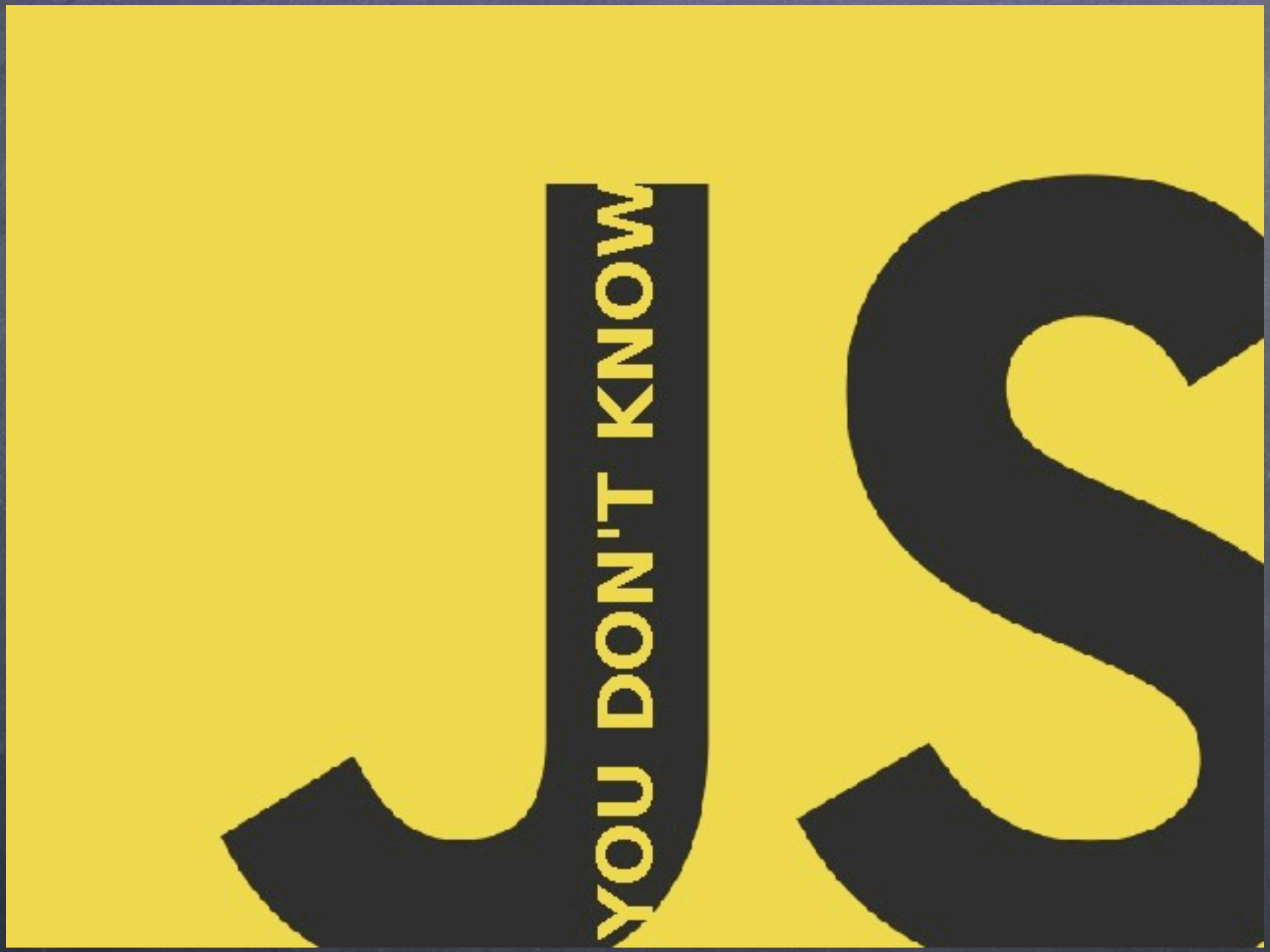
Kyle Simpson
@getify
http://getify.me

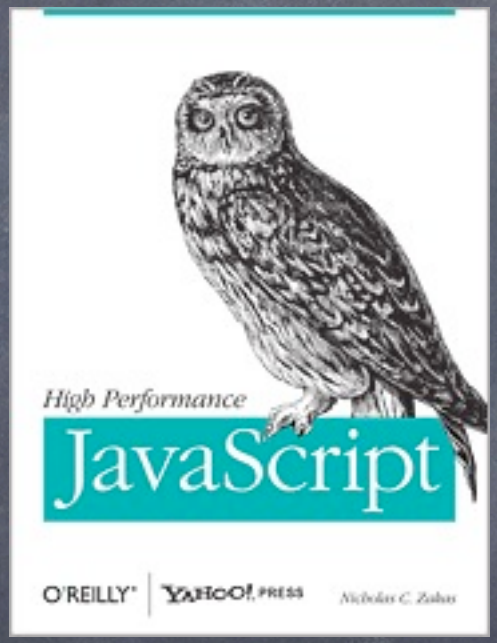
- LABjs
- grips
- asynquence

Kyle Simpson
@getify
http://getify.me

http://speakerdeck.com/getify



http://YouDontKnowJS.com


High Performance
JavaScript
O'REILLY
YAHOO! PRESS
Nicholas C. Zakas

Build Better Applications with Coding and Design Patterns
JavaScript
Patterns
O'REILLY
YAHOO! PRESS
Stoyan Stefanov

https://developer.mozilla.org/en-US/docs/JavaScript

https://github.com/rwldrn/idiomatic.js

http://www.ecma-international.org/ecma-262/5.1/

http://wiki.ecmascript.org/doku.php?id=harmony:proposals

Agenda

JavaScript Syntax

- Primitive Types
- Natives
- Coercion
- Reference vs. Value
- Surprises

- undefined
- string
- number
- boolean
- object
- function
- null

Primitive Types

```
1 typeof foo;              // "undefined"
2 typeof "foo";            // "string"
3 typeof 123;              // "number"
4 typeof true;             // "boolean"
5 typeof { a: 1 };         // "object"
6 typeof function() { alert(e); };    // "function"
```

Primitive Types (literals)

```
2
3 typeof null;     // "object" ... WAT!?
4
```

Primitive Types (literals)

Quiz

```
1 var foo;
2 typeof foo;                    // ???
3
4 var bar = typeof bar;
5 bar;                           // ???
6 typeof bar;                    // ???
7
8 typeof typeof 2;               // ???
```

Quiz

```
1 var foo;
2 typeof foo;                    // "undefined"
3
4 var bar = typeof bar;
5 bar;                           // "undefined"
6 typeof bar;                    // "string"
7
8 typeof typeof 2;               // "string"
```

- NaN ("not a number")
- Infinity, -Infinity
- null
- undefined (void)
- +0, -0

Primitive Types: special values

NaN

```
1 var a = "a" / 2;
2
3 a;                    // NaN
4 typeof a;             // "number"
5 isNaN(a);             // true
6
7 isNaN("foo");         // true! WAT!?
```

Primitive Types: special values

NaN

```
1 if (!Number.isNaN) {
2     Number.isNaN = function(num) {
3         return (
4             typeof num === "number" &&
5             window.isNaN(num)
6         );
7     }
8 }
```

Primitive Types: special values

NaN

```
1 if (!Number.isNan) {
2     Number.isNaN = function(num) {
3         return num !== num;
4     };
5 }
```

Negative Zero

```
1 var foo = 0 / -3;
2 foo === -0;              // true ... yay?
3 foo === 0;               // true ... doh!
4 0 === -0;                // true ... grr!
5 (0/-3) === (0/3);        // true ... sigh
6
7 foo;                     // 0 ... WAT!?
```

Negative Zero

```
1 function isNeg0(x) {
2   return x===0 && (1/x)===-Infinity;
3 }
4
5 isNeg0(0 / -3);      // true ... yay!
6 isNeg0(0 / 3);       // false ... yay!
```

ES6: Object.is(..)

```
1 Object.is( "foo", NaN );      // false
2 Object.is( NaN, NaN );        // true
3
4 Object.is( 0, -0 );           // false
5 Object.is( -0, -0 );          // true
```

Primitive Types: special values

Quiz

```
var baz = 2;
typeof baz;                    // ???
var baz;
typeof baz;                    // ???
baz = null;
typeof baz;                    // ???

baz = "baz" * 3;
baz;                           // ???
typeof baz;                    // ???

baz = 1 / 0;
baz;                           // ???
typeof baz;                    // ???
```

Primitive Types: special values

Quiz

```
var baz = 2;
typeof baz;                    // "number"
var baz;
typeof baz;                    // "number"
baz = null;
typeof baz;                    // "object"

baz = "baz" * 3;
baz;                           // NaN
typeof baz;                    // "number"

baz = 1 / 0;
baz;                           // Infinity
typeof baz;                    // "number"
```

Primitive Types: special values

- String
- Number
- Boolean
- Function
- Object
- Array
- RegExp
- Date
- Error

Native Types?
Native Objects?

- String()
- Number()
- Boolean()
- Function()
- Object()
- Array()
- RegExp()
- Date()
- Error()

Native Functions?

Native Constructors

- new String()
- new Number()
- new Boolean()
- new Function()
- new Object()
- new Array()
- new RegExp()
- new Date()
- new Error()

Quiz

```
var foo = new String("foo");
foo;                                  // ???
typeof foo;                           // ???
foo instanceof String;                // ???
foo instanceof string;                // ???

foo = String("foo");
typeof foo;                           // ???

foo = new Number(37);
typeof foo;                           // ???
```

Quiz

```
1  var foo = new String("foo");
2  foo;                                  // ???
3  typeof foo;                           // "object"
4  foo instanceof String;                // true
5  foo instanceof string;                // false; ref error!
6
7  foo = String("foo");
8  typeof foo;                           // "string"
9
10 foo = new Number(37);
11 typeof foo;                           // "object"
```

```
var foo;

foo = new Array(1,2,3); // don't!

foo = [1,2,3];          // do!

foo = new Object();     // don't!
foo.a = 1;
foo.b = 2;
foo.c = 3;

foo = { a:1, b:2, c:3 };// do!
```

Natives: literals

```
1 var foo;
2
3 foo = new RegExp("a*b","g");
4
5 foo = /a*b/g;
6
7 foo = new Date();
8
9 // no date literal!
```

Natives: literals

Explicit

Implicit

Coercion

Explicit: it's obvious from the code that you're doing it.

Explicit Coercion

string $\longleftrightarrow$ number

Explicit Coercion

```
var foo = "123";
var baz = parseInt(foo,10);
baz;                          // 123

baz = Number(foo);
baz;                          // 123

baz = +foo;                   // explicit?
baz;                          // 123

baz = 456;
foo = baz.toString();
foo;                          // "456"

foo = String(baz);
foo;                          // "456"
```

Explicit Coercion

$$* \longrightarrow \text{boolean}$$

Explicit Coercion

Falsy	Truthy
“”	“foo”
0, +0, -0	23
null	{ a:1 }
NaN	[1,3]
false	true
undefined	function(){..}
	...

Explicit Coercion

```
1  var foo = "123";
2  var baz = Boolean(foo);
3  baz;                                  // true
4
5  baz = !!foo;
6  baz;                                  // true
7
8  // explicitly implicit!
9  baz = foo ? true : false;
10 baz;                                  // true
```

Explicit Coercion

```
1 var now = +new Date();
2 // now = Date.now(); // ES5 only!
3
4 var foo = "foo";
5 // ~N -> -(N+1)
6 if (~foo.indexOf("f")) {
7     alert("Found it!");
8 }
```

Explicit Coercion

Implicit: happens as a side effect of some other operation

Implicit Coercion

Implicit: evil?

```
1  var foo = "123";
2  var baz = foo - 0;
3  baz;                    // 123
4
5  baz = foo - "0";
6  baz;                    // 123
7
8  baz = foo / 1;
9  baz;                    // 123
10
11 baz = 456;
12 foo = baz + "";
13 foo;                    // "456"
14
15 foo = baz - "";
16 foo;                    // 456 ... WAT!?
```

Implicit Coercion

```
1  var foo = "123";
2  if (foo) {                          // yup
3      alert("Sure.");
4  }
5
6  foo = 0;
7  if (foo) {                          // nope
8      alert("Right.");
9  }
10 if (foo == false) {                 // yup
11     alert("Yeah.");
12 }
13
14 var baz = foo || "foo";
15 baz;                                // "foo"
```

Implicit Coercion

```
1  var foo = "123";
2  if (foo == true) {              // nope!
3      alert("WAT!?");
4  }
5
6  foo = [];
7  if (foo) {                      // yup
8      alert("Sure.");
9  }
10 if (foo == false) {             // yup!
11     alert("WAT!?");
12 }
```

Implicit Coercion

```
"0" == null;            // false
"0" == undefined;       // false
"0" == false;           // true -- UH OH!
"0" == NaN;             // false
"0" == 0;               // true
"0" == "";              // false

false == null;          // false
false == undefined;     // false
false == NaN;           // false
false == 0;             // true -- UH OH!
false == "";            // true -- UH OH!
false == [];            // true -- UH OH!
false == {};            // false

"" == null;             // false
"" == undefined;        // false
"" == NaN;              // false
"" == 0;                // true -- UH OH!
"" == [];               // true -- UH OH!
"" == {};               // false

0 == null;              // false
0 == undefined;         // false
0 == NaN;               // false
0 == [];                // true -- UH OH!
0 == {};                // false
```

7/24

17/24

Implicit Coercion

```
1 "0" == false;          // true -- UH OH!
2 false == 0;            // true -- UH OH!
3 false == "";           // true -- UH OH!
4 false == [];           // true -- UH OH!
5 "" == 0;               // true -- UH OH!
6 "" == [];              // true -- UH OH!
7 0 == [];               // true -- UH OH!
```

NEVER use == false or == true!

```
1 [] == ![];        // true
2
```

Ask Yourself:

1. Can either value be true or false?
2. Can either value ever be [], “”, or 0?

```
42 == "43";                    // false
"foo" == 42;                   // false
"true" == true;                // false

42 == "42";                    // true
"foo" == [ "foo" ];            // true
```

Implicit Coercion: The Safe Parts

Primitive $\longleftrightarrow$ Native

Implicit Coercion

```
1 var foo = "123";
2 foo.length;                    // 3
3
4 foo.charAt(2);                 // "3"
5
6 foo = new String("123");
7 var baz = foo + "";
8 typeof baz;                    // "string"
```

Implicit Coercion

"Any sufficiently advanced technology is indistinguishable from magic."

--Arthur C. Clarke

Implicit Coercion

"Any sufficiently complex technology is indistinguishable from magic."

--many devs

Implicit Coercion

"Any sufficiently confusing technology is indistinguishable from magic."

--many devs

Implicit Coercion

== VS. ===

Implicit Coercion

~~== checks value~~

~~=== checks value and type~~

== allows coercion

=== disallows coercion

Implicit Coercion

```
1  var foo = [];
2  var baz = "";
3  if (foo == baz) {          // yup!
4      alert("Doh!");
5  }
6  if (foo === baz) {         // nope
7      alert("Phew.");
8  }
9
10 foo = 0;
11 if (foo == "") {           // yup!
12     alert("Argh!");
13 }
14 if (foo === "") {          // nope
15     alert("Phew.");
16 }
```

Implicit Coercion: == vs. ===

but... but...

Implicit Coercion: == vs. ===

```
1  var foo = "3";
2  if (foo === 3 || foo === "3") { // yup
3      alert("Thanks, but...");
4  }
5
6  if (foo == 3) {                 // yup!
7      alert("That's nicer!");
8  }
9
10 if (typeof foo === "string") {  // yup
11     alert("typeof always returns a string");
12 }
```

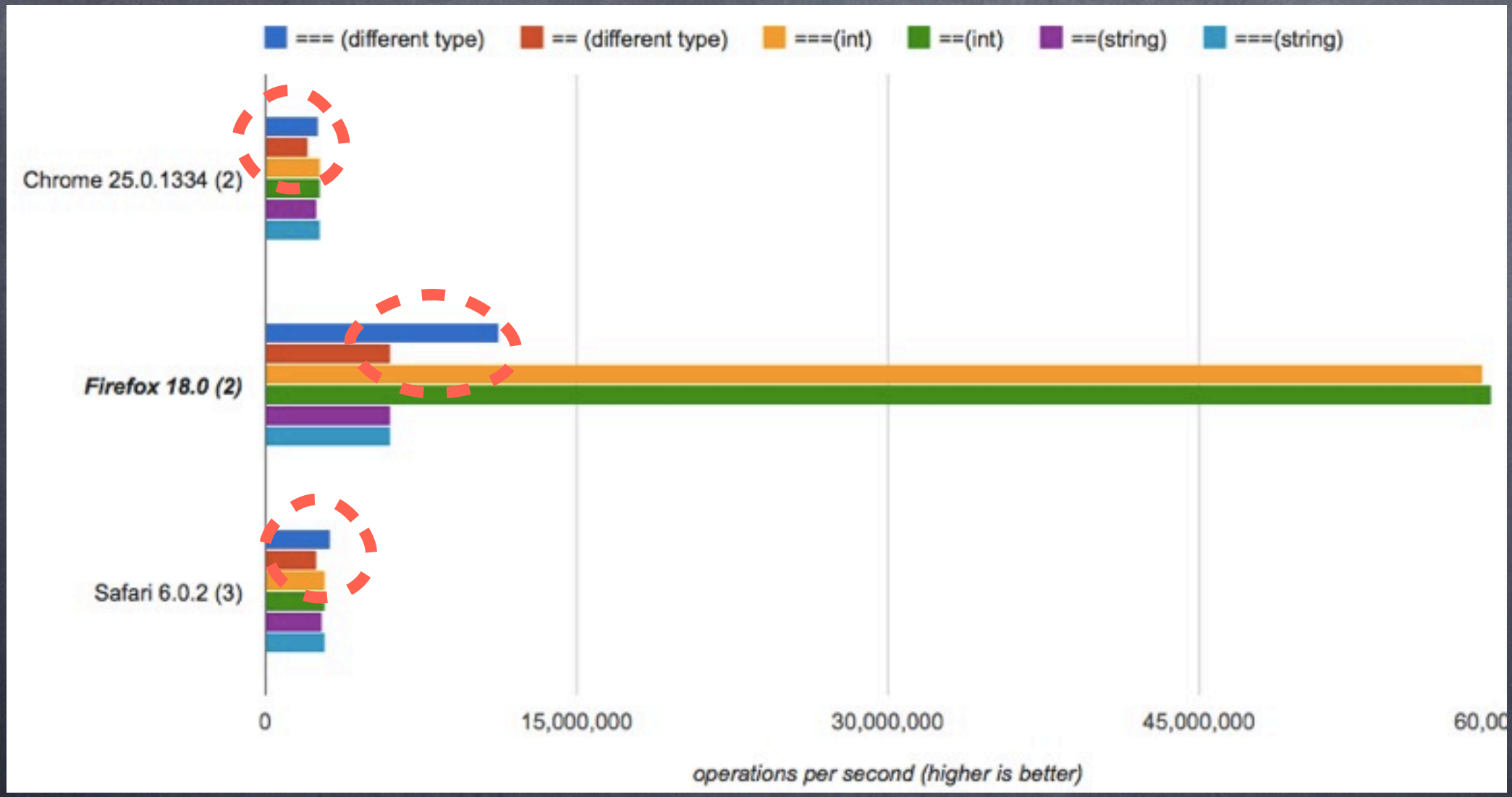
Implicit Coercion: == OK sometimes?

```
1  var foo;
2  if (foo == null) {        // yup!
3      alert("Thanks!");
4  }
5
6  foo = null;
7  if (foo == null) {        // yup
8      alert("Thanks, again!");
9  }
10
11 foo = false;
12 if (foo == null) {        // nope!
13     alert("Phew!");
14 }
```

Implicit Coercion: == OK sometimes?

What about the performance of == or ===?

http://jsperf.com/triple-equals-vs-double-equals/5



Implicit Coercion: == vs. === performance

7

- If the types compared are the same, ***they are identical***. That is to say they use ***the exact same algorithm***.
- If the types are *different*, then performance is irrelevant. Either you need type coercion, or you don't. If you don't need it, don't use `==` because the result you get may be unexpected.

share | edit | flag

answered **Nov 8 '11 at 1:18**

user1034768

add comment

Implicit Coercion: == vs. === performance

#FUD: using == in your code is dangerous. Avoid it!

#protip: use === where it's safer and use == where it's more helpful.

#FUD: JavaScript's implicit coercion is a flaw in the design of the language. Avoid it!

#protip: use explicit coercion where it's safer and use implicit coercion where it's more helpful.

http://jscoercion.qfox.nl/

http://webreflection.blogspot.com/2010/10/javascript-coercion-demystified.html

Coercion

(exercise #1: 5min)

Reference vs. Value

Variables that hold simple primitives (string, number, boolean, undefined, null) are held by value.

Assignments are by-copy.

Variables that hold anything else are held by reference.

Assignments are by-reference.

```
1  function foo(obj) {
2      obj.a = "foo";
3      obj.b = [1,2,3];
4      obj = { a: "baz", b: [4,5,6] };
5  }
6
7  var o = { a: "bar", b: [7,8,9] };
8  var a = o.a;
9  var b = o.b;
10
11 foo(o);
12
13 o.a;          // "foo"
14 o.b;          // [1,2,3]
15 a;            // "bar"
16 b;            // [7,8,9]
```

Reference vs. Value

```
var obj = { b: [1,2,3] };
var b = obj.b;

obj.b;                  // [1,2,3]
b;                      // [1,2,3]

obj.b = null;

obj.b;                  // null
b;                      // [1,2,3]

b = null;

obj.b;                  // null
b;                      // null
```

Reference vs. Value

http://wtfjs.com

```
1
2 parseInt("08");      // 0 ... WAT!?
3
4 parseInt(1/0,19);    // 18 ... WAT!?
5
```

Surprises

```
1 function foo(bar) {
2     return
3         bar + 2;
4 }
5
6 foo(2);       // undefined ... WAT!?
```

Surprises

```
1 42.;     // valid syntax!
2
3 (42).toFixed(2); // "42.00"
4
5 42.toFixed(2); // SyntaxError
6
7 42.888.toFixed(2); // "42.89"
8
9 42..toFixed(2); // "42.00"
```

Surprises

```
1 function hello(what) {
2     what = "world";
3     return "Hello, " + arguments[0] + "!";
4 }
5
6 hello("friends");   // Hello, world!
```

Surprises

```
1 // false -- hmmm...
2 String("abc") instanceof String;
3
4 // true
5 (new String("abc")) instanceof String;
6
7 // true -- wait, WAT!?
8 String("abc") == (new String("abc"));
```

Surprises

```
1
2  3 > 2 > 1;      // false ... WAT!?
3
```

Surprises

```
1
2  Array(5).join("wat" - 1) + " Batman!";
3  // NaNNaNNaNNaN Batman!
4
```

Surprises

Please use JavaScript responsibly.

If you're going to write JavaScript, appreciate it.

Agenda

Optimizing DOM & Events

- CSS Selectors
- DOM Management

```
1  <ul>
2      <li id="foo">
3          <p>Bond, James Bond</p>
4      </li>
5      <li>
6          <p>Never say never again</p>
7      </li>
8      <li class="bar">
9          <p>Shaken, not stirred</p>
10     </li>
11     <li class="bar">
12         <p>Hello, Moneypenny</p>
13     </li>
14 </ul>
```

CSS Selectors

```
1 #foo { ... }
2
3 li.bar { ... }
4
5 #foo ~ li.bar > p { ... }
```

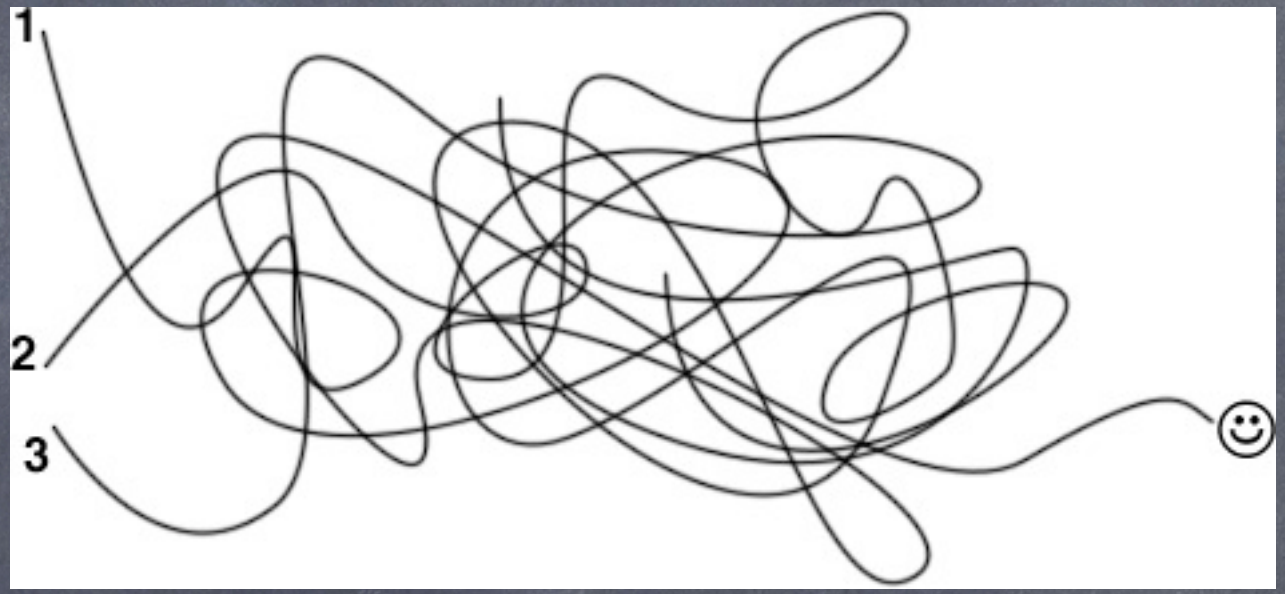
CSS Selectors

```
1 $("#foo ~ li.bar > p")
2 .addClass("highlighted")
3 .attr({
4     title: "Look at me!"
5 });
```

CSS Selectors

Left-to-Right
Right-to-Left

http://csswizardry.com/2011/09/writing-efficient-css-selectors/



CSS Selectors

```
1 $("#foo ~ li.bar > p")
2 .addClass("highlighted")
3 .attr({
4     title: "Look at me!"
5 });
```

CSS Selectors: under-specified?

```
1 $("ul li#foo ~ li.bar > p")
2 .addClass("highlighted")
3 .attr({
4     title: "Look at me!"
5 });
```

CSS Selectors: over-specified

```
1 var $bars = $("#foo ~ li.bar");
2
3 // later
4 $("> p", $bars)
5 .addClass("highlighted")
6 .attr({
7     title: "Look at me!"
8 });
```

CSS Selectors: jQuery "context"

```
1 var $bars = $("#foo ~ li.bar");
2
3 // later
4 $bars.children("p")
5 .addClass("highlighted")
6 .attr({
7     title: "Look at me!"
8 });
```

CSS Selectors: jQuery children

```
1 var $bars = $("#foo ~ li.bar");
2
3 // later
4 $bars.children("p")
5 .addClass("highlighted")
6 .attr({
7     title: "Look at me!"
8 });
```

jQuery Iterations

```
1  var $bars = $("#foo ~ li.bar");
2
3  // later
4  $bars.children("p").each(function(){
5      $(this)
6      .addClass("highlighted")
7      .attr({
8          title: "Look at me!"
9      });
10 });
```

jQuery Iterations

Quiz

```
1  <p id="p1">
2  Hello <em>world</em>!
3  </p>
4  <p class="p">
5  <strong>Now is the <em>time</em></strong>...
6  </p>
7  <script>
8  // find only these <em>'s
9   ??? .each(function(){
10      $(this)
11      .addClass("highlighted")
12      .attr({
13          title: "Look at me!"
14      });
15 });
16 </script>
```

Quiz

```
1  <p id="p1">
2  Hello <em>world</em>!
3  </p>
4  <p class="p">
5  <strong>Now is the <em>time</em></strong>...
6  </p>
7  <script>
8  // find only these <em>'s
9  $("#p1 > em, .p > strong > em").each(function(){
10     $(this)
11     .addClass("highlighted")
12     .attr({
13         title: "Look at me!"
14     });
15 });
16 </script>
```

(exercise #2: 10min)

DOM: toll road

```
1 for (var i=1; i<6; i++) {
2     $("h" + i)
3     .addClass("header");
4 }
```

DOM Management: lookups

```
1 var h = [];
2 for (var i=1; i<6; i++) {
3     h.push("h" + i);
4 }
5
6 $(h.join(","))
7 .addClass("header");
```

DOM Management: lookups

```
1 var mobsters = ["Lou","Frankie","Max"];
2 var $li;
3
4 for (var i=0; i<mobsters.length; i++) {
5     $li = $("<li></li>")
6     .text(mobsters[i]);
7
8     $("#hit_list").append($li);
9 }
```

DOM Management: lookups

```
1  var mobsters = ["Lou","Frankie","Max"];
2  var $li;
3  var $hit_list = $("#hit_list");
4
5  for (var i=0; i<mobsters.length; i++) {
6      $li = $("<li></li>")
7      .text(mobsters[i]);
8
9      $hit_list.append($li);
10 }
```

DOM Management: lookups

```
1  var mobsters = ["Lou","Frankie","Max"];
2  var $li;
3  var $hit_list = $("#hit_list");
4
5  for (var i=0; i<mobsters.length; i++) {
6      $li = $("<li></li>")
7      .text(mobsters[i]);
8
9      $hit_list.append($li);
10 }
```

DOM Management: insertions

```
1  var mobsters = ["Lou","Frankie","Max"];
2  var $li;
3  var lis = [];
4  var $hit_list = $("#hit_list");
5
6  for (var i=0; i<mobsters.length; i++) {
7      $li = $("<li></li>")
8      .text(mobsters[i]);
9
10     lis.push($li);
11 }
12
13 $hit_list.append(lis);
```

DOM Management: insertions

```
1  var mobsters = ["Lou","Frankie","Max"];
2  var lis = [];
3  var $hit_list = $("#hit_list");
4
5  for (var i=0; i<mobsters.length; i++) {
6      lis.push("<li>" + mobsters[i] +
7          "</li>");
8  }
9
10 $hit_list.html(lis.join(""));
```

DOM Management: insertions

Kyle Simpson
@getify
http://getify.me

Thanks!

Questions?