

# *JAVASCRIPT*

Instructors: Crista Lopes  
Copyright © Instructors.

# Outline



- Objects and Functions
- The mysterious Prototype
- Emulating inheritance

# JavaScript

---

- ❑ Classless objects
- ❑ Objects are dictionaries (hash maps)
- ❑ Functions are objects

# Instantiating objects

---

- Via literals
- Via functions

# An object via object literals

```
var apple = {  
  type: "macintosh",  
  color: "red",  
  getInfo: function () {  
    return this.color + ' ' + this.type + ' apple';  
  }  
}
```

and then

```
apple.color = "reddish";  
alert(apple.getInfo());
```

# An object via functions

```
function Apple (type) {  
    this.type = type;  
    this.color = "red";  
    this.getInfo = getAppleInfo;  
}
```

// anti-pattern!

```
function getAppleInfo() {  
    return this.color + ' ' + this.type + ' apple';  
}
```

and then

```
var apple = new Apple('macintosh');  
apple.color = "reddish";  
alert(apple.getInfo());
```

# An object via functions (better)

```
function Apple (type) {  
  this.type = type;  
  this.color = "red";  
  this.getInfo = function getAppleInfo() {  
    return this.color + ' ' + this.type + ' apple';  
  }  
}
```

and then

```
var apple = new Apple('macintosh');  
apple.color = "reddish";  
alert(apple.getInfo());
```

# Hybrid

```
var apple = new function() {  
  this.type = "macintosh";  
  this.color = "red";  
  this.getInfo = function () {  
    return this.color + ' ' + this.type + ' apple';  
  };  
}
```



# 'this'

```
function Apple (type) {  
    this.type = type;  
    this.color = "red";  
    this.getInfo = getAppleInfo;  
}
```

// anti-pattern!

```
function getAppleInfo() {  
    return this.color + ' ' + this.type + ' apple';  
}
```

????

unbound at this point

and then

```
var apple = new Apple('macintosh');  
apple.color = "reddish";  
alert(apple.getInfo());
```

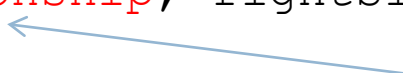
bound to apple

# Function contexts

Hidden magic of JS function calls:

- `Function.call(context: Object, arg1: Object, arg2: Object, ...)`
- `Function.apply(context: Object, args: Array)`

```
var linkEntities = function(leftSide, rightSide) {  
    console.log(leftSide, this.relationship, rightSide);  
};
```



unbound

```
var context = { relationship: 'likes' },  
    lSide = 'John Mason',  
    rSide = 'Falling Skies';
```

```
linkEntities.call(context, lSide, rSide);  
linkEntities.apply(context, [lSide, rSide]);
```

bound!

# Function contexts

- Given automatically to function calls
- Includes at least the object bound to the keyword 'this'

```
var apple = new Apple('macintosh');  
apple.color = "reddish";  
apple.getInfo();
```

→ 

```
var context = { this : apple };  
getInfo.call(context)
```

# Prototypes

- ❑ Prototypes are ‘hidden’ objects associated with every ‘visible’ object
- ❑ Prototypes hold additional information about the objects
- ❑ If a property is not found in an object, JS looks for it in the object’s prototype
- ❑ Mechanism for code reuse

# An object via functions (better)

```
function Apple (type) {  
  this.type = type;  
  this.color = "red";  
  this.getInfo = function getAppleInfo() {  
    return this.color + ' ' + this.type + ' apple';  
  }  
}
```

and then

```
var apple1 = new Apple('macintosh');  
var apple2 = new Apple('gala');
```

getAppleInfo is duplicated

# An object via functions (better)

```
function Apple (type) {  
    this.type = type;  
    this.color = "red";  
}
```

```
Apple.prototype.getInfo = function() {  
    return this.color + ' ' + this.type + ' apple';  
}
```

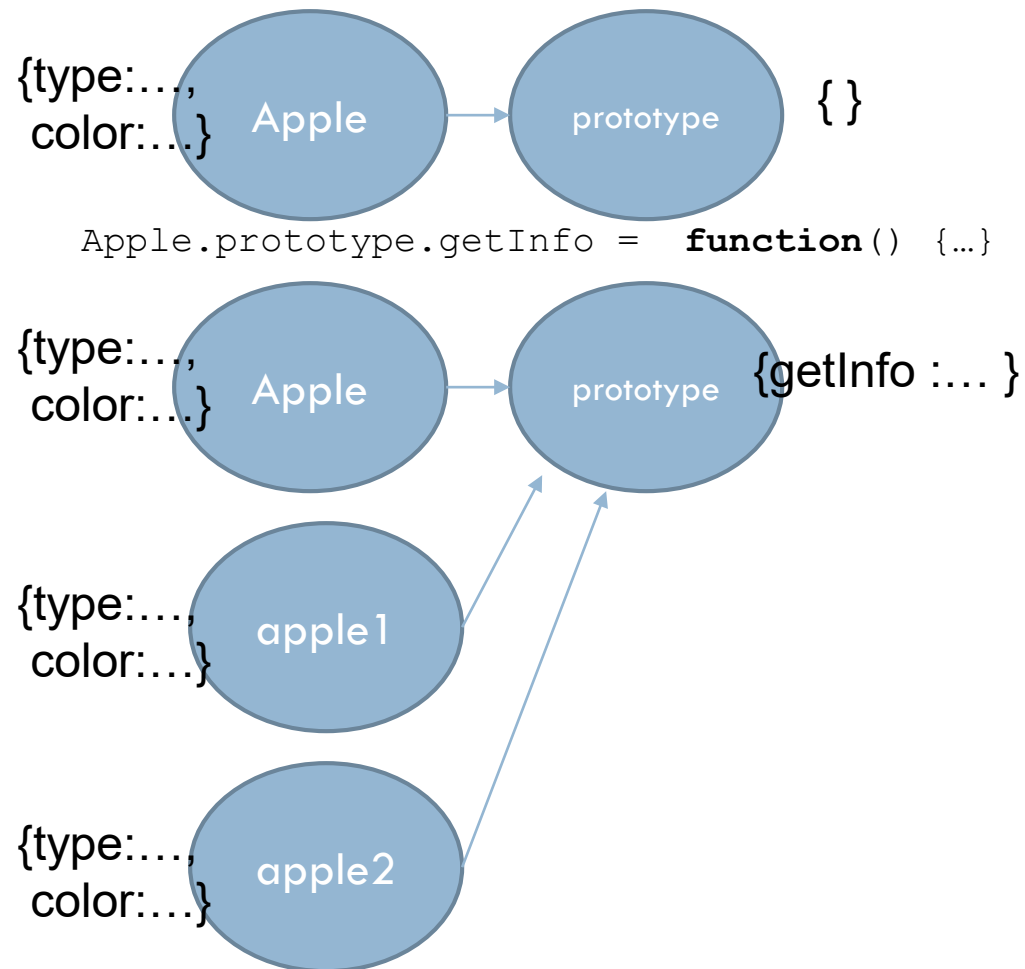
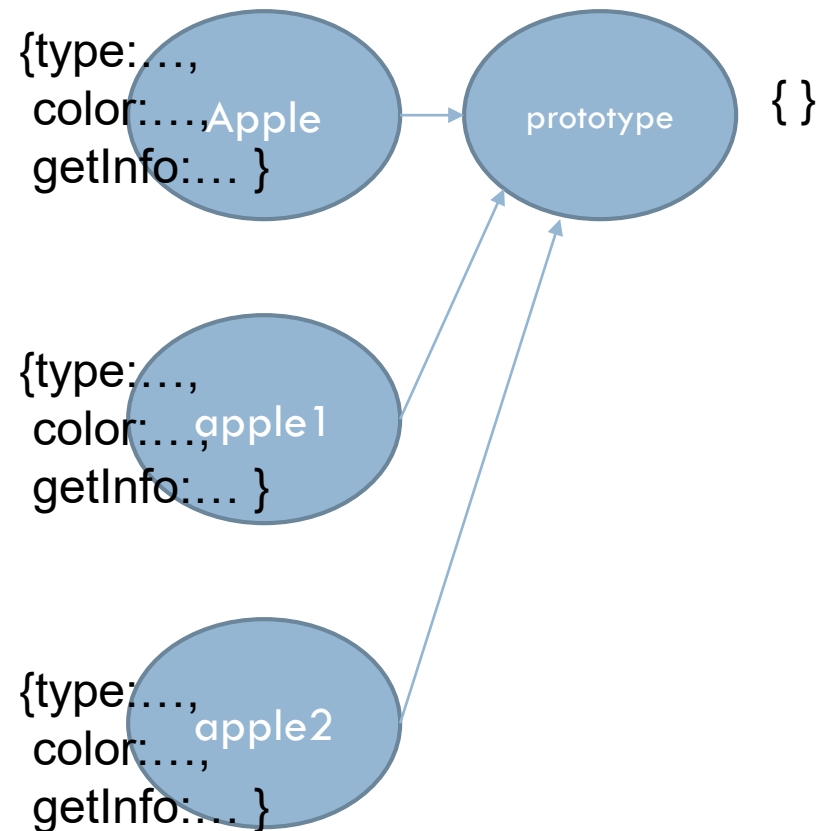
and then

```
var apple1 = new Apple('macintosh');  
var apple2 = new Apple('gala');
```

getAppleInfo: only one

# Pictorially...

## First version



# Inheritance with prototypes

```
function Mammal(name) {
  this.name=name;
  this.offspring=[];
}
Mammal.prototype.haveABaby=function() {
  var newBaby=new Mammal("Baby "+this.name);
  this.offspring.push(newBaby);
  return newBaby;
}
Mammal.prototype.toString=function() {
  return '[Mammal "'+this.name+'"]';
}
```

```
Cat.prototype = new Mammal();
Cat.prototype.constructor=Cat;
function Cat(name) {
  this.name=name;
}
Cat.prototype.toString=function() {
  return '[Cat "'+this.name+'"]';
}
```

```
var someAnimal = new Mammal('Mr. Biggles');
var myPet = new Cat('Felix');
alert('someAnimal is '+someAnimal);
alert('myPet is '+myPet);

myPet.haveABaby();
alert(myPet.offspring.length);
alert(myPet.offspring[0]);
```



# Linkage

- ❑ Objects can be created with a secret link to another object.
- ❑ If an attempt to access a name fails, the secret linked object will be used.
- ❑ The secret link is not used when storing. New members are only added to the primary object.

# Linkage

□ `var myNewObject = new Object(myOldObject);`



# Linkage

- ❑ `myNewObject.name = "Tom Piperson";`
- ❑ `myNewObject.level += 1;`
- ❑ `myNewObject.crime = 'pignapping';`

"name"	"Tom Piperson"
"level"	4
"crime"	"pignapping"



"name"	"Jack B. Nimble"
"goto"	"Jail"
"grade"	"A"
"level"	3

# JS Classless Objects

- 3 ways to define a class

<http://www.phpied.com/3-ways-to-define-a-javascript-class/>

- Function prototypes

<http://ejohn.org/apps/learn/#76>

# Riddle me this

- In JavaScript,
  - ▣ Classes are functions
    - [https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object-oriented\\_JS](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object-oriented_JS)
  - ▣ Functions are Objects
    - <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions>
  - ▣ They are all key-value maps anyway!



Extra

# Outline



- Java Script History
- What is Java Script
- Key Ideas of Java Script
- Java Script Design
- Examples
- Functions, Objects, Linkages, Inheritance
- Web Development
- Comparison with other Languages
- Demo and Practical Applications

# What is a Scripting Language?

- In computing, a **scripting language** is a set of commands for controlling some specific piece of hardware, software, or operating system
- Used to write programs that produce inputs to another language processor
- Scripting languages intended to allow end users to extend the functionality of specific software packages are a subset commonly called **extension languages**
- Simple program structure without complicated declarations



# 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
  - ▣ “JavaScript is to Java as carpet is to car”
  - ▣ Name was part of a marketing deal
- Various implementations available
  - ▣ SpiderMonkey C implementation (from Mozilla)
  - ▣ Rhino Java implementation (also from Mozilla)

# Why JavaScript?

- Easy to implement. Just put your code in the HTML document and tell the browser that it is JavaScript.
- Creates highly responsive interfaces that improve the user experience and provide dynamic functionality, without having to wait for the server to react and show another page.
- Enhances Web pages with dynamic and interactive features:  
Active web pages
- JavaScript can load content into the document if and when the user needs it, without reloading the entire page: Ajax
- JavaScript can help fix browser problems or patch holes in browser support: CSS Layout Issues

# What Is JavaScript?

- ❑ JavaScript, aka Mocha, aka LiveScript, aka JScript, aka ECMAScript is a loosely-typed, embedded in HTML/Browser scripting language.
- ❑ It is an *interpreted object-oriented programming language*.
- ❑ It is used to manipulate the contents of HTML documents.
- ❑ It is used for writing *client-side applications* that cannot access information on a host computer.
- ❑ JavaScript is the world's most misunderstood programming language

# Misconceptions In JavaScript

---

- ❑ The Name
- ❑ Mispositioning
- ❑ Design Errors
- ❑ Bad Implementations
- ❑ The Browser
- ❑ Bad Books
- ❑ Substandard Standard
- ❑ JavaScript is a Functional Language

# Key Ideas Of Java Script

- ❑ Load and go delivery
- ❑ Loose typing
- ❑ Objects as general containers
- ❑ Prototypal inheritance
- ❑ Lambda
- ❑ Linkage though global variables

# What You Can Do With JavaScript:

- Creates highly responsive interfaces that improve the user experience and provide dynamic functionality, replacing server-side scripting
- Can react to events - can be set to execute when something happens (i.e. page loaded or when a user clicks on an HTML element)
- Can read and write HTML elements - change the content of an HTML element - not just form elements!
- Can be used to validate data - although a user could easily turn it off in web browser
- Can be used to create/manipulate cookies - store information on the client through a very controlled mechanism

# What Else Can It Do?

- Defensive computing: Can test what is possible in your browser and react accordingly
- Control browser functions: Can detect the visitor's browser - detect if it is Mozilla/Firefox based, or riddled with holes, non-standards compliant, piece of junk Internet Explorer
- JavaScript enables shopping carts, form validation, calculations, special graphic and text effects, image swapping, image mapping, clocks, and more.
- Page embellishments and special effects, Navigation systems, Basic math calculations

# JavaScript Design

- Functions are based on Lisp/Scheme

- ▣ first-class inline higher-order functions

```
function (y) { return y+100; }
```

- Objects are based on Smalltalk/Self

- ▣ `var pt = {y : 100, move : function(dy){this.y -= dy}}`

- JavaScript is syntactically a C. It differs from C mainly in its type system, which allows functions to be values.



# Language Design

- ❑ Extreme permissiveness: Error-tolerant language which reports very less errors.
- ❑ Lack of modularity: Java script objects are represented in tree structure
- ❑ No information hiding capabilities: Does not distinguish public and private functions from data
- ❑ Syntactic Issues: 'this' keyword required to refer every instance variable.

# Software Configuration

- ❑ No 'include' or 'load' directive in Core JavaScript: no mechanisms for loading in additional JavaScript code while an application is already running
- ❑ No support for unloading parts of a program: unloading capabilities would be essential in a long-running system that relies on modules that can be upgraded without shutting down the system.
- ❑ Loading multiple JavaScript applications into the same virtual machine is problematic

# Development Style

- Evolutionary development approach is a necessity: due to , error-tolerant nature, JavaScript programming requires an incremental, evolutionary software development approach
- Code coverage testing is important: In an interactive command shell and the 'eval' function, each piece of code can be tested immediately after it has been written
- Completeness of applications is difficult to determine: flexible and dynamic languages make it easy to get applications up and running quickly
- Event-oriented programming model works surprisingly well: external occurrence such as a user interface activity or network event is defined as an event

# Example 1: Browser Events

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

Mouse event causes  
page-defined function to  
be called

Other events: onLoad, onMouseMove, onKeyPress, onUnload

# Example 2: Page Manipulation

- Some possibilities
  - ▣ `createElement(elementName)`
  - ▣ `createTextNode(text)`
  - ▣ `appendChild(newChild)`
  - ▣ `removeChild(node)`
- Example: add a new list item

```
var list = document.getElementById('list1')
var newitem = document.createElement('li')
var newtext = document.createTextNode(text)
list.appendChild(newitem)
newitem.appendChild(newtext)
```

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

# Example 3: Using Cookies

- Creating cookies:

```
document.cookie = "myContents=Quackit JavaScript  
cookie experiment; expires=Fri, 19 Oct 2007 12:00:00  
UTC; path=/";
```

- Reading cookies:

```
document.write(document.cookie);
```

- Deleting cookies:

```
document.cookie = "myContents=Quackit JavaScript  
cookie experiment; expires=Fri, 14 Oct 2005 12:00:00  
UTC; path=/";
```

# Language Syntax

- JavaScript is case sensitive
  - ▣ HTML is not case sensitive
- Statements terminated by returns or semi-colons (;)
  - ▣ `y = y+2;` is the same as `y = y+2`
  - ▣ Using semi-colons is a good idea to reduce errors
- “Blocks” of statements enclosed using `{ ... }`
- 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!

# JavaScript Primitive Datatypes

- Boolean :
  - ▣ *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 objects:
  - ▣ null and undefined



# 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
  - ▣ Only function calls and the *with* statement cause a change of scope

# Functions

- Functions are first-class objects
- Functions can be passed, returned, and stored just like any other value
- Functions inherit from `Object` and can store name/value pairs.
- The function operator takes an optional name, a parameter list, and a block of statements, and returns a function object.
- `function name ( parameters ) {statements }`
- A function can appear anywhere that an expression can appear.

# Functions (cont'd)

- 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} };`

# Examples Of Functions

## □ Curried functions

- `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)`

# JavaScript functions and *this*

```
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
```

Both o1.g and o2.g refer to the same function.  
Why are the results for o1.g() and o2.g() different ?

# More About *this*

- Property of the activation object for fctn call
  - ▣ In most cases, *this* points to the object which has the function as a property (or method).

▣ Example :

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

*this* is resolved dynamically when the method is executed

# Anonymous Functions

- Anonymous functions very useful for callbacks
  - ▣ `setTimeout(function() { alert("done"); }, 10000)`
  - ▣ Evaluation of `alert("done")` delayed until function 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) // Works because objs are passed by ref`

# Detour: Lambda Calculus

## □ Expressions

$$x + y \qquad x + 2 * y + z$$

## □ Functions

$$\lambda x. (x + y) \qquad \lambda z. (x + 2 * y + z)$$

## □ Application

$$(\lambda x. (x + y)) (3) \qquad = 3 + y$$

$$(\lambda z. (x + 2 * y + z))(5) \qquad = x + 2 * y + 5$$



# Higher-Order Functions

- Given function  $f$ , return function  $f \circ f$

$\lambda f. \lambda x. f (f x)$

- How does this work?

$(\lambda f. \lambda x. f (f x)) (\lambda y. y+1)$

$= \lambda x. (\lambda y. y+1) ((\lambda y. y+1) x)$

$= \lambda x. (\lambda y. y+1) (x+1)$

$= \lambda x. (x+1)+1$

- In pure lambda calculus, same result if step 2 is altered.

# Same Procedure, Lisp Syntax

- Given function  $f$ , return function  $f \circ f$

`(lambda (f) (lambda (x) (f (f x))))`

- How does this work?

`((lambda (f) (lambda (x) (f (f x)))) (lambda (y) (+ y 1)))`

`= (lambda (x) ((lambda (y) (+ y 1))`

`((lambda (y) (+ y 1)) x))))`

`= (lambda (x) ((lambda (y) (+ y 1)) (+ x 1))))`

`= (lambda (x) (+ (+ x 1) 1))`

# Same Procedure, JavaScript Syntax

- Given function  $f$ , return function  $f \circ f$

```
function (f) { return function (x) { return f(f(x)); }; }
```

- How does this work?

```
(function (f) { return function (x) { return f(f(x)); }; })
```

```
  (function (y) { return y + 1; })
```

```
function (x) { return (function (y) { return y + 1; })
```

```
  ((function (y) { return y + 1; }) (x)); }
```

- ```
function (x) { return (function (y) { return y + 1; }) (x + 1); }
```

- ```
function (x) { return ((x + 1) + 1); }
```

# 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*
    - *Not* a common feature of hash tables or associative arrays
- 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”;`

# Object Construction

- Make a new empty object
- All three of these expressions have exactly the same result:

`new Object()`

`{}`

`object(Object.prototype)`

- `{}` is the preferred form.

# 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","Mustang",2013);  
car.prototype.print = function () {  
    return this.year + " " + this.make + " " + this.model;}  
c.print();
```

# Inheritance

- ❑ Linkage provides simple inheritance.
- ❑ An object can inherit from an older object.
- ❑ Instead of Classical Inheritance, JavaScript has Prototypal Inheritance.
- ❑ Instead of organizing objects into rigid classes, new objects can be made that are similar to existing objects, and then customized.

# Vehicle-Car Inheritance Example

- Constructor function for Vehicle:

```
function Vehicle(hasEngine, hasWheels) {  
  this.hasEngine = hasEngine || false;  
  this.hasWheels = hasWheels || false; }
```

- Constructor function for Car:

```
function Car (make, model, hp) {  
  this.hp = hp; this.make = make;  
  this.model = model; }
```

- Set up dynamic inheritance:

```
Car.prototype = new Vehicle(true, true);
```



# Example (cont'd)

- Extending Car using prototype:

```
Car.prototype.displaySpecs = function () {  
  console.log(this.make + ", " + this.model + ", " +  
  this.hp + ", " + this.hasEngine + ", " + this.hasWheels);  
}
```

- Creating a new car:

```
var myAudi = new Car ("Audi", "A4", 150);
```

- Extend Vehicle using prototype:

```
Vehicle.prototype.hasTrunk = true;
```

# Local Variables Stored In “scope object”

Special treatment for nested functions

```
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 This Class

- 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

# Closures

- A closure takes place when a function creates an environment that binds local variables to it in such a way that they are kept alive after the function has returned.
- A closure is a special kind of object that combines two things: a function, and any local variables that were in-scope at the time that the closure was created.
- Used to bind variables to functions that are called at a later time

# Example 1

```
❑ function setupSomeGlobals() {  
    // Local variable that ends up within closure  
    var num = 666;  
    // Store some references to functions as global variables  
    gAlertNumber = function() { alert(num); }  
    gIncreaseNumber = function() { num++; }  
    gSetNumber = function(x) { num = x; }  
}
```

# 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") {
```

```
  } catch (e if e == "SecondException") {
```

```
  } catch (e){    }
```



# 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

# JavaScript eval

- Evaluate string as code (seen this before?)
  - ▣ The eval function evaluates a string of JavaScript code, in scope of the calling code
    - `var code = "var a = 1";`
    - `eval(code);` // a is now '1'
    - `var obj = new Object();`
    - `obj.eval(code);` // obj.a is now 1
  - ▣ Common use: efficiently deserialize a complicated data structure 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 (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 }`

# Web Development

## JavaScript(in HTML5) vs. Flash

- JavaScript supports any platform
- JavaScript has easier development cycle.  
Because It's impossible to debug a flash object on a remote server
- It's easier to find/ develop HTML/JavaScript expertise
- JavaScript(in HTML5) also get great presentation.

# Java vs. JavaScript(General)

## Compilation

- ❑ JavaScript is not compiled. It runs interpretively.
- ❑ Java is compiled into an intermediate bytecode (which is machine-independent). A machine-specific Java Virtual Machine then interprets this code to run on the specific platform.
- ❑ **Language Philosophy**
- ❑ JavaScript tries to continue running unless it encounters an insurmountable syntax error.
- ❑ Java is on the stricter end of software tools. The compiler disallows many situations that might be allowed in other languages and the runtime engine raises an exception when questionable or dangerous actions take place.
- ❑ **Learning Curve**
- ❑ JavaScript is a smaller and more straightforward to learn.
- ❑ Java is larger. While it is less complex than C++, it is still a substantial language.

# Java vs. JavaScript(Web Capabilities)

- **Web Presentation**

- JavaScript can build dynamic webpages. JavaScript provides more direct control over the browser (e.g. controlling the back button, refreshing the page, etc.)
- Java can be used to build Applet pages but this technology is dated and no longer widely used.

- **Developing User Interfaces**

- Interfaces are developed in HTML which is relatively straightforward to learn. JavaScript is included in the webpage and it interacts with the HTML form elements.
- In plain Java, user interfaces are developed in AWT or Swing. Working directly with these Java libraries is challenging and the user interfaces were often not all that easy to use. Java mostly lost this market.

- **Client-side Security**

- JavaScript does not allow direct access to a user's hard-drive (beyond Cookies which the browser directly controls). Over the year, security holes have been found in JavaScript.
- Java is very strict about not allowing access to memory or devices outside the applet. Certain actions are allowed as long as t

# Java vs. JavaScript (Middle Tier Capabilities)

- **Middle Tier Capabilities**
- In the early days of the Internet, server-side JavaScript could be used to produce webpages with dynamic content (e.g. including database content).
- Java has a comprehensive model for supporting middle tier content called Java 2 Enterprise Edition (J2EE). It is a considerably more comprehensive model than JavaScript.
- **Middle Tier Presence**
- Server-side JavaScript is virtually non-existent.
- Java (J2EE) is widely supported in the middle tier (IBM, Oracle, Apache/Tomcat, etc.) Many vendors produce tools to help create, debug and deploy middle-tier Java solutions. Java has a very strong presence in the middle tier. Its major middle-tier competitor is Microsoft (whose solution uses dot-Net suite in the middle tier).

# JSLint

- ❑ JSLint can help improve the robustness and portability of your programs.
- ❑ It enforces style rules.
- ❑ It can spot some errors that are very difficult to find in debugging.
- ❑ It can help eliminate implied globals.
- ❑ Currently available on the web and as a Konfabulator widget.
- ❑ Soon, in text editors and Eclipse.



# JavaScript Applications Demo

---

- HTML5 + Canvas + JavaScript
- Replace Flash with stunning visual effect
- Game Engine

# alphaTab

- alphaTab is a JavaScript library that allows one to embed notation/guitar tablatures into websites



# HTML5 Canvas Experiment

- This is a self-described “little experiment” which involves 100 tweets related to HTML5 that are loaded and subsequently displayed using a Javascript-based particle engine.



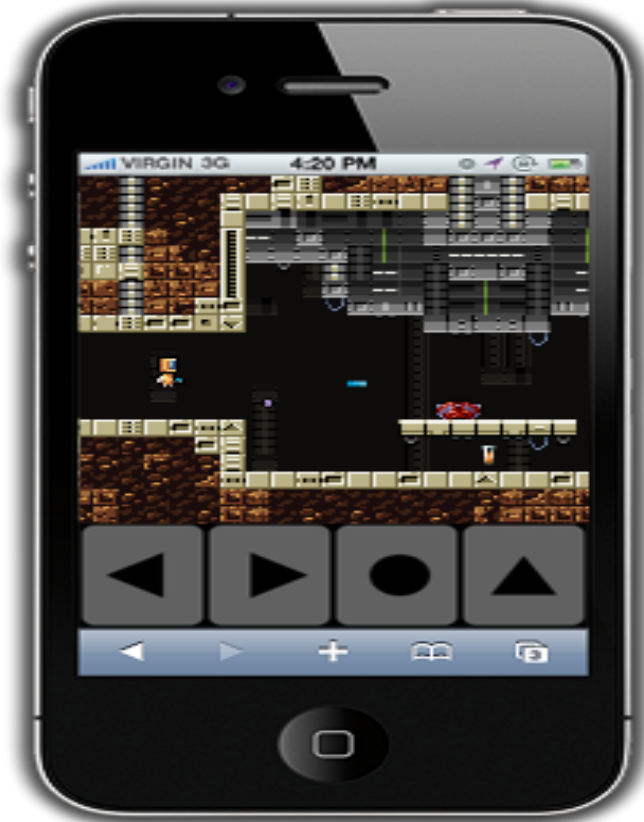
# Tank World

- 3D Shooting games implemented by HTML5 and JavaScript



# Impact Game Engine

- Impact is a JavaScript Game Engine that allows you to develop HTML5 Games in no time.



# 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/>
- 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
  - ▣ <http://www.crockford.com/JavaScript/>
  - ▣ JavaScript: The Good Parts, O'Reilly, 2008. (book)

Reference: <http://developer.mozilla.org/en/docs/>

Core\_JavaScript\_1.5\_Guide :Exception\_Handling\_Statements

# References (Continue)

- JavaScript VS. Flash
  - ▣ <http://flexblog.edchipman.ca/javascript-vs-flash-who-will-win-the-client-side-scripting-battle>
- JavaScript VS. JAVA
  - ▣ <http://www.sislands.com/coin70/week1/javajs.htm>
- JavaScript Canvas
  - ▣ <http://billmill.org/static/canvastutorial/index.html>
- JavaScript HTML5
  - ▣ <http://dev.w3.org/html5/spec/Overview.html>
- JavaScript Game Engine 1
  - ▣ <http://rocketpack.fi/engine/>
- JavaScript Game Engine 2
  - ▣ <http://ntt.cc/2011/01/31/66-open-source-javascript-game-engine-for-serious-developers.html>