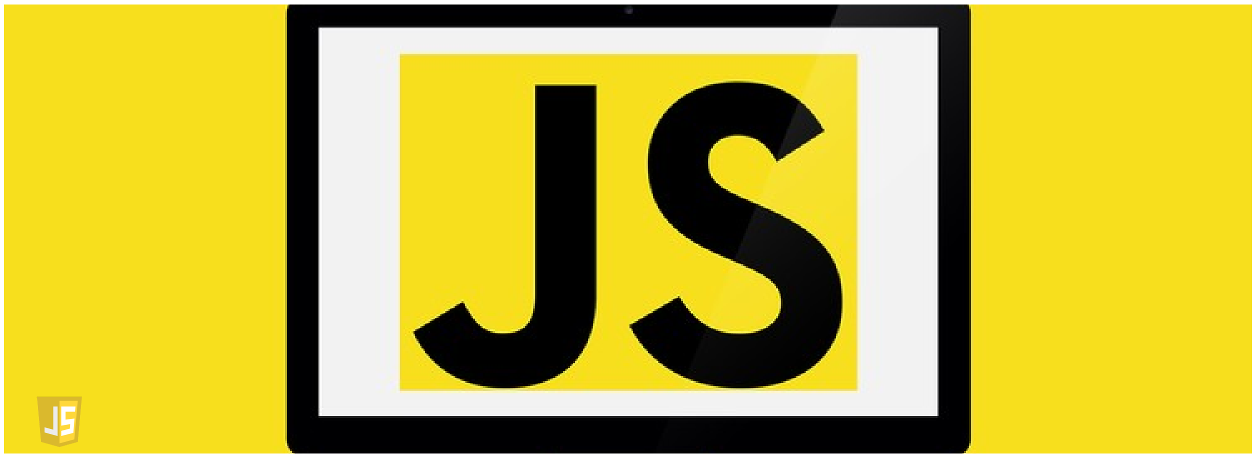

JavaScript

What is JavScript?

JavaScript is the world's most popular programming language.

JavaScript is the programming language of the Web.

JavaScript is easy to learn.

Why Study JavaScript?

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

1. **HTML** to define the content of web pages

2. **CSS** to specify the layout of web pages

3. **JavaScript** to program the behavior of web pages

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:

Example

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

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.

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

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.

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:

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 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>
```

Example

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

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

Example

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

JavaScript variable's

Variables are Containers for Storing Data

JavaScript Variables can be declared in 4 ways:

- Automatically
- Using `var`
- Using `let`
- Using `const`

In this first example, `x`, `y`, and `z` are undeclared variables.

They are automatically declared when first used:

Example

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

From the examples you can guess:

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

Note

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

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

The `var` keyword should only be used in code written for older browsers.

Example using let

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

Example using const

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

Types of JavaScript Operators

There are different types of JavaScript operators:

- Arithmetic Operators
- Assignment Operators
- Comparison Operators
- Logical Operators
- Bitwise Operators

JavaScript Arithmetic Operators

Arithmetic Operators are used to perform arithmetic on numbers:

Arithmetic Operators Example

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

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES2016)
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

JavaScript Assignment Operators

Assignment operators assign values to JavaScript variables.

The **Addition Assignment Operator** (=) adds a value to a variable.

Assignment

```
let x = 10;
x += 5;
```

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

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 String Comparison

All the comparison operators above can also be used on strings:

Example

```
let text1 = "A";
let text2 = "B";
let result = text1 < text2;
```

Note that strings are compared alphabetically:

Example

```
let text1 = "20";
let text2 = "5";
let result = text1 < text2;
```

JavaScript String Addition

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

Example

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

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`

JavaScript Logical Operators

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

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 Comments

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

JavaScript comments can also be used to prevent execution, when testing alternative code.

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 Functions

A JavaScript function is a block of code designed to perform a particular task.

A JavaScript function is executed when "something" invokes it (calls it).

Example

```
// Function to compute the product of p1 and p2
function myFunction(p1, p2) {
return p1 * p2;
}
```

JavaScript Function Syntax

A JavaScript function is defined with the `function` keyword, followed by a **name**, followed by parentheses **()**.

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas:

(parameter1, parameter2, ...)

The code to be executed, by the function, is placed inside curly brackets: **{}**

function *name*(*parameter1*, *parameter2*, *parameter3*) { // *code to be executed*}

Function **parameters** are listed inside the parentheses () in the function definition.

Function **arguments** are the **values** received by the function when it is invoked.

Inside the function, the arguments (the parameters) behave as local variables.

Function Invocation

The code inside the function will execute when "something" **invokes** (calls) the function:

- When an event occurs (when a user clicks a button)
- When it is invoked (called) from JavaScript code
- Automatically (self invoked)

Function Return

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

If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.

Functions often compute a **return value**. The return value is "returned" back to the "caller":

Example

Calculate the product of two numbers, and return the result:

```
// Function is called, the return value will end up in x
let x = myFunction(4, 3);
function myFunction(a, b) {// Function returns the product of a and b
return a * b;
}
```

JavaScript Arrays

An array is a special variable, which can hold more than one value:

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

Why Use Arrays?

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

let car1 = "Saab";let car2 = "Volvo";let car3 = "BMW";

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

Creating an Array

Using an array literal is the easiest way to create a JavaScript Array.

Syntax:

```
const array_name = [item1, item2, ...];
```

Example

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

You can also create an array, and then provide the elements:

Example

```
const cars = [];
cars[0]= "Saab";
cars[1]= "Volvo";
cars[2]= "BMW";
```

Using the JavaScript Keyword new

The following example also creates an Array, and assigns values to it:

Example

```
const cars = new Array("Saab", "Volvo", "BMW");
```

JavaScript Booleans

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

Boolean Values

Very often, in programming, you will need a data type that can only have one of two values, like

- YES / NO
- ON / OFF
- TRUE / FALSE

For this, JavaScript has a **Boolean** data type. It can only take the values **true** or **false**.

The Boolean() Function

You can use the `Boolean()` function to find out if an expression (or a variable) is true:

Example

```
Boolean(10 > 9)
```

Or even easier:

Example

```
(10 > 9)10 >
```

Comparisons and Conditions

The chapter JS Comparisons gives a full overview of comparison operators.

The chapter JS Conditions gives a full overview of conditional statements.

Here are some examples:

Operator	Description	Example
==	equal to	if (day == "Monday")
>	greater than	if (salary > 9000)
<	less than	if (age < 18)

The Boolean value of an expression is the basis for all JavaScript comparisons and conditions.

JavaScript Loops

Loops are handy, if you want to run the same code over and over again, each time with a different value.

Often this is the case when working with arrays:

Instead of writing:

```
text += cars[0] + "<br>";
text += cars[1] + "<br>";
text += cars[2] + "<br>";
text += cars[3] + "<br>";
text += cars[4] + "<br>";
text += cars[5] + "<br>";
```

You can write:

```
for (let i = 0; i < cars.length; i++) {
text += cars[i] + "<br>";
}
```

Different Kinds of Loops

The For Loop

The `for` statement creates a loop with 3 optional expressions:

for (*expression 1*; *expression 2*; *expression 3*) { // *code block to be executed*}

Expression 1 is executed (one time) before the execution of the code block.

Expression 2 defines the condition for executing the code block.

Expression 3 is executed (every time) after the code block has been executed.

Example

```
for (let i = 0; i < 5; i++) {
text += "The number is " + i + "<br>";
}
```

From the example above, you can read:

Expression 1 sets a variable before the loop starts (let i = 0).

Expression 2 defines the condition for the loop to run (i must be less than 5).

Expression 3 increases a value (i++) each time the code block in the loop has been executed.

JavaScript While Loop

The While Loop

The `while` loop loops through a block of code as long as a specified condition is true.

Syntax

while (*condition*) { *// code block to be executed*}

Example

In the following example, the code in the loop will run, over and over again, as long as a variable (i) is less than 10:

Example

```
while (i < 10) {
text += "The number is " + i;  i++;
}
```

The Do While Loop

The `do while` loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax

do { *// code block to be executed*}while (*condition*);

Example

The example below uses a `do while` loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

Example

```
do {  text += "The number is " + i;  i++;}
while (i < 10);
```

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:

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**.

This code assigns a **simple value** (Fiat) to a **variable** named car:

```
let car = "Fiat";
```

Objects are variables too. But objects can contain many values.

This code assigns **many values** (Fiat, 500, white) to a **variable** named car:

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

The values are written as **name:value** pairs (name and value separated by a colon).

Object Definition

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

Example

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

Spaces and line breaks are not important. An object definition can span multiple lines:

Example

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

Object Properties

The **name:values** pairs in JavaScript objects are called **properties**:

Property	Property Value

firstName	John
lastName	Doe
age	50
eyeColor	blue

Accessing Object Properties

You can access object properties in two ways:

```
objectName.propertyName
```

or

```
objectName["propertyName"]
```

Example1

```
person.lastName;
```

Example2

```
person["lastName"];
```

JavaScript objects are containers for **named values** called properties.

Object Methods

Objects can also have **methods**.

Methods are **actions** that can be performed on objects.

Methods are stored in properties as **function definitions**.

Property	Property Value
firstName	John
lastName	Doe
age	50
eyeColor	blue
fullName	function() {return this.firstName + " " + this.lastName;}

A method is a function stored as a property.

Example

```
const person = {
firstName: "John",
lastName : "Doe",
id: 5566,
fullName:
function() {
return this.firstName + " " + this.lastName;
 }
};
```

In the example above, `this` refers to the **person object**.

I.E. **this.firstName** means the **firstName** property of **this**.

I.E. **this.firstName** means the **firstName** property of **person**.