JavaScript Tutorial

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

Commonly Asked Questions

- How do I get JavaScript?
- Where can I download JavaScript?
- Is JavaScript Free?

You don't have to get or download JavaScript.

JavaScript is already running in your browser on your computer, on your tablet, and on your smart-phone.

JavaScript is free to use for everyone.

JavaScript Where To?

1) The <script> Tag

In HTML, JavaScript code is inserted between `<script>` and `</script>` tags.

Example

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

Old JavaScript examples may use a type attribute:
<script type="text/javascript">.
The type attribute is not required. JavaScript is the default scripting language in HTML.

2) 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.

3) JavaScript in <head>

In this example, a JavaScript `function` is placed in the `<head>` section of an HTML page.

The function is invoked (called) when a button is clicked:

Example

```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>
</head>
<body>

<h2>Demo JavaScript in Head</h2>

<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try it</button>

</body>
</html>
```

4) 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:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>Demo JavaScript in Body</h2>

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

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

<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>

</body>
</html>
```

Placing scripts at the bottom of the <body> element improves the display speed, because script interpretation slows down the display.

5) 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 scripts cannot contain `<script>` tags.

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>
```

JavaScript Syntax

JavaScript syntax is the set of rules, how JavaScript programs are constructed:

```
// How to create variables:
var x;
let y;

// How to use variables:
x = 5;
y = 6;
let z = x + y;
```

✓ JavaScript Values

The JavaScript syntax defines two types of values:

- Fixed values
- 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:

```
10.50
1001
```

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

```
"John Doe"
'John Doe'
```

✓ 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, x is defined as a variable. Then, x is assigned (given) the value 6:

```
let x;
x = 6;
```

✓ JavaScript Operators

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

```
(5 + 6) * 10
```

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

```
let x, y;
x = 5;
y = 6;
```

✓ JavaScript Expressions

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

The computation is called an evaluation.

For example, 5 * 10 evaluates to 50:

```
5 * 10
```

Expressions can also contain variable values:

```
x * 10
```

The values can be of various types, such as numbers and strings.

For example, "John" + " " + "Doe", evaluates to "John Doe":

```
"John" + " " + "Doe"
```

✓ JavaScript Keywords

JavaScript **keywords** are used to identify actions to be performed.

The `let` keyword tells the browser to create variables:

```
let x, y;
x = 5 + 6;
y = x * 10;
```

The `var` keyword also tells the browser to create variables:

```
var x, y;
x = 5 + 6;
y = x * 10;
```

In these examples, using `var` or `let` will produce the same result.

✓ JavaScript Comments

Not all JavaScript statements are "executed".

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

Comments are ignored, and will not be executed:

```
      let x = 5;   // I will be executed

      // x = 6;   I will NOT be executed
```

✓ JavaScript Identifiers / Names

Identifiers are JavaScript names.

Identifiers are used to name variables and keywords, and functions.

The rules for legal names are the same in most programming languages.

A JavaScript name must begin with:

- A letter (A-Z or a-z)
- A dollar sign ($)
- Or an underscore (_)

Subsequent characters may be letters, digits, underscores, or dollar signs.

Note

Numbers are **not allowed** as the first character in names.

This way JavaScript can easily distinguish identifiers from numbers.

✓ JavaScript is Case Sensitive

All JavaScript identifiers are **case sensitive**.

The variables `lastName` and `lastname`, are two different variables:

```
let lastname, lastName;
lastName = "Doe";
lastname = "Peterson";
```

JavaScript does not interpret **LET** or **Let** as the keyword **let**.

✓ JavaScript and Camel Case

Historically, programmers have used different ways of joining multiple words into one variable name:

- **Hyphens:**

  first-name, last-name, master-card, inter-city.

  Hyphens are **not allowed** in JavaScript. They are reserved for subtractions.

- **Underscore:**

  first_name, last_name, master_card, inter_city.

- **Upper Camel Case (Pascal Case):**

  FirstName, LastName, MasterCard, InterCity.

- **Lower Camel Case:**

  JavaScript programmers tend to use camel case that starts with a lowercase letter:

  firstName, lastName, masterCard, interCity.

✓ JavaScript Character Set

JavaScript uses the **Unicode** character set.

Unicode covers (almost) all the characters, punctuations, and symbols in the world.

JavaScript Variables...Extra Details

✓ 4 Ways to Declare a JavaScript Variable:

- Using `var`
- Using `let`
- Using `const`
- Using nothing

✓ What are Variables?

Variables are containers for storing data (storing data values).

In this example, `x`, `y`, and `z`, are variables, declared with the `var` keyword:

Example

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

In this example, `x`, `y`, and `z`, are variables, declared with the `let` keyword:

Example

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

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

Example

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

From all the examples above, you can guess:

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

✓ When to Use JavaScript var?

Always declare JavaScript variables with `var`, `let`, or `const`.

> The `var` keyword is used in all JavaScript code from 1995 to 2015.
>
> The `let` and `const` keywords were added to JavaScript in 2015.
>
> If you want your code to run in older browsers, you must use `var`.

✓ When to Use JavaScript const?

> If you want a general rule: always declare variables with const.
>
> If you think the value of the variable can change, use let.

In this example, `price1`, `price2`, and `total`, are variables:

Example

```
const price1 = 5;
const price2 = 6;
let total = price1 + price2;
```

The two variables `price1` and `price2` are declared with the `const` keyword.

These are constant values and cannot be changed.

The variable `total` is declared with the `let` keyword.

This is a value that can be changed.

Just Like Algebra

Just like in algebra, variables hold values:

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

Just like in algebra, variables are used in expressions:

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

From the example above, you can guess that the total is calculated to be 11.

JavaScript Identifiers

All JavaScript **variables** must be **identified** with **unique names**.

These unique names are called **identifiers**.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter.
- Names can also begin with $ and _ (but we will not use it in this tutorial).
- Names are case sensitive (y and Y are different variables).
- Reserved words (like JavaScript keywords) cannot be used as names.

Note

JavaScript identifiers are case-sensitive.

The Assignment Operator

In JavaScript, the equal sign (`=`) is an "assignment" operator, not an "equal to" operator.

This is different from algebra. The following does not make sense in algebra:

```
x = x + 5
```

In JavaScript, however, it makes perfect sense: it assigns the value of x + 5 to x.

(It calculates the value of x + 5 and puts the result into x. The value of x is incremented by 5.)

Note

The "equal to" operator is written like `==` in JavaScript.

JavaScript Data Types

JavaScript variables can hold numbers like 100 and text values like "John Doe".

In programming, text values are called text strings.

JavaScript can handle many types of data, but for now, just think of numbers and strings.

Strings are written inside double or single quotes. Numbers are written without quotes.

If you put a number in quotes, it will be treated as a text string.

Example

```
const pi = 3.14;
let person = "John Doe";
let answer = 'Yes I am!';
```

Declaring a JavaScript Variable

Creating a variable in JavaScript is called "declaring" a variable.

You declare a JavaScript variable with the `var` or the `let` keyword:

```
var carName;
```

or:

```
let carName;
```

After the declaration, the variable has no value (technically it is `undefined`).

To **assign** a value to the variable, use the equal sign:

```
carName = "Volvo";
```

You can also assign a value to the variable when you declare it:

```
let carName = "Volvo";
```

In the example below, we create a variable called `carName` and assign the value "Volvo" to it.

Then we "output" the value inside an HTML paragraph with id="demo":

Example

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

<script>
let carName = "Volvo";
document.getElementById("demo").innerHTML = carName;
</script>
```

One Statement, Many Variables

You can declare many variables in one statement.

Start the statement with `let` and separate the variables by **comma**:

Example

```
let person = "John Doe", carName = "Volvo", price = 200;
```

A declaration can span multiple lines:

Example

```
let person = "John Doe",
carName = "Volvo",
price = 200;
```

Value = undefined

In computer programs, variables are often declared without a value. The value can be something that has to be calculated, or something that will be provided later, like user input.

A variable declared without a value will have the value `undefined`.

The variable carName will have the value `undefined` after the execution of this statement:

Example

```
let carName;
```

Re-Declaring JavaScript Variables

If you re-declare a JavaScript variable declared with `var`, it will not lose its value.

The variable `carName` will still have the value "Volvo" after the execution of these statements:

Example

```
var carName = "Volvo";
var carName;
```

JavaScript Arithmetic

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

Example

```
let x = 5 + 2 + 3;
```

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

Example

```
let x = "John" + " " + "Doe";
```

Also try this:

Example

```
let x = "5" + 2 + 3;
```

Note

If you put a number in quotes, the rest of the numbers will be treated as strings, and concatenated.

Now try this:

Example

```
let x = 2 + 3 + "5";
```

JavaScript Dollar Sign $

Since JavaScript treats a dollar sign as a letter, identifiers containing $ are valid variable names:

Example

```
let $ = "Hello World";
let $$$ = 2;
let $myMoney = 5;
```

Using the dollar sign is not very common in JavaScript, but professional programmers often use it as an alias for the main function in a JavaScript library.

In the JavaScript library jQuery, for instance, the main function `$` is used to select HTML elements. In jQuery `$("p");` means "select all p elements".

JavaScript Underscore (_)

Since JavaScript treats underscore as a letter, identifiers containing _ are valid variable names:

Example

```
let _lastName = "Johnson";
let _x = 2;
let _100 = 5;
```

Using the underscore is not very common in JavaScript, but a convention among professional programmers is to use it as an alias for "private (hidden)" variables.

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.";
```

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.";
```

Using Comments to Prevent Execution

Using comments to prevent execution of code is suitable for code testing.

Adding `//` in front of a code line changes the code lines from an executable line to a comment.

This example uses // to prevent execution of one of the code lines:

Example

```
//document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
```

JavaScript HTML DOM

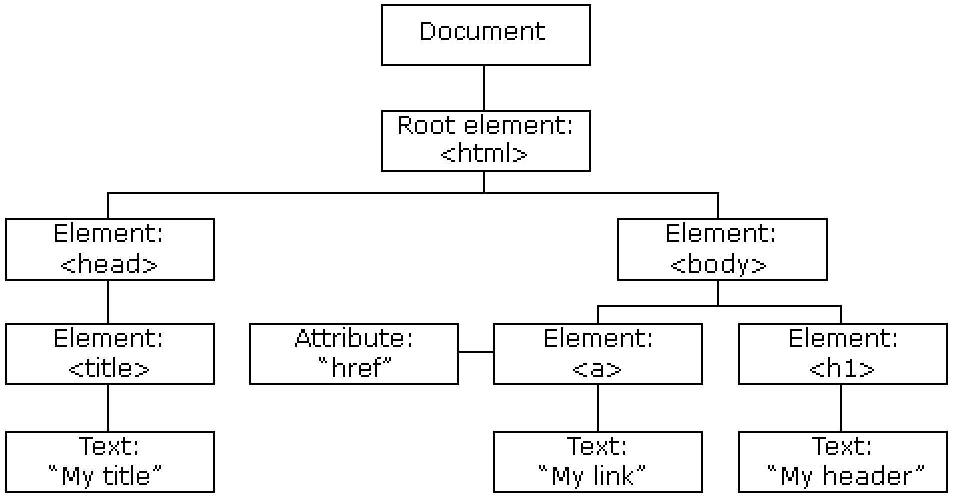
With the HTML DOM, JavaScript can access and change all the elements of an HTML document.

The HTML DOM (Document Object Model)

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

The **HTML DOM** model is constructed as a tree of **Objects**:

The HTML DOM Tree of Objects



With the object model, JavaScript gets all the power it needs to create dynamic HTML:

- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can remove existing HTML elements and attributes

- JavaScript can add new HTML elements and attributes
- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML events in the page

What You Will Learn

In the next chapters of this tutorial you will learn:

- How to change the content of HTML elements
- How to change the style (CSS) of HTML elements
- How to react to HTML DOM events
- How to add and delete HTML elements

What is the HTML DOM?

The HTML DOM is a standard **object** model and **programming interface** for HTML. It defines:

- The HTML elements as **objects**
- The **properties** of all HTML elements
- The **methods** to access all HTML elements
- The **events** for all HTML elements

In other words: **The HTML DOM is a standard for how to get, change, add, or delete HTML elements.**

JavaScript - HTML DOM Methods

HTML DOM methods are **actions** you can perform (on HTML Elements).

HTML DOM properties are **values** (of HTML Elements) that you can set or change.

The DOM Programming Interface

The HTML DOM can be accessed with JavaScript (and with other programming languages).

In the DOM, all HTML elements are defined as **objects**.

The programming interface is the properties and methods of each object.

A **property** is a value that you can get or set (like changing the content of an HTML element).

A **method** is an action you can do (like add or deleting an HTML element).

Example

The following example changes the content (the `innerHTML`) of the `<p>` element with `id="demo"`:

Example

```
<html>
<body>
<p id="demo"></p>


<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>

</body>
</html>
```

In the example above, `getElementById` is a **method**, while `innerHTML` is a **property**.

The getElementById Method

The most common way to access an HTML element is to use the `id` of the element.

In the example above the `getElementById` method used `id="demo"` to find the element.

The innerHTML Property

The easiest way to get the content of an element is by using the `innerHTML` property.

The `innerHTML` property is useful for getting or replacing the content of HTML elements.

The `innerHTML` property can be used to get or change any HTML element, including `<html>` and `<body>`.

JavaScript HTML DOM Document

The HTML DOM document object is the owner of all other objects in your web page.

The HTML DOM Document Object

The document object represents your web page.

If you want to access any element in an HTML page, you always start with accessing the document object.

Below are some examples of how you can use the document object to access and manipulate HTML.

Finding HTML Elements

Method	Description
document.getElementById(id)	Find an element by element id
document.getElementsByTagName(name)	Find elements by tag name
document.getElementsByClassName(name)	Find elements by class name

Changing HTML Elements

Property	Description
element.innerHTML = new html content	Change the inner HTML of an element
element.attribute = new value	Change the attribute value of an HTML element
element.style.property = new style	Change the style of an HTML element
Method	**Description**
element.setAttribute(attribute, value)	Change the attribute value of an HTML element

The HTML DOM allows JavaScript to change the content of HTML elements.

Changing HTML Content

The easiest way to modify the content of an HTML element is by using the `innerHTML` property.

To change the content of an HTML element, use this syntax:

```
document.getElementById(id).innerHTML = new HTML
```

This example changes the content of a `<p>` element:

Example

```
<html>
<body>
<p id="p1">Hello World!</p>
<script>
document.getElementById("p1").innerHTML = "New text!";
</script>
</body>
</html>
```

Example explained:

- The HTML document above contains a `<p>` element with `id="p1"`
- We use the HTML DOM to get the element with `id="p1"`
- A JavaScript changes the content (`innerHTML`) of that element to "New text!"

This example changes the content of an `<h1>` element:

Example

```
<!DOCTYPE html>
<html>
<body>
<h1 id="id01">Old Heading</h1>
<script>
```

```
const element = document.getElementById("id01");
element.innerHTML = "New Heading";
</script>
</body>
</html>
```

Example explained:

- The HTML document above contains an `<h1>` element with `id="id01"`
- We use the HTML DOM to get the element with `id="id01"`
- A JavaScript changes the content (`innerHTML`) of that element to "New Heading"

Changing the Value of an Attribute

To change the value of an HTML attribute, use this syntax:

```
document.getElementById(id).attribute = new value
```

This example changes the value of the src attribute of an `<img>` element:

Example

```
<!DOCTYPE html>
<html>
<body>
<img id="myImage" src="smiley.gif">

<script>
document.getElementById("myImage").src = "landscape.jpg";
</script>

</body>
</html>
```

Example explained:

- The HTML document above contains an `<img>` element with `id="myImage"`
- We use the HTML DOM to get the element with `id="myImage"`
- A JavaScript changes the `src` attribute of that element from "smiley.gif" to "landscape.jpg"

Dynamic HTML content

JavaScript can create dynamic HTML content:

Date : Sat May 20 2023 17:49:38 GMT+0300 (GMT+03:00)

Example

```
<!DOCTYPE html>
<html>
<body>

<script>
document.getElementById("demo").innerHTML = "Date : " + Date(); </script>

</body>
</html
```

document.write()

In JavaScript, `document.write()` can be used to write directly to the HTML output stream:

Example

```
<!DOCTYPE html>
<html>
<body>
<p>Bla bla bla</p>
<script>
document.write(Date());
</script>
<p>Bla bla bla</p>
</body>
</html>
```

Never use `document.write()` after the document is loaded. It will overwrite the document.

Adding and Deleting Elements

Method	Description
document.createElement(element)	Create an HTML element
document.removeChild(element)	Remove an HTML element
document.appendChild(element)	Add an HTML element
document.replaceChild(new, old)	Replace an HTML element
document.write(text)	Write into the HTML output stream

Adding Events Handlers

Method	Description
document.getElementById(id).onclick = function(){code}	Adding event handler code to an onclick event

The following HTML objects (and object collections) are also accessible:

- document.anchors
- document.body
- document.documentElement
- document.embeds
- document.forms
- document.head
- document.images
- document.links
- document.scripts
- document.title

JavaScript Introduction

This page contains some examples of what JavaScript can do.

✓ 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";
```

JavaScript HTML DOM Elements

Finding HTML Elements

Often, with JavaScript, you want to manipulate HTML elements.

To do so, you have to find the elements first. There are several ways to do this:

- Finding HTML elements by id
- Finding HTML elements by tag name
- Finding HTML elements by class name
- Finding HTML elements by CSS selectors
- Finding HTML elements by HTML object collections

Finding HTML Element by Id

The easiest way to find an HTML element in the DOM, is by using the element id.

This example finds the element with `id="intro"`:

Example

```
const element = document.getElementById("intro");
```

If the element is found, the method will return the element as an object (in element).

If the element is not found, element will contain `null`.

Finding HTML Elements by Tag Name

This example finds all `<p>` elements:

Example

```
const element = document.getElementsByTagName("p");
```

This example finds the element with `id="main"`, and then finds all `<p>` elements inside `"main"`:

Example

```
const x = document.getElementById("main");
const y = x.getElementsByTagName("p");
```

Finding HTML Elements by Class Name

If you want to find all HTML elements with the same class name, use `getElementsByClassName()`.

This example returns a list of all elements with `class="intro"`.

Example

```
const x = document.getElementsByClassName("intro");
```

Finding HTML Elements by CSS Selectors

If you want to find all HTML elements that match a specified CSS selector (id, class names, types, attributes, values of attributes, etc), use the `querySelectorAll()` method.

This example returns a list of all `<p>` elements with `class="intro"`.

Example

```
const x = document.querySelectorAll("p.intro");
```

Finding HTML Elements by HTML Object Collections

This example finds the form element with `id="frm1"`, in the forms collection, and displays all element values:

Example

```
const x = document.forms["frm1"];
let text = "";
for (let i = 0; i < x.length; i++) {
  text += x.elements[i].value + "<br>";
}
document.getElementById("demo").innerHTML = text;
```

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;
}
```

Why Functions?

With functions you can reuse code

You can write code that can be used many times.

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

The () Operator

The () operator invokes (calls) the function:

Example

Convert Fahrenheit to Celsius:

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}

let value = toCelsius(77);
```

Accessing a function with incorrect parameters can return an incorrect answer:

Example

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}

let value = toCelsius();
```

Accessing a function without () returns the function and not the function result:

Example

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}

let value = toCelsius;
```

Note

As you see from the examples above, `toCelsius` refers to the function object, and `toCelsius()` refers to the function result.

Functions Used as Variable Values

Functions can be used the same way as you use variables, in all types of formulas, assignments, and calculations.

Example

Instead of using a variable to store the return value of a function:

```
let x = toCelsius(77);
let text = "The temperature is " + x + " Celsius";
```

You can use the function directly, as a variable value:

```
let text = "The temperature is " + toCelsius(77) + " Celsius";
```

Local Variables

Variables declared within a JavaScript function, become **LOCAL** to the function.

Local variables can only be accessed from within the function.

Example

```
// code here can NOT use carName

function myFunction() {
  let carName = "Volvo";
  // code here CAN use carName
}

// code here can NOT use carName
```

Since local variables are only recognized inside their functions, variables with the same name can be used in different functions.

Local variables are created when a function starts, and deleted when the function is completed.

JavaScript Events

HTML events are **"things"** that happen to HTML elements.

When JavaScript is used in HTML pages, JavaScript can **"react"** on these events.

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 `onclick` attribute (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

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

JavaScript Event Handlers

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 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:

Example

```
<!DOCTYPE html>
<html>
<body>
         <h1>My First Web Page</h1>
         <p>My First Paragraph</p>
         <p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>


</body>
</html>
```

Changing the innerHTML property of an HTML element is a common way to display data in HTML.

Using document.write()

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

Example

```
<!DOCTYPE html>
<html>
<body>
       <h1>My First Web Page</h1>
       <p>My first paragraph.</p>
<script>
document.write(5 + 6);
</script>

</body>
</html>
```

Using document.write() after an HTML document is loaded, will **delete all existing HTML**:

Example

```
<!DOCTYPE html>
<html>
<body>
          <h1>My First Web Page</h1>
          <p>My first paragraph.</p>

<button type="button" onclick="document.write(5 + 6)">Try it</button>


</body>
</html>
```

The document.write() method should only be used for testing.

Using window.alert()

You can use an alert box to display data:

Example

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
window.alert(5 + 6);
</script>

</body>
</html>
```

You can skip the `window` keyword.

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

Example

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
alert(5 + 6);
</script>

</body>
</html>
```

Using console.log()

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

You will learn more about debugging in a later chapter.

Example

```
<!DOCTYPE html>
<html>
<body>

<script>
console.log(5 + 6);
</script>

</body>
</html>
```

JavaScript Print

JavaScript does not have any print object or print methods.

You cannot access output devices from JavaScript.

The only exception is that you can call the `window.print()` method in the browser to print the content of the current window.

Example

```
<!DOCTYPE html>
<html>
<body>

<button onclick="window.print()">Print this page</button>

</body>
</html>
```