



JavaScript

.NET CORE

JavaScript (JS) is a programming language that conforms to the ECMAScript specification. JavaScript is a high-level language that is just-in-time compiled. It has curly-bracket syntax, dynamic typing, prototype-based object-orientation, and first-class functions.

[HTTPS://EN.WIKIPEDIA.ORG/WIKI/JAVASCRIPT](https://en.wikipedia.org/wiki/JavaScript)

Create Sample .HTML and .js docs

Create a .html doc and create the HTML template inside. It can be used to experiment with the examples in the presentation. The .js file and the .html file must be in the same folder.

```
<!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>JS Example Document</title>
</head>
<body>
  <script src="functions.js"></script>
</body>
</html>
```

Debugging in Chrome

<https://javascript.info/debugging-chrome>

All modern browsers and most other environments support *Debugging Tools*.

Debugging Tools is a special UI in *developer tools* that makes debugging in the browser much easier. It allows to trace the code step-by-step to see what exactly is going on.

Chrome has many features and most other browsers have a similar process.

Follow this [tutorial](#) to learn how to debug in Chrome (or any other browser).



JavaScript – Overview

<https://www.w3schools.com/js/default.asp>

JavaScript was invented by Brendan Eich in 1995 and became an ECMA standard in 1997. **ECMAScript** is the official name of *JavaScript*.

It's one of the 3 languages all web developers must learn:

1. **HTML** defines the content of web pages.
2. **CSS** specifies the layout of web pages.
3. **JavaScript** is for programing the behavior of web pages.

JS is well-known as the scripting language for Web pages, but many desktop and server programs use JavaScript also. Node.js, jQuery, Angular, React and many others are examples of programs that use JS or are libraries of JS.



JavaScript – Overview

<https://developer.mozilla.org/en-US/docs/Web/JavaScript>

JavaScript (JS) is a:

- prototype-based,
- multi-paradigm,
- single-threaded,
- dynamic,
- case-sensitive,
- interpreted (just-in-time compiled)

...programming language that supports:

- object-oriented,
- imperative, and
- declarative (e.g. functional programming)

styles.

Do not confuse JavaScript with the Java programming language.

Although both "Java" and "JavaScript" are trademarks or registered trademarks of Oracle, the two programming languages have very different syntax, semantics, and uses.

JS Versions

https://www.w3schools.com/js/js_versions.asp

It is important to understand that **JavaScript** has changed over time and will continue to change in the future. The major additions to **EMCAScript** have been:

- with EMCA4 when try/catch handling, better string handling, and numeric output formatting were introduced.
- Then with ES6 and classes, 'let' and 'const', iterators, and arrow functions.
- Then with ES8 and Async Functions.

Since ES7 they decided to release a new version every year with iterative improvements.

ECMAScript Editions

| Ver | Official Name | Description |
|-----|--|---|
| 1 | ECMAScript 1 (1997) | First Edition. |
| 2 | ECMAScript 2 (1998) | Editorial changes only. |
| 3 | ECMAScript 3 (1999) | Added Regular Expressions. Added try/catch. |
| 4 | ECMAScript 4 | Never released. |
| 5 | ECMAScript 5 (2009) Read More: JS ES5 | Added "strict mode". Added JSON support. Added String.trim(). Added Array.isArray(). Added Array Iteration Methods. |
| 5.1 | ECMAScript 5.1 (2011) | Editorial changes. |
| 6 | ECMAScript 2015 Read More: JS ES6 | Added let and const. Added default parameter values. Added Array.find(). Added Array.findIndex(). |
| 7 | ECMAScript 2016 | Added exponential operator (**). Added Array.prototype.includes. |
| 8 | ECMAScript 2017 | Added string padding. Added new Object properties. Added Async functions. Added Shared Memory. |
| 9 | ECMAScript 2018 | Added rest / spread properties. Added Asynchronous iteration. Added Promise.finally(). Additions to RegExp. |

Is there an official JS reference?

Nope. We'll use these.

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide>

[The Modern JavaScript Tutorial](#)

<https://en.wikipedia.org/wiki/JavaScript>

JavaScript – Declaring Variables (***var*** and ***let***)

https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/JavaScript_basics

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Variables#The_difference_between_var_and_let

JS creates variables in two ways. '***let***' and '***var***'

Originally, there was only ***var***. The design of ***var*** can be confusing. You can declare a variable with ***var*** after you initialize it and, because of [hoisting](#), it will still work. You can also redeclare a variable multiple times with ***var***.

let was created to fix issues with ***var***.

Use ***let*** (rather than ***var***) unless you need to support versions of IE below v11. ***var*** is function scoped and ***let*** is block scoped. It can be said that a variable declared with ***var*** is defined throughout the program as compared to ***let***.

```
1 | myName = 'Chris';  
2 |  
3 | function logName() {  
4 |     console.log(myName);  
5 | }  
6 |  
7 | logName();  
8 |  
9 | var myName;
```

```
1 | var myName = 'Chris';  
2 | var myName = 'Bob';
```

JavaScript – Variable Declaration Rules

<https://javascript.info/variables#a-variable>

- Variables objects with helper methods. [\(more\)](#)
- You don't have to declare variable types in JavaScript.
- Numbers don't need quotes, but strings and chars do.
- You can declare multiple variables in one line.
- camelCase is conventionally used for variables
- Variables cannot start with a number.
- Variables are case-sensitive.
- Conventionally, chars (0-9, a-z, A-Z) are used for variables.
- Don't use [JS keywords](#).
- Place **"use strict";** at the top of .js files to enforce newer conventions (like declaring a variable before defining it).
- Declare an unchanging variable with **const**.
- Use ALL CAPS for const variables known before compile-time.
- Use meaningful names for variables.
- JS is **dynamically typed**. This means a variable can be a string and then be a number and then be a float.

```
1 let user = 'John', age = 25, message = 'Hello';
```

```
1 const myBirthday = '18.04.1982';
```

```
1 "use strict";  
2  
3 num = 5; // error: num is not defined
```

```
1 const COLOR_RED = "#F00";
```

```
2 let message = "hello";  
3 message = 123456;
```

JavaScript – Primitive DataTypes

<https://javascript.info/types>

| Datatype | Example | Details |
|---|--|--|
| Number (int) | let num = 10; | Operations include * , / , + , - , etc. Includes NaN (not a number) and infinity |
| Number (floating point) | let num1 = 7087.542 | |
| BigInt | 12345678901234567890123456789012345678901234567890n; | Represents any value $> 2^{53}$ or $< -2^{53}$ (16 digits) or ints of arbitrary length. Use 'n' at the end of a BigInt . |
| String | let str1 = "there"; let str2 = 'tiger'; let str3 = `Hey \${str1}, \${str2}`; | Surrounded by quotes. 'str', "str", and `str` (backticks) are valid. Use `str \${otherStr}` for string interpolation . JS has no char type. |
| Boolean | let isBool = true; let isTrue = 2>1; | Only has 2 values. |
| null | let age = null; | A special value which represents "nothing", "empty" or "value unknown". null is <u>not</u> a reference to an object. |
| undefined | let x; //x is undefined | "value is not assigned". |

JavaScript – Object Data Type and Misc.

<https://javascript.info/types>

| Datatype | Example | Details |
|------------------------|--|---|
| <u>Object</u> | <pre>//use a constructor let john = new User(); // build an object let user = { name: "John", age: 30 };</pre> | Objects are used to store collections of data and more complex entities in a key-value pair format. |
| <u>typeof</u> operator | <pre>Console.log(typeof x); Console.log(typeof(x));</pre> | Returns a string of the type of the argument. It's useful when processing different types differently. |
| <u>Symbol</u> | <pre>let id = Symbol("id");</pre> | Object property keys may only be either of string type, or of symbol type. Symbols are guaranteed to be unique. |

Operands and Operators

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/A_first_splash
<https://javascript.info/operators#terms-unary-binary-operand>

An **operand** is to what **operators** are applied.

For instance, in `5 * 2` there are two operands: the left operand is 5 and the right operand is 2.

JavaScript **operators** allow us to perform tests, do math, join strings together, and other such things. A **unary** operator has a single operand (let `x = 4;`), while a **binary** operator has two operands (let `x = y + z;`).

You can also use the `+` operator to join text strings together (in programming, this is called **concatenation**). Try entering the following lines, one at a time in the browser console.

If one operand is a string, the other is converted to a string, too. Try `let new = "hello" + 4;`

Use PEMDAS for order of operations: Parentheses, Exponents, Multiplication, Division, Addition, Subtraction. `10/(3+2)*4+5^2+6-9`

| Operator | Name | Example |
|----------|----------------|---------|
| + | Addition | 6 + 9 |
| - | Subtraction | 20 - 15 |
| * | Multiplication | 3 * 7 |
| / | Division | 10 / 5 |

```
1 let name = 'Bingo';
2 name;
3 let hello = ' says hello!';
4 hello;
5 let greeting = name + hello;
6 greeting;
```

Operator precedence

<https://javascript.info/operators#operator-precedence>

https://developer.mozilla.org/en/JavaScript/Reference/operators/operator_precedence

Execution order is defined by operator precedence. Parentheses override any precedence. $(1 + 2) * 2 = 6$.

There are many operators in JavaScript. Every operator has a corresponding precedence number. The one with the larger number executes first. If the precedence is the same, the execution order is from left to right.

You can also chain assignments.

In **$a = b = c = 2 + 2$** ; **a**, **b**, and **c** == 4.

| Precedence | Name | Sign |
|------------|----------------|------|
| ... | ... | ... |
| 17 | unary plus | + |
| 17 | unary negation | - |
| 15 | multiplication | * |
| 15 | division | / |
| 13 | addition | + |
| 13 | subtraction | - |
| ... | ... | ... |
| 3 | assignment | = |

More Operators

<https://javascript.info/operators>

| Operator | Example | Description |
|----------|-----------------------------|--|
| % | 6%4 == 2 | % is <i>modulus</i> and gives the remainder. |
| ++ | a = 5, a++ == 6 | ++ increments by 1. -- decrements by 1. Placed before the variable, ++ or -- occurs before the action. Placed after the variable, ++ or -- happens after the action. |
| -- | a = 5, a-- == 4 | |
| ** | a == 4, b == 3; a**b == 48. | ** is the exponent operator. a is multiplied by itself b times. |
| += | let n = 2; n += 5 == 7 | Modify-in-place. Shorthand notation to add, subtract, multiply or divide then save the result to the <i>left-hand variable</i> ; |
| -= | let n = 2; n -= 5 == -3 | |
| *= | let n = 2; n *= 5 == 10 | |
| /= | let n = 10; n /= 5 == 2 | |

=, ==, and === Operators

<https://javascript.info/object#copying-by-reference>

| = | == | === |
|---|---------------------------------|------------------------|
| Assignment | Equality (with type conversion) | Strict equality |
| Let a = { }; Let c = { }; let b = a; let d = "13"; let e = 13 | a == b //true | a === b //true |
| | a == c //false | a === c //false |
| | d == e //true | d === e //false |

| Operator | Name | Example |
|----------|---|---|
| === | Strict equality (is it exactly the same?) | <pre>1 5 === 2 + 4 // false 2 'Chris' === 'Bob' // false 3 5 === 2 + 3 // true 4 2 === '2' // false; number versus string</pre> |
| !== | Non-equality (is it not the same?) | <pre>1 5 !== 2 + 4 // true 2 'Chris' !== 'Bob' // true 3 5 !== 2 + 3 // false 4 2 !== '2' // true; number versus string</pre> |
| < | Less than | <pre>1 6 < 10 // true 2 20 < 10 // false</pre> |
| > | Greater than | <pre>1 6 > 10 // false 2 20 > 10 // true</pre> |

Truthy/Falsy

<https://javascript.info/logical-operators>

<https://developer.mozilla.org/en-US/docs/Glossary/Truthy>

<https://developer.mozilla.org/en-US/docs/Glossary/Falsy>

```
1  if (true)
2  if ({} )
3  if ([])
4  if (42)
5  if ("0")
6  if ("false")
7  if (new Date())
8  if (-42)
9  if (12n)
10 if (3.14)
11 if (-3.14)
12 if (Infinity)
13 if (-Infinity)
```

In JavaScript, a **truthy** value is a value that is considered **true** when viewed in a **Boolean** context. All values are **truthy** unless they are defined as **falsy**.

| | |
|-----------|---|
| false | The keyword false |
| 0 | The number zero |
| -0 | The number negative zero |
| 0n | BigInt, when used as a boolean, follows the same rule as a Number. 0n is falsy. |
| "" | Empty string value |
| null | null - the absence of any value |
| undefined | undefined - the primitive value |
| NaN | NaN - not a number |

A **falsy** value is a value that is considered false when viewed in a Boolean context.

```
1  if (false)
2  if (null)
3  if (undefined)
4  if (0)
5  if (-0)
6  if (0n)
7  if (NaN)
8  if ("" )
```

JavaScript – Type Conversion

<https://javascript.info/type-conversions>

Most of the time, operators and functions automatically convert the values given to them to the right type. The three most widely used are conversions to *string*, to *number*, and to *boolean*.

| String(x) | Number(x) | | Boolean(x) | |
|---|--------------------|---|-------------------|--------|
| Any value can be converted to a string. | If the input is... | The result is.... | Input | Result |
| | undefined | NaN | 0 | false |
| | null | 0 | null | |
| | true / false | 1 / 0 | undefined | |
| | string | Whitespaces are ignored. An error gives <i>NaN</i> . | NaN “ ” ‘ ’ | |
| | string | An empty <i>string</i> becomes 0. | anything else | true |

JavaScript – Math Helper Functions

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math

JavaScript has a built-in Math object which contains a small library of mathematical functions and constants.

| Function | Description | Example |
|--------------------|--|--|
| Math.random(); | returns a floating-point, pseudo-random number in the range 0 to less than 1 | Math.random()*10; //3.229976827519583 |
| Math.abs(x); | returns the absolute value of a number | Math.abs(-10 - 6.3); //16.3 |
| Math.pow(x,y); | returns x to the power of y | Math.pow(7, 3); //343 |
| Math.floor(x); | returns the largest integer less than or equal to a given number | Math.floor(5.05) //5 |
| Math.ceil(x); | rounds a number up to the next largest whole number or integer. | Math.ceil(11.324); //12 |
| Math.max(a,b,...z) | returns the largest of zero or more numbers | Math.max(1, 3, 2); //3 |

Map

<https://javascript.info/map-set#map>

<https://javascript.info/weakmap-weakset#weakmap>

Map

Map is a collection of **key-value** data items, just like an **Object**.

Any type of key is possible, even **Object**.

Insertion order is used for iteration order.

```
1  let map = new Map();
2
3  map.set('1', 'str1');    // a string key
4  map.set(1, 'num1');      // a numeric key
5  map.set(true, 'bool1'); // a boolean key
6
7  // remember the regular Object? it would convert keys to strings
8  // Map keeps the type, so these two are different:
9  alert( map.get(1) );    // 'num1'
10 alert( map.get('1') );  // 'str1'
11
12 alert( map.size );      // 3
```


Weak Map

<https://javascript.info/weakmap-weakset#weakmap>

WeakMap

WeakMap keys must be objects

There is no way to get all **keys** or **values** from a **map**

If you remove all other references to an **object key**, the **object** is removed from memory and the **Map()**.

WeakMap does not support iteration or the methods **keys()**, **values()**, **entries()**

```
1 let john = { name: "John" };
2
3 let weakMap = new WeakMap();
4 weakMap.set(john, "...");
5
6 john = null; // overwrite the reference
7
8 // john is removed from memory!
```

```
1 let weakMap = new WeakMap();
2
3 let obj = {};
4
5 weakMap.set(obj, "ok"); // works fine (object key)
6
7 // can't use a string as the key
8 weakMap.set("test", "Whoops"); // Error, because "test" is not an object
```

Set

<https://javascript.info/map-set#set>

Set

A **Set** is a special type collection – “set of *values*” (without *keys*), where each *value* may occur only once.

A **Set** is analogous to an *array* of strings with code to check for duplicate names.

```
1 let set = new Set();
2
3 let john = { name: "John" };
4 let pete = { name: "Pete" };
5 let mary = { name: "Mary" };
6
7 // visits, some users come multiple times
8 set.add(john);
9 set.add(pete);
10 set.add(mary);
11 set.add(john);
12 set.add(mary);
13
14 // set keeps only unique values
15 alert( set.size ); // 3
16
17 for (let user of set) {
18     alert(user.name); // John (then Pete and Mary)
19 }
```

Weak Set

<https://javascript.info/weakmap-weakset#weakset>

Weak Set

Just like **Set** but only **Objects** are allowed.

An object exists in the set while it is reachable from somewhere else.

Being “weak”, it serves as an additional storage. But only for “yes/no” facts. (use **.has(obj)** helper function).

WeakSet is not iterable and does not support **.size()**, or **.keys()**

```
1 let visitedSet = new WeakSet();
2
3 let john = { name: "John" };
4 let pete = { name: "Pete" };
5 let mary = { name: "Mary" };
6
7 visitedSet.add(john); // John visited us
8 visitedSet.add(pete); // Then Pete
9 visitedSet.add(john); // John again
10
11 // visitedSet has 2 users now
12
13 // check if John visited?
14 alert(visitedSet.has(john)); // true
15
16 // check if Mary visited?
17 alert(visitedSet.has(mary)); // false
18
19 john = null;
20
21 // visitedSet will be cleaned automatically
```

JSON and JSON Methods

<https://javascript.info/json>

JSON (JavaScript Object Notation) is a general format to represent values and objects. Initially it was made for JavaScript, but many other languages have libraries to handle it as well. JSON is used for data exchange. JavaScript provides two JSON methods:

- `JSON.stringify` to convert objects into JSON.
- `JSON.parse` to convert JSON back into an object.

In the example to the right, the method ***JSON.stringify(student)*** takes the object and converts it into a string. The *json* string is called a JSON-encoded or *serialized* or *stringified* object. It is ready to be sent over the wire or put into a plain data store. Pay close attention to the format.

```
1  let student = {
2    name: 'John',
3    age: 30,
4    isAdmin: false,
5    courses: ['html', 'css', 'js'],
6    wife: null
7  };
8
9  let json = JSON.stringify(student);
10
11 alert(typeof json); // we've got a string!
12
13 alert(json);
14 /* JSON-encoded object:
15 {
16   "name": "John",
17   "age": 30,
18   "isAdmin": false,
19   "courses": ["html", "css", "js"],
20   "wife": null
21 }
22 */
```

JSON.parse

<https://javascript.info/json#json-parse>

JSON.parse decodes a *JSON* string.

The *JSON* may be as complex as necessary, objects and arrays can include other objects and arrays. But they must obey the same *JSON* format.

```
5    let userData = '{ "name": "John", "age": 35,  
6    "isAdmin": false, "friends": [0,1,2,3] }';  
7  
8    let user = JSON.parse(userData);  
9  
10   alert( user.friends[1] ); // 1
```

User Interaction in a browser – alert, prompt, confirm

<https://javascript.info/alert-prompt-confirm>

The browser functions ***alert()***, ***prompt()*** and ***confirm()*** allow interaction with and input from the user.

| alert(message) | prompt(title, [default]) | confirm() |
|---|---|--|
| This shows a message and pauses script execution until the user presses “OK”. | Shows a modal window with a text message, an input field for the visitor, and the buttons OK/Cancel. Default is the initial value for the input field. | The function confirm shows a modal window with a question and two buttons: OK and Cancel. |