Javascript

JavaScript Introduction

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. Free to use for everyone.

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.

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.

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

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>
```

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

External References

External scripts can be referenced with a full URL or with a path relative to the current web page.

This example uses a full URL to link to a script:

Example

```
<script src="https://www.Alone3000.tec.com/js/myScript1.js"></script>
```

This example uses a script located in a specified folder on the current web site:

Example

```
<script src="/js/myScript1.js"></script>
```

This example links to a script located in the same folder as the current page:

Example

```
<script src="myScript1.js"></script>
```

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:

```
<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>
```

Note: 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()`:

```
<script>
document.write(5 + 6);
</script>
```

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

```
<script>
document.write(5 + 6);
</script>
```

Using window.alert()

You can use an alert box to display data:

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

Tip! You can skip the `window` keyword.

Using console.log()

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

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


JavaScript Programs

A **computer program** is a list of "instructions" to be "executed" by a computer.

In a programming language, these programming instructions are called **statements**.

A **JavaScript program** is a list of programming **statements**.

JavaScript Statements

JavaScript statements are composed of:

Values, Operators, Expressions, Keywords, and Comments.

The statements are executed, one by one, in the same order as they are written.

JavaScript programs (and JavaScript statements) are often called JavaScript code.

Semicolons ;

Semicolons separate JavaScript statements.

When separated by semicolons, multiple statements on one line are allowed:

JavaScript White Space

JavaScript ignores multiple spaces. You can add white space to your script to make it more readable.

JavaScript Line Length and Line Breaks

For best readability, programmers often like to avoid code lines longer than 80 characters.

If a JavaScript statement does not fit on one line, the best place to break it is after an operator:

Example

```
document.getElementById("demo").innerHTML=
"Hello Dolly!";
```

JavaScript Code Blocks

JavaScript statements can be grouped together in code blocks, inside curly brackets {...}.

The purpose of code blocks is to define statements to be executed together.

JavaScript Variables

In a programming language, **variables** are used to **store** data values.

JavaScript uses the `var` keyword to **declare** variables.

An **equal sign** is used to **assign values** to variables.

Redeclaring

Redeclaring or reassigning an existing `var` or `let` variable to `const`, in the same scope, or in the same block, is not allowed:

Example

```
var x = 2;       // Allowed
const x = 2;     // Not allowed
{
  let x = 2;    // Allowed
  const x = 2;  // Not allowed
}
```

Redeclaring or reassigning an existing `const` variable, in the same scope, or in the same block, is not allowed:

Example

```
const x = 2;     // Allowed
const x = 3;     // Not allowed
```

```
x = 3;          // Not allowed
var x = 3;      // Not allowed
let x = 3;      // Not allowed

{
 const x = 2;  // Allowed
 const x = 3;  // Not allowed
 x = 3;        // Not allowed
 var x = 3;    // Not allowed
 let x = 3;    // Not allowed
}
```

Redeclaring a variable with `const`, in another scope, or in another block, is allowed:

Example

```
const x = 2;      // Allowed

{
 const x = 3;  // Allowed
}

{
 const x = 4;  // Allowed
}
```


JavaScript Keywords

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

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

```
var x, y;
```

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:

JavaScript Identifiers

Identifiers are names.

In JavaScript, the first character must be a letter, or an underscore (_), or a dollar sign ($).

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

Reserved words (like JavaScript keywords) cannot be used as names

JavaScript is Case Sensitive

All JavaScript identifiers are **case sensitive**.

JavaScript does not interpret **VAR** or **Var** as the keyword **var**.

JavaScript Data Types

JavaScript variables can hold many **data types**: numbers, strings, objects and more:

```
var length = 16;                       // Number
var lastName = "Johnson";              // String
var x = {firstName:"John", lastName:"Doe"};   // Object
```

Primitive Data Types

A primitive data value is a single simple data value with no additional properties and methods.

The `typeof` operator can return one of these primitive types:

- `string`
- `number`
- `boolean`
- `undefined`

Complex Data Types

The `typeof` operator can return one of two complex types:

- `function`
- `object`

The `typeof` operator returns "object" for objects, arrays, and null.

The `typeof` operator does not return "object" for functions.

The `typeof` operator returns "`object`" for arrays because in JavaScript arrays are objects.

JavaScript Operators

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic on numbers:

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

JavaScript Assignment Operators

Assignment operators assign values to JavaScript variables.

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
<<=	x <<= y	x = x << y
>>=	x >>= y	x = x >> y
>>>=	x >>>= y	x = x >>> y
&=	x &= y	x = x & y
^=	x ^= y	x = x ^ y
|=	x |= y	x = x | y
**=	x **= y	x = x ** y

The **= operator is an experimental part of the ECMAScript 2016 proposal (ES7). It is not stable across browsers. Do not use it.

JavaScript Comparison Operators

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

JavaScript Logical Operators

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

JavaScript Type Operators

Operator	Description
Typeof	Returns the type of a variable
Instanceof	Returns true if an object is an instance of an object type

JavaScript Bitwise Operators

Bit operators work on 32 bits numbers.

Any numeric operand in the operation is converted into a 32 bit number. The result is converted back to a JavaScript number.

Operator	Description	Example	Same as	Result	Decimal
&	AND	5 & 1	0101 & 0001	0001	1
|	OR	5 | 1	0101 | 0001	0101	5
~	NOT	~ 5	~0101	1010	10
^	XOR	5 ^ 1	0101 ^ 0001	0100	4
<<	Zero fill left shift	5 << 1	0101 << 1	1010	10
>>	Signed right shift	5 >> 1	0101 >> 1	0010	2
>>>	Zero fill right shift	5 >>> 1	0101 >>> 1	0010	2

Operators and Operands

The numbers (in an arithmetic operation) are called **operands**.

The operation (to be performed between the two operands) is defined by an **operator**.

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 myFunction(p1, p2) {
  return p1 * p2;   // The function returns the product of p1 and p2
}
```

The this Keyword

In a function definition, `this` refers to the "owner" of the function.

In other words, `this.firstName` means the `firstName` property of **this object**.

JavaScript Strings

JavaScript strings are used for storing and manipulating text.

String Length

To find the length of a string, use the built-in `length` property:

Example

```
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
var sln = txt.length;
```


Six other escape sequences are valid in JavaScript:

Code	Result
\b	Backspace
\n	New Line
\t	Horizontal Tabulator

String Methods and Properties

Primitive values, like "John Doe", cannot have properties or methods (because they are not objects).

But with JavaScript, methods and properties are also available to primitive values, because JavaScript treats primitive values as objects when executing methods and properties.

String Length

The `length` property returns the length of a string:

Example

```
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
var sln = txt.length;
```

Finding a String in a String

The `indexOf()` method returns the index of (the position of) the `first` occurrence of a specified text in a string:

Example

```
var str = "Please locate where 'locate' occurs!";
var pos = str.indexOf("locate");
```

Converting to Upper and Lower Case

A string is converted to upper case with `toUpperCase()`:

Example

```
var text1 = "Hello World!";     // String
var text2 = text1.toUpperCase(); // text2 is text1 converted to upper
```

A string is converted to lower case with `toLowerCase()`:

Example

```
var text1 = "Hello World!";     // String
var text2 = text1.toLowerCase(); // text2 is text1 converted to lower
```


The concat() Method

`concat()` joins two or more strings:

Example

```
var text1 = "Hello";
var text2 = "World";
var text3 = text1.concat(" ", text2);
```

The `concat()` method can be used instead of the plus operator. These two lines do the same:

Example

```
var text = "Hello" + " " + "World!";
var text = "Hello".concat(" ", "World!");
```

Note: All string methods return a new string. They don't modify the original string.

Strings are immutable: Strings cannot be changed, only replaced.

JavaScript Arrays

What is an Array?

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

Creating an Array

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

Syntax:

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

Example

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

Using the JavaScript Keyword new

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

Example

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


Access the Elements of an Array

You access an array element by referring to the **index number**.

Example

```
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars[0];
```

Changing an Array Element

Example

```
var cars = ["Saab", "Volvo", "BMW"];
cars[0] = "Opel";
document.getElementById("demo").innerHTML = cars[0];
```

Access the Full Array

With JavaScript, the full array can be accessed by referring to the array name:

Example

```
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
```

The length Property

The `length` property of an array returns the length of an array (the number of array elements).

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.length;  // the length of fruits is 4
```

Note: The `length` property is always one more than the highest array index.

Looping Array Elements

The safest way to loop through an array, is using a `for` loop:

Example

```
var fruits, text, fLen, i;
fruits = ["Banana", "Orange", "Apple", "Mango"];
fLen = fruits.length;

text = "<ul>";
for (i = 0; i < fLen; i++) {
  text += "<li>" + fruits[i] + "</li>";
}
text += "</ul>";
```


You can also use the `Array.forEach()` function:

Example

```
var fruits, text;
fruits = ["Banana", "Orange", "Apple", "Mango"];

text = "<ul>";
fruits.forEach(myFunction);
text += "</ul>";

function myFunction(value) {
  text += "<li>" + value + "</li>";
}
```

Adding Array Elements

The easiest way to add a new element to an array is using the `push()` method:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Lemon");  // adds a new element (Lemon) to fruits
```

WARNING !

Adding elements with high indexes can create undefined "holes" in an array:

Converting Variables to Numbers

There are 3 JavaScript methods that can be used to convert variables to numbers:

- The `Number()` method
- The `parseInt()` method
- The `parseFloat()` method

These methods are not **number** methods, but **global** JavaScript methods.

JavaScript Math Object

The JavaScript Math object allows you to perform mathematical tasks on numbers.

Example

```
Math.PI;        // returns 3.141592653589793
```

Math.round()

`Math.round(x)` returns the value of x rounded to its nearest integer:

Example

```
Math.round(4.7);   // returns 5
Math.round(4.4);   // returns 4
```

Math.pow()

`Math.pow(x, y)` returns the value of x to the power of y:

Example

```
Math.pow(8, 2);    // returns 64
```

Math.sqrt()

`Math.sqrt(x)` returns the square root of x:

Example

```
Math.sqrt(64);    // returns 8
```

Math.abs()

`Math.abs(x)` returns the absolute (positive) value of x:

Example

```
Math.abs(-4.7);    // returns 4.7
```

Math.ceil()

`Math.ceil(x)` returns the value of x rounded **up** to its nearest integer:

Example

```
Math.ceil(4.4);    // returns 5
```

Math.floor()

`Math.floor(x)` returns the value of x rounded **down** to its nearest integer:

Example

```
Math.floor(4.7);    // returns 4
```

JavaScript Math Object

The JavaScript Math object allows you to perform mathematical tasks on numbers.

Example

```
Math.PI;        // returns 3.141592653589793
```

Math.round()

`Math.round(x)` returns the value of x rounded to its nearest integer:

Example

```
Math.round(4.7);   // returns 5
Math.round(4.4);   // returns 4
```

Math.pow()

`Math.pow(x, y)` returns the value of x to the power of y:

Example

```
Math.pow(8, 2);    // returns 64
```

Math.sqrt()

`Math.sqrt(x)` returns the square root of x:

Example

```
Math.sqrt(64);    // returns 8
```

Math.abs()

`Math.abs(x)` returns the absolute (positive) value of x:

Example

```
Math.abs(-4.7);    // returns 4.7
```

Math.ceil()

`Math.ceil(x)` returns the value of x rounded **up** to its nearest integer:

Example

```
Math.ceil(4.4);    // returns 5
```

Math.floor()

`Math.floor(x)` returns the value of x rounded **down** to its nearest integer:

Example

```
Math.floor(4.7);  // returns 4
```

JavaScript if else and else if

Conditional Statements

In JavaScript we have the following conditional statements:

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

The if Statement

Use the `if` statement 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
}
```

The else Statement

Use the `else` statement 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
}
```

The else if Statement

Use the `else if` statement 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
}
```

The JavaScript Switch Statement

Use the `switch` statement to select one of many code blocks to be executed.

Syntax

```
switch(expression){
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

This is how it works:

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.
- If there is no match, the default code block is executed.

The default Keyword

The `default` keyword specifies the code to run if there is no case match:

The `default` case does not have to be the last case in a switch block:

Example

```
switch (new Date().getDay()){
  default:
    text = "Looking forward to the Weekend";
    break;
  case 6:
    text = "Today is Saturday";
    break;
  case 0:
    text = "Today is Sunday";
}
```

JavaScript Loops

Different Kinds of Loops

JavaScript supports different kinds of loops:

- **`for`** - loops through a block of code a number of times
- **`while`** - loops through a block of code while a specified condition is true
- **`do/while`** - also loops through a block of code while a specified condition is true

The For Loop

The `for` loop has the following syntax:

```
for (statement 1; statement 2; statement 3) {
  // code block to be executed
}
```

The While Loop

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

Syntax

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


The Do/While Loop

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

Syntax

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

JavaScript Break and Continue

The Break Statement

The `break` statement can also be used to jump out of a loop.

The `break` statement breaks the loop and continues executing the code after the loop (if any):

Example

```
for (i = 0; i < 10; i++) {
  if (i === 3) { break; }
  text += "The number is " + i + "<br>";
}
```

The Continue Statement

The `continue` statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

This example skips the value of 3:

Example

```
for (i = 0; i < 10; i++) {
  if (i === 3) { continue; }
  text += "The number is " + i + "<br>";
}
```


JavaScript Debugging

Code Debugging

Searching for (and fixing) errors in programming code is called code debugging.

JavaScript Debuggers

Debugging is not easy. But fortunately, all modern browsers have a built-in JavaScript debugger.

you activate debugging in your browser with the F12 key, and select "Console" in the debugger menu.

The console.log() Method

If your browser supports debugging, you can use `console.log()` to display JavaScript values in the debugger window:

JavaScript HTML DOM

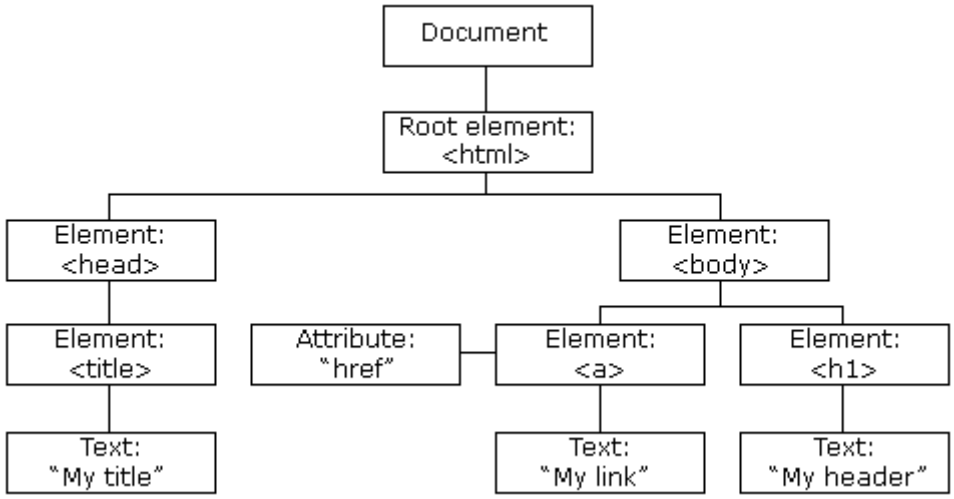
With the HTML DOM, JavaScript can access and change all the elements of an HTML document.

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 is the DOM?

The DOM defines a standard for accessing documents:

"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

The W3C DOM standard is separated into 3 different parts:

- Core DOM - standard model for all document types
- XML DOM - standard model for XML documents
- HTML DOM - standard model for HTML documents

JavaScript - HTML DOM Methods

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

```
<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 `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

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.

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

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

JavaScript HTML DOM Elements

Finding HTML Elements

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

Example

```
var myElement = document.getElementById("intro");
```

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

Finding HTML Elements by Tag Name

Example

```
var x = document.getElementsByTagName("p");
```

Finding HTML Elements by Class Name

Example

```
var 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

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

Changing HTML Content

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

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

Changing the Value of an Attribute

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

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

JavaScript HTML DOM - Changing CSS

Changing HTML Style

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

document.getElementById(*id*).style.*property* = *new style*

Using Events

Events are generated by the browser when "things happen" to HTML elements:

- An element is clicked on
- The page has loaded
- Input fields are changed

Example

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

<h1 id="id1">My Heading 1</h1>

<button type="button"
onclick="document.getElementById('id1').style.color = 'red'">
Click Me!</button>

</body>
</html>
```

JavaScript HTML DOM Events

Examples of HTML events:

- When a user clicks the mouse
- When a web page has loaded
- When an image has been loaded
- When the mouse moves over an element
- When an input field is changed
- When an HTML form is submitted
- When a user strokes a key

To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute:

onclick=*JavaScript*

The onchange Event

The `onchange` event is often used in combination with validation of input fields.

Example

```
<input type="text" id="fname" onchange="upperCase()">
```

The onmouseover and onmouseout Events

The `onmouseover` and `onmouseout` events can be used to trigger a function when the user mouses over, or out of, an HTML element:

The onmousedown, onmouseup and onclick Events

The `onmousedown`, `onmouseup`, and `onclick` events are all parts of a mouse-click. First when a mouse-button is clicked, the onmousedown event is triggered, then, when the mouse-button is released, the onmouseup event is triggered, finally, when the mouse-click is completed, the onclick event is triggered.

JavaScript HTML DOM EventListener

The addEventListener() method

The `addEventListener()` method attaches an event handler to the specified element.

When using the `addEventListener()` method, the JavaScript is separated from the HTML markup, for better readability and allows you to add event listeners even when you do not control the HTML markup.

You can easily remove an event listener by using the `removeEventListener()` method.

Syntax

```
element.addEventListener(event, function, useCapture);
```

The first parameter is the type of the event (like "`click`" or "`mousedown`" or any other HTML DOM Event.)

The second parameter is the function we want to call when the event occurs.

The third parameter is a boolean value specifying whether to use event bubbling or event capturing. This parameter is optional.

Note that you don't use the "on" prefix for the event; use "`click`" instead of "`onclick`".

Add an Event Handler to an Element

Example

Alert "Hello World!" when the user clicks on an element:

```
element.addEventListener("click", function(){ alert("Hello World!"); });
```

You can also refer to an external "named" function:

Example

Alert "Hello World!" when the user clicks on an element:

```
element.addEventListener("click", myFunction);

function myFunction() {
  alert ("Hello World!");
}
```


Prompt Box

A prompt box is often used if you want the user to input a value before entering a page.

When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.

If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

Syntax

```
window.prompt("sometext","defaultText");
```

Example

```
var person = prompt("Please enter your name", "Harry Potter");
```

```
if (person == null || person == "") {
  txt = "User cancelled the prompt.";
} else {
  txt = "Hello " + person + "! How are you today?";
}
```

JavaScript Timing Events

Timing Events

The `window` object allows execution of code at specified time intervals.

These time intervals are called timing events.

The two key methods to use with JavaScript are:

- `setTimeout(`*`function, milliseconds`*`)`
  Executes a function, after waiting a specified number of milliseconds.

- `setInterval(`*`function, milliseconds`*`)`
  Same as setTimeout(), but repeats the execution of the function continuously.

The setTimeout() Method

Example

Click a button. Wait 3 seconds, and the page will alert "Hello":

```
<button onclick="setTimeout(myFunction, 3000)">Try it</button>

<script>
function myFunction() {
  alert('Hello');
}
</script>
```

How to Stop the Execution?

The `clearTimeout()` method stops the execution of the function specified in setTimeout().

window.clearTimeout(*timeoutVariable*)

The `clearTimeout()` method uses the variable returned from `setTimeout()`:

```
myVar = setTimeout(function, milliseconds);
clearTimeout(myVar);
```

The setInterval() Method

Example

Display the current time:

```
var myVar = setInterval(myTimer, 1000);

function myTimer() {
  var d = new Date();
  document.getElementById("demo").innerHTML = d.toLocaleTimeString();
}
```

How to Stop the Execution?

The `clearInterval()` method stops the executions of the function specified in the setInterval() method.

```
window.clearInterval(timerVariable)
```

The `clearInterval()` method uses the variable returned from `setInterval()`:

```
myVar = setInterval(function, milliseconds);
clearInterval(myVar);
```