## Topics

| |
|---|
| **Basics of JavaScript** |
| **JavaScript Introduction** |
| **Use of JavaScript** |
| **Way to include javascript** |
| **Syntax of JavaScript** |
| **Basic event of javascript** |
| **Basic Validation with javascript** |
| |
| **Prepare Demo of Above Topics** |

# An Introduction to JavaScript

### What is JavaScript?

*JavaScript* was initially created to "make web pages alive".

The programs in this language are called *scripts*. They can be written right in a web page's HTML and run automatically as the page loads.

Scripts are provided and executed as plain text. They don't need special preparation or compilation to run.

In this aspect, JavaScript is very different from another language called Java.

Today, JavaScript can execute not only in the browser, but also on the server, or actually on any device that has a special program called the JavaScript engine.

## Why is it called JavaScript?

When JavaScript was created, it initially had another name: "LiveScript". But Java was very popular at that time, so it was decided that positioning a new language as a "younger brother" of Java would help.

But as it evolved, JavaScript became a fully independent language with its own specification called ECMAScript, and now it has no relation to Java at all.

## What can in-browser JavaScript do?

Modern JavaScript is a "safe" programming language. It does not provide low-level access to memory or CPU, because it was initially created for browsers which do not require it.

JavaScript's capabilities greatly depend on the environment it's running in. For instance, Node.js supports functions that allow JavaScript to read/write arbitrary files, perform network requests, etc.

In-browser JavaScript can do everything related to webpage manipulation, interaction with the user, and the webserver.

For instance, in-browser JavaScript is able to:

· Add new HTML to the page, change the existing content, modify styles.
· React to user actions, run on mouse clicks, pointer movements, key presses.
· Send requests over the network to remote servers, download and upload files (so-called AJAX and COMET technologies).
· Get and set cookies, ask questions to the visitor, show messages.
· Remember the data on the client-side ("local storage").

## What makes JavaScript unique?

· Full integration with HTML/CSS.
· Simple things are done simply.
· Support by all major browsers and enabled by default.

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript in Body</h2>
<p id="demo"></p>
```

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

```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction() {
document.getElementById("demo").innerHTML = "Paragraph changed."; }
</script>
</head>
<body>
<h1>A Web Page</h1>
<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try it</button> </body>
</html>
```

## Variables

A variable is a "named storage" for data. We can use variables to store goodies, visitors, and other data.

To create a variable in JavaScript, use the let keyword.

The statement below creates (in other words: *declares*) a variable with the name "message": let message;

Now, we can put some data into it by using the assignment operator =:

```
let message;

message = 'Hello'; // store the string
```

The string is now saved into the memory area associated with the variable. We can access it using the variable name:

```
let message;
message = 'Hello!';

alert(message); // shows the variable content
```

To be concise, we can combine the variable declaration and assignment into a single line:

```
        let message = 'Hello!'; // define the variable and assign the value
    alert(message); // Hello!
```

We can also declare multiple variables in one line:

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

## Variable naming

There are two limitations on variable names in JavaScript:

1. The name must contain only letters, digits, or the symbols $ and _.
2. The first character must not be a digit.

```
let userName;
let test123;
```

When the name contains multiple words, camelCase is commonly used. That is: words go one after another, each word except first starting with a capital letter: myVeryLongName.

What's interesting – the dollar sign '$' and the underscore '_' can also be used in names. They are regular symbols, just like letters, without any special meaning.

These names are valid:

```
<!DOCTYPE html>
<html>
<head>
<script>
let $ = 1; // declared a variable with the name "$" let _ = 2; // and now a variable with the name "_"
alert($ + _); // 3
</script>
</head>
<body>
</body>
</html>
```

Examples of incorrect variable names:

```
let 1a; // cannot start with a digit

let my-name; // hyphens '-' aren't allowed in the name
```

Variables named apple and AppLE are two different variables.

## Constants

To declare a constant (unchanging) variable, use const instead of let:

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

Variables declared using const are called “constants”. They cannot be reassigned. An attempt to  do so would cause an error:

```
const myBirthday = '18.04.1982';

myBirthday = '01.01.2001'; // error, can't reassign the constant!
```

## Data types

A value in JavaScript is always of a certain type. For example, a string or a number. Here, we’ll cover them in general term.

We can put any type in a variable. For example, a variable can at one moment be a string and  then store a number:

```
// no error
let message = "hello";
message = 123456;
```

Programming languages that allow such things, such as JavaScript, are called “dynamically  typed”, meaning that there exist data types, but variables are not bound to any of them.

### Number

The *number* type represents both integer and floating point numbers.

There are many operations for numbers, e.g. multiplication *, division /, addition +, subtraction -, and so on.

Besides regular numbers, there are so-called "special numeric values" which also belong to this data type:

Infinity, -Infinity and NaN.

· Infinity

represents the mathematical Infinity ∞. It is a special value that's greater than any number.

```
alert( 1 / 0 ); // Infinity
alert( Infinity ); // Infinity
alert( "not a number" / 2 ); // NaN, such division is erroneous alert(
"not a number" / 2 + 5 ); // NaN
```

## String

A string in JavaScript must be surrounded by quotes.

```
let str = "Hello";
let str2 = 'Single quotes are ok too';
let phrase = `can embed another ${str}`;
```

In JavaScript, there are 3 types of quotes.

1. Double quotes: "Hello".
2. Single quotes: 'Hello'.
3. Backticks: `Hello`.

Double and single quotes are "simple" quotes. There's practically no difference between them in JavaScript.

Backticks are "extended functionality" quotes. They allow us to embed variables and expressions into a string by wrapping them in ${…}, for example:

```
let name = "John";
// embed a variable
alert( `Hello, ${name}!` ); // Hello, John!
```

```
// embed an expression
alert( `the result is ${1 + 2}` ); // the result is 3
```

There is no *character* type.

In some languages, there is a special "character" type for a single character. For example, in the  C language and in Java it is called "char".

In JavaScript, there is no such type. There's only one type: string. A string may consist of zero  characters (be empty), one character or many of them.

### Boolean (logical type)

The boolean type has only two values: true and false.

This type is commonly used to store yes/no values: true means "yes, correct", and false means "no, incorrect".

For instance:

```
let nameFieldChecked = true; // yes, name field is checked
let ageFieldChecked = false; // no, age field is not checked


let isGreater = 4 > 1;

alert( isGreater ); // true (the comparison result is "yes")
```

## Data Type Conversion

Most of the time, operators and functions automatically convert the values given to them to the  right type.

For example, alert automatically converts any value to a string to show it.
Mathematical  operations convert values to numbers.

There are also cases when we need to explicitly convert a value to the expected type. String Conversion

String conversion happens when we need the string form of a value.

For example, alert(value) does it to show the value.

We can also call the String(value) function to convert a value to a string:

```
let value = true;
alert(typeof value); // boolean

value = String(value); // now value is a string "true"
alert(typeof value); // string
```

## Numeric Conversion

Numeric conversion happens in mathematical functions and expressions automatically. For example, when division / is applied to non-numbers:

```
alert( "6" / "2" ); // 3, strings are converted to numbers
```

We can use the Number(value) function to explicitly convert a value to a number:

```
let str = "123";
alert(typeof str); // string

let num = Number(str); // becomes a number 123
alert(typeof num); // number
```

Explicit conversion is usually required when we read a value from a string-based source like a  text form but expect a number to be entered.

If the string is not a valid number, the result of such a conversion is NaN. For instance:

```
let age = Number("an arbitrary string instead of a number");
    alert(age); // NaN, conversion failed
```

**Numeric conversion rules:**

| Value | Becomes... |
|---|---|
| undefined | NaN |
| null | 0 |
| true and false | 1 and 0 |

| | |
|---|---|
| string | Whitespaces from the start and end are removed. If the remaining string is empty, the result is 0. Otherwise, the number is "read" from the string. An error gives NaN. |

```
alert( Number(" 123 ") ); // 123
alert( Number("123z") ); // NaN (error reading a number at "z") alert(
Number(true) ); // 1
alert( Number(false) ); // 0
```

### Boolean Conversion

Boolean conversion is the simplest one.

It happens in logical operations (later we'll meet condition tests and other similar things) but can also be performed explicitly with a call to Boolean(value).

The conversion rule:

· Values that are intuitively "empty", like 0, an empty string, null, undefined, and NaN, become false.

· Other values become true.

For instance:

```
alert( Boolean(1) ); // true
alert( Boolean(0) ); // false

alert( Boolean("hello") ); // true
alert( Boolean("") ); // false
```

## Decision Making Statements

### The "if" statement

The if(...) statement evaluates a condition in parentheses and, if the result is true, executes a block of code.

```
let year = prompt('In which year was ECMAScript-2015 specification
published?', '');
```

```
if (year == 2015) {
 alert( "That's correct!" );
 alert( "You're so smart!" );
 }
```

### The “else” clause

The if statement may contain an optional “else” block. It executes when the condition is falsy.

```
let year = prompt('In which year was the ECMAScript-2015 specification
published?', '');

if (year < 2015) {
 alert( 'Too early...' );
 } else if (year > 2015) {
 alert( 'Too late' );
 } else {
 alert( 'Exactly!' );
 }
```

### Conditional operator ‘?’

The so-called “conditional” or “question mark” operator lets us do that in a shorter and simpler  way.

The operator is represented by a question mark ?. Sometimes it’s called “ternary”, because the  operator has three operands. It is actually the one and only operator in JavaScript which has that  many.

```
let accessAllowed = (age > 18) ? true : false;
```

## Logical operators

There are four logical operators in JavaScript: || (OR), && (AND), ! (NOT),

Although they are called “logical”, they can be applied to values of any type, not only boolean.  Their result can also be of any type.

### || (OR)

In classical programming, the logical OR is meant to manipulate boolean values only. If any of  its arguments are true, it returns true, otherwise it returns false.

In JavaScript, the operator is a little bit trickier and more powerful. But first, let’s see what  happens with boolean values.

There are four possible logical combinations:

```
alert( true || true ); // true
alert( false || true ); // true
alert( true || false ); // true
alert( false || false ); // false

if (1 || 0) { // works just like if( true || false )
 alert( 'truthy!' );
}

let hour = 12;
let isWeekend = true;
if (hour < 10 || hour > 18 || isWeekend) {
 alert( 'The office is closed.' ); // it is the weekend
}
```

## && (AND)

The AND operator is represented with two ampersands &&:

```
alert( true && true ); // true
alert( false && true ); // false
alert( true && false ); // false
alert( false && false ); // false


let hour = 12;
let minute = 30;
if (hour == 12 && minute == 30) {
 alert( 'The time is 12:30' );
}
```

## ! (NOT)

The boolean NOT operator is represented with an exclamation sign !.

The operator accepts a single argument and does the following:

1. Converts the operand to boolean type: true/false.
2. Returns the inverse value.

```
alert( !true ); // false
alert( !0 ); // true
```

# Loops: while and for

We often need to repeat actions.

For example, outputting goods from a list one after another or just running the same code for each number from 1 to 10.

*Loops* are a way to repeat the same code multiple times.

## The "while" loop

The while loop has the following syntax:

```
while (condition) {
 // code
 // so-called "loop body"
 }
```

```
let i = 0;
while (i < 3) { // shows 0, then 1, then 2
 alert( i );
 i++;
 }
```

A single execution of the loop body is called *an iteration*. The loop in the example above makes three iterations.

If i++ was missing from the example above, the loop would repeat (in theory) forever. In practice, the browser provides ways to stop such loops, and in server-side JavaScript, we can kill the process.

Any expression or variable can be a loop condition, not just comparisons: the condition is evaluated and converted to a boolean by while.

## The "for" loop

The for loop is more complex, but it's also the most commonly used loop.

```
for (begin; condition; step) {
// ... loop body ...
}
```

```
for (let i = 0; i < 3; i++) { // shows 0, then 1, then 2
alert(i);
}
```

| | | |
|---|---|---|
| begin | i = 0 | Executes once upon entering the loop. |
| condition | i < 3 | Checked before every loop iteration. If false, the loop stops. |
| body | alert(i) | Runs again and again while the condition is truthy. |
| step | i++ | Executes after the body on each iteration. |

```
let i = 0; // we have i already declared and assigned
for (; i < 3; i++) { // no need for "begin"
alert( i ); // 0, 1, 2
}
```

## Functions

Quite often we need to perform a similar action in many places of the script.

For example, we need to show a nice-looking message when a visitor logs in, logs out and maybe somewhere else.

Functions are the main "building blocks" of the program. They allow the code to be called many times without repetition.

We've already seen examples of built-in functions, like alert(message), prompt(message, default) and confirm(question). But we can create functions of our own as well.

### Function Declaration

```
function name(parameters) {
...body...
}

function showMessage() {
alert( 'Hello everyone!' );
}
```

## Local variables

A variable declared inside a function is only visible inside that function.

```
function showMessage() {
let message = "Hello, I'm JavaScript!"; // local variable

   alert( message );
}

showMessage(); // Hello, I'm JavaScript!

alert( message ); // <-- Error! The variable is local to the function


let userName = 'John';

function showMessage() {
let message = 'Hello, ' + userName;
alert(message);
}

showMessage(); // Hello, John
```

## Outer variables

A function can access an outer variable as well, for example:

```
let userName = 'John';

function showMessage() {
let message = 'Hello, ' + userName;
alert(message);
}

showMessage(); // Hello, John
```

## Summary

- **Basics of Java Script** ➖
    - JavaScript is a scripting language that enables you to dynamically update content, control multimedia, animate images, and pretty much everything else.
    - JavaScript is an open-source programming language designed for creating web-centric applications. It is lightweight and interpreted which makes it much faster than other languages and is integrated with HTML making it easier to implement in web applications.

    - Java Script is a dynamically typed language.

- **What is JavaScript ?**

  JavaScript is a *lightweight, cross-platform*, *single-threaded,* and *interpreted compiled* programming language. It is also known as the scripting language for web pages. It is well-known for the development of web pages, and many non-browser environments also use it.

JavaScript is a weakly typed language (dynamically typed). JavaScript can be used for Client-side developments as well as Server-side developments. JavaScript is both an imperative and declarative type of language. JavaScript contains a standard library of objects, like Array, Date, and Math, and a core set of language elements like operators, control structures, and statements.

- **Client-side**: It supplies objects to control a browser and its Document Object Model (DOM). Like if client-side extensions allow an application to place elements on an HTML form and respond to user events such as mouse clicks, form input, and page navigation. Useful libraries for the client side are AngularJS, ReactJS, VueJS, and so many others.
- **Server-side:** It supplies objects relevant to running JavaScript on a server. For if the server-side extensions allow an application to communicate with a database, and provide continuity of information from one invocation to another of the application, or perform file manipulations on a server. The useful framework which is the most famous these days is node.js.
- **Imperative language** – In this type of language we are mostly concerned about how it is to be done. It simply controls the flow of computation. The procedural programming approach, object oriented approach comes under this as async await we are thinking about what is to be done further after the async call.
- **Declarative programming** – In this type of language we are concerned about how it is to be done, basically here logical computation requires. Her main goal is to describe the desired result without direct dictation on how to get it as the arrow function does.

- **Use Of JavaScript** ➖
    - Web development
    - UI Development

    - Web application
    - Real time Updates
    - Data manipulation
    - Browser manipulation
    - Form validation
    - API Integration

- **Way to include javascript**

    - **Inline JS**
        - **Inline Script:** You can include JavaScript directly within the HTML document using the <script> tag. Place the JavaScript code between the opening <script> tag and the closing </script> tag. This method is useful for small scripts or for quick testing, but it's generally recommended to separate your JavaScript code from your HTML for better maintainability.
    - **External js**
        - **External Script**: It's a good practice to keep your JavaScript code in separate files and include them in your HTML document using the src attribute of the <script> tag. This allows for better organization, reuse, and caching of your JavaScript code.
    - **Modern Module System (ES6 Modules)**
        - With the adoption of ES6 modules, you can use the import and export statements to organize and load JavaScript code across different files. This approach promotes modularity and better code management.

- **Syntax of JavaScript**
    - // single line Comments
    - /* .... */ multiline comments
    - let keyword for variable declaration.
    - const keyword is used for declaring Constant (Value can not be changed)

    - **Operators**
        - Arithmetic Operators ( + - * / // ** % ++ – )
        - Assignment Operators ( += -= *= /= **= %= )
        - Comparison Operators ( == === != !== > >= < <= )
        - String Operators ( += assignment concatenation , + concatenation , < strings are compared alphabetically )
        - Logical Operators ( && and , || or , ! not )
        - Bitwise Operators ( & and , | or , ~ not , ^ XOR , << left shift , >> right shift ,

        >>>  unsigned right shift )
    - Ternary Operators ( ? )
    - Type Operators ( typeof  ,  instanceof )

- **Condition ➖**
    - **if (condition) {**

        **// Code to execute if the condition is true**

    - **} else if (anotherCondition) {**

        **// Code to execute if the second condition is true**

    - **} else {**

        **// Code to execute if no conditions are true**

    - **}**
- **Loops ➖**

    **// While loop**

    - **while (condition) {**

        **// Code to repeat while the condition is true**

    **}**

    **// For loop**

    - **for (let i = 0; i < 5; i++) {**

        **// Code to execute in each iteration**

    **}**

    **// For...of loop (for arrays and iterable objects)**

    - **for (let item of myArray) {**

    **// Code to execute for each item in the array**

    **}**

- **Functions ➖**

    **// Function declaration**

    - **function myFunction(parameter1, parameter2) {**

```
            // Function body
            return parameter1 + parameter2;
            }
        // Function expression (anonymous function)
```

- let multiply = function(x, y) {

```
            return x * y;
            };
```

- Arrays
  - let colors = ["red", "green", "blue"];
  - console.log(colors[0]); // Accessing array elements
  - colors.push("yellow"); // Adding an element to the end
  - colors.pop(); // Removing the last element

- Object and Properties
  - let person = {

```
            firstName: "John",
            lastName: "Doe",
            age: 30,
            sayHello: function() {
                console.log("Hello!");
                }
        };
```

  - console.log(person.firstName); // Accessing object property
  - person.sayHello(); // Calling a method