

## JavaScript

John Mitchell

Reading: links on last slide    Homework 1: Sept 26 - Oct 3

## Why talk about JavaScript?

- Very widely used, and growing
  - Web pages, AJAX, Web 2.0
  - Increasing number of web-related applications
- Some interesting and unusual features
  - First-class functions - interesting
  - Objects without classes - slightly unusual
  - Powerful modification capabilities - very unusual
    - Add new method to object, redefine prototype, ...
- Many security, correctness issues
  - Not statically typed – type of variable may change ...
  - Difficult to predict program properties in advance

## JavaScript History

- Developed by Brendan Eich at Netscape
  - Scripting language for Navigator 2
- Later standardized for browser compatibility
  - ECMAScript Edition 3 (aka JavaScript 1.5)
- Related to Java in name only
  - Name was part of a marketing deal
- Various implementations available
  - Spidermonkey interactive shell interface
  - Rhino: <http://www.mozilla.org/rhino/>

## Motivation for JavaScript

- Netscape, 1995
  - Netscape > 90% browser market share
  - Opportunity to do "HTML scripting language"
  - Brendan Eich
    - I hacked the JS prototype in ~1 week in May
    - And it showed! Mistakes were frozen early
    - Rest of year spent embedding in browser - ICFP talk, 2006
- Common uses of JavaScript include:
  - Form validation
  - Page embellishments and special effects
  - Navigation systems
  - Basic Math calculations
  - Dynamic content manipulation

## Example 1: simple calculation

```
<html>
...
<p> ... </p>
<script>
var num1, num2, sum
num1 = prompt("Enter first number")
num2 = prompt("Enter second number")
sum = parseInt(num1) + parseInt(num2)
alert("Sum = " + sum)
</script>
...
</html>
```

## Example 2: browser events

```
<script type="text/JavaScript">
function whichButton(event) {
  if (event.button==1) {
    alert("You clicked the left mouse button!")
  }
  else {
    alert("You clicked the right mouse button!")
  }
}
</script>
...
<body onmousedown="whichButton(event)">
...
</body>
```

Other events: `onLoad`, `onMouseMove`, `onKeyPress`, `onUnload`

## Example 3: page manipulation

- Some possibilities
  - `createElement(elementName)`
  - `createTextNode(text)`
  - `appendChild(newChild)`
  - `removeChild(node)`

- Example: Add a new list item:

```
var list = document.getElementById('l1')
var newItem = document.createElement('li')
var newText = document.createTextNode(text)
list.appendChild(newItem)
newItem.appendChild(newText)
```

This example uses the browser Document Object Model (DOM). For now, we will focus on JavaScript as a language, not its use in the browser.

## Design goals

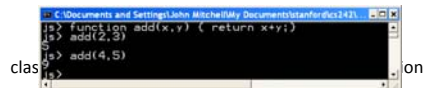
- Brendan Eich's 2006 ICFP talk
  - Make it easy to copy/paste snippets of code
  - Tolerate "minor" errors (missing semicolons)
  - Simplified onclick, onmousedown, etc., event handling, inspired by HyperCard
  - Pick a few hard-working, powerful primitives
    - First class functions for procedural abstraction
    - Objects everywhere, prototype-based
  - Leave all else out!

## Language basics

- JavaScript is case sensitive
  - HTML is not case sensitive; `onClick`, `ONCLICK`, ... are HTML
- Statements terminated by returns or semi-colons (;)
  - `x = x+1;` same as `x = x+1`
  - Semi-colons can be a good idea, to reduce errors
- "Blocks"
  - Group statements using `{ ... }`
  - Not separate scope, unlike other languages (see later slide)
- Variables
  - Define a variable using the `var` statement
  - Define implicitly by its first use, which must be an assignment
    - Implicit definition has global scope, even if it occurs in nested scope?

## Useful implementation

- Spidermonkey command-line interpreter
  - Read-eval-print loop
    - Enter declaration or statement
    - Interpreter executes
    - Displays value
    - Returns to input state
  - Example



```
C:\Documents and Settings\John Mitchell\My Documents\stanfor\js>
js> function add(x,y) { return x+y; }
js> add(2,3)
5
js> add(4,5)
9
class>
```

## JavaScript blocks

- Use `{ }` for grouping; not a separate scope

```
js> var x=3;
js> x
3
js> {var x=4; x}
4
js> x
4
```
- Not blocks in the sense of other languages

## JavaScript primitive datatypes

- Boolean
  - Two values: *true* and *false*
- Number
  - 64-bit floating point, similar to Java double and Double
  - No integer type
  - Special values *NaN* (not a number) and *Infinity*
- String
  - Sequence of zero or more Unicode characters
  - No separate character type (just strings of length 1)
  - Literal strings using `'` or `"` characters (must match)
- Special values
  - *null* and *undefined*
  - `typeof(null) = object`; `typeof(undefined) = object`;

## Objects

- An object is a collection of named properties
  - Simple view: hash table or associative array
  - Can define by set of name:value pairs
    - objBob = {name: "Bob", grade: 'A', level: 3};
  - New members can be added at any time
    - objBob.fullName = 'Robert';
  - Can have methods, can refer to *this*
- Arrays, functions regarded as objects
  - A property of an object may be a function (=method)
  - A function defines an object with method called “{ }”

```
function max(x,y) { if (x>y) return x; else return y; }
max.description = "return the maximum of two arguments";
```

## More about functions

- Declarations can appear in function body
  - Local variables, “inner” functions
- Parameter passing
  - Basic types passed by value, objects by reference
- Call can supply any number of arguments
  - `functionname.length` : # of arguments in definition
  - `functionname.arguments.length` : # args in call
- “Anonymous” functions (expressions for functions)
  - `(function (x,y) {return x+y}) (2,3);`
- Closures and Curried functions
  - `function CurAdd(x){ return function(y){return x+y} ;}`

[More explanation on next slide](#)

## Function Examples

- Anonymous functions make great callbacks

```
setTimeout(function() { alert("done"); }, 10000)
```
- Curried function

```
function CurriedAdd(x){ return function(y){ return x+y} ;}
g = CurriedAdd(2);
g(3)
```
- Variable number of arguments

```
function sumAll() {
  var total=0;
  for (var i=0; i< sumAll.arguments.length; i++)
    total+=sumAll.arguments[i];
  return(total); }
sumAll(3,5,3,5,3,2,6)
```

## Use of anonymous functions

- Anonymous functions very useful for callbacks

```
setTimeout(function() { alert("done"); }, 10000)
// putting alert("done") in function delays evaluation until call
```
- Simulate blocks by function definition and call

```
var u = { a:1, b:2 }
var v = { a:3, b:4 }
(function (x,y) {
  var tempA = x.a; var tempB =x.b; //local variables
  x.a=y.a; x.b=y.b;
  y.a=tempA; y.b=tempB
})(u,v)
// This works because objects are passed by reference
```

## Basic object features

- Use a function to construct an object

```
function car(make, model, year) {
  this.make = make;
  this.model = model;
  this.year = year;
}
```
- Objects have prototypes, can be changed

```
var c = new car("Ford", "Taurus", 1988);
car.prototype.print = function () {
  return this.year + " " + this.make + " " + this.model; }
c.print();
```

## JavaScript in web pages

- Embed in HTML page as `<script>` element
  - With JavaScript written directly in page

```
<script> alert("Hello World!"); </script>
```
  - Linked file as `src` attribute of the `<script>` element

```
<script type="text/JavaScript" src="functions.js"></script>
```
- As event handler attribute such as `onclick`

```
<a href="http://www.yahoo.com" onmouseover="alert('hi');">
```
- As pseudo-URL referenced by a link

```
<a href="JavaScript:alert('You clicked');">Click me</a>
```

We'll look at JavaScript as a language; ignore BOM, DOM, Ajax for now

## Language features in CS242

- Stack memory management
  - Parameters, local variables in activation records
- Garbage collection
  - Automatic reclamation of inaccessible memory
- Closures
  - Function together with environment (global variables)
- Exceptions
  - Jump to previously declared location, passing values
- Object features
  - Dynamic lookup, Encapsulation, Subtyping, Inheritance
- Concurrency
  - Do more than one task at a time (JavaScript is single-threaded)

## Stack memory management

- Local variables in activation record of function

```
function f(x) {  
  var y = 3;  
  function g(z) { return y+z;};  
  return g(x);  
}  
var x= 1; var y =2;  
f(x) + y;
```

## Garbage collection

- Automatic reclamation of unused memory
  - Navigator 2: per page memory management
    - Reclaim memory when browser changes page
  - Navigator 3: reference counting
    - Each memory region has associated count
    - Count modified when pointers are changed
    - Reclaim memory when count reaches zero
  - Navigator 4: mark-and-sweep, or equivalent
    - Garbage collector marks reachable memory
    - Sweep and reclaim unreachable memory

Reference [http://www.unix.org.ua/oreilly/web/jsript/ch11\\_07.html](http://www.unix.org.ua/oreilly/web/jsript/ch11_07.html)  
Discuss garbage collection in connection with Lisp

## Closures

- Return a function from function call

```
function f(x) {  
  var y = x;  
  return function (z){y += z; return y;}  
}  
var h = f(5);  
h(3);
```
- Can use this idea to define objects with “private” fields
  - Description of technique
    - <http://www.crockford.com/JavaScript/private.html>
    - [http://developer.mozilla.org/en/docs/Core\\_JavaScript\\_1.5\\_Guide:Working\\_with\\_Closures](http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Guide:Working_with_Closures)
  - But there are subtleties (look for `__parent__`)

## Exceptions

- Throw an expression of any type

```
throw "Error2";  
throw 42;  
throw {toString: function() { return "I'm an object!"; }};
```
- Catch

```
try {  
} catch (e if e == "FirstException") { // do something  
} catch (e if e == "SecondException") { // do something else  
} catch (e){ // executed if no match above  
}
```

Reference: [http://developer.mozilla.org/en/docs/Core\\_JavaScript\\_1.5\\_Guide:\\_Exception\\_Handling\\_Statements](http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Guide:_Exception_Handling_Statements)

## Object features

- Dynamic lookup
  - Method depends on run-time value of object
- Encapsulation
  - Object contains private data, public operations
- Subtyping
  - Object of one type can be used in place of another
- Inheritance
  - Use implementation of one kind of object to implement another kind of object

## Concurrency

- JavaScript itself is single-threaded
  - How can we tell if a language provides concurrency?
- AJAX provides a form of concurrency
  - Create XMLHttpRequest object, set callback function
  - Call request method, which continues asynchronously
  - Reply from remote site executes callback function
    - Event waits in event queue...
  - Closures important for proper execution of callbacks
- Another form of concurrency
  - use SetTimeout to do cooperative multi-tasking
    - Maybe we will explore this in homework ...

## JavaScript eval

- Evaluate string as code
  - The eval function evaluates a string of JavaScript code, in scope of the calling code
- Examples

```
var code = "var a = 1";
eval(code); // a is now '1'
var obj = new Object();
obj.eval(code); // obj.a is now 1
```
- Most common use
  - Efficiently deserialize a large, complicated JavaScript data structures received over network via XMLHttpRequest
- What does it cost to have eval in the language?
  - Can you do this in C? What would it take to implement?

## Unusual features of JavaScript

- Some built-in functions
  - Eval, Run-time type checking functions, ...
- Regular expressions
  - Useful support of pattern matching
- Add methods to an object
  - Seen examples. Do you like this? Disadvantages?
- Delete methods of an object
  - myobj.a = 5; myobj.b = 12; delete myobj.a;
- Iterate over methods of an object
  - for (variable in object) { statements }
- With statement (“considered harmful” – why??)
  - with (object) { statements }

## What’s a scripting language?

- One language embedded in another
  - A scripting language is used to write programs that compute inputs to another language processor
    - Embedded JavaScript computes HTML input to the browser
    - Shell scripts compute commands executed by the shell
- Common characteristics of scripting languages
  - String processing – since commands often strings
  - Simple program structure
    - Avoid complicated declarations, to make easy to use
    - Define things “on the fly” instead of in complex program
  - Flexibility preferred over efficiency, safety
    - Is lack of safety a good thing? Maybe not for the Web!

## References

- Brendan Eich, slides from ICFP conference talk
  - [www.mozilla.org/js/language/ICFP-Keynote.ppt](http://www.mozilla.org/js/language/ICFP-Keynote.ppt)
- Tutorial
  - <http://www.w3schools.com/js/> (still there?)
- JavaScript 1.5 Guide
  - [http://developer.mozilla.org/en/docs/Core\\_JavaScript\\_1.5\\_Guide](http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Guide)
- Douglas Crockford site
  - <http://www.crockford.com/JavaScript/>
  - <http://20bits.com/2007/03/08/the-philosophy-of-JavaScript/>