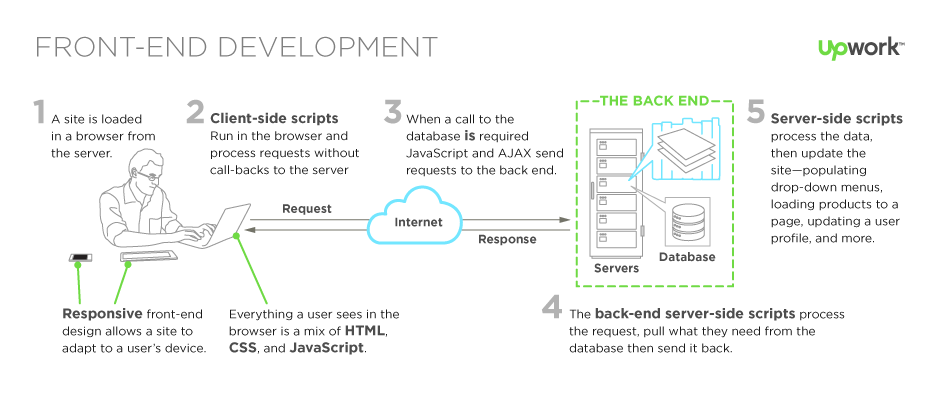
1. **Client Side Scripting 8 hours**

**Java Script: Introduction, Operator, Control, DOM, Array, Object, Smart Form, Class and objects, jQuery: Using jQuery, element finder, events and animations.**



Everything you see, click, and interact with on a website is the work of front-end web development. **Client-side frameworks and scripting languages like JavaScript** and AngularJS have made interactive websites possible. Here’s a look at how this technology works in the scheme of a website, and some of the most popular scripts and frameworks you should know.

All websites run on three components: **the server, the database, and the client.** The client is simply the browser a person is using to view a site, and it’s where client-side technology is unpacked and processed. The server is at a remote location anywhere in the world—housing data, running a site’s back-end architecture, processing requests, and sending pages to the browser. The client is anywhere your users are viewing your site: mobile devices, laptops, or desktop computers. Server-side scripting is executed by a web server; client-side scripting is executed by a browser.

Client-end scripts are embedded in a website’s HTML markup code, which is housed on the server in a language that’s compatible with, or compiled to communicate with, the browser. The browser temporarily downloads that code, and then, apart from the server, processes it. If it needs to request additional information in response to user clicks, mouse-overs, etc. (called “events”), a request is sent back to the server.

Client-side scripting is always evolving—it’s growing simpler, more nimble, and easier to use. As a result, sites are faster, more efficient, and less work is left up to the server.

**HOW DOES CLIENT-SIDE SCRIPTING WORK?**

There is overlap between the two technologies as they work in tandem, but there are core differences. Server-side scripting works in the back end of a site, which the user doesn’t see. It creates a scaffolding for the site to access its database, all the behind-the-scenes mechanics that organize and power a website. Client-side code, however, handles what the user does see.

1. Scripts are embedded within and interact with the HTML of your site, selecting elements of it, then manipulating those elements to provide an interactive experience.
2. Scripts interact with a cascading style sheet (CSS) file that styles the way the page looks.
3. It dictates what work the server-side code is going to have to accomplish (where utility should be built around these front-end functions), and returns data that’s pulled from the site in a way that’s readable by the browser. For example: If there’s a form for updating a profile, the back end is built to pull specific data from the database to populate that form, while front-end scripts populate the form with that information.
4. Scripts put less stress on the server because they don’t require processing on the server once they’re downloaded, just when post-backs are made. “Post-backs” perform specific call-and-answers with the server-side code, and respond to the user immediately.

**CLIENT-SIDE PROGRAMMING LANGUAGES & FRAMEWORKS**

Now that you’ve got a broad view of what front-end technology is and does, here’s a look at some of the most widely used scripting languages and front-end frameworks. Languages are almost always used in the context of their frameworks, which make quick work of complicated code with libraries of pre-packaged, shareable code, and lots of add-ons. Your developer may use one or a combination of these when building the front end of your site.

1. **HTML and CSS:** These are the core building blocks of any site. HTML dictates a site’s organization and content. CSS comprises the code for every graphic element—from backgrounds to fonts—that make up the look and feel of a website. Learn more about HTML and markup languages.
2. **JavaScript:** JavaScript is client-side scripting. The most widely used client-side script—nearly every site’s front end is a combination of JavaScript and HTML and CSS. JavaScript is fueled by an array of excellent frameworks that simplify it and give it more agility.

**JavaScript Frameworks:**

1. **AngularJS:** An incredibly robust JavaScript framework for data-heavy sites
2. **JQuery, jQuery Mobile:** A fast, small, JS object library that streamlines how JavaScript behaves across different browsers
3. **Node.js:** A server-side platform that uses JavaScript, and is changing the way real-time applications can communicate with the server for faster response times and a more seamless user experience. It works with another JavaScript framework, Express.js, to build server-side Web applications.
4. **Bootstrap:** A mobile-first framework that uses HTML, CSS, and JavaScript to facilitate rapid responsive app development
5. **React,** for user interface design
6. Express.js, Backbone.js, Ember.js, MeteorJS, and more
7. **TypeScript:** A compile-to-JavaScript language that is a superset of JavaScript, created by Microsoft
8. **AJAX (JavaScript + XML)—**a technology that allows specific parts of a site to be updated without a full-page refresh by asynchronously connecting to the database and pulling JSON– or XML-based chunks of data.
9. VBScript & JScript are Microsoft’s front-end scripts that run on the ASP.NET framework. JScript is Microsoft’s reverse-engineered version of JavaScript.
10. ActionScript, which creates animated interactive web applications for Adobe Flash Play
11. Java (as “applets”) snippets of back-end code that run independently with a run-time environment in the browser

**Tip:**

It’s worth researching what browsers your primary audience is most likely to use, and what back-end scripts and APIs you’re using, then decide on a script based on a stack or compatibility.

**CLIENT-SIDE SCRIPTING BREAKTHROUGH**

An important breakthrough that changed the hard-and-fast rules for client side vs. server side? AJAX. The old standard was that server-side processing and page post-backs were used when the browser needed to interact with things on the server, like databases. AJAX, with its asynchronous calls to the server, can pull the data instantly and efficiently, without requiring a post-back.

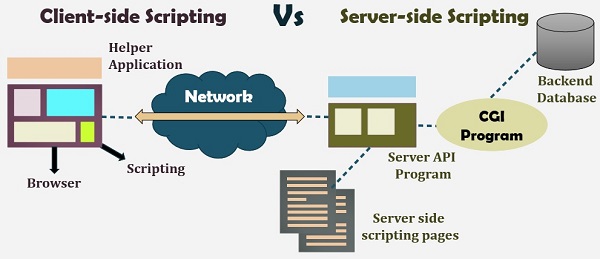
Another major boost is jQuery, a fast, small, and feature-rich JavaScript library with an easy-to-use API that works across a multitude of browsers. Like code libraries do, jQuery changed the way that millions of people write JavaScript, simplifying a number of other client-side scripts like AJAX at the same time.

Learn more about client-side scripting in the context of The Role of the Front-End Web Developer.

Upwork is a freelancing marketplace where businesses of all sizes can find talented professionals across multiple disciplines and categories. If you are a business and are looking to get projects done, consider signing up!

<https://techdifferences.com/difference-between-server-side-scripting-and-client-side-scripting.html>

**Differences between Client-side and Server-side Scripting**



The scripts can be written in two forms, at the server end (back end) or at the client end (server end). The main difference between server-side scripting and client-side scripting is that the server side scripting involves server for its processing. On the other hand, client-side scripting requires browsers to run the scripts on the client machine but does not interact with the server while processing the client-side scripts.

A script is generally a series of program or instruction, which has to be executed on other program or application. As we know that the web works in a client-server environment. The client-side script executes the code to the client side which is visible to the users while a server-side script is executed in the server end which users cannot see.

**Content: Server-side Scripting Vs Client-side Scripting**

1. Comparison Chart
2. Definition
3. Key Differences
4. Conclusion

### Comparison Chart

| **BASIS FOR COMPARISON** | **SERVER-SIDE SCRIPTING** | **CLIENT-SIDE SCRIPTING** |
| --- | --- | --- |
| Basic | Works in the back end which could not be visible at the client end. | Works at the front end and script are visible among the users. |
| Processing | Requires server interaction. | Does not need interaction with the server. |
| Languages involved | PHP, ASP.net, Ruby on Rails, ColdFusion, Python, etcetera. | HTML, CSS, JavaScript, etc. |
| Affect | Could effectively customize the web pages and provide dynamic websites. | Can reduce the load to the server. |
| Security | Relatively secure. | Insecure |

### Definition of Server-side Scripting

**Server-side scripting** is a technique of programming for producing the code which can run software on the server side, in simple words any scripting or programming that can run on the web server is known as server-side scripting. The operations like customization of a website, dynamic change in the website content, response generation to the user’s queries, accessing the database, and so on are performed at the server end.

The server-side scripting constructs a communication link between a server and a client (user). Earlier the server side scripting is implemented by the **CGI (Common Gateway Interface)** scripts. The CGI was devised to execute the scripts from programming languages such as C++ or Perl on the websites.

The server-side involves three parts: server, database, API’s and back-end web software developed by the server-side scripting language. When a browser sends a request to the server for a webpage consisting of server-side scripting, the web server processes the script prior to serving the page to the browser. Here the processing of a script could include extracting information from a database, making simple calculations, or choosing the appropriate content that is to be displayed in the client end. The script is being processed and the output is sent to the browser. The web server abstracts the scripts from the end user until serving the content, which makes the data and source code more secure.

#### **Server-side scripting languages:**

After the advent of CGI, multiple programming languages were evolved such as PHP, Python, Ruby, ColdFusion, C#, Java, C++ and so on for server-side scripting among which some of them are described below:

**PHP:** It is the most prevalent server-side language used on the web which was designed to extract and manipulate information in the database. The language is used in association with SQL language for the Database. It is used in Facebook, WordPress and Wikipedia.

**Python:** The language is fast and contains shorter code. It is good for beginners as it concentrates on the readability and simplicity of the code. Python functions well in the object-oriented environment and used in famous sites like Youtube, Google, etc.

**Ruby:** It contains complex logic which packages the back-end with database utility which can also be provided by PHP and SQL.

### Definition of Client-side Scripting

**Client-side scripting** is performed to generate a code that can run on the client end (browser) without needing the server side processing. Basically, these types of scripts are placed inside an HTML document. The client-side scripting can be used to examine the user’s form for the errors before submitting it and for changing the content according to the user input. As I mentioned before, the web requires three elements for its functioning which are, client, database and server.

The effective client-side scripting can significantly reduce the **server load**. It is designed to run as a scripting language utilizing a web browser as a host program. For example, when a user makes a request via browser for a webpage to the server, it just sent the HTML and CSS as plain text, and the browser interprets and renders the web content in the client end.

#### **Client-side scripting languages:**

**HTML**: It is the fundamental building blocks of web programming which provides the frame to the website. It describes the arrangement of the content.

**CSS**: CSS provides the way to design the graphic elements which help in making the appearance of the web application more attractive.

**JavaScript**: It is also a client-side scripting language which essentially devised for the specific purpose, but currently there are various JavaScript frameworks used as server-side scripting technology.

## Key Differences Between Server-side Scripting and Client-side Scripting

1. Server-side scripting is used at the backend, where the source code is not viewable or hidden at the client side (browser). On the other hand, client-side scripting is used at the front end which users can see from the browser.
2. When a server-side script is processed it communicates to the server. As against, client-side scripting does not need any server interaction.
3. The client-side scripting language involves languages such as HTML, CSS and JavaScript. In contrast, programming languages such as PHP, ASP.net, Ruby, ColdFusion, Python, C#, Java, C++, etc.
4. Server-side scripting is useful in customizing the web pages and implement the dynamic changes in the websites. Conversely, the client-side script can effectively minimize the load to the server.
5. Server-side scripting is more secure than client-side scripting as the server side scripts are usually hidden from the client end, while a client-side script is visible to the users.

### 

### Conclusion

Client-side scripting and server-side scripting works in a coordinated manner with each other. However, both the scripting techniques are very different, where the client-side scripting emphasize on making the interface of the web application or website more appealing and functional. Conversely, server-side scripting emphasizes on the data accessing methods, error handling and fast processing etcetera.

[**Java Script: Introduction, Operator, Control, DOM, Array, Object, Smart Form, Class and**](UNIT%20II%20JAVASCRIPT.docx) **objects,**

**Smart Form:** <http://tacomachirocare.com/Smart%20Forms/documentation/>

**jQuery: Using jQuery, element finder, events and animations**

**What is jQuery?**

<https://www.w3schools.com/jquery/jquery_selectors.asp>

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, negotiate 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 featurerich 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 (CDN).

**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 −

## CDN Based Version

You can include jQuery library into your HTML code directly from Content Delivery Network (CDN). 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 = "https://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 = "https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js">

</script>

<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 −

## 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 = "https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js">

</script>

<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 −

## 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](https://www.tutorialspoint.com/jquery/jquery-noconflict.htm) Method for more detail.

**JQuery Selectors**

The jQuery library harnesses the power of Cascading Style Sheets (CSS) selectors to let us quickly and easily access elements or groups of elements in the Document Object Model (DOM).

A jQuery Selector is a function which makes use of expressions to find out matching elements from a DOM based on the given criteria. Simply you can say, selectors are used to select one or more HTML elements using jQuery. Once an element is selected then we can perform various operations on that selected element.

## The $() factory function

jQuery selectors start with the dollar sign and parentheses − **$()**. The factory function **$()** makes use of following three building blocks while selecting elements in a given document −

|  |  |
| --- | --- |
| **Sr.No.** | **Selector & Description** |
| 1 | **Tag Name**  Represents a tag name available in the DOM. For example **$('p')**selects all paragraphs <p> in the document. |
| 2 | **Tag ID**  Represents a tag available with the given ID in the DOM. For example **$('#some-id')** selects the single element in the document that has an ID of some-id. |
| 3 | **Tag Class**  Represents a tag available with the given class in the DOM. For example **$('.some-class')** selects all elements in the document that have a class of some-class. |

All the above items can be used either on their own or in combination with other selectors. All the jQuery selectors are based on the same principle except some tweaking.

**NOTE** − The factory function **$()** is a synonym of **jQuery()** function. So in case you are using any other JavaScript library where **$** sign is conflicting with some thing else then you can replace **$** sign by **jQuery** name and you can use function **jQuery()** instead of **$()**.

### Example

Following is a simple example which makes use of Tag Selector. This would select all the elements with a tag name **p**.

<html>

<head>

<title>The jQuery Example</title>

<script type = "text/javascript"

src = "https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js">

</script>

<script type = "text/javascript" language = "javascript">

$(document).ready(function() {

$("p").css("background-color", "yellow");

});

</script>

</head>

<body>

<div>

<p class = "myclass">This is a paragraph.</p>

<p id = "myid">This is second paragraph.</p>

<p>This is third paragraph.</p>

</div>

</body>

</html>

This will produce following result −

## How to Use Selectors?

The selectors are very useful and would be required at every step while using jQuery. They get the exact element that you want from your HTML document.

Following table lists down few basic selectors and explains them with examples.

|  |  |
| --- | --- |
| **Sr.No.** | **Selector & Description** |
| 1 | [**Name**](https://www.tutorialspoint.com/jquery/selector-element-name.htm)  Selects all elements which match with the given element **Name**. |
| 2 | [**#ID**](https://www.tutorialspoint.com/jquery/selector-element-id.htm)  Selects a single element which matches with the given **ID**. |
| 3 | [**.Class**](https://www.tutorialspoint.com/jquery/selector-element-class.htm)  Selects all elements which match with the given **Class**. |
| 4 | [**Universal (\*)**](https://www.tutorialspoint.com/jquery/selector-universal.htm)  Selects all elements available in a DOM. |
| 5 | [**Multiple Elements E, F, G**](https://www.tutorialspoint.com/jquery/selector-multiple-elements.htm)  Selects the combined results of all the specified selectors **E, F** or **G**. |

## Selectors Examples

Similar to above syntax and examples, following examples would give you understanding on using different type of other useful selectors −

Here, you have different type of other useful selectors −

You can use all the above selectors with any HTML/XML element in generic way. For example if selector **$("li:first")** works for <li> element then **$("p:first")** would also work for <p> element.

**Element finder, events and animations**

**jQuery - Events Handling**

What are Events?

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 |

We have the ability to create dynamic web pages by using events. Events are actions that can be detected by your Web Application.

Following are the examples events −

* A mouse click
* A web page loading
* Taking mouse over an element
* Submitting an HTML form
* A keystroke on your keyboard, etc.

When these events are triggered, you can then use a custom function to do pretty much whatever you want with the event. These custom functions call Event Handlers.

## Binding Event Handlers

Using the jQuery Event Model, we can establish event handlers on DOM elements with the **bind()** method as follows −

<html>

<head>

<title>The jQuery Example</title>

<script type = "text/javascript"

src = "https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js">

</script>

<script type = "text/javascript" language = "javascript">

$(document).ready(function() {

$('div').bind('click', function( event ){

alert('Hi there!');

});

});

</script>

<style>

.div{ margin:10px;padding:12px; border:2px solid #666; width:60px;}

</style>

</head>

<body>

<p>Click on any square below to see the result:</p>

<div class = "div" style = "background-color:blue;">ONE</div>

<div class = "div" style = "background-color:green;">TWO</div>

<div class = "div" style = "background-color:red;">THREE</div>

</body>

</html>

This code will cause the division element to respond to the click event; when a user clicks inside this division thereafter, the alert will be shown.

This will produce following result −

The full syntax of the bind() command is as follows −

*selector*.bind( eventType[, eventData], handler)

Following is the description of the parameters −

* **eventType** − A string containing a JavaScript event type, such as click or submit. Refer to the next section for a complete list of event types.
* **eventData** − This is optional parameter is a map of data that will be passed to the event handler.
* **handler** − A function to execute each time the event is triggered.

## Removing Event Handlers

Typically, once an event handler is established, it remains in effect for the remainder of the life of the page. There may be a need when you would like to remove event handler.

jQuery provides the **unbind()** command to remove an exiting event handler. The syntax of unbind() is as follows −

*selector*.unbind(eventType, handler)

or

*selector*.unbind(eventType)

Following is the description of the parameters −

* **eventType** − A string containing a JavaScript event type, such as click or submit. Refer to the next section for a complete list of event types.
* **handler** − If provided, identifies the specific listener that's to be removed.

## Event Types

The following are cross platform and recommended event types which you can bind using JQuery −

## The Event Object

The callback function takes a single parameter; when the handler is called the JavaScript event object will be passed through it.

The event object is often unnecessary and the parameter is omitted, as sufficient context is usually available when the handler is bound to know exactly what needs to be done when the handler is triggered, however there are certain attributes which you would need to be accessed.

## The Event Attributes

The following event properties/attributes are available and safe to access in a platform independent manner −

<html>

<head>

<title>The jQuery Example</title>

<script type = "text/javascript"

src = "https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js">

</script>

<script type = "text/javascript" language = "javascript">

$(document).ready(function() {

$('div').bind('click', function( event ){

alert('Event type is ' + event.type);

alert('pageX : ' + event.pageX);

alert('pageY : ' + event.pageY);

alert('Target : ' + event.target.innerHTML);

});

});

</script>

<style>

.div{ margin:10px;padding:12px; border:2px solid #666; width:60px;}

</style>

</head>

<body>

<p>Click on any square below to see the result:</p>

<div class = "div" style = "background-color:blue;">ONE</div>

<div class = "div" style = "background-color:green;">TWO</div>

<div class = "div" style = "background-color:red;">THREE</div>

</body>

</html>

This will produce following result −

## The Event Methods

There is a list of methods which can be called on an Event Object −

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | [**preventDefault()**](https://www.tutorialspoint.com/jquery/events-preventdefault.htm)  Prevents the browser from executing the default action. |
| 2 | [**isDefaultPrevented()**](https://www.tutorialspoint.com/jquery/events-isdefaultprevented.htm)  Returns whether event.preventDefault() was ever called on this event object. |
| 3 | [**stopPropagation()**](https://www.tutorialspoint.com/jquery/events-stoppropagation.htm)  Stops the bubbling of an event to parent elements, preventing any parent handlers from being notified of the event. |
| 4 | [**isPropagationStopped()**](https://www.tutorialspoint.com/jquery/events-ispropagationstopped.htm)  Returns whether event.stopPropagation() was ever called on this event object. |
| 5 | [**stopImmediatePropagation()**](https://www.tutorialspoint.com/jquery/events-stopimmediatepropagation.htm)  Stops the rest of the handlers from being executed. |
| 6 | [**isImmediatePropagationStopped()**](https://www.tutorialspoint.com/jquery/events-isimmediatepropagationstopped.htm)  Returns whether event.stopImmediatePropagation() was ever called on this event object. |

## Event Manipulation Methods

Following table lists down important event-related methods −

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | [**bind( type, [data], fn )**](https://www.tutorialspoint.com/jquery/events-bind.htm)  Binds a handler to one or more events (like click) for each matched element. Can also bind custom events. |
| 2 | [**off( events [, selector ] [, handler(eventObject) ] )**](https://www.tutorialspoint.com/jquery/events-off.htm)  This does the opposite of live, it removes a bound live event. |
| 3 | [**hover( over, out )**](https://www.tutorialspoint.com/jquery/events-hover.htm)  Simulates hovering for example moving the mouse on, and off, an object. |
| 4 | [**on( events [, selector ] [, data ], handler )**](https://www.tutorialspoint.com/jquery/events-on.htm)  Binds a handler to an event (like click) for all current − and future − matched element. Can also bind custom events. |
| 5 | [**one( type, [data], fn )**](https://www.tutorialspoint.com/jquery/events-one.htm)  Binds a handler to one or more events to be executed once for each matched element. |
| 6 | [**ready( fn )**](https://www.tutorialspoint.com/jquery/events-ready.htm)  Binds a function to be executed whenever the DOM is ready to be traversed and manipulated. |
| 7 | [**trigger( event, [data] )**](https://www.tutorialspoint.com/jquery/events-trigger.htm)  Trigger an event on every matched element. |
| 8 | [**triggerHandler( event, [data] )**](https://www.tutorialspoint.com/jquery/events-triggerhandler.htm)  Triggers all bound event handlers on an element. |
| 9 | [**unbind( [type], [fn] )**](https://www.tutorialspoint.com/jquery/events-unbind.htm)  This does the opposite of bind, it removes bound events from each of the matched elements. |

## Event Helper Methods

jQuery also provides a set of event helper functions which can be used either to trigger an event to bind any event types mentioned above.

## Trigger Methods

Following is an example which would triggers the blur event on all paragraphs −

$("p").blur();

## Binding Methods

Following is an example which would bind a **click** event on all the <div> −

$("div").click( function () {

// do something here

});