
MOOTOOLS
tutorialspoint
SIMPLYEASYLEARNING

About the Tutorial

The full form of MooTools is My Object-Oriented Tools. It is an object-oriented, lightweight JavaScript framework. It is released under the free, open-source MIT License. It is one of the most popular JavaScript libraries.

In this tutorial, we will walk through MooTools and its features.

Audience

This tutorial is designed for software professionals who are willing to learn MooTools (a JavaScript library) in simple and easy steps. This tutorial will give you a great understanding on various MooTools concepts.

Prerequisites

We assume that the reader has prior knowledge of HTML coding. It would help if the reader has had an exposure to object-oriented programming concepts and a general idea on creating online applications.

Disclaimer & Copyright

Table of Contents

1. MOOTOOLS – INTRODUCTION

MooTools is an object-oriented, lightweight JavaScript framework. The full form of MooTools is My Object-Oriented Tools. It is released under the free, open-source MIT License. It is one of most popular JavaScript libraries.

MooTools is a powerful, lightweight JavaScript library. It creates an easy interaction of JavaScript in web development. It can also do a lot of things as CSS extensions. MooTools has all sorts of nifty extensions, which gives you the ability to create animated effects.

Components of MooTools

MooTools includes a number of components. The following are the different component categories:

- **Core** — A collection of utility functions that all the other components require.
- **More** — An official collection of add-ons that extend the core and provide enhanced functionality.
- **Class** — The base library for class object instantiation.
- **Natives** — A collection of JavaScript native object enhancements. The natives add functionality, compatibility, and new methods that simplify coding.
- **Element** — Contains a large number of enhancements and compatibility standardization to the HTML Element Object.
- **FX** — An Advanced effects-API that helps to animate page elements.
- **Request** — Includes XHR interface, Cookie JSON, and HTML retrieval-specific tools for developers to exploit.
- **Window** — Provides a cross-browser interface to client-specific information, such as the dimensions of the window.

MooTools – Advantages

MooTools come with a number of advantages over native JavaScript. These advantages include the following:

- MooTools is an extensive and modular framework that allows developers to create their own customized combination of components.

- MooTools follows object-oriented paradigm and the DRY principle (Don't Repeat Yourself).

- MooTools provides advanced component effects, with optimized transitions. It is mostly used for flash developers.
- MooTools provides different enhancements to the DOM. This helps the developers to add, modify, select, and delete DOM elements. And, it also supports storing and retrieving element storage.

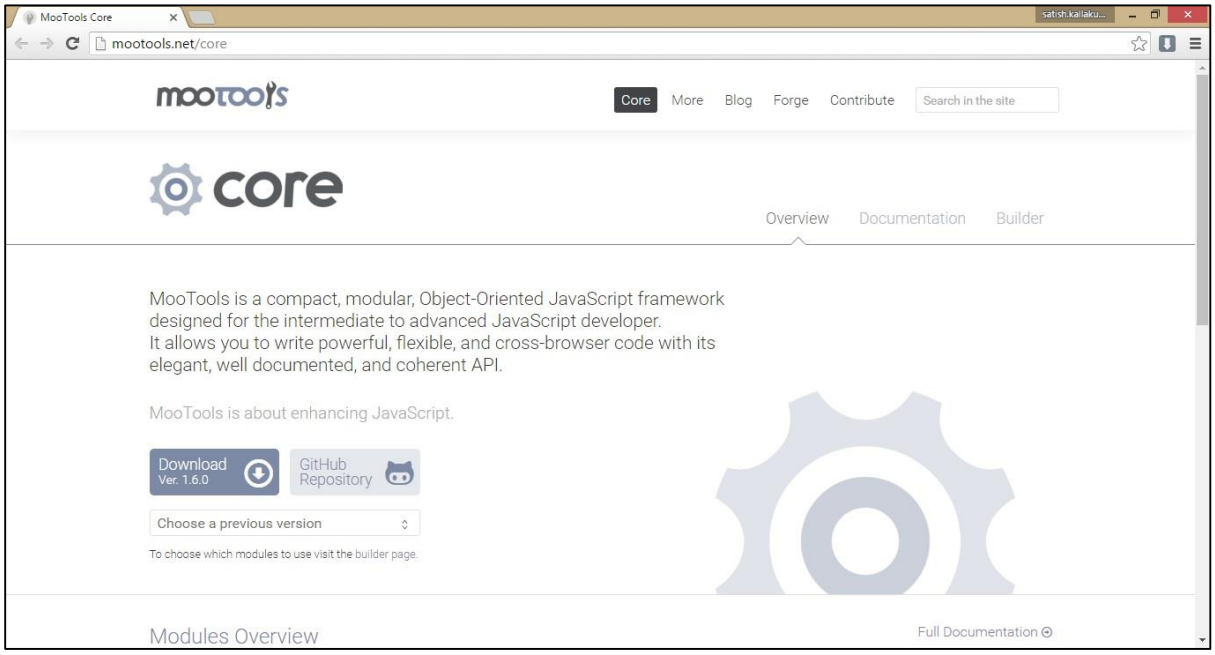
2. MOOTOOLS – INSTALLATION

MooTools is a powerful, JavaScript library to design DOM objects using object-oriented paradigm. This chapter explains how to install and use MooTools library along with JavaScript.

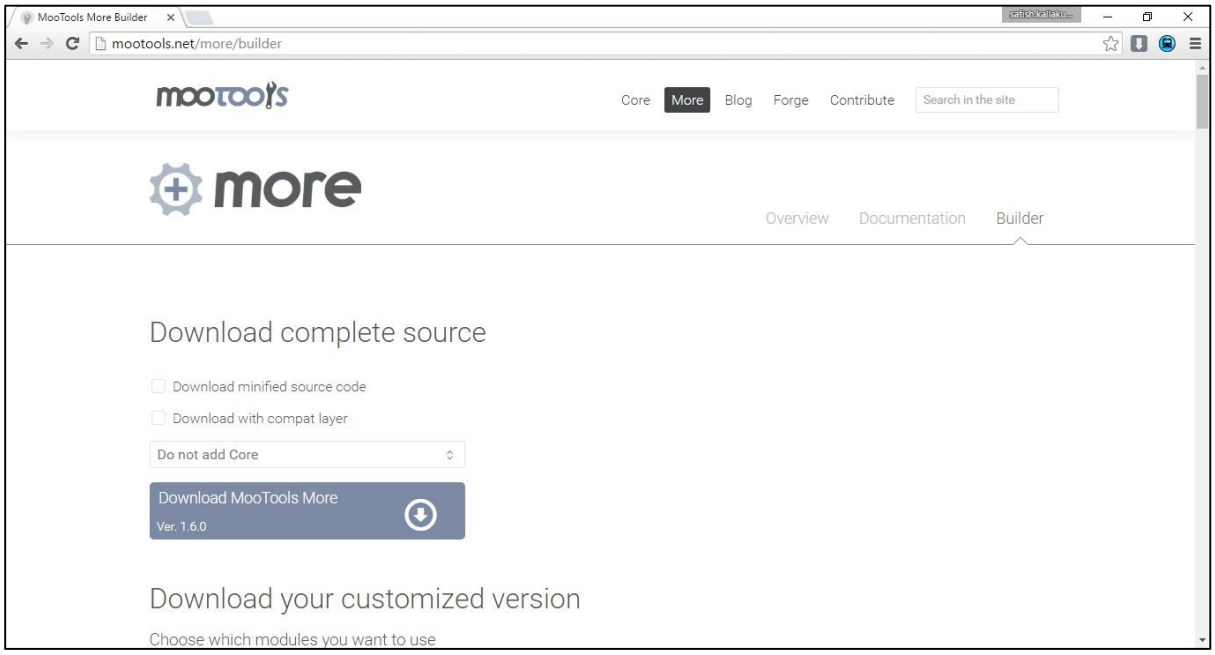
To install MooTools library, follow the steps given below:

Step 1: Download MooTools Core and MooTools More library

You can download the latest version of MooTools Core and MooTools More libraries from the following link MooTools-Core and MooTools-More. When you click on the links, you will be directed to the following screens in your browser:



And,



Click on the download buttons, you will get the latest version of MooTools libraries. For this tutorial, we are using **MooTools-Core-1.6.0.js** and **MooTools-More-1.6.0.js** libraries.

Step 2: Upload the MooTools Core and More libraries into the server

You now have the MooTools libraries in your file system. We have to copy these libraries into the server (the workspace) where the application web pages are available. For this tutorial, we are using **C:\MooTools\workspace** directory location.

Therefore, copy the **MooTools-Core-1.6.0.js** and **MooTools-Core-1.6.0.js** files into the given directory location.

Step 3: Link the MooTools Core and More libraries into the script tag

The JavaScript library is a **.js** file. If you include this library into your JavaScript code, include it with the script tag as follows. Take a look at the following code snippet.

```
<script type = "text/javascript" src = "MooTools-Core-1.6.0.js"></script>
```

```
<script type = "text/javascript" src = "MooTools-More-1.6.0.js"></script>
```

3. MOOTOOLS – PROGRAM STRUCTURE

MooTools is a tool which can be used to design object-oriented models. Let us discuss in this chapter a simple example of MooTools library.

Example

Here we will design a model named Rectangle using Class. For this, we need to declare the properties — Width and Height.

Take a look at the following code, and save it into sample.html.

```
<html>
<head>
<script type = "text/javascript" src = "MooTools-Core-1.6.0.js"></script>
<script type = "text/javascript" src = "MooTools-More-1.6.0.js"></script>
<script type = "text/javaScript">
var Rectangle = new Class({
   //properties
   width: 0,
   height: 0,
   //methods
   initialize: function(widthVal, heightVal)
   {
      this.width = widthVal;
      this.height = heightVal;
   },
   details: function()
   {
      document.write("Welcome to MooTools demo program");
      document.write("Width: "+this.width+" Height: "+this.height);
   },
});

var rec = new Rectangle(5,4);
```

```
rec.details();
</script>
</head>
<body>
</body>
</html>
```

You will receive the following output:

```
Welcome to MooTools demo program
Width: 5 Height: 4
```

4. MOOTOOLS – SELECTORS

Selectors are used to select HTML elements. Whenever you want to make interactive web pages, you need to select some data or an action from that web page. Selectors help us receive data through HTML request from elements.

Basic Selector ($)

The **$** is the basic selector in MooTools. Using this, you can select DOM element by its ID. For example, suppose you have an HTML element (such as div) named **body_id**.

```
<div id="body_id">
</div>
```

If you want to select this div, use the following syntax:

```
//selects the element with the ID 'body_id'
$('body_id');
```

getElement( )

getElement() is a method which extends basic selector ($). It allows you to refine your selection using element ID. getElement() only selects the single element and will return the first if there are multiple options. You can also use Class name to get the first occurrence of an element. But it will not get array of elements.

Multiple Selector ($$)

The $$ is used to select multiple elements and place those multiple elements into an array. From that array we can manipulate, retrieve, and reorder the list in different ways. Take a look at the following syntax. It defines how to select all div elements from a collection of HTML elements on a webpage.

```
<div>
    <div>a div</div>
    <span id="id_name">a span</span>
</div>
```

If you want to select all divs, use the following syntax:

```
//all divs in the page
$$('div');
```

If you want to select multiple divs with the same id name, use the following syntax.

```
//selects the element with the id 'id_name' and all divs
$$('#id_name', 'div');
```

getElements()

getElements() method is similar to getElement() method. This method returns all elements according to the criteria. You can use either **element name (a, div, input)** to select those collections or a particular element **class name** for selecting collection of elements of the same class.

Include and exclude results with operators

MooTools supports different operators used to refine your selections. You can use all these operators in getElements() method. Each of these operators can be used to select an input element by name.

Take a look at the following table. It defines the different operators that MooTools supports.

S. No.	Operator	Description & Example
1	= (equal to)	Select input element by its name. For example, $('body_wrap').getElements('input[name=phone_number]');
2	^= (starts with)	Select input element by comparing its starting letters of the name. For example, $('body_wrap').getElements('input[name^=phone]');
3	$= (ends with)	Select the input element by comparing its ending letters of the name. For example, $('body_wrap').getElements('input[name$=number]');

4	!= (is not equal to)	De-select the input element by is name. For example, $('body_wrap').getElements('input[name!=address]');
5	*= (Contains)	Select the input element which contains particular letter pattern. For example, $('body_wrap').getElements('input[name*=phone]');

Selectors based on element order

MooTools selectors follow a particular order in element selection. The selectors mainly follow two orders; one is even and the other is odd.

Note: This selector starts at 0, so the first element is even.

Even order

In this order, the selector selects the elements which are placed in an even order. Use the following syntax to select all even divs in your HTML page.

```
// selects all even divs
$$('div:even');
```

Odd order

In this order, the selector selects the element placed in an odd order. Use the following syntax to select all odd divs in your HTML page.

```
// selects all odd divs
$$('div:odd');
```

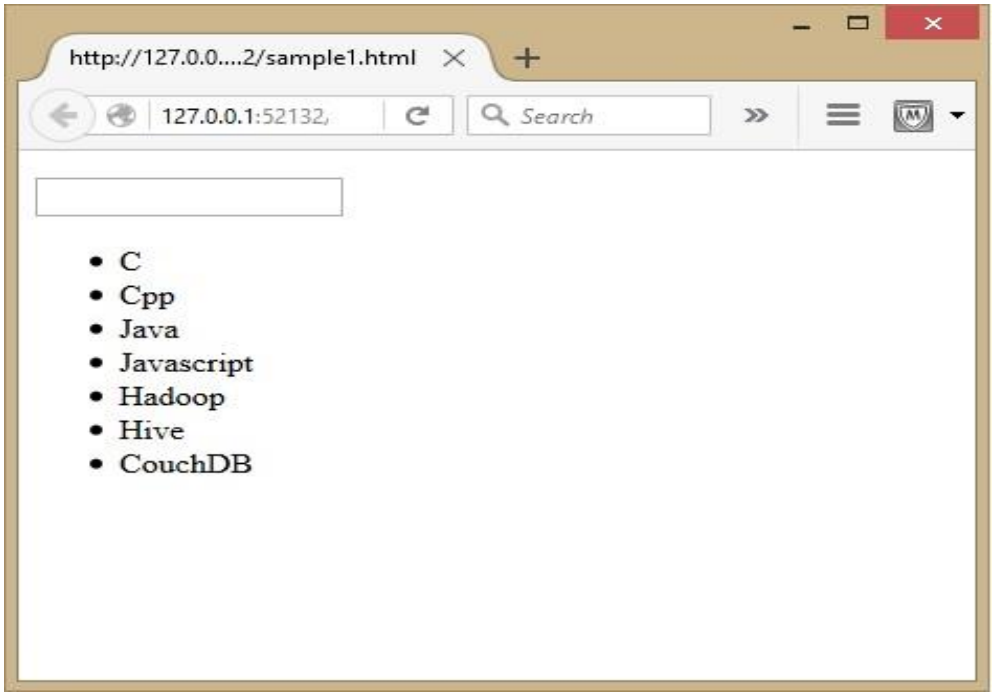
Example

The following example shows how a selector works. Suppose, there is a textbox and a list of technologies on a webpage. If you pick one technology from the list by entering that name into the textbox, then the list shows the filtered results based on your input. This is possible using the MooTools selector. Using selector, we can add an event to the textbox. The event listener will pick the data from the textbox and check it from the list. If it is there in the list, then the list shows the filtered results. Take a look at the following code.

```
<html>
<head>
<script type = "text/javascript" src = "MooTools-Core-1.6.0.js"></script>
<script type = "text/javascript" src = "MooTools-More-1.6.0.js"></script>
<script type = "text/javascript">
window.addEvent('domready',function(){

    var input = $('filter');

    // set the title attribute of every element
    // to it's text in lowercase
   $$('ul > li').each(function(item){
        item.set('title', item.get('text').toLowerCase());
   });

   // the function we'll call when the user types
    var filterList = function(){
        var value = input.value.toLowerCase();
       $$('li').setStyle('display','none');


      // check the title attribute if it contains whatever the user is typing
       $$('ul > li[title*=' + value + ']').setStyle('display','');
   };


    // make it happen
    input.addEvent('keyup', filterList);
});
</script>
</head>
<body>
```

```
<p><input id="filter" type="text" /></p>
<ul>
    <li>C</li>
    <li>Cpp</li>
    <li>Java</li>
    <li>JavaScript</li>
    <li>Hadoop</li>
    <li>Hive</li>
    <li>CouchDB</li>
</ul>
</body>
</html>
```

You will receive the following output:



5. MOOTOOLS – USING ARRAYS

MooTools is a lightweight JavaScript library which helps to create dynamic web pages. While managing DOM element, we need to select all DOM elements of a web page. This collection can be handled using arrays.

This chapter explains about how to use arrays to manage DOM elements.

each() method

This is the basic method to deal with arrays. It iterates all the elements through a list. You can use this method based on the requirement. For example, if you want to select all the div elements of a page, follow the script given below. Take a look at the following html page which contains multiple divs.

```
<div>One</div>
<div>Two</div>
```

You can use the following script to select **each individual div** from a collection of divs on the page. The script will select each div and pass an alert. Take a look at the following script.

```
$$('div').each(function() {
    alert('a div');
});
```

You can use the following syntax to handle the above given example. Take a look at the HTML page.

```
<div id="body_div">
    <div>One</div>
    <div>Two</div>
</div>
```

Here, the two divs are enclosed with another div — **body_div**. While designing a script, we have to select only one external div. Later, by using getElements() method, we can select the two internal divs. Take a look at the following script.

```
$('body_wrap').getElements('div').each(function() {
    alert('a div');
});
```

You can use a different method to write the above script as follows. Here, we are using a separate variable to select the **body_div**.

```
var myArray = $('body_div').getElements('div');
myArray.each(function() {
    alert('a div');
});
```

Select Specific Elements from an Array

While manipulating an array of elements, we can select a specific element from an array of elements. The following are some important methods used to manipulate the DOM elements:

getLast()

This method returns the last element of an array. Let us set up an array to understand this method.

```
var myArray = $('body_div').getElements('div');
```

We can now grab the last element within the array.

```
var lastElement = myArray.getLast();
```

The variable **lastElement** now represents the last element within myArray.

getRandom()

getRandom() method works the similar way like the getLast() method, but will get a random element from array.

```
var randomElement = myArray.getRandom();
```

The variable **randomElement** now represents a randomly chosen element within **myArray**.

Copy of an Array

MooTools provides a way to copy an array using the $A() function. The following is the syntax for the $A() function.

```
var <variable-name> = $A ( <array-variable>);
```

Add an Element to an Array

There are two different methods for adding elements into an array. The first method lets you add elements one by one or you can merge two different arrays into one.

include()

include() method is used to add an item into an array of DOM elements. For example, consider the following HTML code which contains two div elements and one span element under a single and enclosed div — **body_div**.

```
<div id="body_div">
    <div>one</div>
    <div>two</div>
    <span id="add_to_array">add to array</span>
</div>
```

In the above code, if we call getElements('div') method on the **body_div** element, we get one and two div but the span element is not included into the array. If you want to add it into the array you call **include()** method on the array variable. Take a look at the following script.

```
//creating array variable by selecting div elements
var myArray = $('body_wrap').getElements('div');

//first add your element to a var
var newToArray = $('add_to_array');

//then include the var in the array
myArray.include(newToArray);
```

Now, the myArray contains both divs and span element.

combine()

This method is used to combine the elements of one array with the elements of another array. This also takes care of duplicate content. For example, consider the following HTML code which contains two div elements and two span elements under single and enclosed div — **body_div**.

```
<div id="body_div">
    <div>one</div>
    <div>two</div>
    <span class="class_name">add to array</span>
    <span class="class_name">add to array, also</span>
    <span class="class_name">add to array, too</span>
</div>
```

In the above code, call getElements('div') method on the **body_div** element. You get one and two div. Call $$('.class_name') method selects the two span elements. You now have one array of div elements and another array of span elements. If you want to merge these two arrays, then you can use the combine method(). Take a look at the following script.

```
//create your array just like we did before
var myArray= $('body_wrap').getElements('div');

//then create an array from all elements with .class_name
var newArrayToArray = $$('.class_name');

//then combine newArrayToArray with myArray
myArray.combine(newArrayToArray );
```

Now, the myArray contains all the elements of newArrayToArray variable.

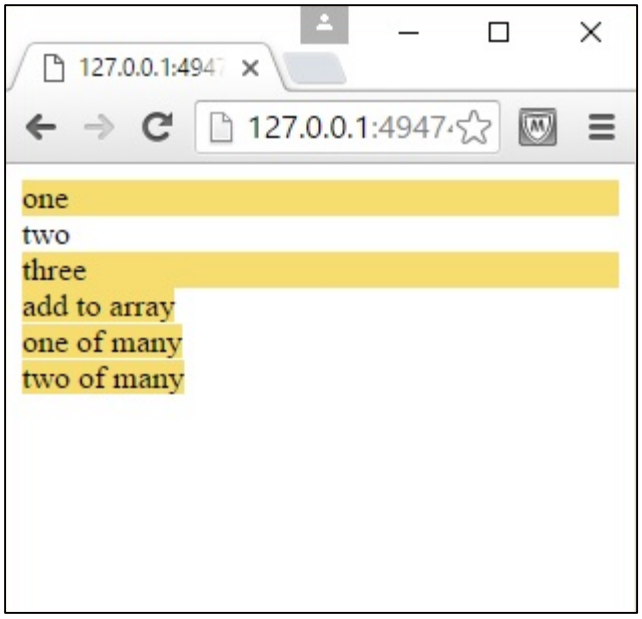
Example

This will help you understand arrays in MooTools. Suppose, we apply the background color to the array of element which contains divs and span. Take a look at the following code. Here, the second array of elements does not belong to any id or class group and that is why it does not reflect any background color. Take a look at the following code.

```
<!DOCTYPE html>
<html>
<head>
```

```
<script type = "text/javascript" src = "MooTools-Core-1.6.0.js"></script>
<script type = "text/javascript" src = "MooTools-More-1.6.0.js"></script>
<script type = "text/javascript">
window.addEvent('domready', function() {
var myArray = $('body_wrap').getElements('.class_name');
var addSpan = $('addtoarray');
var addMany = $$('.addMany');
myArray.include(addSpan);
myArray.combine(addMany);
var myArrayFunction = function(item) {
     item.setStyle('background-color', '#F7DC6F');
}
myArray.each(myArrayFunction);
});
</script>
</head>
<body>
<div id="body_wrap">
<div class="class_name">one</div>
<div>two</div>
<div class="class_name">three</div>
<span id="addtoarray">add to array</span>
<br /><span class="addMany">one of many</span>
<br /><span class="addMany">two of many</span>
</div>
</body>
</html>
```

You will receive the following output:



6. MOOTOOLS – FUNCTIONS

Functions in MooTools is a concept from JavaScript. We already know how to use functions in JavaScript. Generally, it is better to keep the function outside the page body in the script tag. In MooTools, we follow the same pattern. Here, you can design your own function according to the requirement. We now have to call all the user-defined functions in the **domready** function.

Take a look at the following syntax to understand how to use generalized function in MooTools.

```
<script type="text/javascript">
     /*
     Function definitions go here
     */
     window.addEvent('domready', function() {
         /* Calls to functions go here */
     });
</script>
```

Basic Structure

There are a few basic ways to define a function in MooTools. There is no difference between the function syntaxes of JavaScript and MooTools but the difference is in calling a function. Let us take a small example that defines a function named demo_function. Take a look at the following code.

```
<html>
<head>
<script type = "text/javascript" src = "MooTools-Core-1.6.0.js"></script>
<script type = "text/javascript" src = "MooTools-More-1.6.0.js"></script>
<script type = "text/javascript">
//Define simple_function as a function
var simple_function = function(){
     document.write('This is a simple function');
}
```

```
window.addEvent('domready', function() {
     //Call simple_function when the dom(page) is ready
     simple_function();
});
</script>
    </head>
<body>

</body>
</html>
```

You will receive the following output:

```
This is a simple function
```

Single Parameter Function

You can also create a function that accepts a parameter. To use parameters with functions, you need to add a variable name in the parenthesis. Once you provide it, the variable is available inside for use. Let us take an example that defines a function that takes a single parameter and prints a message along with the parameter.

Take a look at the following code.

```
<!DOCTYPE html>
<html>
<head>
<script type = "text/javascript" src = "MooTools-Core-1.6.0.js"></script>
<script type = "text/javascript" src = "MooTools-More-1.6.0.js"></script>
<script type = "text/javascript">
var single_parameter_function = function(parameter){
     document.write('The parameter is : ' + parameter);
}

window.addEvent('domready', function(){
     single_parameter_function('DEMO PARAMETER');
});
```

```
</script>
</head>
<body>
</body>
</html>
```

You will receive the following output:

```
The parameter is: DEMO PARAMETER
```

Returning a Value

Whenever you want to use the result of one function as input for another variable, you are required to use the return value for that function. You can use the return keyword for returning a value from the function. Let us take an example that defines a function that will accept two parameter values and return the sum of those two parameters. Take a look at the following code.

```
<!DOCTYPE html>
<html>
<head>
<script type = "text/javascript" src = "MooTools-Core-1.6.0.js"></script>
<script type = "text/javascript" src = "MooTools-More-1.6.0.js"></script>
<script type = "text/javascript">
var two_parameter_returning_function = function(first_number, second_number){
      var third_number = first_number + second_number;
      return third_number;
}

window.addEvent('domready', function(){
      var return_value = two_parameter_returning_function(10, 5);
      document.write("Return value is : " + return_value);
});
</script>
</head>
<body>
</body>
```

```
</html>
```

You will receive the following output:

```
Return value is: 15
```

End of ebook preview
If you liked what you saw…
Buy it from our store @ **https://store.tutorialspoint.com**