

Web Technologies (CSC 3215)

Lecture Note 12

Week 12

Sazzad Hossain

sazzad@aiub.edu

Faculty of Science and Technology

Department of Computer Science

Contents

[AJAX 3](#_Toc40311965)

[Advantages of AJAX 3](#_Toc40311966)

[How Ajax works 4](#_Toc40311967)

[Conventional method 5](#_Toc40311968)

[Ajax method 6](#_Toc40311969)

[Asynchronous vs synchronous loading 6](#_Toc40311970)

[AJAX - The XMLHttpRequest Object 7](#_Toc40311971)

[Create an XMLHttpRequest Object 7](#_Toc40311972)

[XMLHttpRequest Object Methods 7](#_Toc40311973)

[XMLHttpRequest Object Properties 9](#_Toc40311974)

[AJAX - Request 10](#_Toc40311975)

[GET or POST? 10](#_Toc40311976)

[GET Requests 10](#_Toc40311977)

[POST Requests 11](#_Toc40311978)

[Asynchronous - Request 12](#_Toc40311979)

[The onreadystatechange Property 12](#_Toc40311980)

[Synchronous Request 13](#_Toc40311981)

[AJAX - Server Response 13](#_Toc40311982)

[The onreadystatechange Property 13](#_Toc40311983)

[Server Response Properties 15](#_Toc40311984)

[Server Response Methods 15](#_Toc40311985)

[The responseText Property 15](#_Toc40311986)

[The responseXML Property 15](#_Toc40311987)

[AJAX XML Example 16](#_Toc40311988)

[AJAX PHP Example 16](#_Toc40311989)

[JSON 18](#_Toc40311990)

[JSON vs XML 18](#_Toc40311991)

[JSON is Like XML Because 19](#_Toc40311992)

[JSON is Unlike XML Because 19](#_Toc40311993)

[Why JSON is Better Than XML 19](#_Toc40311994)

[JSON.parse() 19](#_Toc40311995)

[JSON.stringify() 20](#_Toc40311996)

[AJAX Exercises 21](#_Toc40311997)

[jQuery 22](#_Toc40311998)

[Using jQuery 22](#_Toc40311999)

[jQuery Example 23](#_Toc40312000)

[jQuery Selectors 24](#_Toc40312001)

[The element Selector 24](#_Toc40312002)

[The #id Selector 25](#_Toc40312003)

[The .class Selector 26](#_Toc40312004)

[jQuery Event Methods 28](#_Toc40312005)

[jQuery Syntax For Event Methods 28](#_Toc40312006)

[Commonly Used jQuery Event Methods 29](#_Toc40312007)

[Exercises 33](#_Toc40312008)

[References 34](#_Toc40312009)

# AJAX

AJAX stands for **A**synchronous **Ja**vaScript and **X**ML. AJAX is a new technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS, and Java Script.

* Ajax uses XHTML for content, CSS for presentation, along with Document Object Model and JavaScript for dynamic content display.
* Conventional web applications transmit information to and from the sever using synchronous requests. It means you fill out a form, hit submit, and get directed to a new page with new information from the server.
* With AJAX, when you hit submit, JavaScript will make a request to the server, interpret the results, and update the current screen. In the purest sense, the user would never know that anything was even transmitted to the server.
* XML is commonly used as the format for receiving server data, although any format, including plain text, can be used.
* AJAX is a web browser technology independent of web server software.
* A user can continue to use the application while the client program requests information from the server in the background.
* Intuitive and natural user interaction. Clicking is not required, mouse movement is a sufficient event trigger.
* Data-driven as opposed to page-driven.

## Advantages of AJAX

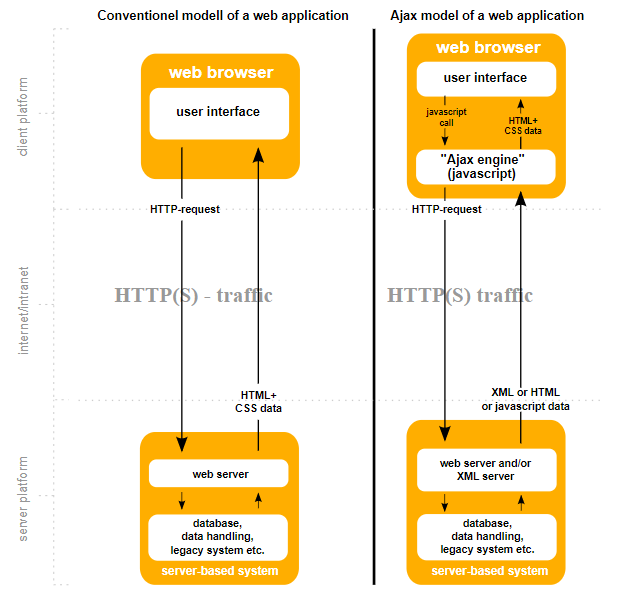
AJAX is the most viable Rich Internet Application (RIA) technology so far. It is getting tremendous industry momentum and several tool kit and frameworks are emerging. But at the same time, AJAX has browser incompatibility and it is supported by JavaScript, which is hard to maintain and debug.

AJAX is based on the following open standards −

* Browser-based presentation using HTML and Cascading Style Sheets (CSS).
* Data is stored in XML format and fetched from the server.
* Behind-the-scenes data fetches using XMLHttpRequest objects in the browser.
* JavaScript to make everything happen.

## How Ajax works

The way Ajax works is quite simple. To help explain, check out the image below which shows a comparison between the conventional method of requesting data from a web server vs using the Ajax method. I'll explain what is taking place on either side in the section below.



### Conventional method

Starting from the top of the conventional method, we can see that the browser wants to make a request to the web server for some data. therefore the following takes place.

1. An HTTP request is made from the browser to the web server. Therefore, the user must wait for this request to be processed and return a response before they can see the data requested.
2. The request reaches the web server and retrieves the appropriate data.
3. The requested data is then sent back to the browser and the user is able to see that data.

### Ajax method

The following takes place when requesting the same data, however, this time using the Ajax method.

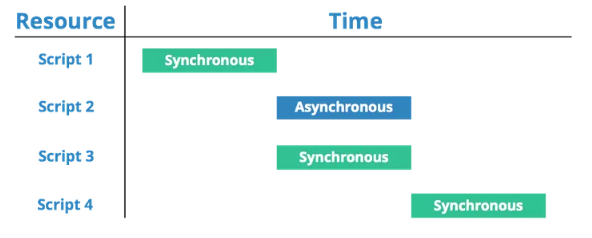
1. The browser performs a JavaScript call to the Ajax engine. In other words, create an XMLHttpRequest object.
2. In the background, an HTTP request is made to the server and the appropriate data is retrieved.
3. HTML, XML, or JavaScript data is returned to the Ajax engine which then delivers the requested data to the browser.

Using this method, the user **does not experience any downtime** from the point when they made a request to the point when they receive the actual information.

## Asynchronous vs synchronous loading

When it comes to loading scripts, traditionally scripts were loading using the synchronous method. This was more or less of a "first-come first-served" process whereas if during the page rendering process, the HTML came across a script tag then it would load that script tag in sequence, thus blocking the parsing of HTML. Once the script was fully loaded, the HTML would continue to parse and load the rest of the web page.

As we can see by the image below, there are 3 synchronous scripts being loaded one after another. However, there is also an asynchronous script which can be **loaded at the same time** as the synchronous script which makes for a much more efficient loading process.



# AJAX - 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

Syntax for creating an XMLHttpRequest object:

*variable*= new XMLHttpRequest();

**Example**

var xhttp = new XMLHttpRequest();

## XMLHttpRequest Object Methods

|  |  |
| --- | --- |
| Method | Description |
| new XMLHttpRequest() | Creates a new XMLHttpRequest object |
| abort() | Cancels the current request |
| getAllResponseHeaders() | Returns header information |
| getResponseHeader() | Returns specific header information |
| open(method,url,async,user,psw) | Specifies the request  method: the request type GET or POST url: the file location async: true (asynchronous) or false (synchronous) user: optional user name psw: optional password |
| send() | Sends the request to the server Used for GET requests |
| send(string) | Sends the request to the server. Used for POST requests |
| setRequestHeader() | Adds a label/value pair to the header to be sent |

## XMLHttpRequest Object Properties

|  |  |
| --- | --- |
| Property | Description |
| onreadystatechange | Defines a function to be called when the readyState property changes |
| readyState | Holds the status of the XMLHttpRequest. 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready |
| responseText | Returns the response data as a string |
| responseXML | Returns the response data as XML data |
| status | Returns the status-number of a request 200: "OK" 403: "Forbidden" 404: "Not Found" For a complete list go to the [Http Messages Reference](https://www.w3schools.com/tags/ref_httpmessages.asp) |
| statusText | Returns the status-text (e.g. "OK" or "Not Found") |

# AJAX - Request

XMLHttpRequest object is used to exchange data with 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

**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 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 - Request

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

## The onreadystatechange Property

The function is defined in the **onreadystatechange** property of the XMLHttpResponse object:

**Example**

xhttp.onreadystatechange = function() {  
  if (this.readyState == 4 && this.status == 200) {  
    document.getElementById("demo").innerHTML = this.responseText;  
  }  
};  
xhttp.open("GET", "ajax\_info.txt", true);  
xhttp.send();

## Synchronous Request

To execute a synchronous request, change the third parameter in the open() method to false:

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

Sometimes async = false are used for quick testing. You will also find synchronous requests in older JavaScript code.

Since the code will wait for server completion, there is no need for an onreadystatechange function:

**Example**

xhttp.open("GET", "ajax\_info.txt", false);  
xhttp.send();  
document.getElementById("demo").innerHTML = xhttp.responseText;

# AJAX - Server Response

## The onreadystatechange Property

The **readyState** property holds the status of the XMLHttpRequest. The **onreadystatechange** property defines a function to be executed when the readyState changes. The **status** property and the **statusText** property holds the status of the XMLHttpRequest object.

|  |  |
| --- | --- |
| Property | Description |
| onreadystatechange | Defines a function to be called when the readyState property changes |
| readyState | Holds the status of the XMLHttpRequest. 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready |
| status | 200: "OK" 403: "Forbidden" 404: "Page not found" For a complete list go to the [Http Messages Reference](https://www.w3schools.com/tags/ref_httpmessages.asp) |
| statusText | Returns the status-text (e.g. "OK" or "Not Found") |

The onreadystatechange function is called every time the readyState changes.

When readyState is 4 and status is 200, the response is ready:

**Example**

function loadDoc() {  
    var xhttp = new XMLHttpRequest();  
    xhttp.onreadystatechange = function() {  
        if (this.readyState == 4 && this.status == 200) {  
            document.getElementById("demo").innerHTML =  
            this.responseText;  
       }  
    };  
    xhttp.open("GET", "ajax\_info.txt", true);  
    xhttp.send();  
}

## Server Response Properties

|  |  |
| --- | --- |
| Property | Description |
| responseText | get the response data as a string |
| responseXML | get the response data as XML data |

## Server Response Methods

|  |  |
| --- | --- |
| Method | Description |
| getResponseHeader() | Returns specific header information from the server resource |
| getAllResponseHeaders() | Returns all the header information from the server resource |

## The responseText Property

The **responseText** property returns the server response as a JavaScript string, and you can use it accordingly:

**Example**

document.getElementById("demo").innerHTML = xhttp.responseText;

## The responseXML Property

The XML HttpRequest object has an in-built XML parser.

The **responseXML** property returns the server response as an XML DOM object.

Using this property you can parse the response as an XML DOM object:

**Example**

xmlDoc = xhttp.responseXML;  
txt = "";  
x = xmlDoc.getElementsByTagName("ARTIST");  
for (i = 0; i < x.length; i++) {  
  txt += x[i].childNodes[0].nodeValue + "<br>";  
  }  
document.getElementById("demo").innerHTML = txt;  
xhttp.open("GET", "cd\_catalog.xml", true);  
xhttp.send();

# AJAX XML Example

AJAX XML can be found on below link

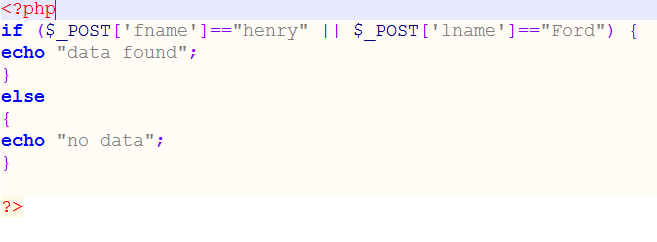
<https://www.w3schools.com/xml/ajax_xmlfile.asp>

# AJAX PHP Example

AJAX and PHP example is given below

**Ajax.php** file

**Post\_demo.php** file



# JSON

* JSON stands for **J**ava**S**cript **O**bject **N**otation
* JSON is a lightweight data-interchange format
* JSON is "self-describing" and easy to understand
* JSON is language independent

The JSON format was originally specified by Douglas Crockford. Since the JSON format is text only, it can easily be sent to and from a server, and used as a data format by any programming language.

JavaScript has a built in function to convert a string, written in JSON format, into native JavaScript objects:

JSON.parse()

So, if you receive data from a server, in JSON format, you can use it like any other JavaScript object.

## JSON vs XML

|  |  |
| --- | --- |
| JSON | XML |
| {"employees":[   { "firstName":"John", "lastName":"Doe" },   { "firstName":"Anna", "lastName":"Smith" },   { "firstName":"Peter", "lastName":"Jones" } ]} | <employees>  <employee>   <firstName>John</firstName><lastName>Doe</lastName>   </employee>   <employee>    <firstName>Anna</firstName><lastName>Smith</lastName>   </employee>   <employee>    <firstName>Peter</firstName><lastName>Jones</lastName>   </employee> </employees> |

## JSON is Like XML Because

* Both JSON and XML are "self describing" (human readable)
* Both JSON and XML are hierarchical (values within values)
* Both JSON and XML can be parsed and used by lots of programming languages
* Both JSON and XML can be fetched with an XMLHttpRequest

## JSON is Unlike XML Because

* JSON doesn't use end tag
* JSON is shorter
* JSON is quicker to read and write
* JSON can use arrays

## Why JSON is Better Than XML

For AJAX applications, JSON is faster and easier than XML:

Using XML

* Fetch an XML document
* Use the XML DOM to loop through the document
* Extract values and store in variables

Using JSON

* Fetch a JSON string
* JSON.Parse the JSON string

## JSON.parse()

A common use of JSON is to exchange data to/from a web server. When receiving data from a web server, the data is always a string. Parse the data with JSON.parse(), and the data becomes a JavaScript object.

**Example**

XMLHttpRequest to get data from the server:

var xmlhttp = new XMLHttpRequest();  
xmlhttp.onreadystatechange = function() {  
  if (this.readyState == 4 && this.status == 200) {  
    var myObj = JSON.parse(this.responseText);  
    document.getElementById("demo").innerHTML = myObj.name;  
  }  
};  
xmlhttp.open("GET", "json\_demo.txt", true);  
xmlhttp.send();

**json\_demo.txt** file

{

"name":"John",

"age":31,

"pets":[

{ "animal":"dog", "name":"Fido" },

{ "animal":"cat", "name":"Felix" },

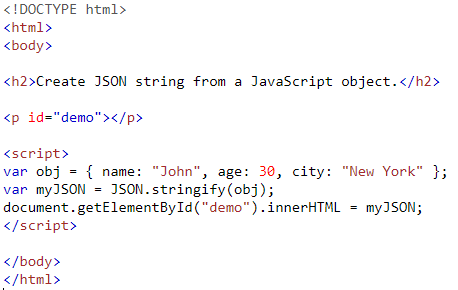
{ "animal":"hamster", "name":"Lightning" }

]

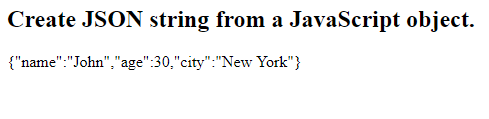
}

## JSON.stringify()

 common use of JSON is to exchange data to/from a web server. When sending data to a web server, the data has to be a string. Convert a JavaScript object into a string with JSON.stringify().



**Output:**



# AJAX Exercises

1. Create a XMLHttpRequest that will retrieve data from below XML file and display into HTML table.

**Food\_details.xml** file

<breakfast\_menu>

<food>

<name>Belgian Waffles</name>

<price>$5.95</price>

<description>

Two of our famous Belgian Waffles with plenty of real maple syrup

</description>

<calories>650</calories>

</food>

<food>

<name>Strawberry Belgian Waffles</name>

<price>$7.95</price>

<description>

Light Belgian waffles covered with strawberries and whipped cream

</description>

<calories>900</calories>

</food>

<food>

<name>French Toast</name>

<price>$4.50</price>

<description>

Thick slices made from our homemade sourdough bread

</description>

<calories>600</calories>

</food>

</breakfast\_menu>

1. Create a XMLHttpRequest that will retrieve data from below JSON file and display into HTML table.

**Employee\_data.json** file

{ "person1" : {

"name": "Morgan",

"age": "30",

"location": "Boston",

"desire": "Singing",

"fear": "Violence"

},

"person2": {

"name": "Joss",

"age": "42",

"location": "Boston",

"desire": "Hiking",

"fear": "Irrationality"

}

# jQuery

jQuery is a lightweight, "write less, do more", JavaScript library. The purpose of jQuery is to make it much easier to use JavaScript on your website. jQuery takes a lot of common tasks that require many lines of JavaScript code to accomplish, and wraps them into methods that you can call with a single line of code. jQuery also simplifies a lot of the complicated things from JavaScript, like AJAX calls and DOM manipulation.

The jQuery library contains the following features:

* HTML/DOM manipulation
* CSS manipulation
* HTML event methods
* Effects and animations
* AJAX
* Utilities

There are lots of other JavaScript libraries out there, but jQuery is probably the most popular, and also the most extendable.

Many of the biggest companies on the Web use jQuery, such as:

* Google
* Microsoft
* IBM
* Netflix

## Using jQuery

There are several ways to start using jQuery on your web site. You can:

* Download the jQuery library from jQuery.com
* Include jQuery from a CDN, like Google

There are two versions of jQuery available for downloading:

* Production version - this is for your live website because it has been minified and compressed
* Development version - this is for testing and development (uncompressed and readable code)

Both versions can be downloaded from [jQuery.com](http://jquery.com/download/).

The jQuery library is a single JavaScript file, and you reference it with the HTML <script> tag (notice that the <script> tag should be inside the <head> section):

<head>  
<script src="jquery-3.5.1.min.js"></script>  
</head>

Or it can be done using **google CDN**

<head>  
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>  
</head>

## jQuery Example

The jQuery syntax is tailor-made for **selecting** HTML elements and performing some **action** on the element(s).

Basic syntax is: **$(*selector*).*action*()**

* A $ sign to define/access jQuery
* A (*selector*) to "query (or find)" HTML elements
* A jQuery *action*() to be performed on the element(s)

Examples:

$(this).hide() - hides the current element.

$("p").hide() - hides all <p> elements.

$(".test").hide() - hides all elements with class="test".

$("#test").hide() - hides the element with id="test".

## jQuery Selectors

jQuery selectors allow you to select and manipulate HTML element(s).

jQuery selectors are used to "find" (or select) HTML elements based on their name, id, classes, types, attributes, values of attributes and much more. It's based on the existing **CSS Selectors**, and in addition, it has some own custom selectors.

All selectors in jQuery start with the dollar sign and parentheses: $().

### The element Selector

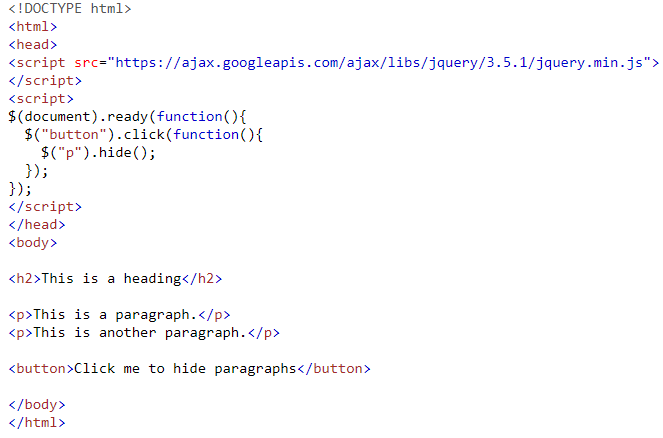
The jQuery element selector selects elements based on the element name.

You can select all <p> elements on a page like this:

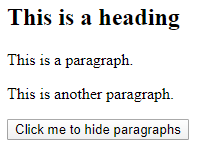
$("p")

**Example**

When a user clicks on a button, all <p> elements will be hidden:



Output



### The #id Selector

The jQuery #id selector uses the id attribute of an HTML tag to find the specific element.

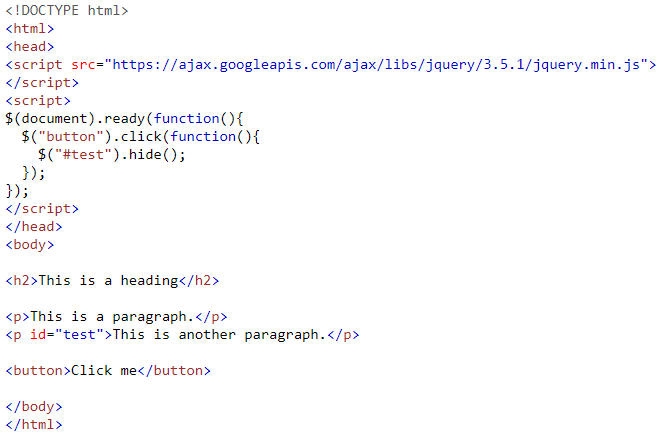
An id should be unique within a page, so you should use the #id selector when you want to find a single, unique element.

To find an element with a specific id, write a hash character, followed by the id of the HTML element:

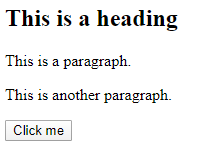
$("#test")

**Example**

When a user clicks on a button, the element with id="test" will be hidden:



Output



### The .class Selector

The jQuery .class selector finds elements with a specific class.

To find elements with a specific class, write a period character, followed by the name of the class:

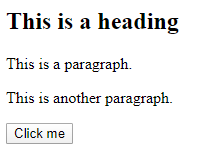
$(".test")

**Example**

When a user clicks on a button, the elements with class="test" will be hidden:



Output



## jQuery Event Methods

All the different visitors' actions that a web page can respond to are called events.

An event represents the precise moment when something happens.

Examples:

* moving a mouse over an element
* selecting a radio button
* clicking on an element

The term **"fires/fired"** is often used with events. Example: "The keypress event is fired, the moment you press a key".

Here are some common DOM events:

|  |  |  |  |
| --- | --- | --- | --- |
| Mouse Events | Keyboard Events | Form Events | Document/Window Events |
| Click | keypress | submit | load |
| dblclick | keydown | change | resize |
| mouseenter | keyup | focus | scroll |
| mouseleave |  | blur | unload |

## jQuery Syntax For Event Methods

In jQuery, most DOM events have an equivalent jQuery method.

To assign a click event to all paragraphs on a page, you can do this:

$("p").click();

The next step is to define what should happen when the event fires. You must pass a function to the event:

$("p").click(function(){  
  // action goes here!!  
});

## Commonly Used jQuery Event Methods

**$(document).ready()**

The $(document).ready() method allows us to execute a function when the document is fully loaded.

**click()**

The click() method attaches an event handler function to an HTML element.

The function is executed when the user clicks on the HTML element.

The following example says: When a click event fires on a <p> element; hide the current <p> element:

**Example**

$("p").click(function(){  
  $(this).hide();  
});

**dblclick()**

The dblclick() method attaches an event handler function to an HTML element.

The function is executed when the user double-clicks on the HTML element:

**Example**

$("p").dblclick(function(){  
  $(this).hide();  
});

**mouseenter()**

The mouseenter() method attaches an event handler function to an HTML element.

The function is executed when the mouse pointer enters the HTML element:

**Example**

$("#p1").mouseenter(function(){  
  alert("You entered p1!");  
});

**mouseleave()**

The mouseleave() method attaches an event handler function to an HTML element.

The function is executed when the mouse pointer leaves the HTML element:

**Example**

$("#p1").mouseleave(function(){  
  alert("Bye! You now leave p1!");  
});

**mousedown()**

The mousedown() method attaches an event handler function to an HTML element.

The function is executed, when the left, middle or right mouse button is pressed down, while the mouse is over the HTML element:

**Example**

$("#p1").mousedown(function(){  
  alert("Mouse down over p1!");  
});

**mouseup()**

The mouseup() method attaches an event handler function to an HTML element.

The function is executed, when the left, middle or right mouse button is released, while the mouse is over the HTML element:

**Example**

$("#p1").mouseup(function(){  
  alert("Mouse up over p1!");  
});

**hover()**

The hover() method takes two functions and is a combination of the mouseenter() and mouseleave() methods.

The first function is executed when the mouse enters the HTML element, and the second function is executed when the mouse leaves the HTML element:

**Example**

$("#p1").hover(function(){  
  alert("You entered p1!");  
},  
function(){  
  alert("Bye! You now leave p1!");  
});

**focus()**

The focus() method attaches an event handler function to an HTML form field.

The function is executed when the form field gets focus:

**Example**

$("input").focus(function(){  
  $(this).css("background-color", "#cccccc");  
});

**blur**()

The blur() method attaches an event handler function to an HTML form field.

The function is executed when the form field loses focus:

**Example**

$("input").blur(function(){  
  $(this).css("background-color", "#ffffff");  
});

**The on() Method**

The on() method attaches one or more event handlers for the selected elements.

Attach a click event to a <p> element:

**Example**

$("p").on("click", function(){  
  $(this).hide();  
});

Attach multiple event handlers to a <p> element:

**Example**

$("p").on({  
  mouseenter: function(){  
    $(this).css("background-color", "lightgray");  
  },  
  mouseleave: function(){  
    $(this).css("background-color", "lightblue");  
  },  
  click: function(){  
    $(this).css("background-color", "yellow");  
  }  
});

## Exercises

**jQuery Selectors**

Create a HTML file apply following;

* 1. Use the correct selector to hide all <p> elements.

$("").hide();

2. Use the correct selector to hide an element with id="test".

$("").hide();

3. Use the correct selector to hide all elements with class="test".

$("").hide();

**jQuery Event**

Create a HTML file apply following;

1. Use the correct event to hide all <p> elements with a "click".

$("p").(function(){

$(this).hide();

});

2. Use the correct event to hide all <p> elements with a "double-click".

$("p").(function(){

$(this).hide();

});

3. Use the on() method to attach a click event handler to all <p> elements.

$("").on(, function(){

$(this).hide();

});

# References

1. [www.tutorialspoint.com](http://www.tutorialspoint.com)

2. [www.keycdn.com/support/ajax-programming](http://www.keycdn.com/support/ajax-programming)

3. [www.w3schools.com](https://www.w3schools.com/)

used when performing queries.

Closing the MySQL Database Server Connection

The connection to the MySQL database server will be closed automatically as soon as the execution of

the script ends. However, if you want to close it earlier you can do this by simply calling the PHP

mysqli\_close() function.