Internet Software Technologies
JavaScript – part three

IMCNE

A.A. 2008/09

Gabriele Cecchetti

The *Number* object

- The JavaScript Number object does not allow you to set specific number types (like integer, short, long or double).
- In JavaScript all numbers are 64bit floating point numbers and have a number range from 5e-324 (negative) to 1.7976931348623157e+308 (positive).

Number object syntax

- The Number object is an object wrapper for primitive numeric values.
- **Syntax for creating a Number object:**

```
var myNum = new Number(number);
```

- **Note:** If the number parameter cannot be converted into a number, it returns NaN.

Number object: properties

- Useful constants

`NaN`	Represents "Not-a-number" value
`MAX_VALUE`	Returns the largest possible value in JavaScript
`MIN_VALUE`	Returns the smallest possible value in JavaScript
`NEGATIVE_INFINITY`	Represents a value that is <= `MIN_VALUE`
`POSITIVE_INFINITY`	Represents a value that is >= `MAX_VALUE`

- Other properties:

`constructor`	Returns a reference to the Number function that created the object
`prototype`	Allows you to add properties and methods to the object

Number object: methods

`toExponential()`	Converts the value of the object into an exponential notation
`toFixed()`	Formats any number for "x" number of trailing decimals. The number is rounded up, and "0"s are used after the decimal point if needed to create the desired decimal length.
`toLocaleString()`	
`toPrecision()`	Formats any number so it is of "x" length. Also called significant digits. A decimal point and "0"s are used if needed to create the desired lenght
`toString()`	Converts the Number object into a string
`valueOf()`	Returns the value of the Number object

Number object: examples (1/2)

- In this example we will convert a number to an exponential notation:

```
Show 10,000 as an exponential notation:
  <script type="text/javascript">
  var num = new Number(10000);
  document.write (num.toExponential(1)); </script>
```

- The output of the code above will be:

```
Show 10,000 as an exponential notation: 1.0e+4
```

Number object: examples (2/2)

- **Number.toFixed()**

```
var profits=2489.8237
  profits.toFixed(3) //returns 2489.824 (round up)
  profits.toFixed(2) //returns 2489.82
  profits.toFixed(7) //returns 2489.8237000 (padding)
```

- **Number.toPrecision()**

```
var anumber=123.45
  anumber.toPrecision(6) //returns 123.450 (padding)
  anumber.toPrecision(4) //returns 123.5 (round up)
  anumber.toPrecision(2) //returns 1.2e+2 (you figure
  it out!)
```

toPrecision() is useful if your number must be of a certain length.

Browser considerations about toFixed() and toPrecision() methods.

- This methods are JavaScript 1.5 version methods.
- This means that they'll only work in IE5.5+ and NS6+ and any other browser supporting this version.
- To ensure that these methods will degrade well will other legacy (older) browser we need to do some browser detection. For example:

```
var profits=2489.8237
  if (profits.toFixed)    //if browser supports
                          //toFixed() method
     profits.toFixed(2)
```

The *Math* object

- The Math object allows you to perform mathematical tasks.
- The Math object includes several mathematical constants and methods.
- Syntax:

```
var myvar1 = Math.someMathProperty
var myvar2 = Math.someMathMethod()
```

- All properties and methods of Math can be called by using Math as an object without creating it.

Math properties

`E`	The constant of E, the base of natural logarithms.
`LN2`	The natural logarithm of 2.
`LN10`	The natural logarithm of 10.
`LOG2E`	Base 2 logarithm of E.
`LOG10E`	Base 10 logarithm of E.
`PI`	Returns PI.
`SQRT1_2`	Square root of 1/2.
`SQRT2`	Square root of 2.

- Example:

```
var pi_value=Math.PI;
```

Math methods

`abs(x)`	Returns absolute value of x.
`acos(x)`	Returns arc cosine of x in radians.
`asin(x)`	Returns arc sine of x in radians.
`atan(x)`	Returns arc tan of x in radians.
`atan2(y,x)`	Counterclockwise angle between x axis and point (x,y).
`ceil(x)`	Returns the smallest integer greater than or equal to x.
`cos(x)`	Returns cosine of x, where x is in radians.
`exp(x)`	Returns ex
`floor(x)`	Returns the largest integer less than or equal to x.
`log(x)`	Returns the natural logarithm (base E) of x.
`max(a,b)`	Returns the larger of a and b.
`min(a,b)`	Returns the lesser of a and b.
`pow(x,y)`	Returns x^y
`random()`	Returns a pseudorandom number between 0 and 1.
`round(x)`	Rounds x up or down to the nearest integer.
`sin(x)`	Returns the sin of x, where x is in radians.
`sqrt(x)`	Returns the square root of x.
`tan(x)`	Returns the tan of x, where x is in radians.

Math methods examples (1/2)

- //calculate e^5

```
Math.exp(5)
```

- //calculate cos(2PI)

```
Math.cos(2*Math.PI)
```

- If you intend to invoke Math multiple times in your script, a good statement to remember is "with." Using it you can omit the "Math." prefix for any subsequent Math properties/methods:

```
with (Math){
   var x= sin(3.5)
   var y=tan(5)
   var result=max(x,y)
}
```

Math methods examples (2/2)

- Round a number to the nearest integer:

```
document.write(Math.round(4.7));
```

→ 5

- Return a random number between 0 and 1:

```
document.write(Math.random());
```

→ 0.37916376078027325

- Return a random number between 0 and 10:

```
document.write(Math.floor(Math.random()*11));
```

→ 9

The *RegExp* object

- The regular expression object describes a pattern of characters.
- **Syntax for creating a RegExp object:**

```
var txt=new RegExp(pattern,attributes);
```

where:

  - pattern specifies the pattern of the regular expression
  - attributes specifies global ("g"), case-insensitive ("i"), and multiline matches ("m")

RegExp properties

`global`	Specifies if the "g" modifier is set
`ignoreCase`	Specifies if the "i" modifier is set
`input`	The string on which the pattern match is performed
`lastIndex`	An integer specifying the index at which to start the next match
`lastMatch`	The last matched characters
`lastParen`	The last matched parenthesized substring
`leftContext`	The substring in front of the characters most recently matched
`multiline`	Specifies if the "m" modifier is set
`prototype`	Allows you to add properties and methods to the object
`rightContext`	The substring after the characters most recently matched
`source`	The text used for pattern matching

RegExp methods and usage (1/3)

`RegExp.exec(string)`

Applies the RegExp to the given string, and returns the match information. Example:

```
var match = /s(amp)le/i.exec("Sample text")
```

→ match then contains ["Sample","amp"]

`RegExp.test(string)`

Tests if the given string matches the Regexp, and returns true if matching, false if not. Example:

```
var match = /sample/.test("Sample text")
```

→ match then contains false

RegExp methods and usage (2/3)

`String.match(pattern)`

Matches given string with the RegExp. With g flag returns an array containing the matches, without g flag returns just the first match or if no match is found returns null. Example:

```
var str = "Watch out for the rock!".match(/r?or?/g)
```

→ str then contains ["o","or","ro"]

`String.search(pattern)`

Matches RegExp with string and returns the index of the beginning of the match if found, -1 if not. Example:

```
var ndx = "Watch out for the rock!".search(/for/)
```

→ ndx then contains 10

RegExp methods and usage (2/3)

`String.replace(pattern,string)`

Replaces matches with the given string, and returns the edited string. Example:

```
var str = "Liorean said: My name is
  Liorean!".replace(/Liorean/g,'Big Fat Dork')
```

→ str then contains "Big Fat Dork said: My name is Big Fat Dork!"

`String.split(pattern)`

Cuts a string into an array, making cuts at matches. Example:

```
var str = "I am confused".split(/\s/g)
```

→ str then contains ["I","am","confused"]

The *Function* object

- The **Function** object permits a function to have methods and properties associated with it.
- Note that JavaScript treats the function, itself, as a data type that has a value. To return that value, the function must have a **return** statement.
- When a **Function** object is created by using the **Function** constructor, it is evaluated each time. This is not as efficient as the alternative method of declaring a function using the **function** statement where the code is compiled.
- Syntax:

```
new Function([arg1[, arg2[, ... argN]],] functionBody)
```


Function object: examples (1/2)

- Example: calculate the average of two numbers, and then displays that average:

```
var twoNumAverage = new Function("x", "y", "return (x + y)/2")
  document.write(twoNumAverage(3,7))
  var average = twoNumAverage(12,17)
```

- If a function changes the value of a parameter, this change is not reflected globally or in the calling function, unless that parameter is an object, in which case any changes made to any if its properties will be reflected outside of it.

Function object: examples (2/2)

- Example: an object called TaxiCo is created with a property containing the size of the taxi fleet. A **Function** object called AddCar is then created which allows a user to alter the size of the Fleet property to reflect an increase in the number of cars, and then an instance of this function adds 2 to the Fleet property with the final line of code displaying the new size of the taxi fleet:

```
taxiCo = {name:"City Cabs", phone:"321765", fleet:17}
  var addCar = new Function("obj", "x", "obj.fleet =
  obj.fleet + x")
  addCar(taxiCo, 2)
  document.write("New fleet size = " + taxiCo.fleet)
→ New fleet size = 19
```


Function object properties

`arguments` It is an array of all the arguments passed to a function.

`arguments.callee` It can only be used within the body of a function and returns a string specifying what that function is.

`arguments.caller` It returns the number of arguments passed to a function.

`arguments.length` It returns the number of arguments passed to a function.

`arity` It specifies the number of arguments expected by a function.

`constructor` It specifies the function that creates an object's prototype. It is a direct reference to the function itself rather than a string containing the function's name.

`length` It specifies the number of arguments expected by a function.

`prototype Property` It is a value from which all instances of an object are constructed, and which also allows you to add other properties and methods to an object.

Function object methods

`apply()`	It allows you to apply to a function a method from another function.
`call()`	It allows you to call a method from another object
`toSource()`	It creates a string representing the source code of the function.
`toString()`	It returns a string which represents the source code of a function. (it is like **valueOf)**