Javascript Intro

Basic Language Constructs
Working in the Browser

Goes with examples in javascript1 directory

Overview

First steps – command-line tools

- We will quickly get to the browser, but the 1st examples can be run from the command-line
- I simply use node <name of file>
- You also could reference the scripts from a simple web page and run them locally through an http server (we'll do this soon)
- There are several tools, Scratchpad from Firefox may be easiest

Topics for this Javascript overview:

- Variables, Operations, and Statements
- Conditionals and Loops
- Arrays
- Strings
- Functions
- Object basics and Prototypes
- Static methods

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 *no 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. 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
  - 0 is false, 1 is true
  - Many people avoid this and use strict booleans

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"];`

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

- Similar to Java for/each loop, but
  - For arrays, values are array indexes, not array values
    - Indexes are treated as strings ("0")
    - Shows only indexes with values
  - For objects, values are the property names
-
```
var names = ["Joe", "Jane", "John", "Juan"];
for(var i in names) {
    doSomethingWith(names[i]);
}
```

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;`
    - This is legal regardless of which way myArray was made

Arrays have methods

- `join`, `reverse`, `sort`, `concat`, `slice`, `splice`, `toString`, etc.

Regular objects can be treated like arrays

- You can use numbers (indexes) as properties
  ```

Arrays Example *(arrays.js)*

```
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]
  }
}

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.)

Firebug - Examples: Looping Down Arrays

File View Help

Inspect Clear Profile

Console HTML CSS Script DOM Net Options

```
>>> arrayLoops();
[printArray1] array[0] is "Joe"
[printArray1] array[1] is "Jane"
[printArray1] array[2] is "John"
[printArray2] array["0"] is "Joe"
[printArray2] array["1"] is "Jane"
[printArray2] array["2"] is "John"
[printArray1] array[0] is "Joe"
[printArray1] array[1] is "Jane"
[printArray1] array[2] is "John"
[printArray1] array[3] is undefined
[printArray1] array[4] is undefined
[printArray1] array[5] is undefined
[printArray2] array["0"] is "Joe"
[printArray2] array["1"] is "Jane"
[printArray2] array["2"] is "John"
>>>
```

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 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 *(regexps.js)*

- `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
  - Usually you are going to construct HTML strings explicitly anyhow

Functions 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 *(functions1.js)*

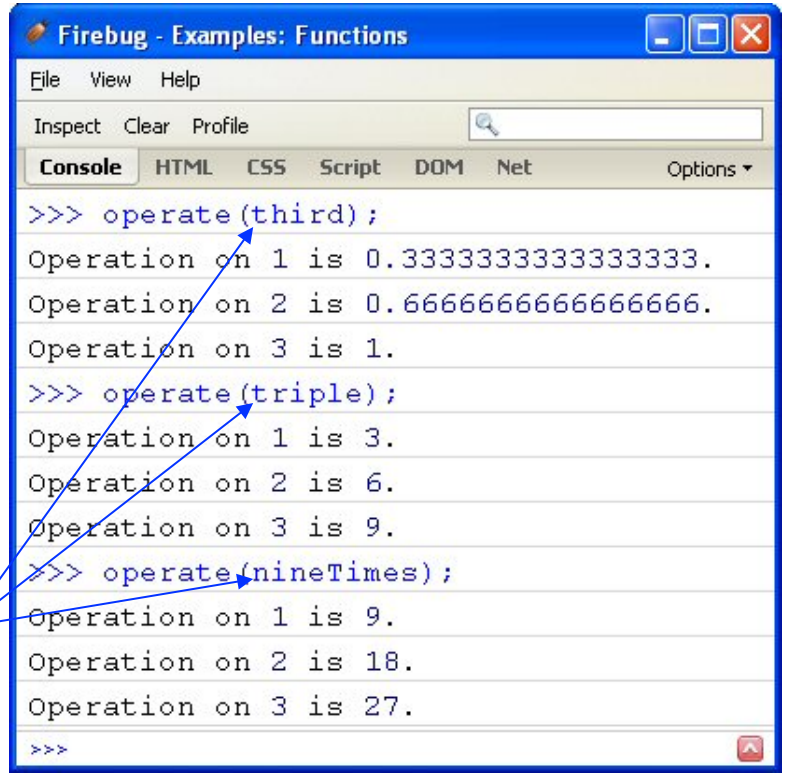
```
function third(x) {
  return(x / 3);
}

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

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


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));
  }
}
```

Function as argument.



Anonymous Functions

Anonymous functions (or closures) let you capture local variables inside a function

- You can't do Ajax without this!

Basic anonymous function

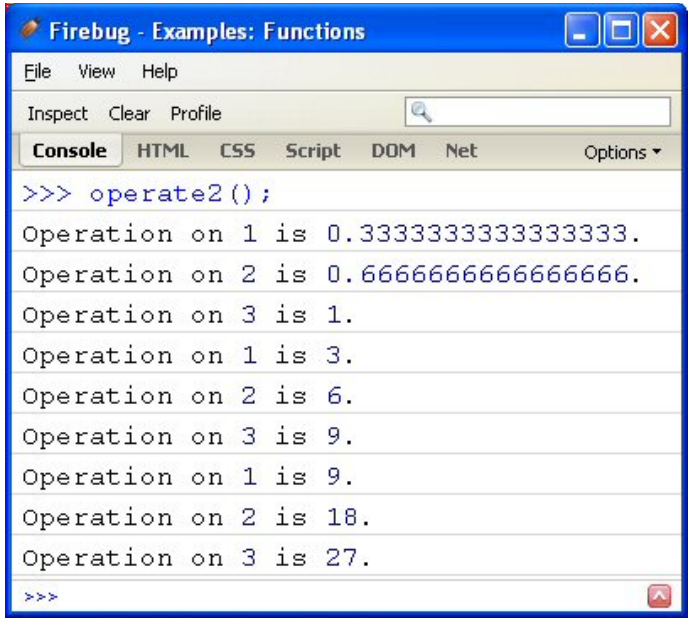
- `operate(function(x) { return(x * 20); });`
  - Outputs 20, 40, 60
  - The "operate" function defined on previous page

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"
```

Anonymous Functions: Example *(functions2.js)*

```
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));
    }
  }
}
```

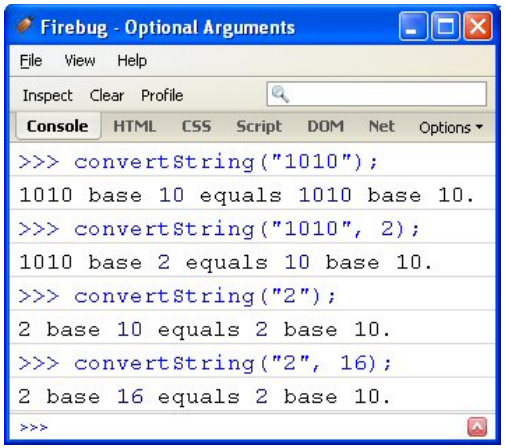
Optional Args and Varargs

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 */) { ... }`

Optional and Variable Arguments Examples

```
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);
}
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"
```



varargs.js and optional-args.js

Object 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) *[objects1.js]*

```
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...
```

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 (if many Circles), since function definition never changes

The prototype Property *(objects2.js)*

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

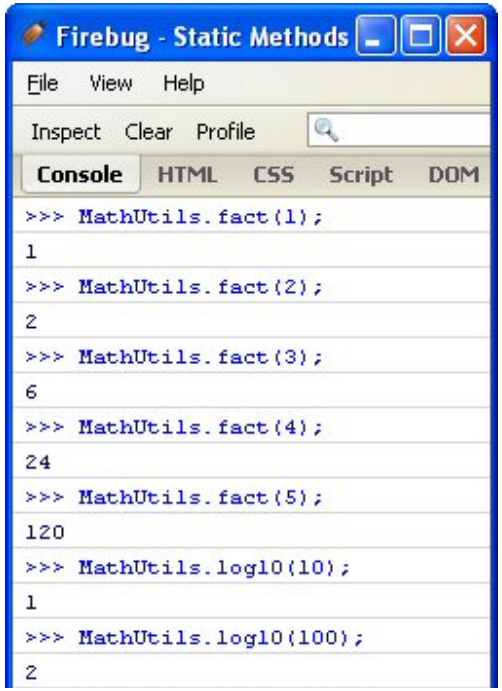
```
function Circle(radius) {
  this.radius = radius;
  Circle.prototype.getArea = function() {
      return(Math.PI * this. radius * this.radius);
  };
}
var c = new Circle(10);
c.getArea(); // Returns 3.141592...
```

Static Methods *(math-utils.js)*

Idea

- Several 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.
- Very similar to static methods in Java
- Assign functions to properties of an object, but don't define a constructor

```
var MathUtils = new Object();
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));
}
```



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`
```