**Events**

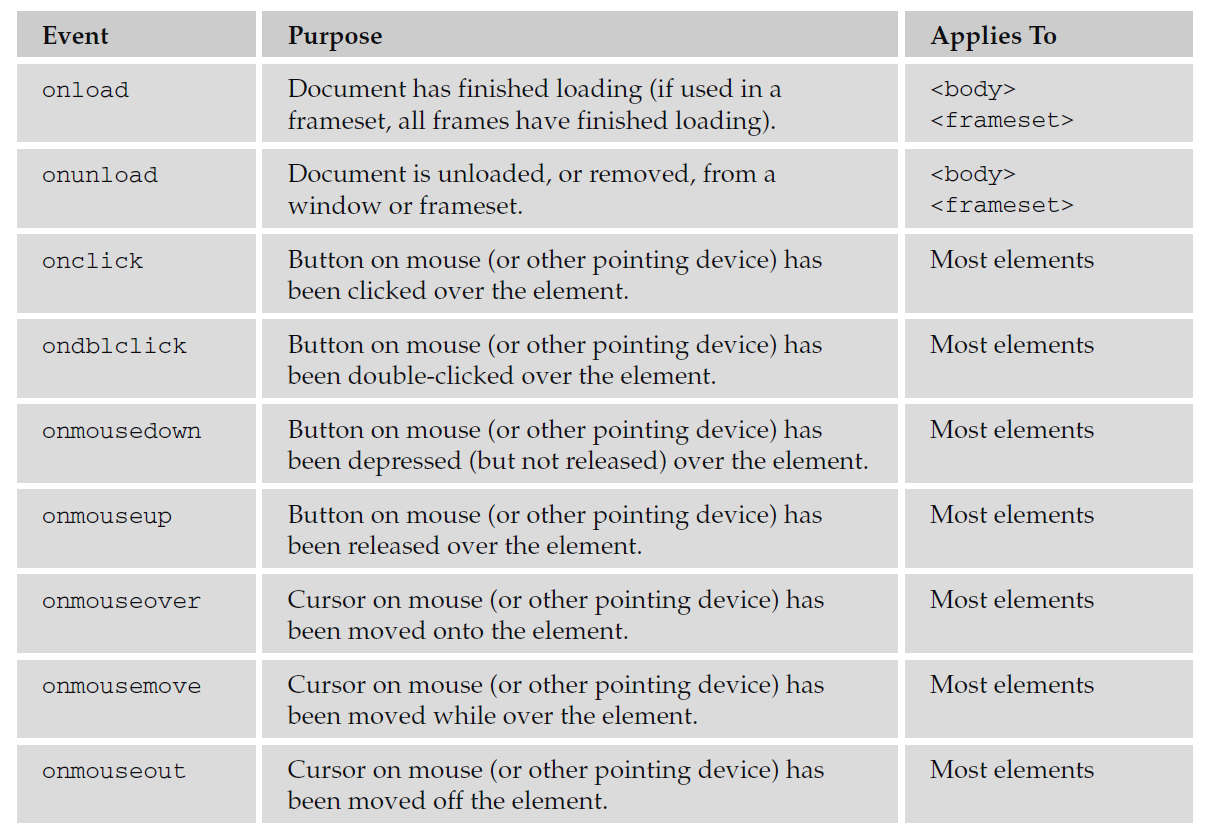
All browsers are expected to support a set of events known as *intrinsic events* such as the onload event, which happens when a page has finished loading, onclick for when a user clicks on an element, and onsubmit for when a form is submitted. These events can be used to trigger a script.

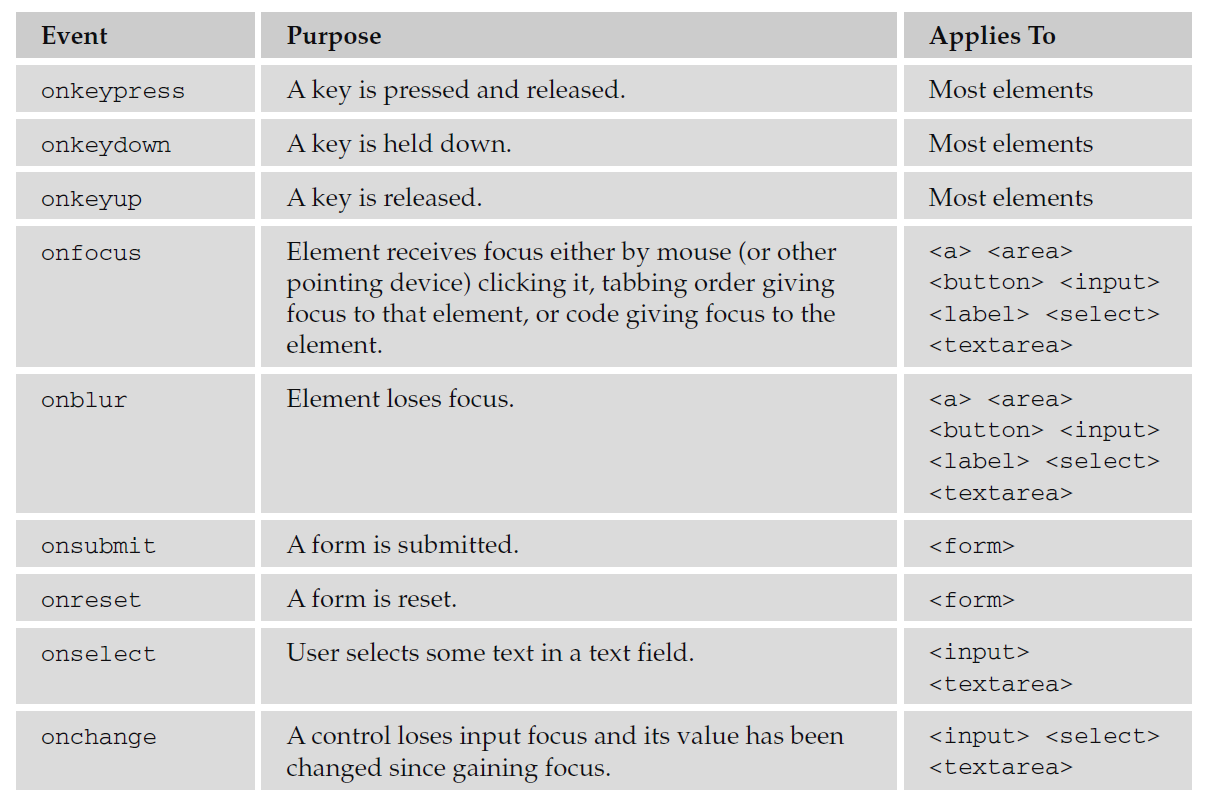
You have already seen event handlers used as attributes on XHTML elements — such as the onclick attribute on an <a> element and the onsubmit attribute on the < form > element. The value of the attribute is the script that should be executed when the event occurs on that element (sometimes this will be a function in the <head> of the document).

There are two types of events that can be used to trigger scripts:

* Window events, which occur when something happens to a window. For example, a page loads or unloads (is replaced by another page or closed) or focus is being moved to or away from a window or frame.
* User events, which occur when the user interacts with elements in the page using a mouse (or other pointing device) or a keyboard, such as placing the mouse over an element, clicking on an element, or moving the mouse off an element.

The table that follows provides a recap of the most common events you are likely to come across.





**Built - in Objects:**

You learned about the document object at the beginning of the chapter and now it is time to see some of the objects that are built into the JavaScript language. You will see the methods that allow you to perform actions upon data, and properties that tell you something about the data.

**String:**

The string object allows you to deal with strings of text. Before you can use a built - in object, you need to create an instance of that object. You create an instance of the string object by assigning it to a variable like so:

myString = new String(‘Here is some bold text’);

The string object now contains the words “ Here is some bold text ” and this is stored in a variable called myString . Once you have this object in a variable, you can write the string to the document or perform actions upon it. For example, the following method writes the string as if it were in a <b> element:

document.write(myString.bold());

*Note that if you viewed the source of this element, it would not actually have the* < b > *element in it; rather, you would see the JavaScript, so that a user who did not have JavaScript enabled would not see these words at all.*

You can check the length of this string like so; the result will be the number of characters including spaces and punctuation:

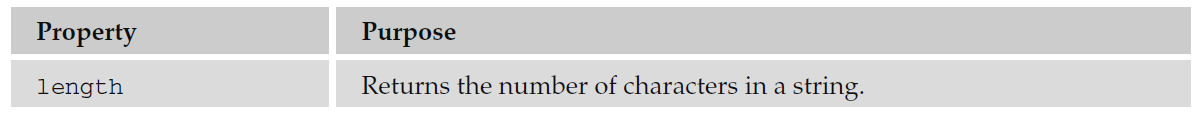
myString = new String(“How many characters are in this sentence?”);

alert(myString.length);

Before you can use the string object, remember you first have to create it and then give it a value.

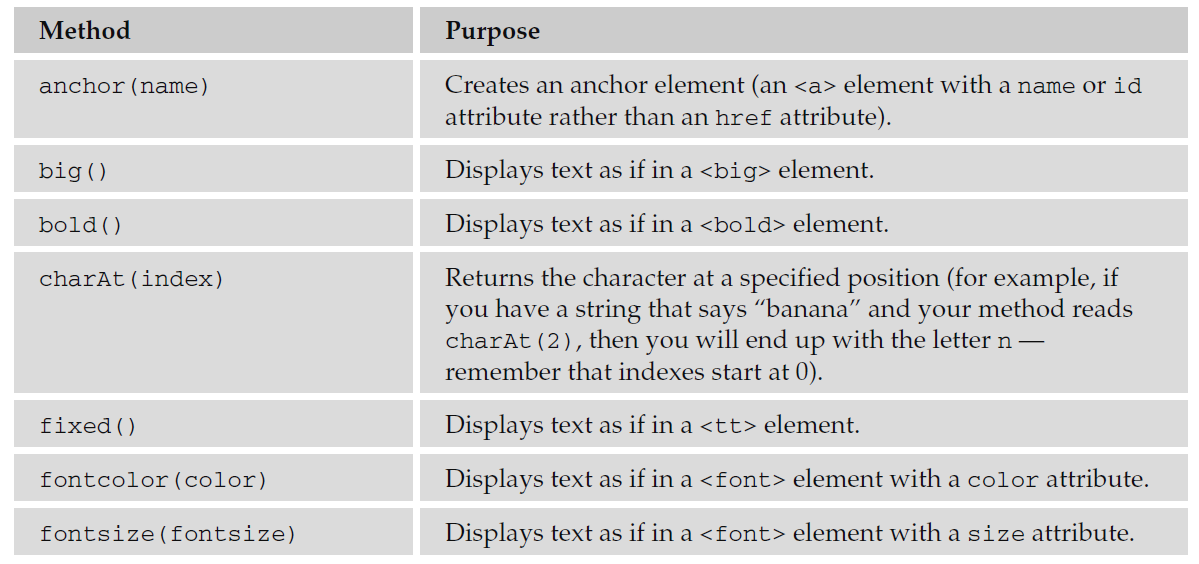
**Properties:**

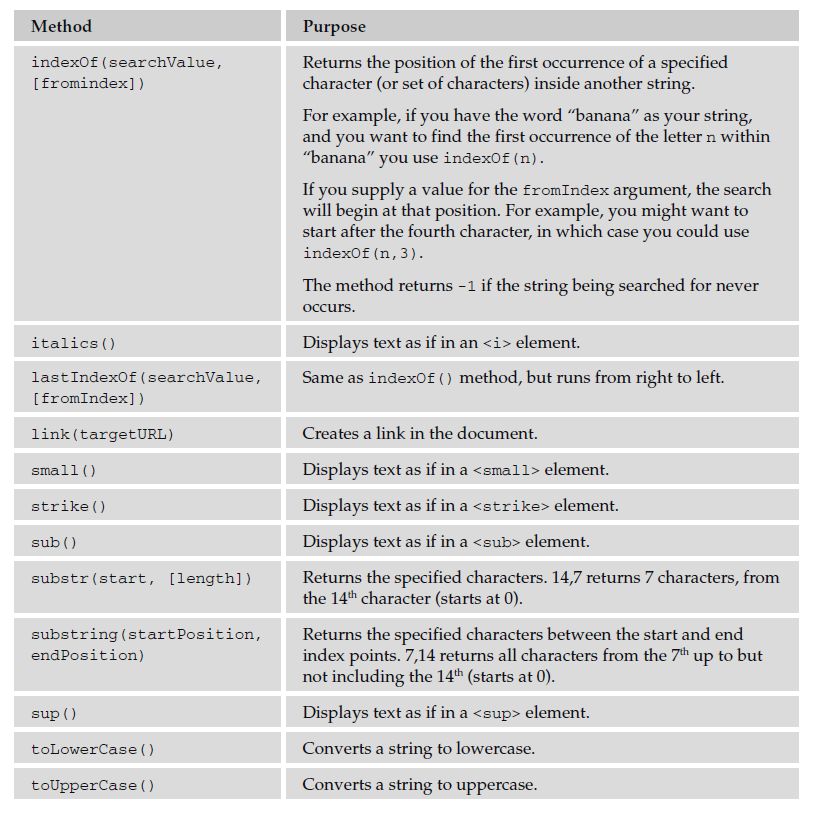
The following table shows the main property for the string object and its purpose.



**Methods:**

The following table lists the methods for the string object and their purposes.





**Date:**

The date object helps you work with dates and times. You create a new date object using the date constructor like so:

new Date();

You can create a date object set to a specific date or time, in which case you need to pass it one of four parameters:

* milliseconds : This value should be the number of milliseconds since 01/01/1970.
* dateString : Can be any date in a format recognized by the parse() method.
* yr\_num , mo\_num , day\_num : Represents year, month, and day.
* yr\_num , mo\_num , day\_num , hr\_num , min\_num , seconds\_num , ms\_num : Represents the years, days, hours, minutes, seconds, and milliseconds.

Here are some examples; the first uses milliseconds and will read Thu Nov 27 05:33:20 UTC 1975 :

var birthDate = new Date(8298400000);

document.write(birthDate);

The second uses a dateString , and will read Wed Apr 16 00:00:00 UTC+0100 1975 :

var birthDate = new Date(“April 16, 1975”);

document.write(birthDate);

The third uses yr\_num , mo\_num , and day\_num , and will read Mon May 12 00:00:00 UTC+0100 1975 :

var birthDate = new Date(1975, 4, 28);

document.write(birthDate);

There are a few things to watch out for:

* The first confusing thing you might notice here is that the number 4 corresponds to the month of May! That makes January 0. Similarly, when working with days, Sunday is treated as 0.
* You might find that you get different time zones than author because he is based in London, so he run on Greenwich Mean Time (GMT) or Coordinated Universal Time (UTC). All the date object’s workings are performed using UTC, even though your computer may display a time that is consistent with your time zone.
* While you can add or subtract dates, your result will end up in milliseconds. For example, if I wanted to find out the number of days until the end of the year, I might use something like this:

var today = new Date();

var newYear = new Date(2010,11,31);

var daysRemaining = (newYear - today);

document.write(daysRemaining);

The problem with this is that you end up with a result that is very long (plus if you read this during 2010 or minus if you read it after 2010). With 86,400,000 milliseconds in each day, you are likely to see a very large figure.

So, you need to divide the daysRemaining by the number of milliseconds in the day (86400000) to find the number of days:

var today = new Date()

var newYear = new Date(2010,11,31);

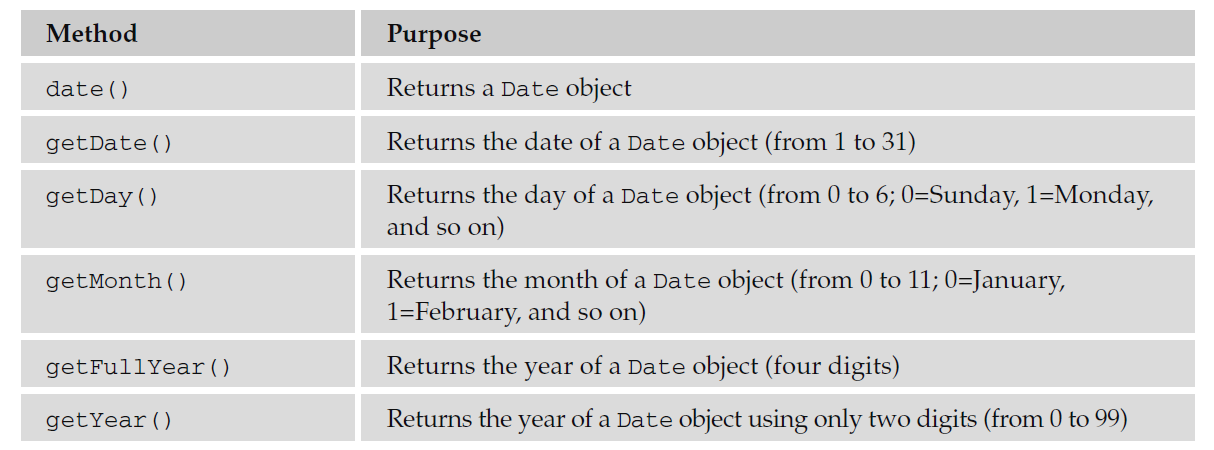
var daysRemaining = (newYear - today);

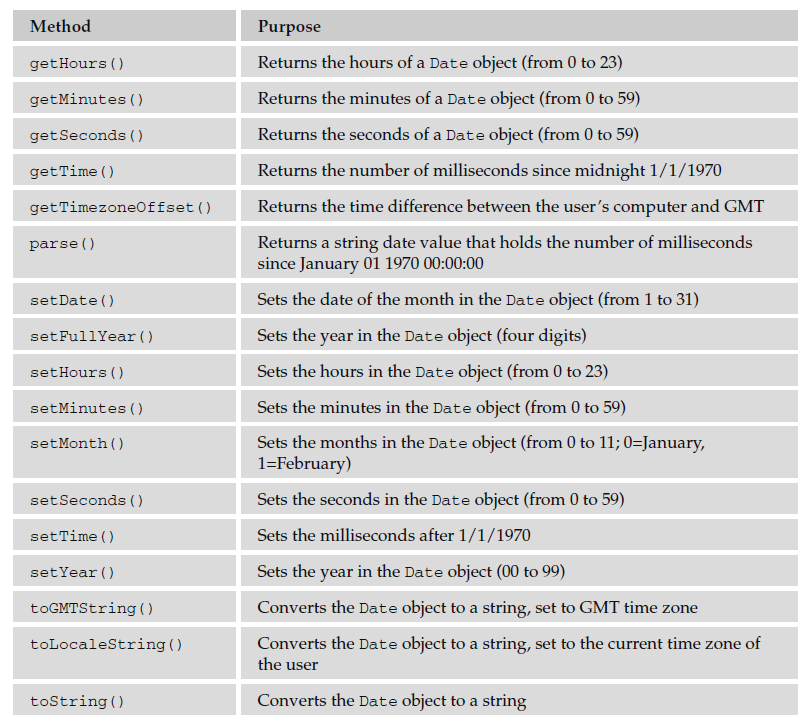
daysRemaining = daysRemaining/86400000;

document.write(daysRemaining);

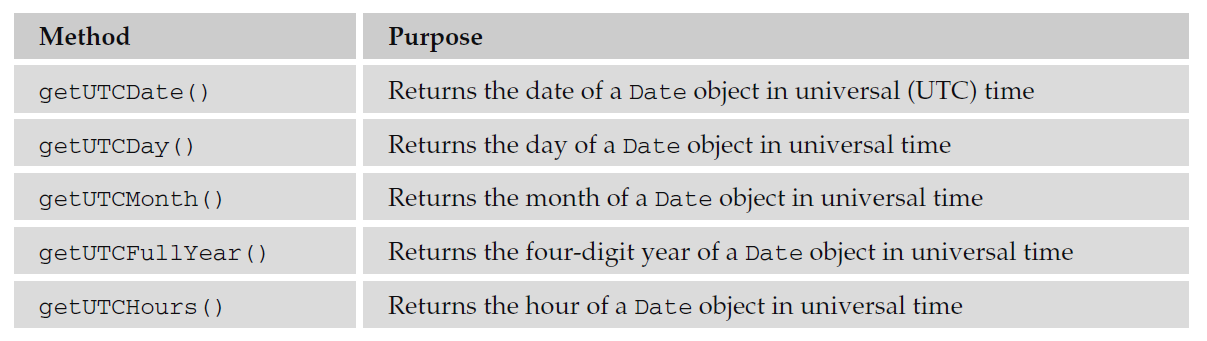
When you use the date object, you need to bear in mind that a user’s computer click may well be inaccurate and the fact that different users could be in various time zones.

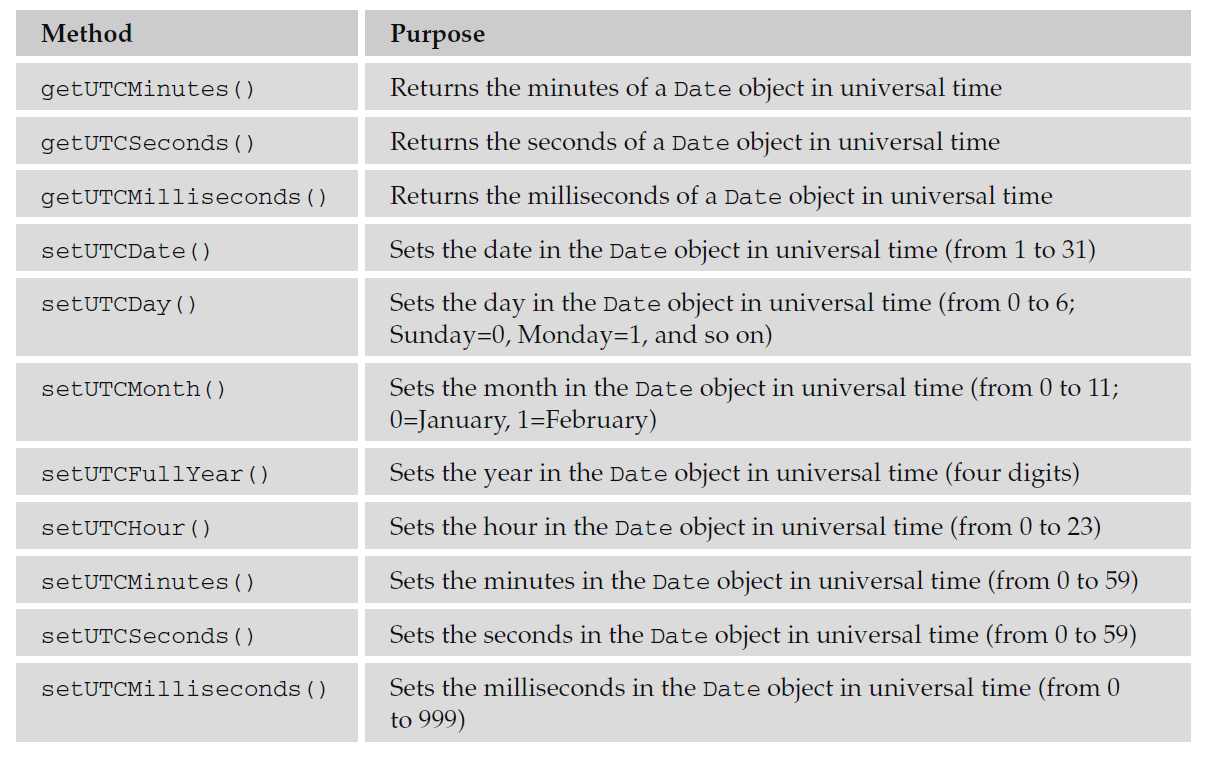
The following table shows some commonly used methods of the date object.





Many of the methods in the table that follows were then added offering support for the universal (UTC) time, which takes the format Day Month Date, hh,mm,ss UTC Year.





**Array:**

An *array* is like a special variable. It ’ s special because it can hold more than one value, and these values can be accessed individually. Arrays are particularly helpful when you want to store a group of values in the same variable rather than having separate variables for each value. You may want to do this because all the values correspond to one particular item, or just for the convenience of having several values in the same variable rather than in differently named variables; or it might be because you do not know how many items of information are going to be stored (for example, you might store the items that would appear in a shopping basket in an array). You often see arrays used in conjunction with loops, where the loop is used to add information into an array or read it from the array.

You need to use a *constructor* with an array object, so you can create an array by specifying either the name of the array and how many values it will hold or by adding all the data straight into the array. For example, here is an array that is created with three items; it holds the names of musical instruments:

instruments = new Array(“guitar”, “drums”, “piano”);

The items in the array can be referred to by a number that reflects the order in which they are stored in the array. The number is an index, so it begins at 0. For example, you can refer to the guitar as instruments[0] , the drums as instruments[1] , and so on.

An array does need to know how many items you want to store in it, but you do not need to provide values for each item in the array when it is created; you can just indicate how many items you want to be able to store (to confuse matters, this value does not start at 0 so it will create three elements not four):

instruments = new Array(3);

This number is stored in the length property of the array object and the contents are not actually assigned yet. If you want to increase the size of an array, you can just assign a new value to the length property that is higher than the current length.

Here is an example that creates an array with five items and then checks how many items are in the array using the length property:

fruit = new Array(“apple”, “banana”, “orange”, “mango”, “lemon”);

document.write(fruit.length);

Here is an example of the toString() method, which converts the array to a string.

document.write(‘These are ‘ + fruit.toString());

Keeping the related information in the one variable tends to be easier than having five variables, such as fruit1, fruit2, fruit3, fruit4, and fruit5. Using one array like this also takes up less memory than storing five separate variables, and in situations when you might have varying numbers of fruit it allows the variable to grow and shrink in accordance with your requirements (rather than creating ten variables, half of which might be empty).

**Methods:**

The table that follows lists the methods of an array:

