Secrets of JavaScript Libraries

John Resig
http://ejohn.org/
http://twitter.com/jeresig

Raise your hand!

or Ask Questions on Twitter: @jeresig Huh?

About Me

- jQuery
- Processing.js
- Test Suite Integration in Firefox
- Standards work:
  W3C, WHATWG, ECMAScript
- Current Book:
  “Pro JavaScript Techniques”

Material

- “Secrets of the JavaScript Ninja”
- Manning Publishing, Winter 2008
- Assuming intermediate knowledge of JavaScript - go to the next level.

Tenants of Libraries

- Advanced use of the JavaScript language
- Apt construction of cross-browser code
- Tied together by best practices

Cross-Browser Code

- Strategies for handling cross-browser code
- Testing
- Additional Topics:
  - CSS Selector Engine
  - DOM Modification
  - Events

Good JavaScript Code

- Writing Good Code
- Topics:
  - Functions
  - Closures
  - Function Prototypes

Some Libraries...

- ...that I like. Opinions will differ!
- Prototype, jQuery, base2
- Good point for initial analysis
- Examine their techniques
- Not necessarily the best but a wide variety of techniques are employed

Some Libraries

- Prototype.js
  - Godfather of modern JavaScript libraries
  - Released in 2005 by Sam Stephenson
  - Features:
    - DOM
    - Events
    - Ajax
  - Techniques:
    - Object-Oriented
    - Aspect-Oriented
    - Functional

Some Libraries

- jQuery.js
  - Focuses on the relation between DOM and JavaScript
  - Written by John Resig, released Jan 2006
  - Features:
    - DOM
    - Events
    - Ajax
    - Animations
  - Techniques:
    - Functional

Some Libraries

- base2
  - Adds missing JavaScript/DOM features
  - Released 2007 by Dean Edwards
  - Features:
    - DOM
    - Events
  - Techniques:
    - Object-Oriented
    - Functional

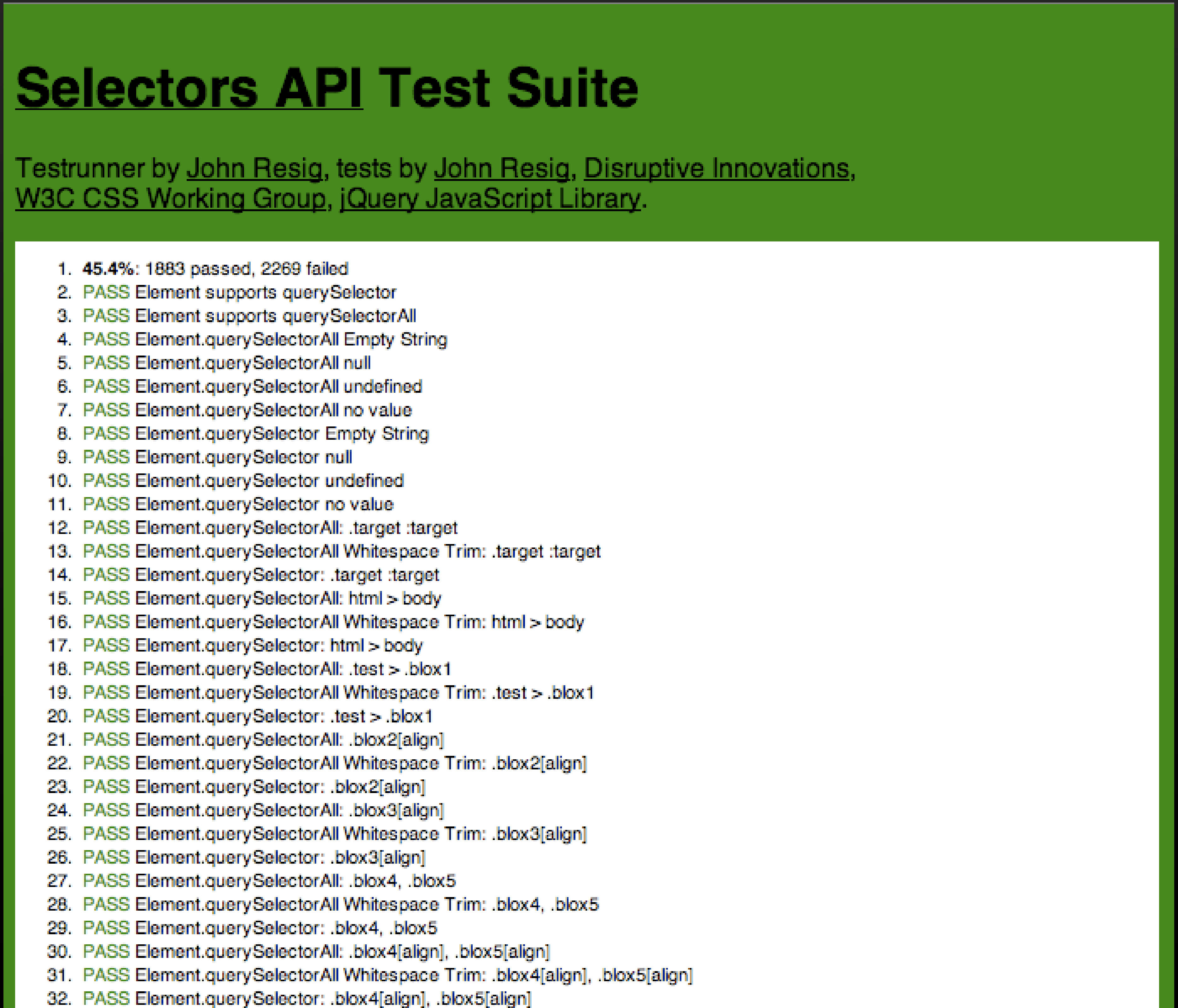
Testing

- JavaScript testing can be painfully simple
- There’s rarely a need for more than a couple useful methods and some basic output.

assert()

```
assert( true, "I always pass!" );
assert( false, "I always fail!" );
```

Simple Output



Selectors API Test Suite

Testrunner by John Resig, tests by John Resig, Disruptive Innovations, W3C CSS Working Group, jQuery JavaScript Library.

```
 1. 45.4%: 1883 passed, 2269 failed
 2. PASS Element supports querySelector
 3. PASS Element supports querySelectorAll
 4. PASS Element.querySelectorAll Empty String
 5. PASS Element.querySelectorAll null
 6. PASS Element.querySelectorAll undefined
 7. PASS Element.querySelectorAll no value
 8. PASS Element.querySelector Empty String
 9. PASS Element.querySelector null
10. PASS Element.querySelector undefined
11. PASS Element.querySelector no value
12. PASS Element.querySelectorAll: .target :target
13. PASS Element.querySelectorAll Whitespace Trim: .target :target
14. PASS Element.querySelector: .target :target
15. PASS Element.querySelectorAll: html > body
16. PASS Element.querySelectorAll Whitespace Trim: html > body
17. PASS Element.querySelector: html > body
18. PASS Element.querySelectorAll: .test > .blox1
19. PASS Element.querySelectorAll Whitespace Trim: .test > .blox1
20. PASS Element.querySelector: .test > .blox1
21. PASS Element.querySelectorAll: .blox2[align]
22. PASS Element.querySelectorAll Whitespace Trim: .blox2[align]
23. PASS Element.querySelector: .blox2[align]
24. PASS Element.querySelectorAll: .blox3[align]
25. PASS Element.querySelectorAll Whitespace Trim: .blox3[align]
26. PASS Element.querySelector: .blox3[align]
27. PASS Element.querySelectorAll: .blox4, .blox5
28. PASS Element.querySelectorAll Whitespace Trim: .blox4, .blox5
29. PASS Element.querySelector: .blox4, .blox5
30. PASS Element.querySelectorAll: .blox4[align], .blox5[align]
31. PASS Element.querySelectorAll Whitespace Trim: .blox4[align], .blox5[align]
32. PASS Element.querySelector: .blox4[align], .blox5[align]
```

assert()

```
(function(){
  var results, queue = [];

  this.assert = function(pass, msg){
    var type = pass ? "PASS" : "FAIL";
    var str = "<li class='" + type + "'><b>" +
              type + "</b> " + msg + "</li>";

    if ( queue )
      queue.push( str );
    else
      results.innerHTML += str;
  };

  window.addEventListener("load", function(){
    results = document.getElementById("results");
    results.innerHTML = queue.join('');
    queue = null;
  });
})();
```

Delayed Tests

```
 test(function(){
  pause();
  setTimeout(function(){
    assert( true, "First test completed" );
    resume();
  }, 400);
});

test(function(){
  pause();
  setTimeout(function(){
    assert( true, "Second test completed" );
    resume();
  }, 100);
});
```

Delayed Tests

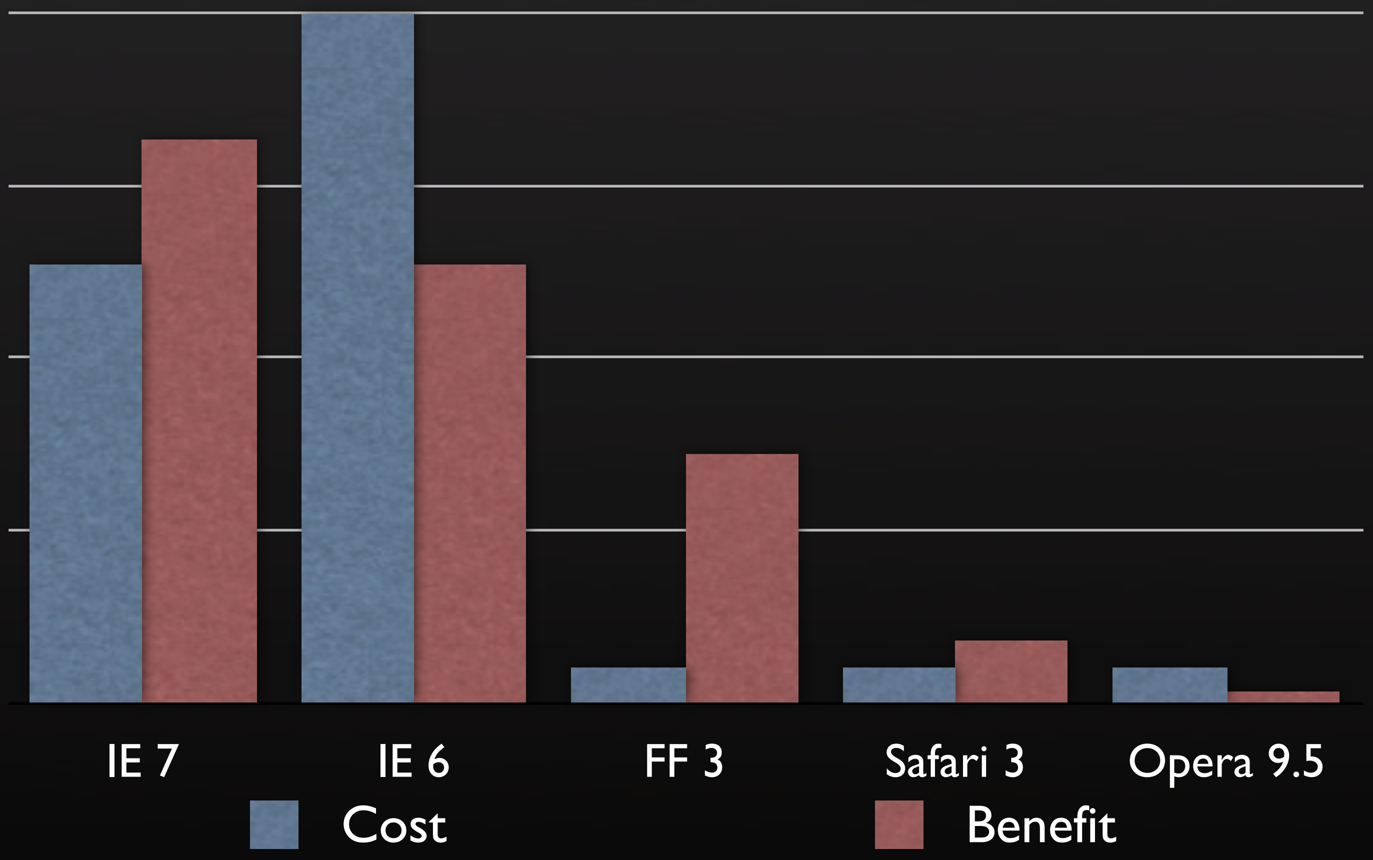
```
(function(){
  var queue = [], timer;
  this.test = function(fn){
    queue.push( fn );
    resume();
  };
  this.pause = function(){
    clearInterval( timer );
    timer = 0;
  };
  this.resume = function(){
    if ( !timer ) return;
    timer = setInterval(function(){
      if ( queue.length )
        queue.shift()();
      else
        pause();
    }, 1);
  };
})();
```

Cross-Browser Code

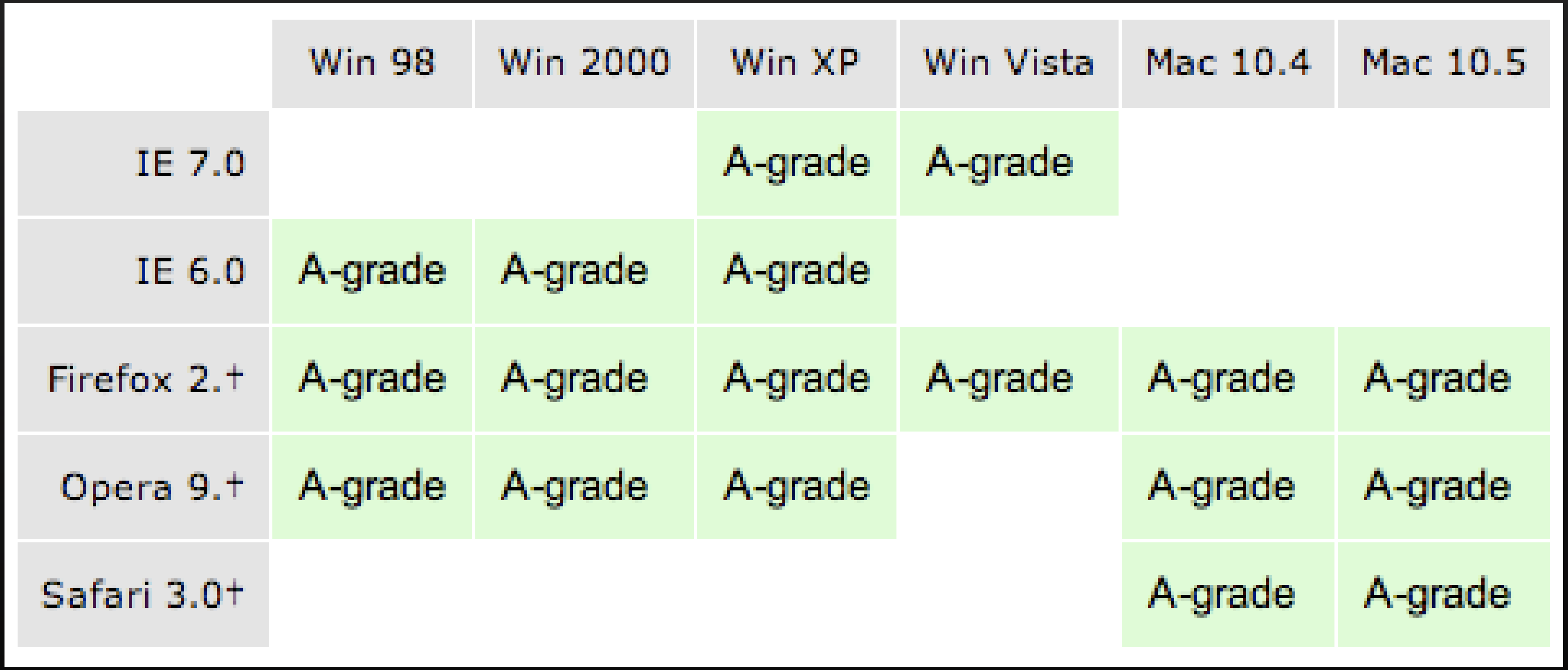
Strategies

- Pick your browsers
- Know your enemies
- Write your code

Cost / Benefit



Graded Support



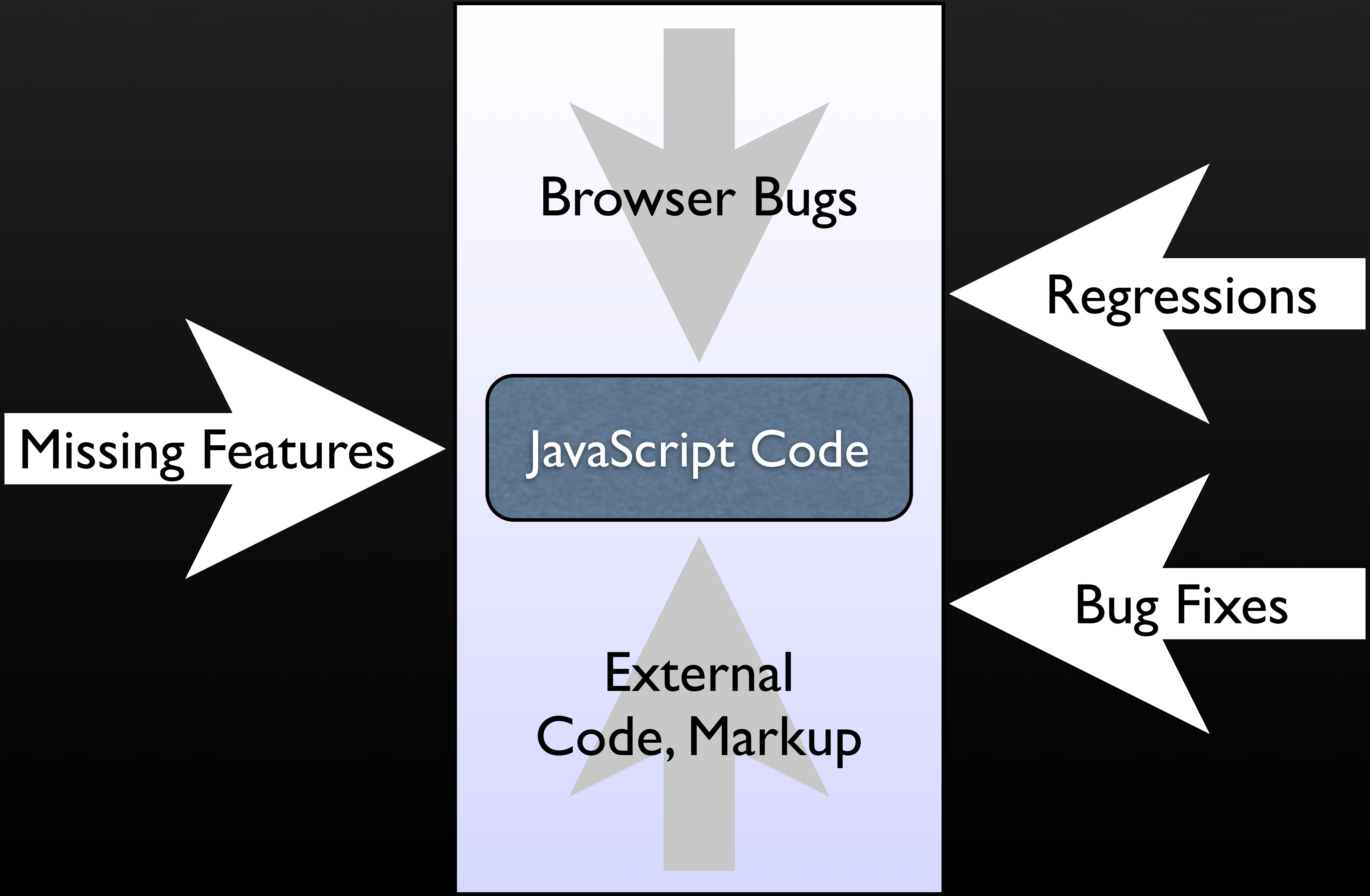
	Win 98	Win 2000	Win XP	Win Vista	Mac 10.4	Mac 10.5
IE 7.0			A-grade	A-grade		
IE 6.0	A-grade	A-grade	A-grade			
Firefox 2.†	A-grade	A-grade	A-grade	A-grade	A-grade	A-grade
Opera 9.†	A-grade	A-grade	A-grade		A-grade	A-grade
Safari 3.0†					A-grade	A-grade

Browser Support Grid

	IE	Firefox	Safari	Opera
Previous	6.0	2.0	2.0	9.2
Current	7.0	3.0	3.1	9.5
Next	8.0	3.1	4.0	10.0

Know Your Enemies

Points of Concern for JavaScript Code



Browser Bugs

- Generally your primary concern
- Your defense is a good test suite
  - Prevent library regressions
  - Analyze upcoming browser releases
- Your offense is feature simulation
- What is a bug?
  - Is unspecified, undocumented, behavior capable of being buggy?

External Code

- Making your code resistant to any environment
  - Found through trial and error
  - Integrate into your test suite
    - Other libraries
    - Strange code uses
- Make sure your code doesn’t break outside code
  - Use strict code namespacing
  - Don’t extend outside objects, elements

Object.prototype

```
Object.prototype.otherKey = "otherValue";

var obj = { key: "value" };
for ( var prop in object ) {
  if ( object.hasOwnProperty( prop ) ) {
    assert( prop, "key",
     "There should only be one iterated property." );
  }
}
```

Greedy IDs

```
<form id="form">
  <input type="text" id="length"/>
  <input type="submit" id="submit"/>
</form>
```

```
document.getElementsByTagName("input").length
```

Order of Stylesheets

- Putting stylesheets before code guarantees that they'll load before the code runs.
- Putting them after can create an indeterminate situation.

Missing Features

- Typically older browsers missing specific features
- Optimal solution is to gracefully degrade
  - Fall back to a simplified page
- Can’t make assumptions about browsers that you can’t support
  - If it’s impossible to test them, you must provide a graceful fallback
- Object detection works well here.

Object Detection

- Check to see if an object or property exists
- Useful for detecting an APIs existence
- Doesn’t test the compatibility of an API
  - Bugs can still exist - need to test those separately

Event Binding

```
function attachEvent( elem, type, handle ) {
  // bind event using proper DOM means
  if ( elem.addEventListener )
    elem.addEventListener(type, handle, false);

  // use the Internet Explorer API
  else if ( elem.attachEvent )
    elem.attachEvent("on" + type, handle);
}
```

Fallback Detection

```
if ( typeof document !== "undefined" &&
     (document.addEventListener
      || document.attachEvent) &&
     document.getElementsByTagName &&
     document.getElementById ) {
  // We have enough of an API to
  // work with to build our application
} else {
  // Provide Fallback
}
```

Fallback

- Figure out a way to reduce the experience
- Opt to not execute any JavaScript
  - Guarantee no partial API
    - (e.g. DOM traversal, but no Events)
- Redirect to another page

Bug Fixes

- Don’t make assumptions about browser bugs.
  - Assuming that a browser will always have a bug is foolhardy
  - You will become susceptible to fixes
  - Browsers will become less inclined to fix bugs
- Look to standards to make decisions about what are bugs

Failed Bug Fix

```
// Shouldn't work
var node = documentA.createElement("div");
documentB.documentElement.appendChild( node );

// Proper way
var node = documentA.createElement("div");
documentB.adoptNode( node );
documentB.documentElement.appendChild( node );
```

Feature Simulation

- More advanced than object detection
- Make sure an API works as advertised
- Able to capture bug fixes gracefully

Verify API

```
// Run once, at the beginning of the program
var ELEMENTS_ONLY = (function(){
  var div = document.createElement("div");
  div.appendChild( document.createComment("test" ) );
  return div.getElementsByTagName("*").length === 0;
})();

// Later on:
var all = document.getElementsByTagName("*");

if ( ELEMENTS_ONLY ) {
  for ( var i = 0; i < all.length; i++ ) {
    action( all[i] );
  }
} else {
  for ( var i = 0; i < all.length; i++ ) {
    if ( all[i].nodeType === 1 ) {
      action( all[i] );
    }
  }
}
```

Figure Out Naming

```
<div id="test" style="color:red;"></div>
<div id="test2"></div>
<script>
// Perform the initial attribute check
var STYLE_NAME = (function(){
  var div = document.createElement("div");
  div.style.color = "red";

  if ( div.getAttribute("style") )
    return "style";

  if ( div.getAttribute("cssText") )
    return "cssText";
})();

// Later on:
window.onload = function(){
  document.getElementsById("test2").setAttribute( STYLE_NAME,
      document.getElementById("test").getAttribute( STYLE_NAME ) );
};
</script>
```

Regressions

- Removing or changing unspecified APIs
- Object detection helps here
- Monitor upcoming browser releases
  - All vendors provide access to beta releases
  - Diligence!
- Example: IE 7 introduced XMLHttpRequest with file:// bug
- Test Suite Integration

Object Failover

```
function attachEvent( elem, type, handle ) {
  // bind event using proper DOM means
  if ( elem.addEventListener )
    elem.addEventListener(type, handle, false);

  // use the Internet Explorer API
  else if ( elem.attachEvent )
    elem.attachEvent("on" + type, handle);
}
```

Safe Cross-Browser Fixes

- The easiest form of fix
- Unifies an API across browsers
- Implementation is painless

Unify Dimensions

```
// ignore negative width and height values
if ( (key == 'width' || key == 'height') &&
     parseFloat(value) < 0 )
  value = undefined;
```

Prevent Breakage

```
if ( name == "type" && elem.nodeName.toLowerCase()
        == "input" && elem.parentNode )
  throw "type attribute can't be changed";
```

Untestable Problems

- Has an event handler been bound?
- Will an event fire?
- Do CSS properties like color or opacity actually affect the display?
- Problems that cause a browser crash.
- Problems that cause an incongruous API.

Impractial to Test

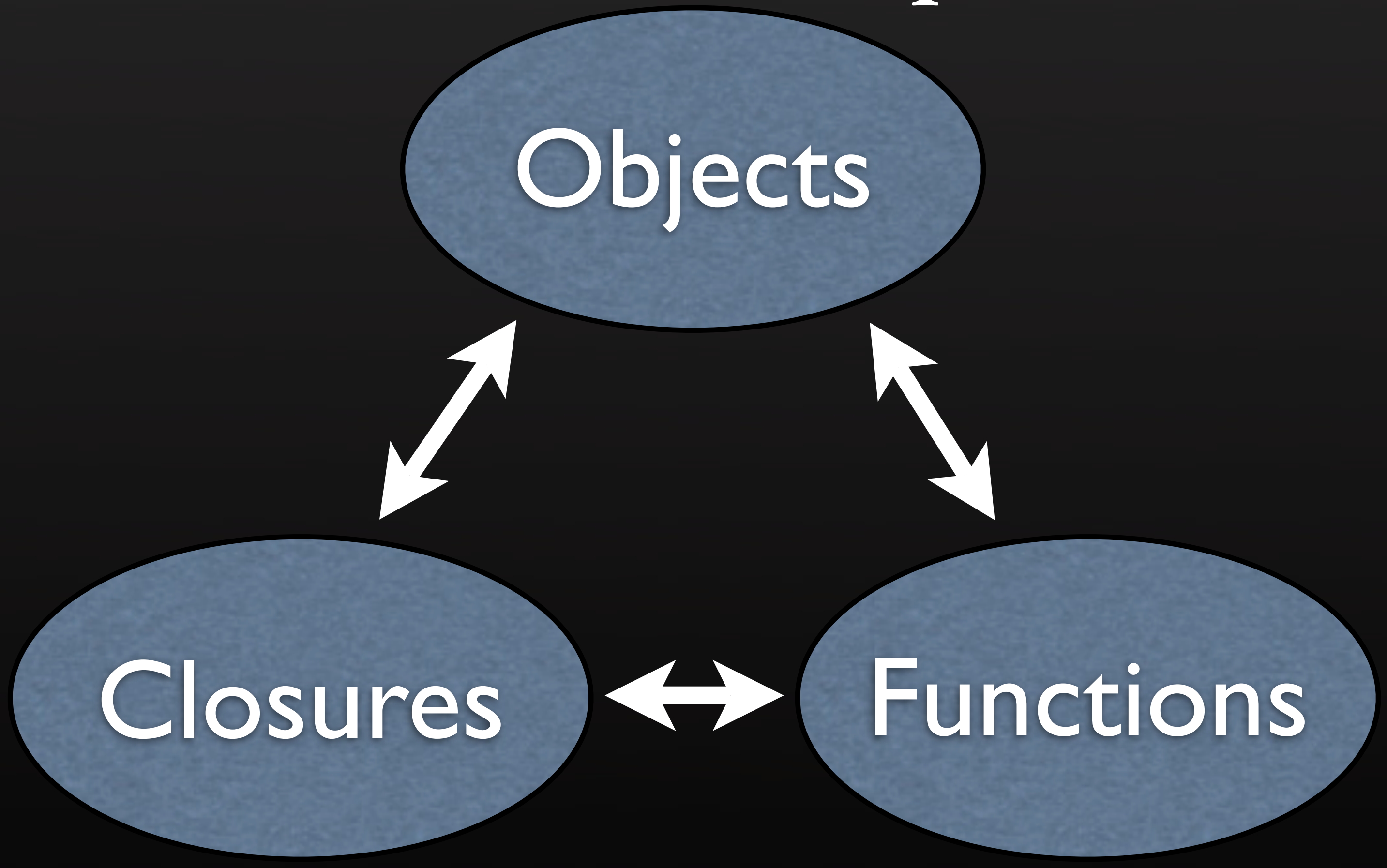
- Performance-related issues
- Determining if Ajax requests will work

Battle of Assumptions

- Cross-browser development is all about reducing the number of assumptions
- No assumptions indicates perfect code
- Unfortunately that's an unobtainable goal
- Prohibitively expensive to write
- Have to draw a line at some point

Good JavaScript Code

- Three-fold relationship between:



- Understanding these three and their interplay makes you a better programmer.

Functions

Function Definition

Function Definition

```
function isNimble(){ return true; }
var canFly = function(){ return true; };
window.isDeadly = function(){ return true; };

assert( isNimble() && canFly() && isDeadly(),
 "All are functions, all return true" );
```

Function Definition

```
var canFly = function(){ return true; };
window.isDeadly = function(){ return true; };

assert( isNimble() && canFly() && isDeadly(),
 "Still works, even though isNimble is moved." );

function isNimble(){ return true; }
```

Function Definition

```
function stealthCheck(){
  var ret = stealth() == stealth();
  return assert( ret,
   "We'll never get below this line, but that's OK!" );
  function stealth(){ return true; }
}

stealthCheck();
```

Function Definition

```
assert( typeof canFly == "undefined",
 "canFly doesn't get that benefit." );
assert( typeof isDeadly == "undefined",
  "Nor does isDeadly." );

var canFly = function(){ return true; };
window.isDeadly = function(){ return true; };
```

Anonymous Functions and Recursion

Recursion

```
function yell(n){
  return n > 0 ? yell(n-1) + "a" : "hiy";
}

assert( yell(4) == "hiyaaaa",
 "Calling the function by itself comes naturally." );
```

Recursion w/ Objects

```
var ninja = {
  yell: function(n){
    return n > 0 ? ninja.yell(n-1) + "a" : "hiy";
  }
};

assert( ninja.yell(4) == "hiyaaaa",
 "A single object isn't too bad, either." );
```

Recursion w/ Objects

```
var ninja = {
  yell: function(n){
    return n > 0 ? ninja.yell(n-1) + "a" : "hiy";
  }
};

assert( ninja.yell(4) == "hiyaaaa",
 "A single object isn't too bad, either." );

var samurai = { yell: ninja.yell };
var ninja = {};

try {
  samurai.yell(4);
} catch(e){
  assert( true,
   "Uh, this isn't good! Where'd ninja.yell go?" );
}
```

Named Anonymous

```
var ninja = {
  yell: function yell(n){
    return n > 0 ? yell(n-1) + "a" : "hiy";
  }
};

assert( ninja.yell(4) == "hiyaaaa",
 "Works as we would expect it to!" );

var samurai = { yell: ninja.yell };
var ninja = {};

assert( samurai.yell(4) == "hiyaaaa",
 "The method correctly calls itself." );
```

Named Anonymous

```
var ninja = function myNinja(){
  assert( ninja == myNinja,
   "This function is named two things - at once!" );
};

ninja();

assert( typeof myNinja == "undefined",
 "But myNinja isn't defined outside." );
```

arguments.callee

```
var ninja = {
  yell: function(n){
    return n > 0 ?
      arguments.callee(n-1) + "a" :
      "hiy";
  }
};

assert( ninja.yell(4) == "hiyaaaa",
 "arguments.callee is the function itself." );
```

Functions as Objects

Function Assignment

```
var obj = {};
var fn = function(){};

assert( obj && fn,
 "Both the object and function exist." );
```

Attaching Properties

```
var obj = {};
var fn = function(){};
obj.prop = "some value";
fn.prop = "some value";

assert( obj.prop == fn.prop,
 "Both are objects, both have the property." );
```

Storing Functions

```
var store = {
  id: 1,
  cache: {},
  add: function( fn ) {
    if ( !fn.id ) {
      fn.id = store.id++;
      return !!(store.cache[fn.uuid] = fn);
    }
  };
};

function ninja(){}

assert( store.add( ninja ),
  "Function was safely added." );
assert( !store.add( ninja ),
  "But it was only added once." );
```

Self-Memoization

```
function isPrime( num ) {
  if ( isPrime.answers[ num ] != null )
    return isPrime.answers[ num ];

  // Everything but 1 can be prime
  var prime = num != 1;
  for ( var i = 2; i < num; i++ ) {
    if ( num % i == 0 ) {
      prime = false;
      break;
    }
  }
  return isPrime.answers[ num ] = prime;
}
isPrime.answers = {};

assert( isPrime(5),
 "Make sure the function works, 5 is prime." );
assert( isPrime.answers[5],
 "Make sure the answer is cached." );
```

Self-Caching

```
function getElements( name ) {
  return getElements.cache[ name ] =
    getElements.cache[ name ] ||
      document.getElementsByTagName( name );
}
getElements.cache = {};

// Before Caching:
// 12.58ms
// After Caching:
// 1.73ms
```

Context

Context

```
var katana = {
  isSharp: true,
  use: function(){
    this.isSharp = !!this.isSharp;
  }
};

katana.use();

assert( !katana.isSharp,
 "Verify the value of isSharp has been changed." );
```

Context

```
function katana(){
  this.isSharp = true;
}
katana();

assert( isSharp === true,
 "A global object now exists with that name." );

var shuriken = {
  toss: function(){
    this.isSharp = true;
  }
};

shuriken.toss();

assert( shuriken.isSharp === true,
 "The value is set within the object." );
```

.call()

```
var object = {};

function fn(){
  return this;
}

assert( fn() == this,
 "The context is the global object." );

assert( fn.call(object) == object,
 "The context is changed to a specific object." );
```

.call() and .apply()

```
function add(a, b){
  return a + b;
}

assert( add.call(this, 1, 2) == 3,
 ".call() takes individual arguments" );

assert( add.apply(this, [1, 2]) == 3,
 ".apply() takes an array of arguments" );
```

Looping

```
function loop(array, fn){
  for ( var i = 0; i < array.length; i++ )
    if ( fn.call( array, array[i], i ) === false )
      break;
}

var num = 0;
loop([0, 1, 2], function(value, i){
  assert(value == num++,
   "Make sure the contents are as we expect it.");
});
```

Array Simulation

```
<input id="first"><input id="second">
<script>
var elems = {
  find: function(id){
    this.add( document.getElementById(id) );
  },
  length: 0,
  add: function(elem){
    Array.prototype.push.call( this, elem );
  }
};

elems.find("first");
assert( elems.length == 1 && elems[0].nodeType,
 "Verify that we have an element in our stash" );

elems.find("second");
assert( elems.length == 2 && elems[1].nodeType,
 "Verify the other insertion" );
</script>
```

Variable Arguments

Max/Min in Array

```
function smallest(array){
  return Math.min.apply( Math, array );
}
function largest(array){
  return Math.max.apply( Math, array );
}

assert(smallest([0, 1, 2, 3]) == 0,
 "Locate the smallest value.");
assert(largest([0, 1, 2, 3]) == 3,
 "Locate the largest value.");
```

Function Overloading

```
function merge(root){
  for ( var i = 1; i < arguments.length; i++ )
    for ( var key in arguments[i] )
      root[key] = arguments[i][key];
  return root;
}

var merged = merge({name: "John"}, {city: "Boston"});

assert( merged.name == "John",
 "The original name is intact." );
assert( merged.city == "Boston",
 "And the city has been copied over." );
```

Slicing Arguments

```
function multiMax(multi){
  return multi * Math.max.apply( Math,
    Array.prototype.slice.call( arguments, 1 ));
}

assert( multiMax(3, 1, 2, 3) == 9,
 "3*3=9 (First arg, by largest.)" );
```

Function Length

```
function makeNinja(name){}
function makeSamurai(name, rank){}

assert( makeNinja.length == 1,
 "Only expecting a single argument" );

assert( makeSamurai.length == 2,
 "Multiple arguments expected" );
```

addMethod

```
function Ninjas(){
  var ninjas = [ "Dean Edwards", "Sam Stephenson", "Alex Russell" ];
  addMethod(this, "find", function(){
    return ninjas;
  });
  addMethod(this, "find", function(name){
    var ret = [];
    for ( var i = 0; i < ninjas; i++ )
      if ( ninjas[i].indexOf(name) == 0 )
        ret.push( ninjas[i] );
    return ret;
  });
  addMethod(this, "find", function(first, last){
    var ret = [];
    for ( var i = 0; i < ninjas; i++ )
      if ( ninjas[i] == (first + " " + last) )
        ret.push( ninjas[i] );
    return ret;
  });
}
```

addMethod

```
var ninjas = new Ninjas();

assert( ninjas.find().length == 3,
 "Finds all ninjas" );

assert( ninjas.find("Sam").length == 1,
 "Finds ninjas by first name" );

assert( ninjas.find("Dean", "Edwards") == 1,
 "Finds ninjas by first and last name" );

assert( ninjas.find("Alex", "X", "Russell") == null,
 "Does nothing" );
```

addMethod

```
function addMethod(object, name, fn){
  var old = object[ name ];
  object[ name ] = function(){
    if ( fn.length == arguments.length )
      return fn.apply( this, arguments )
    else if ( typeof old == 'function' )
      return old.apply( this, arguments );
  };
}
```

Function Type

Function Type

```
function ninja(){}

assert( typeof ninja == "function",
 "Functions have a type of function" );
```

Function Type

- Browsers give mixed results, unfortunately
  - Firefox 2 and 3:
    - typeof <object/> == "function"
  - Firefox 2:
    - typeof /regexp/ == "function"
      /regexp/("regexp")
  - IE 6 and 7:
    - typeof elem.getAttribute == "object"
  - Safari 3
    - typeof document.body.childNodes
      == "function"

isFunction()

```
function isFunction( fn ) {
  return !!fn && !fn.nodeName &&
    fn.constructor != String &&
    fn.constructor != RegExp &&
    fn.constructor != Array &&
    /function/i.test( fn + "" );
}
```

Closures

How Closures Work

Closures

```
var stuff = true;

function a(arg1){
  var b = true;
  assert( a && stuff,
   "These come from the closure." );

  function c(arg2){
    assert( a && stuff && b && c && arg1,
     "All from a closure, as well." );
  }
  c(true);
}

a(true);

assert( stuff && a,
 "Globally-accessible variables and functions." );
```

Private Variables

```
function Ninja(){
  var slices = 0;

  this.getSlices = function(){
    return slices;
  };
  this.slice = function(){
    slices++;
  };
}

var ninja = new Ninja();
ninja.slice();

assert( ninja.getSlices() == 1,
 "We're able to access the internal slice data." );
assert( ninja.slices === undefined,
 "And the private data is inaccessible to us." );
```

Callbacks

```
<div></div>
<script src="jquery.js"></script>
<script>
var elem = jQuery("div").html("Loading...");
jQuery.ajax({
  url: "test.html",
  success: function(html){
    assert( elem,
     "The element to append to, via a closure." );
    elem.html( html );
  }
});
</script>
```

Timers

```
<div id="box" style="position:absolute;">Box!</div>
<script>
var elem = document.getElementById("box");
var count = 0;

var timer = setInterval(function(){
  if ( count <= 100 ) {
    elem.style.left = count + "px";
    count++;
  } else {
    assert( count == 100,
     "Count came via a closure, accessed each step" );
    assert( timer,
     "The timer reference is also via a closure." );
    clearInterval( timer );
  }
}, 10);
</script>
```

Enforcing Context

Enforcing Context

```
<button id="test">Click Me!</button>
<script>
var Button = {
  click: function(){
    this.clicked = true;
  }
};

var elem = document.getElementById("test");
elem.addEventListener("click", Button.click, false);

trigger( elem, "click" );

assert( elem.clicked,
 "The clicked property was set on the element" );
</script>
```

.bind()

```
function bind(context, name){
  return function(){
    return context[name].apply(context, arguments);
  };
}
```

Enforcing Context

```
<button id="test">Click Me!</button>
<script>
var Button = {
  click: function(){
    this.clicked = true;
  }
};

var elem = document.getElementById("test");
elem.addEventListener("click",
 bind(Button, "click"), false);

trigger( elem, "click" );

assert( Button.clicked,
 "The clicked property was set on our object" );
</script>
```

Prototype’s .bind()

```
Function.prototype.bind = function(){
  var fn = this, args =
   Array.prototype.slice.call(arguments),
   object = args.shift();

  return function(){
    return fn.apply(object,
      args.concat(
       Array.prototype.slice.call(arguments)));
  };
};

var myObject = {};
function myFunction(){
  return this == myObject;
}

assert( !myFunction(), "Context is not set yet" );
assert( myFunction.bind(myObject)(), "Context is set properly" );
```

Partially Applying Functions

Currying

```
Function.prototype.curry = function() {
  var fn = this,
      args = Array.prototype.slice.call(arguments);
  return function() {
    return fn.apply(this, args.concat(
      Array.prototype.slice.call(arguments)));
  };
};

String.prototype.csv =
 String.prototype.split.curry(/,\s*/);

var results = ("John, Resig, Boston").csv();
assert( results[1] == "Resig",
 "The text values were split properly" );
```

Partial Application

```
Function.prototype.partial = function(){
  var fn = this,
      args = Array.prototype.slice.call(arguments);

  return function(){
    var arg = 0;
    for ( var i = 0; i < args.length &&
                 arg < arguments.length; i++ )
      if ( args[i] === undefined )
        args[i] = arguments[arg++];
    return fn.apply(this, args);
  };
};

String.prototype.csv =
 String.prototype.split.partial(/,\s*/);

var results = ("John, Resig, Boston").csv();
assert( results[1] == "Resig",
 "The text values were split properly" );
```

Partial Application

```
var delay = setTimeout.partial(undefined, 10);

delay(function(){
  assert( true, "Verify the delay." );
});
```

Partial Application

```
var bindClick = document.body.addEventListener
  .partial("click", undefined, false);

bindClick(function(){
  assert( true, "Click event bound." );
});
```

Overriding Function Behavior

Memoization

```
Function.prototype.memoized = function(key){
  this._values = this._values || {};
  return this._values[key] !== undefined ?
    this._values[key] :
    this._values[key] = this.apply(this, arguments);
};

function isPrime( num ) {
  var prime = num != 1;
  for ( var i = 2; i < num; i++ ) {
    if ( num % i == 0 ) {
      prime = false;
      break;
    }
  }
  return prime;
}

assert( isPrime.memoized(5),
 "Make sure the function works, 5 is prime." );
assert( isPrime._values[5], "Make sure the answer is cached." );
```

Memoization

```
Function.prototype.memoize = function(){
  var fn = this;
  return function(){
    return fn.memoized.apply( fn, arguments );
  };
};

var isPrime = (function( num ) {
  var prime = num != 1;
  for ( var i = 2; i < num; i++ ) {
    if ( num % i == 0 ) {
      prime = false;
      break;
    }
  }
  return prime;
}).memoize();

assert( isPrime(5),
 "Make sure the function works, 5 is prime." );
assert( isPrime._values[5], "Make sure the answer is cached." );
```

Function Wrapping

```
function wrap(object, method, wrapper){
  var fn = object[method];
  return object[method] = function(){
    return wrapper.apply(this, [ fn.bind(this) ]
      .concat(Array.prototype.slice.call(arguments)));
  };
}

// Example adapted from Prototype
if (Prototype.Browser.Opera) {
  wrap(Element.Methods, "readAttribute",
    function(original, elem, attr) {
      return attr == "title" ?
        elem.title :
        original(elem, attr);
    });
}
```

(function(){})()

Temporary Scope

```
(function(){
  var numClicks = 0;

  document.addEventListener("click", function(){
    alert( ++numClicks );
  }, false);
})();
```

Return Value

```
document.addEventListener("click", (function(){
  var numClicks = 0;

  return function(){
    alert( ++numClicks );
  };
})(), false);
```

Variable Shortcut

```
(function(v) {
  Object.extend(v, {
    href:         v._getAttr,
    src:          v._getAttr,
    type:         v._getAttr,
    action:       v._getAttrNode,
    disabled:     v._flag,
    checked:      v._flag,
    readonly:     v._flag,
    multiple:     v._flag,
    onload:       v._getEv,
    onunload:     v._getEv,
    onclick:      v._getEv,
    ...
  });
})(Element._attributeTranslations.read.values);
```

Loops

```
<div></div>
<div></div>
<script>
var div = document.getElementsByTagName("div");
for ( var i = 0; i < div.length; i++ ) {
  div[i].addEventListener("click", function(){
    alert( "div #" + i + " was clicked." );
  }, false);
}
</script>
```

Loops

```
<div></div>
<div></div>
<script>
var div = document.getElementsByTagName("div");
for ( var i = 0; i < div.length; i++ ) (function(i){
  div[i].addEventListener("click", function(){
    alert( "div #" + i + " was clicked." );
  }, false);
})(i);
</script>
```

Library Wrapping

```
(function(){
  var jQuery = window.jQuery = function(){
    // Initialize
  };

  // ...
})();
```

Library Wrapping

```
var jQuery = (function(){
  function jQuery(){
    // Initialize
  }

  // ...

  return jQuery;
})();
```

Function Prototypes

Instantiation and Prototypes

Instatiation

```
function Ninja(){}

Ninja.prototype.swingSword = function(){
  return true;
};

var ninja1 = Ninja();
assert( !ninja1,
 "Is undefined, not an instance of Ninja." );

var ninja2 = new Ninja();
assert( ninja2.swingSword(),
 "Method exists and is callable." );
```

Instantiation

```
function Ninja(){
  this.swung = false;

  // Should return true
  this.swingSword = function(){
    return !!this.swung;
  };
}

// Should return false, but will be overridden
Ninja.prototype.swingSword = function(){
  return this.swung;
};

var ninja = new Ninja();
assert( ninja.swingSword(),
 "Calling the instance method." );
```

Live Updates

```
function Ninja(){
  this.swung = true;
}

var ninja = new Ninja();

Ninja.prototype.swingSword = function(){
  return this.swung;
};

assert( ninja.swingSword(),
 "Method exists, even out of order." );
```

Object Type

Object Type

```
function Ninja(){}

var ninja = new Ninja();

assert( typeof ninja == "object",
 "The type of the instance is still an object." );
assert( ninja instanceof Ninja,
 "The object was instantiated properly." );
assert( ninja.constructor == Ninja,
 "ninja object was created by the Ninja function." );
```

Constructor

```
var ninja = new Ninja();
var ninja2 = new ninja.constructor();

assert( ninja2 instanceof Ninja,
 "Still a ninja object." );
```

Inheritance and Prototype Chain

Inheritance

```
function Person(){}
Person.prototype.dance = function(){};

function Ninja(){}

// Achieve similar, but non-inheritable, results
Ninja.prototype = Person.prototype;
Ninja.prototype = { dance: Person.prototype.dance };

// Only this maintains the prototype chain
Ninja.prototype = new Person();

var ninja = new Ninja();
assert( ninja instanceof Ninja,
 "ninja inherits from the Ninja prototype" );
assert( ninja instanceof Person,
 "... and the Person prototype" );
assert( ninja instanceof Object,
 "... and the Object prototype" );
```

$$\begin{array}{rll} & \left.\begin{array}{l}\text{Object.prototype}\\ \text{Person.prototype}\\ \text{Ninja.prototype}\end{array}\right\} & \text{Update Live} \\ + & \text{Instance Properties} \leftarrow & \begin{array}{l}\text{Always supersedes}\\ \text{prototyped properties}\end{array} \\ \hline & \text{Object Properties} & \end{array}$$

(Person.prototype and Ninja.prototype are each preceded by +)

Native Prototype

```
if (!Array.prototype.forEach) {
  Array.prototype.forEach = function(fn){
    for ( var i = 0; i < this.length; i++ ) {
      fn( this[i], i, this );
    }
  };
}

["a", "b", "c"].forEach(function(value, index, array){
  assert( value, "Item found. );
});
```

HTML Prototypes

```
<div id="a">I'm going to be removed.</div>
<div id="b">Me too!</div>
<script>
// IE8 and above, or any other browser
HTMLElement.prototype.remove = function(){
  if ( this.parentNode )
    this.parentNode.removeChild( this );
};

// Old way
var a = document.getElementById("a");
a.parentNode.removeChild( a );

// New way
document.getElementById("b").remove();
</script>
```

Gotchas

Object.prototype

```
Object.prototype.keys = function(){
  var keys = [];
  for ( var i in this )
    keys.push( i );
  return keys;
};

var obj = { a: 1, b: 2, c: 3 };

assert( obj.keys().length == 4,
 "3 existing properties plus the new keys method." );
```

hasOwnProperty

```
Object.prototype.keys = function(){
  var keys = [];
  for ( var i in this )
    if ( this.hasOwnProperty( i ) )
      keys.push( i );
  return keys;
};

var obj = { a: 1, b: 2, c: 3 };

assert( obj.keys().length == 3,
 "Only the 3 existing properties are included." );
```

Numbers

```
Number.prototype.add = function(num){
  return this + num;
};

var n = 5;
assert( n.add(3) == 8,
 "It works fine if the number is in a variable." );

assert( (5).add(3) == 8,
 "Also works if a number is in parentheses." );

// Won't work, causes a syntax error
// assert( 5.add(3) == 8,
//   "Doesn't work, causes error." );
```

Sub-Classing Native Arrays

```
function MyArray(){}
MyArray.prototype = new Array();

var mine = new MyArray();
mine.push(1, 2, 3);

assert( mine.length == 3,
 "All the items are on our sub-classed array." );
assert( mine instanceof Array,
 "Verify that we implement Array functionality." );
```

Simulating Arrays

```
function MyArray(){}
MyArray.prototype.length = 0;

(function(){
  var methods = ['push', 'pop', 'shift',
   'unshift', 'slice', 'splice', 'join'];

  for ( var i = 0; i < methods.length; i++ ) (function(name){
    MyArray.prototype[ name ] = function(){
      return Array.prototype[ name ].apply( this, arguments );
    };
  })(methods[i]);
})();

var mine = new MyArray();
mine.push(1, 2, 3);

assert( mine.length == 3,
 "All the items are on our sub-classed array." );
assert( !(mine instanceof Array),
 "We aren't sub-classing Array, though." );
```

Instantiation

```
function User(first, last){
  this.name = first + " " + last;
}

var user = new User("John", "Resig");

assert( typeof user == "undefined",
 "Since ‘new’ wasn't used, no instance is defined." );
```

Instantiation

```
function User(first, last){
  this.name = first + " " + last;
}

var name = "Resig";
var user = User("John", name);

assert( name == "John Resig",
 "The name variable is accidentally overridden." );
```

Instantiation

```
function User(first, last){
  if ( !(this instanceof arguments.callee) )
    return new User(first, last);

  this.name = first + " " + last;
}

var name = "Resig";
var user = User("John", name);

assert( user,
 "This was defined correctly, by mistake." );
assert( name == "Resig",
 "The right name was maintained." );
```

Parts of a Library

CSS Selectors

- querySelectorAll
- CSS to XPath
- DOM
  - Work-down and merge
  - Work-up and remove

querySelectorAll

- The Selectors API spec from the W3C
- Two methods:
  - querySelector (first element)
  - querySelectorAll (all elements)
- Works on:
  - document
  - elements
  - DocumentFragments
- Implemented in:
  - Firefox 3.1, Safari 3, Opera 10, IE 8

CSS to XPath

- Browsers provide XPath functionality
- Collect elements from a document
- Works in all browsers
  - In IE it only works on HTML documents
- Fast for a number of selectors (.class, “div div div”)
  - Slow for some others: #id

Traditional DOM

- getElementsByTagName
- getElementById
- getElementsByClassName
  - in FF3, Safari 3
- .children
  - only returns elements (all but FF)
- getElementsByName
- .all[id]
  - Match multiple elements by ID

Top-Down

- Traditional style of traversal
  - Used by all major libraries
- Work from left-to-right
- “div p”
  - Find all divs, find paragraphs inside
- Requires a lot of result merging
- And removal of duplicates

Bottom-Up

- Work from right-to-left
- “div p”
  - Find all paragraphs, see if they have a div ancestor, etc.
- Fast for specific queries
  - “div #foo”
- Deep queries get slow
  - “div div div div”
  - parentNode traversal is slow

Events

- Memory Leaks
- Central Structure
- Unique Element ID

Leaks

- Internet Explorer 6 leaks horribly
- Attaching functions (that have a closure to another node) as properties
- Makes a leak:

```
elem.test = function(){
  anotherElem.className = “foo”;
};
```

Central Structure

- Store all bound event handlers in a central object
- Link elements to handlers
- Keep good separation
- Easy to manipulate later
  - Trigger individual handlers
  - Easy to remove again, later

Unique Element ID

- The structure must hook to an element
- Elements don’t have unique IDs
- Must generate them and manage them
- jQuery.data(elem, “events”);
- Unique attribute name:
  - elem.jQuery123456789 = 45;
  - Prevents collisions

DOM Modification

- Clean up bound events
- DocumentFragment
- Inline Script Execution

.remove()

- Have to clean up bound events
  - (again, memory leaks)

DocumentFragment

- Fragments can collect nodes
- Can be appended or cloned in bulk
- Super-fast (2-3x faster than normal)

Inline Script Execution

- .append("<script>var foo = 5;</script>");
- Must execute scripts globally
- window.execScript() (for IE)
- eval.call(window, "var foo = 5;");
- Cross-browser way is to build a script element then inject it
  - Executes globally

Materials

```
// Please keep private!
// Covered today:
http://jsninja.com/Introduction
http://jsninja.com/Functions
http://jsninja.com/Closures
http://jsninja.com/Function_Prototype
http://jsninja.com/Strategies_for_Cross-Browser_Code

// Feedback appreciated:
jeresig@gmail.com
http://twitter.com/jeresig
```