is replaced.

**Regular Expression Modifiers**

Modifiers can be used to perform case-insensitive more global searches:

**Modifier Description**

i Perform case-insensitive matching

g Perform a global match (find all matches rather than stopping after the first match)

m Perform multiline matching

**Regular Expression Patterns**

Brackets are used to find a range of characters:

**Expression Description**

[abc] Find any of the characters between the brackets

[0-9] Find any of the digits between the brackets

(x|y) Find any of the alternatives separated with |

Metacharacters are characters with a special meaning:

**Metacharacter Description**

\d Find a digit

\s Find a whitespace character

\b Find a match at the beginning of a word like this: \bWORD, or at the end of a word like this: WORD\b

\uxxxx Find the Unicode character specified by the hexadecimal number XXXX

**Quantifiers define quantities:**

**Quantifier Description**

n+ Matches any string that contains at least one n

n\* Matches any string that contains zero or more occurrences of n

n? Matches any string that contains zero or one occurrence of n

**The Error Object**

JavaScript has a built-in error object that provides error information when an error occurs.

The error object provides two useful properties: name and message.

**Error Name Values**

Six different values can be returned by the error name property:

**Error Name Description**

EvalError An error has occurred in the eval() function

RangeError A number "out of range" has occurred

ReferenceError An illegal reference has occurred

SyntaxError A syntax error has occurred

TypeError A type error has occurred

URIError An error in encodeURI() has occurred

The six different values are described below.

**JavaScript Hoisting**

Hoisting is JavaScript's default behavior of moving declarations to the top.

**JavaScript Declarations are Hoisted**

In JavaScript, a variable can be declared after it has been used.

In other words; a variable can be used before it has been declared.

Example 1 gives the same result as Example 2:

Example 1

x = 5; // Assign 5 to x

elem = document.getElementById("demo"); // Find an element

elem.innerHTML = x; // Display x in the element

var x; // Declare x

Example 2

var x; // Declare x

x = 5; // Assign 5 to x

elem = document.getElementById("demo"); // Find an element

elem.innerHTML = x; // Display x in the element

To understand this, you have to understand the term "hoisting".

Hoisting is JavaScript's default behavior of moving all declarations to the top of the current scope (to the top of the current script or the current function).

* **Declare Your Variables At the Top!**

Hoisting is (to many developers) an unknown or overlooked behavior of JavaScript. If a developer doesn't understand hoisting, programs may contain bugs (errors). To avoid bugs, always declare all variables at the beginning of every scope. Since this is how JavaScript interprets the code, it is always a good rule.

* **JavaScript Use Strict**

"use strict"; Defines that JavaScript code should be executed in "strict mode".

* **Declaring Strict Mode**

Strict mode is declared by adding "use strict"; to the beginning of a script or a function.

Declared at the beginning of a script, it has global scope (all code in the script will execute in strict mode):

Example

"use strict";

x = 3.14; // This will cause an error because x is not declared

Example

"use strict";

myFunction();

function myFunction() {

  y = 3.14; // This will also cause an error because y is not declared

}

Declared inside a function, it has local scope (only the code inside the function is in strict mode):

x = 3.14; // This will not cause an error.

myFunction();

function myFunction() {

  "use strict";

 y = 3.14; // This will cause an error

}

**Why Strict Mode?**

Strict mode makes it easier to write "secure" JavaScript. Strict mode changes previously accepted "bad syntax" into real errors.

As an example, in normal JavaScript, mistyping a variable name creates a new global variable. In strict mode, this will throw an error, making it impossible to accidentally create a global variable.

In normal JavaScript, a developer will not receive any error feedback assigning values to non-writable properties.

In strict mode, any assignment to a non-writable property, a getter-only property, a non-existing property, a non-existing variable, or a non-existing object, will throw an error.

**Future Proof!**

Keywords reserved for future JavaScript versions can NOT be used as variable names in strict mode.

These are:

* implements
* interface
* let
* package
* private
* protected
* public
* static
* yield

**The JavaScript this Keyword**

Example

const person = {

 firstName: "John",

  lastName : "Doe",

 id : 5566,

 fullName : function() {

    return this.firstName + " " + this.lastName;

  }

};

**What is this?**

In JavaScript, this keyword refers to an object. Which object depends on how this is being invoked (used or called).

The this keyword refers to different objects depending on how it is used:

In an object method, this refers to the object. Alone, this refers to the global object. In a function, this refers to the global object. In a function, in strict mode, this is undefined. In an event, this refers to the element that received the event.

Methods like call(), apply(), and bind() can refer to any object.

Note

this is not a variable. It is a keyword. You cannot change the value of this.

**This Precedence**

To determine which object this refers to; Use the following precedence of order.

**Precedence Object**

1 bind() Is this in a function being called using bind()?

2 apply() and call() Is this in a function is being called using apply()?, Is this in a function that is being called using call()?

3 Object method Is this in an object function (method)?

4 Global scope Is this in a function in the global scope?

**JavaScript Arrow Function**

Arrow functions were introduced in ES6. Arrow functions allow us to write shorter function syntax:

let myFunction = (a, b) => a \* b;

**JavaScript Classes**

ECMAScript 2015, also known as ES6, introduced JavaScript Classes.

JavaScript Classes are templates for JavaScript Objects.

JavaScript Class Syntax

Use the keyword class to create a class.

Always add a method named constructor():

Syntax

class ClassName {

 constructor() { ... }

}

Exam

**JavaScript Modules**

JavaScript modules allow you to break up your code into separate files. This makes it easier to maintain the code-base. JavaScript modules rely on the import and export statements.

**Export**

You can export a function or variable from any file. Let us create a file named person.js, and fill it with the things we want to export. There are two types of exports: Named and Default.

**JavaScript JSON**

JSON is a format for storing and transporting data. JSON is often used when data is sent from a server to a web page.

**What is JSON?**

JSON stands for JavaScript Object Notation. JSON is a lightweight data-interchange format. JSON is language independent. JSON is "self-describing" and easy to understand.

The JSON syntax is derived from JavaScript object notation syntax, but the JSON format is text only. Code for reading and generating JSON data can be written in any programming language.

JSON Example

This JSON syntax defines an employees object: an array of 3 employee records (objects):

JSON Example

{

"employees":[

  {"firstName":"John", "lastName":"Doe"},

  {"firstName":"Anna", "lastName":"Smith"},

 {"firstName":"Peter", "lastName":"Jones"}

]

}

**JavaScript Debuggers**

Debugging is not easy. But fortunately, all modern browsers have a built-in JavaScript debugger. Built-in debuggers can be turned on and off, forcing errors to be reported to the user. With a debugger, you can also set breakpoints (places

where code execution can be stopped), and examine variables while the code is executing.

Normally, otherwise, follow the steps at the bottom of this page, you activate debugging in your browser with the F12 key, and select "Console" in the debugger menu.

**The debugger Keyword**

The debugger keyword stops the execution of JavaScript, and calls (if available) the debugging function. This has the same function as setting a breakpoint in the debugger. If no debugging is available, the debugger statement has no effect.

With the debugger turned on, this code will stop executing before it executes the third line.

Example

let x = 15 \* 5;

debugger;

document.getElementById("demo").innerHTML = x;

**JavaScript Style Guide**

Always use the same coding conventions for all your JavaScript projects.

**JavaScript Coding Conventions**

Coding conventions are style guidelines for programming. They typically cover:

Naming and declaration rules for variables and functions. Rules for the use of white space, indentation, and comments. Programming practices and principles

Coding conventions secure quality:

Improves code readability. Make code maintenance easier. Coding conventions can be documented rules for teams to follow, or just be your coding practice.

**Performance**

Coding conventions are not used by computers. Most rules have little impact on the execution of programs. Indentation and extra spaces are not significant in small scripts. For code in development, readability should be preferred. Larger production scripts should be minified.

* JavaScript Best Practices
* Avoid Global Variables
* Always Declare Local Variables
* Declarations on Top
* Declare Objects with const
* Don't Use new Object()
* Use "" instead of new String()
* Use 0 instead of new Number()
* Use false instead of new Boolean()
* Use {} instead of new Object()
* Use [] instead of new Array()
* Use /()/ instead of new RegExp()
* Use function (){} instead of new Function()

**JavaScript Performance**

How to speed up your JavaScript code.

* Reduce Activity in Loops
* Reduce DOM Access
* Reduce DOM Size
* Avoid Unnecessary Variables
* JavaScript Reserved Words

**Removed Reserved Words**

The following reserved words have been removed from the ECMAScript 5/6 standard:

* abstract boolean byte char
* double final float goto
* int long native short
* synchronized throws transient volatile

Do not use these words as variables. ECMAScript 5/6 does not have full support in all browsers.

**ADVERTISEMENT**

JavaScript Objects, Properties, and Methods

You should also avoid using the name of JavaScript built-in objects, properties, and methods:

* Array Date eval function
* hasOwnProperty Infinity isFinite isNaN
* isPrototypeOf length Math NaN
* name Number Object prototype
* String toString undefined valueOf

**Java Reserved Words**

JavaScript is often used together with Java. You should avoid using some Java objects and properties as JavaScript identifiers:

* getClass java JavaArray javaClass
* JavaObject JavaPackage

**Other Reserved Words**

JavaScript can be used as the programming language in many applications.

You should also avoid using the name of HTML and Window objects and properties:

* alert all anchor anchors
* area assign blur button
* checkbox clearInterval clearTimeout clientInformation
* close closed confirm constructor
* crypto decodeURI decodeURIComponent defaultStatus
* document element elements embed
* embeds encodeURI encodeURIComponent escape
* event fileUpload focus form
* forms frame innerHeight innerWidth
* layer layers link location
* mimeTypes navigate navigator frames
* framerate hidden history image
* images offscreenBuffering open opener
* option outerHeight outerWidth packages
* pageXOffset pageYOffset parent parseFloat
* parseInt password pkcs11 plugin
* prompt propertyIsEnum radio reset
* screenX screenY scroll secure
* select self setInterval setTimeout
* status submit taint text
* textarea top unescaped untaint
* window

**HTML Event Handlers**

In addition, you should avoid using the name of all HTML event handlers.

Examples:

* onblur onclick onerror onfocus
* onkeydown onkeypress onkeyup onmouseover
* onload onmouseup onmousedown onsubmit