prototype

javascript framework

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

This tutorial gives a complete understanding on *Prototype. Prototype* is distributed as a single file called *prototype.js*. Prototype is an object in javascript from which other objects inherit properties.

Audience

This tutorial has been written for users willing to learn the Javascript Prototype object and its usage. Beginners as well as experienced users can refer this tutorial to brush up or learn the Javascript prototypes.

Prerequisites

To learn prototype you should have the basic knowledge of Javascript and its properties.

Copyright & Disclaimer

© Copyright 2016 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com

tutorialspoint
SIMPLYEASYLEARNING

Table of Contents

tutorialspoint
SIMPLYEASYLEARNING

tutorialspoint
SIMPLYEASYLEARNING

tutorialspoint
SIMPLYEASYLEARNING

tutorialspoint
SIMPLYEASYLEARNING

tutorialspoint
SIMPLYEASYLEARNING

tutorialspoint
SIMPLYEASYLEARNING

tutorialspoint
SIMPLYEASYLEARNING

tutorialspoint
SIMPLYEASYLEARNING

tutorialspoint
SIMPLYEASYLEARNING

tutorialspoint
SIMPLYEASYLEARNING

tutorialspoint
SIMPLYEASYLEARNING

tutorialspoint
SIMPLYEASYLEARNING

1. PROTOTYPE – OVERVIEW

What is Prototype ?

Prototype is a JavaScript Framework that aims to ease the development of dynamic web applications. Prototype was developed by Sam Stephenson.

Prototype is a JavaScript library, which enables you to manipulate DOM in a very easy and fun way that is also safe (cross-browser).

Scriptaculous and other libraries, such as *Rico* are build on Prototype's foundations to create widgets and other end-user stuff.

Prototype:

- Extends DOM elements and built-in types with useful methods.
- Has built-in support for class-style OOP including inheritance.
- Has advanced support for event management.
- Has powerful Ajax features.
- Is not a complete application development framework.
- Does not provide widgets or a full set of standard algorithms or I/O systems.

How to Install Prototype?

Prototype is distributed as a single file called prototype.js. Follow the below mentioned steps to setup the prototype library:

- Go to the download page (http://prototypejs.org/download/) to grab the latest version in a convenient package.
- Now, put prototype.js file in a directory of your website, e.g. /javascript.

You are now ready to use the powerful Prototype framework in your web pages.

How to Use Prototype Library?

Now, you can include the *Prototype* script as follows:

tutorialspoint
SIMPLYEASYLEARNING

```
<html>

<head>


<title>Prototype examples</title>

   <script type="text/javascript"

   src="/javascript/prototype.js">

   </script>

</head>

<body>

........


</body>

</html>
```

Example

Here is a simple example showing how you can use Prototype's $() function to get DOM elements in your JavaScript:

```
<html>

<head>


<title>Prototype examples</title>

   <script type="text/javascript"

   src="/javascript/prototype.js">

   </script>
```


tutorialspoint
SIMPLYEASYLEARNING

```
   <script>
   function test(){
       node = $("firstDiv");
       alert(node.innerHTML);
   }
   </script>

</head>

<body>
   <div id="firstDiv">
      <p>This is first paragraph</p>
   </div>
   <div id="secondDiv">

      <p>This is another paragraph</p>
   </div>

   <input type="button" value="Test $()" onclick="test();"/>

</body>
</html>
```


tutorialspoint
SIMPLYEASYLEARNING

Why This Tutorial?

A very good documentation for Prototype Framework is available at prototypejs.org then why should one refer to this tutorial!

The answer is that we have put all the most commonly used functionalities together in this tutorial. Secondly, we have explained all the useful methods along with suitable examples, which are not available at the official site.

If you are an advanced user of Prototype Framework, then you can directly jump to the official website, otherwise this tutorial could be a good start for you and you can use it like a reference manual.

tutorialspoint
SIMPLYEASYLEARNING

2. PROTOTYPE – USEFUL FEATURES

Let's now look at what Prototype can do specifically for us to develop a Dynamic Web Application.

Cross Browser Support

While doing JavaScript programming, it is required to handle different Web Browsers differently. Prototype Library has been written in such a way that it takes care of all the compatibility issues and you can do cross browser programming without any hassle.

The Document Object Model

Prototype provides helper methods that ease some of the strain of DOM programming. Using Prototype, you can manipulate DOM very easily.

HTML Forms

With Ajax, other input mechanisms such as drag and drop, can be used as part of a conversation between the browser and the server. With conventional JavaScript programming, it is difficult to capture these inputs and pass them to the server. Prototype provides a set of utilities for working with HTML forms.

JavaScript Events

Prototype provides some excellent cross-browser support while coding events, and also extends the Function object to make it easy to work with event handling.

Ajax Utilities

The most important feature of Prototype is it's support for Ajax. All major browsers support a version of the XMLHttpRequest object that makes Ajax possible, either as an ActiveX component or as a native JavaScript object.

XMLHttpRequest, however, exposes the HTTP protocol at a very low level, which gives the developer a lot of power, but also requires her to write a lot of code in order to do simple things.

tutorialspoint
SIMPLYEASYLEARNING

Prototype uses it's own object inheritance system to provide a hierarchy of Ajax helper objects, with more generic base classes being subclassed by more focused helpers that allow the most common types of Ajax request to be coded in a single line.

tutorialspoint
SIMPLYEASYLEARNING

3. PROTOTYPE – UTILITY METHODS

The Prototype library comes with lot of predefined objects and utility functions. You can use those functions and objects directly in your JavaScript programming.

These methods are one of the cornerstones of efficient Prototype-based JavaScript coding. Spend some time to study them to become comfortable with the methods.

This chapter details all these useful methods with examples.

Method	Description
$()	If provided with a string, returns the element in the document with matching ID; otherwise returns the passed element.
$$()	Takes an arbitrary number of CSS selectors (strings) and returns a document-order array of extended DOM elements that match any of them.
$A()	Converts the single argument it receives into an Array object.
$F()	Returns the value of a form control. This is a convenience alias of Form.Element.getValue.
$H()	Converts objects into enumerable Hash objects that resemble associative arrays.
$R()	Creates a new ObjectRange object.
$w()	Splits a string into an Array, treating all whitespace as delimiters.
Try.these	Accepts an arbitrary number of functions and returns the result of the first one that doesn't throw an error.

tutorialspoint
SIMPLYEASYLEARNING

$() Method

The most commonly used and convenient function, $(), provides an easy way of getting a handle on a DOM element.

Syntax

```
$(id | element)


OR


$((id | element)...)
```

Return Value

- Found HTMLElement.
- In case it finds more than one elements, then it returns array of HTML elements.

Here is an old way of writing Javascript statement to get a node of DOM.

```
node = document.getElementById("elementID");
```

Using $(), we can shorten it up as follows:

```
node = $("elementID");
```

Example

```
<html>

<head>

<title>Prototype examples</title>

   <script type="text/javascript"
```


tutorialspoint
SIMPLYEASYLEARNING

```
      src="/javascript/prototype.js">
      </script>
      <script>
      function test(){
          node = $("firstDiv");
          alert(node.innerHTML);
      }
      </script>
   </head>

   <body>
      <div id="firstDiv">
         <p>This is first paragraph</p>
      </div>
      <div id="secondDiv">
         <p>This is another paragraph</p>
      </div>

      <input type="button" value="Test $()" onclick="test();"/>

   </body>
   </html>
```


tutorialspoint
SIMPLYEASYLEARNING

Fetching Multiple Values Using $()

The $() function is also more powerful than **document.getElementById()** because the ability to retrieve multiple elements is built into the function.

Another nice thing about this function is that you can pass either the id string or the element object itself, which makes this function very useful when creating other functions that can also take either form of argument.

Example

In this example, we see the $() function now returning an array of our elements, which can then be accessed with a simple **for** loop.

```
<html>

<head>

<title>Prototype examples</title>

   <script type="text/javascript"

   src="/javascript/prototype.js">

   </script>

   <script>

   function test(){

       allNodes = $("firstDiv", "secondDiv");

       for(i = 0; i < allNodes.length; i++) {

         alert(allNodes[i].innerHTML);

       }

   }

   </script>

</head>
```


tutorialspoint
SIMPLYEASYLEARNING

```
<body>
   <div id="firstDiv">
      <p>This is first paragraph</p>
   </div>
   <div id="secondDiv">
      <p>This is another paragraph</p>
   </div>

   <input type="button" value="Test $()" onclick="test();"/>

</body>
</html>
```

$$() Method

The $$() method parses one or more CSS filtering expressions, analogous to the ones used to define CSS rules, and returns the elements that match these filters.

Syntax

```
$$(cssRule...);
```

Return Value

- An array of HTML elements.

Example

Here is an old way of writing Javascript statement to get all the nodes of DOM with name div.

tutorialspoint
SIMPLYEASYLEARNING

```
nodes = document.getElementsByTagName('div');
```

Using $$(), we can shorten it up as follows:

```
nodes = $$('div');
```

Following is same as $('contents'), only it returns an array anyway.

```
$$('#contents');
```

Example

```
<html>

<head>

<title>Prototype examples</title>

   <script type="text/javascript"

   src="/javascript/prototype.js">

   </script>

   <script>


  function test(){

       allNodes = $$("div");

       for(i = 0; i < allNodes.length; i++) {

         alert(allNodes[i].innerHTML);

       }

   }

   </script>
```


tutorialspoint
SIMPLYEASYLEARNING

```
</head>

<body>
   <div id="firstDiv" name="div">
      <p>This is first paragraph</p>
   </div>

   <div id="secondDiv" name="div">
      <p>This is another paragraph</p>
   </div>

   <input type="button" value="Test $()" onclick="test();"/>

</body>
</html>
```

More Examples

Following returns all links inside the element of ID "contents" with a rel attribute.

```
$$('#contents a[rel]');
```

Following returns all links with a href attribute of value "#" (eyeew!).

```
$$('a[href="#"]');
```

Following returns all links within the elements of ID "navbar" or "sidebar".

tutorialspoint
SIMPLYEASYLEARNING

```
$$('#navbar a', '#sidebar a');
```

Following returns all links, excluding those whose rel attribute contains the word "nofollow".

```
$$('a:not([rel~=nofollow])');
```

Following returns all even rows within all table bodies.

```
$$('table tbody > tr:nth-child(even)');
```

Following returns all DIVs without content (i.e., whitespace-only).

```
$$('div:empty');
```

$A() Method

The $A() function converts the single argument it receives into an Array object.

One suggested use is to convert DOM NodeLists into regular arrays, which can be traversed more efficiently.

Syntax

```
$A(iterable)
```

Return Value

- List of elements in the form of array .

Example

```
<html>

<head>

<title>Prototype examples</title>
```


tutorialspoint
SIMPLYEASYLEARNING

```
   <script type="text/javascript"
   src="/javascript/prototype.js">
   </script>
<script>

function showOptions(){
   var NodeList = $('employees').getElementsByTagName('option');
   var nodes = $A(NodeList);

   nodes.each(function(node){
      alert(node.nodeName + ': ' + node.innerHTML);
   });
}
</script>
</head>

<body>

   <select id="employees" size="10" >
      <option value="5">Mohtashim, Mohd</option>
      <option value="8">Debi, Patnaik</option>
      <option value="1">Madisetti, Praveen</option>
   </select>
```


tutorialspoint
SIMPLYEASYLEARNING

```
    <br />

    <input type="button" value="Show the options"
              onclick="showOptions();"/>

</body>

</html>
```

$F() Method

The $F() function returns the value of any field input control, like text boxes or drop-down lists. This is a convenience alias of *Form.Element.getValue*.

The function can take as argument either the element id or the element object itself.

Syntax

```
$F(element)
```

Return Value

- Form's element Value.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"
```


tutorialspoint
SIMPLYEASYLEARNING

```
   src="/javascript/prototype.js">
</script>
<script>

function ShowValue(){
      var value = $F('userName');

      alert( 'Entred  Value :' + value);
}
</script>
</head>

<body>

   <p>Enter any value in the box and then click "Show Value"</p>

   <form>
   <input type="text" id="userName" value="Madisetti, Praveen">
   <br/>

   <input type="button" value="Show Value" onclick="ShowValue();"/>
   </form>
```


tutorialspoint
SIMPLYEASYLEARNING

```
</body>

</html>
```

$H() Method

The $H() function converts objects into enumerable Hash objects that resemble associative arrays.

The $H function is the shorter way to obtain a hash.

Syntax

```
$H([obj])
```

Return Value

- A hash object.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>



function ShowHash()

{
```


tutorialspoint
SIMPLYEASYLEARNING

```
   //let's create the object
   var a = {
           first: 10,
           second: 20,
           third: 30
   };

   //now transform it into a hash
   var h = $H(a);
   alert( h.toQueryString());

}
</script>
</head>

<body>

   <p>Click "Show Value" button to see the result</p>

   <form>
   <input type="button" value="Show Value" onclick="ShowHash();"/>
   </form>
```


tutorialspoint
SIMPLYEASYLEARNING

```
</body>

</html>
```

This will display the following result:

```
first=10&second=20&third=30
```

$R() Method

The $R() function is simply a short hand to writing new ObjectRange(lowerBound, upperBound, excludeBounds).

Syntax

```
$R(start, end[, exclusive = false]);
```

Here, *start* is the starting element of the range and *end* is the last element of the range. If *exclusive* flag is set to false, then it will include the ending elements, otherwise it will not be included in the range.

Return Value:

- Range Object.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>
```


tutorialspoint
SIMPLYEASYLEARNING

```
<script>

function ShowValues()
{
   var range = $R(10, 20, false);
   range.each(function(value, index){
         alert(value);
   });

}
</script>
</head>

<body>

   <p>Click "Show Value" button to see the result</p>

   <form>
   <input type="button" value="Show Value" onclick="ShowValues();"/>
   </form>

</body>
</html>
```


tutorialspoint
SIMPLYEASYLEARNING

More Examples

Following statement returns *true* value:

```
$R(0, 10).include(10);
```

Following statement returns a string "0, 1, 2, 3, 4, 5":

```
$A($R(0, 5)).join(', ');
```

Following statement returns a string "aa, ab, ac, ad, ae, af, ag, ah":

```
$A($R('aa', 'ah')).join(', ');
```

Following statement returns *false*:

```
$R(0, 10, true).include(10);
```

Following statement will be invoked 10 times for value = 0 to 9:

```
$R(0, 10, true).each(function(value) {

  // invoked 10 times for value = 0 to 9
```

$w() Method

The $w() function splits a string into an Array, treating all whitespace as delimiters.

Syntax

```
$w(String);
```

Return Value

- An array of strings.

tutorialspoint
SIMPLYEASYLEARNING

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function ShowValues()

{

   var str = "Apples Bananas Kiwis";


   // Convert string into Array

   var arr = $w(str);


   arr.each(function(value){

         alert(value);

   });


}

</script>

</head>
```


tutorialspoint
SIMPLYEASYLEARNING

```
<body>

   <p>Click "Show Value" button to see the result</p>

   <form>
   <input type="button" value="Show Value" onclick="ShowValues();"/>
   </form>

</body>
</html>
```

Try.these Method

The Try.these() function makes it easy when you want to try different function calls, until one of them works.

It takes a number of functions as arguments and calls them one by one, in sequence, until one of them works, returning the result of that successful function call.

If none of the blocks succeeded, Try.these will return undefined, i.e., false.

Syntax

```
Try.these(Function...);
```

Return Value

- First OK result.

Example

tutorialspoint
SIMPLYEASYLEARNING

There are different ways to create XMLHttp object in different browsers. Using the Try.these() function we can return the one that works.

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


createXMLHttp: function()

{

   return Try.these(

    function() { return new XMLHttpRequest() },

    function() { return new ActiveXObject('Msxml2.XMLHTTP') },

    function() { return new ActiveXObject('Microsoft.XMLHTTP') }

  ) || false;


}

</script>

</head>


<body>

......
```


tutorialspoint
SIMPLYEASYLEARNING

```
</body>

</html>
```

If none of the blocks succeeded, Try.these will return undefined, which will cause the createXMLHttp method in the example above to return false, provided as a fallback result value.

document.getElementsByClassName

This method retrieves (and extends) all the elements that have a CSS class name of *className*.

However, this method has been deprecated in the latest versions of Prototype.

tutorialspoint
SIMPLYEASYLEARNING

4. PROTOTYPE – ELEMENT OBJECT

The *Element* object provides various utility functions for manipulating elements in the DOM. Here is a list of all the utility functions with examples. All the methods defined here are automatically added to any element accessed using the $() function.

So, writing `Element.show('firstDiv');` is the same as writing `$('firstDiv').show();`

Prototype Element Method

NOTE: Make sure you have at least version 1.6 of prototype.js.

Method	Description
absolutize()	Turns element into an absolutely-positioned element without changing its position in the page layout.
addClassName()	Adds the given CSS class name to the element's class names.
addMethods()	Makes it possible to mix in your own methods to the Element object, which you can later use as methods of extended elements.
adjacent()	Finds all siblings of the current element that match the given selector(s).
ancestors()	Collects all of element's ancestors and returns them as an array of extended elements.
childElements()	Collects all of the element's children and returns them as an array of extended elements.

tutorialspoint
SIMPLYEASYLEARNING

classNames()	Deprecated. Returns a new instance of ClassNames, an Enumerable object used to read and write CSS class names of element.
cleanWhitespace()	Removes all of element's text nodes, which contain only whitespace. Returns element.
clonePosition()	Clones the position and/or dimensions of source onto element as defined by the optional argument options.
cumulativeOffset()	Returns the offsets of element from the top left corner of the document.
cumulativeScrollOffset()	Calculates the cumulative scroll offset of an element in nested scrolling containers.
descendantOf()	Checks if the element is a descendant of ancestor.
descendants()	Collects all of element's descendants and returns them as an array of extended elements.
down()	Returns element's first descendant that matches cssRule. If no cssRule is provided, all descendants are considered. If no descendant matches these criteria, undefined is returned.
empty()	Tests whether element is empty (i.e., contains only whitespace).
extend()	Extends element with all of the methods contained in Element.Methods and Element.Methods.Simulated.
fire()	Fires a custom event with the current element as its target.

firstDescendant()	Returns the first child that is an element. This is opposed to firstChild DOM property, which will return any node.
getDimensions()	Finds the computed width and height of an element and returns them as key/value pairs of an object.
getElementsByClassName	Deprecated. Fetches all of element's descendants, which have a CSS class of className and returns them as an array of extended elements. Please use $$().
getElementsBySelector	Deprecated. Takes an arbitrary number of CSS selectors (strings) and returns an array of extended children of element that match any of them. Please use $$().
getHeight()	Finds and returns the computed height of element.
getOffsetParent()	Returns element's closest positioned ancestor. If none is found, the body element is returned.
getStyle()	Returns the given CSS property value of element. Property can be specified in either of its CSS or camelized form.
getWidth()	Finds and returns the computed width of element.
hasClassName()	Checks whether element has the given CSS className.
hide()	Hides and returns element.
identify()	Returns element's id attribute if it exists, or sets and returns a unique, auto-generated id.
immediateDescendants()	Deprecated. Collects all of the element's immediate descendants (i.e., children) and returns them as an array of extended elements. Please use childElements().

tutorialspoint
SIMPLYEASYLEARNING

insert()	Inserts content before, after, at the top of, or at the bottom of element.
inspect()	Returns the debug-oriented string representation of element.
makeClipping()	Simulates the poorly supported CSS clip property by setting element's overflow value to 'hidden'. Returns element.
makePositioned()	Allows for the easy creation of CSS containing block by setting element's CSS position to 'relative' if its initial position is either 'static' or undefined. Returns element.
match()	Checks if element matches the given CSS selector.
next()	Returns element's following sibling that matches the given cssRule.
nextSiblings()	Collects all of element's next siblings and returns them as an array of extended elements.
observe()	Registers an event handler on element and returns element.
positionedOffset ()	Returns element's offset relative to its closest positioned ancestor.
previous ()	Returns element's previous sibling that matches the given cssRule.
previousSiblings ()	Collects all of element's previous siblings and returns them as an array of extended elements.
readAttribute ()	Returns the value of element's attribute or null if attribute has not been specified.

tutorialspoint
SIMPLYEASYLEARNING

recursivelyCollect ()	Recursively collects elements whose relationship is specified by property.
relativize ()	Turns element into an relatively-positioned element without changing its position in the page layout.
remove ()	Completely removes element from the document and returns it.
removeClassName ()	Removes element's CSS className and returns element.
replace ()	Replaces element by the content of the html argument and returns the removed element.
scrollTo ()	Scrolls the window so that element appears at the top of the viewport. Returns element.
select ()	Takes an arbitrary number of CSS selectors (strings) and returns an array of extended descendants of element that match any of them.
setOpacity ()	Sets the visual opacity of an element while working around inconsistencies in various browsers.
setStyle ()	Modifies element's CSS style properties.
show ()	Displays and returns element.
siblings ()	Collects all of element's siblings and returns them as an array of extended elements.
stopObserving ()	Unregisters handler and returns element.

tutorialspoint
SIMPLYEASYLEARNING

toggle ()	Toggles the visibility of element.
toggleClassName ()	Toggles element's CSS className and returns element.
undoClipping ()	Sets element's CSS overflow property back to the value it had before Element.makeClipping() was applied. Returns element.
undoPositioned ()	Sets element back to the state it was before Element.makePositioned was applied to it. Returns element.
up()	Returns element's first ancestor that matches the given cssRule.
update()	Replaces the content of element with the provided newContent argument and returns element.
viewportOffset()	Returns the X/Y coordinates of element relative to the viewport.
visible()	Returns a Boolean indicating whether or not element is visible.
wrap()	Wraps an element inside another, then returns the wrapper.
writeAttribute()	Adds, specifies or removes attributes passed as either a hash or a name/value pair.

absolutize() Method

This method turns element into an absolutely-positioned element without changing its position in the page layout.

tutorialspoint
SIMPLYEASYLEARNING

Syntax

```
element.absolutize();
```

Return Value

- An absolutely-positioned HTML element.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function changetoAbs(){

   node = $("firstDiv");

   node.style.fontSize='20px';

   //node.style.position = 'absolute';

   node.absolutize();

   node.style.border = '1px dashed #f00';

   node.style.left = '100px';

}

</script>
```


tutorialspoint
SIMPLYEASYLEARNING

```
</head>


<body>
   <div id="firstDiv">
      <p>This is first paragraph</p>
   </div>

   <br />
   <input type="button" value="Make Absolute"
             onclick="changetoAbs();"/>

</body>
</html>
```

addClassName() Method

This method Adds a CSS class to element.

Syntax

```
element.addClassName(className);
```

Return Value

- An HTML element in which CSS class is added.

Example

tutorialspoint
SIMPLYEASYLEARNING

```
<html>
<head>
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>

function addClass(){
   node = $("firstDiv");
   node.addClassName("title");
}
</script>
</head>

<style type="text/css">
.title{
    color:#36C;
    font-size:20px;
}
</style>
<body>
   <div id="firstDiv">
```


```
        <p>This is first paragraph</p>
    </div>

    <br />
    <input type="button" value="Add Class"
              onclick="addClass();"/>

</body>
</html>
```

addMethods() Method

This method makes it possible to mix in your own methods to the Element object, which you can later use as methods of extended elements.

To add new methods, simply feed Element.addMethods with a hash of methods. Note that each method's first argument has to be an element.

Syntax

```
element.addMethods([hash of methods]);

OR

element.addMethods(tagName, methods);
```

Here, second form of the method will make added method available for a particular tag only.

tutorialspoint
SIMPLYEASYLEARNING

Return Value

- None.

Example

```
<html>
<head>
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>

// Make changeColor method available for all the elements
Element.addMethods({
    changeColor: function(element, colorName) {
    element = $(element);
    element.style.color = colorName;
    return element;
  }
});

function ShowEffect(){
   node = $("firstDiv");
```


tutorialspoint
SIMPLYEASYLEARNING

```
      // Now call changeColor method
      node.changeColor( "red" );
   }

   </script>
   </head>

   <body>
      <div id="firstDiv">
         <p>This is first paragraph</p>
      </div>

      <br />
      <input type="button" value="ShowEffect"
                onclick="ShowEffect();"/>

   </body>
   </html>
```

adjacent() Method

This method finds all siblings of the current element that matches the given selector(s) and returns them as an array.

tutorialspoint
SIMPLYEASYLEARNING

Syntax

```
element.adjacent([ selectors...]);
```

Return Value

- An array of HTML elements.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function listCities(){

   var arr = $('nyc').adjacent('li.us');


   arr.each(function(node){

      alert(node.nodeName + ': ' + node.innerHTML);

   });


}
```


tutorialspoint
SIMPLYEASYLEARNING

```
</script>
</head>

<body>
   <ul id="cities">
    <li class="us" id="nyc">New York</li>
    <li class="uk" id="lon">London</li>
    <li class="us" id="chi">Chicago</li>
    <li class="jp" id="tok">Tokyo</li>
    <li class="us" id="la">Los Angeles</li>
    <li class="us" id="aus">Austin</li>
  </ul>

   <br />
   <input type="button" value="List Cities"
             onclick="listCities();"/>
</body>
</html>
```

ancestors() Method

This method collects all of element's ancestors and returns them as an array of extended elements.

Keep in mind that the body and html elements will also be included.

Syntax

```
element.ancestors();
```

Return Value

- An array of HTML elements.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>



function showElements(){

   var arr =  $('kid').ancestors();

   arr.each(function(node){

      alert(node.nodeName + ': ' + node.innerHTML);

   });


}
```


tutorialspoint
SIMPLYEASYLEARNING

```
</script>
</head>

<body>
   <div id="father">
      <p id="kid">This is first paragraph</p>
   </div>

   <br />
   <input type="button" value="showElements"
             onclick="showElements();"/>

</body>
</html>
```

childElements() Method

Collects all of the element's children and returns them as an array of extended elements.

An index of 0 refers to the topmost child of an element.

Syntax

```
element.childElements();
```

Return Value

- An array of HTML elements.

tutorialspoint
SIMPLYEASYLEARNING

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>



function showElements(){

   var arr =  $('father').childElements();

   arr.each(function(node){

      alert(node.nodeName + ': ' + node.innerHTML);

   });


}


</script>

</head>


<body>

   <div id="father">
```


```
        <p id="kid1">This is first paragraph</p>
       <p id="kid2">This is second paragraph</p>
      </div>

      <br />
      <input type="button" value="showElements"
                onclick="showElements();"/>

   </body>
   </html>
```

cleanWhitespace() Method

This method removes all of element's text nodes, which contain only whitespace and returns element.

Element.cleanWhitespace removes whitespace-only text nodes. This can be very useful when using standard methods like *nextSibling, previousSibling, firstChild* or *lastChild* to walk the DOM.

Syntax

```
element.cleanWhitespace();
```

Return Value

- An HTML element

Example

Consider the following example:

tutorialspoint
SIMPLYEASYLEARNING

```
<html>
<head>
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>



function showElements(){
   var element = $('apples');
   alert(element.firstChild.innerHTML);
}


</script>
</head>


<body>
  <ul id="apples">
     <li>Mutsu</li>
     <li>McIntosh</li>
     <li>Ida Red</li>
  </ul>
```


tutorialspoint
SIMPLYEASYLEARNING

```
   <br />
   <input type="button" value="showElements"
            onclick="showElements();"/>

</body>
</html>
```

That doesn't seem to work to well. Why is that ? ul#apples's first child is actually a text node containing only whitespace that sits between <ul id="apples"> and Mutsu.

Now, let's use cleanWhitespace function and see the result:

```
<html>
<head>
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>



function showElements(){
   var element = $('apples');
   element.cleanWhitespace();
   alert(element.firstChild.innerHTML);
```


tutorialspoint
SIMPLYEASYLEARNING

```
   }


   </script>
   </head>


   <body>
     <ul id="apples">
        <li>Mutsu</li>
        <li>McIntosh</li>
        <li>Ida Red</li>
     </ul>


      <br />
      <input type="button" value="showElements"
                 onclick="showElements();"/>


   </body>
   </html>
```

This will display the following result:

```
   'Mutsu'
```


tutorialspoint
SIMPLYEASYLEARNING

clonePostion() Method

This method clones the position and/or dimensions of source into element as defined by the optional argument options.

Syntax

```
element.clonePosition(source[, options]);
```

Here is the list of possible options:

Name	Default	Description
setLeft	true	clones source's left CSS property onto element.
setTop	true	clones source's top CSS property onto element.
setWidth	true	clones source's width onto element.
setHeight	true	clones source's width onto element.
offsetLeft	0	Number by which to offset element's left CSS property.
offsetTop	0	Number by which to offset element's top CSS property.

Return Value

- An HTML element.

Example

```
<html>

<head>
```


tutorialspoint
SIMPLYEASYLEARNING

```
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>



function clonePosition(){
   var firstElement = $('firstDiv');
   var secondElement = $('secondDiv');
   secondElement.clonePosition( firstElement);
}


</script>
</head>


<body>


   <p>Click Clone Position button to see the result.</p>
   <div id="firstDiv">
      <p>This is first paragraph</p>
   </div>
```


tutorialspoint
SIMPLYEASYLEARNING

```
    <div id="secondDiv">
       <p>This is second paragraph</p>
    </div>

    <br />
    <input type="button" value="Clone Position"
              onclick="clonePosition();"/>

</body>
</html>
```

cumulativeOffset() Method

This method returns the offsets of element from the top left corner of the document.

This method returns an array keeping offsetLeft and offsetTop of the element.

Note that all values are returned as numbers, only although they are expressed in pixels.

Syntax

```
element.cumulativeOffset();
```

Return Value

- An array of two numbers [offset left, offset top].

Example

```
<html>
<head>
```


```
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>



function getOffset(){
   var firstElement = $('firstDiv');
   var arr = firstElement.cumulativeOffset();
   alert ( "Offset Left: " +arr[0]+ " Offset Top : " +arr[0] );
}


</script>
</head>


<body>


   <p>Click getOffset button to see the result.</p>
   <div id="firstDiv">
      <p>This is first paragraph</p>
   </div>
```


tutorialspoint
SIMPLYEASYLEARNING

```
    <br />
    <input type="button" value="getOffset"
              onclick="getOffset();"/>


</body>
</html>
```

cumulativeScrollOffset() Method

This method calculates and returns the cumulative scroll offset of an element in nested scrolling containers. This adds the cumulative scrollLeft and scrollTop of an element and all its parents.

This is used for calculating the scroll offset of an element that is in more than one scroll container (e.g., a draggable in a scrolling container, which is itself part of a scrolling document).

This method returns an array keeping offsetLeft and offsetTop of the element.

Syntax

```
element.cumulativeScrollOffset();
```

Return Value

- An array of two numbers [offset let, offset top].

Example

```
<html>
<head>
<title>Prototype examples</title>
```


tutorialspoint
SIMPLYEASYLEARNING

```
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>



function getOffset(){
   firstElement = $('firstDiv');
   var arr = firstElement.cumulativeScrollOffset();
   alert ( "Offset Left: " +arr[0]+ " Offset Top : " +arr[0]);
}


</script>
</head>


<body>


   <p>Click getOffset button to see the result.</p>
   <div id="firstDiv">
      <p>This is first paragraph</p>
   </div>


   <br />
```


tutorialspoint
SIMPLYEASYLEARNING

```
    <input type="button" value="getOffset"

              onclick="getOffset();"/>


</body>

</html>
```

descendantOf() Method

This method checks if element is a descendant of ancestor.

As Element.descendantOf internally applies $() to ancestor, it accepts indifferently an element or an element's id as its second argument.

Syntax

```
element.descendantOf(ancestor);
```

Return Value

- If it finds that element is a decendant of an ancestor, then it returns true, otherwise false.

tutorialspoint
SIMPLYEASYLEARNING

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

tutorialspoint
SIMPLYEASYLEARNING