# JavaScript:
# A Crash Course
## Part I: Core Language Syntax

Originals of Slides and Source Code for Examples:
http://courses.coreservlets.com/Course-Materials/ajax.html

---

## For live Ajax & GWT training, see training courses at http://courses.coreservlets.com/.

**Taught by the author of *Core Servlets and JSP*, *More Servlets and JSP*, and this tutorial. Available at public venues, or customized versions can be held on-site at your organization.**

# Topics in This Section

- **Overview**
- **JavaScript references**
- **Embedding in browser**
- **Basic syntax**
- **Strings and regular expressions**
- **Functions**
- **Objects**

# Intro

# Books

- ***JavaScript the Definitive Guide***
  - By David Flanagan, O'Reilly. The only really complete reference on the JavaScript language. Thorough and well-written.
    - Makes the global variable blunder when covering Ajax.
- ***JavaScript: The Good Parts***
  - By Douglas Crockford (of JSON and YUI fame), O'Reilly
  - Outstanding advanced guide to best practices in core JavaScript, especially functions, objects, and regular expressions. *Very* short.
    - Does not cover Ajax at all. No DOM scripting. "The K&R of JS".
- ***Pro JavaScript Techniques***
  - By John Resig (of jQuery fame), APress
  - Excellent guide to best practices; not a thorough reference
    - Does not make the global variable blunder when covering Ajax.
- ***DOM Scripting***
  - By Jeremy Keith, FriendsOf Press
  - Focuses on manipulating DOM and CSS
    - Makes the global variable blunder when briefly covering Ajax.

# Online References

- **JavaScript tutorial (language syntax)**
  - http://www.w3schools.com/js/
  - http://developer.mozilla.org/en/docs/ Core_JavaScript_1.5_Guide
- **JavaScript API references (builtin objects)**
  - http://www.w3schools.com/jsref/
  - http://www.devguru.com/technologies/ecmascript/ QuickRef/
  - http://www.devguru.com/technologies/JavaScript/
  - http://www.javascriptkit.com/jsref/
  - http://developer.mozilla.org/en/docs/ Core_JavaScript_1.5_Reference
- **HTML DOM reference (with JavaScript Examples)**
  - http://www.w3schools.com/htmldom/dom_reference.asp
- **Official ECMAScript specification**
  - http://www.ecma-international.org/publications/standards/ Ecma-262.htm

# Firebug

- **Install Firebug in Firefox**
  - http://getfirebug.com/
- **Use Firebug console for interactive testing**
  - http://getfirebug.com/cl.html
- **Can also use Firebug Lite in Internet Explorer**
  - Not great, but better than nothing
  - http://getfirebug.com/lite.html
    - See especially "bookmarklet" link
- **For more details on Firebug usage**
  - See section on Ajax development and debugging tools

9

# Embedding JavaScript in HTML

# Loading Scripts

- **script with src**
  - <script src="my-script.js" type="text/javascript"></script>
  - – Purpose
    - To define functions, objects, and variables.
    - Functions will later be triggered by buttons, other user events, inline script tags with body content, etc.
- **script with body content**
  - <script type="text/javascript">JavaScript code</script>
  - – Purpose
    - To directly invoke code that will run as page loads
      - – E.g., to output HTML content built by JavaScript
    - Don't use this approach for defining functions or for doing things that could be done in external files.
      - – Slower (no browser caching) and less reusable

# Example (phish.js)

```
function getMessage() {
  var amount = Math.round(Math.random() * 100000);
  var message =
    "You won $" + amount + "!\n" +
    "To collect your winnings, send your credit card\n" +
    "and bank details to oil-minister@phisher.com.";
  return(message);
}
                              "alert" pops up dialog box

function showWinnings1() {
  alert(getMessage());
}
                        "document.write" inserts text into page at current location

function showWinnings2() {
  document.write("<h1><blink>" + getMessage() +
                 "</blink></h1>");
}
```

# Example (loading-scripts.html)

```
<!DOCTYPE ...><html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Loading Scripts</title>
...
<script src="./scripts/phish.js"
        type="text/javascript"></script>
</head>
<body>
...
    <input type="button" value="How Much Did You Win?"
           onclick='showWinnings1()'/>
...
  <script type="text/javascript">showWinnings2()</script>
...
</body></html>
```

Loads script from previous page

Calls showWinnings1 when user presses button. Puts result in dialog box.

Calls showWinnings2 when page is loaded in browser. Puts result at this location in page.

# Example (Results)

# Loading Scripts: Special Cases

- **Internet Explorer bug**
  - Scripts with src fail to load if you use <script.../>.
    - You must use <script src="..." ...></script>
- **XHTML: Scripts with body content**
  - It is an error if the body of the script contains special XML characters such as & or <
  - E.g. <script...>if (a<b) { this(); } else { that(); }</script>
  - So, use CDATA section unless body content is simple and clearly has no special characters
    - <script type="text/javascript"><![CDATA[
      *JavaScript Code*
      ]]></script>

15

# Basic Syntax

# Variables

- **Introduce with "var"**
  - For global variables (!) and local variables.
  - No "var" for function arguments
- **You do not declare types**
  - Some people say JavaScript is "untyped" language, but really it is "dynamically typed" language
  - JavaScript is *very* liberal about converting types
- **There are only two scopes**
  - Global scope
    - Be very careful with this when using Ajax.
    - Can cause race conditions.
  - Function (lexical) scope
  - There is *not* block scope as in Java

# Operators and Statements

- **Almost same set of operators as Java**
  - + (addition and String concatenation), -, *, /
  - &&, ||, ++, --, etc
  - The == comparison is more akin to Java's "equals"
  - The === operator (less used) is like Java's ==
- **Statements**
  - Semicolons are technically optional
    - But highly recommended
  - Consider
    - return x
    - return
      x
    - They are not identical! The second one returns, then evaluates x. You should act as though semicolons are required as in Java.
- **Comments**
  - Same as in Java (/* ... */ and // ...)

# Conditionals and Simple Loops

- **if/else**
  - Almost identical to Java except test can be converted to true/false instead of strict true/false
    - "false": false, null, undefined, "" (empty string), 0, NaN
    - "true": anything else (including the string "false")
- **Basic for loop**
  - Identical to Java except for variable declarations
    - for(var i=0; i<someVal; i++) { doLoopBody(); }
- **while loop**
  - Same as Java except test can be converted to boolean
    - while(someTest) { doLoopBody(); }
- **do/while loop**
  - Same as Java except test can be converted to boolean

# Array Basics

- **One-step array allocation**
  - var primes = [2, 3, 5, 7, 11, 13];
  - var names = ["Joe", "Jane", "John", "Juan"];
    - <u>No</u> trailing comma after last element (see later slide)
- **Two-step array allocation**
  - var names = new Array(4);
    names[0] = "Joe";
    ...
    names[3] = "Juan";
- **Indexed at 0 as in Java**
  - for(var i=0; i<names.length; i++) {
    doSomethingWith(names[i]);
    }

# Other Conditionals and Loops

- **switch**
  - Differs from Java in two ways
    - The "case" can be an expression
    - Values need not be ints (compared with ===)
- **for/in loop**
  - On surface, looks similar to Java for/each loop, but
    - For arrays, values are array indexes, not array values
      - Use this loop for objects (to see property names), not arrays! Fails with Prototype or other extended arrays
    - For objects, values are the property names
  - var names = ["Joe", "Jane", "John", "Juan"];
    for(var i in names) {
        doSomethingWith(names[i]);
    }

21

# More on Arrays

- **Arrays can be sparse**
  - var names = new Array();
    names[0] = "Joe";
    names[100000] = "Juan";
- **Arrays can be resized**
  - Regardless of how arrays is created, you can do:
    - myArray.length = someNewLength;
    - myArray[anyNumber] = someNewValue;
    - myArray.push(someNewValue)
      - These are legal regardless of which way myArray was made
- **Arrays have methods**
  - push, pop, join, reverse, sort, concat, slice, splice, etc.
    - See API reference
- **Regular objects can be treated like arrays**
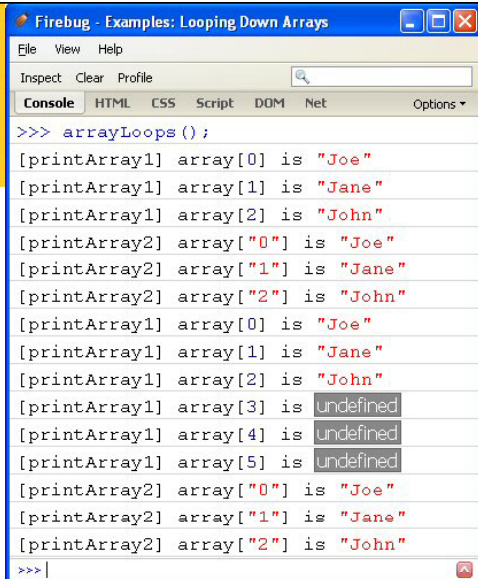  - You can use numbers (indexes) as properties

22

# Arrays Example

```
function arrayLoops() {
  var names =
    ["Joe", "Jane", "John"];
  printArray1(names);
  printArray2(names);
  names.length = 6;
  printArray1(names);
  printArray2(names);
}

function printArray1(array) {
  for(var i=0; i<array.length; i++) {
    console.log("[printArray1] array[%o] is %o", i, array[i]);
  }
}

function printArray2(array) {
  for(var i in array) {
    console.log("[printArray2] array[%o] is %o", i, array[i]);
  }
}
arrayLoops();
```
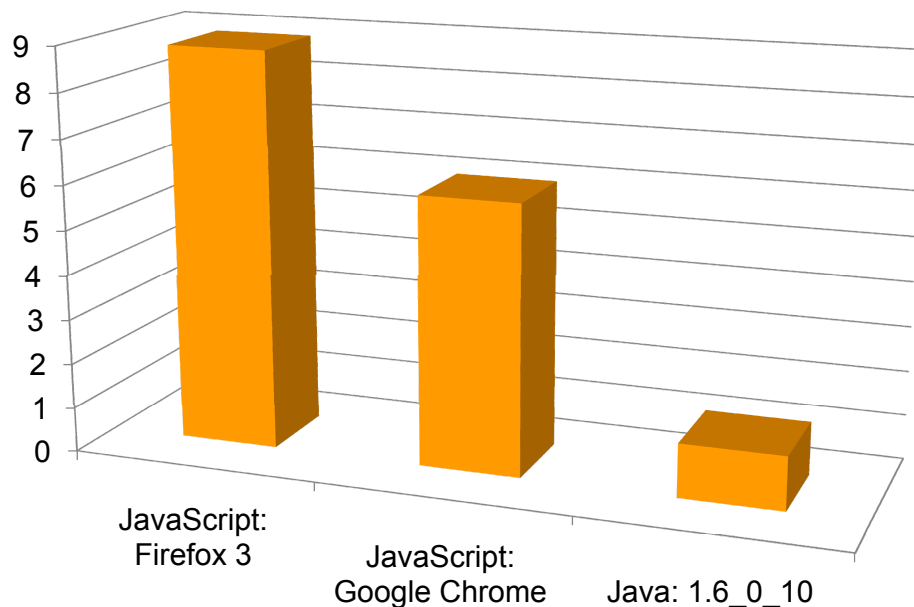
console.log is a printf-like way to print output in Firebug Console window. For testing/debugging only.

Direct call for interactive testing in Firebug console. (Cut/paste all code into console command line.)

---

# Array Performance

**Time to create and sum array of 16 million random numbers**



JavaScript: Firefox 3

JavaScript: Google Chrome

Java: 1.6_0_10

Note: Internet Explorer 7 was more than 10 times slower than Firefox, so times are not shown here.
Source code for benchmarks is in downloadable Eclipse project at coreservlets.com.

# The Math Class

- **Almost identical to Java**
  - Like Java, static methods (Math.cos, Math.random, etc.)
  - Like Java, logs are base e, trig functions are in radians
- **Functions**
  - Math.abs, Math.acos, Math.asin, Math.atan, Math.atan2, Math.ceil, Math.cos, Math.exp, Math.floor, Math.log, Math.max, Math.min, Math.pow, Math.random, Math.round, Math.sin, Math.sqrt, Math.tan
- **Constants**
  - Math.E, Math.LN10, Math.LN2, Math.LOG10E, Math.PI, Math.SQRT1_2, Math.SQRT2

---

# Strings and Regular Expressions

# String Basics

- **You can use double or single quotes**
  - var names = ["Joe", 'Jane', "John", 'Juan'];
- **You can access length property**
  - E.g., "foobar".length returns 6
- **Numbers can be converted to strings**
  - Automatic conversion during concatenations. String need not be first as in Java
    - var val = 3 + "abc" + 5;  // Result is "3abc5"
  - Conversion with fixed precision
    - var n = 123.4567;
      var val = n.toFixed(2); // Result is 123.46 (not 123.45)
- **Strings can be compared with ==**
  - "foo" == 'foo' returns true
- **Strings can be converted to numbers**
  - var i = parseInt("37 blah"); // Result is 37 – ignores blah
  - var d = parseFloat("6.02 blah"); // Ignores blah

# Core String Methods

- **Simple methods similar to Java**
  - charAt, indexOf, lastIndexOf, substring, toLowerCase, toUpperCase
- **Methods that use regular expressions**
  - match, replace, search, split
- **HTML methods**
  - anchor, big, bold, fixed, fontcolor, fontsize, italics, link, small, strike, sub, sup
    - "test".bold().italics().fontcolor("red") returns
      '<font color="red"><i><b>test</b></i></font>'
  - These are technically nonstandard methods, but supported in all major browsers
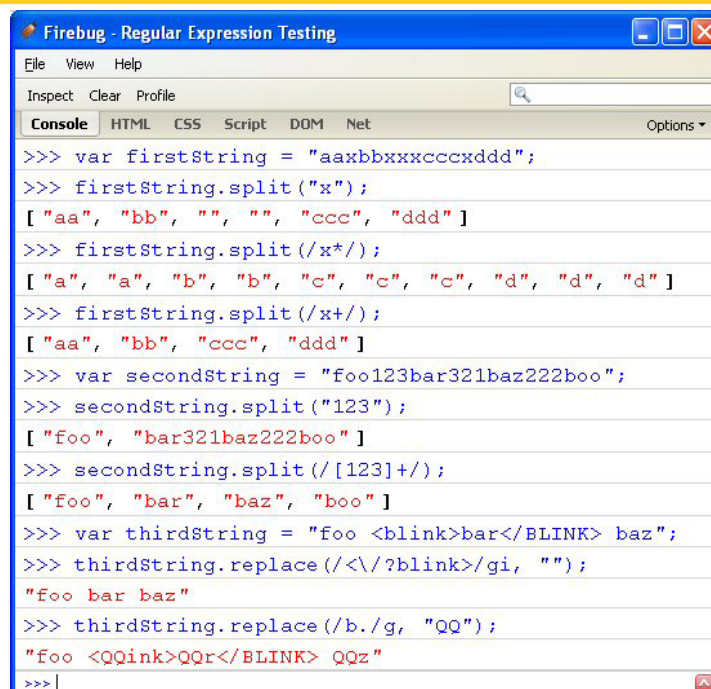    - But I prefer to construct HTML strings explicitly anyhow

# Regular Expressions

- **You specify a regexp with /pattern/**
  - *Not* with a String as in Java
- **Most special characters same as in Java/Unix/Perl**
  - ^, $, .      – beginning, end of string, any one char
  - \              – escape what would otherwise be a special character
  - *, +, ?      – 0 or more, 1 or more, 0 or 1 occurrences
  - {n}, {n,}   – exactly n, n or more occurrences
  - []             – grouping
  - \s, \S      – whitespace, non-whitespace
  - \w, \W     – word char (letter or number), non-word char
- **Modifiers**
  - /pattern/g – do global matching (find all matches, not just first one)
  - /pattern/i  – do case-insensitive matching
  - /pattern/m – do multiline matching

# Regular Expression: Examples

# More Information on Regular Expressions

- **Online API references given earlier (See RegExp class)**
  - http://www.w3schools.com/jsref/jsref_obj_regexp.asp
  - http://www.devguru.com/technologies/ecmascript/QuickRef/regexp.html
- **JavaScript Regular Expression Tutorials**
  - http://www.evolt.org/article/Regular_Expressions_in_JavaScript/17/36435/
  - http://www.javascriptkit.com/javatutors/re.shtml

---

# Functions

"It is Lisp in C's clothing."
- JSON and YUI guru Douglas Crockford, describing
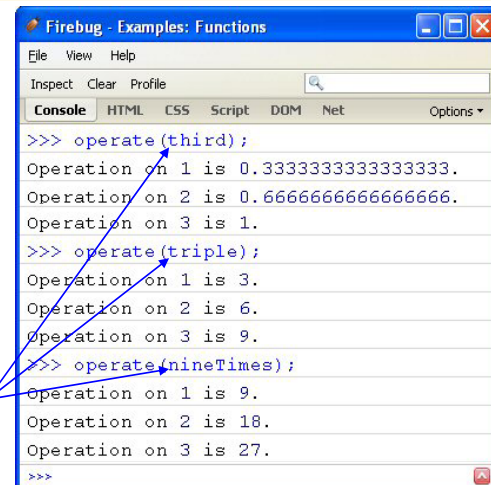the JavaScript language in *JavaScript: The Good Parts*.

# Overview

- **Not similar to Java**
  - JavaScript functions *very* different from Java methods
- **Main differences from Java**
  - You can have global functions
    - Not just methods (functions as part of objects)
  - You don't declare return types or argument types
  - Caller can supply any number of arguments
    - Regardless of how many arguments you defined
  - Functions are first-class datatypes
    - You can pass functions around, store them in arrays, etc.
  - You can create anonymous functions (closures)
    - Critical for Ajax
    - These are equivalent
      - function foo(...) {...}
      - var foo = function(...) {...}

# Passing Functions: Example

```
function third(x) {
  return(x / 3);
}

function triple(x) {
  return(x * 3);
}

function nineTimes(x) {
  return(x * 9);
}
```

Function as argument.

```
function operate(f) {
  var nums = [1, 2, 3];
  for(var i=0; i<nums.length; i++) {
    var num = nums[i];
    console.log("Operation on %o is %o.",
                num, f(num));
  }
}
```



Firebug - Examples: Functions

File  View  Help

Inspect  Clear  Profile

Console  HTML  CSS  Script  DOM  Net  Options

```
>>> operate(third);
Operation on 1 is 0.3333333333333333.
Operation on 2 is 0.6666666666666666.
Operation on 3 is 1.
>>> operate(triple);
Operation on 1 is 3.
Operation on 2 is 6.
Operation on 3 is 9.
>>> operate(nineTimes);
Operation on 1 is 9.
Operation on 2 is 18.
Operation on 3 is 27.
>>>
```
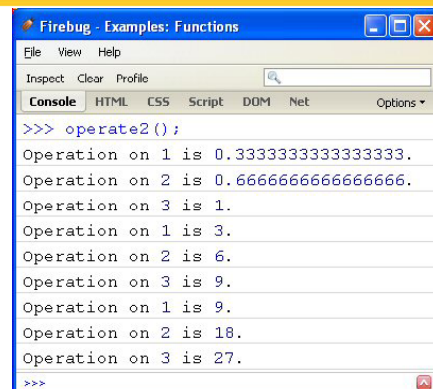
# Anonymous Functions

- **Anonymous functions (or closures) let you capture local variables inside a function**
  - You can't do Ajax without this!
- **Basic anonymous function**
  - `operate(function(x) { return(x * 20); });`
    - Outputs 20, 40, 60
    - The "operate" function defined on previous page
- **Anonymous function with captured data**
  - ```
    function someFunction(args) {
      var val = someCalculation(args);
      return(function(moreArgs) {
              doSomethingWith(val, moreArgs);
          });
    }
    var f1 = someFunction(args1);
    var f2 = someFunction(args2);
    f1(args3);  // Uses one copy of "val"
    f2(args3);  // Uses a different copy of "val"
    ```

35

---

# Anonymous Functions: Example

```
function multiplier(m) {
  return(function(x)
         { return(x * m); });
}




function operate2() {
  var nums = [1, 2, 3];
  var functions =
    [multiplier(1/3), multiplier(3), multiplier(9)];
  for(var i=0; i<functions.length; i++) {
    for(var j=0; j<nums.length; j++) {
      var f = functions[i];
      var num = nums[j];
      console.log("Operation on %o is %o.",
                  num, f(num));
    }
  }
}
```

```
Firebug - Examples: Functions
File   View   Help
Inspect   Clear   Profile
Console   HTML   CSS   Script   DOM   Net          Options ▾
>>> operate2();
Operation on 1 is 0.3333333333333333.
Operation on 2 is 0.6666666666666666.
Operation on 3 is 1.
Operation on 1 is 3.
Operation on 2 is 6.
Operation on 3 is 9.
Operation on 1 is 9.
Operation on 2 is 18.
Operation on 3 is 27.
>>>
```

36

# Optional Args: Summary

- **Fixed number of optional args**
  - Functions can *always* be called with any number of args
  - Compare typeof args to "undefined"
  - See next page and upcoming convertString function
- **Arbitrary args**
  - Discover number of args with arguments.length
  - Get arguments via arguments[i]
  - See next page and upcoming longestString function
- **Optional args via anonymous object**
  - Caller always supplies same number of arguments, but one of the arguments is an anonymous (JSON) object
    - This object has optional fields
  - See later example in "Objects" section

37

# Optional Args: Details

- **You can call any function with any number of arguments**
  - If called with fewer args, extra args equal "undefined"
    - You can use typeof arg == "undefined" for this
      - You can also use boolean comparison if you are sure that no real value could match (e.g., 0 and undefined both return true for !arg)
    - Use comments to indicate optional args
      - function foo(arg1, arg2, /* Optional */ arg3) {...}
  - If called with extra args, you can use "arguments" array
    - Regardless of defined variables, arguments.length tells you how many arguments were supplied, and arguments[i] returns the designated argument
    - Use comments to indicate extra args
      - function bar(arg1, arg2 /* varargs */) { ... }

38

# Optional Arguments

```
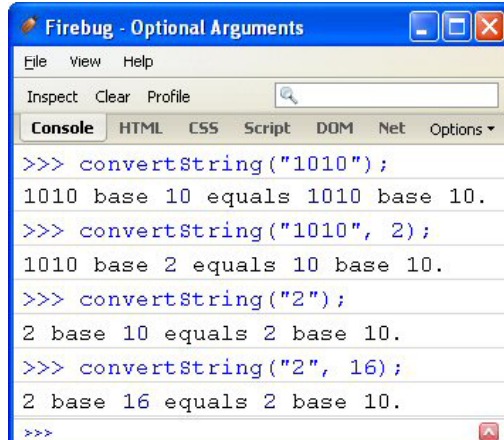function convertString(numString, /* Optional */ base) {
  if (typeof base == "undefined") {
    base = 10;
  }
  var num = parseInt(numString, base);
  console.log("%s base %o equals %o base 10.",
              numString, base, num);
}
```

# Varargs

```
function longestString(/* varargs */) {
  var longest = "";
  for(var i=0; i<arguments.length; i++) {
    var candidateString = arguments[i];
    if (candidateString.length > longest.length) {
      longest = candidateString;
    }
  }
  return(longest);
}

longestString("a", "bb", "ccc", "dddd");
    // Returns "dddd"
```

# Objects

# Basics

- ## Constructors
  - Functions named for class names. Then use "new".
    - No separate class definition! No "real" OOP in JavaScript!
  - Can define properties with "this"
    - You <u>must</u> use "this" for properties used in constructors
    ```
    function MyClass(n1) { this.foo = n1; }
    var m = new MyClass(10);
    ```
- ## Properties (instance variables)
  - You don't define them separately
    - Whenever you refer to one, JavaScript just creates it
    ```
    m.bar = 20; // Now m.foo is 10 and m.bar is 20
    ```
    - Usually better to avoid introducing new properties in outside code and instead do entire definition in constructor
- ## Methods
  - Properties whose values are functions

## Objects: Example (Circle Class)

```
function Circle(radius) {
  this.radius = radius;

  this.getArea =
    function() {
      return(Math.PI * this.radius * this.radius);
    };
}


var c = new Circle(10);
c.getArea(); // Returns 314.1592...
```

# The prototype Property

- **In previous example**
  - Every new Circle got its own copy of radius
    - Fine, since radius has per-Circle data
  - Every new Circle got its own copy of getArea function
    - Wasteful since function definition never changes
- **Class-level properties**
  - Classname.prototype.propertyName = value;
- **Methods**
  - Classname.prototype.methodName = function() {...};
    - Just a special case of class-level properties
  - This is legal anywhere, but it is best to do it in constructor
- **Pseudo-Inheritance**
  - The prototype property can be used for inheritance
  - But complex. See later section on Prototype library

## Objects: Example (Updated Circle Class)

```
function Circle(radius) {
  this.radius = radius;

  Circle.prototype.getArea =
    function() {
      return(Math.PI * this.radius * this.radius);
    };
}

var c = new Circle(10);
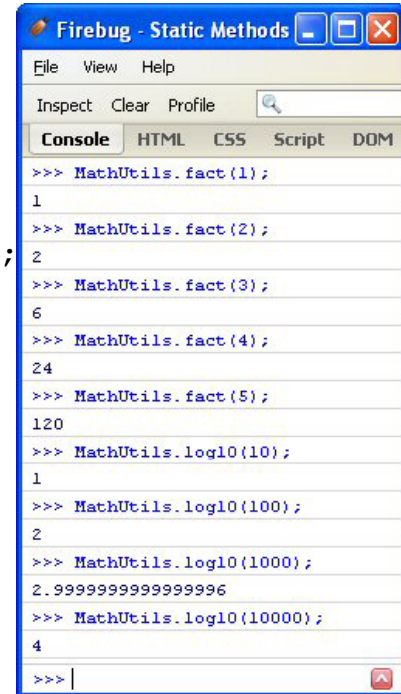c.getArea(); // Returns 314.1592...
```

# Rolling Your Own Namespace

- **Idea**
  - Have related functions that do not use object properties
  - You want to group them together and call them with Utils.func1, Utils.func2, etc.
    - Grouping is a syntactic convenience. Not real methods.
    - Helps to avoid name conflicts when mixing JS libraries
  - Similar to static methods in Java
- **Syntax**
  - Assign functions to properties of an object, but do not define a constructor. E.g.,
    - var Utils = {}; // Or "new Object()", or make function Utils
      Utils.foo = function(a, b) { … };
      Utils.bar = function(c) { … };
      var x = Utils.foo(val1, val2);
      var y = Utils.bar(val3);

# Static Methods: Example (Code)

```javascript
var MathUtils = {};

MathUtils.fact = function(n) {
  if (n <= 1) {
    return(1);
  } else {
    return(n * MathUtils.fact(n-1));
  }
};

MathUtils.log10 = function(x) {
  return(Math.log(x)/Math.log(10));
};
```

**Firebug - Static Methods**

File   View   Help

Inspect  Clear  Profile

**Console**  HTML  CSS  Script  DOM

```
>>> MathUtils.fact(1);
1
>>> MathUtils.fact(2);
2
>>> MathUtils.fact(3);
6
>>> MathUtils.fact(4);
24
>>> MathUtils.fact(5);
120
>>> MathUtils.log10(10);
1
>>> MathUtils.log10(100);
2
>>> MathUtils.log10(1000);
2.9999999999999996
>>> MathUtils.log10(10000);
4
>>>
```

---

# Namespaces in Real Applications

- **Best practices in large projects**
  – In many (most?) large projects, *all* global variables (including functions!) are forbidden due to the possibility of name collisions from pieces made by different authors.
  – So, these primitive namespaces play the role of Java's packages. Much weaker, but still very valuable.

- **Fancy variation: repeat the name**
  - var MyApp = {};
  - MyApp.foo = function foo(…) { … };
  - MyApp.bar = function bar(…) { … };
  – The name on the right does not become a global name. The only advantage is for debugging
    - Firebug and other environments will show the name when you print the function object.

# JSON (JavaScript Object Notation)

- **Idea**
  - A simple textual representation of JavaScript objects
  - Main applications
    - One-time-use objects (rather than reusable classes)
    - Objects received via strings
- **Directly in JavaScript**
  - ```
    var someObject =
        { property1: value1,
          property2: value2,
          ... };
    ```
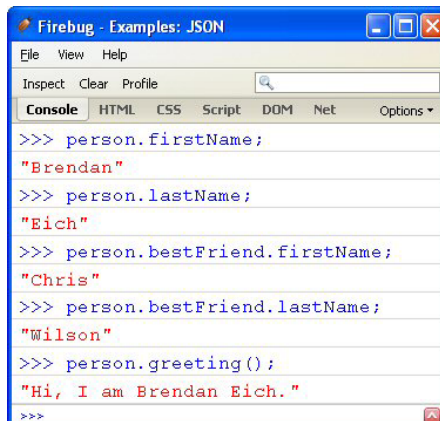- **In a string (e.g., when coming in on network)**
  - Surround object representation in parens
  - Pass to the builtin "eval" function

# JSON: Example

```
var person =
   { firstName:  'Brendan',
     lastName:   'Eich',
     bestFriend: { firstName: 'Chris',
                   lastName:  'Wilson' },
     greeting:  function() {
                  return("Hi, I am " + this.firstName +
                         " " + this.lastName + ".");
                }
   };
```



```
>>> person.firstName;
"Brendan"
>>> person.lastName;
"Eich"
>>> person.bestFriend.firstName;
"Chris"
>>> person.bestFriend.lastName;
"Wilson"
>>> person.greeting();
"Hi, I am Brendan Eich."
>>>
```

# Using JSON for Optional Arguments

- **Idea**
  - Caller always supplies same number of arguments, but one of the arguments is an anonymous (JSON) object
    - This object has optional fields
  - This approach is widely used in Prototype, Scriptaculous, and other JavaScript libraries
- **Example (a/b: required, c/d/e/f: optional)**
  - someFunction(1.2, 3.4, {c: 4.5, f: 6.7});
  - someFunction(1.2, 3.4, {c: 4.5, d: 6.7, e: 7.8});
  - someFunction(1.2, 3.4, {c: 9.9, d: 4.5, e: 6.7, f: 7.8});
  - someFunction(1.2, 3.4);

# Using JSON for Optional Arguments: Example Code
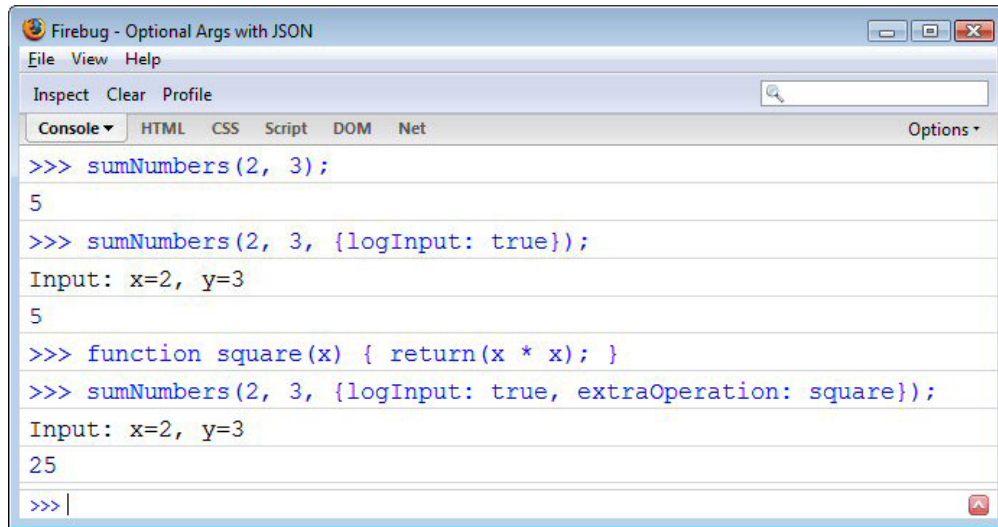
```
function sumNumbers(x, y, extraParams) {
  var result = x + y;
  if (isDefined(extraParams)) {
    if (isTrue(extraParams.logInput)) {
      console.log("Input: x=%s, y=%s", x, y);
    }
    if (isDefined(extraParams.extraOperation)) {
      result = extraParams.extraOperation(result);
    }
  }
  return(result)
}

function isDefined(value) {
  return(typeof value != "undefined");
}

function isTrue(value) {
  return(isDefined(value) && (value == true))
}
```

# Using JSON for Optional Arguments: Example Results

```
Firebug - Optional Args with JSON

File  View  Help

Inspect  Clear  Profile                                    🔍

Console ▼   HTML   CSS   Script   DOM   Net              Options ▼

>>> sumNumbers(2, 3);
5
>>> sumNumbers(2, 3, {logInput: true});
Input: x=2, y=3
5
>>> function square(x) { return(x * x); }
>>> sumNumbers(2, 3, {logInput: true, extraOperation: square});
Input: x=2, y=3
25
>>>
```

# Internet Explorer and Extra Commas

- **Firefox tolerates trailing commas in both arrays and JSON**
  - var nums = [1, 2, 3, ];
  - var obj = { firstName: "Joe", lastName: "Hacker", };
- **IE will crash in both cases.**
  – And, since it is not technically legal anyway, you should write it *without* commas after the final element:
    - var nums = [1, 2, 3];
    - var obj = { firstName: "Joe", lastName: "Hacker"};
  – This issue comes up moderately often, especially when building JavaScript data on the server, as we will do in upcoming lectures.

# Other Object Tricks

- ## The instanceof operator
  - Determines if lhs is a member of class on rhs
    - if (blah instanceof Array) {
        doSomethingWith(blah.length);
      }
- ## The typeof operator
  - Returns direct type of operand, as a String
    - "number", "string", "boolean", "object", "function", or "undefined".
      - Arrays and null both return "object"
- ## Adding methods to builtin classes
  ```
  String.prototype.describeLength =
      function() { return("My length is " + this.length); };
  "Any Random String".describeLength();
  ```
- ## eval
  - Takes a String representing *any* JavaScript and runs it
    - `eval("3 * 4 + Math.PI");  // Returns 15.141592`

55

---

# Wrap-up

# Summary

- **Use Firebug for testing and debugging**
- **Bookmark references**
  - http://www.w3schools.com/js/
- **Embedding in browser**
  - <script src="blah.js" type="test/javascript"></script>
- **Basic syntax**
  - Mostly similar to Java
- **Functions**
  - Totally different from Java. Passing functions around and making anonymous functions very important.
- **Objects**
  - Constructor also defines class. Use "this".
  - Totally different from Java. Not like classical OOP at all.

57

# Questions?