

JQUERY - OVERVIEW

What is jQuery?

jQuery is a fast and concise JavaScript Library created by John Resig in 2006 with a nice motto – **Write less, do more.**

jQuery simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development.

jQuery is a JavaScript toolkit designed to simplify various tasks by writing less code. Here is the list of important core features supported by jQuery –

- **DOM manipulation** – The jQuery made it easy to select DOM elements, traverse them and modifying their content by using cross-browser open source selector engine called **Sizzle**.
- **Event handling** – The jQuery offers an elegant way to capture a wide variety of events, such as a user clicking on a link, without the need to clutter the HTML code itself with event handlers.
- **AJAX Support** – The jQuery helps you a lot to develop a responsive and feature-rich site using AJAX technology.
- **Animations** – The jQuery comes with plenty of built-in animation effects which you can use in your websites.
- **Lightweight** – The jQuery is very lightweight library - about 19KB in size Minified and gzipped.
- **Cross Browser Support** – The jQuery has cross-browser support, and works well in IE 6.0+, FF 2.0+, Safari 3.0+, Chrome and Opera 9.0+
- **Latest Technology** – The jQuery supports CSS3 selectors and basic XPath syntax.

How to use jQuery?

There are two ways to use jQuery.

- **Local Installation** – You can download jQuery library on your local machine and include it in your HTML code.
- **CDN Based Version** – You can include jQuery library into your HTML code directly from Content Delivery Network CDNCN.

Local Installation

- Go to the <https://jquery.com/download/> to download the latest version available.
- Now put downloaded **jquery-2.1.3.min.js** file in a directory of your website, e.g. /jquery.

Example

Now you can include *jquery* library in your HTML file as follows –

```
<html>

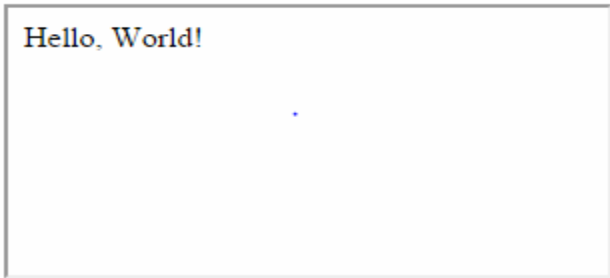
  <head>
    <title>The jQuery Example</title>
    <script type = "text/javascript"      src = "/jquery/jquery-
2.1.3.min.js"></script>

    <script type = "text/javascript">
      $(document).ready(function(){
        document.write("Hello, World!");
      });
    </script>
  </head>

  <body>
    <h1>Hello</h1>
  </body>

</html>
```

This will produce following result –



Hello, World!

CDN Based Version

You can include jQuery library into your HTML code directly from Content Delivery Network CDNCN. Google and Microsoft provides content deliver for the latest version.

We are using Google CDN version of the library throughout this tutorial.

Example

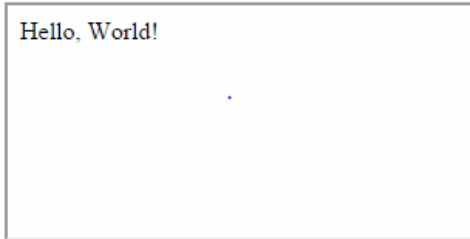
Now let us rewrite above example using jQuery library from Google CDN.

```
<html>
  <head>
    <title>The jQuery Example</title>
    <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"><
  /script>

    <script type = "text/javascript">
      $(document).ready(function(){
        document.write("Hello, World!");
      });
    </script>
  </head>

  <body>
    <h1>Hello</h1>
  </body>
</html>
```

This will produce following result –



How to call a jQuery library functions?

As almost everything we do when using jQuery reads or manipulates the document object model DOM, we need to make sure that we start adding events etc. as soon as the DOM is ready.

If you want an event to work on your page, you should call it inside the `$(document).ready()` function. Everything inside it will load as soon as the DOM is loaded and before the page contents are loaded.

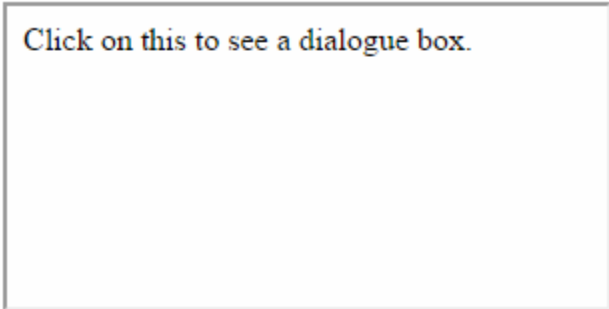
To do this, we register a ready event for the document as follows –

```
$(document).ready(function() {  
    // do stuff when DOM is ready  
});
```

To call upon any jQuery library function, use HTML script tags as shown below –

```
<html>  
  
  <head>  
    <title>The jQuery Example</title>  
    <script type = "text/javascript"  
      src  
      "http://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></scr  
ipt>  
  
    <script type = "text/javascript" language = "javascript">  
      $(document).ready(function() {  
        $("div").click(function() {alert("Hello, world!");});  
      });  
    </script>  
  </head>  
  
  <body>  
    <div id = "mydiv">  
      Click on this to see a dialogue box.  
    </div>  
  </body>  
  
</html>
```

This will produce following result –



Click on this to see a dialogue box.

How to use Custom Scripts?

It is better to write our custom code in the custom JavaScript file : **custom.js**, as follows –

```
/* Filename: custom.js */
$(document).ready(function() {

    $("div").click(function() {
        alert("Hello, world!");
    });

});
```

Now we can include **custom.js** file in our HTML file as follows –

```
<html>

    <head>
        <title>The jQuery Example</title>
        <script type = "text/javascript"
            src
            "http://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></scr
            ipt>

        <script type = "text/javascript" src =
            "/jquery/custom.js"></script>
    </head>

    <body>
        <div id = "mydiv">
            Click on this to see a dialogue box.
        </div></body></html>
```

This will produce following result –

Click on this to see a dialogue box.

Using Multiple Libraries

You can use multiple libraries all together without conflicting each others. For example you can use jQuery and MooTool javascript libraries together.

You can check [jQuery noConflict](#) Method for more detail.

JQUERY - BASICS

jQuery is a framework built using JavaScript capabilities. So while developing your applications using jQuery, you can use all the functions and other capabilities available in JavaScript.

This chapter would explain most basic concepts but frequently used in jQuery based applications.

String

A string in JavaScript is an immutable object that contains none, one or many characters.

Following are the valid examples of a JavaScript String –

```
"This is JavaScript String"  
'This is JavaScript String'  
'This is "really" a JavaScript String'  
"This is 'really' a JavaScript String"
```

Numbers

Numbers in JavaScript are double-precision 64-bit format IEEE 754 values. They are immutable, just as strings.

Following are the valid examples of a JavaScript Numbers –

```
5350  
120.27  
0.26
```

Boolean

A boolean in JavaScript can be either **true** or **false**. If a number is zero, it defaults to false. If an empty string defaults to false – Following are the

valid examples of a JavaScript Boolean –

```
true    // true  
false   // false  
0       // false  
1       // true  
""      // false  
"hello" // true
```

Objects

JavaScript supports Object concept very well. You can create an object using the object literal as follows –

```
var emp = {  
  name: "Zara",  
  age: 10  
};
```

You can write and read properties of an object using the dot notation as follows –

```
// Getting object properties  
emp.name // ==> Zara  
emp.age  // ==> 10  
  
// Setting object properties  
emp.name = "Daisy" // <== Daisy  
emp.age  = 20      // <== 20
```

Arrays

You can define arrays using the array literal as follows –

```
var x = [];  
var y = [1, 2, 3, 4, 5];
```

An array has a **length** property that is useful for iteration –

```
var x = [1, 2, 3, 4, 5];  
  
for (var i = 0; i < x.length; i++) {  
  // Do something with x[i]  
}
```

Functions

A function in JavaScript can be either named or anonymous. A named function can be defined using *function* keyword as follows –

```
function named(){  
  // do some stuff here  
}
```

An anonymous function can be defined in similar way as a normal function but it would not have any name.

A anonymous function can be assigned to a variable or passed to a method as shown below.

```
var handler = function () {  
  // do some stuff here
```



```
}
```

JQuery makes a use of anonymous functions very frequently as follows –

```
$(document).ready(function(){  
    // do some stuff here  
});
```

Arguments

JavaScript variable *arguments* is a kind of array which has *length* property. Following example explains it very well –

```
function func(x){  
    console.log(typeof x, arguments.length);  
}  
  
func();           //==> "undefined", 0  
func(1);          //==> "number", 1  
func("1", "2", "3"); //==> "string", 3
```

The arguments object also has a *callee* property, which refers to the function you're inside of. For example –

```
function func() {  
    return arguments.callee;  
}  
  
func();           // ==> func
```

Context

JavaScript famous keyword **this** always refers to the current context. Within a function **this** context can change, depending on how the function is called –

```
$(document).ready(function() {  
    // this refers to window.document  
});  
  
$("div").click(function() {  
    // this refers to a div DOM element  
});
```

You can specify the context for a function call using the function-built-in methods **call** and **apply** methods.

The difference between them is how they pass arguments. Call passes all arguments through as arguments to the function, while apply accepts an array as the arguments.

```
function scope() {  
  console.log(this, arguments.length);  
}  
  
scope() // window, 0  
scope.call("foobar", [1,2]); //==> "foobar", 1  
scope.apply("foobar", [1,2]); //==> "foobar", 2
```

Scope

The scope of a variable is the region of your program in which it is defined. JavaScript variable will have only two scopes.

- **Global Variables** – A global variable has global scope which means it is defined everywhere in your JavaScript code.
- **Local Variables** – A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

Within the body of a function, a local variable takes precedence over a global variable with the same name –

```
var myVar = "global"; // ==> Declare a global variable  
  
function () {  
  var myVar = "local"; // ==> Declare a local variable  
  document.write(myVar); // ==> local  
}
```

Callback

A callback is a plain JavaScript function passed to some method as an argument or option. Some callbacks are just events, called to give the user a chance to react when a certain state is triggered.

jQuery's event system uses such callbacks everywhere for example –

```
$("body").click(function(event) {  
  console.log("clicked: " + event.target);  
});
```

Most callbacks provide arguments and a context. In the event-handler example, the callback is called with one argument, an Event.

Some callbacks are required to return something, others make that return value optional. To prevent a form submission, a submit event handler can return false as follows –

```
$("#myform").submit(function() {  
    return false;  
});
```

Closures

Closures are created whenever a variable that is defined outside the current scope is accessed from within some inner scope.

Following example shows how the variable **counter** is visible within the create, increment, and print functions, but not outside of them –

```
function create() {  
    var counter = 0;  
  
    return {  
        increment: function() {  
            counter++;  
        },  
  
        print: function() {  
            console.log(counter);  
        }  
    }  
}  
  
var c = create();  
c.increment();  
c.print();    // ==> 1
```

This pattern allows you to create objects with methods that operate on data that isn't visible to the outside world. It should be noted that **data hiding** is the very basis of object-oriented programming.

Proxy Pattern

A proxy is an object that can be used to control access to another object. It implements the same interface as this other object and passes on any method invocations to it. This other object is often called the real subject.

A proxy can be instantiated in place of this real subject and allow it to be accessed remotely. We can save jQuery's `setArray` method in a closure and overwrites it as follows –

```
(function() {
```

```
// log all calls to setArray
var proxied = jQuery.fn.setArray;

jQuery.fn.setArray = function() {
  console.log(this, arguments);
  return proxied.apply(this, arguments);
};

})();
```

The above wraps its code in a function to hide the *proxied* variable.

The proxy then logs all calls to the method and delegates the call to the original method. Using *apply*, *this*, *arguments* guarantees that the caller won't be able to notice the difference between the original and the proxied method.

Built-in Functions

JavaScript comes along with a useful set of built-in functions. These methods can be used to manipulate Strings, Numbers and Dates.

Following are important JavaScript functions –

S.N.	Method & Description
1	charAt Returns the character at the specified index.
2	Concat Combines the text of two strings and returns a new string.
3	forEach Calls a function for each element in the array.
4	indexOf Returns the index within the calling String object of the first occurrence of the specified value, or -1 if not found.
5	Length

	Returns the length of the string.
6	Pop Removes the last element from an array and returns that element.
7	Push Adds one or more elements to the end of an array and returns the new length of the array.
8	Reverse Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first.
9	Sort Sorts the elements of an array.
10	Substr Returns the characters in a string beginning at the specified location through the specified number of characters.
11	toLowerCase Returns the calling string value converted to lower case.
12	toString Returns the string representation of the number's value.
13	toUpperCase Returns the calling string value converted to uppercase.

The Document Object Model

The Document Object Model is a tree structure of various elements of HTML as follows –

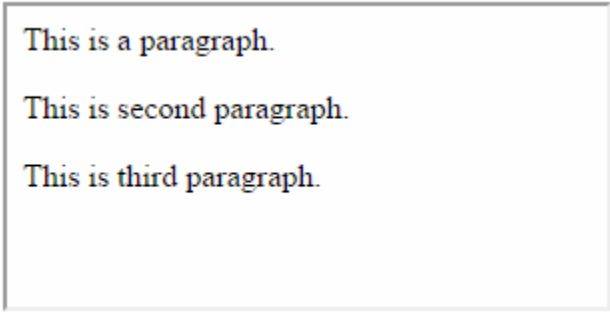
```
<html>

  <head>
    <title>The jQuery Example</title>
  </head>

  <body>
    <div>
      <p>This is a paragraph.</p>
      <p>This is second paragraph.</p>
      <p>This is third paragraph.</p>
    </div>
  </body>

</html>
```

This will produce following result –



This is a paragraph.

This is second paragraph.

This is third paragraph.

Following are the important points about the above tree structure –

- The <html> is the ancestor of all the other elements; in other words, all the other elements are descendants of <html>.
- The <head> and <body> elements are not only descendants, but children of <html>, as well.
- Likewise, in addition to being the ancestor of <head> and <body>, <html> is also their parent.
- The <p> elements are children and descendant sand descendants of <div>, descendants of <body> and <html>, and siblings of each other <p> elements.