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.

The browser has an embedded engine sometimes called a “JavaScript virtual machine”.

Different engines have different “codenames”. For example:

* V8 – in Chrome and Opera.
* SpiderMonkey – in Firefox.
* …There are other codenames like “Trident” and “Chakra” for different versions of IE, “ChakraCore” for Microsoft Edge, “Nitro” and “SquirrelFish” for Safari, etc.

The terms above are good to remember because they are used in developer articles on the internet. We’ll use them too. For instance, if “a feature X is supported by V8”, then it probably works in Chrome and Opera.

What CAN’T in-browser JavaScript do?

JavaScript’s abilities in the browser are limited for the sake of the user’s safety. The aim is to prevent an evil webpage from accessing private information or harming the user’s data.

Examples of such restrictions include:

* JavaScript on a webpage may not read/write arbitrary files on the hard disk, copy them or execute programs. It has no direct access to OS functions.

Modern browsers allow it to work with files, but the access is limited and only provided if the user does certain actions, like “dropping” a file into a browser window or selecting it via an <input> tag.

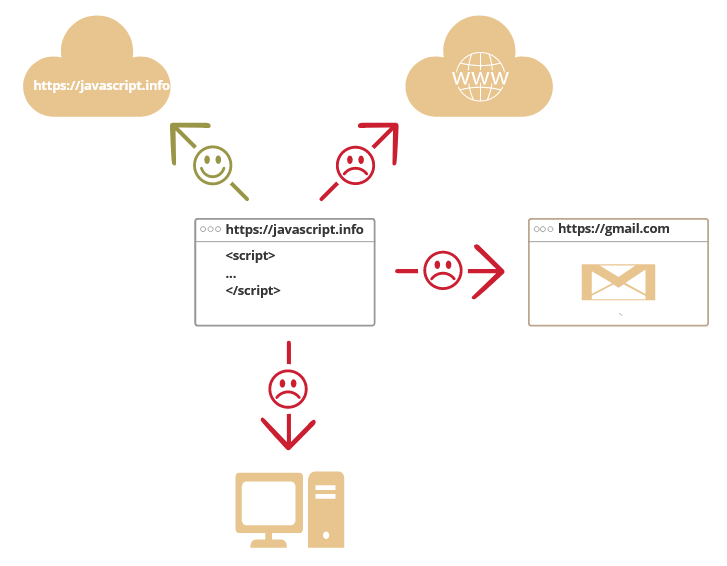
There are ways to interact with camera/microphone and other devices, but they require a user’s explicit permission. So a JavaScript-enabled page may not sneakily enable a web-camera, observe the surroundings and send the information to the NSA.

* Different tabs/windows generally do not know about each other. Sometimes they do, for example when one window uses JavaScript to open the other one. But even in this case, JavaScript from one page may not access the other if they come from different sites (from a different domain, protocol or port).

This is called the “Same Origin Policy”. To work around that, *both pages* must agree for data exchange and contain a special JavaScript code that handles it. We’ll cover that in the tutorial.

This limitation is, again, for the user’s safety. A page from http://anysite.com which a user has opened must not be able to access another browser tab with the URL http://gmail.com and steal information from there.

* JavaScript can easily communicate over the net to the server where the current page came from. But its ability to receive data from other sites/domains is crippled. Though possible, it requires explicit agreement (expressed in HTTP headers) from the remote side. Once again, that’s a safety limitation.



Such limits do not exist if JavaScript is used outside of the browser, for example on a server. Modern browsers also allow plugin/extensions which may ask for extended permissions.

What makes JavaScript unique?

There are at least *three* great things about JavaScript:

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

JavaScript is the only browser technology that combines these three things.

That’s what makes JavaScript unique. That’s why it’s the most widespread tool for creating browser interfaces.

That said, JavaScript also allows to create servers, mobile applications, etc.

Languages “over” JavaScript

The syntax of JavaScript does not suit everyone’s needs. Different people want different features.

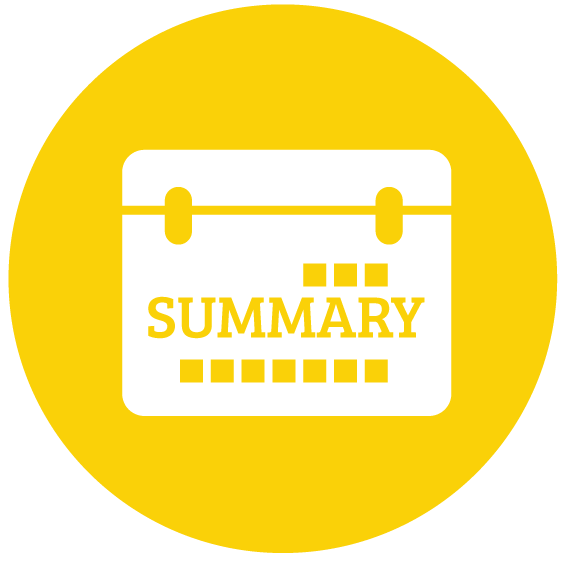
That’s to be expected, because projects and requirements are different for everyone.

So recently a plethora of new languages appeared, which are *transpiled* (converted) to JavaScript before they run in the browser.

Modern tools make the transpilation very fast and transparent, actually allowing developers to code in another language and auto-converting it “under the hood”.

Examples of such languages:

* CoffeeScript is a “syntactic sugar” for JavaScript. It introduces shorter syntax, allowing us to write clearer and more precise code. Usually, Ruby devs like it.
* TypeScript is concentrated on adding “strict data typing” to simplify the development and support of complex systems. It is developed by Microsoft.
* Flow also adds data typing, but in a different way. Developed by Facebook.
* Dart is a standalone language that has its own engine that runs in non-browser environments (like mobile apps), but also can be transpiled to JavaScript. Developed by Google.

 Summary:

* JavaScript was initially created as a browser-only language, but is now used in many other environments as well.
* Today, JavaScript has a unique position as the most widely-adopted browser language with full integration with HTML/CSS.
* There are many languages that get “transpiled” to JavaScript and provide certain features. It is recommended to take a look at them, at least briefly, after mastering JavaScript.

Data types

A value in JavaScript is always of a certain type. For example, a string or a number.

There are eight basic data types in JavaScript. Here, we’ll cover them in general and in the next chapters we’ll talk about each of them in detail.

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

let n = 123;   
n = 12.345;

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.

We can get it as a result of division by zero:

alert( 1 / 0 ); // Infinity

         Or just reference it directly:

alert( Infinity ); // Infinity

* NaN represents a computational error. It is a result of an incorrect or an undefined mathematical operation, for instance:

alert( "not a number" / 2 ); // NaN, such division is erroneous

* NaN is sticky. Any further operation on NaN returns NaN:

alert( "not a number" / 2 + 5 ); // NaN

So, if there’s a NaN somewhere in a mathematical expression, it propagates to the whole result.

BigInt

In JavaScript, the “number” type cannot represent integer values larger than (253-1) (that’s 9007199254740991), or less than -(-253-1) for negatives. It’s a technical limitation caused by their internal representation.

For most purposes that’s quite enough, but sometimes we need really big numbers, e.g. for cryptography or microsecond-precision timestamps.

BigInt type was recently added to the language to represent integers of arbitrary length.

A BigInt value is created by appending n to the end of an integer:

// the "n" at the end means it's a BigInt   
const bigInt = 1234567890123456789012345678901234567890n;

As BigInt numbers are rarely needed, we don’t cover them here, but devoted them a separate chapter BigInt. Read it when you need such big numbers.

**Compatability issues:***Right now BigInt is supported in Firefox/Chrome/Edge, but not in Safari/IE.*

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

The expression inside ${…} is evaluated and the result becomes a part of the string. We can put anything in there: a variable like name or an arithmetical expression like 1 + 2 or something more complex.

Please note that this can only be done in backticks. Other quotes don’t have this embedding functionality!

alert( "the result is ${1 + 2}" ); // the result is ${1 + 2} (double quotes do nothing)

**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 only 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

Boolean values also come as a result of comparisons:

let isGreater = 4 > 1;   
alert( isGreater ); // true (the comparison result is "yes")

The “null” value

The special null value does not belong to any of the types described above.

It forms a separate type of its own which contains only the null value:

let age = null;

In JavaScript, null is not a “reference to a non-existing object” or a “null pointer” like in some other languages.

It’s just a special value which represents “nothing”, “empty” or “value unknown”.

The code above states that age is unknown.

The “undefined” value

The special value undefined also stands apart. It makes a type of its own, just like null.

The meaning of undefined is “value is not assigned”.

If a variable is declared, but not assigned, then its value is undefined:

let age; alert(age); // shows "undefined"

Technically, it is possible to explicitly assign undefined to a variable:

let age = 100;   
  
// change the value to undefined   
age = undefined;  
   
alert(age); // "undefined"

***…But we don’t recommend doing that. Normally, one uses null to assign an “empty” or “unknown” value to a variable, while undefined is reserved as a default initial value for unassigned things.***

Objects and Symbols

The object type is special.

All other types are called “primitive” because their values can contain only a single thing (be it a string or a number or whatever). In contrast, objects are used to store collections of data and more complex entities.

Being that important, objects deserve a special treatment. We’ll deal with them later in the chapter Objects, after we learn more about primitives.

The symbol type is used to create unique identifiers for objects. We have to mention it here for the sake of completeness, but also postpone the details till we know objects.

The typeof operator

The typeof operator returns the type of the argument. It’s useful when we want to process values of different types differently or just want to do a quick check.

It supports two forms of syntax:

1. As an operator: typeof x.
2. As a function: typeof(x).

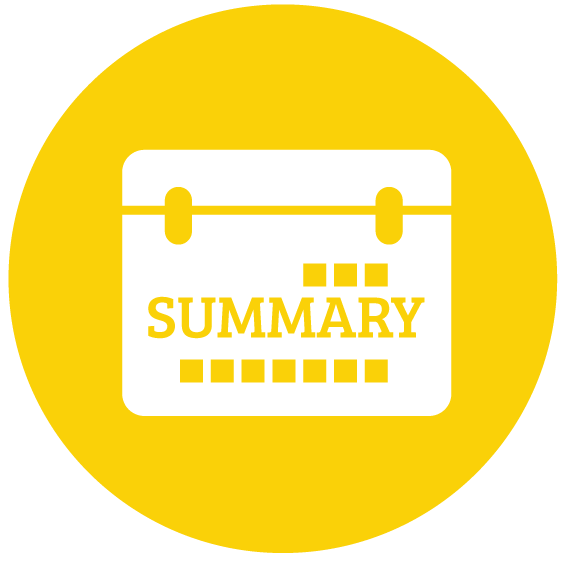
In other words, it works with parentheses or without them. The result is the same.

The call to typeof x returns a string with the type name:

typeof undefined // "undefined"   
typeof 0 // "number"   
typeof 10n // "bigint"   
typeof true // "boolean"   
typeof "foo" // "string"   
typeof Symbol("id") // "symbol"   
typeof Math // "object" (1)   
typeof null // "object" (2)   
typeof alert // "function" (3)

The last three lines may need additional explanation:

1. Math is a built-in object that provides mathematical operations. We will learn it in the chapter Numbers. Here, it serves just as an example of an object.
2. The result of typeof null is "object". That’s an officially recognized error in typeof behavior, coming from the early days of JavaScript and kept for compatibility. Definitely, null is not an object. It is a special value with a separate type of its own.
3. The result of typeof alert is "function", because alert is a function. We’ll study functions in the next chapters where we’ll also see that there’s no special “function” type in JavaScript. Functions belong to the object type. But typeof treats them differently, returning "function". That also comes from the early days of JavaScript. Technically, such behavior isn’t correct, but can be convenient in practice.

 Summary:

There are 8 basic data types in JavaScript.

* number for numbers of any kind: integer or floating-point, integers are limited by ±253.
* bigint is for integer numbers of arbitrary length.
* string for strings. A string may have zero or more characters, there’s no separate single-character type.
* boolean for true/false.
* null for unknown values – a standalone type that has a single value null.
* undefined for unassigned values – a standalone type that has a single value undefined.
* object for more complex data structures.
* symbol for unique identifiers.

The typeof operator allows us to see which type is stored in a variable.

* Two forms: typeof x or typeof(x).
* Returns a string with the name of the type, like "string".
* For null returns "object" – this is an error in the language, it’s not actually an object.

**JavaScript Operators**

JavaScript includes operators as in other languages. An operator performs some operation on single or multiple operands (data value) and produces a result. For example 1 + 2, where + sign is an operator and 1 is left operand and 2 is right operand. + operator adds two numeric values and produces a result which is 3 in this case.

Syntax:

<*Left operand*> operator <*right operand*>

<*Left operand*> operator

JavaScript includes following categories of operators.

1. Arithmetic Operators
2. Comparison Operators
3. Logical Operators
4. Assignment Operators
5. Conditional Operators

Arithmetic Operators

Arithmetic operators are used to perform mathematical operations between numeric operands.

| **Operator** | **Description** |
| --- | --- |
| + | Adds two numeric operands. |
| - | Subtract right operand from left operand |
| \* | Multiply two numeric operands. |
| / | Divide left operand by right operand. |
| % | Modulus operator. Returns remainder of two operands. |
| ++ | Increment operator. Increase operand value by one. |
| -- | Decrement operator. Decrease value by one. |

The following example demonstrates how arithmetic operators perform different tasks on operands.

Example: Arithmetic Operator

var x = 5, y = 10, z = 15;

x + y; //returns 15

y - x; //returns 5

x \* y; //returns 50

y / x; //returns 2

x % 2; //returns 1

x++; //returns 6

x--; //returns 4

**+** operator performs concatenation operation when one of the operands is of string type.

The following example shows how **+** operator performs operation on operands of different data types.

Example: + operator

var a = 5, b = "Hello ", c = "World!", d = 10;

a + b; // "5Hello "

b + c; // "Hello World!"

a + d; // 15

Comparison Operators

JavaScript language includes operators that compare two operands and return Boolean value true or false.

| **Operators** | **Description** |
| --- | --- |
| == | Compares the equality of two operands without considering type. |
| === | Compares equality of two operands with type. |
| != | Compares inequality of two operands. |
| > | Checks whether left side value is greater than right side value. If yes then returns true otherwise false. |
| < | Checks whether left operand is less than right operand. If yes then returns true otherwise false. |
| >= | Checks whether left operand is greater than or equal to right operand. If yes then returns true otherwise false. |
| <= | Checks whether left operand is less than or equal to right operand. If yes then returns true otherwise false. |

The following example demonstrates how comparison operators perform different tasks.

Example: Comparison Operators

var a = 5, b = 10, c = "5";

var x = a;

a == c; // returns true

a === c; // returns false

a == x; // returns true

a != b; // returns true

a > b; // returns false

a < b; // returns true

a >= b; // returns false

a <= b; // returns true

a >= c; // returns true

a <= c; // returns true

Logical Operators

Logical operators are used to combine two or more conditions. JavaScript includes following logical operators.

| **Operator** | **Description** |
| --- | --- |
| && | && is known as AND operator. It checks whether two operands are non-zero (0, false, undefined, null or "" are considered as zero), if yes then returns 1 otherwise 0. |
| || | || is known as OR operator. It checks whether any one of the two operands is non-zero (0, false, undefined, null or "" is considered as zero). |
| ! | ! is known as NOT operator. It reverses the boolean result of the operand (or condition) |

Example: Logical Operators

var a = 5, b = 10;

(a != b) && (a < b); // returns true

(a > b) || (a == b); // returns false

(a < b) || (a == b); // returns true

!(a < b); // returns false

!(a > b); // returns true

Assignment Operators

JavaScript includes assignment operators to assign values to variables with less key strokes.

| **Assignment operators** | **Description** |
| --- | --- |
| = | Assigns right operand value to left operand. |
| += | Sums up left and right operand values and assign the result to the left operand. |
| -= | Subtract right operand value from left operand value and assign the result to the left operand. |
| \*= | Multiply left and right operand values and assign the result to the left operand. |
| /= | Divide left operand value by right operand value and assign the result to the left operand. |
| %= | Get the modulus of left operand divide by right operand and assign resulted modulus to the left operand. |

Example: Assignment operators

var x = 5, y = 10, z = 15;

x = y; //x would be 10

x += 1; //x would be 6

x -= 1; //x would be 4

x \*= 5; //x would be 25

x /= 5; //x would be 1

x %= 2; //x would be 1

Ternary Operator

JavaScript includes special operator called ternary operator :? that assigns a value to a variable based on some condition. This is like short form of if-else condition.

Syntax:

<condition> ? <value1> : <value2>;

Ternary operator starts with conditional expression followed by ? operator. Second part ( after ? and before : operator) will be executed if condition turns out to be true. If condition becomes false then third part (after :) will be executed.

Example: Ternary operator

var a = 10, b = 5;

var c = a > b? a : b; // value of c would be 10

var d = a > b? b : a; // value of d would be 5

**Points to Remember :**

1. JavaScript includes operators that perform some operation on single or multiple operands (data value) and produce a result.
2. JavaScript includes various categories of operators: Arithmetic operators, Comparison operators, Logical operators, Assignment operators, Conditional operators.
3. Ternary operator ?: is a conditional operator.

**JavaScript Conditional Statements**

Conditional Statements

Very often when you write code, you want to perform different actions for different decisions.

You can use conditional statements in your code to do this.

In JavaScript we have the following conditional statements:

* Use ifto specify a block of code to be executed, if a specified condition is true
* Use elseto specify a block of code to be executed, if the same condition is false
* Use else ifto specify a new condition to test, if the first condition is false
* Use switchto specify many alternative blocks of code to be executed

The if Statement

Use the ifstatement to specify a block of JavaScript code to be executed if a condition is true.

Syntax

if (*condition*) {  
  //*block of code to be executed if the condition is true*}

Note that ifis in lowercase letters. Uppercase letters (If or IF) will generate a JavaScript error.

Example

Make a "Good day" greeting if the hour is less than 18:00:

if (hour < 18) {  
  greeting = "Good day";  
}

The result of greeting will be:

Good day

The else Statement

Use the elsestatement to specify a block of code to be executed if the condition is false.

if (*condition*) {  
  //*block of code to be executed if the condition is true*} else {  
  //*block of code to be executed if the condition is false*}

Example

If the hour is less than 18, create a "Good day" greeting, otherwise "Good evening":

if (hour < 18) {  
  greeting = "Good day";  
} else {  
  greeting = "Good evening";  
}

The result of greeting will be:

Good day

The else if Statement

Use the else ifstatement to specify a new condition if the first condition is false.

Syntax

if (*condition1*) {  
  //*block of code to be executed if condition1 is true*} else if (*condition2*) {  
  //*block of code to be executed if the condition1 is false and condition2 is true*  
} else {  
  //*block of code to be executed if the condition1 is false and condition2 is false*}

Example

If time is less than 10:00, create a "Good morning" greeting, if not, but time is less than 20:00, create a "Good day" greeting, otherwise a "Good evening":

if (time < 10) {  
  greeting = "Good morning";  
} else if (time < 20) {  
  greeting = "Good day";  
} else {  
  greeting = "Good evening";  
}

The result of greeting will be:

Good day

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.model = 500  car.weight = 850kg  car.color = white | car.start()  car.drive()  car.brake()  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**.

JavaScript Objects

You have already learned that JavaScript variables are containers for data values.

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

var 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:

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

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

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

Object Definition

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

Example

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

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

Example

var 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"];

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

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

The **this** Keyword

In a function definition, thisrefers to the "owner" of the function.

In the example above, thisis the **person object** that "owns" the fullNamefunction.

In other words, this.firstNamemeans the firstNameproperty of **this object**.

Read more about the thiskeyword at [JS this Keyword (Links to an external site.)](https://www.w3schools.com/js/js_this.asp).

Accessing Object Methods

You access an object method with the following syntax:

*objectName.methodName()*

Example

name = person.fullName();

If you access a method **without** the () parentheses, it will return the **function definition**:

Example

name = person.fullName;

Do Not Declare Strings, Numbers, and Booleans as Objects!

When a JavaScript variable is declared with the keyword "new", the variable is created as an object:

var x = new String();        // Declares x as a String object  
var y = new Number();        // Declares y as a Number object  
var z = new Boolean();       // Declares z as a Boolean object

String, Number, and Boolean. They complicate your code and slow down execution speed.

HTML Events

An HTML event can be something the browser does, or something a user does.

Here are some examples of HTML events:

* An HTML web page has finished loading
* An HTML input field was changed
* An HTML button was clicked

Often, when events happen, you may want to do something.

JavaScript lets you execute code when events are detected.

HTML allows event handler attributes, **with JavaScript code**, to be added to HTML elements.

With single quotes:

<*element* *event*=**'*some JavaScript*'**>

With double quotes:

<*element* *event*=**"*some JavaScript*"**>

In the following example, an onclickattribute (with code), is added to a <button>element:

Example

<button onclick="document.getElementById('demo').innerHTML = Date()">The time is?</button>

In the example above, the JavaScript code changes the content of the element with id="demo".

In the next example, the code changes the content of its own element (using **this**.innerHTML):

Example

<button onclick="this.innerHTML = Date()">The time is?</button>

JavaScript code is often several lines long. It is more common to see event attributes calling functions:

Example

<button onclick="displayDate()">The time is?</button>

Common HTML Events

Here is a list of some common HTML events:

|  |  |
| --- | --- |
| **Event** | **Description** |
| onchange | An HTML element has been changed |
| onclick | The user clicks an HTML element |
| onmouseover | The user moves the mouse over an HTML element |
| onmouseout | The user moves the mouse away from an HTML element |
| onkeydown | The user pushes a keyboard key |
| onload | The browser has finished loading the page |

The list is much longer, You can retrieve other events in [W3Schools JavaScript Reference HTML DOM Events. (Links to an external site.)](https://www.w3schools.com/jsref/dom_obj_event.asp)

What can JavaScript Do?

Event handlers can be used to handle, and verify, user input, user actions, and browser actions:

* Things that should be done every time a page loads
* Things that should be done when the page is closed
* Action that should be performed when a user clicks a button
* Content that should be verified when a user inputs data
* And more ...

Many different methods can be used to let JavaScript work with events:

* HTML event attributes can execute JavaScript code directly
* HTML event attributes can call JavaScript functions
* You can assign your own event handler functions to HTML elements
* You can prevent events from being sent or being handled
* And more ...

JavaScript Forms

JavaScript Form Validation

HTML form validation can be done by JavaScript.

If a form field (fname) is empty, this function alerts a message, and returns false, to prevent the form from being submitted:

JavaScript Example

function validateForm() {  
  var x = document.forms["myForm"]["fname"].value;  
  if (x == "") {  
    alert("Name must be filled out");  
    return false;  
  }  
}

The function can be called when the form is submitted:

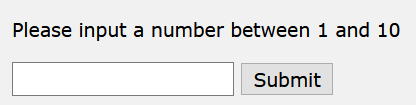
HTML Form Example

<form name="myForm" action="/action\_page.php" **onsubmit="return validateForm()"** method="post">  
Name: <input type="text" name="fname">  
<input type="submit" value="Submit">  
</form>

**Note:** you can try the examples in [Online HTML Editor (Links to an external site.)](https://www.onlinegdb.com/online_html_compiler), [Online HTML Compiler (Links to an external site.)](https://www.tutorialspoint.com/online_html_editor.php), or you may use Notepad or Notepad++.

JavaScript Can Validate Numeric Input

JavaScript is often used to validate numeric input:



Automatic HTML Form Validation

HTML form validation can be performed automatically by the browser:

If a form field (fname) is empty, the requiredattribute prevents this form from being submitted:

HTML Form Example

<form action="/action\_page.php" method="post">  
  <input type="text" name="fname" **required**>  
  <input type="submit" value="Submit">  
</form>

Note: Automatic HTML form validation does not work in Internet Explorer 9 or earlier.

Data Validation

Data validation is the process of ensuring that user input is clean, correct, and useful.

Typical validation tasks are:

* has the user filled in all required fields?
* has the user entered a valid date?
* has the user entered text in a numeric field?

Most often, the purpose of data validation is to ensure correct user input.

Validation can be defined by many different methods, and deployed in many different ways.

**Server side validation** is performed by a web server, after input has been sent to the server.

**Client side validation** is performed by a web browser, before input is sent to a web server.

HTML Constraint Validation

HTML5 introduced a new HTML validation concept called **constraint validation**.

HTML constraint validation is based on:

* Constraint validation **HTML** **Input Attributes**
* Constraint validation **CSS Pseudo Selectors**
* Constraint validation **DOM Properties and Methods**

Constraint Validation HTML Input Attributes

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| disabled | Specifies that the input element should be disabled |
| max | Specifies the maximum value of an input element |
| min | Specifies the minimum value of an input element |
| pattern | Specifies the value pattern of an input element |
| required | Specifies that the input field requires an element |
| type | Specifies the type of an input element |

For a full list, go to [HTML Input Attributes (Links to an external site.)](https://www.w3schools.com/html/html_form_attributes.asp).

Constraint Validation CSS Pseudo Selectors

|  |  |
| --- | --- |
| **Selector** | **Description** |
| :disabled | Selects input elements with the "disabled" attribute specified |
| :invalid | Selects input elements with invalid values |
| :optional | Selects input elements with no "required" attribute specified |
| :required | Selects input elements with the "required" attribute specified |
| :valid | Selects input elements with valid values |

For a full list, go to [CSS Pseudo Classes (Links to an external site.)](https://www.w3schools.com/css/css_pseudo_classes.asp).