SOFTWARE GUILD

Reference Types: Part 2

.NET Cohort

Coding Bootcamp

Lesson Goals

1. Learn about the Date type
2. Learn about Regex
3. Learn about Function types

The Date Type

JavaScript dates store the number of milliseconds that have elapsed since midnight on 1/1/1970 UTC. It can store dates 285,616 years before or after 1/1/1970.

Dates can be declared simply by writing:

```
var now = new Date()
```

Dates are automatically set to the current date/time.

Creating Dates

JavaScript provides Date.parse() and Date.UTC() to create specific dates. You can specify date information in a few formats:

1. month/date/year (6/13/2013)
2. month_name date, year (June 13, 2013)
3. add time information (Tue May 25 2004 00:00:00 GMT-0700)
4. ISO specification (YYYY-MM-DDTHH:mm:ss.sssZ)

Passing an invalid string to parse() will set the variable to NaN.

Formatting Dates

The Date type gives us a few ways to format dates:

- toDateString(): displays day of week, month, day of month, and year
- toTimeString(): displays hours, minutes, seconds, and time zone
- toLocaleDateString(): displays day of week, month, day of month, and year in locale-specific format
- toLocaleTimeString(): displays hours, minutes, seconds in locale-specific format
- toUTCString(): displays the complete UTC date

Each of these method's outputs varies by browser. So, if you want all browsers to look the same, you can't use them.

Other Date Methods

METHOD	DESCRIPTION
getTime()	Returns the milliseconds representation of the date
setTime(milliseconds)	Sets the milliseconds of the data, thus changing the date
getFullYear()	Returns the four-digit year
setFullYear(year)	Sets the year of the date, must provide 4 digits
getMonth()	Returns the month of the date; 0=January, 11=December
setMonth(month)	Sets the month; 0=January, 11=December
getDate()	Returns the day of month (1-31) of the date
setDate(date)	Sets the day of month for the date; if the number is higher than the number of days in the month, it adds months
getDay()	Returns the day of week (0=Sunday)

SOFTWAREGUILD

More Date Methods

METHOD	DESCRIPTION
getHours	Returns the hours between 0 and 23
setHours(hours)	Sets the hours; if > 23, adds days to compensate
getMinutes	Returns the minutes between 0 and 59
setMinutes(minutes)	Sets the minutes; if > 59, adds hours to compensate
getSeconds()	Returns the seconds between 0 and 59
setSeconds(seconds)	Sets the seconds; if > 59, adds minutes to compensate
getMilliseconds()	Returns the milliseconds between 0 and 999
setMilliseconds(milliseconds)	Sets the milliseconds, adds seconds to compensate for large numbers

The RegExp Type

JavaScript regular expression syntax is similar to Perl and is quite easy to read. We can create a regular expressions by using the /pattern/options syntax.

Options are:

- g: Global mode: the pattern will be applied to the whole string and not just the first match
- i: Case insensitive mode
- m: Multiline mode: normally, stops at the end of a line, but this continues it

Some Basic Patterns

```
/*
* Match all instances of “at” in a string.
*/
var pattern1 = /at/g;

/*
* Match the first instance of “bat” or “cat”, regardless of case.
*/
var pattern2 = /[bc]at/i;

/*
* Match all three-character combinations ending with ”at”, regardless of case.
*/
var pattern3 = /.at/gi;
```

Regex Meta Characters

Like most regex languages, there are some special characters to modify behavior:

([{ \ ^ $ | }]) ? * + .

So, if you are literally indicating one of these characters, you have to escape it with a \

Example: Literals and Metas

```
/*
* Match the first instance of “bat” or “cat”, regardless of case.
*/
var pattern1 = /[bc]at/i;

/*
* Match the first instance of ”[bc]at”, regardless of case.
*/
var pattern2 = /\[bc\]at/i;

/*
* Match all three-character combinations ending with ”at”, regardless of case.
*/
var pattern3 = /.at/gi;

/*
* Match all instances of ”.at”, regardless of case.
*/
var pattern4 = /\.at/gi;
```

Evaluating Patterns

Here we have a non-global and a global pattern.

The first, non-global pattern always returns the first match (cat) after you execute (exec).

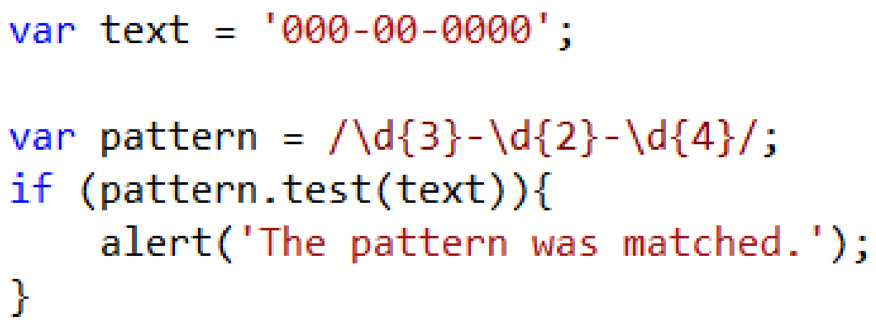
The second, global pattern moves to the next match every time it is executed.

Notice that pattern2's lastIndex is incremented in global mode.

```
var text = 'cat, bat, sat, fat';

var pattern1 = /.at/;

var matches = pattern1.exec(text);
alert(matches.index); //0
alert(matches[0]); //cat
alert(pattern1.lastIndex); //0

matches = pattern1.exec(text);
alert(matches.index); //0
alert(matches[0]); //cat
alert(pattern1.lastIndex); //0


var pattern2 = /.at/g;

var matches = pattern2.exec(text);
alert(matches.index); //0
alert(matches[0]); //cat
alert(pattern2.lastIndex); //3

matches = pattern2.exec(text);
alert(matches.index); //5
alert(matches[0]); //bat
alert(pattern2.lastIndex); //8
```

Testing for a Match

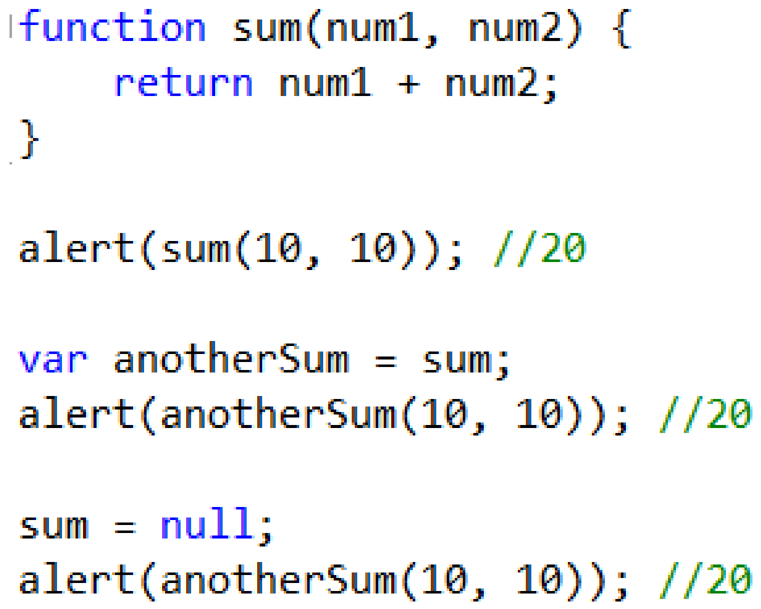
The test() method of a pattern returns true/false whether a pattern is matched:



```
var text = '000-00-0000';

var pattern = /\d{3}-\d{2}-\d{4}/;
if (pattern.test(text)){
    alert('The pattern was matched.');
}
```

Functions as Types?

Functions can actually be assigned to variables. A function name is just a pointer to a function, so we can repoint them or have multiple names for a single function.



```
function sum(num1, num2) {
    return num1 + num2;
}

alert(sum(10, 10)); //20

var anotherSum = sum;
alert(anotherSum(10, 10)); //20

sum = null;
alert(anotherSum(10, 10)); //20
```

Functions as Values

Consider the code to the right.

We can pass functions as parameters to other functions because they can be stored in variables.

This is the basis behind some of the array functionality we discussed previously.

```
function callSomeFunction(someFunction, someArgument) {
    return someFunction(someArgument);
}

function add10(num) {
    return num + 10;
}

var result1 = callSomeFunction(add10, 10);
alert(result1); //20

function getGreeting(name) {
    return 'Hello, ' + name;
}

var result2 = callSomeFunction(getGreeting, 'Nicholas');
alert(result2); //"Hello, Nicholas"
```

Example: Function in a Function

```
function createComparisonFunction(propertyName) {
    return function (object1, object2) {
        var value1 = object1[propertyName];
        var value2 = object2[propertyName];
        if (value1 < value2) {
            return -1;
        } else if (value1 > value2) {
            return 1;
        } else {
            return 0;
        }
    };
}

var data = [{name: 'Zachary', age: 28}, {name: 'Nicholas', age: 29}];

data.sort(createComparisonFunction('name'));
alert(data[0].name); //Nicholas

data.sort(createComparisonFunction('age'));
alert(data[0].name); //Zachary
```

Using this Inside a Function

We have to be careful using the *this* keyword inside a function because of the scope.

By default, any function in the global scope will set *this* to the window.

We can use the call method to explicitly set *this* to another object, as in the example to the right.

This can have interesting effects for reuse, but it’s best as a beginner to just pass the object along as a parameter.

Framework authors use call, apply, and bind quite a bit behind the scenes.

```
window.color = 'red';
var o = { color: 'blue' };

function sayColor(){
    alert(this.color);
}

sayColor(); //red
sayColor.call(this); //red
sayColor.call(window); //red
sayColor.call(o); //blue
```

NOW SOME USEFUL METHODS

Numbers

METHOD	DESCRIPTION	EXAMPLES

toString()	Returns the string version of a number in a specified format
	
	Passing in a number displays it in binary, octal, hex, etc.
		var num = 10
		num.toString() // 10
		num.toString(2) // 1010
		num.toString(8) // 12
		num.toString(10) // 10
		num.toString(16) // a

toFixed()	Returns a string with specified number of decimal places	var num = 10.005 num.toFixed(2) // 10.01

toExponential()	Returns the e-notation	var num = 10;
		num.toExponential(1)
		// 1.0e+1

SOFTWARE GUILD

Strings

METHOD / PROPERTY	DESCRIPTION	EXAMPLE
length	Returns the length, in characters, of the string	var s1 = “hello”; s1.length; // 5
charAt()	Returns the character at a position	var s1 = “hello”; s1.charAt(1); // e
concat()	Puts two strings together	var s1 = “hello”; var s2 = s1.concat(“ world”);

slice()	Cut off all characters outside a range	var s1 = “hello world”;
		s1.slice(3); // “lo world”
		s1.slice(3, 7); // lo w”

substring()	Same as slice	
substr()	Second param is the number of characters	var s1 = “hello world”; s1.substr(3, 7); // “lo worl”

Strings, Continued

METHOD / PROPERTY	DESCRIPTION	EXAMPLE

indexOf()	Finds the position of a character in a string, from the beginning	var s1 = “hello world”;
		s1.indexOf(“o”); // 4
		s1.indexOf(“o”, 6); // 7

lastIndexOf()	Finds the position of a character in a string from the end	var s1 = “hello world”;
		s1.lastIndexOf(“o”); // 7
		s1.lastIndexOf(“o”, 6); // 4

trim()	Removes all leading and trailing spaces	var s1 = “ hello ”; var s2 = s1.trim(); // “hello”

toUpperCase()
toLowerCase()
	Converts a string to upper- or lowercase	var s1 = “hello world”;
		var s2 = s1.toUpperCase();
		// “HELLO WORLD”

SOFTWARE GUILD

Even More String Methods

METHOD / PROPERTY	DESCRIPTION	EXAMPLE

match()	Regex on the string	var text = “cat, bat, sat, fat”;
		var pattern = /.at/;
		
		var matches = text.match(pattern);
		matches.index; //0
		matches[0]; //”cat”
		pattern.lastIndex; //0

search()	Returns the position of a string	var text = “cat, bat, sat, fat”; var pos = text.search(/at/); //1
replace()	Replaces the first string with the second; specify global or only first match	var text = “cat, bat, sat, fat”; var result = text.replace(“at”, “ond”); //”cond, bat, sat, fat” result = text.replace(/at/g, “ond”); //”cond, bond, sond, fond”

SOFTWARE GUILD

And Finally…

METHOD / PROPERTY	DESCRIPTION	EXAMPLE

split()	splits a string into an array; optionally, can specify a number of elements to return	var colorText = "red,blue,green,yellow";
		
		var colors1 = colorText.split(",");
		//["red", "blue", "green", "yellow"]
		
		var colors2 = colorText.split(",", 2);
		//["red", "blue"]

SOFTWARE GUILD

The Math Object

METHOD / PROPERTY	DESCRIPTION	EXAMPLE
Math.E	The value of e, the base of logarithms	
Math.LN10	The natural log of 10	
Math.LN2	The natural log of 2	
Math.PI	The value of pi	

Math.min()
Math.max()
	Returns the lowest or highest number in a group/array	var min = Math.min(3, 54, 32, 16);
		alert(min); //3
		
		var max = Math.max(3, 54, 32, 16);
		alert(max); //54
		
		var values = [1, 2, 3, 4, 5, 6, 7, 8];
		var max = Math.max.apply(Math, values);

Math, Continued

METHOD / PROPERTY	DESCRIPTION	EXAMPLE
Math.ceil()	Always round up	Math.ceil(25.1); // 26
Math.floor()	Always round down	Math.floor(25.9); // 25
Math.round()	"School rounding"	Math.round(25.5); // 26

Math.random()	Use with floor() to get a random number.
	
	Math.floor(Math.random * num_choices + first value);
		var num = Math.floor(Math.random() * 10 + 1);
		
		// number between 1 and 10

Math.abs()	Absolute value	Math.abs(-5); // 5
Math.exp(x)	Math.E raised to the xth power	
Math.log(x)	Logarithm of x	

SOFTWAREGUILD

Almost Done…

METHOD / PROPERTY	DESCRIPTION	EXAMPLE
Math.pow(x,n)	Raise x to the nth power	Math.pow(5, 2); // 25
Math.sqrt(x)	Square root of x	Math.sqrt(4); // 2
Math.cos(x)	Cosine of x	
Math.sin(x)	Sine of x	
Math.tan(x)	Tangent of x	

Conclusion

That was a lot of stuff. Functions can have some pretty odd behaviors.

Dates, RegExp, and the various string, number, and Math functions can help you build full-featured applications.