Web Development Intensive

Macy's

Part 1

- Web Development Basics
- Browser Developer Tools
- HTML Introduction
- Semantic HTML5 Elements
- CSS Introduction

Part 2

- CSS Selectors and CSS3 in-depth
- CSS Specificity & the Cascade
- CSS Layout: Box Model, Display & Positioning
- Browser Dependencies

Part 3

- JavaScript Introduction
- Basic Objects
- Control Flow
- Arrays
- Document Object Model (DOM) Manipulation
- jQuery Introduction

Part 4

- Basic Event Handling
- Browser Object Model (BOM)
- JavaScript Built-in Objects
- Objects In-depth
- JavaScript Inheritance

Part 3

Macy's

Part 3

- JavaScript Introduction
- Basic Objects
- Arrays and Loops
- Document Object Model (DOM) Manipulation
- jQuery Introduction

Review

Web Development Basics

- Build applications that are useful, usable and easy to maintain

- HTML: Application building blocks
- CSS: Application styling
- JS: Application logic and UI

JavaScript Introduction

History

- Designed and built in 10 days
- Netscape via Brendan Eich created it in 1995
  - Originally going to be called “Mocha”
  - Released as “LiveScript” in Netscape Navigator 2.0
  - Changed to JavaScript by end of 1995

History [cont.]

- Submitted to ECMA for standardization in 1998
  - ES2/1.3
  - ECMA: http://www.ecma-international.org/
- IE 9+ browsers
  - ES5/1.8 in 2009

History [cont.]

- ES6/ES2015
  - Partial support in most current browsers
- ES7: In progress
- Transpilers

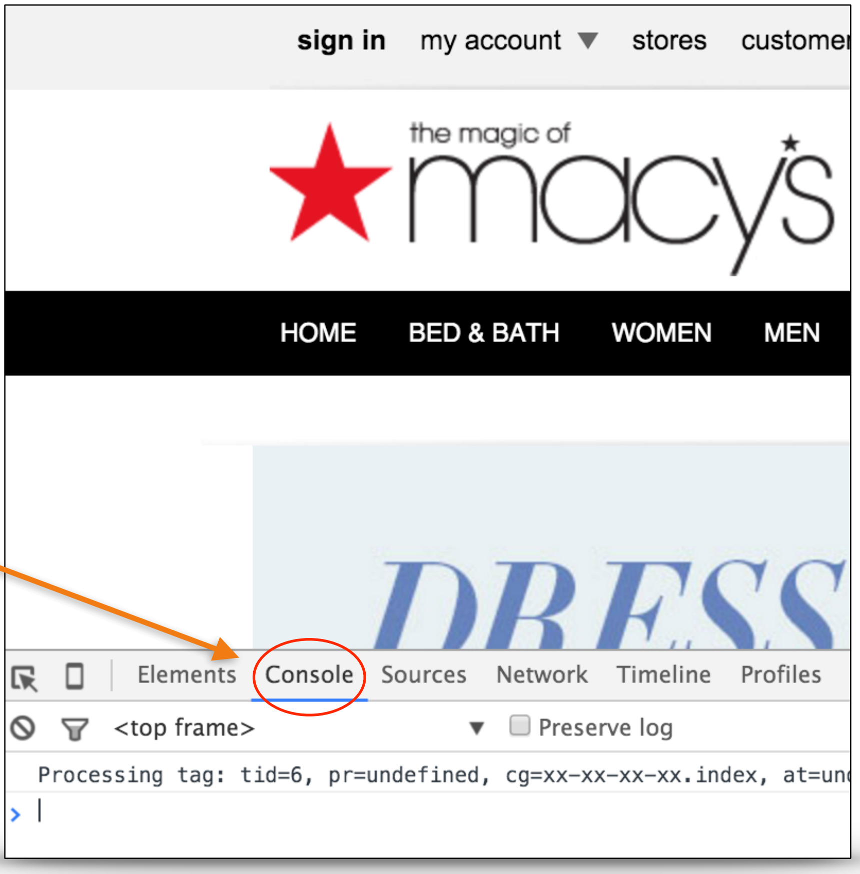
What is JavaScript Like?

- Interpreted
- “Loosely” typed
- Fully dynamic
- Case-sensitive C-style syntax
- Single threaded event loop
- Prototype-based

To the console

- Provides JavaScript console for interactive debugging

Console view

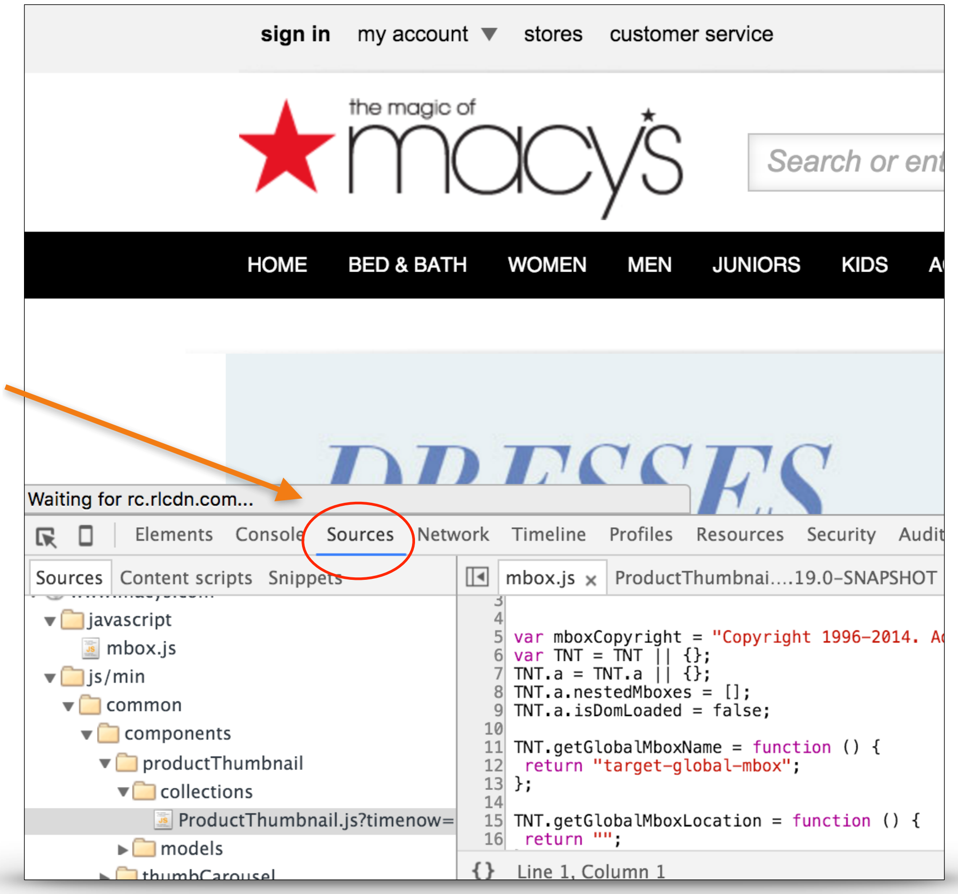


```
Processing tag: tid=6, pr=undefined, cg=xx-xx-xx-xx.index, at=un
>
```

Debugging

- Shows the JavaScript sources and debugging functionality

Sources view

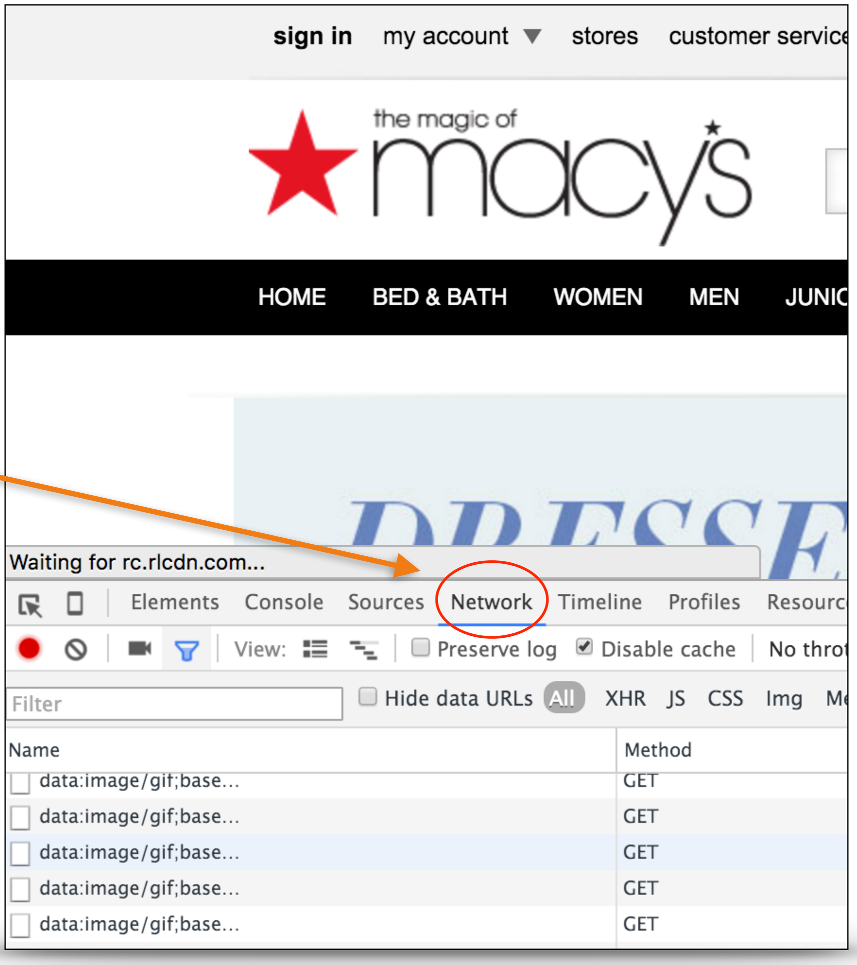


```
var mboxCopyright = "Copyright 1996-2014. A
var TNT = TNT || {};
TNT.a = TNT.a || {};
TNT.a.nestedMboxes = [];
TNT.a.isDomLoaded = false;

TNT.getGlobalMboxName = function () {
 return "target-global-mbox";
};

TNT.getGlobalMboxLocation = function () {
 return "";
```

Application Files

- Lists the all the media that has been downloaded

Network view



Waiting for rc.rlcdn.com...

Name	Method
data:image/gif;base...	GET
data:image/gif;base...	GET
data:image/gif;base...	GET
data:image/gif;base...	GET
data:image/gif;base...	GET

Language fundamentals

- Lexical structure
- Comments
- Data types
- Variables
- Constants
- Operators
- Control structures

Lexical structure

- Instructions are statements
  - Separated by semicolons
- Whitespace
  - Spaces, tabs and newlines
  - Generally not an issue
- Blocks
  - Wrapped in curly braces

Comments

- Single line

```
// A single-line comment
```

- Multiple line

```
/*
 A comment
 spanning multiple
 lines
*/
```

- Documentation comments

```
/**
 * Consumable by JSDoc for documentation purposes.
 */
```

Variables

- Must start with a letter, underscore or dollar sign

```
var hat;
var _hat;
var $hat;
```

- Subsequent characters can be alphanumeric, _ or $
- Case-sensitivity

Declaring Variables

- Use the **var** keyword
- One by one declaration

```
var foo = 'bar';
var thing1 = 2;
```

- In sequence

```
var a = 1, b = 2;
```

Declaring Variables [cont.]

- What happens if we don't use var?

```
foo = 'bar';
```

Constants

- Read-only named identifier
- ES5 has no implementation of a constant

- Constants can be specified by convention

```
var FOO = 'bar';
```

Primitive data types

- undefined
- null
- boolean
- number
- string

Getting the type

- The **typeof** keyword reveals the type

```
var pi = 3.14;
console.log(typeof pi);
```

undefined & null

- Not much difference between them

- Variables declared and not assigned are given **undefined**

```
var hat;
console.log(hat);
```

- Using a variable that hasn’t been declared throws an error

```
console.log(someHat);
```

undefined & null [cont.]

- Null can be used to specify a variable exists, but has no type or value

```
var hat = null;
console.log(hat);
```

- What is the typeof null and undefined?

```
console.log(typeof undefined);
console.log(typeof null);
```

number

- Numbers can be expressed as
  - Decimal: -9.81
  - Scientific: 2.998e8
  - Hexadecimal: 0xFF
  - Octal: 0777
  - Binary: 0b10000000000000000000000000000000
- Special numbers
  - NaN
  - Infinity
  - -Infinity

Number gotcha

- Numbers are stored internally as 64bit floating points

- What does the following code produce?

```
//That is 16 nines
var w = 9999999999999999;
console.log(w);

//Floating point math :)
var y = 0.2 + 0.1;
console.log(y);
```

String

- Can be enclosed by double or single quotes

```
//These are both acceptable string assignments
var x = "My string";
var y = 'My string';
```

- Operator for string concatenation

```
var z = 'My string' + ' ' + 'another string';
```

Basic Objects

Basic Objects

- new operator
- Object literal

new Operator

- JavaScript allows for newing up an object

```
//Verbose syntax
var x = new Object();
```

- **new** operator doesn’t need to be used to create a new object instance
- Hardly ever used to make a new Object instance

Object literal

- Objects are made up of key/value pairs
  - Similar to a Dictionary, Hash or Map

- Keys are unordered and are strings
- Values can be any type of data

```
//Verbose syntax
var x = { bar: 'baz' };
x.foo = true;
console.log('bar is' + obj.bar);
console.log(obj['foo']);
```

new Operator

- JavaScript allows for newing up an array

```
//Verbose syntax
var aTeam = new Array();
aTeam[0] = 'Tigers';
aTeam[1] = 'Lions';
aTeam[2] = 'Wings';

var fibonacci = new Array(2, 3, 5, 8);
```

- **new** operator is very rarely used to create a new array instance

Array literal

- Arrays are for storing sequenced data

```
var aTeam = ['Tigers', 'Lions', 'Wings'];
var fibonacci = [2, 3, 5, 8];
```

Literals

- What is a literal?
  - A fixed value that is “literally” provided in the script
  - Not a variable

```
//Literals
5          // number literal
'a'        // string literal
true       // boolean literal
{}         // object literal
[]         // array literal
/^(.*)$/   // regex literal
```

Literals [cont.]

- Primitives in JavaScript have built-in object counterparts

```
//Literals
var x = 5;
var y = Number('5');
```

- **null** and **undefined** have no built-in counterpart

Literals [cont.]

- Literals are stored as low-level values until they are accessed
  - Like they are instances of the object counterpart

```
"I'm a string!".length;
"foo".toUpperCase();
0xFF.toString();
```

Wrapper objects

- We could directly interact with the object counterparts
  - Usually this will never happen

```
var s = "test", n = 1, b = true;
var S = new String(s);
var N = new Number(n);
var B = new Boolean(b);
```

Type conversion

- Variable type can be converted on the fly

- String conversation

```
// String conversion
var x = 'The answer is ' + 42;
var y = 42 + ' is the answer';
```

- Numeric conversion

```
// Numeric conversion
var x = Number('42');
```

Type conversion [cont.]

- What output is given?

```
var a = 10 + ' objects';
console.log('a is: ' + a);
var b = '7' * '4';
console.log('b is: ' + b);
var c = 1 - 'x';
console.log('c is: ' + c);
var d = '37' - 7;
console.log('d is: ' + d);
var e = 8 * null;
console.log('e is: ' + e);
```

Type conversion [cont.]

- What output is given?

```
//Hijinks
var a = 8 * null;
console.log('a is: ' + a);
var b = [] + [];
console.log('b is: ' + b);
var c = [] + {};
console.log('c is: ' + c);
var d = {} + {};
console.log('d is: ' + d);
```

Type conversion [cont.]

- What output is given?

```
var a = Number('42');
var b = 42;
var c = true;
var d = '42';
var e = new Boolean();
var f = undefined;
var g = null;
var h = {};
var i = [];
```

```
console.log(typeof a);
console.log(typeof b);
console.log(typeof c);
console.log(typeof d);
console.log(typeof e);
console.log(typeof f);
console.log(typeof g);
console.log(typeof h);
console.log(typeof i);
```

Check the instance

- **instanceof** allows for checking whether the object on left is an instance of object on right

- What output is given?

```
var aBool =  new Boolean();
aBool instanceof Object;
aBool instanceof Boolean;
aBool instanceof String;
aBool instanceof Number;
```

```
var aString =  '42';
aString instanceof String;
aString instanceof Object;
```

```
var anArray =  ['a', 'b', 'c'];
anArray instanceof Object;
anArray instanceof Array;
anArray instanceof String;
anArray instanceof Number;
```

```
var aNum =  42;
aNum instanceof Number;

var bNum = new Number(42);
bNum instanceof Number;
```

Recap

- How many primitive types are there?
- What is an object used for?
- How do we declare variables?
- What does it mean when we say, types are implicitly converted

- Remember, almost everything is an object

Exercise: JavaScript

- Goal: Gain familiarity with JavaScript syntax

- Specifications:
  - Declare 2 to 5 variables (name them as you like)
    - Make at least one string, one number and one boolean.
    - Log the typeof each to the console
  - Declare a new variable that stores the result of a mathematical expression
    - You can add/multiply/divide/modulo any set of numbers
    - Log the result to the console
  - Combine at least three strings together
    - Log the result to the console

Exercise: JavaScript

- Goal: Gain familiarity with JavaScript syntax

- Specifications:
  - Create an array with at least 5 values stored inside
    - Log the result to the console
  - Use Array.prototype.concat to add another array to your original array
    - Describe what happens to the original array?
    - Log the result to the console

Exercise: JavaScript

- Goal: Gain familiarity with JavaScript syntax
- Specifications:
  - Define several properties within the object (name, hairColor, age, etc..)
  - Define a property that stores an array ("siblings" or "favoriteColors")
    - Log the object to the console
    - Then log ONLY the "name" property to the console
    - Then change the "name" property to something else, then log the object to the console

Operators

Operators

- Unary
- Binary

Unary

- Examples

```
var obj = { x:5 }
delete obj.x

typeof 5
+'5'
!true

var x = 0
++x
x++
--x
x--
```

Arithmetic

- Examples

```
5 + 5
5 - 3
5 * 2
10 / 2
10 % 3
```

Relational

- Examples

```
'foo' in {foo: true}
[] instanceof Array
5 < 4
5 > 4
4 <= 4
5 >= 10
'zebra' > 'aardvark'
'Zebra' > 'aardvark'
```

Assignment

- = Assignment operator
- ! Not operator

```
var isTheBest = true;

console.log('Is this the best: ' + !isTheBest);
```

Assignment

- Example

```
x=5
x+=1
x-=2
x*=3
x/=4
x%=2
```

Equality

- == Comparison operator
- != Non-equality operator
- Type conversion takes place
  - A string could be equal to a number =)

```
5 == 5;
5 == '5';
5 == 'a';
```

Identity (true comparison)

- === Identity operator
- !== Non-identical operator
- No type conversion takes place
  - A string could not be equal to a number

```
5 === 5;
5 === '5';
5 === 'a';
```

Equality & Identity

- What output is given?

```
var a = Number('42');
var b = 42;
var c = 23;
var d = '42';

console.log(a == c);
console.log(a == d);
console.log(a == b);
console.log(a === d);
console.log(a === b);
```

Logical Operators

logic	operator
and	&&
or	||
not	!

```
if ( (raining || snowing) && !inside ) {
       console.log('You are in Michigan!');
}
```

```
false && 'foo'
false || 'foo'
```

Control Structures

Control Structures

- Conditionals
- Ternary operator
- while loop
- do while loop
- for loop
- for in loop

If Statement

- Flow control based on conditions
  - if conditionals

```
if('condition to test') {
  console.log('condition is true');
}
```

  - if ... else operations

```
if ('condition to test') {
  console.log('condition is true');
} else {
  console.log('condition is false');
}
```

If Statement

- Flow control based on conditions
  - More specific control

```
var a = 42;
var b = 23;
var c = '42';

if (a === b) {
  console.log('a and b are both are identical');
} else if (a === c) {
  console.log('a and c are both are identical');
} else {
  console.log('no match found');
}
```

Switch

- Control flow based on conditions
    - More than one if/else

```
var foo = 0;
switch (foo) {
  case -1:
    console.log('negative 1');
    break;
  case 0:
    console.log(0);
    break;
  default:
    console.log('default');
}
```

- What happens if we set **foo = ‘0’**

Ternary

- Ternary operator
  - More specific control

```
condition ? result1 : result2;
  result1 if the condition is true
  else result2 if the condition false
```

```
  var c = 6;
  var b = (c === 6) ? true : false;
  console.log('b is: ' + b);
```

Truthy / Falsy

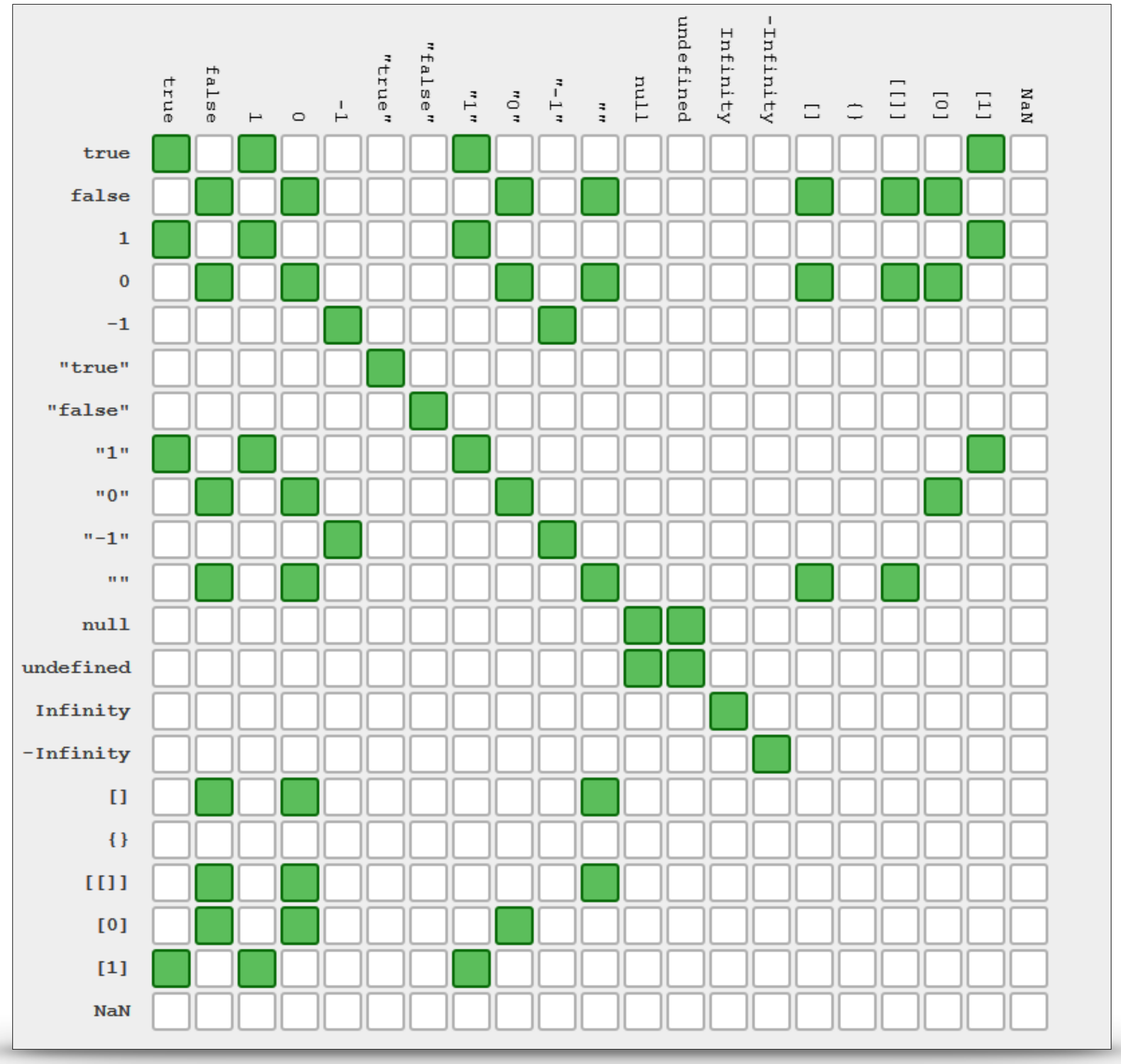
- Falsy

```
false
null
undefined
''
0
NaN
```

- Truthy

```
{}
[]
'0'
'false'
```

Truthy / Falsy



	true	false	1	0	-1	"true"	"false"	"1"	"0"	"-1"	""	null	undefined	Infinity	-Infinity	[]	{}	[[]]	[0]	[1]	NaN
true	🟩		🟩					🟩												🟩	
false		🟩		🟩					🟩		🟩					🟩		🟩	🟩		
1	🟩		🟩					🟩												🟩	
0		🟩		🟩					🟩		🟩					🟩		🟩	🟩		
-1					🟩					🟩											
"true"						🟩															
"false"							🟩														
"1"	🟩		🟩					🟩												🟩	
"0"		🟩		🟩					🟩										🟩		
"-1"					🟩					🟩											
""		🟩		🟩							🟩					🟩		🟩			
null												🟩	🟩								
undefined												🟩	🟩								
Infinity														🟩							
-Infinity															🟩						
[]		🟩		🟩							🟩										
{}																					
[[]]		🟩		🟩							🟩										
[0]		🟩		🟩					🟩												
[1]	🟩		🟩					🟩													
NaN																					

Loops

- for
- while
- do/while
- for … in

While Loop

- Repeats as long as condition is true

```
while (condition_to_check_at_each_iteration) {
  repeat over and over while the condition is true;
}
```

```
var x=0;
var listOfNumbers='';

while (x < 5) {
  listOfNumbers +=  x + ', ';
  x++;
}
console.log(listOfNumbers);
```

Do ... While Loop

- Similar to while loop
  - However, if condition is false the loop still runs once

```
var i = 0;
do {
   //Does stuff 10 times
   statement_1;
   statement_2;
   statement_3;
   i++;
} while (i < 10)
```

For Loop

- Similar to while loop
- However, all of the requirements are combined:
  - Initialization
  - A condition to test for that eventually proves false
  - An amount by which to increment the variable

```
for (count_to_initialize; condition_to_test;
change_value_of_count) {
  repeat over and over while the condition is true;
}
```

```
for (var i = 0; i < 3; i++) {
  console.log('Loop: ' + i);
}
```

For Loop Optimization

- Alright

```
var i = 0;
for (i; i < anArray.length; i++) {
  console.log('Loop: ' + i);
}
```

- Better

```
var i = 0,
  len = 0;
for (i, len = anArray.length; i < len; i++) {
  console.log('Loop: ' + i);
}
```

for…in

- Loops over enumerable properties of an object

```
var x;
var obj = { foo: true, bar: false };

for (x in obj) {
  console.log('This is the property: ' + x);
  console.log('This is the value: ' + obj[x]);
}
```

for…in

- Loops over inherited properties as well

- A better implementation

```
var x;
var obj = { foo: true, bar: false };

for (x in obj) {
  if (obj.hasOwnProperty(x)) {
    console.log('This is the property: ' + x);
    console.log('This is the value: ' + obj[x]);
  }
}
```

Loopy Thoughts

- **for loops**: usually used when you know how many times you want to loop
- **while loops**: usually used when you don't know how many times you want to loop
- **do while loops**: used when you want at least one iteration of the loop

Speed Demon

- While loop decrementing

```
var x = 5;
var listOfNumbers = "";

while(x--) {
	listOfNumbers += x + ",";
}
console.log("listOfNumbers decrement: " + listOfNumbers);
```

Exercise: JS Loops

- Goal: Gain familiarity with JS loops

- Specifications
  - Write a program that console.logs from 1 to 100 separated by spaces
    - For numbers that are a multiple of 3, log “Fizz”
    - For numbers that are a multiple of 5, log “Buzz”
    - For numbers that are a multiple of both, log “FizzBuzz”

Arrays

Array

- Used in storing an ordered list of values
  - Store whatever you want in an array
  - Dynamic size
  - zero-based

```
var aTeam = new Array();
aTeam[0] = 'Tigers';
aTeam[1] = 'Lions';
aTeam[2] = 'Wings';

var fibonacci = new Array(2, 3, 5, 8);

var weird = new Array(5, false, 'z');
```

Array Literal

- Same as using Array Constructor for all intents and purposes

```
var aTeam = ['Tigers', 'Lions', 'Wings'];

var fibonacci = [2, 3, 5, 8];
```

Associative Array

- Also known as hash, map, or dictionary
- Think of them as key/value pairs
- Use keys rather than indexes to get to elements

```
var states = [];
states["AL"] = "Alabama";
states["AK"] = "Alaska";
states["AZ"] = "Arizona";

console.log("The first state is: " + states[0]);
console.log("The first state is: " + states["AL"]);
console.log("The first state is: " + states.AL);
console.log(states);
```

Array.forEach()

- Provides a callback one time for each element in the array
- Parameters passed into the callback
  - **element**: element value at the specified index
  - **index**: the indexed number as an integer
  - **array**: the array that is being traversed

```
var anArray = ['a', 'b', 'c'];

anArray.forEach(function(element, index, array) {
    console.log("element:" + element +
          " index:" + index +
          " array:" + array);
});
```

Array.forEach() [cont.]

- ECMAScript 5 addition: Doesn't work in IE8
- Demo @
  - http://jsbin.com/xexised/1/edit?html,js,console

Array.every()

- ECMAScript 5 addition: Doesn’t work in IE8
- Provides a callback one time for each element in the array until one is found where the callback returns a falsy value
- Does not change the array it is called on
- Parameters passed into the callback
  - **element**: element value at the specified index
  - **index**: the indexed number as an integer
  - **array**: the array that is being traversed

Array.every() [cont.]

- Callback function for test
  - Each element has to pass for a truthy returned value
- Once callback gets a falsy value the looping stops
- Demo @
  - http://jsbin.com/pikibir/1/edit?html,js,console

```
function isSmaller(element, index) {
    console.log("index: " + index);
      return element <= 42;
}

var goodSet = [3, 25, 34, 42].every(isSmaller);
var badSet = [3, 25, 45, 42].every(isSmaller);

console.log("Did they pass- Good:" + goodSet +
  " Bad:" + badSet);
```

Array.some()

- ECMAScript 5 addition: Doesn’t work in IE8
- Provides a callback one time for each element in the array until one is found where the callback returns a truthy value
- Does not change the array it is called on
- Parameters passed into the callback
  - **element**: element value at the specified index
  - **index**: the indexed number as an integer
  - **array**: the array that is being traversed

Array.some() [cont.]

- Callback function for test
  - Each element has to pass for a truthy returned value
- Once callback gets a truthy value the looping stops
- Demo @
  - http://jsbin.com/xafero/1/edit?html,js,console

```
function isSmaller(element, index) {
  console.log("index: " + index);
  return element <= 42;
}

var goodSet = [3, 25, 45, 42].some(isSmaller);
var badSet = [45, 46].some(isSmaller);

console.log("Did they pass- Good:" + goodSet +
  " Bad:" + badSet);
```

Array.filter()

- ECMAScript 5 addition: Doesn’t work in IE8
- Provides a callback one time for each element in the array and creates an array with all the elements that pass the test
- Does not change the array it is called on
- Parameters passed into the callback
  - **element**: element value at the specified index
  - **index**: the indexed number as an integer
  - **array**: the array that is being traversed

Array.filter() [cont.]

- Callback function for test
  - Each element has to pass for a truthy returned value
- Processes every element
  - Returns the filtered array
- Demo @
  - http://jsfiddle.net/kamrenz/z6kKG/1/

```
function isSmaller(element) {
  return element <= 42;
}

var anArray = [3, 25, 26, 45, 32,  42];

var filteredSet = anArray.filter(isSmaller);

console.log("Filtered Set: " + filteredSet);
```

Array.map()

- ECMAScript 5 addition: Doesn’t work in IE8
- Provides a callback one time for each element in the array and creates an array with the result of the callback on each element
- Does not change the array it is called on
- Parameters passed into the callback
  - **element**: element value at the specified index
  - **index**: the indexed number as an integer
  - **array**: the array that is being traversed

Array.map() [cont.]

- Processes every element
  - Returns the mapped array
- Demo @
  - http://jsfiddle.net/kamrenz/5Lssc/

```
function areaOfASquare(length) {
  return length * length;
}

var lengths = [1, 3, 5];

var areas = lengths.map(areaOfASquare);

console.log("Areas are: " + areas);
```

Array.indexOf()

- ECMAScript 5 addition: Doesn’t work in IE8
- Used to find an index of an element
  - if found it returns the first index of the found element
  - if not found it returns a -1
- Demo @
  - http://jsfiddle.net/kamrenz/vg44Y/

```
var anArray = ['a', 'b', 'c', 'b'];

console.log("The index of 'b' is: " + anArray.indexOf('b'));
```

Array.lastIndexOf()

- ECMAScript 5 addition: Doesn’t work in IE8
- Used to find the last index of an element
  - if found it returns the last index of the found element
  - if not found it returns a -1
- Demo @
  - http://jsfiddle.net/kamrenz/ex6b2/

```
var anArray = ['a', 'b', 'c', 'b'];

console.log("The last index of 'b' is: " +
anArray.lastIndexOf('b'));
```

Array.push()

- The push method places elements at the end of an array
- Changes the array it is called on
- Demo @
  - http://jsfiddle.net/kamrenz/FV5Ys/1/

```
var aTeam = ['Tigers', 'Broncos', 'Wings'];

console.log("length: " + aTeam.length);

aTeam.push('Pistons');

console.log("length: " + aTeam.length);
console.log("I like the " + aTeam[3]);
```

Array.pop()

- The pop method removes the element at the end of an array
  - It also returns the item that was removed
- Returns the element that was removed
- Changes the array it is called on: LIFO
- Demo @
  - http://jsfiddle.net/kamrenz/FV5Ys/1/

```
var aTeam = ['Tigers', 'Lions', 'Wings'];
console.log("length: " + aTeam.length);
aTeam.push('Pistons');
console.log("length: " + aTeam.length);
aTeam.pop()
console.log("I like the " + aTeam[3]);
```

Array.shift()

- The shift method takes away elements at the beginning of an array
- Your array can function like a queue
- Returns the element that was removed
- Changes the array it is called on: FIFO
- Demo @
  - http://jsfiddle.net/kamrenz/CQM4v/

```
var aTeam = ['Tigers', 'Lions', 'Wings'];
var removed = aTeam.shift();

console.log("Only these teams " + aTeam);
```

Array.unshift()

- The unshift method adds an element to the front of the array
  - Takes an argument to put into the array
  - Returns the length of the array
- Your array can function like a queue

```
var aTeam = ['Tigers', 'Lions', 'Wings'];
var length = aTeam.unshift('Pistons');

console.log("Only these teams " + aTeam);
```

Array.join()

- join allows the contents of the array to be saved in a user-friendly textual format (i.e. joined via a specified separator)
- Does not change the array it is called on
- Demo @
  - http://jsfiddle.net/kamrenz/PB95D/

```
var aTeam = ['Tigers', 'Lions', 'Wings'];

console.log("I like " + aTeam.join());
console.log("I like " + aTeam.join(", "));
```

Array.concat()

- concat allows the contents of 2 disparate arrays to be merged together
- Does not change the array it is called on
- Demo @
  - http://jsfiddle.net/kamrenz/LhQQx/

```
var aColors = ['Red', 'Blue', 'Yellow'];
var bColors = ['Green', 'Orange', 'Purple'];
var allColors = aColors.concat(bColors);

console.log("I like all colors:" +
allColors.join(", "));
```

Array.slice()

- slice allows a selection of array elements to be take from an array
- Does not change the array it is called on
- Parameters
  - begin: zero-based index
  - end: zero-based index, slicing up to but not including it

Array.slice() [cont.]

- Demo @
  - http://jsfiddle.net/kamrenz/EXLuG/1/

```
var allColors = ['Red', 'Blue', 'Yellow', 'Green',
   'Orange', 'Purple'];

var primaryColors = allColors.slice(0,3);

console.log("I like primary colors:" +
  primaryColors.join(", "));
```

Array.splice()

- splice allows a selection of array elements to be take from an array
- Change the array it is called on
- Parameters
  - begin: zero-based index
  - end: length (i.e. number) of elements to take off the original array

Array.slice() [cont.]

- Demo @
  - http://jsbin.com/baqusag/2/edit?html,js,console

```
var allColors = ['Red', 'Blue', 'Yellow', 'Green',
  'Orange', 'Purple'];

var aColor = allColors.splice(4,1);

console.log("I like the color: " + aColor);
console.log("We have less colors now: " +
  allColors.join(", "));
```

Array.reverse()

- reverse rearranges the array from back to front
- Changes the array it is called on
- Demo @
  - http://jsfiddle.net/kamrenz/M8dyx/

```
var aTeam = ['Tigers', 'Lions', 'Wings'];
aTeam.reverse();

console.log("I like " + aTeam.join(", "));
```

Exercise: JS Arrays Part A

- Goal: Gain familiarity with JS arrays

- Specifications
  - How can you print out the following sentence by only using single quotes.
    - print me: This isn’t too hard
  - Create an array of your favorite 3 hobbies
    - Add a couple of other hobbies via array methods
    - Remove your very first hobby (i.e. index 0)
      - Can you think of another way?

Exercise: JS Arrays Part A [cont.]

- Goal: Gain familiarity with JS arrays

- Specifications [cont.]
  - Prompt the user to enter an array separator (i.e. :)
    - Display the array contents separated by the separator via an alert
    - Look up a “prompt” at Mozilla Developer Network
  - Display the location of the first vowel in your very first hobby
    - Use the specific character for search (i.e. “a”, “e”, or “u”) not a generic vowel. :)

Exercise: JS Arrays Part B

- Goal: Gain familiarity with JS loops

- Specifications [cont.]
  - Write code that compares 2 arrays of length 3 to see if they are equal
    - Make console statement say “Are these two arrays equal?”
    - **yes** or **no**

Functions

First Class Citizens

- Just like numbers and booleans
- Can be constructed at run-time
- Can be assigned to a variable
- Can be passed to a function
- Can be returned from a function

Functions Arguments

- No overloading of a function
- No need to define different functions for argument lists

```
function doIt(a, b, c) {
   console.log('a:' + a + ' b:' + b + ' c:' + c);
}

doIt(4, 5, 6);
doIt("three", "two");
doIt(9, 8, 7, 6, 5, 4, 3, 2, 1);
```

State

- The ability to store values in variables
- The ability to retrieve values from variables
- Scope is the defined set of rules for holding state
  - Storing the variables somewhere
  - Finding the variables in some location
  - Retrieving the variables at a later point

Functions Scope

- **var** sets a variable in a local scope (i.e. the function doIt())
  - The local scope is the functions execution context
  - Variable y is in scope when the function doIt is running because it is accessible anywhere in that function

```
function doIt () {
  var y = 24;
  console.log("y inside: " + y);
}

doIt();
console.log("y outside: " + y);
```

Functions Scope [cont.]

- Without a var we have global scope
  - Global scope is accessed via **window** in browsers

```
function doIt () {
  console.log("here");
  y=24;
  console.log("y inside: " + y);
}

doIt();
console.log("y outside: " + y);
```

Functions Scope [cont.]

- How about this…

```
function outer(a, b, c) {
  var b = a * 4;

  function inner(c) {
    console.log("variables: " + a + " " +
      b + " " + c);
  }

  inner(b * 2);
}

outer(3);
```

Functions Scope [cont.]

- JavaScript has function-based scope

```
//var i is function scoped
for (var i=0; i<10; i++) {
  console.log("hello");
}

//var aNumber is function scoped
if (aVariable) {
  var aNumber = 42;
}
```

Freedom

- They are not bound to any objects
- They themselves are objects
- They do not need to be embedded in a class
- They can be passed around as arguments

Freedom [cont.]

- We can pass functions as arguments

```
function speakIt (phrase, aFunction) {
  var sentence = '';
  var count = aFunction();
  var i = 0;
  for (i=0; i<count; i++) {
    sentence += phrase + ' ';
  }
  console.log(sentence);
}

speakIt('woof', function () { return 4/2; });
```

Immediate Invoked

- Functions can immediate call themselves if need be
  - IIFE: Immediate Invoked Function Expressions
  - The function is wrapped in parenthesis because the parser then knows it is a function **expression** not **declaration**

```
//Douglas Crockford's suggestion
(function () {
  var speak = ‘yelp’;
  console.log(speak);
}());

//This works fine also
(function () {
  var speak = ‘yelp’;
  console.log(speak);
})();
```

```
//Syntax error: a declaration
function () {
  console.log("yelp");
}();

//You meant: an expression
var aFunction = function () {
  console.log("yelp");
}();
```

Immediate Invoked [cont.]

- What if we want to reference global variables within our IIFE

```
var aNumber = 42;

(function (global) {
  var aNumber = 26;
  console.log("Numbers- local:" + aNumber +
    " global:" + global.aNumber);
}(window));
```

Functions as Values

- Everything in JavaScript is a value
- We can use function names like we would strings

```
var aVariable = null;
function aFunction() { return "Hello"; }
console.log((aVariable || aFunction)());
```

A Final Example

- Let's make sure you get it

```
//Function signature
var calculateArea = function (length) {
  area = length * length;
  console.log("inside area: " + area);
  return area;
};

//Calling the function
var outsideArea = calculateArea(2);
console.log("outside area:" + outsideArea);
console.log("inside area 2:" + area);
```

Exercise: JS Functions

- Goal: Gain familiarity with JS functions

- Specifications
  - Let's take the logic from our previous FizzBuzz exercise and make it functional

  - Create a function that:
    - Accepts a single number argument
    - Returns the proper FizzBuzz result for that number
  - Loop through 1...100 as before, but using the function to output the proper values

jQuery

jQuery Foundation

http://jquery.com

http://jqueryui.com

http://sizzlejs.com

http://qunitjs.com

http://jquerymobile.com

jQuery Foundation [cont.]

- “jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers.” - jQuery.com

Simple Scripting

- How do we find which element of a radio group is currently checked and then get its value?

Simple Scripting [cont.]

- How do we find which element of a radio group is currently checked and then get its value?
  - IE8 holds us back a bit

```
var checkedValue;
var elements = document.getElementsByTagName('input');
for (var i=0; i < elements.length i++) {
  if (elements[i].type == 'radio' &&
    elements[i].name == 'someRadioGroup' &&
    elements[i].checked) {
      checkedValue = elements[i].value;
}
```

Simple Scripting [cont.]

- How do we find which element of a radio group is currently checked and then get its value?

jQuery

```
var checkedValue = $('[name="someRadioGroup"]:checked').val();
```

- jQuery takes advantage of Sizzle JS Selector Engine
  - http://www.sizzlejs.com

Simple Scripting [cont.]

- How do we find which element of a radio group is currently checked and then get its value?
  - POJS does get us pretty close we still have to loop

```
var elements = document.querySelectorAll('[name="someRadioGroup"]');
for (var i=0; i < elements.length i++) {
  if (elements[i].checked) {
      checkedValue = elements[i].value;
}
```

jQuery

- How do we get it?
  - http://jquery.com/download/

- jQuery 1.x
  - Supports IE 6/7/8 and above
- jQuery 2.x
  - Supports IE 9 and above

```
<!--[if lt IE 9]>
   <script src="js/libs/jquery-1.x.js"></script>
<![endif]-->
<!--[if gte IE 9]><!-->
   <script src="js/libs/jquery-2.x.js"></script>
<!--<![endif]-->
```

jQuery API

- Where do we get our hands on the API?
  - http://api.jquery.com
- How do we use jQuery
  - DOM manipulation
  - Utility methods
  - Unobtrusive JavaScript

jQuery Basics

jQuery Basics

- Getting DOM elements
- Manipulating the DOM
- Utility methods
- Unobtrusive JavaScript
- Loops

jQuery Basics

Grabbing DOM Elements

Grab Element by the ID

- Grabs 1 element
- Plain old JavaScript
  - document.getElementById(‘sweetId’);
- jQuery uses CSS selectors
  - $(‘#sweetId’);

Grab Element by Its Name

- Grabs a list of elements

- Plain old JavaScript
  - document.getElementsByTagName(‘input’);
- jQuery
  - $(‘input’);

Grab Elements by CSS

- CSS selector grabs a list of elements

- Plain old JavaScript
  - document.querySelectorAll(‘[name=“someRadioGroup"]’);
- jQuery
  - $(‘[name="someRadioGroup"]’);

jQuery Wrapper

- How does this $/jQuery work?

- $() is an alias for jQuery()
- This "wrapper" wraps the collected elements
- Allows for extended JavaScript functionality

jQuery Wrapper [cont.]

- What is returned from jQuery() / $()?

- A jQuery object instance
- Allows for jQuery's famous chaining abilities

jQuery Basics

DOM Traversal

Filtering Grabbed Elements

- Searches through all elements
- Allows us to reduce the set of returned elements from a selector query
- **.filter()**
  - http://api.jquery.com/filter/

```
$('p.message');
$('p').filter('.message')
```

Filtering Grabbed Elements

- jQuery expands the CSS3 offerings
  - :even … 0 based selection
  - http://api.jquery.com/category/selectors/jquery-selector-extensions/

- A best practice when using these additional selectors is to first grab a selection of elements and then filter

```
//Not as performant
$(‘li:even');

//Better optimization
$('li').filter(':even')
```

Filtering [cont.]

- .**filter**() can also take a function as its argument
- Allows for matching against complex tests
- This shows us the foundation of implicit iteration in jQuery

```
$('a').filter(function() {
  return this.hostname !== location.hostname;
}).addClass('external');
```

- Not just one anchor, but all anchors will be run through this filter to have an **external** class added

Find Elements in List

- Searches through all child elements only
  - A context needs to be provided
- .**find**()
  - http://api.jquery.com/find/
  - Allows for a search through the descendants of the wrapped set
  - Sometimes more complex selectors can be simplified with find

```
$('p span');
$('p').find('span')
```

- Finds all the **p** elements and then **span** elements that are their children

Let’s explore

- Let’s take a look at find(), filter(), descendant selector and the combinator
  - http://jsfiddle.net/kamrenz/6tzeP/9/

Grab Next Element

- **.next()**
  - http://api.jquery.com/next/
  - Grabs the sibling that is immediately following the selected element
- **.nextAll()**
  - http://api.jquery.com/nextAll/
  - Grabs all of the following siblings of the selected element
- Both can take a selector as an argument
  - .next("div")
  - Will the select the next element if it is a div
- Comparison
  - http://jsfiddle.net/kamrenz/txnB8/13/

Up the tree

- **.closest()**
  - http://api.jquery.com/closest/
  - Returns the closest ancestor matching the selector parameter
    - This could include itself
- **.parents()**
  - http://api.jquery.com/parents/
  - Returns the ancestors matching the selector parameter
  - If no selector added it will return all the parent elements including the **body** and **html** root element
- **.parent()**
  - Similar to parents() but it only goes up one level
- https://jsfiddle.net/kamrenz/Lm1rj6eL/2/

Down the tree

- **.children()**
  - http://api.jquery.com/children/
  - Returns the descendants (i.e. children)
  - Different from .find() in that it only traverses 1 level
  - http://jsfiddle.net/kamrenz/9CLu5/6/
- **.siblings()**
  - http://api.jquery.com/siblings/
  - Returns the siblings of the selected element
  - It can take a CSS selector as a way to go down the tree to find siblings
  - http://jsfiddle.net/kamrenz/XGY4A/5/

jQuery Basics

Manipulating the Elements

Element Creation

- How do we create elements from scratch?

- Simple jQuery functionality

```
$("<h1>In the beginning…</h1>");
```

  - Creates an h1 element

```
$("<div>");
```

  - Creates an empty div element
  - Shortcut for $("<div></div>")

Class Interaction

- **.addClass()**
  - http://api.jquery.com/addClass/
  - Add a CSS class to an element

```
$("<h1 class='title'>In the beginning…</h1>").addClass("header");
```

- **.removeClass()**
  - http://api.jquery.com/removeClass/
  - Removes a CSS class on an element

```
$("h1").removeClass("title");
```

CSS Manipulation

- http://api.jquery.com/css/
- **.css()** as getter
  - Gets a specified CSS property from first element

```
$('h1').css('backgroundColor');
```

```
var properties = $('h1').css(['backgroundColor', 'color']);
console.log('Text color: ' + properties['backgroundColor']);
```

CSS Manipulation [cont.]

- http://api.jquery.com/css/
- **.css()** as setter
  - Sets a CSS properties on selected elements

```
$('<img>', {
  src: 'images/logo.tiny.png',
  alt: 'Company logo',
  title: 'The Company Logo'
}).css({
  cursor: 'pointer',
  border: '1px solid black',
  backgroundColor: 'white'
});
```

Appending Elements

- Add it to the DOM **inside** an element
  - http://api.jquery.com/category/manipulation/dom-insertion-inside/

- **createdElement.appendTo('body')**
  - Adds whatever was created before and attaches it to the end of the body element in the document
  - It will attach as the last element within the body element
  - http://jsfiddle.net/kamrenz/qc38A/2/

Appending Elements [cont.]

- Add it to the DOM **inside** an element
  - http://api.jquery.com/category/manipulation/dom-insertion-inside/
- **createdElement.appendTo('.spots')**
  - Adds whatever was created before and attaches it as the last element of **all** elements with the class spots
  - http://jsfiddle.net/kamrenz/F7FBM/2/

Appending Elements [cont.]

- Add it to the DOM **inside** an element
  - http://api.jquery.com/category/manipulation/dom-insertion-inside/

- **$('body').append(createdElement)**
  - Adds whatever was created before and attaches it to the end of the body element in the document
  - It will attach as the last element within the body element
  - http://jsfiddle.net/kamrenz/S3cng/2/

Inserting Elements

- Add it to the DOM **outside** an element
  - http://api.jquery.com/category/manipulation/dom-insertion-outside/

- **createdElement.insertAfter('h1')**
  - Inserts whatever was created before and attaches it after the h1 title element in the document
  - The placement element is taken as the function parameter
  - http://jsfiddle.net/kamrenz/AMBFN/2/

Inserting Elements [cont.]

- Add it to the DOM **outside** an element
  - http://api.jquery.com/category/manipulation/dom-insertion-outside/

- **$('h1').after(createdElement)**
  - Same functionality as .**insertAfter**()
  - The content is taken as the function parameter
  - http://jsfiddle.net/kamrenz/69CV7/1/

Removing Elements

- Removing elements from the DOM
  - http://api.jquery.com/category/manipulation/dom-removal/

- **remove()** removes the element itself and everything inside of it
  - **$('h1').remove()**
    - Removes the h1 element from the document
    - http://jsfiddle.net/kamrenz/B2Jev/
  - **$('a').remove('.external')**
    - Removes the link elements with class external
    - http://jsfiddle.net/kamrenz/SqV36/

Removing Elements [cont.]

- Removing elements from the DOM
  - http://api.jquery.com/category/manipulation/dom-removal/

- **empty()** removes the child nodes and not the element itself
  - **$('section').empty()**
    - Removes the child nodes and not the element itself
    - It accepts no arguments
    - http://jsfiddle.net/kamrenz/24CN7/

Replacing Elements

- Removing elements from the DOM
  - http://api.jquery.com/category/manipulation/dom-replacement/

- **replaceWith()** replaces the content with the content supplied
  - **$('section').replaceWith(domFragment)**
    - http://api.jquery.com/replaceWith/
    - Takes whatever DOM elements we create and replaces them with the element selected

Element HTML

- Interacting with the HTML

- **.html()**
  - Gets the html() from the element
- **.html('<span>hello</span>')**
  - Sets the html for the element to **hello**

- Example:
  - http://jsfiddle.net/kamrenz/h5EUy/4/

Element Test

- Interacting with the Text

- **.text()**
  - Gets the html() from the element
- **.text('hello')**
  - Sets the html for the element to **hello**

- Example:
  - http://jsfiddle.net/kamrenz/yspye4qm/

Elements [cont.]

- Interacting with Element Attributes

- **.attr('href')**
  - Gets the url from the element
- **.attr('href', 'www.developintelligence.com')**
  - Sets the url for the element

- Example:
  - http://jsfiddle.net/kamrenz/k4HDA/

Elements [cont.]

- Interacting with Element Properties
  - Useful on form inputs for disabled, checked, selected
  - Useful on DOM elements for tagName, nodeName, nodeType
- **.prop('checked')**
  - Gets the checked property from the element
- **.prop('checked', true)**
  - Sets the checked property to true
- Example:
  - http://jsfiddle.net/kamrenz/uxhvLu2t/2/

Elements [cont.]

- Properties vs. Attributes
  - Attributes are additives to the DOM element
    - <div id="box">
  - Properties are utilized via JavaScript on the element itself
    - div.id

- Let's look at **examples/prop-attr**

Clone Elements

- **clone()**
  - http://api.jquery.com/category/manipulation/copying/
  - Creates a deep copy of the element (i.e. a duplicate)
  - Without the use of cloning elements simply get passed around or moved (i.e. no copying occurs)
  - http://jsfiddle.net/kamrenz/BLLSQ/1/

```
//Going to move the message element to the header
$(".message").appendTo("#header");

//Going to copy the message element and append the copy
//to the header element
$(".message").clone().appendTo("#header");
```

Elements

- **is()**
  - Checks the elements agains the parameter
  - If at least one "is" then returns true
- **:hidden** … jQuery extension
  - Elements need to take up no space in the document
  - display of none or has a width & height of 0
  - form element of type="hidden"
- **:visible** … jQuery extension
  - opacity of 0 and visibility:hidden are still visible

```
console.log("hidden? " + $("#hidden").is(":hidden"));
console.log("visible? " + $("#visible").is(":visible"));
```

  - http://jsfiddle.net/kamrenz/ged6t/1/

Forms

- **val()**
  - Returns the value of the form input
- **:checked**
  - Used for interacting with check boxes and radio buttons
- **:selected** … jQuery extension
  - Used for interacting with drop-down inputs

```
console.log($("input[type='radio']:checked").val());
console.log("brightness: " + $("select option:selected").val());
```

  - http://jsfiddle.net/kamrenz/NPbcv/1/

Data

- How can we interact with data-* attributes?

- **.attr()** allows us to read / write to the properties
  - Writing to the attribute will change the HTML attribute

HTML

```
<p>Welcome back <span id="user" data-eid="E012345">Bill</
span>
```

JavaScript

```
//Read the data-eid attribute
console.log($('#user').attr('data-eid'));
//Write the data-eid attribute
$('#user').attr('data-eid', ‘E011111');
console.log($('#user').attr('data-eid'));
```

Data [cont.]

- jQuery allows us to add extra data to the wrapped element
  - Similar to the HTML5 data-* attributes

- **.data()** allows us to read/write to jQuery elements data
  - This does not change the HTML attribute
  - It changes cached data on the jQuery element
  - Don’t use the data- to interact with the data
    - Not “**data-employee-id**”, just “**employee-id**”

Data [cont.]

- .data() adds the data to jQuery cache

HTML

```
<p>Welcome back <span id="user" data-eid="E012345">Bill</
span>
```

JavaScript

```
//Read the data-eid attribute
console.log($(‘#user').data('eid'));
//Write the data… we won’t be setting the data-eid
//  Data will be stored in the jQuery cache
$('#user').data('eid', ‘E022222');
console.log($(‘#user').data('eid'));
```

- Let’s see the jQuery data vs. data- in action
  - http://jsfiddle.net/kamrenz/j58z5owj/4/

jQuery Basics

Utility Methods

Utility Methods

- jQuery can be utilized as a namespace?
- This namespace gives access to utility methods

```
var trimmed = $.trim(someString);
```

```
var trimmed = jQuery.trim(someString);
```

jQuery Type Checking

- There could be multiple global objects!
  - Different windows have different globals
  - **instanceof** only works if the types are created in the same window
  - We wouldn't want arrays sharing the same resources across windows (i.e. Array.prototype)
- Safest way to check for Arrays
  - jQuery.isArray(aValue);

jQuery Basics

Unobtrusive JavaScript

Unobtrusive JavaScript

- Why have unobtrusive JavaScript?

- JavaScript is for web page control / behavior
- Browser inconsistencies need to be managed
- The global namespace must not be polluted

Unobtrusive JavaScript [cont.]

- Why have unobtrusive JavaScript?

POJS

```
window.onload = function() {
  // do stuff here
};
```

- Allows for code to be executed after entire document is loaded
- Problematic?

Unobtrusive JavaScript [cont.]

- Why have unobtrusive JavaScript?

With jQuery

```
$(document).ready(function() {
  $("div.notLongForThisWorld").hide();
});
```

jQuery shorthand

```
$(function() {
  $("div.notLongForThisWorld").hide();
});
```

- Waits only until the document structure is parsed

jQuery Basics

Looping

Looping [cont.]

- What does typical iteration look like?

- With jQuery: $.each(container, callback)
  - jQuery's main way of iteration
  - container
  - callback: (Function)
    - Invokes the callback for each iteration
    - First parameter is the index
    - Second parameter is the iteration

Looping [cont.]

- What does typical iteration look like?

- With jQuery: $.each(container, callback)
  - Works on arrays and objects

```
var arrayToIterate = ['first', 'second', 'third'];
$.each(arrayToIterate, function(index, value) {
  console.log(index + ' ' + value);
});
```

```
var objectToIterate = {first:1, second:2, third:3};
$.each(objectToIterate, function(index, value) {
  console.log(index + ' ' + value);
});
```

Sublime & JSHint

- JSHint Gutter:
  - https://packagecontrol.io/packages/JSHint%20Gutter
  - https://github.com/victorporof/Sublime-JSHint

BROWSE

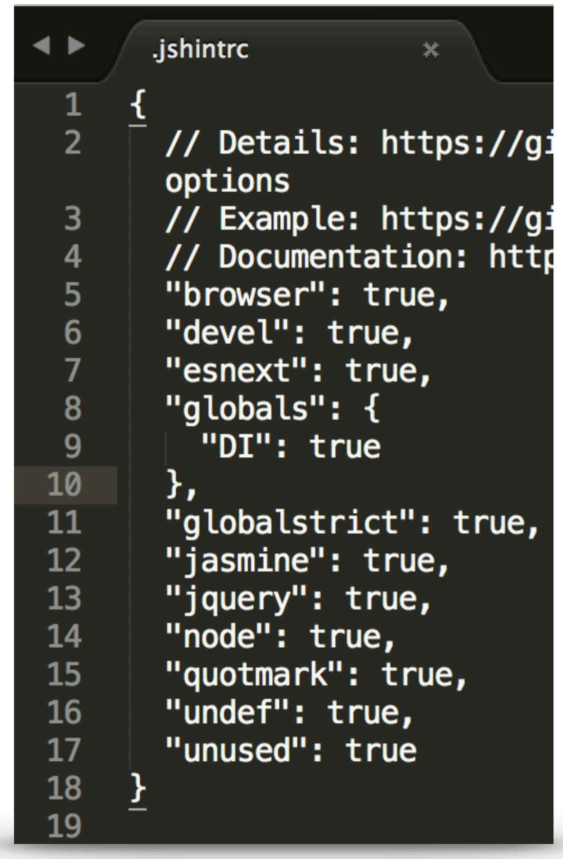
JSHint Gutter

by victorporof ST2/ST3

  - In the plugin options set **“lint_on_save”: true**

.jshintrc

- We can configure via its JavaScript Object Literal config file … **.jshintrc**
  - JShint options: http://www.jshint.com/docs/options/



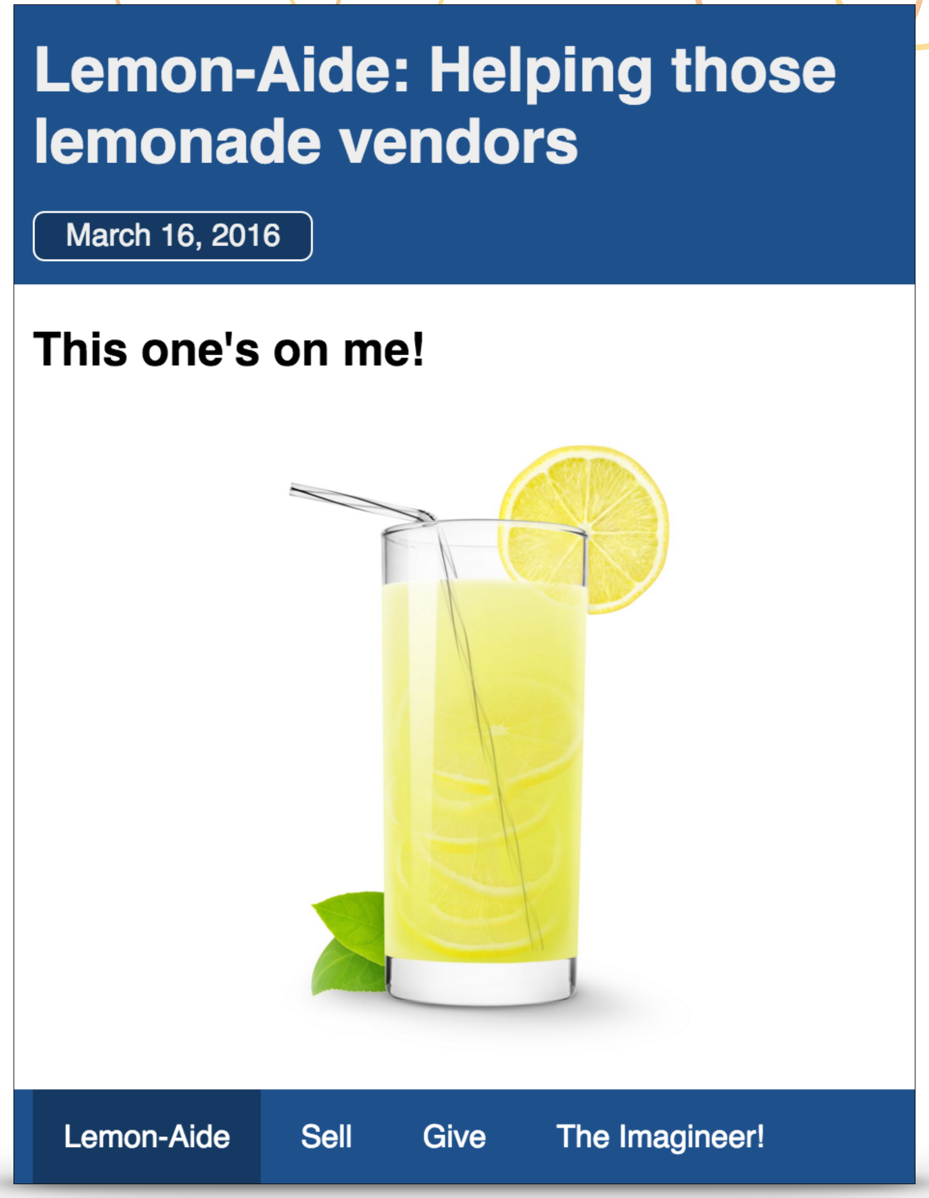
```
{
  // Details: https://gi
  options
  // Example: https://gi
  // Documentation: http
  "browser": true,
  "devel": true,
  "esnext": true,
  "globals": {
    "DI": true
  },
  "globalstrict": true,
  "jasmine": true,
  "jquery": true,
  "node": true,
  "quotmark": true,
  "undef": true,
  "unused": true
}
```

Lab 4

- Install JSHint Gutter package into Sublime Text

- Modify the time element with jQuery
  - Use jQuery to delete the <time> element on the page
  - Create a brand new <time> element
    - Use today's date
      - Don't calculate it… simply a string
    - Specify the element date as a readable string (i.e. January 1, 1970)
    - Specify the attribute date as a machine readable string (i.e. 1970-01-01)
  - Place the created <time> element within the header

Lab 4



Lemon-Aide: Helping those lemonade vendors

March 16, 2016

This one's on me!

End