References

• Brendan Eich, slides from ICFP conference talk
• Tutorial
  – http://www.w3schools.com/js/
• JavaScript 1.5 Guide
• Douglas Crockford
  – http://www.crockford.com/JavaScript/
• David Flanagan
Simple Example

<!DOCTYPE html>
<html>
<body>
<h1>My First Web Page</h1>

<script>
    document.write("<p>My First JavaScript</p>"zbek);
</script>

</body>
</html>
Another Example

<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>

<p id="demo">My First Paragraph.</p>

<script>
    document.getElementById("demo").innerHTML="My First JavaScript";
</script>

</body>
</html>
Why talk about JavaScript?

• Very widely used, and growing
  – Web pages, AJAX, Web 2.0
  – Increasing number of web-related applications
• Illustrates core PL concepts
  – First-class functions
  – Objects, in a pure form
• Some interesting trade-offs and consequences
  – Powerful modification capabilities
    • Add new method to object, redefine prototype, access caller ...
  – Difficult to predict program properties in advance
    • Challenge for programmers, implementation, security, correctness
Keys to Good Language Design

• Motivating application
  – C: systems prog, Lisp: symbolic computation, Java: set-top box, JavaScript: web scripting

• Abstract machine
  – Underlying data structures that programs manipulate
  – JavaScript: web page -> document object model

• Theoretical considerations
  – ECMA Standard specifies semantics of JavaScript
What’s a scripting language?

• One language embedded in another
  – A scripting language is used to write programs that produce 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 elsewhere in program
  – Flexibility preferred over efficiency, safety
    • Is lack of safety a good thing? Maybe not for the Web!
  • Small programs
JavaScript History

- Developed by Brendan Eich at Netscape, 1995
  - Scripting language for Navigator 2
- Later standardized for browser compatibility
  - ECMAScript Edition 3 (aka JavaScript 1.5) -> ES5, ...
- 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/
  - Browser JavaScript consoles
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, 2005

• Common uses of JavaScript have included:
  – Form validation
  – Page embellishments and special effects
  – Dynamic content manipulation
  – Web 2.0: functionality implemented on web client
    • Significant JavaScript applications: Gmail client, Google maps
Design goals

• Brendan Eich’s 2005 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!
JavaScript design

• Functions based on Lisp/Scheme
  – first-class inline higher-order functions
    function (x) { return x+1; }
• Objects based on Smalltalk/Self
  – var pt = {x : 10, move : function(dx){this.x += dx}}
• Lots of secondary issues ...
  – “In JavaScript, there is a beautiful, elegant, highly expressive language that is buried under a steaming pile of good intentions and blunders.”
    Douglas Crockford
Sample “details”

• Which declaration of g is used?

```javascript
var f = function() {
    var a = g();
    function g() { return 1; }
    function g() { return 2; }
    var g = function() { return 3; }
    return a;
}
var result = f(); // what is result?
```

```javascript
var scope = "global";
function f() {
    alert(scope);
    var scope = "local";
    alert(scope);
} // variable initialized here
// but defined throughout f
```
What makes a good programming language design?
Language syntax

• 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 a 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
Stand-alone implementation

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

—

```plaintext
js> function add(x, y) { return x+y; }
js> add(2,3)
5
js> add(4,5)
9
```
Web example: page manipulation

• Some possibilities
  – createElement(elementName)
  – createTextNode(text)
  – appendChild(newChild)
  – removeChild(node)

• Example: Add a new list item:

```javascript
var list = document.getElementById('list1')
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)
We will focus on JavaScript as a language, not its use in the browser
Web example: browser events

A script is added to the page to listen to the 'mousedown' event. When the mouse button is pressed, a function is called that determines which button was clicked based on the 'button' property of the event object:

```javascript
function whichButton(event) {
  if (event.button==1) {
    alert("You clicked the left mouse button!")
  } else {
    alert("You clicked the right mouse button!")
  }
}
```

The `onmousedown` event is assigned to the body of the page, causing the `whichButton` function to be called when the mouse is clicked:

```
<body onmousedown="whichButton(event)"
```

Other events include: `onLoad`, `onMouseMove`, `onKeyPress`, and `onUnLoad`. Mouse event causes page-defined function to be called.
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)=undefined`
Type Checking

- **Static types (Algol, C, Java)**
  - The type of expressions are checked at compile-time

- **Dynamic types (lisp)**
  - The type of expressions are checked at run-time
  - Soft typing can be used

- **Duck-Typing (Python, Ruby, Javascript)**
  - If an object can walk and swim like a duck, it is treated as a duck
JavaScript blocks

• Use { } for grouping; not a separate scope

```javascript
var x = 3;
x;
{
  var x = 4;  
  x
}
x;
```

• Not blocks in the sense of other languages
  – Only function calls and the *with* statement introduce a nested scope
JavaScript 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

• 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

• Simulate blocks by function definition and call

```javascript
var u = { a:1, b:2 }
var v = { a:3, b:4 }
(function (x,y) {
    // “begin local block”
    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) // “end local block”

// Side effects on u,v because objects are passed by reference
```

• Anonymous functions very useful for callbacks

```javascript
setTimeout( function(){ alert("done"); }, 10000)
// putting alert("done") in function delays evaluation until call
```
Objects

• An object is a collection of named properties
  – Simplistic view in some documentation: hash table or associative array
  – Can define by set of name:value pairs
    • objBob = {name: “Bob", grade: 'A', level: 3};
  – New properties can be added at any time
    • objBob.fullname = 'Robert';
  – A property of an object may be a function (=method)

• Functions are also objects
  – 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”;
Basic object features

• Creating and modifying objects
  var r = new Rectangle(8.5, 11);
  r. area = function () { return this.width * this.height ;}
  var a = r. area;

• Better to do it in the constructor
  function Rectangle(w, h) {
    this.width = w; this.height = h ;
    this. area = function () { return this.width * this.height ;}
  }
  var r = new Rectangle(8.5, 11);
  var a = r.area();
Code and data can be shared via Prototypes

• Rectangle with shared area computation
  
  function Rectangle(w, h) {
    this.width = w; this.height = h;
  }
  Rectangle.prototype.area = function() {
    return this.width * this.height;
  }
  var r = new Rectangle(8.5, 11);
  var a = r.area();

  Also supports inheritance (see the Definitive Guide)
Changing Prototypes

• Use a function to construct an object
  
  ```javascript
  function car(make, model, year) {
    this.make = make;
    this.model = model;
    this.year = year;
  }
  ```

• Objects have prototypes, can be changed
  
  ```javascript
  var c = new car("Tesla","S",2014);
  car.prototype.print = function () {
    return this.year + " " + this.make + " " + this.model;
  }
  c.print();
  ```
Objects and \textit{this}

\begin{itemize}
  \item Property of the activation object for function call
    \begin{itemize}
      \item In most cases, \textit{this} points to the object which has the function as a property (or \textquotedblleft method\textquotedblright).
      \item Example:
        \begin{verbatim}
        var o = {x : 10, f : function(){return this.x}}
        o.f();
        10
        \end{verbatim}
    \end{itemize}
\end{itemize}

\textit{this} is resolved dynamically when the method is executed.
var x = 5; var y = 5;
function f() {return this.x + y;}
var o1 = {x : 10}
var o2 = {x : 20}
o1.g = f; o2.g = f;
o1.g() ➞ 15
o2.g() ➞ 25
var f1 = o1.g ; f1() ➞ 10

Both o1.g and o2.g refer to the same function. Why are the results for o1.g() and o2.g() different?
Local variables stored in “scope object”

Special treatment for nested functions

```javascript
var o = { x: 10,
    f : function() {
        function g(){ return this.x } ;
        return g();
    }
};
o.f()
```

Function g gets the global object as its this property!
Language features in the course

• 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
Stack memory management

- Local variables in activation record of function

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

var x= 1; var y =2;
f(x) + y;
```
Closures

- Return a function from function call
  ```javascript
  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
  ```javascript
  uniqueld function () {
    if (!argument.calle.id) arguments.callee.id=0;
    return arguments.callee.id++;
  }
  ```
- Can implement breakpoints
Implementing Closures

```javascript
function f(x) {
    function g(y) { return x + y; };
    return g ;
}
var h = f(3);
var j = f(4);
var z = h(5);
var w = j(7);
```

Diagram:

```
  global
     ↓        ↓        ↓
   undefined w z j undefined
     ↓        ↓        ↓
   undefined undefined undefined
```

Implementing Closures(1)

function f(x) {
    function g(y) { return x + y; };
    return g ;
}
var h = f(3);
var j = f(4);
var z = h(5);
var w = j(7);
Implementing Closures(2)

function f(x) {
    function g(y) { return x + y; };
    return g ;
}

var h = f(3);
var j = f(4);
var z = h(5);
var w = j(7);

```javascript
var h = f(3);
var j = f(4);
var z = h(5);
var w = j(7);
```
Implementing Closures(3)

function f(x) {
    function g(y) { return x + y; };  
    return g ;  
}

var h = f(3);
var j = f(4);
var z = h(5);
var w = j(7);
function f(x) {
    function g(y) { return x + y; };
    return g ;
}

var h = f(3);
var j = f(4);
var z = h(5);
var w = j(7);
h= null;
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/orelly/web/jscript/ch11_07.html
Discuss garbage collection in connection with memory management
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
  }

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 ...
Unusual features of JavaScript

• Some built-in functions
  – Eval (next slide), Run-time type checking functions, ...

• Regular expressions
  – Useful support of pattern matching

• Add, delete methods of an object dynamically
  – Seen examples adding methods. Do you like this? Disadvantages?
  – myobj.a = 5; myobj.b = 12; delete myobj.a;

• Redefine native functions and objects (incl undefined)

• Iterate over methods of an object
  – for (variable in object) { statements }

• With statement (“considered harmful” – why??)
  – with (object) { statements }
JavaScript eval

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

- Examples
  ```javascript
  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?
Other code/string conversions

• String computation of property names

```javascript
var m = "toS"; var n = "tring";
Object.prototype[m + n] = function(){return undefined};
```

• In addition
  • for (p in o){....}
  • o[p]
  • eval(...)  
    allow strings to be used as code and vice versa
Sample Javascript delights

• return
  {
    status: true
  };

• var length = 3;
  console.log(length);

• What does “a + b” mean

• Falsy values: 0, Nan, “”, false, null, undefined
Lessons Learned

• Few constructs make a powerful language
• Simplifies the interpreter
• But the interaction can be hard to understand for programmers
  — JSLint

• Hard for compilation, verification, ...
• Some efforts to create safe Javascript sub-languages (Adsafe,...)