**WEB PROGRAMMING**

JavaScript

JavaScript (js) is a light-weight object-oriented programming language which is used by several websites for scripting the webpages. It is an interpreted, full-fledged programming language that enables dynamic interactivity on websites when applied to an HTML document. It was introduced in the year 1995 for adding programs to the webpages in the Netscape Navigator browser. Since then, it has been adopted by all other graphical web browsers. With JavaScript, users can build modern web applications to interact directly without reloading the page every time. The traditional website uses js to provide several forms of interactivity and simplicity.

Although, JavaScript has no connectivity with Java programming language. The name was suggested and provided in the times when Java was gaining popularity in the market. In addition to web browsers, databases such as CouchDB and MongoDB uses JavaScript as their scripting and query language.

Features of JavaScript

There are following features of JavaScript:

1. All popular web browsers support JavaScript as they provide built-in execution environments.
2. JavaScript follows the syntax and structure of the C programming language. Thus, it is a structured programming language.
3. JavaScript is a weakly typed language, where certain types are implicitly cast (depending on the operation).
4. JavaScript is an object-oriented programming language that uses prototypes rather than using classes for inheritance.
5. It is a light-weighted and interpreted language.
6. It is a case-sensitive language.
7. JavaScript is supportable in several operating systems including, Windows, macOS, etc.
8. It provides good control to the users over the web browsers.

History of JavaScript

In 1993, **Mosaic**, the first popular web browser, came into existence. In the **year 1994**, **Netscape** was founded by **Marc Andreessen**. He realized that the web needed to become more dynamic. Thus, a 'glue language' was believed to be provided to HTML to make web designing easy for designers and part-time programmers. Consequently, in 1995, the company recruited **Brendan Eich** intending to implement and embed Scheme programming language to the browser. But, before Brendan could start, the company merged with **Sun Microsystems** for adding Java into its Navigator so that it could compete with Microsoft over the web technologies and platforms. Now, two languages were there: Java and the scripting language. Further, Netscape decided to give a similar name to the scripting language as Java's. It led to 'Javascript'. Finally, in May 1995, Marc Andreessen coined the first code of Javascript named '**Mocha**'. Later, the marketing team replaced the name with '**LiveScript**'. But, due to trademark reasons and certain other reasons, in December 1995, the language was finally renamed to 'JavaScript'. From then, JavaScript came into existence.

Application of JavaScript

JavaScript is used to create interactive websites. It is mainly used for:

* Client-side validation,
* Dynamic drop-down menus,
* Displaying date and time,
* Displaying pop-up windows and dialog boxes (like an alert dialog box, confirm dialog box and prompt dialog box),
* Displaying clocks etc.

JavaScript Example

1. **<script>**
2. document.write("Hello JavaScript by JavaScript");
3. **</script>**

HTML5 introduces two mechanisms, similar to HTTP session cookies, for storing structured data on the client side and to overcome following drawbacks.

* Cookies are included with every HTTP request, thereby slowing down your web application by transmitting the same data.
* Cookies are included with every HTTP request, thereby sending data unencrypted over the internet.
* Cookies are limited to about 4 KB of data. Not enough to store required data.

The two storages are **session storage** and **local storage** and they would be used to handle different situations.

The latest versions of pretty much every browser supports HTML5 Storage including Internet Explorer.

## Local Storage

The *Local Storage* is designed for storage that spans multiple windows, and lasts beyond the current session. In particular, Web applications may wish to store megabytes of user data, such as entire user-authored documents or a user's mailbox, on the client side for performance reasons.

Again, cookies do not handle this case well, because they are transmitted with every request.

### Example

HTML5 introduces the *localStorage* attribute which would be used to access a page's local storage area without no time limit and this local storage will be available whenever you would use that page.

Following is the code which would set a local storage variable and access that variable every time this page is accessed, even next time, when you open the window −

<!DOCTYPE HTML>

<html>

<body>

<script type = "text/javascript">

if( localStorage.hits ) {

localStorage.hits = Number(localStorage.hits) +1;

} else {

localStorage.hits = 1;

}

document.write("Total Hits :" + localStorage.hits );

</script>

<p>Refresh the page to increase number of hits.</p>

<p>Close the window and open it again and check the result.</p>

</body>

</html>

This will produce the following result −

Total Hits :3

Refresh the page to increase number of hits.

Close the window and open it again and check the result.

# **Web Workers**

JavaScript was designed to run in a single-threaded environment, meaning multiple scripts cannot run at the same time. Consider a situation where you need to handle UI events, query and process large amounts of API data, and manipulate the DOM.

JavaScript will hang your browser in situation where CPU utilization is high. Let us take a simple example where JavaScript goes through a big loop −

<!DOCTYPE HTML>

<html>

<head>

<title>Big for loop</title>

<script>

function bigLoop() {

for (var i = 0; i <= 10000; i += 1) {

var j = i;

}

alert("Completed " + j + "iterations" );

}

function sayHello(){

alert("Hello sir...." );

}

</script>

</head>

<body>

<input type = "button" onclick = "bigLoop();" value = "Big Loop" />

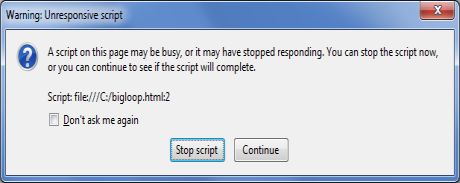
<input type = "button" onclick = "sayHello();" value = "Say Hello" />

</body>

</html>

It will produce the following result −

When you click Big Loop button it displays following result in Firefox −



## What is Web Workers?

The situation explained above can be handled using **Web Workers** who will do all the computationally expensive tasks without interrupting the user interface and typically run on separate threads.

Web Workers allow for long-running scripts that are not interrupted by scripts that respond to clicks or other user interactions, and allows long tasks to be executed without yielding to keep the page responsive.

Web Workers are background scripts and they are relatively heavy-weight, and are not intended to be used in large numbers. For example, it would be inappropriate to launch one worker for each pixel of a four megapixel image.

When a script is executing inside a Web Worker it cannot access the web page's window object (window.document), which means that Web Workers don't have direct access to the web page and the DOM API. Although Web Workers cannot block the browser UI, they can still consume CPU cycles and make the system less responsive.

## How Web Workers Work?

Web Workers are initialized with the URL of a JavaScript file, which contains the code the worker will execute. This code sets event listeners and communicates with the script that spawned it from the main page. Following is the simple syntax −

var worker = new Worker('bigLoop.js');

If the specified javascript file exists, the browser will spawn a new worker thread, which is downloaded asynchronously. If the path to your worker returns an 404 error, the worker will fail silently.

If your application has multiple supporting JavaScript files, you can import them **importScripts()** method which takes file name(s) as argument separated by comma as follows −

importScripts("helper.js", "anotherHelper.js");

Once the Web Worker is spawned, communication between web worker and its parent page is done using the **postMessage()** method. Depending on your browser/version, postMessage() can accept either a string or JSON object as its single argument.

Message passed by Web Worker is accessed using **onmessage** event in the main page. Now let us write our bigLoop example using Web Worker. Below is the main page (hello.htm) which will spawn a web worker to execute the loop and to return the final value of variable**j** −

[Live Demo](http://tpcg.io/cWcw0d)

<!DOCTYPE HTML>

<html>

<head>

<title>Big for loop</title>

<script>

var worker = new Worker('bigLoop.js');

worker.onmessage = function (event) {

alert("Completed " + event.data + "iterations" );

};

function sayHello() {

alert("Hello sir...." );

}

</script>

</head>

<body>

<input type = "button" onclick = "sayHello();" value = "Say Hello"/>

</body>

</html>

Following is the content of bigLoop.js file. This makes use of **postMessage()** API to pass the communication back to main page −

for (var i = 0; i <= 1000000000; i += 1) {

var j = i;

}

postMessage(j);

This will produce the following result −

## Stopping Web Workers

Web Workers don't stop by themselves but the page that started them can stop them by calling **terminate()** method.

worker.terminate();

A terminated Web Worker will no longer respond to messages or perform any additional computations. You cannot restart a worker; instead, you can create a new worker using the same URL.

## Handling Errors

The following shows an example of an error handling function in a Web Worker JavaScript file that logs errors to the console. With error handling code, above example would become as following −

<!DOCTYPE HTML>

<html>

<head>

<title>Big for loop</title>

<script>

var worker = new Worker('bigLoop.js');

worker.onmessage = function (event) {

alert("Completed " + event.data + "iterations" );

};

worker.onerror = function (event) {

console.log(event.message, event);

};

function sayHello() {

alert("Hello sir...." );

}

</script>

</head>

<body>

<input type = "button" onclick = "sayHello();" value = "Say Hello"/>

</body>

</html>

## Checking for Browser Support

Following is the syntax to detect a Web Worker feature support available in a browser −

[Live Demo](http://tpcg.io/Fmx7cx)

<!DOCTYPE HTML>

<html>

<head>

<title>Big for loop</title>

<script src = "/js/modernizr-1.5.min.js"></script>

<script>

function myFunction() {

if (Modernizr.webworkers) {

alert("Congratulation!! you have web workers support." );

} else {

alert("Sorry!! you do not have web workers support." );

}

}

</script>

</head>

<body>

<button onclick = "myFunction()">Click me</button>

</body>

</html>

This will produce the following result −

Click me

HTML5 - Drag & drop

Drag and Drop (DnD) is powerful User Interface concept which makes it easy to copy, reorder and deletion of items with the help of mouse clicks. This allows the user to click and hold the mouse button down over an element, drag it to another location, and release the mouse button to drop the element there.

To achieve drag and drop functionality with traditional HTML4, developers would either have to either have to use complex JavaScript programming or other JavaScript frameworks like jQuery etc.

Now HTML 5 came up with a Drag and Drop (DnD) API that brings native DnD support to the browser making it much easier to code up.

HTML 5 DnD is supported by all the major browsers like Chrome, Firefox 3.5 and Safari 4 etc.

Drag and Drop Events

There are number of events which are fired during various stages of the drag and drop operation. These events are listed below −

Sr.No. Events & Description

1

dragstart

Fires when the user starts dragging of the object.

2

dragenter

Fired when the mouse is first moved over the target element while a drag is occurring. A listener for this event should indicate whether a drop is allowed over this location. If there are no listeners, or the listeners perform no operations, then a drop is not allowed by default.

3

dragover

This event is fired as the mouse is moved over an element when a drag is occurring. Much of the time, the operation that occurs during a listener will be the same as the dragenter event.

4

dragleave

This event is fired when the mouse leaves an element while a drag is occurring. Listeners should remove any highlighting or insertion markers used for drop feedback.

5

drag

Fires every time the mouse is moved while the object is being dragged.

6

drop

The drop event is fired on the element where the drop was occurred at the end of the drag operation. A listener would be responsible for retrieving the data being dragged and inserting it at the drop location.

7

dragend

Fires when the user releases the mouse button while dragging an object.

Note − Note that only drag events are fired; mouse events such as mousemove are not fired during a drag operation.

The DataTransfer Object

The event listener methods for all the drag and drop events accept Event object which has a readonly attribute called dataTransfer.

The event.dataTransfer returns DataTransfer object associated with the event as follows −

function EnterHandler(event) {

DataTransfer dt = event.dataTransfer;

.............

}

The DataTransfer object holds data about the drag and drop operation. This data can be retrieved and set in terms of various attributes associated with DataTransfer object as explained below −

Sr.No. DataTransfer attributes and their description

1

dataTransfer.dropEffect [ = value ]

Returns the kind of operation that is currently selected.

This attribute can be set, to change the selected operation.

The possible values are none, copy, link, and move.

2

dataTransfer.effectAllowed [ = value ]

Returns the kinds of operations that are to be allowed.

This attribute can be set, to change the allowed operations.

The possible values are none, copy, copyLink, copyMove, link, linkMove, move, all and uninitialized.

3

dataTransfer.types

Returns a DOMStringList listing the formats that were set in the dragstart event. In addition, if any files are being dragged, then one of the types will be the string "Files".

4

dataTransfer.clearData ( [ format ] )

Removes the data of the specified formats. Removes all data if the argument is omitted.

5

dataTransfer.setData(format, data)

Adds the specified data.

6

data = dataTransfer.getData(format)

Returns the specified data. If there is no such data, returns the empty string.

7

dataTransfer.files

Returns a FileList of the files being dragged, if any.

8

dataTransfer.setDragImage(element, x, y)

Uses the given element to update the drag feedback, replacing any previously specified feedback.

9

dataTransfer.addElement(element)

Adds the given element to the list of elements used to render the drag feedback.

Drag and Drop Process

Following are the steps to be carried out to implement Drag and Drop operation −

Step 1 - Making an Object Draggable

Here are steps to be taken −

If you want to drag an element, you need to set the draggable attribute to true for that element.

Set an event listener for dragstart that stores the data being dragged.

The event listener dragstart will set the allowed effects (copy, move, link, or some combination).

Following is the example to make an object draggable −

Live Demo

<!DOCTYPE HTML>

<html>

<head>

<style type = "text/css">

#boxA, #boxB {

float:left;padding:10px;margin:10px; -moz-user-select:none;

}

#boxA { background-color: #6633FF; width:75px; height:75px; }

#boxB { background-color: #FF6699; width:150px; height:150px; }

</style>

<script type = "text/javascript">

function dragStart(ev) {

ev.dataTransfer.effectAllowed = 'move';

ev.dataTransfer.setData("Text", ev.target.getAttribute('id'));

ev.dataTransfer.setDragImage(ev.target,0,0);

return true;

}

</script>

</head>

<body>

<center>

<h2>Drag and drop HTML5 demo</h2>

<div>Try to drag the purple box around.</div>

<div id = "boxA" draggable = "true"

ondragstart = "return dragStart(ev)">

<p>Drag Me</p>

</div>

<div id = "boxB">Dustbin</div>

</center>

</body>

</html>

This will produce the following result −

Step 2 - Dropping the Object

To accept a drop, the drop target has to listen to at least three events.

The dragenter event, which is used to determine whether or not the drop target is to accept the drop. If the drop is to be accepted, then this event has to be canceled.

The dragover event, which is used to determine what feedback is to be shown to the user. If the event is canceled, then the feedback (typically the cursor) is updated based on the dropEffect attribute's value.

Finally, the drop event, which allows the actual drop to be performed.

Following is the example to drop an object into another object −

Live Demo

<html>

<head>

<style type="text/css">

#boxA, #boxB {

float:left;padding:10px;margin:10px;-moz-user-select:none;

}

#boxA { background-color: #6633FF; width:75px; height:75px; }

#boxB { background-color: #FF6699; width:150px; height:150px; }

</style>

<script type="text/javascript">

function dragStart(ev) {

ev.dataTransfer.effectAllowed='move';

ev.dataTransfer.setData("Text", ev.target.getAttribute('id'));

ev.dataTransfer.setDragImage(ev.target,0,0);

return true;

}

function dragEnter(ev) {

event.preventDefault();

return true;

}

function dragOver(ev) {

return false;

}

function dragDrop(ev) {

var src = ev.dataTransfer.getData("Text");

ev.target.appendChild(document.getElementById(src));

ev.stopPropagation();

return false;

}

</script>

</head>

<body>

<center>

<h2>Drag and drop HTML5 demo</h2>

<div>Try to move the purple box into the pink box.</div>

<div id="boxA" draggable="true" ondragstart="return dragStart(event)">

<p>Drag Me</p>

</div>

<div id="boxB" ondragenter="return dragEnter(event)" ondrop="return dragDrop(event)" ondragover="return dragOver(event)">Dustbin</div>

</center>

</body>

</html>

This will produce the following result −

**Drag and drop HTML5 demo**

Try to drag the purple box around.

**Client-side scripting**

[Web browsers](https://www.geeksforgeeks.org/difference-between-web-browser-and-web-server/) execute client-side scripting. It is used when browsers have all code. Source code is used to transfer from [webserver](https://www.geeksforgeeks.org/web-server-and-its-type/) to user’s computer over the [internet](https://www.geeksforgeeks.org/the-internet-and-the-web/) and run directly on browsers. It is also used for validations and functionality for user events.

It allows for more interactivity. It usually performs several actions without going to the user. It cannot be basically used to connect to databases on a web server. These scripts cannot access the file system that resides in the web browser. Pages are altered on basis of the user’s choice. It can also be used to create “cookies” that store data on the user’s computer.

Client-side scripting

* Source code is visible to the user.
* It usually depends on the browser and its version.
* It runs on the user’s computer.
* There are many advantages linked with this like faster.

response times, a more interactive application.

* It does not provide security for data.
* It is a technique used in web development in which scripts run on the client’s browser.
* HTML, CSS, and javascript are used.

JavaScript Basics, Functions, Objects, Arrays, JavaScript Objects – learn from javatpoint - https://www.javatpoint.com/javascript-comment.

## JavaScript Declarations are Hoisted

In JavaScript, a variable can be declared after it has been used.

In other words; a variable can be used before it has been declared.

**Example 1** gives the same result as **Example 2**:

### Example 1

x = 5; // Assign 5 to x  
  
elem = document.getElementById("demo"); // Find an element  
elem.innerHTML = x;                     // Display x in the element  
  
var x; // Declare x

### Example 2

var x; // Declare x  
x = 5; // Assign 5 to x  
  
elem = document.getElementById("demo"); // Find an element  
elem.innerHTML = x;                     // Display x in the element

To understand this, you have to understand the term "hoisting".

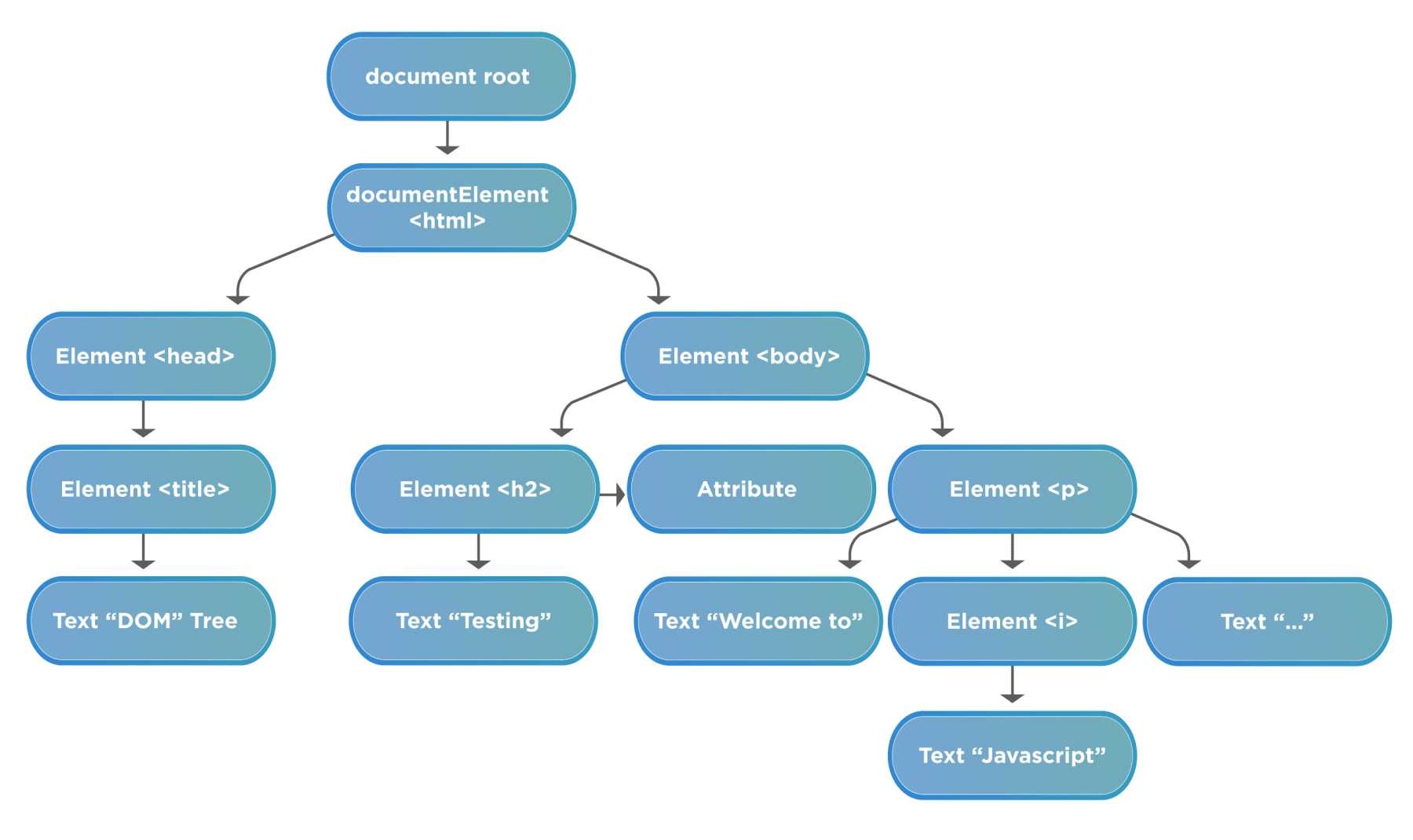
Hoisting is JavaScript's default behavior of moving all declarations to the top of the current scope (to the top of the current script or the current function).

**Unit II**

# DOM in JavaScript

## What is DOM?

[***DOM***](https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction) is a data representation of the objects in the HTML and XML pages. The document loaded in your browser is represented by a ***document object model***. Moreover, it is a "***tree structure***" representation created by the browser that enables the HTML structure to be easily accessed by programming languages. Additionally, the DOM represents the document as nodes and objects. In this way, programming languages can connect to the page. Furthermore, a simple structure of a web page DOM will look like below:



Before starting working on the DOM, we should understand the following terminologies related to DOM:

* ***Element node***: It represents any element that exists in the DOM.
* ***Root node***: This is the top node in the tree, which in the case of HTML, is always the "***HTML***" node.
* ***Child node***: This is a node that is directly inside another node. For example, *<h2>* is a child of  *<body>* in the above example.
* ***Descendant node***: \*This is a node that exists anywhere in the hierarchy of another node. For example, <h2> is a descendant of <html> in the above example.
* ***Parent node***: This is a node that has another node inside it. For example, *<body>* is the parent node of  *<h2>* in the above example.
* ***Sibling nodes***: The nodes which sit on the same level in the DOM tree. For example, *<h2>* and  *<p>* are siblings in the above example.
* ***Text node***: This is a node that contains a text string.

## How to access DOM elements using JavaScript?

A webpage in JavaScript is a document, and JavaScript provides an object "***document***", which designates the complete webpage. Moreover, the document object provides various properties and methods to access and manipulate the web elements loaded on the page. To identify and access the DOM elements, JavaScript uses three ways:

* First, Accessing elements By ID
* Second, Accessing elements By TagName
* Third, Accessing elements By className

Let's understand the details of each of these ways in the following sections:

### *****Accessing a DOM element By ID:*****

JavaScript can find HTML elements in the DOM based on the "***id***" of the element. The document object provides a method "***getElementById()***" to accomplish this task. Moreover, its syntax looks like below:

***Syntax:***

document.getElementById(“IDName”);

* Let's understand the usage of the "***getElementById***" method with the help of the following code snippet.

<html>

<body>

Demonstrating getElementById in javascript:    </br>

   <b id="bold">Tools QA Tutorial</b>

  <script type="text/javascript">

// Get the element by Id and update text on that

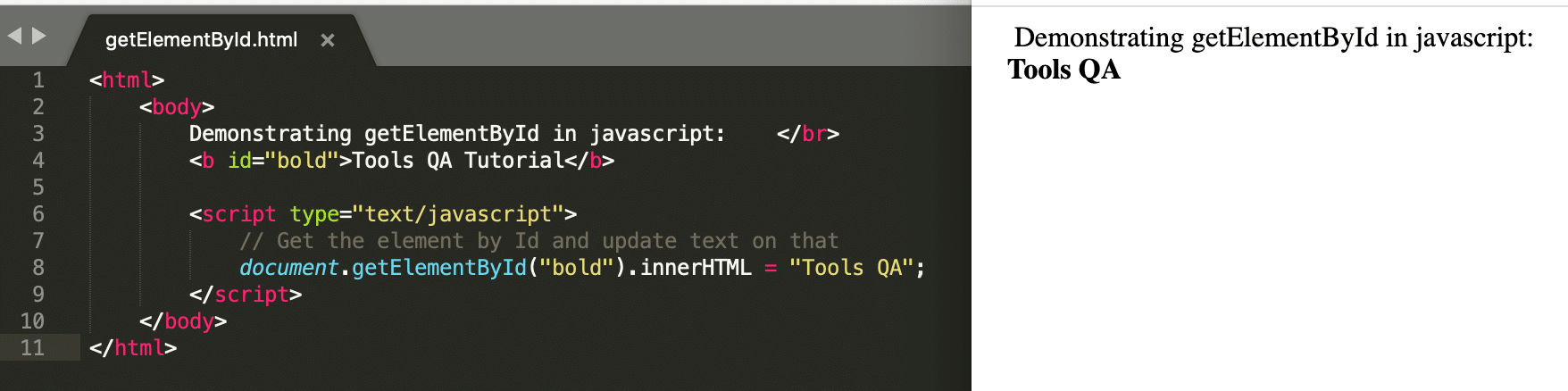
document.getElementById("bold").innerHTML = "Tools QA";

</script>

</body>

</html>

* Save the file with name  ***getElementByID.html***. After that, open it in any browser (Chrome, Firefox, or IE). It should show the output as:



* In the above example, we can see that element ***by id*** "***bold***" has been found. In addition to that, we also changed its attribute innerHTML to Tools QA.

***Note:***- We will be covering the InnerHTML function in the below section, "***How to manipulate DOM elements by using JavaScript?***"

### *****Accessing a DOM element By TagName:*****

JavaScript can find the elements in the HTML based on the "***tagName***" and return an array of matching nodes. The inbuilt function, of document object, available for this is getElementByTagName(). Additionally, its syntax looks like below:

***Syntax:***

document.getElementByTagName(“tagName”);

* Let's understand the usage of "***getElementByTagName***" method with the help of following code snippet:

<html>

<body>

Demonstrating getElementByTag in javascript:    </br>

   <b>Tools QA Tutorial</b>

  <script type="text/javascript">

// Get the element by tag <b> and update text on that

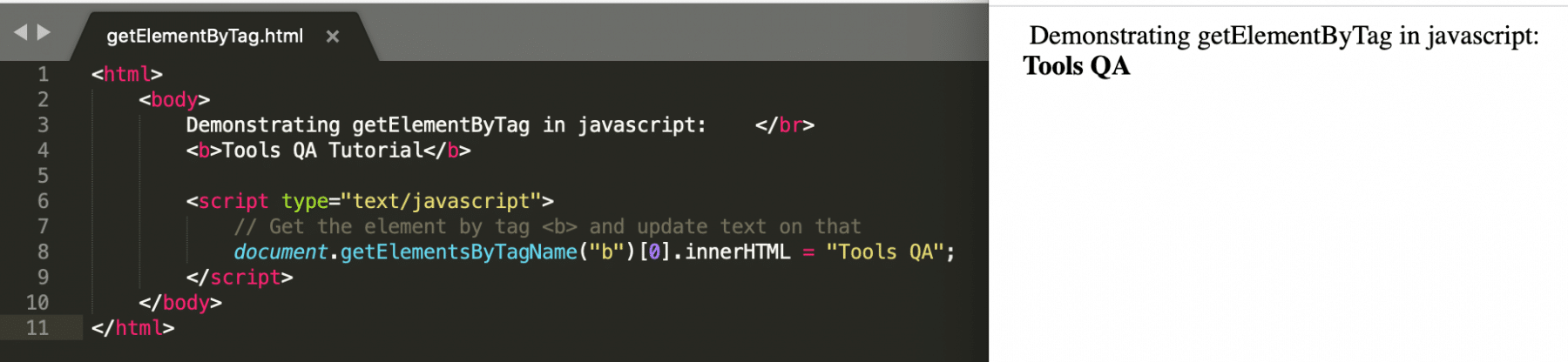
document.getElementsByTagName("b")[0].innerHTML = "Tools QA";

</script>

</body>

</html>

* Save the file with name  ***getElementByTagName.html***. After that, open it in any browser (Chrome, Firefox, or IE). It should show the output as:



* In the above example, we can see that the element ***by HTML tag "<b>"*** has been found. Additionally, we changed its attribute innerHTML to Tools QA.

### *****Accessing a DOM element By ClassName:*****

JavaScript can find the element in the HTML based on the className attribute of the element and returns an array of matching nodes. The inbuilt function available in this operation is ***getElementByClassName().*** Additionally, its syntax looks like below:

***Syntax:***

document.getElementByClassName(“ClassName”);

* \*Let's understand the usage of "***getElementByClassName***" method with the help of following code snippet:

<html>

<body>

Demonstrating getElementsByClassName in javascript:    </br>

   <b class="bold">Tools QA Tutorial</b>

  <script type="text/javascript">

// Get the element by className "bold" and update text on that

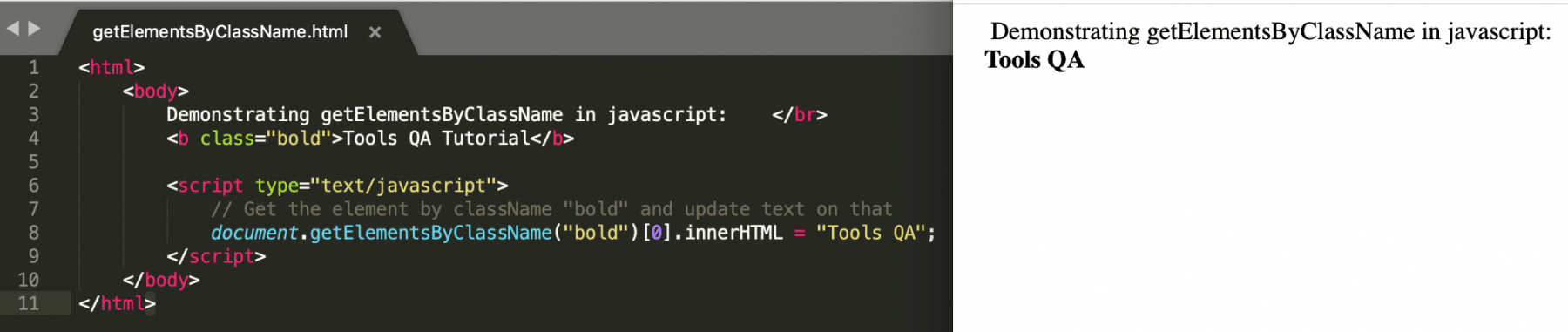
document.getElementsByClassName("bold")[0].innerHTML = "Tools QA";

</script>

</body>

</html>

* Save the file with name  ***getElementByClassName.html***. After that, open it in any browser (Chrome, Firefox, or IE). It should show the output as:



In the above example, we can see that element ***by className "bold"*** has been found. Additionally, we changed its attribute innerHTML to Tools QA.

### *****How to manipulate DOM elements by using JavaScript?*****

Apart from accessing the elements, JavaScript provides some methods and properties which can manipulate or change the values of the DOM elements. Few of those methods and properties are:

* write
* innerHTML
* attributeName
* Style.property
* setAttribute
* createElement
* appendChild
* removeChild
* replaceChild

Let's discuss all of these methods and properties in detail in the below sections:

#### **write**

This method writes new elements or text to the HTML page. Additionally, its syntax looks like below:

***Syntax:***

document.write(“data”);

* Let's understand the usage of "***write()***" method with the help of following code snippet:

<html>

<body>

Demonstrating document's write function in javascript:

</br>

<script type = "text/javascript">

document.write("Tools QA Tutorials");

document.write("</br>");

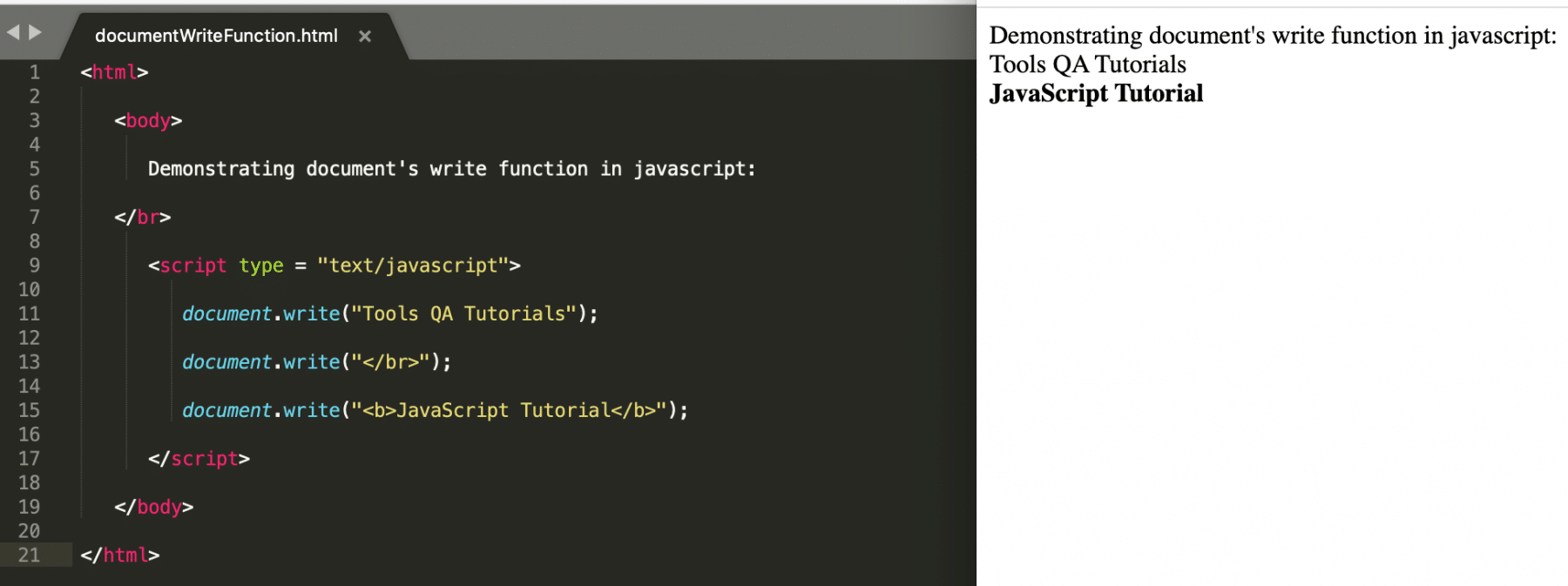
document.write("<b>JavaScript Tutorial</b>");

</script>

</body>

</html>

* Save the file with the name  ***documentWriteFunction.html***. After that, open it in any browser (Chrome, Firefox, or IE). It should show the output as:



* In the above example, we use the write function to add a text as well as to add a new HTML element.

#### **innerHTML**

It is a property that we use to get or set the HTML or XML markup contained within the element. Also, its syntax looks like below:

***Syntax:***

node.innerhtml = “changingText”;

Where,

***node***: is any web element that can be found on the web page using ***document.getElementBy<Id/tagName/className>.***

* Let's understand the usage of "***innerHTML***" property with the help of following code snippet:

<html>

<body>

Demonstrating innerHTML property in javascript

</br>

<b id="example">JavaScript Tutorial</b>

<script type = "text/javascript">

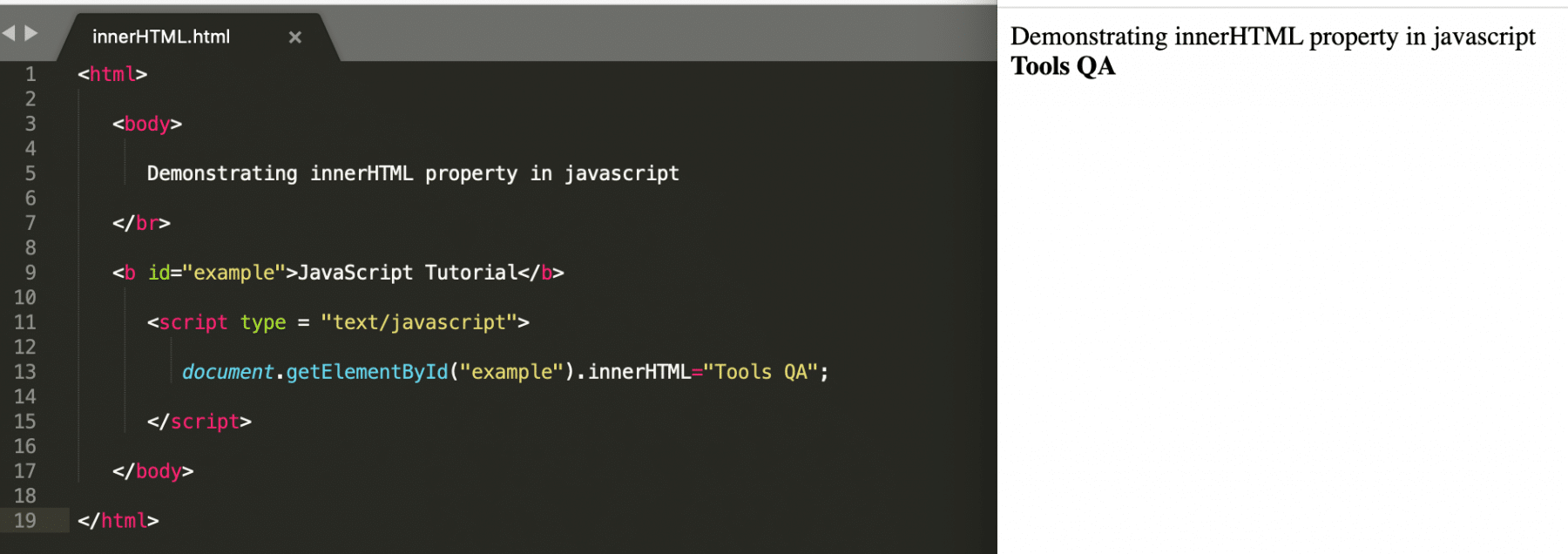
document.getElementById("example").innerHTML="Tools QA";

</script>

</body>

</html>

* Save the file with the name  ***innerHTML.html***. After that, open it in any browser (Chrome, Firefox, or IE). It should show the output as:



* In the above example, we can see that the ***innerHTML property*** updates the text of the HTML element.

#### **attributeName**

We use his property is used to get and update the value of an attribute of an HTML element. Additionally, its syntax looks like below:

***Syntax:***

node.atrributeName = value;

Where,

***node***: is any web element that can be found on the web page using ***document.getElementBy<Id/tagName/className>***.

* Let's understand the usage of "***attributeName***" property with the help of following code snippet:

<html>

<body>

Demonstrating attributeName property in javascript

</br>

<b id="example">JavaScript Tutorial</b>

<script type = "text/javascript">

// Update the "id" of the element to "demo" which has "id" as "example"

document.getElementById("example").id="demo";

// Get the element with "id" as "demo" and update its innerHTML text

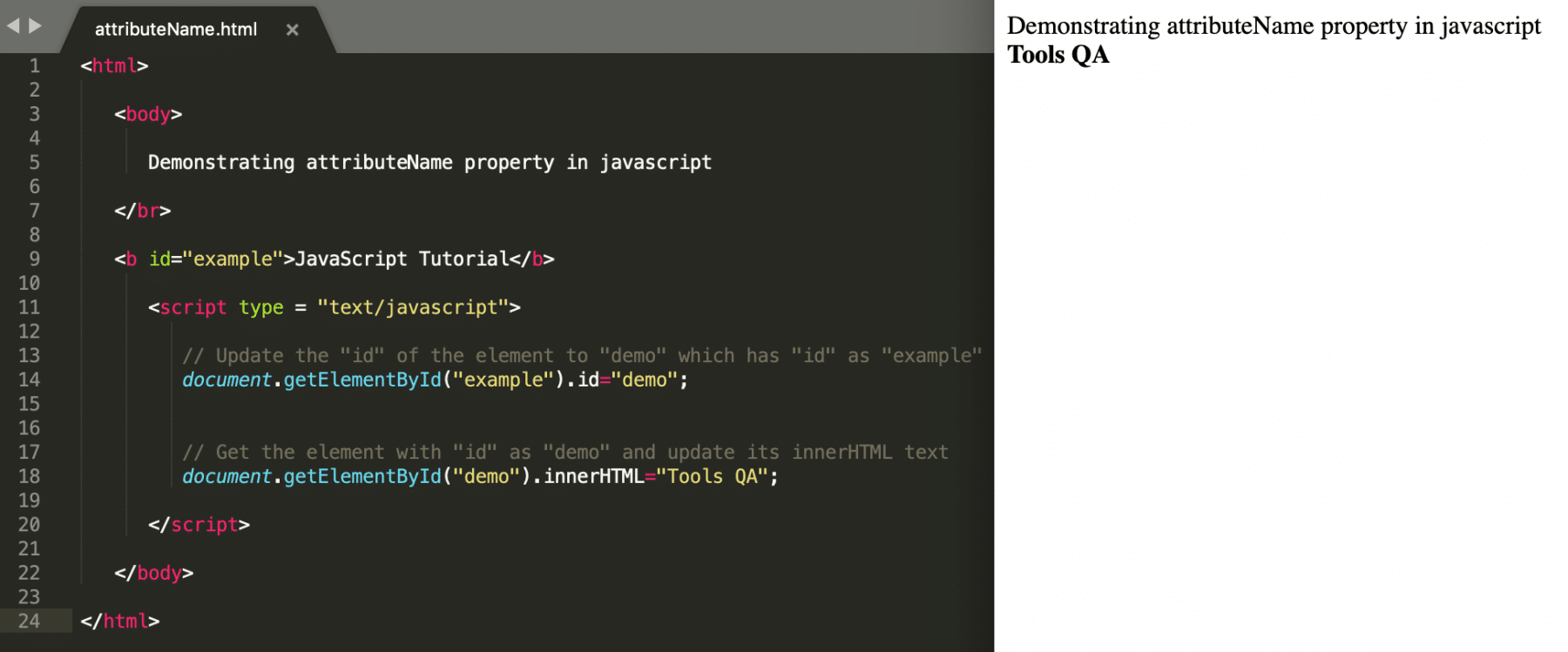
document.getElementById("demo").innerHTML="Tools QA";

</script>

</body>

</html>

* Save the file with name  ***attributeName.html***. After that, open it in any browser (Chrome, Firefox, or IE). It should show the output as:



* In the above example, the element is first changed from "***example***" to "***demo***" and then based on id "***demo***," we are manipulating the text of the element.

#### **Style.property**

We use this property to set or edit the existing style properties of an HTML tag. Also, its syntax looks like below:

***Syntax:***

node.Style.attribute = value;

Where,

***node***: is any web element that can be found on the web page using ***document.getElementBy<Id/tagName/className>***.

* Let's understand the usage of "***Style.attribute***" method with the help of following code snippet:

<html>

<body>

Demonstrating updating style properties in javascript

</br>

<b id="example">JavaScript Tutorial</b>

<script type = "text/javascript">

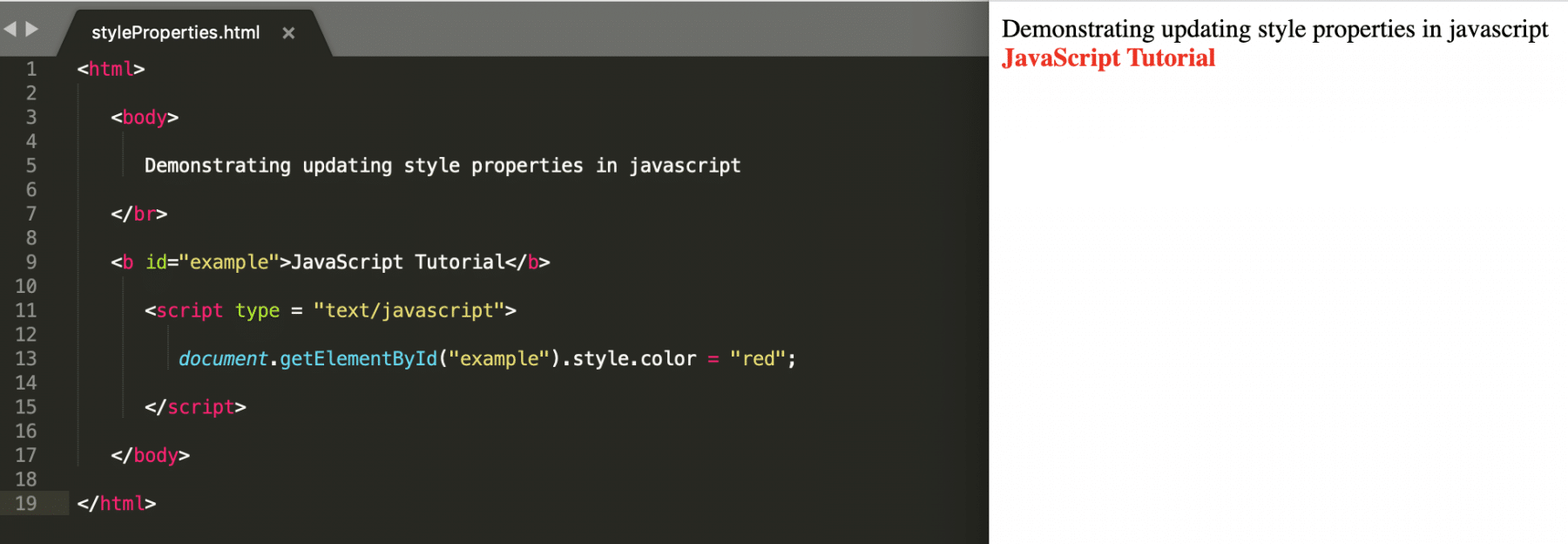
document.getElementById("example").style.color = "red";

</script>

</body>

</html>

* Save the file with name  ***styleProperties.html***. After that, open it in any browser (Chrome, Firefox, or IE). It should show the output as:



* In the above example, the style attribute "***color***" has been updated to "***red***".

#### **setAttribute**

We use this function to create or update an attribute for the existing HTML element. Additionally, its syntax looks like below:

***Syntax:***

node.setAttribute(attributeName, attributeValue);

Where,

***node***: is any web element that can be found on the web page using ***document.getElementBy<Id/tagName/className>***.

* Let's understand the usage of "***setAttribute()***" method with the help of following code snippet:

<html>

<body>

Demonstrating setAttribute function in javascript

</br>

<b>JavaScript Tutorial</b>

<script type = "text/javascript">

document.getElementsByTagName("b")[0].setAttribute("id","example");

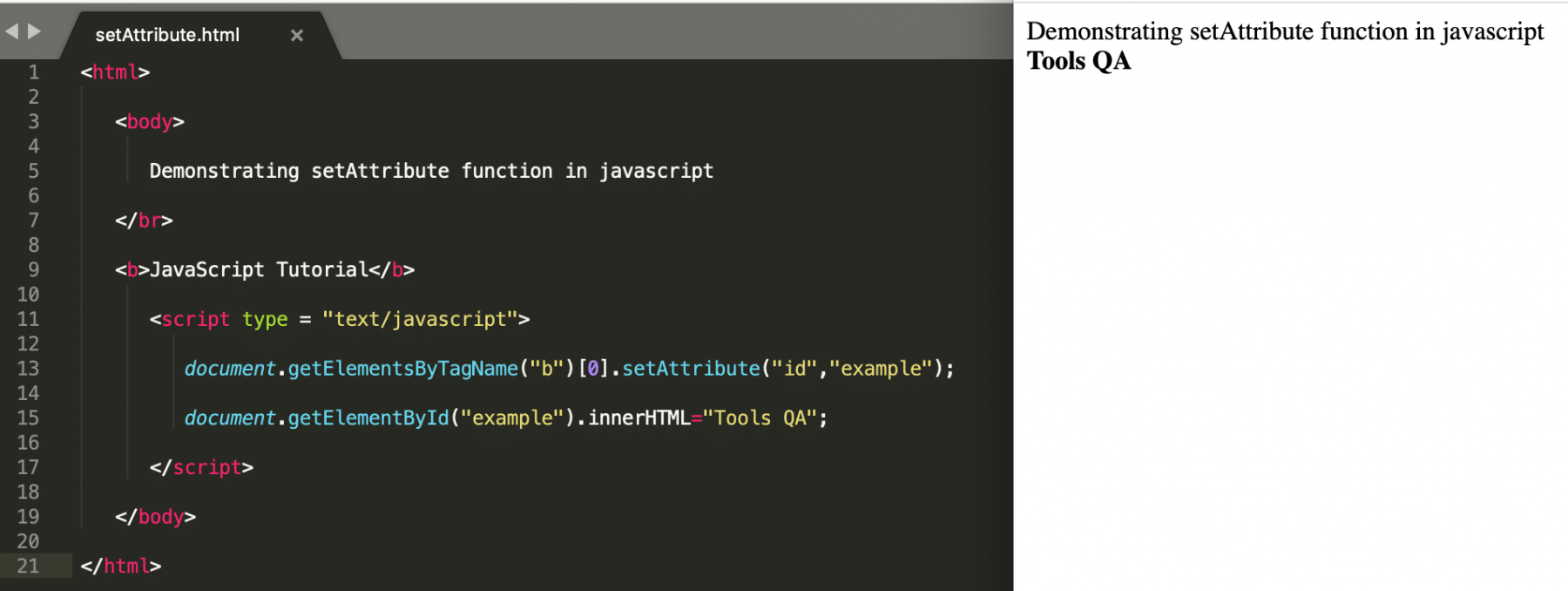
document.getElementById("example").innerHTML="Tools QA";

</script>

</body>

</html>

* Save the file with name  ***setAttribute.html***. After that, open it in any browser (Chrome, Firefox, or IE). It should show the output as:



* In the above example, the element is first added a new attribute "***id***" with value "***example***" to the *<b>* tagged HTML element and then based on id "***example***", we are manipulating the text of the element.

#### **createElement and appendChild**

This createElement() method is used to create a new element in the HTML DOM. Once the creation of element happens, it can append to a parent element using the appendChild() method. Moreover, its syntax looks like below:

***Syntax:***

// Create a new node

var node = document.createElement(tagName);

// Append the node to parent

document.parentTag.appendChild(node);

* Let's understand the usage of "***createElement()***" method with the help of following code snippet:

<html>

   <body>

      Demonstrating createElement function in javascript

   </br>

   <b>JavaScript Tutorial</b>

      <script type = "text/javascript">

         var b = document.createElement("a");

         b.innerHTML="ClickME";

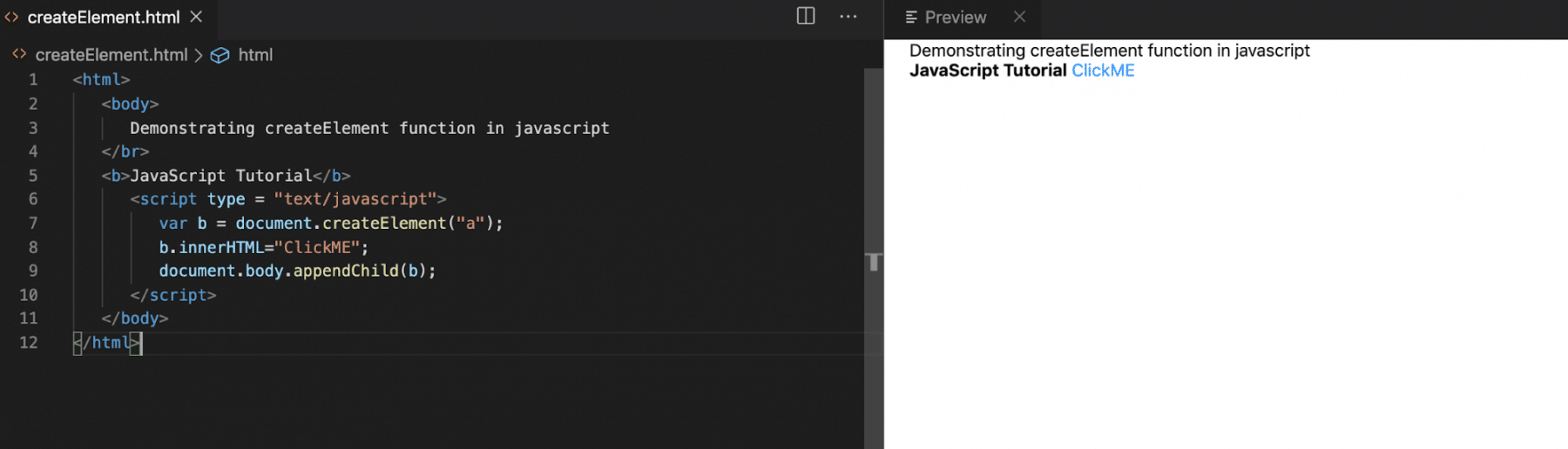
         document.body.appendChild(b);

      </script>

   </body>

</html>

* Save the file with name  ***createElement.html***. After that, open it in any browser (Chrome, Firefox, or IE). It should show the output as:



* In the above example, we created a new "***anchor***" element with the help of the "***createElement***" method and then appended the element to the document's *<body>* element. Hence, it displays on the HTML page.

#### **removeChild**

This function removes an HTML element from the document. Also, its syntax looks like below:

***Syntax:***

node.removechild(childNode);

Where,

***node***: is any web element that can be found on the web page using ***document.getElementBy<Id/tagName/className>***.

* Let's understand the usage of "***removeChild()***" method with the help of following code snippet:

<html>

   <body>

      Demonstrating removeChild function in javascript

   </br>

   <b>TOOLS QA</b>

   </br>

   <b id="demo">JavaScript Tutorial</b>

      <script type = "text/javascript">

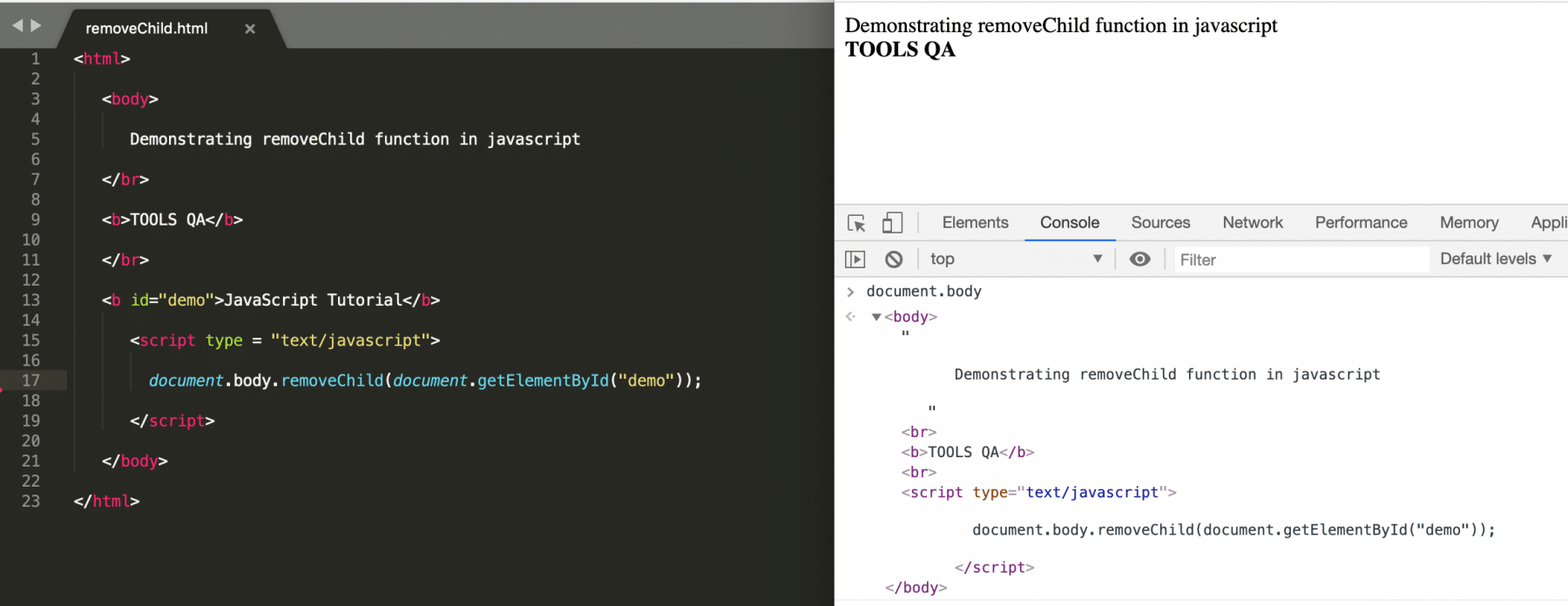
        document.body.removeChild(document.getElementById("demo"));

      </script>

   </body>

</html>

* Save the file with name  ***removeChild.html***. After this, open it in any browser (Chrome, Firefox, or IE). It should show the output as:



* In the above example, we removed the HTML element, which was having "***id***" as "***demo***", due to which the value "***JavaScript Tutorial***" could not print on the webpage. Also, as shown in the above screenshot, the same can be validated by checking the value of "***document.body***" on the Chrome console.

#### **replaceChild**

The replaceChild() method replaces a child node with a new node. The new node could be an existing node in the document, or you can create a new node. Additionally, its syntax looks like below:

***Syntax:***

node.replaceChild(newnode, oldnode);

Where,

***node***: is any web element that can be found on the web page using ***document.getElementBy<Id/tagName/className>***.

* Let's understand the usage of "***replaceChild()***" method with the help of following code snippet:

<html>

<body>

Demonstrating replaceChild function in javascript: </br>

<ul id="myList"><li>Coffee</li><li>Tea</li><li>Milk</li></ul>

<button onclick="replaceListValue()">Replace first value of list</button>

<script type = "text/javascript">

function replaceListValue() {

// Create a new list element

var newListElement = document.createElement("li");

var textNode = document.createTextNode("Water");

newListElement.appendChild(textNode);

// Replace the first element of list by the newly created element

var list = document.getElementById("myList");

list.replaceChild(newListElement, list.childNodes[0]);

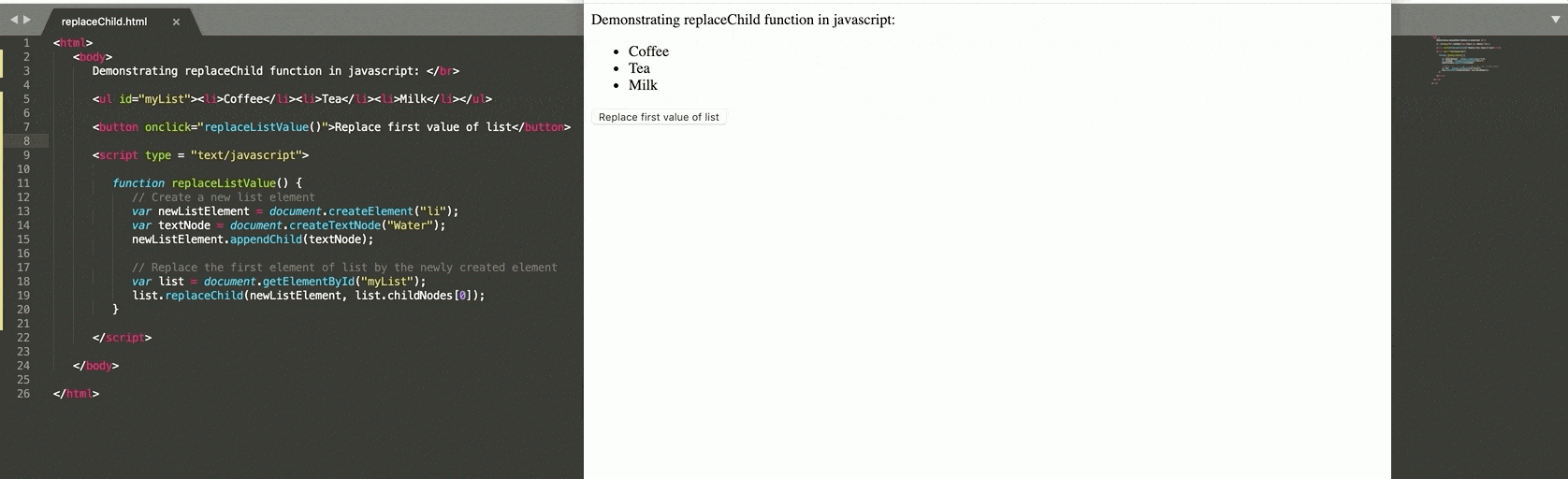
}

</script>

</body>

</html>

* Save the file with name  ***replaceChild.html***. After that, open it in any browser (Chrome, Firefox, or IE). It should show the output as:



In the above example, we replace the first element of the list on the click of the button. The replaceChild method also achieves the same.

# **JavaScript - Events**

## What is an Event ?

JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.

When the page loads, it is called an event. When the user clicks a button, that click too is an event. Other examples include events like pressing any key, closing a window, resizing a window, etc.

Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable.

Events are a part of the Document Object Model (DOM) Level 3 and every HTML element contains a set of events which can trigger JavaScript Code.

# **onload Event**

## Definition and Usage

The onload event occurs when an object has been loaded.

onload is most often used within the <body> element to execute a script once a web page has completely loaded all content (including images, script files, CSS files, etc.).

The onload event can be used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information.

The onload event can also be used to deal with cookies (see "More Examples" below).

Syntax

In HTML:

<*element* onload="*myScript*">

In JavaScript:

object.onload = function(){*myScript*};

Example

Using onload on an <img> element. Alert "Image is loaded" immediately after an image has been loaded:

<img src="w3javascript.gif" onload="loadImage()" width="100" height="132">  
  
<script>  
function loadImage() {  
  alert("Image is loaded");  
}  
</script>

# **HTML DOM MouseEvent**

## The MouseEvent Object

Events that occur when the mouse interacts with the HTML document belongs to the MouseEvent Object.

## Event Types

These event types belongs to the MouseEvent Object:

**Event Description**

onclick The event occurs when the user clicks on an element

oncontextmenu The event occurs when the user right-clicks on an element to open a context menu

ondblclick The event occurs when the user double-clicks on an element

onmousedown The event occurs when the user presses a mouse button over an element

onmouseenter The event occurs when the pointer is moved onto an element

onmouseleave The event occurs when the pointer is moved out of an element

onmousemove The event occurs when the pointer is moving while it is over an element

onmouseout The event occurs when a user moves the mouse pointer out of an element, or out of one of its children

onmouseover The event occurs when the pointer is moved onto an element, or onto one of its children

onmouseup The event occurs when a user releases a mouse button over an element

# **onclick Event**

### Example

Execute a JavaScript when a button is clicked:

<button onclick="myFunction()">Click me</button>

## Definition and Usage

The onclick event occurs when the user clicks on an element.

In HTML:

<*element* onclick="*myScript*">

In JavaScript:

object.onclick = function(){*myScript*};

More Examples

Click a <button> to display the date:

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

# **ondblclick Event**

### Example

Execute a JavaScript when a <p> element is double-clicked:

<p ondblclick="myFunction()">Double-click me</p>

**Definition and Usage**

The ondblclick event occurs when the user double-clicks on an element.

## Syntax

In HTML:

<*element* ondblclick="*myScript*">

In JavaScript:

object.ondblclick = function(){*myScript*};

# **onmousedown Event**

### Example

Execute a JavaScript when pressing a mouse button over a paragraph:

<p onmousedown="myFunction()">Click the text!</p>

## Definition and Usage

The onmousedown event occurs when a user presses a mouse button over an element.

## Syntax

In HTML:

<*element* onmousedown="*myScript*">

In JavaScript:

object.onmousedown = function(){*myScript*};

# **onmouseenter Event**

### Example

Execute a JavaScript when moving the mouse pointer onto an image:

<img onmouseenter="bigImg(this)" src="smiley.gif" alt="Smiley”>

## Definition and Usage

The onmouseenter event occurs when the mouse pointer is moved onto an element.

**Tip:** This event is often used together with the [onmouseleave](https://www.w3schools.com/jsref/event_onmouseleave.asp) event, which occurs when the mouse pointer is moved out of an element.

**Tip:** The onmouseenter event is similar to the [onmouseover](https://www.w3schools.com/jsref/event_onmouseover.asp) event. The only difference is that the onmouseenter event does not bubble (does not propagate up the document hierarchy). See "More Examples" at the bottom of this page to better understand the differences.

## Syntax

In HTML:

<*element* onmouseenter="*myScript*">

In JavaScript:

object.onmouseenter = function(){*myScript*};

# **onmouseleave Event**

### Example

Execute a JavaScript when moving the mouse pointer out of an image:

<img onmouseleave="normalImg(this)" src="smiley.gif" alt="Smiley">

## Definition and Usage

The onmouseleave event occurs when the mouse pointer is moved out of an element.

**Tip:** This event is often used together with the [onmouseenter](https://www.w3schools.com/jsref/event_onmouseenter.asp) event, which occurs when the mouse pointer is moved onto an element.

**Tip:** The onmouseleave event is similar to the [onmouseout](https://www.w3schools.com/jsref/event_onmouseout.asp) event. The only difference is that the onmouseleave event does not bubble (does not propagate up the document hierarchy). See "More Examples" at the bottom of this page to better understand the differences.

## Syntax

In HTML:

<*element* onmouseleave="*myScript*">

In JavaScript:

object.onmouseleave = function(){*myScript*};

# **onmouseup Event**

Example

Execute a JavaScript when releasing a mouse button over a paragraph:

<p onmouseup="mouseUp()">Click the text!</p>

## Definition and Usage

The onmouseup event occurs when a user releases a mouse button over an element.

## Syntax

In HTML:

<*element* onmouseup="*myScript*">

In JavaScript:

object.onmouseup = function(){*myScript*};

# **KeyboardEvent**

## The KeyboardEvent Object

Events that occur when user presses a key on the keyboard, belongs to the KeyboardEvent Object.

Event Types

These event types belongs to the KeyboardEvent Object:

Event Description

onkeydown The event occurs when the user is pressing a key

onkeypress The event occurs when the user presses a key

onkeyup The event occurs when the user releases a key

# **onkeydown Event**

### Example

Execute a JavaScript when a user is pressing a key:

<input type="text" onkeydown="myFunction()">

## Definition and Usage

The onkeydown event occurs when the user is pressing a key (on the keyboard)

## Syntax

In HTML:

<*element* onkeydown="*myScript*">

In JavaScript:

object.onkeydown = function(){*myScript*};

# **onkeypress Event**

Example

Execute a JavaScript when a user presses a key:

<input type="text" onkeypress="myFunction()">

## Definition and Usage

The onkeypress event occurs when the user presses a key (on the keyboard).

## Syntax

In HTML:

<*element* onkeypress="*myScript*">

In JavaScript:

object.onkeypress = function(){*myScript*};

# **onkeyup Event**

Example

Execute a JavaScript when a user releases a key:

<input type="text" onkeyup="myFunction()">

## Definition and Usage

The onkeyup event occurs when the user releases a key (on the keyboard).

## Syntax

In HTML:

<*element* onkeyup="*myScript*">

In JavaScript:

object.onkeyup = function(){*myScript*};

### Example

Output the actual key that was released inside a text field:

Enter your name: <input type="text" id="fname" onkeyup="myFunction()">  
  
<script>  
function myFunction() {  
  var x = document.getElementById("fname").value;  
  document.getElementById("demo").innerHTML = x;  
}  
</script>

**Form events**

Event Performed Event Handler Description

focus onfocus When the user focuses on an element

submit onsubmit When the user submits the form

blur onblur When the focus is away from a form element

change onchange When the user modifies or changes the value of a form element

# **onfocus Event Attribute**

Example

Execute a JavaScript when an input field gets focus:

<input type="text" id="fname" onfocus="myFunction(this.id)">

## Definition and Usage

The onfocus attribute fires the moment that the element gets focus.

Onfocus is most often used with <input>, <select>, and <a>.

**Tip:** The onfocus attribute is the opposite of the [onblur](https://www.w3schools.com/tags/ev_onblur.asp) attribute.

Syntax

<*element* onfocus="*script*">

# **onsubmit Event Attribute**

Example

Execute a JavaScript when a form is submitted:

<form onsubmit="myFunction()">  
  Enter name: <input type="text">  
  <input type="submit">  
</form>

## Definition and Usage

The onsubmit attribute fires when a form is submitted.

Syntax

<form onsubmit="*script*">

# **onblur Event Attribute**

Example

Validate an input field when the user leaves it:

<input type="text" name="fname" id="fname" onblur="myFunction()">

## Definition and Usage

The onblur attribute fires the moment that the element loses focus.

Onblur is most often used with form validation code (e.g. when the user leaves a form field).

**Tip:** The onblur attribute is the opposite of the [onfocus](https://www.w3schools.com/tags/ev_onfocus.asp) attribute.

Syntax

<*element* onblur="*script*">

# **onchange Event Attribute**

Example

Execute a JavaScript when a user changes the selected option of a <select> element:

<select onchange="myFunction()">

## Definition and Usage

The onchange attribute fires the moment when the value of the element is changed.

**Tip:** This event is similar to the [oninput](https://www.w3schools.com/tags/ev_oninput.asp) event. The difference is that the oninput event occurs immediately after the value of an element has changed, while onchange occurs when the element loses focus. The other difference is that the onchange event also works on <select> elements.

Syntax

<*element* onchange="*script*">

# Event bubbling in JavaScript

**Event bubbling** is a method of event propagation in the HTML DOM API when an event is in an element inside another element, and both elements have registered a handle to that event. It is a process that starts with the element that triggered the event and then bubbles up to the containing elements in the hierarchy. In event bubbling, the event is first captured and handled by the innermost element and then propagated to outer elements.

**Syntax:**

addEventListener(type, listener, useCapture)

* **type**: Use to refer to the type of event.
* **listener**: Function we want to call when the event of the specified type occurs.
* **userCapture**: Boolean value. Boolean value indicates event phase. By Default useCapture is false. It means it is in the bubbling phase.

**Example 1:** This example shows the working of event bubbling in JavaScript.

* HTML

|  |
| --- |
| <!DOCTYPE html>  <**html**>    <**head**>      <**title**>          Bubbling Event in Javascript      </**title**>  </**head**>    <**body**>        <**h2**>Bubbling Event in Javascript</**h2**>        <**div** id="parent">        <**button**>            <**h2**>Parent</**h2**>        </**button**>        <**button** id="child">    <**p**>Child</**p**>          </**button**>      </**div**><**br**>          <**script**>          document.getElementById(  "child").addEventListener("click", function () {              alert("You clicked the Child element!");          }, false);            document.getElementById(  "parent").addEventListener("click", function () {              alert("You clicked the parent element!");          }, false);      </**script**>    </**body**>    </**html**> |

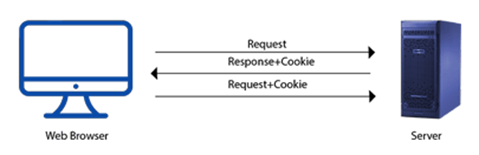
# **JavaScript Cookies**

A cookie is an amount of information that persists between a server-side and a client-side. A web browser stores this information at the time of browsing.

A cookie contains the information as a string generally in the form of a name-value pair separated by semi-colons. It maintains the state of a user and remembers the user's information among all the web pages.

## How Cookies Works?

* When a user sends a request to the server, then each of that request is treated as a new request sent by the different user.
* So, to recognize the old user, we need to add the cookie with the response from the server.
* browser at the client-side.
* Now, whenever a user sends a request to the server, the cookie is added with that request automatically. Due to the cookie, the server recognizes the users.



## How to create a Cookie in JavaScript?

In JavaScript, we can create, read, update and delete a cookie by using **document.cookie** property.

The following syntax is used to create a cookie:

1. document.cookie="name=value";

## JavaScript Cookie Example

### Example 1

Let's see an example to set and get a cookie.

1. <!DOCTYPE html**>**
2. **<html>**
3. **<head>**
4. **</head>**
5. **<body>**
6. **<input** type="button" value="setCookie" onclick="setCookie()"**>**
7. **<input** type="button" value="getCookie" onclick="getCookie()"**>**
8. **<script>**
9. function setCookie()
10. {
11. document.cookie="username=Duke Martin";
12. }
13. function getCookie()
14. {
15. if(document.cookie.length!=0)
16. {
17. alert(document.cookie);
18. }
19. else
20. {
21. alert("Cookie not available");
22. }
23. }
24. **</script>**
26. **</body>**
27. **</html>**

## Synthetic events in React

React implements a [synthetic events](https://reactjs.org/docs/events.html) system that brings consistency and high performance to React apps and interfaces. It achieves consistency by normalizing events so that they have the same properties across different browsers and platforms.

A synthetic event is a cross-browser wrapper around the browser’s native event. It has the same interface as the browser’s native event, including stopPropagation() and preventDefault(), except the events work identically across all browsers.

It achieves high performance by automatically using event delegation. In actuality, React doesn’t attach event handlers to the nodes themselves. Instead, a single event listener is attached to the root of the document. When an event is fired, React maps it to the appropriate component element.

1. Function ActionLink() {
2. function handleClick(e) {
3. e.preventDefault();
4. console.log('You had clicked a Link.');
5. }
6. **return** (
7. <a href="#" onClick={handleClick}>
8. Click\_Me
9. </a>
10. );
11. }

In the above example, e is a **Synthetic Event** which defines according to the **W3C** spec.

**Unit III**

# Multipurpose Internet Mail Extension (MIME) Protocol

**Multipurpose Internet Mail Extension (MIME)** is a standard that was proposed by Bell Communications in 1991 in order to expand the limited capabilities of email.   
MIME is a kind of add-on*or a supplementary protocol* that allows non-ASCII data to be sent through SMTP. It allows the users to exchange different kinds of data files on the Internet: audio, video, images, application programs as well.

**Why do we need MIME?**   
Limitations of Simple Mail Transfer Protocol (SMTP):

1. SMTP has a very simple structure
2. Its simplicity however comes with a price as it only sends messages in NVT 7-bit ASCII format.
3. It cannot be used for languages that do not support 7-bit ASCII format such as French, German, Russian, Chinese and Japanese, etc. so it cannot be transmitted using SMTP. So, in order *to make SMTP more broad*,*we use MIME*.
4. It cannot be used to send binary files or video or audio data.

**Purpose and Functionality of MIME –**   
Growing demand for Email Messages as people also want to express themselves in terms of Multimedia. So, MIME another email application is introduced as it is not restricted to textual data.

MIME *transforms non-ASCII data* at the sender side to NVT 7-bit data and delivers it to the client SMTP. The message on the receiver side is transferred back to the original data. As well as we can send video and audio data using MIME as it transfers them also in 7-bit ASCII data.

**Features of MIME –**

1. It is able to send multiple attachments with a single message.
2. Unlimited message length.
3. Binary attachments (executables, images, audio, or video files) may be divided if needed.
4. MIME provided support for varying content types and multi-part messages.

**Working of MIME –**   
Suppose a user wants to send an email through a user agent and it is in a non-ASCII format so there is a MIME protocol that converts it into 7-bit NVT ASCII format. The message is transferred through the e-mail system to the other side in the 7-bit format now MIME protocol again converts it back into non-ASCII code and now the user agent of the receiver side reads it and then information is finally read by the receiver. MIME header is basically inserted at the beginning of any e-mail transfer.

**MIME with SMTP and POP –**   
SMTP transfers the mail being a message transfer agent from the sender’s side to the mailbox of the receiver side and stores it and MIME header is added to the original header and provides additional information. while POP being the message access agent organizes the mails from the mail server to the receiver’s computer. POP allows the user agent to connect with the message transfer agent.

**MIME Header:**   
It is added to the original e-mail header section to define transformation. There are *five headers* that we add to the original header:

1. **MIME-Version –** Defines the version of the MIME protocol. It must have the parameter *Value 1.0*, which indicates that message is formatted using MIME.
2. **Content-Type –**Type of data used in the body of the message. They are of different types like text data (plain, HTML), audio content, or video content.
3. **Content-Type Encoding –** It defines the method used for encoding the message. Like 7-bit encoding, 8-bit encoding, etc.
4. **Content Id –** It is used for uniquely identifying the message.
5. **Content description –** It defines whether the body is actually an image, video, or audio.

# **HTTP (Hyper Text Transfer Protocol)**

The Hypertext Transfer Protocol (HTTP) is application-level protocol for collaborative, distributed, hypermedia information systems. It is the data communication protocol used to establish communication between client and server.

HTTP is TCP/IP based communication protocol, which is used to deliver the data like image files, query results, HTML files etc on the World Wide Web (WWW) with the default port is TCP 80. It provides the standardized way for computers to communicate with each other.



**The Basic Characteristics of HTTP (Hyper Text Transfer Protocol):**

* It is the protocol that allows web servers and browsers to exchange data over the web.
* It is a request response protocol.
* It uses the reliable TCP connections by default on TCP port 80.
* It is stateless means each request is considered as the new request. In other words, server doesn't recognize the user by default.

**The Basic Features of HTTP (Hyper Text Transfer Protocol):**

There are three fundamental features that make the HTTP a simple and powerful protocol used for communication:

* **HTTP is media independent:** It specifies that any type of media content can be sent by HTTP as long as both the server and the client can handle the data content.
* **HTTP is connectionless:** It is a connectionless approach in which HTTP client i.e., a browser initiates the HTTP request and after the request is sent the client disconnects from server and waits for the response.
* **HTTP is stateless:** The client and server are aware of each other during a current request only. Afterwards, both of them forget each other. Due to the stateless nature of protocol, neither the client nor the server can retain the information about different request across the web pages.

**The Basic Architecture of HTTP (Hyper Text Transfer Protocol):**

The below diagram represents the basic architecture of web application and depicts where HTTP stands:



HTTP is request/response protocol which is based on client/server based architecture. In this protocol, web browser, search engines, etc. behave as HTTP clients and the Web server like Servlet behaves as a server

# httpd is the Apache HyperText Transfer Protocol (HTTP) server program. It is designed to be run as a standalone daemon process. When used like this it will create a pool of child processes or threads to handle requests.

# **Server: Web vs. Application**

Server is a device or a computer program that accepts and responds to the request made by other program, known as client. It is used to manage the network resources and for running the program or software that provides services.

There are two types of servers:

1. Web Server
2. Application Server

**Web Server**

Web server contains only web or servlet container. It can be used for servlet, jsp, struts, jsf etc. It can't be used for EJB.

It is a computer where the web content can be stored. In general web server can be used to host the web sites but there also used some other web servers also such as FTP, email, storage, gaming etc.

Examples of Web Servers are: **Apache Tomcat**and **Resin**.

## Web Server Working

It can respond to the client request in either of the following two possible ways:

* Generating response by using the script and communicating with database.
* Sending file to the client associated with the requested URL.

The block diagram representation of Web Server is shown below:



**Important points**

* If the requested web page at the client side is not found, then web server will sends the HTTP response: Error 404 Not found.
* When the web server searching the requested page if requested page is found then it will send to the client with an HTTP response.
* If the client requests some other resources then web server will contact to application server and data is store for constructing the HTTP response.

**Application Server**

Application server contains Web and EJB containers. It can be used for servlet, jsp, struts, jsf, ejb etc. It is a component based product that lies in the middle-tier of a server centric architecture.

It provides the middleware services for state maintenance and security, along with persistence and data access. It is a type of server designed to install, operate and host associated services and applications for the IT services, end users and organizations.

The block diagram representation of Application Server is shown below:



The Example of Application Servers are:

1. **JBoss:** Open-source server from JBoss community.
2. **Glassfish:** Provided by Sun Microsystem. Now acquired by Oracle.
3. **Weblogic:** Provided by Oracle. It more secured.
4. **Websphere:** Provided by IBM.

# **HTTP Request**

HTTP Requests are messages which are sent by the client or user to initiate an action on the server.

The first line of the message includes the request message from the client to the server, the method which is applied to the resource, identifier of the resource, and the protocol version.

**Syntax**

1. Request       = Request-Line
2. \*(( general-header
3. | request-header
4. | entity-header ) CRLF)
5. CRLF
6. [ message-body ]

## Request Line

The Request-Line starts with a method token, which is followed by the Request-URI, the protocol version, and ending with CRLF. Using the SP characters, the elements are separated.

**Syntax**

1. Request-Line   = Method SP Request-URI SP HTTP-Version CRLF

**1) Method**

The method token is used to indicate the method which was performed on the resource identified by the Request-URI. The method is case sensitive.

**Syntax**

1. Method = "OPTIONS"
2. | "GET"
3. | "HEAD"
4. | "POST"
5. | "PUT"
6. | "DELETE"
7. | "TRACE"
8. | "CONNECT"
9. | extension-method
10. extension-method = token

A resource is allowed a list of methods and that methods can be specified in an Allow header field. The response's return code always notifies the client whether a method is currently allowed on a resource. Since the set of allowed methods can be changed dynamically.

**Method and Description:**

**i) GET**

This method retrieves information from the given server using a given URI. GET request can retrieve the data. It cannot apply other effects on the data.

**ii) HEAD**

The HEAD method is the same as the GET method. It is used to transfer the status line and header section only.

**iii) POST**

The POST request sends the data to the server. For example, file upload, customer information, etc. using the HTML forms.

**iv) PUT**

The PUT method is used to replace all the current representations of the target resource with the uploaded content.

**v) DELETE**

The DELETE method is used to remove all the current representations of the target resource, which is given by URI.

**vi) CONNECT**

The CONNECT method is used to establish a tunnel to the server, which is identified by a given URI.

**2) Return-URI**

The Request-URI is a Uniform Resource Identifier. It is used to identify the resource upon which to apply the request.

**Syntax**

1. Request-URI = "\*" | absoluteURI | abs\_path | authority

On the nature of the request, these four options for Request-URI depend.

a) The asterisk **"\*"** is used to show that the request does not apply to a particular resource, but it will apply to the server itself. It is allowed only when the method used does not necessarily apply to a resource.

**Example**

OPTIONS \* HTTP/1.1

b) The **absoluteURI** form is used only when the request is being made to a proxy. The requested proxy is used to forward the request and return the response.

**Example**

GET [http://www.javatpoint.com/WWW/TheProject.html](https://www.javatpoint.com/WWW/TheProject.html) HTTP/1.1

# **HTTP Response**

HTTP Response sent by a server to the client. The response is used to provide the client with the resource it requested. It is also used to inform the client that the action requested has been carried out. It can also inform the client that an error occurred in processing its request.

An HTTP response contains the following things:

1. Status Line
2. Response Header Fields or a series of HTTP headers
3. Message Body

In the request message, each HTTP header is followed by a carriage returns line feed (CRLF). After the last of the HTTP headers, an additional CRLF is used and then begins the message body.

## Status Line

In the response message, the status line is the first line. The status line contains three items:

**a) HTTP Version Number**

It is used to show the HTTP specification to which the server has tried to make the message comply.

**Example**

1. HTTP-Version = HTTP/1.1

**b) Status Code**

It is a three-digit number that indicates the result of the request. The first digit defines the class of the response. The last two digits do not have any categorization role. There are five values for the first digit, which are as follows:

**Code and Description**

**1xx: Information**

It shows that the request was received and continuing the process.

**2xx: Success**

It shows that the action was received successfully, understood, and accepted.

**3xx: Redirection**

It shows that further action must be taken to complete the request.

**4xx: Client Error**

It shows that the request contains incorrect syntax, or it cannot be fulfilled.

**5xx: Server Error**

It shows that the server failed to fulfil a valid request.

**c) Reason Phrase**

It is also known as the status text. It is a human-readable text that summarizes the meaning of the status code.

An example of the response line is as follows:

1. HTTP/1.1 200 OK

Here,

* HTTP/1.1 is the HTTP version.
* 200 is the status code.
* OK is the reason phrase.

## Response Header Fields

The HTTP Headers for the response of the server contain the information that a client can use to find out more about the response, and about the server that sent it. This information is used to assist the client with displaying the response to a user, with storing the response for the use of future, and with making further requests to the server now or in the future.

1. response-header = Accept-Ranges
2. | Age
3. | ETag
4. | Location
5. | Proxy-Authenticate
6. | Retry-After
7. | Server
8. | Vary
9. | WWW-Authenticate

The name of the Response-header field can be extended reliably only in combination with a change in the version of the protocol.

## Message Body

The response's message body may be referred to for convenience as a response body.

The body of the message is used for most responses. The exceptions are where a server is using certain status codes and where the server is responding to a client request, which asks for the headers but not the response body.

For a response to a successful request, the body of the message contains either some information about the status of the action which is requested by the client or the resource which is requested by the client. For the response to an unsuccessful request, the body of the message might provide further information about some action the client needs to take to complete the request successfully or about the reason for the error.

## Installation

The Apache web server is easy to install. With one command, you can install it and all necessary dependencies:

***$*dnf install httpd**

All the configuration files for Apache are located in **/etc/httpd/conf** and **/etc/httpd/conf.d**. The data for websites you'll run with Apache is located in **/var/www** by default, but you can change that if you want.

## Configuration

The primary Apache configuration file is **/etc/httpd/conf/httpd.conf**. It contains a lot of configuration statements that don't need to be changed for a basic installation. In fact, only a few changes must be made to this file to get a basic website up and running. The file is very large so, rather than clutter this article with a lot of unnecessary stuff, I will show only those directives that you need to change.

First, take a bit of time and browse through the **httpd.conf** file to familiarize yourself with it. One of the things I like about Red Hat versions of most configuration files is the number of comments that describe the various sections and configuration directives in the files. The **httpd.conf** file is no exception, as it is quite well commented. Use these comments to understand what the file is configuring.

The first item to change is the **Listen** statement, which defines the IP address and port on which Apache is to listen for page requests. Right now, you just need to make this website available to the local machine, so use the **localhost** address. The line should look like this when you finish:

**Listen 127.0.0.1:80**

With this directive set to the IP address of the **localhost**, Apache will listen only for connections from the local host. If you want the web server to listen for connections from remote hosts, you would use the host's external IP address.

The **DocumentRoot** directive specifies the location of the HTML files that make up the pages of the website. That line does not need to be changed because it already points to the standard location. The line should look like this:

**DocumentRoot "/var/www/html"**

The Apache installation RPM creates the **/var/www** directory tree. If you wanted to change the location where the website files are stored, this configuration item is used to do that. For example, you might want to use a different name for the **www** subdirectory to make the identification of the website more explicit. That might look like this:

**DocumentRoot "/var/mywebsite/html"**

These are the only Apache configuration changes needed to create a simple website. For this little exercise, only one change was made to the **httpd.conf** file—the **Listen** directive. Everything else is already configured to produce a working web server.

One other change is needed, however: opening port 80 in our firewall. I use [iptables](https://en.wikipedia.org/wiki/Iptables) as my firewall, so I change **/etc/sysconfig/iptables** to add a statement that allows HTTP protocol. The entire file looks like this:

**# sample configuration for iptables service  
# you can edit this manually or use system-config-firewall  
# please do not ask us to add additional ports/services to this default configuration  
\*filter  
:INPUT ACCEPT [0:0]  
:FORWARD ACCEPT [0:0]  
:OUTPUT ACCEPT [0:0]  
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT  
-A INPUT -p icmp -j ACCEPT  
-A INPUT -i lo -j ACCEPT  
-A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT  
-A INPUT -p tcp -m state --state NEW -m tcp --dport 80 -j ACCEPT  
-A INPUT -j REJECT --reject-with icmp-host-prohibited  
-A FORWARD -j REJECT --reject-with icmp-host-prohibited  
COMMIT**

The line I added is the third from the bottom, which allows incoming traffic on port 80. Now I reload the altered iptables configuration.

**[root@testvm1 ~]# cd /etc/sysconfig/ ; iptables-restore iptables**

## Create the index.html file

The **index.html** file is the default file a web server will serve up when you access the website using just the domain name and not a specific HTML file name. In the **/var/www/html** directory, create a file with the name **index.html**. Add the content **Hello World**. You do not need to add any HTML markup to make this work. The sole job of the web server is to serve up a stream of text data, and the server has no idea what the date is or how to render it. It simply transmits the data stream to the requesting host.

After saving the file, set the ownership to **apache.apache**.

**[root@testvm1 html]# chown apache.apache index.html**

## Start Apache

Apache is very easy to start. Current versions of Fedora use **systemd**. Run the following commands to start it and then to check the status of the server:

**[root@testvm1 ~]# systemctl start httpd  
[root@testvm1 ~]# systemctl status httpd  
● httpd.service - The Apache HTTP Server  
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled; vendor preset: disabled)  
   Active: active (running) since Thu 2018-02-08 13:18:54 EST; 5s ago  
     Docs: man:httpd.service(8)  
 Main PID: 27107 (httpd)  
   Status: "Processing requests..."  
    Tasks: 213 (limit: 4915)  
   CGroup: /system.slice/httpd.service  
           ├─27107 /usr/sbin/httpd -DFOREGROUND  
           ├─27108 /usr/sbin/httpd -DFOREGROUND  
           ├─27109 /usr/sbin/httpd -DFOREGROUND  
           ├─27110 /usr/sbin/httpd -DFOREGROUND  
           └─27111 /usr/sbin/httpd -DFOREGROUND  
  
Feb 08 13:18:54 testvm1 systemd[1]: Starting The Apache HTTP Server...  
Feb 08 13:18:54 testvm1 systemd[1]: Started The Apache HTTP Server.**

The commands may be different on your server. On Linux systems that use SystemV start scripts, the commands would be:

**[root@testvm1 ~]# service httpd start  
Starting httpd: [Fri Feb 09 08:18:07 2018]          [  OK  ]  
[root@testvm1 ~]# service httpd status  
httpd (pid  14649) is running...**

If you have a web browser like Firefox or Chrome on your host, you can use the URL **localhost** on the URL line of the browser to display your web page, simple as it is. You could also use a text mode web browser like [Lynx](http://lynx.browser.org/) to view the web page. First, install Lynx (if it is not already installed).

**[root@testvm1 ~]# dnf -y install lynx**

Then use the following command to display the web page.

**[root@testvm1 ~]# lynx localhost**

The result looks like this in my terminal session. I have deleted a lot of the empty space on the page.

**Hello World  
  
<snip>  
  
  
Commands: Use arrow keys to move, '?' for help, 'q' to quit, '<-' to go back.  
  Arrow keys: Up and Down to move.  Right to follow a link; Left to go back.  
 H)elp O)ptions P)rint G)o M)ain screen Q)uit /=search [delete]=history list**

Next, edit your **index.html** file and add a bit of HTML markup so it looks like this:

**<h1>Hello World</h1>**

Now refresh the browser. For Lynx, use the key combination Ctrl+R. The results look just a bit different. The text is in color, which is how Lynx displays headings if your terminal supports color, and it is now centered. In a GUI browser the text would be in a large font.

**Hello World  
  
<snip>  
  
  
Commands: Use arrow keys to move, '?' for help, 'q' to quit, '<-' to go back.  
  Arrow keys: Up and Down to move.  Right to follow a link; Left to go back.  
 H)elp O)ptions P)rint G)o M)ain screen Q)uit /=search [delete]=history list**

# **Debug Apache Errors**

Once Apache starts, it will create two log files at /opt/bitnami/apache2/logs/access\_log and /opt/bitnami/apache2/logs/error\_log respectively.

* The access\_log file is used to track client requests. When a client requests a document from the server, Apache records several parameters associated with the request in this file, such as: the IP address of the client, the document requested, the HTTP status code, and the current time.
* The error\_log file is used to record important events. This file includes error messages, startup messages, and any other significant events in the life cycle of the server. This is the first place to look when you run into a problem when using Apache.

If no error is found, you will see a message similar to:

Syntax OK

### Startup Errors

* Check the Apache error log file

Check the Apache error log file at /opt/bitnami/apache2/logs/error\_log for information about why the error occurred.

* Check if another process is listening to that port

If another process is using that address you’ll get:

(98)Address already in use: AH00072: make\_sock: could not bind to address 0.0.0.0:port\_number

no listening sockets available, shutting down

To see which process is already using that port you can run the following from a command prompt. Replace the PORT placeholder with the correct port number, such as 80 or 443.

sudo netstat -ltnp | grep PORT

In the last column you’ll see the process id or process name. You can then use:

ps aux | grep process\_name

Look for the pid in the second column and you’ll get more information about that process.

In case another process is using that port, use another port or stop that process.

* Check permissions and ownership

Check if you have permissions to bind Apache to the requested port. To bind Apache to privileged ports, start Apache as root. If you don’t have permissions to bind Apache to some port, you’ll see this error:

(13)Permission denied: AH00072: make\_sock: could not bind to address 0.0.0.0:port\_number

no listening sockets available, shutting down

If Apache is unable to open the configuration or the log file, check that the owner of those files is the same user account that installed Apache and that it has write permissions on logs and read permissions on the configuration file. If this is not the case, you will see these errors:

(13)Permission denied: AH00649: could not open transfer log file .../access\_log.

AH00015: Unable to open logs

(13)Permission denied: AH00091: httpd: could not open error log file .../error\_log.

AH00015: Unable to open logs

httpd: Could not open configuration file .../httpd.conf: Permission denied

apache config test fails, aborting

### SSL Errors

The message “Your connection to this site is only partially encrypted…” appears when you enable SSL for your site but there are some resources referenced by unencrypted HTTP URLs in your page.

To check if this is the case, view the page source and check for any http:// references. To resolve the issue, manually update your themes or templates and change any URLs to relative URLs. More specifically, instead of using http:// in your code, use //:, as below:

<img src='//example.com/img.png'/>

IMPORTANT: For security reasons, never post or disclose your server’s SSL private key in a public forum.

# What is the .htaccess File?

The .htaccess file is a powerful website file that controls high-level configuration of your website. On servers that run Apache (a web server software), the .htaccess file allows you to make changes to your website’s configuration without having to edit server configuration files.

Where is the .htaccess file located?

**Apache (the software that powers your web server) is configured so that all files named *.htaccess* are hidden.** Why? The files have important configuration information and can be used to compromise your server.

Although the file is hidden, the .htaccess file location is most commonly found in your website’s public\_html folder.

**If you don’t see your .htaccess file in your public\_html folder, you may have to check “Show Hidden Files.”**

**Common Uses of .htaccess File**

There are several use cases for the .htaccess file. The most common examples include:

* Add redirections for certain URLs
* Load custom error pages, like 404 pages
* Force your site to use HTTPS instead of HTTP
* Password-protect certain directories on your server
* Prevent hotlinking

**Unit IV**

AJAX is an acronym for **Asynchronous JavaScript and XML**. It is a group of inter-related technologies like [JavaScript](https://www.javatpoint.com/javascript-tutorial), DOM, [XML](https://www.javatpoint.com/xml-tutorial), [HTML](https://www.javatpoint.com/html-tutorial)/[XHTML](https://www.javatpoint.com/xhtml-tutorial), [CSS](https://www.javatpoint.com/css-tutorial), [XMLHttpRequest](https://www.javatpoint.com/understanding-xmlhttprequest) etc.

AJAX allows you to send and receive data asynchronously without reloading the web page. So it is fast.

AJAX allows you to send only important information to the server not the entire page. So only valuable data from the client side is routed to the server side. It makes your application interactive and faster.

What is AJAX used for?

AJAX stands for Asynchronous JavaScript And XML. In a nutshell, it is the use of the XMLHttpRequest object **to communicate with servers**. It can send and receive information in various formats, including JSON, XML, HTML, and text files.

### Where it is used?

There are too many web applications running on the web that are using ajax technology like **gmail**, **facebook**,**twitter**,**google map**, **youtube** etc.

# How to make ajax call from JavaScript ? (system calls)

**Approach 1:**In this approach, we will use the [XMLHttpRequest object](https://www.geeksforgeeks.org/ajax-full-form/) to make Ajax call. The[**XMLHttpRequest()**](https://www.geeksforgeeks.org/how-to-make-put-request-using-xmlhttprequest-by-making-custom-http-library/) method which create XMLHttpRequest object which is used to make request with server.

**Syntax:**

var xhttp = new XMLHttpRequest();

Above syntax is used to create XMLHttpRequest object. This object has many different methods which are used to interact with the server to send, receive or interrupt responses from the server. In the response, we get a string from the server that we print.

**Example:**

* Javascript

|  |
| --- |
| <script>  **function** run() {            // Creating Our XMLHttpRequest object  **var** xhr = **new** XMLHttpRequest();            // Making our connection  **var** url = '<https://jsonplaceholder.typicode.com/todos/1>';          xhr.open("GET", url, **true**);            // function execute after request is successful          xhr.onreadystatechange = **function** () {  **if** (**this**.readyState == 4 && **this**.status == 200) {                  console.log(**this**.responseText);              }          }          // Sending our request          xhr.send();      }      run();  </script> |

**Output:**

"{

"userId": 1,

"id": 1,

"title": "delectus aut autem",

"completed": false

}"

**Approach 2:**In this approach, we will use jQuery to make an ajax call. The[**ajax()**](https://www.geeksforgeeks.org/jquery-ajax-method/)method is used in jQuery to make ajax calls. It is used as a replacement for all approaches which are not working to make ajax calls.

**Syntax:**

$.ajax({arg1: value, arg2: value, ... });

**Parameter:**It takes a configuration file that configures the URL, type, function to call when we get our response or if error, etc.

**Example:**

* HTML

|  |
| --- |
| <!DOCTYPE HTML>  <**html**>    <**head**>      <**script** src=  "<https://code.jquery.com/jquery-3.6.0.min.js>">      </**script**>  </**head**>    <**body**>        <**script**>            function ajaxCall() {              $.ajax({                    // Our sample url to make request                  url:      '<https://jsonplaceholder.typicode.com/todos/1>',                    // Type of Request                  type: "GET",                    // Function to call when to                  // request is ok                  success: function (data) {                      var x = JSON.stringify(data);                      console.log(x);                  },                    // Error handling                  error: function (error) {                      console.log(`Error ${error}`);                  }              });          }          ajaxCall();      </**script**>  </**body**>    </**html**> |

**Output:**

{

"userId": 1,

"id": 1,

"title": "delectus aut autem",

"completed": false

}

**Approach 3:**In this approach, we will use [**fetch()**](https://www.geeksforgeeks.org/fetch-api/)API which is used to make XMLHttpRequest with the server. Because of its flexible structure, it is easy to use. This API makes a request to the server and gets the result as a [promise](https://www.geeksforgeeks.org/javascript-promises/) which is resolved to the string.

**Syntax:**

fetch(url, {config}).then().catch();

**Parameter:**It takes URL and config of request as parameters.

We will configure the data required and make the request to the server. Since it is a resolved promise we use [**then()**](https://www.geeksforgeeks.org/why-we-use-then-method-in-javascript/)function and **catch()** function to create output for the result. In response, we get the string that we print.

**Example:**

* Javascript

|  |
| --- |
| <script>        // Url for the request  **var** url = '<https://jsonplaceholder.typicode.com/todos/1>';        // Making our request      fetch(url, { method: 'GET' })          .then(Result => Result.json())          .then(string => {                // Printing our response              console.log(string);                // Printing our field of our response              console.log(`Title of our response :  ${string.title}`);          })          .**catch**(errorMsg => { console.log(errorMsg); });  </script> |

**Output:**

{ userId:1 ,id:1 ,title : "delectus aut autem" ,completed : false

\_\_proto\_\_:Object }

Title of our response : delectus aut autem

**Ajax file uploads**

A developer can perform an Ajax-based file upload to a server with JavaScript in five steps:

1. An HTML5 input form element must be included in the webpage that renders in the client’s browser;
2. A JavaScript method must be coded to initiate the asynchronous Ajax based file upload;
3. A component must exist on the server to handle the file upload and save the resource locally;
4. The server must send a response to the browser indicating the JavaScript file upload was successful; and
5. The client’s browser must provide an Ajax-based response indicating the file uploaded successfully.

### HTML5 file tags

HTML5 introduced a new type of input form field named file. When a browser encounters this tag, it renders a fully functional file picker on the web page. When it’s combined with an HTML5 button tag that can trigger a JavaScript method, these two elements represent the required markup elements to begin the JavaScript and Ajax file upload process.

The following HTML5 tags provide the required components to add a file selector and an upload button to any web page:

<input id="fileupload" type="file" name="fileupload" />

<button id="upload-button" onclick="uploadFile()"> Upload </button>

The button kicks off a method named uploadFile(), which contains the JavaScript file upload logic.

<script>

async function uploadFile() {

let formData = new FormData();

formData.append("file", fileupload.files[0]);

await fetch('/upload.php', {

method: "POST",

body: formData

});

alert('The file has been uploaded successfully.');

}

</script>

### JavaScript file upload logic

The above script tag contains nothing but pure JavaScript. There’s no jQuery or Dojo thrown into the mix and the logic is straightforward:

* Create a FormData object to contain the information to be sent to the server;
* Add the chosen file to be uploaded to the FormData object;
* Asynchronously call server-side resource to handle the upload; and
  + The server-side resource is invoked through the POST method
  + The server-side resource is passed the FormData which contains the file
  + In this example that server-side resource is named upload.php
* When notified that the JavaScript file upload was successful, send an Ajax based alert to the client.

All the HTML and JavaScript logic will be contained in a single file named uploader.html. The complete HTML looks as follows:

<!DOCTYPE html>

<html>

<head>

<title> Ajax JavaScript File Upload Example </title>

</head>

<body>

<!-- HTML5 Input Form Elements -->

<input id="fileupload" type="file" name="fileupload" />

<button id="upload-button" onclick="uploadFile()"> Upload </button>

<!-- Ajax JavaScript File Upload Logic -->

<script>

async function uploadFile() {

let formData = new FormData();

formData.append("file", fileupload.files[0]);

await fetch('/upload.php', {

method: "POST",

body: formData

});

alert('The file has been uploaded successfully.');

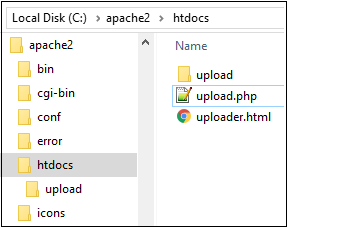
}

</script>

</body>

</html>

### Apache file upload processing

[](https://youtu.be/UWMyOleUbys)

Required JavaScript file upload components.

When an asynchronous JavaScript file upload happens, a server-side component must exist to handle the incoming file and store it. Since this example [uses an Apache HTTP Server](https://www.theserverside.com/video/Tomcat-vs-Apache-HTTP-Server-Whats-the-difference) (AHS), and since PHP is the language of AHS, it requires a file named upload.php that contains a small PHP script to save the incoming file to a folder named uploads:

<?php

/\* Get the name of the uploaded file \*/

$filename = $\_FILES['file']['name'];

/\* Choose where to save the uploaded file \*/

$location = "upload/".$filename;

/\* Save the uploaded file to the local filesystem \*/

if ( move\_uploaded\_file($\_FILES['file']['tmp\_name'], $location) ) {

echo 'Success';

} else {

echo 'Failure';

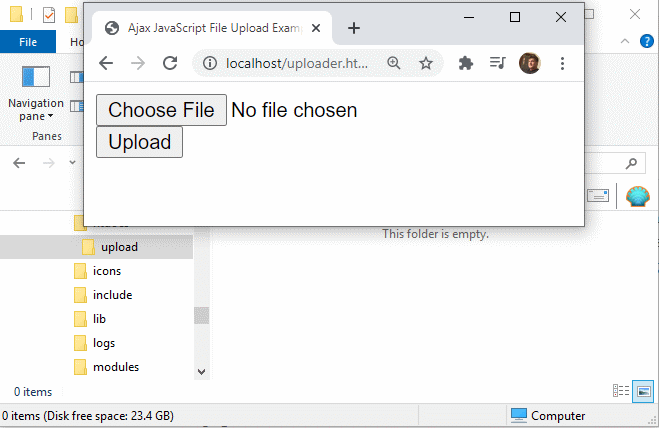
}

?>

The PHP script is also straightforward. It obtains the name of the file being uploaded, and then creates a spot in a folder named upload to save the file. PHP’s move\_uploaded\_file method is then used to save the uploaded file to this new location.

### Run the JavaScript file upload example

The files used in this example, along with a folder named upload, must be added to the htdocs folder of AHS. When a client accesses the uploader.html file through a browser, the client [will be able to](http://www.openjs.com/articles/ajax/ajax_file_upload/) upload a file to the server using Ajax and pure JavaScript.

[](https://youtu.be/EEfXGYxIegA)

A pure JavaScript file uploader simplifies Ajax based interactions with the server.

# **JavaScript Regular Expression**

A regular expression is a sequence of characters that forms a search pattern.

The search pattern can be used for text search and text replace operations.

## What Is a Regular Expression?

A regular expression is a sequence of characters that forms a **search pattern**.

When you search for data in a text, you can use this search pattern to describe what you are searching for.

A regular expression can be a single character, or a more complicated pattern.

Regular expressions can be used to perform all types of **text search** and **text replace** operations.

## Syntax

/pattern/modifiers;

### Example

/w3schools/i;

Example explained:

**/w3schools/i**  is a regular expression.

**w3schools**  is a pattern (to be used in a search).

**i**  is a modifier (modifies the search to be case-insensitive).

## Using String Methods

In JavaScript, regular expressions are often used with the two **string methods**: search() and replace().

The search() method uses an expression to search for a match, and returns the position of the match.

The replace() method returns a modified string where the pattern is replaced.

## Using String search() With a String

The search() method searches a string for a specified value and returns the position of the match:

### Example

Use a string to do a search for "W3schools" in a string:

let text = "Visit W3Schools!";  
let n = text.search("W3Schools");

The result in n will be:

6

## Using String search() With a Regular Expression

### Example

Use a regular expression to do a case-insensitive search for "w3schools" in a string:

let text = "Visit W3Schools";  
let n = text.search(/w3schools/i);

The result in n will be:

6

## Using String replace() With a String

The replace() method replaces a specified value with another value in a string:

let text = "Visit Microsoft!";  
let result = text.replace("Microsoft", "W3Schools");

## Use String replace() With a Regular Expression

### Example

Use a case insensitive regular expression to replace Microsoft with W3Schools in a string:

let text = "Visit Microsoft!";  
let result = text.replace(/microsoft/i, "W3Schools");

The result in res will be:

Visit W3Schools!

## Did You Notice?

Regular expression arguments (instead of string arguments) can be used in the methods above.  
Regular expressions can make your search much more powerful (case insensitive for example).

## Regular Expression Modifiers

**Modifiers** can be used to perform case-insensitive more global searches:

|  |  |  |
| --- | --- | --- |
| **Modifier** | **Description** | **Try it** |
| i | Perform case-insensitive matching | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_regexp_i) |
| g | Perform a global match (find all matches rather than stopping after the first match) | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_regexp_g) |
| m | Perform multiline matching | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_regexp_m) |

## Regular Expression Patterns

**Brackets** are used to find a range of characters:

|  |  |  |
| --- | --- | --- |
| **Expression** | **Description** | **Try it** |
| [abc] | Find any of the characters between the brackets | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_regexp_abc) |
| [0-9] | Find any of the digits between the brackets | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_regexp_0-9) |
| (x|y) | Find any of the alternatives separated with | | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_regexp_xy) |

**Metacharacters** are characters with a special meaning:

|  |  |  |
| --- | --- | --- |
| **Metacharacter** | **Description** | **Try it** |
| \d | Find a digit | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_regexp_d) |
| \s | Find a whitespace character | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_regexp_s) |
| \b | Find a match at the beginning of a word like this: \bWORD, or at the end of a word like this: WORD\b | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_regexp_b) [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_regexp_b2) |
| \uxxxx | Find the Unicode character specified by the hexadecimal number xxxx | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_regexp_ux) |

**Quantifiers** define quantities:

|  |  |  |
| --- | --- | --- |
| **Quantifier** | **Description** | **Try it** |
| n+ | Matches any string that contains at least one n | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_regexp_n1) |
| n\* | Matches any string that contains zero or more occurrences of n | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_regexp_n2) |
| n? | Matches any string that contains zero or one occurrences of n | [Try it »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_regexp_n3) |

## Using the RegExp Object

In JavaScript, the RegExp object is a regular expression object with predefined properties and methods.

## Using test()

The test() method is a RegExp expression method.

It searches a string for a pattern, and returns true or false, depending on the result.

The following example searches a string for the character "e":

### Example

const pattern = /e/;  
pattern.test("The best things in life are free!");

Since there is an "e" in the string, the output of the code above will be:

true

You don't have to put the regular expression in a variable first. The two lines above can be shortened to one:

/e/.test("The best things in life are free!");

## Using exec()

The exec() method is a RegExp expression method.

It searches a string for a specified pattern, and returns the found text as an object.

If no match is found, it returns an empty *(null)* object.

The following example searches a string for the character "e":

### Example

/e/.exec("The best things in life are free!");

# **JavaScript Strings**

JavaScript strings are for storing and manipulating text.

A JavaScript string is zero or more characters written inside quotes.

### Example

let text = "John Doe";

You can use single or double quotes:

### Example

let carName1 = "Volvo XC60";  // Double quotes  
let carName2 = 'Volvo XC60';  // Single quotes

You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

### Example

let answer1 = "It's alright";  
let answer2 = "He is called 'Johnny'";  
let answer3 = 'He is called "Johnny"';

## String Length

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

### Example

let text = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";  
let length = text.length;

## Escape Character

Because strings must be written within quotes, JavaScript will misunderstand this string:

let text = "We are the so-called "Vikings" from the north.";

The string will be chopped to "We are the so-called ".

The solution to avoid this problem, is to use the **backslash escape character**.

The backslash (\) escape character turns special characters into string characters:

|  |  |  |
| --- | --- | --- |
| **Code** | **Result** | **Description** |
| \' | ' | Single quote |
| \" | " | Double quote |
| \\ | \ | Backslash |

The sequence \"  inserts a double quote in a string:

### Example

let text = "We are the so-called \"Vikings\" from the north.";

The sequence \'  inserts a single quote in a string:

### Example

let text= 'It\'s alright.';

The sequence \\  inserts a backslash in a string:

### Example

let text = "The character \\ is called backslash.";

Six other escape sequences are valid in JavaScript:

|  |  |
| --- | --- |
| **Code** | **Result** |
| \b | Backspace |
| \f | Form Feed |
| \n | New Line |
| \r | Carriage Return |
| \t | Horizontal Tabulator |
| \v | Vertical Tabulator |

The 6 escape characters above were originally designed to control typewriters, teletypes, and fax machines. They do not make any sense in HTML.

## Breaking Long Code Lines

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!";

You can also break up a code line **within a text string** with a single backslash:

### Example

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

The \ method is not the preferred method. It might not have universal support.  
Some browsers do not allow spaces behind the \ character.

A safer way to break up a string, is to use string addition:

### Example

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_break_ok)

You cannot break up a code line with a backslash:

### Example

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_strings_codebreak)

## JavaScript Strings as Objects

Normally, JavaScript strings are primitive values, created from literals:

let x = "John";

But strings can also be defined as objects with the keyword new:

let y = new String("John");

### Example

let x = "John";  
let y = new String("John");

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_object)

Do not create Strings objects.

The new keyword complicates the code and slows down execution speed.

String objects can produce unexpected results:

When using the == operator, x and y are **equal**:

let x = "John";  
let y = new String("John");

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_object1)

When using the === operator, x and y are **not equal**:

let x = "John";  
let y = new String("John");

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_object2)

Note the difference between (x==y) and (x===y).

(x == y) true or false?

let x = new String("John");  
let y = new String("John");

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_object3)

(x === y) true or false?

let x = new String("John");  
let y = new String("John");

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_object4)

# **JavaScript String Methods**

## JavaScript String slice()

slice() extracts a part of a string and returns the extracted part in a new string.

The method takes 2 parameters: the start position, and the end position (end not included).

This example slices out a portion of a string from position 7 to position 12 (13-1):

### Example

let str = "Apple, Banana, Kiwi";

let part = str.slice(7, 13);

## JavaScript String substring()

substring() is similar to slice().

The difference is that substring() cannot accept negative indexes.

### Example

let str = "Apple, Banana, Kiwi";  
let part = str.substring(7, 13);

## JavaScript String substr()

substr() is similar to slice().

The difference is that the second parameter specifies the **length** of the extracted part.

### Example

let str = "Apple, Banana, Kiwi";  
let part = str.substr(7, 6);

## Replacing String Content

The replace() method replaces a specified value with another value in a string:

### Example

let text = "Please visit Microsoft!";  
let newText = text.replace("Microsoft", "W3Schools");

JavaScript Session

 Description

Basically, a **Session** is a storage that consists of information on **server-side**. JavaScript Session will be in active state till the user interacts with a website or web application.

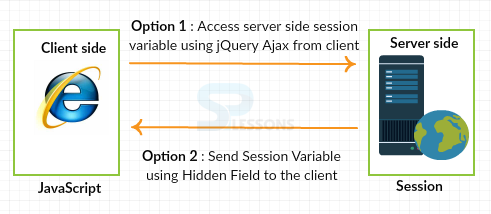
As a replacement to cookies that store constantly changing data in user’s browser, a unique identifier is stored on client side known as **Session Id**. Whenever the browser makes an HTTP request, the session id is passed to the web server every time. This session id pairs up with internal database and retrieves the variables as per the request.

Sometimes, server cannot identify the client from where it is getting requests. This happens when continuous requests and responses are received from the same client to the server. So, JavaScript Session tracking is used by providing session id’s to maintain the conversational state.

shape Conceptual

JavaScript is a client-side application and Session is a server-side application. So, there will be two options to set the JavaScript Session values.

* On the client side, query the session value using AJAX on the server.
* Sending the session value from the server to a client using HiddenField. So, JavaScript updates this value and send back to the server using submission or postback.

[](https://cdn.splessons.com/wp-content/uploads/2016/03/javascript-session-01-splessons.png)

**JavaScript Homework Help**:  
If you want someone to assist you with JavaScript assignment, let experts from [**AssignmentCore**](https://assignmentcore.com/javascript-homework/) help with your JavaScript homework.

shape Examples

Below code shows how to set the **JavaScript Session** value in PageLoad event method.

|  |
| --- |
| protected void Page\_Load(object sender, EventArgs e)    {        if (!IsPostBack)        {            //Set the session value first time            Session["EmpName"] = "Scott";            txtSessionValue.Text = Convert.ToString(Session["EmpName"]);        }    } |

**HiddenField value** is obtained when the page is submitted.

|  |
| --- |
| protected void btnSubmit\_Click(object sender, EventArgs e)    {        //Set the session value from HiddenField        Session["EmpName"] = hdnSession.Value;        txtSessionValue.Text = Convert.ToString(Session["EmpName"]);    } |

The resultant **HiddenField** value will be used in the form by setting the text box to **HiddenField valu**e.

|  |
| --- |
| <script type="text/javascript">    $(document).ready(function ()  {        $('#<% =btnSubmit.ClientID %>').click(function (e)      {            $('#<% =hdnSession.ClientID %>').attr('value', $('#<%= txtSessionValue.ClientID %>').val());        });    });    </script> |

shape Storage

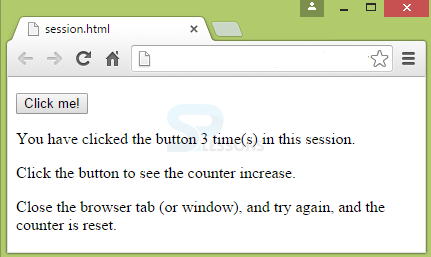
To store the data for only one session, **sessionStorage** object can be used. In sessions, the data will be deleted when the user closes the browser tab.

shape Example

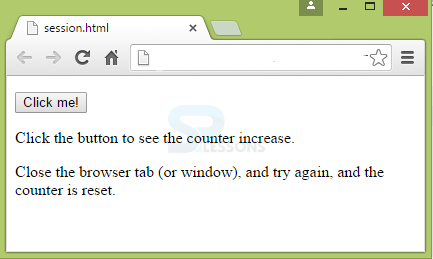
In the below example, the number of clicks on a button in the current session will be counted.

|  |
| --- |
| <!DOCTYPE html>  <html>      <head>          <script>              function clickCounter() {                  if(typeof(Storage) !== "undefined") {                      if (sessionStorage.clickcount) {                          sessionStorage.clickcount = Number(sessionStorage.clickcount)+1;                      } else {                          sessionStorage.clickcount = 1;                      }                      document.getElementById("result").innerHTML = "You have clicked the button " + sessionStorage.clickcount + " time(s) in this session.";                  } else {                      document.getElementById("result").innerHTML = "Sorry, your browser does not support web storage...";                  }              }          </script>      </head>        <body>          <p><button onclick="clickCounter()" type="button">Click me!</button></p>          <div id="result"></div>          <p>Click the button to see the counter increase.</p>          <p>Close the browser tab (or window), and try again, and the counter is reset.</p>      </body>  </html> |

**Output**:The below output appears when the button is clicked for 3 times.

[](https://cdn.splessons.com/wp-content/uploads/2016/03/javascript-session-02-splessons.png)

If the browser is re-opened, the page appear as follows.

[](https://cdn.splessons.com/wp-content/uploads/2016/03/javascript-session-03-splessons.png)

shape Key Points

* Session will run on Server-side.
* Session stores the data only till the browser is active.
* Session is stored using session.Storage object.

# **JavaScript Classes**

## JavaScript Class Syntax

Use the keyword class to create a class.

Always add a method named constructor():

### Syntax

class ClassName {  
  constructor() { ... }  
}

### Example

class Car {  
  constructor(name, year) {  
    this.name = name;  
    this.year = year;  
  }  
}

A JavaScript class is **not** an object.

It is a **template** for JavaScript objects.

## Using a Class

When you have a class, you can use the class to create objects:

### Example

let myCar1 = new Car("Ford", 2014);  
let myCar2 = new Car("Audi", 2019);

The Constructor Method

The constructor method is a special method:

* It has to have the exact name "constructor"
* It is executed automatically when a new object is created
* It is used to initialize object properties

If you do not define a constructor method, JavaScript will add an empty constructor method.

## Class Methods

Class methods are created with the same syntax as object methods.

Use the keyword class to create a class.

Always add a constructor() method.

Then add any number of methods.

### Syntax

class ClassName {  
  constructor() { ... }  
  method\_1() { ... }  
  method\_2() { ... }  
  method\_3() { ... }  
}

Create a Class method named "age", that returns the Car age:

### Example

class Car {  
  constructor(name, year) {  
    this.name = name;  
    this.year = year;  
  }  
  age() {  
    let date = new Date();  
    return date.getFullYear() - this.year;  
  }  
}  
  
let myCar = new Car("Ford", 2014);  
document.getElementById("demo").innerHTML =  
"My car is " + myCar.age() + " years old.";

# **AJAX - The XMLHttpRequest Object**

## The XMLHttpRequest Object

All modern browsers support the XMLHttpRequest object.

The XMLHttpRequest object can be used to exchange data with a server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

## Create an XMLHttpRequest Object

All modern browsers (Chrome, Firefox, Edge (and IE7+), Safari, Opera) have a built-in XMLHttpRequest object.

Syntax for creating an XMLHttpRequest object:

*variable*= new XMLHttpRequest();

### Example

var xhttp = new XMLHttpRequest();

## Access Across Domains

For security reasons, modern browsers do not allow access across domains.

This means that both the web page and the XML file it tries to load, must be located on the same server.

The examples on W3Schools all open XML files located on the W3Schools domain.

If you want to use the example above on one of your own web pages, the XML files you load must be located on your own server.

# **AJAX - Send a Request To a Server**

The XMLHttpRequest object is used to exchange data with a server.

## Send a Request To a Server

To send a request to a server, we use the open() and send() methods of the XMLHttpRequest object:

xhttp.open("GET", "ajax\_info.txt", true);  
xhttp.send();

|  |  |
| --- | --- |
| **Method** | **Description** |
| open(*method, url, async*) | Specifies the type of request  *method*: the type of request: GET or POST *url*: the server (file) location *async*: true (asynchronous) or false (synchronous) |
| send() | Sends the request to the server (used for GET) |
| send(*string*) | Sends the request to the server (used for POST) |

## GET or POST?

GET is simpler and faster than POST, and can be used in most cases.

However, always use POST requests when:

* A cached file is not an option (update a file or database on the server).
* Sending a large amount of data to the server (POST has no size limitations).
* Sending user input (which can contain unknown characters), POST is more robust and secure than GET.

## GET Requests

A simple GET request:

### Example

xhttp.open("GET", "demo\_get.asp", true);  
xhttp.send();

In the example above, you may get a cached result. To avoid this, add a unique ID to the URL:

### Example

xhttp.open("GET", "demo\_get.asp?t=" + Math.random(), true);  
xhttp.send();

If you want to send information with the GET method, add the information to the URL:

### Example

xhttp.open("GET", "demo\_get2.asp?fname=Henry&lname=Ford", true);  
xhttp.send();

## POST Requests

A simple POST request:

### Example

xhttp.open("POST", "demo\_post.asp", true);  
xhttp.send();

To POST data like an HTML form, add an HTTP header with setRequestHeader(). Specify the data you want to send in the send() method:

### Example

xhttp.open("POST", "demo\_post2.asp", true);  
xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");  
xhttp.send("fname=Henry&lname=Ford");

|  |  |
| --- | --- |
| **Method** | **Description** |
| setRequestHeader(*header, value*) | Adds HTTP headers to the request  *header*: specifies the header name *value*: specifies the header value |

## The url - A File On a Server

The url parameter of the open() method, is an address to a file on a server:

xhttp.open("GET", "ajax\_test.asp", true);

The file can be any kind of file, like .txt and .xml, or server scripting files like .asp and .php (which can perform actions on the server before sending the response back).

## Asynchronous - True or False?

Server requests should be sent asynchronously.

The async parameter of the open() method should be set to true:

xhttp.open("GET", "ajax\_test.asp", true);

By sending asynchronously, the JavaScript does not have to wait for the server response, but can instead:

* execute other scripts while waiting for server response
* deal with the response after the response is ready

**UNIT 4**

| JSON | XML |
| --- | --- |
| It is JavaScript Object Notation | It is Extensible markup language |
| It is based on JavaScript language. | It is derived from SGML. |
| It is a way of representing objects. | It is a markup language and uses tag structure to represent data items. |
| It does not provides any support for namespaces. | It supports namespaces. |
| It supports array. | It doesn’t supports array. |
| Its files are very easy to read as compared to XML. | Its documents are comparatively difficult to read and interpret. |
| It doesn’t use end tag. | It has start and end tags. |
| It is less secured. | It is more secured than JSON. |
| It doesn’t supports comments. | It supports comments. |
| It supports only UTF-8 encoding. | It supports various encoding. |