jQuery

Interaction for the Masses

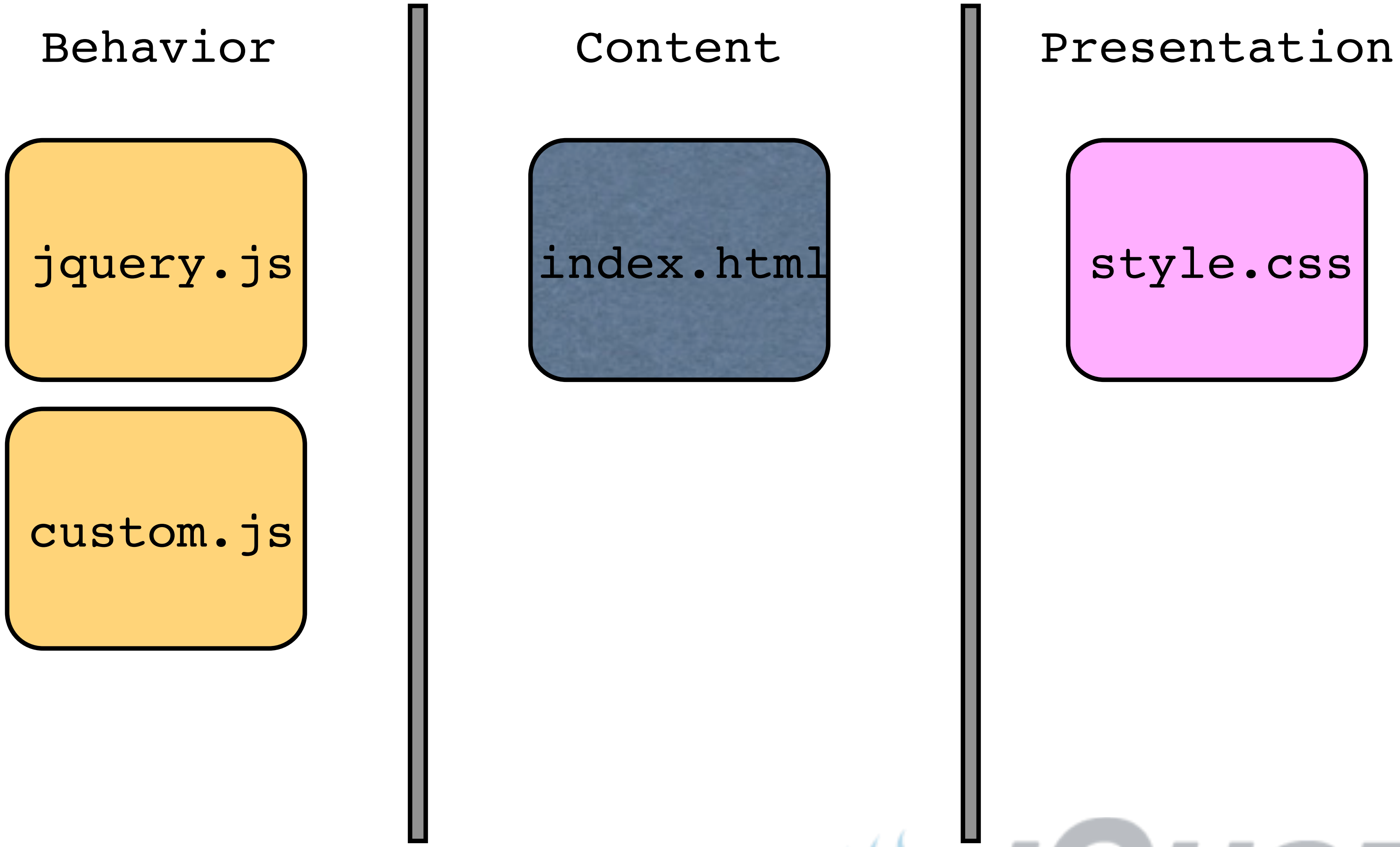
Outline

- Philosophy of jQuery and API Walkthrough
- Dev Tools
- Bare-bones JavaScript
- jQuery Selectors and Traversing

Just a few of jQuery's Benefits

- Lets you move quickly from beginner to advanced
- Improves developer efficiency
- Excellent documentation // pats self on back
- Unobtrusive from the ground up
- Reduces browser inconsistencies
- At its core, a simple concept

Unobtrusive



Reduces browser inconsistencies

- Example:
  Get the height of the viewport…

DOM Scripting

```
var x,y;
if (self.innerHeight) { // all except Explorer
  x = self.innerWidth;
  y = self.innerHeight;
}
else if (document.documentElement &&
  document.documentElement.clientHeight) {
   // Explorer 6 Strict Mode
  x = document.documentElement.clientWidth;
  y = document.documentElement.clientHeight;
}
else if (document.body) { // other Explorers
  x = document.body.clientWidth;
  y = document.body.clientHeight;
}
```

jQuery

```
var x = $(window).width();
var y = $(window).height();
```

Documentation & Support

- **API**: api.jquery.com
- **Forum**: forum.jquery.com
- **IRC**: irc.freenode.net, #jquery
- **Coming Soon**: learn.jquery.com

Simple Concept

- Find something
- Do something

Find Something

"Select" elements in the document

jQuery

write less, do more.

$

$( )

```
$('div')
```

```
$('#id')
```

Do Something

Do Something

1. Let elements "listen" for something to happen …
    - the document is ready
    - user does something
    - another "listener" acts
    - a certain amount of time elapses

Do Something

2. … and then do something
    a. Manipulate elements
    b. Animate elements
    c. Communicate with the server

Dev Tools

jQuery
write less, do more.

Chrome Developer Tools

- In many ways, leapfrogging Firebug
- Live debugging, code changing
- Lots of "hidden" goodies
- http://code.google.com/chrome/devtools/
- Paul Irish screencast: http://youtu.be/nOEw9iiopwI

Bare-bones JavaScript

jQuery

write less, do more.

The Basics

In JavaScript, you can work with the following things:

- **Strings**: textual content. wrapped in quotation marks (single or double).
  - 'hello, my name is Karl'
  - "hello, my name is Karl"
- **Numbers**: integer (2) or floating point (2.4) or octal (012) or hexadecimal (0xff) or exponent literal (1e+2)
- **Booleans**: true or false

The Basics

In JavaScript, you can work with the following things:

- **Arrays**: simple lists. *indexed* starting with 0
    - ['Karl', 'Sara', 'Ben', 'Lucia']
    - ['Karl', 2, 55]
    - [['Karl', 'Sara'], ['Ben', 'Lucia']]
- **Objects**: lists of key, value pairs
    - {firstName: 'Karl', lastName: 'Swedberg'}
    - {parents: ['Karl', 'Sara'], kids: ['Ben', 'Lucia']}

Variables

- Always declare your variables!
- If you don't, they will be placed in the **global scope** (more about that later).
    - **bad:** myName = 'Karl';
    - **good:** **var** myName = 'Karl';
    - **still good:** **var** myName = 'Karl';
      // more stuff
      myName = 'Joe';

Conditionals and Operators

- conditionals:
  - if, else
  - switch
- operators:
  - +, -, *, %, ++, --
  - >, <, ==, !=, >=, <=, ===, !==
  - !, &&, ||

Loops

- Loops *iterate* through a list of some kind.
- A common pattern in JavaScript is to build a list, or collection, and then do something with each item in that list.

Loops

- The two most common loops...
  - **for** loops — for general-purpose iteration. Used with arrays or array-like objects)
  - **for-in** loops — used with arrays or objects (but don't use with arrays)
- The other two are...
  - **while** loops
  - **do-while** loops

for Loops

- three statements and a code block

1. initial value
2. condition
3. increment

```
for (initial value; condition; increment) {
  // code block
}
```

for Loops

```
for (var i = 0; i < 3; i++) {
   alert(i+1);
}
```

This is your variable, so it can be anything! (but developers often use "i")

for Loops

```
var divs = document.getElementsByTagName('div');

for (var i = 0; i < divs.length; i++) {
  // do something with each div individually
    divs[i].style.color = 'red';
}
```

for Loops

```
var divs = document.getElementsByTagName('div');

// can store it directly in the initializer
for (var i=0, divCount=divs.length; i < divCount; i++) {
  // do something with each div individually
  divs[i].style.color = 'red';
}
```

for-in Loops

```
var family = {
  dad: 'Karl',
  mom: 'Sara',
  son: 'Benjamin',
  daughter: 'Lucia'
}

for (var person in family) {
  alert('The ' + person + ' is ' + family[person]);
}
```

This is your variable, so it can be anything!

while and do-while

```
var i = 1;
while (i < 4) {
  alert(i);
  i++;
}

var j = 1;
// code block always executed at least once
do {
  alert(j);
  j++;
} while (j < 4)
```

Functions

The Basics: Functions

In JavaScript, you can also work with **functions:**

- Functions allow you to **define** a block of code, name that block, and then **call** it later as many times as you want.
  - **function** myFunction( ) { /* code goes here */ } // defining
  - **myFunction**( ) // calling the function *myFunction*
- You can define functions with **parameters**
  - function myFunction(**param1**, **param2**) { /* code goes here */ }
- You can call functions with **arguments**:
  - myFunction(**'one'**, **'two'**)

Functions

```
  // define a function
function doSomething() {
  alert('I am something');
}



// call the function
doSomething();
```

Functions

```
  // define a function
function sumThing(a, b) {
  return a + b;
}



// call the function
alert( sumThing(1, 2) );
```

Functions

```
   // define a function
function sumThing(a, b) {
  return a + b;
}

var mySum = sumThing(1, 2);

// call the function
alert( mySum );
```

The `arguments` Object

- Every function has an `arguments` object
  - a collection of the arguments passed to the function when it is called
  - an "array-like object" in that it is indexed and has a **length** property but can't attach array methods to it
  - can be looped through
  - allows for variable number of arguments

Functions

```
// call the function
function logThing() {
  for (var i=0; i < arguments.length; i++) {
    console.log(arguments[i]);
  }
}

// call the function
logThing(1, 2, 'three');

/* prints to the console:
 >> 1
 >> 2
 >> three
*/
```

Exercise

Convert the sumThing function to allow for variable number of arguments.

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

Use a *for* loop to loop through the *arguments* object, adding to a "sum" variable with each iteration.
After the loop, return *sum*.

(Simple) Solution

```
  // define a function
function sumThing() {
  var sum = 0,
      countArgs = arguments.length;

  for (var i = 0; i < countArgs; i++) {
    sum += arguments[i];
  }
  return sum;
}
// call the function
console.log( sumThing(1, 2, 4 ) );
```

Returning Functions

- Functions can return other functions

```
function multiple(n) {
  function f(x) {
    return x * n;
  }
  return f;
}
var triple = multiple(3);
var quadruple = multiple(4);

console.log( triple(5) ); // 15
console.log( quadruple(5) ); // 20
console.log( multiple(4)(5) ); // 20
```

Named vs. Anonymous Functions

- Named:
  - function **foo**() { } // function **declaration**
  - var foo = function **foo**() { }; // function **expression**
- Anonymous:
  - var foo = function() { }; // function **expression**

Anonymous Functions

- Prevalent in jQuery
- Good for creating closures
- Used as "callback" functions
- Can be used as object properties (methods)

let’s take a look ...

Anonymous Functions

- Prevalent in jQuery

```
$(document).ready(function() {

});
```

Anonymous Functions

- Good for creating closures

```
function() {
  // variables are defined within this scope
  // avoid name collisions
}
```

Anonymous Functions

- Good for creating closures
- Can be *defined* and then immediately *invoked:* "immediately invoked function expression," (a.k.a. **IIFE**; pronounced "**iffy**")

```
(function() {
  // variables are defined within this scope
  // avoid name collisions
})();
```

Anonymous Functions

- Good for creating closures
- Used by plugins to keep jQuery safe.

```
(function($) { // "$" is the function's param

})(jQuery); // function is called with "jQuery"
```

Anonymous Functions

- Used as "callback" functions

```
$('p').slideDown('slow', function() {
   // code in here is not executed
   // until after the slideDown is finished
   // jQuery calls the code in here when effect ends
});
```

Objects

Objects

In JavaScript, everything is an object. Well, almost everything.

- Objects are objects : **{ }**
- Arrays are objects : **[ ]**
- even Functions are objects : **function( ) { }**
- jQuery is an object
- Numbers, strings, and booleans (true/false) are primitive data types, but they have object wrappers.

Global Object

In the browser environment, the global object is **window**
It collects all functions and variables that are global in scope.
Usually implied.

Comes with some useful properties and methods:

- `location`
- `parseInt(); parseFloat()`
- `isNaN()`
- `encodeURI(); decodeURI()`
- `setTimeout(); clearTimeout()`
- `setInterval(); clearInterval()`

Date Object

```
var now = new Date(); // current date and time
var then = new Date('08/12/2000 14:00');

console.log( then.getTime() ); // 966103200000

console.log( then.toString() );
  // Sat Aug 12 2000 14:00:00 GMT-0400 (EDT)

console.log( then.getMonth() ); // 7 !!!!
```

RegExp Object
Regular Expression

- Object constructor
  - `var re = new RegExp('hello');`
- Regular expression literal
  - `var re = /hello/;`

Creating a RegExp

- Object constructor
  - `var re = new RegExp('hello');`
- Regular expression literal
  - `var re = /hello/;`

Using a RegExp

```
   var text = 'The quick brown fox';

var re = new RegExp('quick');
console.log( re.test(text) ); // true

console.log( /brown/.test(text) ); // true
console.log( /red/.test(text) ); // false
```

RegExp Syntax

- Most characters (incl. all alphanumerics) represent themselves
- Special characters can be escaped with a backslash (\)

Character Classes

- /t.p/ matches 'tap' and 'tip' and 'top'
- /t[ai]p/ matches 'tap' and 'tip', not 'top'
- /t[a-k]p/ matches 'tap' and 'tip', not 'top'
- /t[^m-z]p/ matches 'tap' and 'tip', not 'top'

Repetition

- /frog*/ matches 'fro', 'frog', 'frogg', ...
- /frog+/ matches 'frog', 'frogg', ...
- /frog?/ matches 'fro' or 'frog'
- /frog{2,3}/ matches 'frogg' or 'froggg'

Grouping

- Grouping
  - /(frog)*/ matches "frog" or "frogfrog"
- Alternation
  - /th(is|at)/ matches "this" and "that"

Anchor Points

- ^ matches the beginning of a string
- $ matches the end of a string
- \b matches word boundaries

Exercises

Write a regular expression that matches any word that starts with a vowel.

Write a regular expression that matches any HTML tag.

String RegExp Methods

`str.search(re)`

- `str.match(re)`
- `str.replace(re, replacement)`
- `str.split(re)`

String Replacement

```
    var str =
   'The quick brown fox jumps over the lazy dog.';

console.log(str.replace(/[aeiou]/, '*'));
// Th* quick brown fox jumps over the lazy dog.
```

RegExp Flags

- Placed after closing / character
- Global (g): find as many as possible
- Case insensitive (i)
- Multiline (m): ^ and $ work with newlines

String Replacement

```
    var str =
   'The quick brown fox jumps over the lazy dog.';

console.log(str.replace(/[aeiou]/g, '*'));
// Th* q**ck br*wn f*x j*mps *v*r th* l*zy d*g.

console.log(str.replace(/the/gi, 'a'));
// a quick brown fox jumps over a lazy dog.
```

Replacement Functions

```
   var str = 'Kill 5+9 birds with 2+5 stones.';

function add(match, first, second) {
  return parseInt(first, 10) + parseInt(second, 10);
}
str = str.replace(/([0-9]+)\+([0-9]+)/g, add);
console.log(str);
// Kill 14 birds with 7 stones.
```

Math Object

- Not a constructor, a singleton
- Gathers useful methods and properties

```
Math.PI
Math.abs(), Math.sin(), Math.pow(), Math.random(),
Math.max(), Math.min()
Math.round(), Math.floor(), Math.ceil()
```

CSS Tip

- Object literal notation looks a lot like CSS style rule notation!

CSS:

```
h3 {
  font-size: 1.2em;
  line-height: 1;
}
```

JS:

```
var h3 = {
  fontSize: '1.2em',
  'line-height': 1
};
```

Object Literals

- **person** is the **object**
- **firstName** and **lastName** are **properties**
- **hello** is a **method** (a property that is a function)

```
var person = {
  firstName: 'Karl',
  lastName: 'Swedberg',
  hello: function() {
    return 'Hello, my name is ' +
          this.firstName + ' ' + this.lastName;
  }
};
```

Object Literals

- **interests** is a **property** *and* an **object**

```
var person = {
  firstName: 'Karl',
  lastName: 'Swedberg',
  hello: function() {
    return 'Hello, my name is ' +
          this.firstName + ' ' + this.lastName;
  },
  interests: {
    athletic: ['racquetball', 'karate', 'running'],
    musical:  ['rock', 'folk', 'jazz', 'classical']
  }
};
```

Object Literals

```
var person = {
  firstName: 'Karl',
  lastName: 'Swedberg',
  hello: function() {
    return 'Hello, my name is ' +
          this.firstName + ' ' + this.lastName;
  } // ⬅ notice, no comma here!
};
```

Object Literals

“dot” notation

```
var person = {
  firstName: 'Karl',
  lastName: 'Swedberg',
  hello: function() {
    return 'Hello, my name is ' +
          this.firstName + ' ' + this.lastName;
  }
};

// "dot" notation
person.firstName; // 'Karl'
person.lastName; // 'Swedberg'
person.hello() // 'Hello, my name is Karl Swedberg'
```

Object Literals

array notation

```
var person = {
  firstName: 'Karl',
  lastName: 'Swedberg',
  hello: function() {
    return 'Hello, my name is ' +
          this.firstName + ' ' + this.lastName;
  }
};

// array notation
person['firstName']; // 'Karl'
person['lastName']; // 'Swedberg'
person['hello']() // 'Hello, my name is Karl Swedberg'
```

Object Literals

```
var person = {
  firstName: 'Karl',
  lastName: 'Swedberg',
  hello: function() {
    return 'Hello, my name is ' +
          this.firstName + ' ' + this.lastName;
  },
  interests: {
    athletic: ['racquetball', 'karate', 'running'],
    musical:  ['rock', 'folk', 'jazz', 'classical']
  }
};
// person['interests']['musical'][1] == ??
```

Object Literals

```
var person = {
  firstName: 'Karl',
  lastName: 'Swedberg',
  hello: function() {
    return 'Hello, my name is ' +
           this.firstName + ' ' + this.lastName;
  }
};
person.firstName = 'Karl';

var prop = 'firstName';
person[ prop ]; // 'Karl'

prop = 'lastName';
person[ prop ]; // 'Swedberg'
```

Object Literals

```
var blah;
var person = {
  firstName: 'Karl',
  lastName: 'Swedberg',
  hello: function() {
    return 'Hello, my name is ' +
          this.firstName + ' ' + this.lastName;
  }
};

for (var el in person) {
  blah = typeof person[el] == 'function' ?
          person[el]() :
          person[el];
  console.log( blah );
}
```

Object Literals

- Great as function arguments
- single argument allows flexibility when calling the function

```
doSomething({
  speed: 'fast',
  height: 500,
  width: 200,
  somethingElse: 'yes'
});

doSomething({width: 300});
```

JSON

JavaScript Object Notation

- a *data interchange* format. In other words, a format for passing data back and forth
- "discovered" and popularized by Douglas Crockford
- a *subset* of JavaScript Object Literal Notation
  - a tree-like structure of object(s) and/or array(s)
  - no functions
  - all strings, including object keys, take double quotes

JSON

```
{
  "firstName": "Karl",
  "lastName": "Swedberg",
  "age": 24,
  "interests": {
    "athletic": [
      "racquetball",
      "karate"
    ]
  }
}
```

JSON

```
{"firstName":"Karl","lastName":"Swedberg","age":
24,"interests":{"athletic":["racquetball","karate"]}}
```

Referencing Scripts in the HTML

browser slides

Selectors & Traversal

Friday, November 4, 11

At the heart of jQuery...

- **Find something**
- Do something

CSS Selectors

- element {}
- #id {}
- .class {}
- **selector1, selector2** {}
- ancestor **descendant** {}
- parent > **child** {}
- :nth-child() {}

CSS Selectors

(jQuery Equivalents)

- $('element')
- $('#id')
- $('.class')
- $('**selector1**, **selector2**')
- $('ancestor **descendant**')
- $('parent > **child**')
- $(':nth-child(n)')

CSS Selectors

(jQuery Equivalents)

- $('element')
- $('#id')
- $('.class')
- $('**selector1**, **selector2**')
- $('ancestor **descendant**')
- $('parent > **child**')
- $(':nth-child(n)')
- *and others …*
- $('prev + **selector**')
- $('prevAll ~ **selector**')
- $(':nth-child(an+b')
- $(':not(selector)')
- $(':checked')
- $(':disabled')

CSS Attribute Selectors

$('input[name=firstname\\[\\]]')	
$('[**title**]')	has the attribute
$('[attr**=**"val"]')	attr equals val
$('[attr**!=**"val"]')	attr does not equal val
$('[attr**~=**"val"]')	attr has val as one of space-sep. vals
$('[attr**^=**"val"]')	attr begins with val
$('[attr**$=**"val"]')	attr ends with val
$('[attr***=**"val"]')	attr has val anywhere within

Custom Form Selectors

• $('div.myclass :checkbox')	
• $(':input')	<input>, <textarea>, <select>, <button>
• $(':text')	<input type="text">
• $(':radio')	<input type="radio">
• $(':button')	<input type="button">, <button>
• $(':selected')	<option selected="selected">
• etc.	

Custom Misc. Selectors

- $(':animated')
- $(':**has**(descendant)')
- $(':eq(n)')
- $(':lt(n)')
- $(':gt(n)')
- $(':even')
- $(':odd')
- $(':visible')
- $(':hidden')
- $(':header')
- $(':contains(string)')

Selectors

- List of all selectors on the jQuery API site
- **http://api.jquery.com/category/selectors**

Traversal Methods

- Move Up
- Move Sideways
- Move Down
- Filter
- Context
- Check

Move Up

- parent() : up one level **$('li.bottom').parent();**
- parent**s**() : up multiple levels **$('span').parents('ul');**
- parentsUntil() : possibly multiple **$('span').parentsUntil('ul');**

```
<ul>
  <li>level 1
    <ul class="foo">
      <li>level 2
        <ul>
          <li class="bottom"><span>level</span> 3</li>
        </ul>
      </li>
    </ul>
  </li>
</ul>
```

Move Up

- .closest(selector) : up 0 or more levels
  - **$('span').closest('ul');**
  - **$('.bottom').closest('li');**

```
<ul>
  <li>level 1
    <ul>
      <li>level 2
        <ul>
          <li class="bottom"><span>level</span> 3</li>
        </ul>
      </li>
    </ul>
  </li>
</ul>
```

Move Sideways

- .siblings()
- .next()
- .nextAll()
- .nextUntil()
- .prev()
- .prevAll()
- .prevUntil()

Move Down

- .children()
- .find()

Filter

- .filter(selector)
  - `.filter('.some-class')`
- .filter(function)
  -
    ```
    .filter(function() {
        return $(this).parents('li').length >= 2;
    });
    ```

Filter

- .not(selector)
  - `.not('.some-class')`
- .not(function)
  -
    ```
    .not(function() {
      return $(this).parents('li').length >= 2;
    });
    ```

Filter

- .slice()
  - `.slice(2)`
  - `.slice(-2)`
  - `.slice(3, 6)`
  - `.slice(2, -1)`
- .eq()
  - `.eq(2)`
  - `.eq(-2)`

Context

- $('selector', 'context')
  - Different from "or" selector – $('selector1, selector2')
  - Same as $('context').find('selector')
  - Not worth using; too confusing.
- .add()
- .andSelf()
- .end()

Check

- .hasClass(class)
- .is(selector)

** returns a boolean

Traversal Methods

- List of all traversal methods on the jQuery API site
- **http://api.jquery.com/category/traversing**

Chaining

- JavaScript has chaining built in.
  - 'swap this text'.replace(/w/, 'n').replace(/this/,'that');
  - '616-555-1212'.split('-').join('.');
- jQuery takes advantage of this concept by having almost all methods return the jQuery object.

Chaining

- Chain traversal methods together

```
$('a').parent('li').siblings().find('a')
```

Chaining

- Attach multiple behaviors.

```
$('a').removeClass('old').addClass('new');
```

Chaining

- DOM Traversal methods are different from other jQuery chaining methods!
    - New jQuery instance is created with each one.

```
$('a').addClass('foo').parent('li').removeClass('foo')
```

Chaining

- JavaScript ignores white space, so use it to your advantage.

```
var lis = $('.container li:first')
.addClass('first-li')
  .next()
  .addClass('second-li')
.end()
  .nextAll()
  .addClass('not-first-li')
.end(); // unnecessary; added for symmetry
```

write less, do more.

Looping

- Implicit Iteration
- Explicit Iteration (Looping)

```
$('li').removeClass('myclass');       //implicit

$('li').each(function(index) {       //explicit
  $(this).append( ' #' + (index+1) );
});
```

`this` Keyword

- Refers to the current object
- jQuery sets **this** to matched elements in the jQuery object.

```
$('li').each(function() {
  console.log( this ); // DOM element
  console.log( $(this) );
});
```

Tips

- Store selectors used more than once in variables
- Use length property to check existence
  - ...but often no need for the check

```
var $listItems = $('li');
var numItems = $listItems.length

//no need for length check
$listItems.addClass('pretty');

if (numItems) {
  // do something with other elements
}
```

Tips

- Concatenate to pass in a variable

```
$('#menu li').each(function(index) {
  $(this).click(function() {
    $('#footer li:eq(' + index + ')')
    .addClass('active');
  });
});
```

Tips

- Avoid jQuery's custom selectors when possible

```
// bad
$(':checkbox')
// better
$('input:checkbox')
// best
$('input[type="checkbox"]')
```

Tips

- Avoid jQuery's custom selectors when possible

```
// uncool
$('div:first')

// cool
$('div').first();
```

Tips

- Avoid jQuery's custom selectors when possible

```
// slower
$('li:eq(3)')
$('li:lt(3)')

// faster
$('li').eq(3)
$('li').slice(0, 3)
```