JavaScript Native Objects
Broswer Objects

JavaScript Native Objects

So far we have just been looking at what objects are, how to create them, and how to use them. Now, let's take a look at some of the more useful objects that are native to JavaScript, that is, those that JavaScript makes available for us to use.

We won't be looking at all of the native JavaScript objects, just some of the more commonly used ones, namely the String object, the Math object, the Array object, and the Date object.

String Objects

Like most objects, String objects need to be created before they can be used. To create a String object, we can write

```
var string1 = new String("Hello");
var string2 = new String(123);
var string3 = new String(123.456);
```

However, as we have seen, we can also declare a string primitive and use it as if it were a String object, letting JavaScript do the conversion to an object for us behind the scenes. For example

```
var string1 = "Hello";
```

The length Property

The length property simply returns the number of characters in the string. For example

```
var myName = new String("Paul");
document.write(myName.length);
```

will write the length of the string "Paul" (that is, 4) to the page.

The charAt() and charCodeAt() Methods—Selecting a Single Character from a String

The charAt(pos) method returns the character at position 'pos'.

charAt() treats the positions of the string characters as starting at 0, so the first character is at index 0, the second at index 1, and so on.

For example, to find the last character in a string, we could use the code

```
var myString = prompt("Enter some text","Hello World!");
var theLastChar = myString.charAt(myString.length - 1);
document.write("The last character is " + theLastChar);
```

In the first line we prompt the user for a string, with the default of "Hello World!" and store this string in the variable myString.

In the next line, we use the charAt() method to retrieve the last character in the string. We use the index position of (myString.length - 1). Why? Let's take the string "Hello World!" as an example. The length of this string is 12, but the last character position is 11 since the indexing starts at 0. Therefore, we need to subtract one from the length of the string to get the last character position.

In the final line, we write the last character in the string to the page.

The charCodeAt() method is similar in use to the charAt() method, but instead of returning the character itself, it returns a number that represents the decimal character code in the Unicode character set for that character.

For example, to find the character code of the first character in a string, we could write

```
var myString = prompt("Enter some text","Hello World!");
var theFirstCharCode = myString.charCodeAt(0);
document.write("The first character code is " + theFirstCharCode);
```

The above code will get the character code for the character at index position zero in the string given by the user, and write it out to the page.

Ascci values of character go in order so, for example, the letter A has the code 65, B has 66, and so on. Lowercase letters start at 97 (a is 97, b is 98, and so on). Digits go from 48 (for the number 0) to 57 (for the number 9).

Character	***Ascii value***
A	65
B	66
C	67
.	.
.	.
.	.
Z	91
a	97
b	98
.	.
.	.
z	123
0	48
1	49
.	.
.	.
9	58

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
function checkCharType(charToCheck)
{
   var returnValue = "O";
   var charCode = charToCheck.charCodeAt(0);

   if (charCode >= "A".charCodeAt(0) && charCode <= "Z".charCodeAt(0))
   {
      returnValue = "U";

   }
   else if (charCode >= "a".charCodeAt(0) && charCode <= "z".charCodeAt(0))
   {
      returnValue = "L";
   }
   else if (charCode >= "0".charCodeAt(0) && charCode <= "9".charCodeAt(0))
   {
      returnValue = "N";
   }
   return returnValue;
}
</script>
<head>

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

var myString = prompt("Enter some text","Hello World!");
switch (checkCharType(myString))
{
   case "U":
      document.write("First character was upper case");
      break;
   case "L":
      document.write("First character was lower case");
      break;
   case "N":
      document.write("First character was a number");
      break;
   default:
      document.write("First character was not a character or a number");
}
</script>
</body>
</html>
```

The fromCharCode() Method—Converting Character Codes to a String

The method fromCharCode() can be thought of as the opposite to charCodeAt(), in that you pass it a series of comma-separated numbers representing character codes, and it converts them to a single string.

However, the fromCharCode() method is unusual in that it's a *static* method—we don't need to have created a String object to use it with. Instead, we can use the String expression

For example, the following lines put the string "ABC" into the variable myString.

```
var myString;
myString = String.fromCharCode(65,66,67);
```

What is the output of the follwing code?

```
var myString = "";
var charCode;

for (charCode = 65; charCode <= 90; charCode++)
{
      myString = myString + String.fromCharCode(charCode);
}

document.write(myString);
```

The indexOf() and lastIndexOf() Methods—Finding a String Inside Another String

The methods indexOf() and lastIndexOf() are used for searching for the occurrence of one string inside another. A string contained inside another is usually termed a *substring*.

Both indexOf() and lastIndexOf() take two parameters:

- The string you want to find
- The character position you want to start searching from (optional)

If you dont specify the 2nd parameter, function starts searching from position 0.

The return value of indexOf() and lastIndexOf() is the character position in the string at which the substring was found. If the substring is found at the start of the string, then 0 is returned. If there is no match, then the value -1 is returned.

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

var myString = "Hello paul. How are you Paul";
var foundAtPosition;

foundAtPosition = myString.indexOf("Paul");
alert(foundAtPosition);

</script>
```

This code should result in a message box containing the number 24, which is the character position of "Paul". You might be wondering why it's 24, which clearly refers to the second "Paul" in the string, rather than 6 for the first "paul". Well, this is due to case sensitivity again.

We've seen indexOf() in action, but how does lastIndexOf() differ? Well, whereas indexOf() starts searching from the beginning of the string, or the position you specified in the second parameter, and works towards the end, lastIndexOf() starts at the end of the string, or the position you specified, and works towards the beginning of the string.

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

var myString = "Hello Paul. How are you Paul";
var foundAtPosition;

foundAtPosition = myString.indexOf("Paul");

alert(foundAtPosition);

foundAtPosition = myString.lastIndexOf("Paul");
alert(foundAtPosition);

</script>
```

There will be 2 outputs in this example, first output is 6. Because indexOf() starts searching from the beginning it will match “Paul” with the first Paul in string mystring which occurs at position 6. So 6 is returned.

Second output is 24, because lastIndexOf() starts searching from the end. It will start from the end moving towards beginning, in the way it finds “Paul” in the string myString at position 24. So 24 is returned.

The substr() Method—Copying Part of a String

If we wanted to cut out part of a string and assign that cut out part to another variable, we would use the substr() method.

The method substr() takes two parameters: the start position and the end position of the part of the string we want. The second parameter is optional; if you don't include it, all characters from the start position to the end of the string are included.

For example, if our string is "JavaScript" and we want just the text "Java", we could call the method like so:

```
var myString = "JavaScript";
var mySubString = myString.substr(0,4);
alert(mySubString);
```

Why the end position is 4? It helps to think of the parameters as specifying the *length* of the string being returned: the parameters 0 and 4 will return (4 - 0) number of characters starting at and including the character at position 0.

Let us take an example where I can use substr() and lastIndexOf() function together.

Suppose I want to extract the name of the file from the URL "http://mywebsite/temp/myfile.htm".

```
1. var fileName = window.location.href;
2. var pos =  fileName.lastIndexOf("\");
3. fileName = fileName.substr(pos+1);
4. document.write("The file name of this page is " + fileName);
```

Line1 stores "http://mywebsite/temp/myfile.htm" in variable fileName. window.location.href returns the URL from the addressbar of browser.

We can use lastIndexOf() function to find last occurrence of "\" in variable fileName. Line 2 find the last occurrence of "\" in variable fileName and stores the result in pos.

Then we extract the string following last "\". We use substr function for that in line3. Finally we print the extracted substring.

The toLowerCase() and toUpperCase() Methods—Changing the Case of a String

If we want to change the case of the string, javascript provides 2 functions for that toLowerCase() and toUpperCase.

```
var myString = "I Donot Care About Case".

window.alert(myString.toLowerCase());

window.alert(myString.toUpperCase());
```

There will be 2 outputs, first output will print "i dont care about case".

Second output will print "I DONT CARE ABOUT CASE".

Name of the function	***Description***
big:	Returns the string in big tags.
blink:	Returns the string in blink tags.
bold	Returns the string in b tags.
fixed	Returns the string in tt tags.
fontcolor	Returns the string in font tags with the color attribute set.
fontsize	Returns the string in font tags with the size attribute set.
italics	Returns the string in i tags.
small	Returns the string in small tags.
strike	Returns the string in strike tags.
sub	Returns the string in sub tags (subscript effect).
super	Returns the string in sup tags (superstring effect).

Below is the code to understand the above methods.

```
<body>

        <script language="JavaScript">
        <!--

                var str1="Hello";
                document.write("This is normal text"+"<br>");
                document.write(str1.big()+"<br>");
                document.write(str1.bold()+"<br>");
                document.write(str1.fixed()+"<br>");
                document.write(str1.fontcolor("blue")+"<br>");
                document.write(str1.fontsize(30)+"<br>");
                document.write(str1.italics()+"<br>");
                document.write(str1.small()+"<br>");
                document.write("How are you"+str1.sub()+"<br>");
                document.write("2"+str1.sup()+"<br>");
                document.write(str1.toLowerCase()+"<br>");
                document.write(str1.toUpperCase()+"<br>");
        // -->
        </script>

</body>
```

Output should be

This is normal text
Hello
Hello
`Hello`
Hello

Hello

Hello
Hello
How are you$_{\text{Hello}}$
2$^{\text{Hello}}$
hello
HELLO

If we want to appply more than one formatting option, you can specify them in a single statement. To understand this, refer to example below.

```
<body>
        <script language="JavaScript">
        <!--

                var str1="Hello";
                document.write("This is normal text"+"<br>");
                document.write(str1.big().bold().italics().toUpperCase());
        // -->
        </script>

</body>
```

Output should be
This is normal text
HELLO

Math Object

The Math object is a little unusual in that JavaScript automatically creates it for you. There's no need to declare a variable as a Math object or define a new Math object before being able to use it, making it a little bit easier to use.

Properties:

- •Math.E

Math.LN10

- •Math.LN2
- •Math.LOG10E
- •Math.LOG2E
- •Math.PI

Functions

•Math.abs(x)	:	Returns the absolute value of x.
•Math.ceil(x)	:	Rounds a number up.
•Math.floor(x)	:	Rounds a number down.
•Math.random()	:	Returns a random number between 0 and 1.
•Math.round(x)	:	Math.round(25.9) = 26 and Math.round(25.2) = 25
•Math.sqrt(x)	:	Calculates the square root of x.
•Math.max(x,y)	:	Returns the larger of two numbers
•Math.min(a,b)	:	Returns the smaller of two number
•Math.pow(x,y)	:	Returns the y raise to the power x.
•Math.log(x)	:	Returns the natural log of x.
•math.exp(x)	:	Returns the exponential of x.
•Math.sin(x)	:	Returns the sine of x.
•Math.cos(x)	:	Returns the cos of x.
•Math.tan(x)	:	Returns the tan of x.
•Math.asin(x)	:	Returns the arc sine of x in radians
•Math.acos(x)	:	Returns the arc cosine of x in radians
•Math.atan(x)	:	Returns the arc tan of x in radians.

The properties of the Math object include some useful math constants, such as the PI property (giving the value 3.14159 and so on).

The abs() Method

The abs() method returns the absolute value of the number passed as its parameter. Essentially, this means that it returns the positive value of the number. So -1 is returned as 1, -4 as 4, and so on. However, 1 would be returned as 1 because it's already positive.

For example, the following code would write the number 101 to the page.

```
var myNumber = -101;
document.write(Math.abs(myNumber));
```

The ceil() Method

The ceil() method always rounds a number up to the next largest whole number or integer. So 10.01 becomes 11, and –9.99 becomes –9 (because –9 is greater than –10). The ceil() method has just one parameter, namely the number you want rounded up.

For example, the following code writes two lines in the page, the first containing the number 102 and the second containing the number 101.

```
var myNumber = 101.01;
document.write(Math.ceil(myNumber) + "<br>");
document.write(parseInt(myNumber));
```

The floor() Method

Like the ceil() method, the floor() method removes any numbers after the decimal point, and returns a whole number or integer. The difference is that floor() always rounds the number down. So if we pass 10.01 we would be returned 10, and if we pass –9.99, we will see –10 returned.

The round() Method

The round() method is very similar to ceil() and floor(), except that instead of always rounding up or always rounding down, it rounds up only if the decimal part is .5 or greater, and rounds down otherwise.

For example

```
var myNumber = 44.5;
document.write(Math.round(myNumber) + "<br>");

myNumber = 44.49;
document.write(Math.round(myNumber));
```

would write the numbers 45 and 44 to the page.

Parameter	***parseInt() returns***	***ceil() returns***	***floor() returns***	***round() returns***
10.25	10	11	10	10
10.75	10	11	10	11
10.5	10	11	10	11
–10.25	–10	–10	–11	–10
–10.75	–10	–10	–11	–11
–10.5	–10	–10	–11	–10

The random() Method

The random() method returns a random floating-point number in the range between 0 and 1, where 0 is included and 1 is not. This can be very useful for displaying random banner images or for writing a JavaScript game.

```
<html>
<body>
<script language="JavaScript" type="text/javascript">
var throwCount;
var diceThrow;
for (throwCount = 0; throwCount < 10; throwCount++)
{
   diceThrow = (Math.floor(Math.random() * 6) + 1);
   document.write(diceThrow + "<br>");
}

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

We want diceThrow to be between 1 and 6. The random() function returns a floating-point number between 0 and just under 1. By multiplying this number by 6, we get a number between 0 and just under 6. Then by adding 1, we get a number between 1 and just under 7. By using floor() to always round it down to the next lowest whole number, we can ensure that we'll end up with a number between 1 and 6.

If we wanted a random number between 1 and 100, we would just change the code so that Math.random() is multiplied by 100 rather than 6.

The pow() Method

The pow() method raises a number to a specified power. It takes two parameters, the first being the number you want raised to a power, and the second being the power itself. For example, to raise 2 to the power of 8 (that is, to calculate 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2), we would write Math.pow(2,8)—the result being 256.

Number Object

As with the String object, Number objects need to be created before they can be used. To create a Number object, we can write

```
var firstNumber = new Number(123);
var secondNumber = new Number('123');
```

However, as we have seen, we can also declare a number as primitive and use it as if it were a Number object, letting JavaScript do the conversion to an object for us behind the scenes. For example

```
var myNumber = 123.765;
```

The toFixed() Method

The toFixed() method is new to JavaScript 1.5 and Jscript 5.5—so basically it's available in Netscape 6+ and IE 5.5+ only. The method cuts a number off after a certain point. Let's say we wanted to display a price after sales tax. If our price is $9.99 and sales tax is 7.5%, that means the after-tax cost will be $10.73925. Well, this is rather an odd amount for a money transaction—what we really want to do is fix the number to no more than two decimal places. Let's create an example.

```
var itemCost = 9.99;
var itemCostAfterTax = 9.99 * 1.075;
document.write("Item cost is $" + itemCostAfterTax + "<br>");
itemCostAfterTax = itemCostAfterTax.toFixed(2);
document.write("Item cost fixed to 2 decimal places is " + itemCostAfterTax);
```

The first document.write() will output the following to the page:

```
Item cost is $10.73925
```

However, this is not the format we want; instead we want two decimal places, so on the next line, we enter

```
itemCostAfterTax = itemCostAfterTax.toFixed(2);
```

We use the toFixed() method of the Number object to fix the number variable that itemCostAfterTax holds to two decimal places. The method's only parameter is the number of decimal places we want our number fixed to.

This line means that the next document.write displays

```
Item cost fixed to 2 decimal places is $10.74
```