VOICES THAT MATTER CONFERENCE

GOOGLE WEB TOOLKIT

JavaScript Bootcamp

Alexei White - Nitobi

A bit about me

-Development:

 -Ruby/Rails, PHP, Coldfusion, Some ASP.NET, C++ back in the day, Turbo Pascal, etc, etc

-Design

-Enterprise Ajax

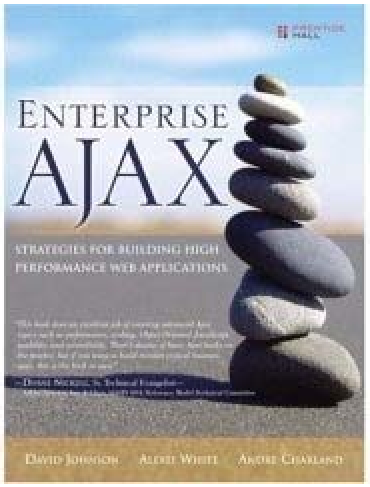
-Recent Projects:

 -CompleteUI Component Suite

 -Nintendo

 -RobotReplay

 -SayZu



nitobi
built for people

About Nitobi

- Rich Internet Application development.

- Off-the-shelf Ajax UI components

- Cross Platform
  - •Java
  - •ASP.NET
  - •PHP
  - •Coldfusion
  - •Classic ASP
  - •Ruby on Rails

Fidelity INVESTMENTS
CISCO
SITA
USAID
Adobe
Nintendo
Lucent Technologies Bell Labs Innovations
Bank of America
SIEMENS
CERIDIAN
TimeWarner
H&R BLOCK
BMW
DEPARTMENT OF THE NAVY UNITED STATES OF AMERICA
Duke Energy
Nestlé
Business Objects
tyco
Goldman Sachs
NASA
IKEA
Coleman
BANK OF SCOTLAND
HunterDouglas

Bootcamp Goals

- Develop practical understanding of ECMAScript
- Actual coding
- Not too deep, not too shallow
- Remove mystery behind Rich Internet Applications
- Expose the trouble spots

Today's Format

- **Part 1 (Basics)**
  - Lexical Structure
  - Datatypes
  - Variables
  - Operators, Expressions & Statements

  - 10 minute break

- **Part 2 (More Advanced)**
  - Debugging
  - Basic DOM
  - **Exercise 1**
  - Threading
  - JavaScript & DHTML
  - **Exercise 2**

  - 10 minute break

- **Part 3 (More Advanced)**
  - Object Oriented Programming
  - Ajax (XHR)
  - DOM Events
  - **Exercise 3**

VOICES THAT MATTER CONFERENCE GOOGLE WEB TOOLKIT

JavaScript Gotcha's

- These slides will highlight
  - Some browser-specific difficulty
  - General wierdness
  - Something else

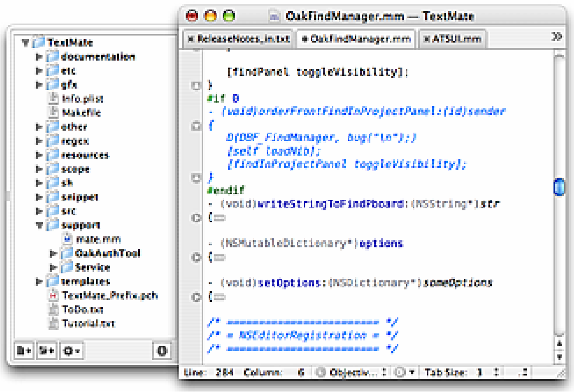
IDE's - PC

- **Aptana Studio**
  - Integrated Debugging (IE/FF)
  - Library Support
    - AIR, Rails, PHP, etc
  - SVN Support
- Visual Studio
  - Integrated debugging
  - Great Project Support
  - Intellisense
- JSEclipse
- Text Editor
  - Notepad++, Textedit
- Dreamweaver

I recommend

- MAC

- **Textmate**
  - project support
  - syntax highlighting
  - snippets
- Aptana
- JSEclipse
- Text Editor's



```
   [findPanel toggleVisibility];
}
#if 0
- (void)orderFrontFindInProjectPanel:(id)sender
{
   D(DBF_FindManager, bug("\n");)
   [self loadNib];
   [findInProjectPanel toggleVisibility];
}
#endif
- (void)writeStringToFindPboard:(NSString*)str
{

- (NSMutableDictionary*)options
{

- (void)setOptions:(NSDictionary*)someOptions
{

/* ======================== */
/* = NSEditorRegistration = */
/* ======================== */
```

VOICES THAT MATTER CONFERENCE

GOOGLE WEB TOOLKIT

Part 1

The Basics

Essence of JavaScript (1/2)

- JavaScript != Java
- JavaScript != Simple

Essence of JavaScript (2/2)

- ECMAScript
- Dynamic
- Client-side
- Prototype based
- Associative arrays
- Weakly typed

```
Obj.x = 10;
Obj[“x”] = 10;
```

Understanding the Client-Server Boundary

- JavaScript has no secrets
- Client is unknown
- Dynamic – eval()
- Cross-site communication restricted

The impact of Browsers

- Sandbox implementation errors
- Minor implementation differences
  - JavaScript
  - Cascading Style Sheets
  - Layout
  - Supported media types

What Can you do in JavaScript?

- Draw boxes, images, and text
- Open and close windows
- Animate on-screen contents
- Modify the document
- Communicate with the server
- Talk to Java, Flash, Silverlight
- Snoop on the user.. record what they do.
- Read the mouse/keyboard

What it can’t do (1/2)

- Can’t open files
- Write to the file system
- Talk directly to hardware
- Read freely from memory
- Perform general networking
  - open a socket etc
- Do Ajax across domains

“Security Sandbox”

What it can’t do (2/2)

- Can’t close windows willy-nilly
- Hide the destination of links
- Open windows that are too small
- Set the value of FileUpload fields
- Rotate graphics
- Make sound

VOICES THAT MATTER CONFERENCE

GOOGLE WEB TOOLKIT

JAVASCRIPT BASICS

Lexical Structure

Lexical Structure

- All Unicode, all the time
  - UTF-16 can represent most languages
  - String operators respect encoding
- Case Sensitive (Most of the time)

```
>>> var A = 1;
>>> var a = 2;
>>> a
2
>>> A
1
```

~~var a = 0;~~
~~WHILE(a <= 2) {~~
~~a+=1;~~
~~};~~

```
var a = 0;
while (a <= 2)
{
    a+=1;
};
```

Lexical Structure

- Whitespace ignored

```
var a = 2;
if (a == 2)
    console.log('yep');
console.log('Carrying on');
```

```
var a = 2; if (a == 2) console.log('yep'); console.log('Carrying on');
```

- Optional semicolons

```
a = 3
b = 4
```

Will be executed as

```
a = 3;
b = 4;
```

```
return
true;
```

Will be executed as

```
return;
true;
```

But you probably meant

```
return true;
```

Whitespace & Semicolons

- Filesizes reduced by removing whitespace
- Missing semicolons can cause JS errors

```
a = 3
b = 4
```

Will be become

```
a = 3 b = 4
```

No Worky

Lexical Structure

- Comments

```
// This is a single-line comment.
/* This is also a comment */    // here is another comment

/*
 * This is a multiline comment
 *
*/
```

- Literals

```
12                 // The number 12
“hello world”      // A string of text
‘hey now’          // Another string
true               // A boolean value
/javascript/gi     // A “regular expression” literal
Null               // Absense of an object
```

Lexical Structure

- Object Literals

```
{ name: “Alexei”, weight: 160 }    // An object initializer
[23,345,12,341,2]                  // An array initializer
[{a:2},{a:56,name:’Buck’},{c:65}] // An array of objects..
```

- Identifiers (Variables)

```
a
my_variable_name
v382
_something
$
$fgj
```

~~34fg~~
~~.vgf~~

Lexical Structure

- Reserved Words

`break`	`null`
`case`	`return`
`catch`	`switch`
`continue`	`this`
`default`	`throw`
`delete`	`true`
`do`	`try`
`else`	`typeof`
`false`	`var`
`finally`	`void`
`for`	`while`
`function`	`with`
`if`	
`in`	
`instanceof`	
`new`	

- Reserved & Special

A biggie

`abstract`	`int`
`boolean`	`interface`
`byte`	`long`
`char`	`native`
`class`	`package`
`const`	`private`
`debugger`	`protected`
`double`	`public`
`enum`	`short`
`export`	`static`
`extends`	`super`
`final`	`synchronized`
`float`	`throws`
`goto`	`transient`
`implements`	`volatile`
`import`	

Lexical Structure

- Words to avoid

A biggie

`arguments`	`NaN`
`Array`	`Number`
`Boolean`	`Object`
`console`	`parseFloat`
`Date`	`parseInt`
`decodeURI`	`RangeError`
`decodeURIComponent`	`ReferenceError`
`encodeURI`	`RegExp`
`Error`	`String`
`escape`	`SyntaxError`
`eval`	`TypeError`
`EvalError`	`undefined`
`Function`	`unescape`
`Infinity`	`URIError`
`isFinite`	
`isNan`	
`Math`	

JAVASCRIPT BASICS

Datatypes

Datatypes

- Number
- String
- Boolean
- null
- undefined

Primitive types

Reference types

- Objects
  - Arrays
  - Functions
  - Dates
  - RegExp
  - Error

Datatypes – Useful Information

- Numbers
  - All floating point
  - 64 bit (huge)
    - (+/-) $1.7976931348623157x10^{308}$
    - Small: 1.474334E-32
  - Hexadecimal Literals
    - 0xff , 0xCAFE911
  - Numeric Constants
    - Infinity, NaN, Number.MAX_VALUE, Number.NEGATIVE_INFINITY

Datatypes – Useful Information

- Strings
  - Modern ECMAScript implementations support Unicode
  - Escape sequences:
    - ‘You\’re always saying what can\’t be done’
    - \u for unicode (eg: \u03c0 for π)
  - Concatenation
    - Myname = first_name + ‘ danger ‘ + last_name;
  - Strings are objects
    - Myname.length, Myname.indexOf()
  - Convert numbers to strings
    - Msg = 100 + ‘’; Msg = String(100); Msg = 100.toString();
  - Strings to Numbers
    - Myval = parseInt(“3 people”) // 3

Datatypes – Useful Information

- Boolean Values
  - Often the result of comparisons (a == 4)
  - Used in control structures:

```
if (a == 4)
  b = b + 1;
else
  b = b – 1;
```

  - Type conversions are often automatic
    - Numeric : true == 1, false == 0

```
c = true + true    // 2
d = true + ‘’     // ‘true’
```

  - Type conversion is easy

```
var x = Boolean(1);        // true
a = 1;
var y = !!a;               // true
```

Datatypes – functions

```
function square(x) {  // function is called square, expects 1 argument
  return x*x;         // the function squares its arg and returns result
}                     // function ends here
```

- Functions can be stored in variables, arrays, etc.
- Can define lambda or anonymous functions

```
function square(x) {
  return x*x;              =     var square = function(x){ return x*x; }
}
```

Datatypes – objects

- JavaScript has built-in objects

```
document.forms                window.innerWidth
document.images               window.scrollY
```

- Objects are associative arrays

```
document.forms                document[“forms”]
document.images        =      document[“images”]
```

- Invoked using ***new*** keyword

```
var a = new Object();   ——————————————————>   a.x = 1.1;  a.y = 344;
var now = new Date();
var pattern = new RegExp(“\\sjava\\s”, “i”)
```

Datatypes – objects

- Object literals
  - Comma-separated list of name-value pairs
  - AKA JSON (JavaScript Object Notation)

```
var vertex = { x:3.2, y:13.2, z:64.3 };
var user = {
    "fullName": "John Smith",
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "postalCode": 10021
    },
    "phoneNumbers": [
        "212 732-1234",
        "646 123-4567"
    ]
}
```

Datatypes – objects

- Object conversion: Does an object exist?

```
if (myObj) {
  // Do something
}
```

- Objects as strings

```
a = {b:34,t:4}
b = a + “”        // “[object Object]”
```

Datatypes - arrays

- A collection of values, like an object
- Can contain any type of JS data

```
var a = new Array(1.1, “something”, true, {x:43,y:34});

var b = new Array();
b[0] = 1.1;
b[1] = “something”;
b[2] = true;
b[3] = {x:43,y:34};

var c = new Array(20);
```

Datatypes - arrays

- Array literals

```
var a = [1.2, “something”, true, {x:33,y:34}];
```

- Sparse arrays

```
var a = [1,,,,5];
```

- Array length

```
a.length
```

- Searching

```
var array = [2, 5, 9];
var index = array.indexOf(5); // index is 1
index = array.indexOf(7);  // index is -1
```

Datatypes – null & undefined

- null
  - Indicates no value
  - No object
  - Converts to false in Boolean context (if (null))
- undefined
  - Not null
  - Used when a variable is declared but has no value
- Comparing the two
  - null == undefined
  - null !== undefined

Datatypes - dates

- Not a fundamental type, but a class of object
- Easily use local or GMT time.

```
var now = new Date(); // create an object holding current
                      // date and time
var xmas = new Date(2007,11,25);
```

- Date math, and type conversion

```
xmas.setFullYear(xmas.getFullYear() + 1);  // Next christmas
var weekday = xmas.getDay();               // Day of week

console.log(“Today is: “ + now.toLocaleString());
                 // Convert to string
```

More on this Later!

Datatypes – regular expressions

- Pattern matching
- Search and replace

```
/^NITOBI/
/[1-9][0-9]*/
/\bnitobi\b/i
```

- Use the string method replace

```
Mystr.replace(/\bnitobi\b/I, “NITOBI”);    // capitalize all instances
                                           // of NITOBI.
```

Datatypes – error objects

- Represents a runtime error
- Besides the base Error, there are six other core error types in JavaScript
  - **EvalError**
  - **RangeError**
  - **ReferenceError**
  - **SyntaxError**
  - **TypeError**
  - **URIError**
- Contains the following properties:
  - **constructor**
  - **message**
  - **name**
  - **prototype**

More on this Later!

```
try {
    throw new Error("Whoops!");
} catch (e) {
    alert(e.name + ": " + e.message);
}
```

Comparing Types

- Equality vs Identity

```
var a =1;
var b = true;
a == true;        // true

a === true;       // false
b === true;       // true
a !== true;       // true
```

- Inspecting an identifier (typeof)

```
typeof(a);        // “number”
typeof(b);        // “boolean”
```

Value vs Reference

- 3 ways to manipulate data values
  - Copy it
  - Pass it as an argument
  - Compare it
- 2 ways to manipulate variables
  - By value
  - By reference

Value vs Reference

```
// copying by Value
var a = 1;
var b = a;       // copy by value.. Two distinct values now


// passing an argument by Value
function add_to_sum(sum, x) {
  sum = sum + x;          // this only changes the internal
}                         // copy of x


add_to_sum(a,1);   <----------   Won't actually do
                                 anything


if (b == 1) b = 2;        // b is a distinct numeric value
                          // a is still 1, and b is 2
```

Value vs Reference

```
// copying by reference
var xmas = new Date(2007, 11, 25);
var gift_day = xmas;      // both variables refer to the same object

gift_day.setDate(26);     // we've now changed both variables

xmas.getDate();           // returns 26 not 25

function add_to_sum(sum, x) {
         sum[0] = sum[0] + x;
         sum[1] = sum[1] + x;
         sum[2] = sum[1] + x;
}
(xmas == gift_day)        // this is true still

var newdate1 = new Date(2007,10,10);
var newdate2 = new Date(2007,10,10);
(newdate1 == newdate2)    // this is false
```

Permanently changes 'sum' in global context

Note:
Strings are compared by value

JAVASCRIPT BASICS

Variables

Typing

- JavaScript is weakly typed
  - A variable can hold a value of any type at any time

```
i = 10;
i = "ten";
```

OK, no problem.

Variable Declaration

- Explicit declaration not required
  - Implicit declaration has scope consequences

```
var i;             var message = “hello”;
var sum;           var i = 0, j = 0, k = 43;
var i, sum;
```

- Repeated declaration OK too.

```
var i = 1;          // 1
var i = 2;          // 2
```

Variable Scope

- Global – available everywhere
- Local – available within a function
- Local > Global

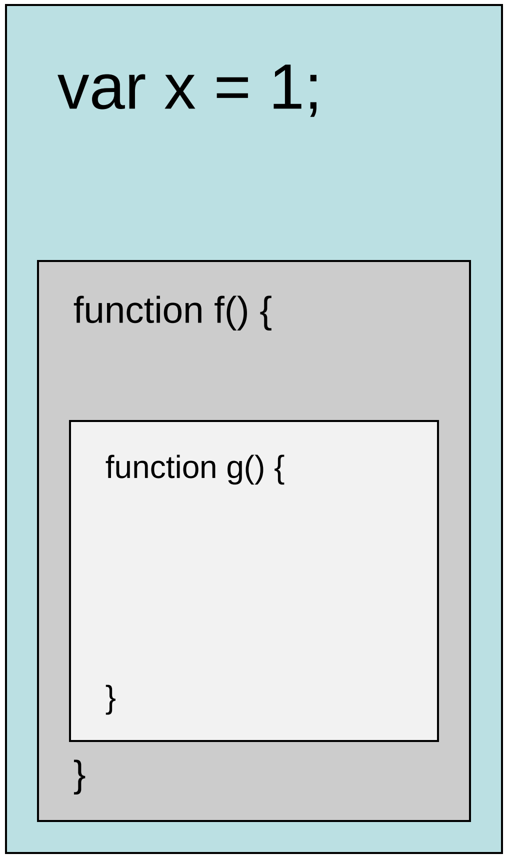
```
var scope= “global”;

function checkscope() {
   var scope = “local”;
   console.log(scope);    // prints ‘local’;
}

console.log(scope);       // prints ‘global’;
```

Note:
No Block Scope
except SWITCH
& WITH

Variable Scope

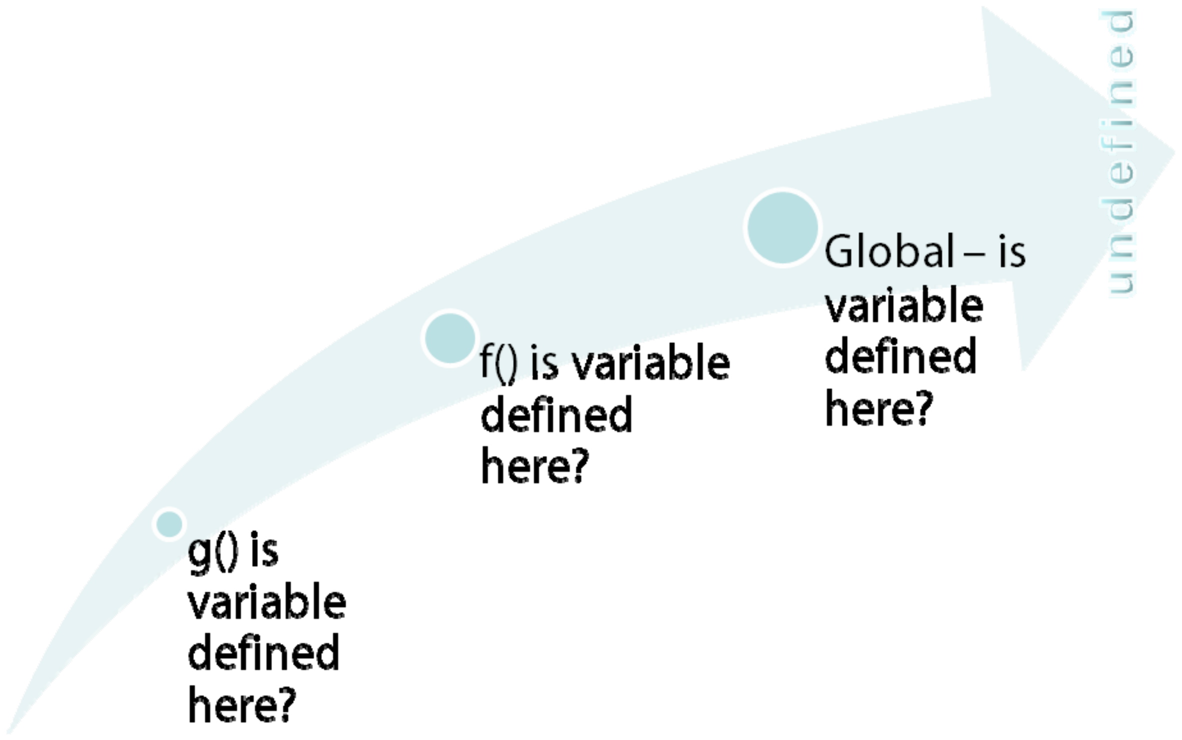


```
var x = 1;

function f() {

    function g() {

    }
}
```

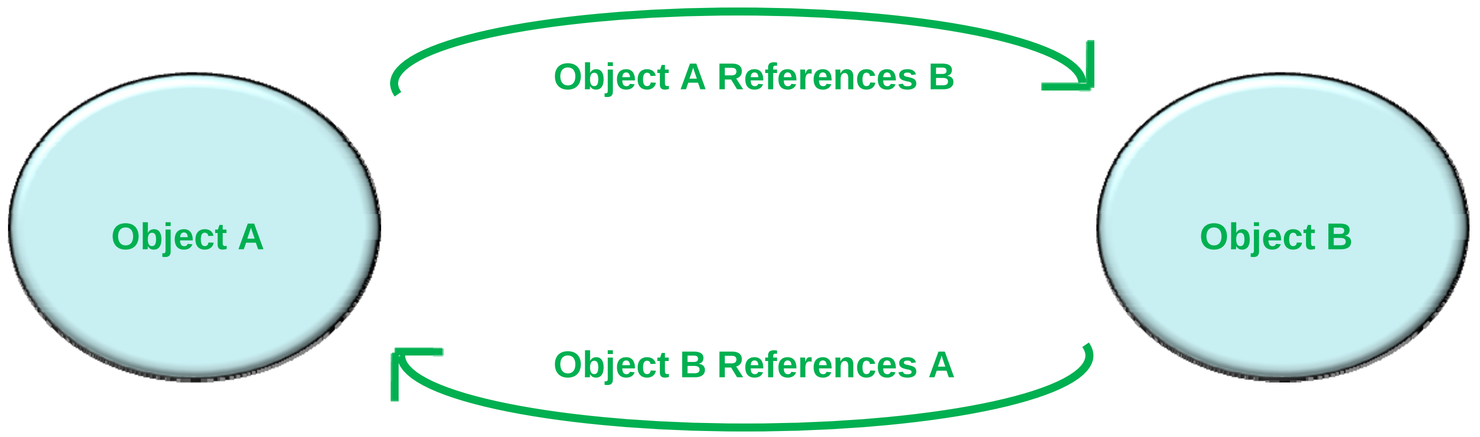

g() is variable defined here?
f() is variable defined here?
Global – is variable defined here?
undefined

Garbage Collection

- Memory allocated and deallocated automatically
- Interpreter detects when allocated memory is unreachable

```
var s = “hello”;            // allocate some mem
var u = s.toUpperCase();  // copy the value to a new string
s = u;                      // “hello” now unreachable and is destroyed
```

Explorer Memory Leaks



- Circular references can cause memory leaks.

VOICES THAT MATTER CONFERENCE
GOOGLE WEB TOOLKIT

JAVASCRIPT BASICS

Operators, Expressions & Statements

Operators

Operator	Operand type(s)	Operation performed
.	Object, identifier	Property access
[ ]	Array, integer	Array index
!	Boolean	Logical complement
==	Any	Equality
===	Any	Identity
&&	Booleans	Logical AND
||	Booleans	Logical OR
?:	Booleans, any, any	Conditional operator
,	Any	Multiple evaluation

Assignment with Operation

Operator	Example	Equivalent
+=	a += b	a = a + b
-=	a -= b	a = a – b
*=	a *= b	a = a * b
/=	a /= b	a = a / b
%=	a %= b	a = a % b
<<=	a <<= b	a = a <<b
>>=	a >>= b	a = a >>b
>>>=	a >>>= b	a = a >>> b
&=	a &= b	a = a & b
|=	a |= b	a = a | b
^=	a ^= b	a = a ^ b

Conditional Operator (?:)

- Ternary operator (three operands)

```
greeting = “hello “ + (username != null ? username : “there”);
```

- is equivalent to..

```
greeting = “hello “;
if (username != null)
        greeting += username;
else
        greeting += “there”;
```

- Fast, and compact.

Don’t rely on a compiler for code optimization

Notes about Statements

- Brackets around expressions are required

```
if (expression)
        statement
```

- Single statements do not require curly-braces

```
if (expression)                 if (expression) {
        statement                       statement
                                        statement
                                }
```

- Whitespace ignored

```
if (expression) statement  else  statement
```

If / else if

- Execute multiple pieces of conditional code

```
if (n == 1) {
        // Execute code block 1
} else if (n == 2) {
        // Execute code block 2
} else if (n == 3) {
        // Execute block 3
} else {
        // if all else fails, do this
}
```

```
if (n == 1) {
        // block 1
} else {
        if (n == 2) {
                // block 2
        } else {
                if (n == 3) {
                        // block 3
                } else {
                        // do this
                }
        }
}
```

Equivilent

Switch

- Better if you are just checking the same var over and over

```
switch(n) {
 case 1:                    // start here if n == 1
         // code block 1
         break;
 case 2:
         // code block 2
         break;
 case 3:
         // code block 3
         break;
 default:                   // if all else fails…
         // code block 4
         break;
}
```

Only use constants in CASE expressions

while, do/while

```
var count = 0;
while (count < 10) {
        console.log(count);
        count++;
}

var count = 0;
do {
        console.log(count);
        count++;
} while (count < 10)
```

Will execute at least once

for

```
for(initialize ; test ; increment)
        statement

for (var count = 0; count < 10; count++)
        console.log(count);

for(variable in object)
        statement

var o = {x:1, y:2, z:3}
var a = new Array();
var I = 0;
for(a[i++] in o) {}
```

Curly braces {} are not required if just one statement

Copies the object "o" to the array "a"

Not all properties are enumerable!

Performance Pitfall

re-calculated EVERY TIME --SLOW-- !!

- Array.length is expensive

```
for (var count = 0; count < myarray.length; count++)
        console.log(count);
```

- better:

```
for (var count = 0, mylen = myarray.length; count < mylen; count++)
        console.log(count);
```

- best:

```
for (var count = myarray.length-1; count > 0; count--)
        console.log(count);
```

labels

- Any statement may be labeled

```
myloop:
        while(something != null) {
                // code block
        }
```

- Usually just used for loops
- Used for breaking and continuing

break

- Causes the innermost enclosing loop or switch to exit.

```
break;
```

- Can be combined with labels

```
outerloop:
  for(var i = 0; i < 10; i++) {
        innerloop:
          for(var j = 0; j < 10; j++) {
                if (j > 3) break;    // quit innermost loop
                if (i == 2) break innerloop;
                if (i == 4) break outerloop;
          }
  }
```

continue

- Like break.. but just skips to the next iteration of the current loop

```
for(i = 0; i < data.length; i++) {
        if (data[i] == null)
                continue;
        total += data[i];
}
```

- Can be used with labels

try/catch/finally/throw

- Create an exception

```
try {
        43534987yhfh     // clearly an error
} catch(myerr) {
        console.log(myerr);
} finally {
        //code block
}
```

always executes regardless of what happens in catch() or try()

- Custom exceptions

```
try {
        throw(“User entered invalid data..”);
} catch(myerr) {
        console.log(myerr);
} finally {
        //code block
}
```

Will write “Something bad happened..”

with

- Code block that modifies the scope chain

```
with(frames[1].document.forms[0]) {
        // access form elements directly here. eg:
        name.value = “something”;
        address.value = “someplace”;
        email.value = “me@home.com”;
}
```

- Generally avoided.
  - slow
  - some unexpected behaviors (with init’d vars)

; (empty)

- Has no effect.. but can be useful.

```
var o = {x:1, y:2, z:3}
var a = new Array();
var I = 0;
for(a[i++] in o) ;
```

Part 2

More Advanced

VOICES THAT MATTER CONFERENCE

GOOGLE WEB TOOLKIT

ADVANCED JAVASCRIPT

Debugging

Firebug

- Free Firefox plugin (http://www.getfirebug.com)
- Shows you 2 important things:
  - What's going on in your page when it loads
  - & After it loads
- Advanced features
- Code profiling
- CSS debugging
- DOM Inspection
- JavaScript breakpoints and step-through
- XHR Debugging
- JSON Object Inspection
- Integration with Aptana

MS Script Debugger

- Step over, into
- Console window
- Not much else
- Visual Studio better (if you have it)

IE Developer Toolbar

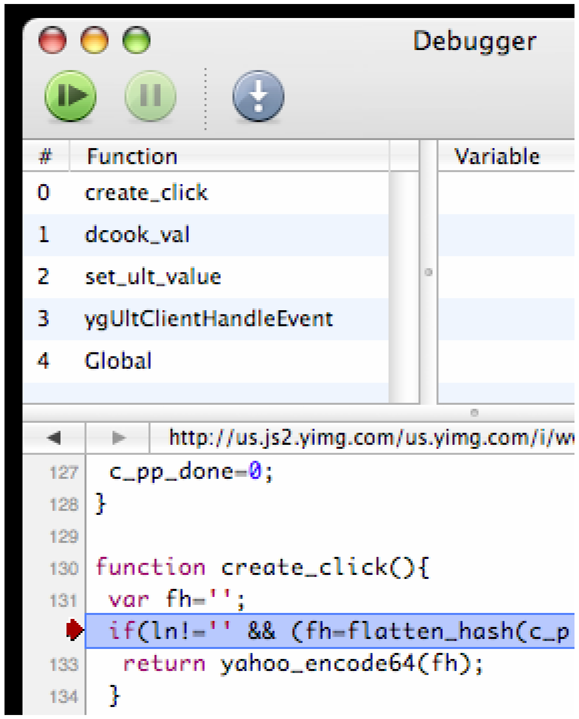
- Useful for debugging CSS in IE
- Lacks many of the features in Firebug
- Convenient cache clearing
- Change CSS attributes on the fly
- Inspect Cache
- DOM browsing

Firebug Lite

- JavaScript extension for NON-Firefox browsers
- Mimics some of the console features of Firebug
- Evaluate JavaScript in-line
- Makes cross-browser testing a lot easier.

Drosera - Safari

- WebKit JS debugger
- Code stepping
- DOM Inspection
- Mac-only

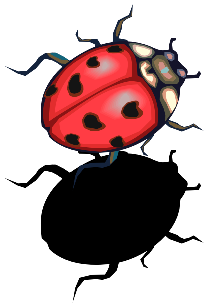


#	Function
0	create_click
1	dcook_val
2	set_ult_value
3	ygUltClientHandleEvent
4	Global

http://us.js2.yimg.com/us.yimg.com/i/w

```
  c_pp_done=0;
}

function create_click(){
  var fh='';
  if(ln!='' && (fh=flatten_hash(c_p
   return yahoo_encode64(fh);
  }
```



ADVANCED JAVASCRIPT

Basic DOM

JavaScript in the Browser

- JavaScript should be unobtrusive
  - Separation of concerns
  - Keep it away from markup
  - Modularize it
  - Degrade gracefully* ← Sometimes this is real hard

```
<html>
<head>
<script type="text/javascript" src="code.js”></script>
</head>
<body>

</body>
</html>
```

Window

- window is global context
- literally means the browser window
- global variables can be referred to as window.variable
- Document object is a member
  - Contains a hierarchical representation of document.
- Contains window info
  - Geometry
  - scroll position

```
window.innerHeight
window.innerWidth
```

VOICES THAT MATTER CONFERENCE GOOGLE WEB TOOLKIT

JavaScript Gotcha's

Window Geometry Browser Differences

Browser	window.innerHeight	document.body.clientHeight	document.documentElement.clientHeight
Opera 9.5+ strict	window	document	window
Opera 9.5+ quirks	window	window	document
Opera 7-9.2	window	window	document
Opera 6	window	window	N/A
Mozilla strict	window	document	window
Mozilla quirks	window	window	document
KHTML	window	document	document
Safari	window	document	document
iCab 3	window	document	document
iCab 2	window	window	N/A
IE 6+ strict	N/A	document	window
IE 5-7 quirks	N/A	window	0
IE 4	N/A	window	N/A
ICEbrowser	window	window	document
Tkhtml Hv3	window	window	document
Netscape 4	window	N/A	N/A

onload Event Handler

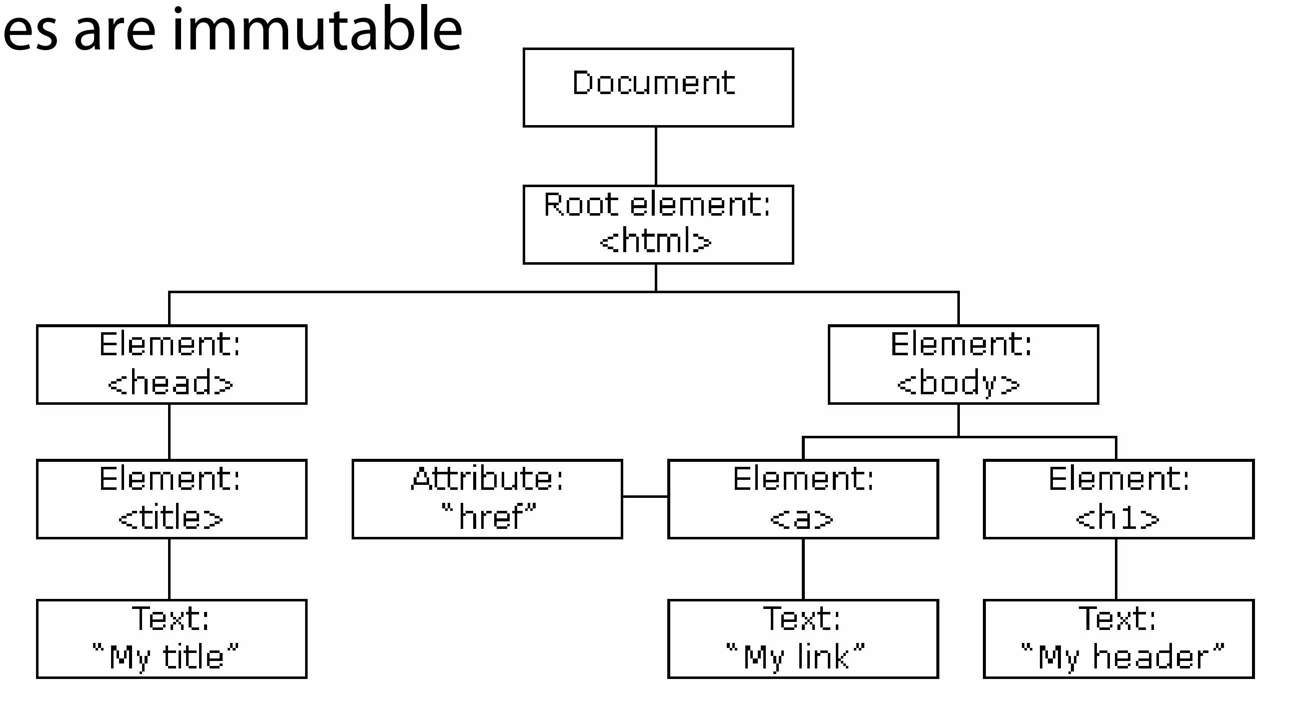
- onload fires after all HTML, CSS, Images, and JS is downloaded to the client.
- At this time, all JavaScript functions and objects part of the window object have been registered.

```
<html>
<head>
<script type="text/javascript" src="code.js”></script>
</head>
<body onload=“myFunction()”>

</body>
</html>
```

Document Object Model

- Tree-structure document
- elements, attributes, and text
  - text nodes are immutable



Form Objects, fields, etc

- Forms are held in a collection

```
var myForm = document.forms["customerForm"];
myForm = document.customerForm;
```

- Elements accessible as an array or an object

```
for (var i = 0, j = myForm.length; i < j; i++)
     console.log(myForm[i].value);

console.log(myForm.myname.value);
```

Finding HTML elements

- Get reference to an HTML element
- Use window.document object
- getElementById()
  - returns a reference to the first object with the specified ID
- getElementsByName()
  - Returns an array of objects with the specified NAME attribute
- getElementsByTagName()
  - Returns a collection of objects with the specified type (ie DIV, SPAN, TABLE, etc)

innerHTML

- Change the content of HTML element
- Read and Write property
- Cornerstone of AJAX
- Elements must have opening and closing tags: <p></p> <div></div>
  - but not <img />

```
document.getElementById('myId').innerHTML = "some new text";
```

setAttribute

- Apply HTML attributes with JavaScript

```
object.setAttribute(sName, vValue [, iFlags])


var myEl = document.getElementById('info_area');
myEl.setAttribute("class", "hoverClass");
```

createElement / appendChild

- Allow you to insert HTML into the DOM
- Faster than adding with innerHTML
  - Text nodes are immutable

```
var myDiv = document.createElement('div');
myDiv.setAttribute("class", "hoverClass2");
var body = document.getElementsByTagName('body')[0];
body.appendChild(myDiv);
```

Other ways to create elements

- Four ways to do it
  - **node.cloneNode(bool)** – creates a copy of a node.. and depending on the bool.. copies contents too.
  - **document.createElement(el)** creates a new element node
  - **document.createTextNode(txt)** creates new text node
  - **el.innerHTML** – Create elements in text and add to the DOM that way.

Frowned upon.. not always useful. Can be slow.

Adding/Removing Nodes in the Document

- **node.removeChild(oldNode)** removes the child **oldNode** from **node**
- **node.appendChild(newNode)** adds **newNode** as a new (last) child node to **node**
- **node.insertBefore(newNode, oldNode)** inserts **newNode** as a new child node of **node** before **oldNode**
- **node.replaceChild(newNode, oldNode)** replaces the child node **oldNode** of **node** with **newNode**

DOM Navigation

- node.firstChild
  - Get the first element of the child array
  - same as childNodes[0]
- node.childNodes
  - Returns collection of all nodes belonging to that node.. 1st level only.
- node.parentNode
  - Returns the element that this node belongs to
- node.nextSibling
  - Returns the next sibling belonging to this elements parent
- node.previousSibling
  - Returns the earlier sibling belonging to this elements parent
- chaining:

```
myNode.childNodes[0].childNodes[2].parentNode.nextSibling.previousSibling;
```

firstChild & Firefox

- Firefox considers text to be nodes
  - when you think about it, this is correct
  - but its not that convenient
  - whitespace matters

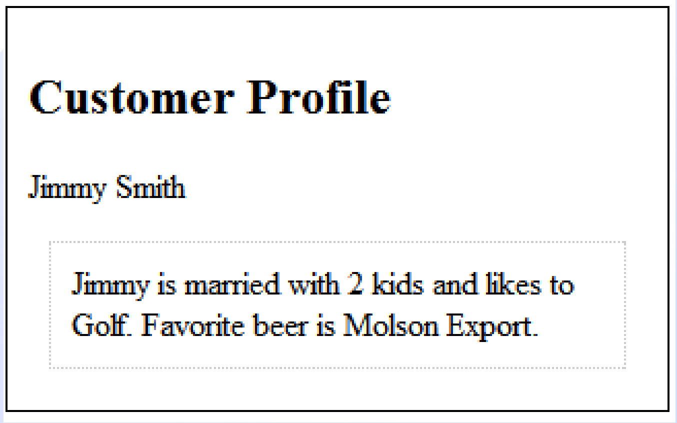
```
<table><tr><td></td></tr></table>
```

$\neq$

```
<table>
<tr>
<td></td>
</tr>
</table>
```

Exercise 1 – DOM Manipulation

- Use JavaScript and the DOM to create this document:



Customer Profile

Jimmy Smith

Jimmy is married with 2 kids and likes to Golf. Favorite beer is Molson Export.

Get optional application template at:
http://www.nitobi.com/gwt/ex1.zip

Don't worry about styling

```
<div style="border: 1px solid rgb(0, 0, 0); padding: 10px; width: 300px;">
    <h2>Customer Profile</h2>
    <p>Jimmy Smith</p>
    <div style="border: 1px dotted rgb(204, 204, 204); margin: 10px;
    padding: 10px;">Jimmy is married with 2 kids and likes to Golf.
    Favorite beer is Molson Export.</div>
</div>
```

Exercise 1 – Possible Solution

```
InsertDOM = function() {

          var myDiv = document.createElement('div');
          var myHeading = document.createElement('h2');
          myHeading.innerHTML = "Customer Profile";
          var myP1 = document.createElement('p');
          myP1.innerHTML = "Jimmy Smith";
          var myDiv2 = document.createElement('div');
          myDiv2.innerHTML = "Jimmy is married with 2 kids and likes to Golf. Favorite beer
is Molson Export.";

          // Here we asseble everything into one node
          myDiv.appendChild(myHeading);
          myDiv.appendChild(myP1);
          myDiv.appendChild(myDiv2);

          var body = document.getElementsByTagName('body')[0];
          body.appendChild(myDiv);
}
```

Modifying Style Attributes

- Get a reference to the style rule

```
var myStyle = myElement.style;
```

- Modify an attribute:

```
myStyle.backgroundColor = ‘#ffff00’;       // yellow
```

DOM Stylesheet (1/2)

- Lets you step through each rule in each stylesheet
- Change selectors
- Read/write styles
- Add new rules
- Affect several elements at same time
- document.styleSheets collection contains all
  - cssRules
  - href
  - other stuff

DOM Stylesheet (2/2)

- To find and change a class
  - loop through all stylesheets
  - look at the selectorText
  - modify your style when you’ve found it

```
if (myRules[k].selectorText == '.specialDiv') {
        myRules[k].style.position = 'absolute';
        myRules[k].style.top = '100px';
        myRules[k].style.left = '100px';
}
```

Cross Browser Alert!

- In FF/Safari you look for cssRules
- In IE you look for rules

```
var myRules = (mySheets[i].rules || mySheets[i].cssRules);
```

ADVANCED JAVASCRIPT

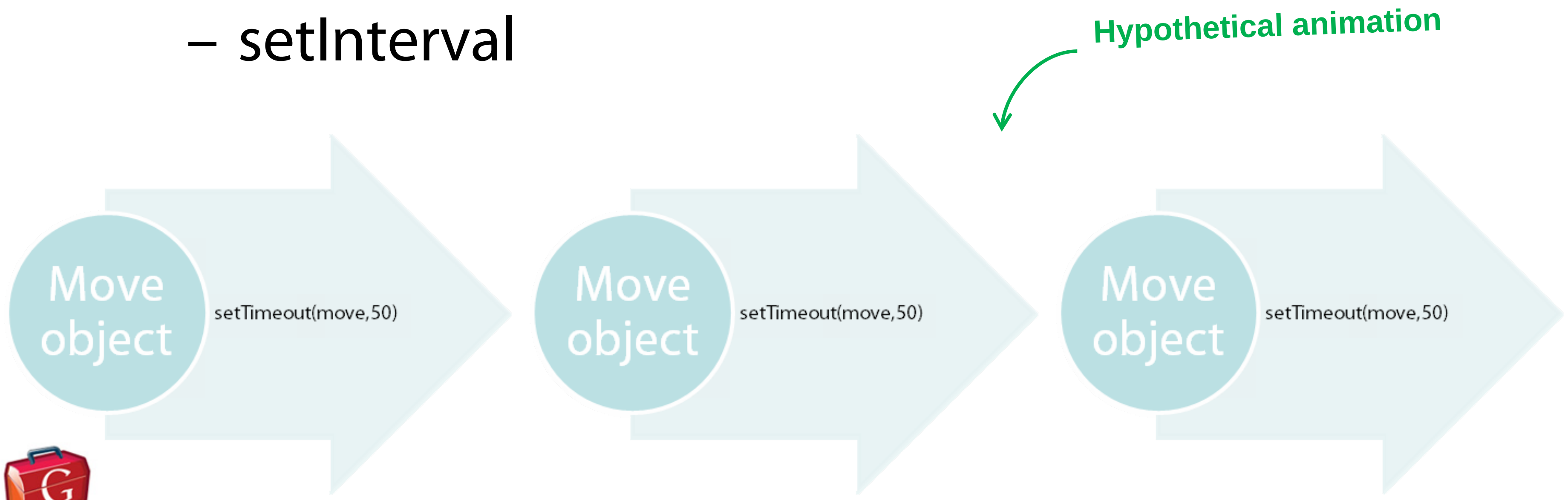
Threading

Threading Model

- JavaScript is single-threaded
- Doc parsing stops when scripts are embedded
- Browser stops responding to input when event handlers are executed
- Its possible to mimic multi-threading

Pseudo-threading in JavaScript

- Use the timer object to trigger events and JavaScript processing
  - setTimeout
  - setInterval



A Simple Thread

- Initiate a timer

```
myGlobalReference = setTimeout(function() {drawObject(50)}, 50);
```

- Do the work

```
drawObject = function(x) {
        // do some calculation or move an object a few pixels
         myGlobalReference  = setTimeout(function() {drawObject(x+1)}, 50);
}
```

- Stop the thread

```
clearTimeout(myGlobalReference);
```

VOICES THAT MATTER CONFERENCE

GOOGLE WEB TOOLKIT

ADVANCED JAVASCRIPT

JavaScript & DHTML

CSS for DHTML

- Key to dynamic HTML is modifying CSS with JS

Attribute(s)	Description
position	The type of positioning applied to an element
top,left	The position from the top left corner of the parent
width,height	Size of the element
z-index	Stacking order.. the 3rd dimension
display	Whether or not the element is rendered at all
visibility	Whether or not the element is visible
overflow	What to do with overflow content
opacity	How opaque or translucent an element is.. CSS3 attribute. IE has alternative

The KEY to DHTML

- Absolute positioning
  - element.style.position = 'absolute'
  - class { position: absolute; }
- Other options:
  - **relative** positioning – position is adjusted relative to its position in the normal flow
  - **fixed** positioning – relative to the browser window.

NO IE6

Position something with JS

- Get the element

```
myElement = document.getElementById('myEL');
```

- Set positioning (you could and shouldalso do this with a class).

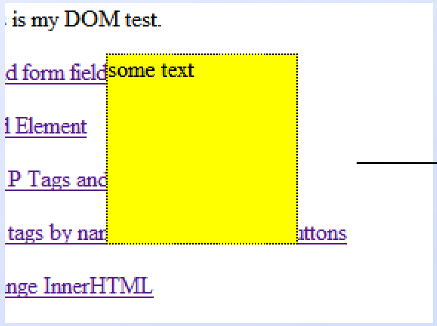
```
myElement.style.position = 'absolute';
```

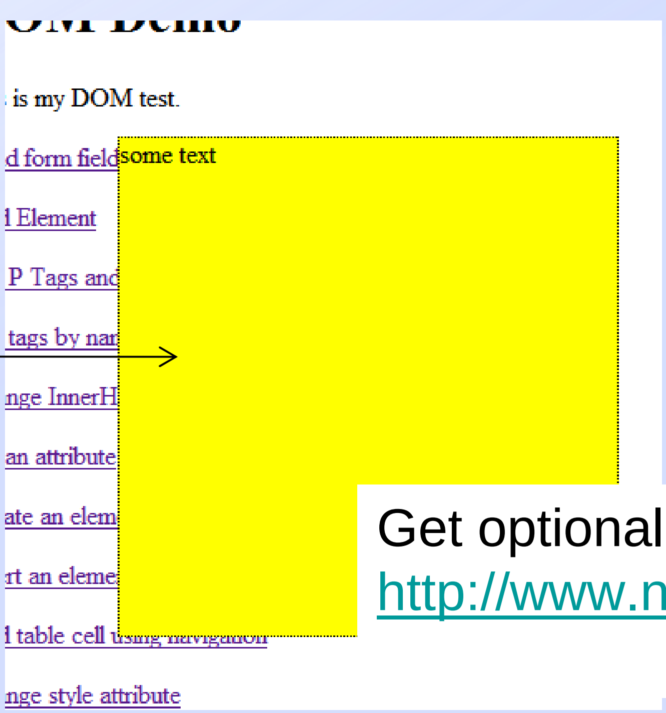
- Set coordinates

```
myElement.style.left = '20px';
myElement.style.top = '100px';
```

Exercise 2 – Animation & Threading

- Animate a box resizing itself from 100px / 100px to 300 px / 300 px over several seconds





Get optional application template at:
http://www.nitobi.com/gwt/ex2.zip

Exercise 2 – Possible Solution

```
AnimateDOMWithThread = function() {

        var myDiv = document.createElement('div');
        myDiv.style.backgroundColor = "#FFFF00";
        myDiv.style.width = "100px";
        myDiv.style.height = "100px";
        myDiv.innerHTML = "Some text";
        var body = document.getElementsByTagName('body')[0];
        body.appendChild(myDiv);
        window.myAnimObj = setTimeout(function() {animateBox(myDiv,100, 100)},
20);
}

animateBox = function(myBox, w, h) {
        myBox.style.width = w + 'px';
        myBox.style.height = h + 'px';
        var neww = w+1; var newh = h+1;
        if ((neww <= 300) || (newh <= 300))
                window.myAnimObj = setTimeout(function()
{animateBox(myBox,neww,newh)}, 20);
}
```

Absolute positioning not required

Part 3

More Advanced

VOICES THAT MATTER CONFERENCE

GOOGLE WEB TOOLKIT

ADVANCED JAVASCRIPT

Object Oriented Programming

Object Oriented JavaScript

- JavaScript is a prototypal language
- Class-based OO programming can be achieved

Java	JavaScript
Strongly Typed	Loosely Typed
Static	Dynamic
Classical	Prototypical
Classes	Functions
Constructors	Functions
Methods	Functions

Functions are Objects Too

- Important function methods:
  - call(scope, arg1, arg2 …);
  - apply(scope, [arg1, arg2 …]);
  - caller
- Call and apply used to dynamically execute a function in arbitrary scope

Using Call

```
function showLength() {
  alert(this.length);
}
```

“this” refers to the new Array

```
showLength.call(new Array(10)); // Alerts 10!
```

First, a review of an object

- This is an object

```
Person = function(fn, ln) {}
```

Public Members

- Use this keyword

```
Person = function(fn, ln) {
  this.firstName = fn;
  this.lastName = ln;
}

Person.prototype.getFullName = function() {
  return this.firstName + “ “ + this.lastName;
}

var a = new Person(“Alex”, “White”);

console.log(a.firstName)  // “Alex”
```

Private Members

- Local scope variables.
- Only accessible from within the constructor

```
Person = function(fn, ln) {
  var firstName = fn;
  var lastName = ln;
  var getFullName = function() {
    return firstName + “ “ + lastName;
  }
}
```

Privileged Members

- getFullName creates closure and therefor exposes private vars through a getter();

```
Person = function(fn, ln) {
  var firstName = fn;
  var lastName = ln;
  this.getFullName = function() {
    return firstName + " " + lastName;
  }
}
```

Classical JavaScript

- A Function() is a constructor
- Can use *new* keyword to instantiate to create new instances (copy).

```
Person = function() {
}

var john = new Person();
```

- Use *this* to add instance methods and properties

```
Person = function() {
        this.age = 12;
}
```

Basic Non-prototypal Class

- Start with a rectangle:

```
Rectangle = function(w, h) {
        this.width = w;
        this.height = h;
        this.area = function() {return this.width*this.height;}
}
```

- Create an instance:

```
var r = New Rectangle(10,20);
var a = r.area();                 // 200;
```

- Innefficient, memory hungry, inflexible

Classical JavaScript

- Use “prototype” keyword to define instance properties and methods
- Same result different approach

```
Rectangle = function(w, h) {
        this.width = w;
        this.height = h;
}

Rectangle.prototype.area = function() {
        return this.width*this.height;
}


  var r = New Rectangle(10,20);
  var a = r.area();                 // 200;
```

Another Advantage – Change the class

- Modify the prototype to add functionality to all instances of that prototype

```
<!-- … -->

Rectangle.prototype.widthSquared = function() {
        return this.width*this.width;
}

// var r = New Rectangle(10,20);   -- OUR OLD Rectangle OBJECT
var ws = r.widthSquared();                // 100;
```

Classical Inheritance

- The simple approach

```
PositionedRectangle = function(x,y,w,h) {
        Rectangle.call(this,w,h);

        // Now we store the left and right coords
        this.x = x;
        this.y = y;
}

PositionedRectangle.prototype = new Rectangle();
```

Inheritance – Simple Approach

- Why this might be bad
  - Explicit reference to Rectangle in the constructor – brittle

```
PositionedRectangle = function(x,y,w,h) {
        Rectangle.call(this,w,h);

        <!-- … -->
```

Will still work
in most cases

  - Constructor assigned to prototype – potentially brittle at compile-time if DOM is being drawn

```
PositionedRectangle.prototype = new Rectangle();
```

Inheritance Function

```
extend = function(subClass, baseClass) {
  function inheritance() {};
  inheritance.prototype = baseClass.prototype;
  subClass.prototype = new inheritance();
  subClass.baseConstructor = baseClass;
  if (baseClass.base) {
    baseClass.prototype.base = baseClass.base;
  }
  subClass.base = baseClass.prototype;
}

Customer = function (firstName, lastName) {
  Customer.baseConstructor.call(this, firstName, lastName);
  this.balance = 0;
}

Customer.prototype.getFullName = function() {
  Customer.base.getFullName.call(this);
}
extend(Customer, Person);
```

remove compile-time constructor execution

base constructor pointer

base method pointers

More on the many ways to Inherit

- http://truecode.blogspot.com/2006/08/object-oriented-super-class-method.html
- Douglas Crockford – my hero
  - http://www.crockford.com/javascript/inheritance.html

Classical Interfaces - Mixins

- No compilation in JavaScript so interfaces are tough
- Need to rely on mutability of JavaScript objects or “mixins”

```
var customer1 = new Customer();
var customer2 = new Customer();

customer1.pay = function(amout) {
  this.balance -= amount;
}

customer1.pay();
customer2.pay(); // ERROR!
```

Classical Interfaces - Mixins

- Mutate classes at runtime
- Think of the implications for AOP

```
var customer1 = new Customer();
var customer2 = new Customer();

Customer.prototype.pay = function(amount) {
  this.balance -= amount;
}

customer1.pay();
customer2.pay();
```

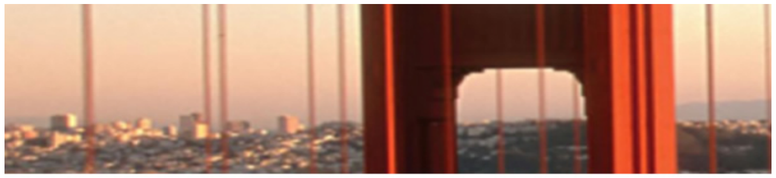
```
var f = Customer.oldFunction
Customer.oldFunction = function() {
  f.call(this);
  somethingElse();
}
```

VOICES THAT MATTER CONFERENCE

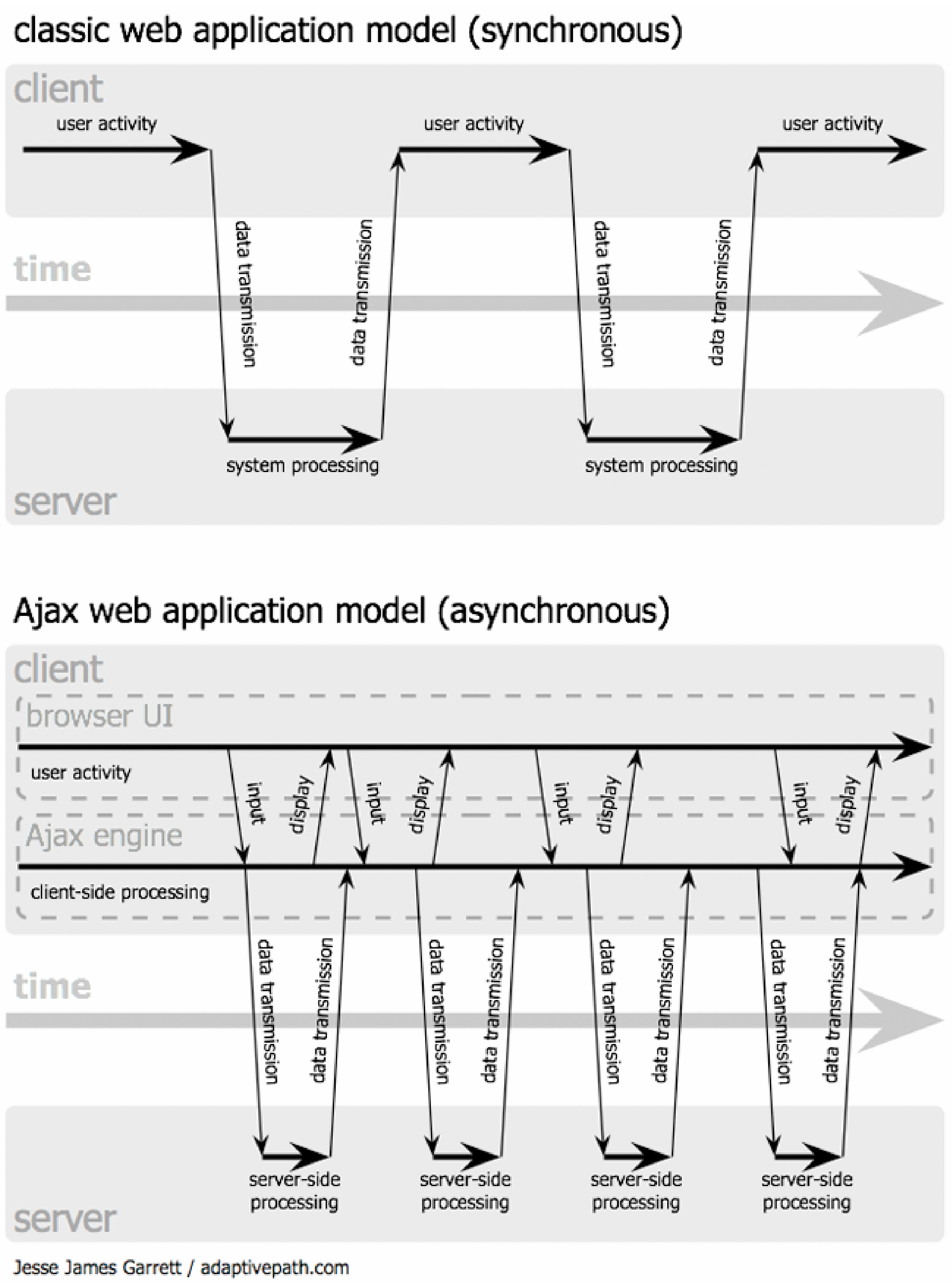
GOOGLE WEB TOOLKIT

ADVANCED JAVASCRIPT

Ajax



Ajax Versus Traditional



XMLHttpRequest

- The core of Ajax

```
var xhr = null;
try {
  xhr = new ActiveXObject("Microsoft.XMLHTTP");
} catch(e) {
  xhr = new XMLHttpRequest();
}

xhr.open("GET", "http://www.example.com/myResource", false);
xhr.send(null);
showResult(xhr);
```

IE 6 and 7

Everybody Else

Async = false

XHR Factory

- Use Factory pattern to create XHR objects in a cross-browser manner

```
xhrFactory = {
  create: function() {
    try {
      xhr = new ActiveXObject("Microsoft.XMLHTTP");
    } cstch(e) {
      xhr = new XMLHttpRequest();
    }
    return xhr;
  }
}
var xhr = xhrFactory.create();
```

Synchronous Requests

- Simplest case
- However, JavaScript thread is locked!

```
var xhr = xhrFactory.create();
xhr.open("GET", “http://www.example.com/resource”, false);
var response = xhr.send(null);
```

$\nearrow$ Async = false

Asynchronous Requests

- Use async requests to prevent locking the JavaScript thread

```
xhr.open("GET", “http://www.example.com/resource”, true);
xhr.onreadystatechange = function() {
  if (xhr.readyState == 4) {
    if (xhr.status == 200) {
      //   deal with the response
    }
  }
}
```

Async = true

Regular HTTP status code

Request Types

- GET

```
xhr.open(“GET”, “http://www.example.com/resource”, false);
var response = xhr.send(null);
```

- POST

```
xhr.open(“POST”, “http://www.example.com/resource”, false);
var response = xhr.send(“firstName=john&lastName=doe”);
```

Data Types

- POST data to the server as either XML or form encoded data
- Use XHR setRequestHeader() method

XML

```
xhr.setRequestHeader("Content-Type","text/xml");
```

Form data

```
xhr.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
```

Response Data

- We can expect the response from the server as XML, JSON, HTML or text

```
xhr.open("GET", "http://www.example.com/resource", false);
var response = xhr.send(null);

alert(response.responseXml); // Should show a [Document] for XML response
alert(response.responseText); // Should show the XML, JSON, or HTML data
```

Make sure you set the response type on the server too!

What Type of Response?

- XML
  - Good for Web Services and XML RPC
  - A bit more work on the client.. browser differences
- JSON
  - Easy, fast
- HTML
  - No rendering logic on client
  - bandwidth considerations

Just yse what you prefer.....

XML Response

- Various ways of dealing with XML data
  - XML DOM – most compatible
  - XPath – fast and easy
  - XSLT – not supported everywhere

```
xhr.open(“GET”, “http://www.example.com/resource”, false);
var response = xhr.send(null);

var html = “”;
var customers = response.responseXml.getElementsByTagName(“customer”);
for (var i=0; i<customers.length; i++) {
  var customer = customers[i];
  html += “<div>”+customer.childNodes[0].nodeValue+”</div>”;
  html += “<div>”+customer.childNodes[1].nodeValue+”</div>”;
}
alert(html);
```

JSON Response

- Need to instantiate the data into JavaScript objects

```
xhr.open(“GET”, “http://www.example.com/resource”, false);
var response = xhr.send(null);

var html = “”;
var customers = eval(“(“+response.responseText+”)”);
// OR eval(“a = “ + response.responseText);
for (var i=0; i<customers.length; i++) {
  var customer = customers[i];
  html += “<div>”+customer.firstName+”</div>”;
  html += “<div>”+customer.lastName+”</div>”;
}
alert(html);
```

HTML Response

- Take the HTML from the server and put it into the web page DOM

```
xhr.open(“GET”, “http://www.example.com/resource”, false);
var response = xhr.send(null);

var html = response.responseText
alert(html);
```

Cross-Domain XHR

- Create <script> element dynamically

```
var script = document.createElement(“script”);
script.src = “http://www.example.com/resource?callback=myCallback”;
document.getElementsByTagName(“head”)[0].appendChild(script);
```

- Response from server includes JavaScript and calls a callback function
- Called JSONP or XMLP

Dynamically generated function call

```
var customers = [{firstName:”John”,lastName:”Doe”}]
myCallback(customers);
```

Cross-Domain JSONP Security

- There are serious security risks with JSON or XMLP
- Also serious risks with JSON in general
  - Return JSON data in comments to prevent non XHR access

```
<!--
[{firstName:"John",lastName:"Doe"}]
-->
```

VOICES THAT MATTER CONFERENCE
GOOGLE WEB TOOLKIT

ADVANCED JAVASCRIPT

DOM Events

DOM Events

- Native Event object contains information about the event
- Two approaches to defining events:
  - Inline
  - Unobtrusive
- Unobtrusive approach requires cross-browser event attachment

Native Events

- Document
  - load, unload, resize, scroll
- Mouse
  - mouseover, mouseout, mouseup, mousedown, click
- Key
  - keydown, keyup, keypress
- Forms
  - focus, blur, change, keydown, keyup, keypress

JavaScript Gotcha's

onload Event

- Need the page JavaScript to execute as soon as possible
- onload waits for all images etc to load

Firefox

```
if (document.addEventListener)
  document.addEventListener('DOMContentLoaded', init, false);
```

Internet Explorer

```
<!--[if IE]><script defer src="ie_onload.js"></script><![endif]-->
```

The rest

```
window.onload = init;
```

Inline Events

- Most simple event attachment

```
<div onmouseover=“swapColor(event)” onmouseout=“swapColor(event)”></div>
```

- What about separating our control from our view?

DOM Event Decoration

- Attach event handlers to DOM nodes through JavaScript

```
var domNode = document.getElementById(“myNode”);
domNode.onmouseover = highlight;
```

Function pointer

- This can create memory leaks when using anonymous functions

Most common way of creating memory leaks in IE

```
var domNode = document.getElementById(“myNode”);
domNode.onmouseover = function() { domNode.style.color = ‘red’;};
```

DOM Event Decoration

W3C - Firefox

Capture

```
var domNode = document.getElementById(“myNode”);
domNode.addEventListener(“mouseover”, hightlight, false);
```

Function pointer

Internet Explorer

```
var domNode = document.getElementById(“myNode”);
domNode.attachEvent(“onmouseover”, highlight);
```

Prefixed with “on”

Event Object

Internet Explorer	W3C	document.documentElement.clientHeight
clientX / Y	clientX / Y, pageX / Y	clientX / Y returns the event coordinates without the document scroll position taken into account, whereas pageX / Y does take scrolling into account.
N/A	currentTarget	The HTML element to which the event handler was attached.
keyCode, altKey, ctrlKey, shiftKey	keyCode, altKey, ctrlKey, shiftKey	Various key event modifiers to check if ctrlKey, shiftKey ctrlKey, shiftKey the Shift or Ctrl key are pressed.
srcElement	target	The HTML element on which the event actually took place. Both properties are supported in Opera and Safari.
type	type	The event type without the “on” prefix.
fromElement / toElement	relatedTarget	from is used only for mouseover and mouseout events. Both properties are supported in Opera and Safari

Event Questions

- How do you access the Event object?
- What does “this” refer to in the event handler function?

Event Object

- Passed as argument to event handler in W3C model and as a global in IE

W3C

```
function swapColor(evt) {
}
```

Internet Explorer

```
function swapColor() {
  var evt = window.event;
}
```

Handler Execution Scope

- “this” is the element that handled the event (W3C) or the window (IE)

W3C

```
function swapColor(evt) {
  this.style.color = “#FF0000”;
}
```

Internet Explorer

```
function swapColor() {
  window.event.srcElement.style.color
}
```

Cross-Browser Event Façade

- Make Internet Explorer look like W3C

Event type “mouseover”, etc

```
eventManager = {}; // Singleton object

eventManager.attachEvent = function(elem, type, handler, capture) {
  // Browser checking for IE vs W3C compliant browser
  if (elem.attachEvent) {
    // Create two expando properties with function references
    elem[‘evt_' + type] = function() {
      handler.call(elem);
    };
    // Attach one of our expando function references to the event
    elem.attachEvent('on'+type, elem[‘evt_' + type]);
    // Set the capture if it was specified
    if (capture) elem.setCapture(true);
  } else if (elem.addEventListener) {
    elem.addEventListener(type, handler, capture);
  }
}
```

Detects IE

Sets scope of “this”

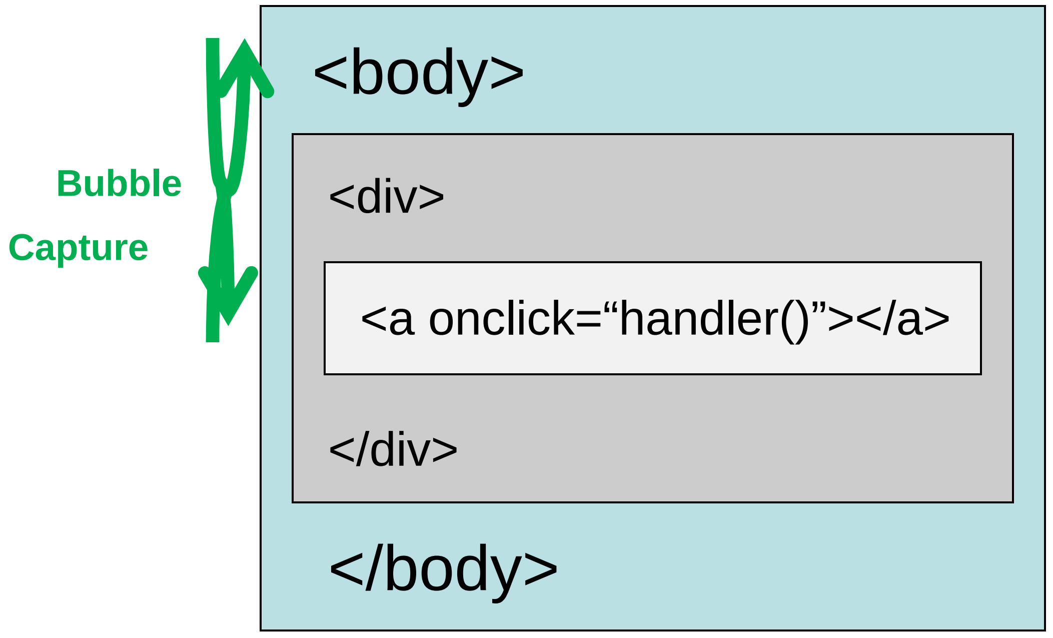
IE

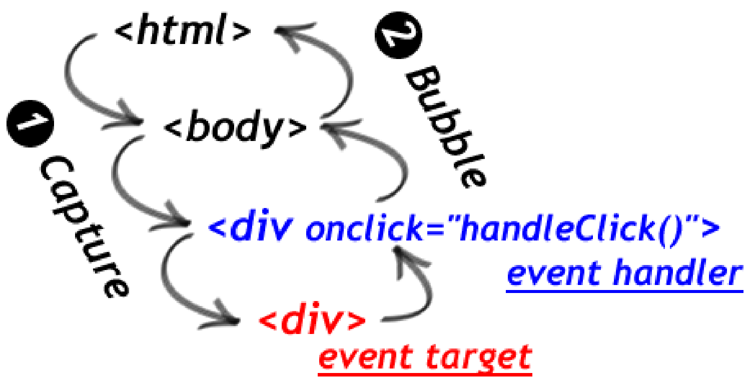
W3C

Event Flow

- Events have two phases
  - Events are first captured and propagate from the <body> to the target element
  - Event then bubbles back up from the target element to the <body>
- Capture is *very* different in IE and W3C but important nonetheless

Event Flow


Bubble
Capture
<body>
<div>
<a onclick="handler()"></a>
</div>
</body>


<html>
<body>
<div onclick="handleClick()">
event handler
<div>
event target
1 Capture
2 Bubble

Event Creation

- Programmatic event creation

```
if (window.attachEvent) // Internet Explorer
{
  element.fireEvent('on'+evtName);
}
else
{
  // create and init a new event
  var newEvent = document.createEvent(evtType);
  newEvent.initKeyEvent(evtName, true, true, document.defaultView,
ctrlKey, altKey, shiftKey, metaKey, keyCode, charCode);
  // dispatch new event
  element.dispatchEvent(newEvent);
}
```

Exercise 3 – OO JavaScript (if time)

- Create the following objects:
  - Person class
    - age
    - name
    - height
  - Customer class
    - account balance
    - inherits from Person
- Put at least 1 prototype method on each class.
- Create a few customers

Get optional application template at:
http://www.nitobi.com/gwt/ex3.zip

Exercise 3 – Possible Solution

```
Person = function(fname, lname, age) {
        this.firstName = fname;
        this.lastName = lname;
        this.age = age;
}
Person.prototype.getFullName = function() {
        return this.firstName + " " + this.lastName; }

Customer = function(fname, lname, age, balance) {
        this.balance = balance;
        Person.call(this,fname, lname, age);

}
Customer.prototype = new Person();
Customer.prototype.getBalance = function() {
        return '$' + this.balance;}
setupCustomers = function() {
        var cust1 = new Customer("John", "Smith", 24, 233.23);
        console.log(cust1.getFullName());
        console.log(cust1.getBalance());

}
```

Finito

Questions?