jQuery and AJAX for Responsive Web Sites

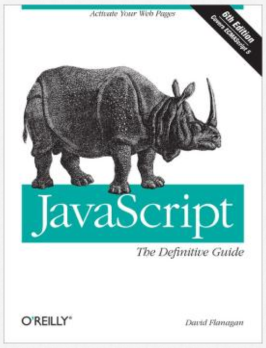
Lecture 3
JavaScript

Objectives

- Loose ends
- JavaScript overview
- Types and scope
- Objects, prototype-based inheritance
- Arrays
- First class functions
- Closures
- This

Loose Ends

- Office Hours
  - Uptown Espresso, 4th and Wall St.
  - Mon and Thurs, 6pm till 7:30pm
  - "niekjsanders" on Skype
- Assignment 1
  - Grading, question feedback, answer sheet
- Assignment 2
  - Posting tomorrow



developer.mozilla.org

Big Picture….

- Interpreted
- Weakly typed
- Garbage collected
- Prototype-based inheritance
- First class functions, closures

Interpreted vs. Compiled

… but with JIT.

Interpreted vs. Compiled

- Interpreted, usually
  - JavaScript
  - Java
  - C#
  - Python
  - PHP
- Compiled
  - C++
  - C
  - Fortran
- n/a
  - Machine code (x86 instructions)

Weak vs. Strong Typing

```
// C++ or Java
int cppFunc( int a, double b ) {
    return a / b;
}

// JavaScript
function cppFunc( a, b ) {
    return a / b;
}
```

Weak vs. Strong Typing

Image from Wikipedia

Waddles?
Swims?
Quacks?

"Duck Typing"

Weak vs. Strong Typing

Image from Wikipedia

```
// Javascript
function myFunc( obj ) {
    return obj.quack();
}
```

```
// C++ and polymorphism
std::string myFunc( DuckBase& duck ) {
    return duck.quack();
}
```

Weak vs. Strong Typing

The downside of being weak…

Garbage Collected

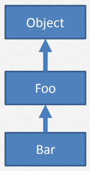
Image from Wikipedia

Garbage Collected

- Manual management
- GC pauses
- Cyclic references, closures

Image from Wikipedia

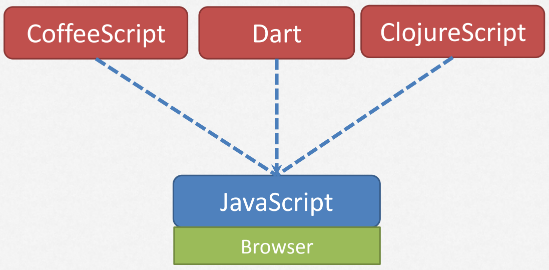
Prototype Inheritance



… not class based

First Class Functions, Closures

JavaScript in Perspective



JavaScript Types

- Number *
- String
- Boolean
- Null
- Undefined
- Object

Data Types - Numbers

- Everything is a double precision float
- Normal languages have separate type for integers
- Usually not an issue in practice

< Integer Division Demo >

Scope

```
foo = 7;                    // global scope, implicit
window.foo = 7;             // global scope, explicit
var foo = 7;                // global scope, from context

function square( num ) {
  var result = num * num;   // function scope
  return result;
}

square( 3 )
```

Not block scope!

```
function dangerDanger() {

    var x = 3;

    {
        var x = 5;
        console.log( x );     // prints 5
    }
    console.log( x );         // prints 5

}
```

var

“use strict”;

< Strict Demo >

Strings are immutable

```
var myString = "Hi ";
myString += "there";
```

Immutable Strings

$$\begin{tikzcd}
\text{var myString} \arrow[r] \arrow[dr] & \text{"Hi "} \\
& \text{"Hi there"}
\end{tikzcd}$$

```
var myString = "Hi ";
myString += "there";
```

Shallow object copies

Shallow Objects

Image from Wikipedia

var wheezy =

Wheezy the cat.

var wheezy =

var bella =

Dave says "Wheezy"
Morgan says "Bella"

Two names, one cat.

… remember this separation of name and value

var wheezy

var bella

wheezy.shave() ?

var wheezy

var bella

< Shallow Copy Demo >

```
var foo = [1,2,3];
var bar = foo;

bar.push( 4 );

// what is foo?
```

Objects Pass by Reference

```
var foo = [1,2,3];

function myFunc( bar ) {
    bar.push( 4 );
    bar = 100;
}

myFunc( foo );
```

Everything Else by Value

```
var foo = 7;

function myFunc( bar ) {
    bar = 100;
}

myFunc( foo );
```

Exploiting Pass by Reference

```
var $ = 5000;

(function( $ ) {

    $('p').hide();

})(jQuery)
```

< Object Demo >

Objects

Key/Value + Prototype

```
var cheese = new Object();

cheese.name = "Gouda";
cheese.age = 3;

cheese['yumminess'] = "extreme";
```

Object literal notation

```
// setting properties explicitly
var cheese = new Object();

cheese.name      = "Gouda";
cheese.age       = 3;
cheese.yumminess = "extreme";


// object literal notation
var cheese = { "name":      "Gouda",
               "age":       3,
               "yumminess": "extreme" };
```

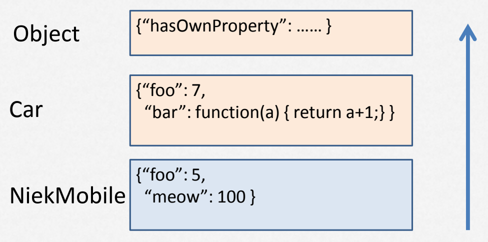
JSON!

… kinda

< Nested Objects Demo >

Prototypes

Prototype-based Inheritance



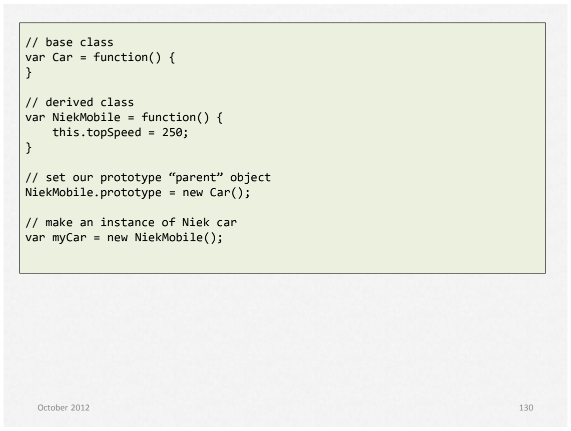
Prototypes

- We walk up prototype chain to resolve property lookups
- Either another object or null as our prototype
- Prototype changes seen by all child objects
- Not standard: __proto__
- Normally set via constructor functions

Constructors

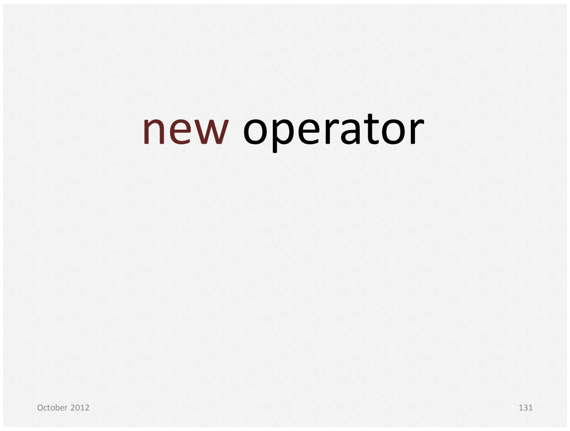
Constructors

- Just functions
- First letter of name capitalized by default
- Interact with the new operator



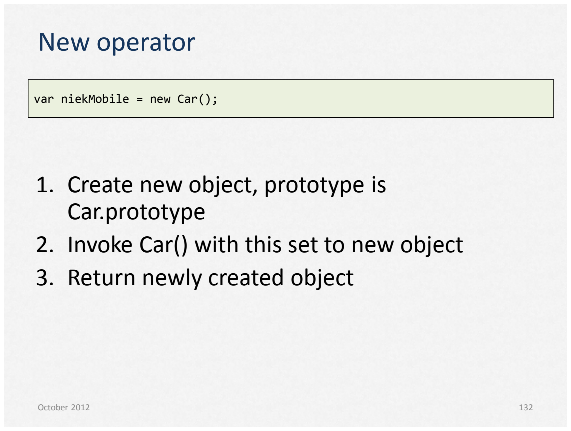
```
// base class
var Car = function() {
}

// derived class
var NiekMobile = function() {
    this.topSpeed = 250;
}

// set our prototype “parent” object
NiekMobile.prototype = new Car();

// make an instance of Niek car
var myCar = new NiekMobile();
```

October 2012 130



new operator

October 2012 131



New operator

```
var niekMobile = new Car();
```

1. Create new object, prototype is Car.prototype
2. Invoke Car() with this set to new object
3. Return newly created object

October 2012 132