learning spike

JavaScript in the Web

Ryan Morris
@mrmorris

Introductions

- About me…
- About you…
  - What do you do?
  - What is your programming background?
    - Any front-end?
    - 😍, 😡 or 😭 JavaScript?
  - What do you hope to gain from this course?
- What is your current development environment and process?

How the class works

- Mixture of labs and lecture
- Informal
  - Stop me anytime
  - Discussion > Lecture
  - Outline is flexible
    - **There is too much to cover** so we'll adjust as needed
- You'll help define areas of focus
- Class assessment towards the end of the day

Get the most out of the class

- Ask **questions**!
- Do the **labs** (pair up if needed)
- Be **punctual**
- **Avoid distractions**
- Master your **google-fu**
- **Play along** in the console
- Don't be afraid to **break stuff**

What we'll cover

- JS, HTML, CSS refresher
- Working with the DOM
- Basic Event Handling
- jQuery; Events and the DOM
- jQuery UI introduction
- HTML Form Validation

I wasn't planning to cover

- Object Oriented JS
- Ajax (incl. in jQuery)
- Core JS concepts
- ES6 in depth
- All the HTML5 APIs

~Mostly for beginner/intermediates~

~You should be familiar with js, html, css~

~let's shape it~

Resources

- Reading List
  - https://javascript.info/intro
- Documentation
  - http://devdocs.io
  - https://developer.mozilla.org/en-US/docs/Web
  - http://kapeli.com/dash (Mac only)
  - Google it.
- Compatibility checks
  - http://caniuse.com

Lab prep - set up our toolkit

- A browser with dev tools
  - Preference for Chrome in class
  - Open your browser and hit F12 or alt/opt/⌥ -⌘-i
- Our web editor, jsfiddle
  - http://jsfiddle.net/
  - Sign up!

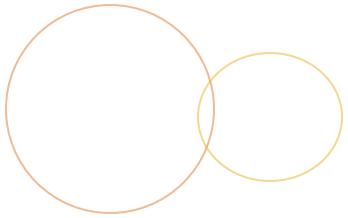
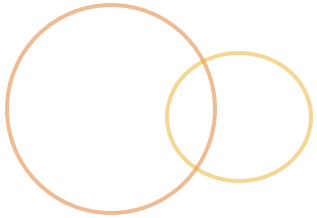
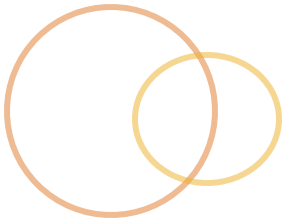
👆 Everyone OK with the above?

jsFiddle

- **Sandbox** to simulate a full “web page”
  - HTML + CSS + JavaScript -> Result!
  - The panels are just iframes
- You can specify **how to load your javascript**
  - noWrap, onLoad, onReady, etc…
  - This will affect if you can access globals from the console
- You can **include libraries** like jQuery
  - And attach external js files
- **When you fork my labs…**
  - You immediately own the fork (copy)
  - You should “update” regularly (save)
  - “Tidy” to clean up your code formatting
  - “Run” to execute all your changes and see the result

PDF for today

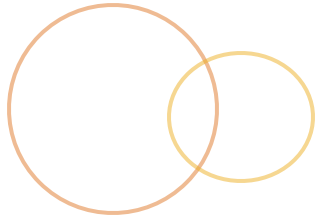
- In case you didn't quite catch something, or can't see the screen well:
  - Head to my repository
    - ###
  - Go to “**/docs**” folder
  - Or just download the whole repo…

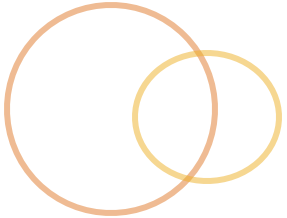




module

JAVASCRIPT IN THE WEB

The Web





- System of servers that support specially formatted documents
  - You visit a site, it provides a document
  - Document requests additional resources
- **Structured** documents with **information**
  - HTML
  - CSS
  - …eventually **behavior**
    - JavaScript

History of JavaScript

- “Make webpages alive”
- 1995 - Netscape wanted interactivity like HyperCard w/ Java in the name
- Designed & built in 10 days by Brendan Eich as "Mocha", released as “LiveScript"
  - Became “JavaScript” once name could be licensed from Sun
- Combines influences from:
  - Java, "Because people like it"
  - SmallTalk, prototypal

What is JavaScript?

- Standardized as **ECMAScript**
- **Interpreted**
- Case-sensitive C-style syntax
- Dynamically typed (with weak typing)
- Fully **dynamic**
- **Single-threaded** event loop
- Unicode (UTF-16, to be exact)
- **Prototype**-based (vs. class-based)
- Kind of weird but enjoyable

JavaScript Versions

- ES3/1.5
  - Released in 1999 – in all browsers by 2011
  - IE6-8
- **ES5/1.8**
  - Released in 2009
  - IE9+
  - http://kangax.github.io/compat-table/es5/
- **ES6 [EcmaScript 2015] mostly supported**
- ES7 [EcmaScript 2016] finalized, but weak support
- ES8 [EcmaScript 2017] finalized in June 2017
- ES.Next…

JavaScript and the Browser

- Why JavaScript
  - Runs everywhere (on the web)
  - Interactivity (events)
  - Simple to learn (though hard to master)
  - Load and manage data in the page
  - The web is the new application platform…
- How it fits
  - HTML for view data & ui structure
  - CSS for presentation
  - JavaScript for behavior

Where does JavaScript live?

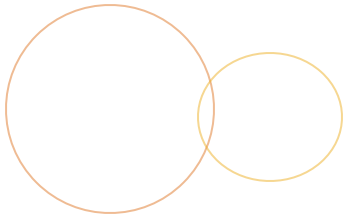
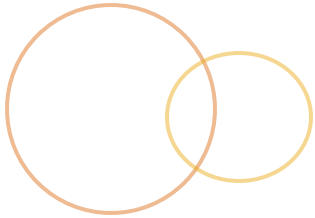
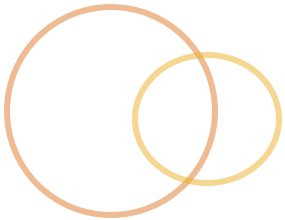
- Plain text files, not compiled
  - *Though this is changing*
- Browser (Built-in Engine)
  - Inline <script> blocks
  - Linked <script src="file.js"> files
- Server (Node)
  - One script file
  - Set of modules

Languages on JavaScript

- JS doesn’t always meet everyone’s needs
- Transpile (compile) down to plain JavaScript
  - CoffeeScript - syntax sugar
  - TypeScript - strict data typing
  - Dart - non-browser environments
  - ClosureCompiler
  - and more!

JavaScript and the future

- Language of the web…
  - Basic behavior
  - Complex behavior, data fetching/dynamic pages
  - Single Page Applications
  - Full frameworks
  - Servers
  - Command-line
  - Native applications
- As it matures…
  - Transpilers
  - Compilers
  - ~~Replacing~~ rendering and optimizing HTML

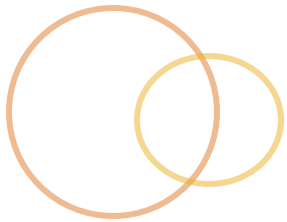
module

DEBUGGING

Debugging in the Browser

- Use your browser developer tools to access its JavaScript **console**
  - The browser's "`console`" is a REPL
- All major browsers are converging to the same API for console
- When in doubt, *echo*
  - print, log, echo, sprintf(), debug(), vardump()
    - `console.log();`
- You can code directly in the console
  - shift+enter to break to a new line
  - Usually shares the same global object/scope
- This is where we'll be working; follow along!

Browser's Console



- "`console`" object
  - `log`
  - `dir (lists all properties)`
  - `info`
  - `warn`
  - `error`
  - `table(object)`
  - `group(name); groupEnd();`
  - `assert(expr, message); // shows only if false`
- Typically on "`window`", though doesn't always exist
- Examples:
  - http://jsfiddle.net/mrmorris/fp9zgnh9/

Working with the dev tools

- Inspect and edit HTML
- Inspect and edit CSS
- View Network happenings, Ajax, etc
- Simulate mobile devices
- View memory usage/debug issues
- Inspect and debug JavaScript
- Manage/Delete cookies, sessions, data

Monitor scope and functions

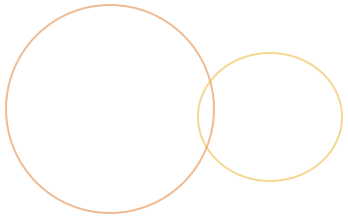
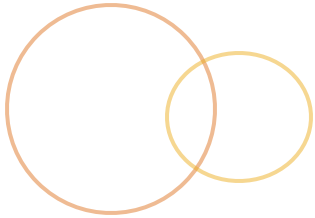
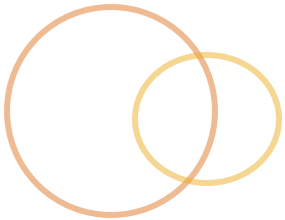
- Set breakpoints
  - Let you see scoped variables and context
  - Can set a conditional break-point
    - debugger;
- Monitor and log all calls to a function
  - monitor(functionName);
- Trigger debugging on a function’s call
  - debug(functionName);

Debugging - Events

- View Event Listeners registered in the page
  - Event Listeners Panel
  - getEventListeners(document)
- Monitor events on an element
  - monitorEvents(node, eventType);
  - unmonitorEvents(node);

Other debugging options

- Snippets
  - Sources > Snippets > New
- Persistence through WorkSpaces
  - https://developers.google.com/web/tools/setup/setup-workflow

refresher

JAVASCRIPT

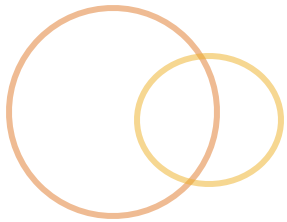
Wizard Check

- **Everyone OK with programming constructs?**
  - if-else
  - for loops
  - functions
- **Everyone OK with Core JS concepts?**
  - Coercion
  - Hoisting
  - Scope
  - Context
  - Functions
  - Prototype

Quick Refresher

- There are 5 primitive types (string, number, boolean, null, undefined) and then Objects
  - Functions are a callable Object
  - Objects are property names referencing data
  - Arrays are for sequential data
- Declare variables with "var"
  - **Block scope**
- Types are coerced
  - Including when a primitive is used like an object
- *Almost Everything* is an object, except the primitives
  - despite them having object counterparts

Exercise: Hoisting



- What will the output be?

```
function foo() {
 x = 42;
 var x;

 console.log(x); // ?
}
```

Solution: Hoisting

This…

```
function foo() {
 x = 42;
 var x;

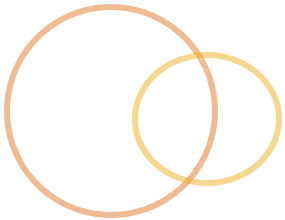
 console.log(x);
}
```

Becomes…

```
function foo() {
 var x;
 x = 42;

 console.log(x);
}
```

Exercise: Hoisting



- And this?

```
function foo() {
 console.log(x); // ?
 var x = 42;
}
```

Solution: Hoisting

This…

```
function foo() {
 console.log(x);
 var x = 42;
}
```

Becomes…

```
function foo(x) {
 var x;
 console.log(x);
 x = 42;
 return x;
}
```

Exercise: Variable scope

- What is the scope of **w**, **x**, **y** and **z**?

```
function foo(x) {
    var y = 0;
    if (x === 1) {
        var z = 1;
        w = x;
    }
}
```

Exercise: Callbacks & Async

- What does this code do?

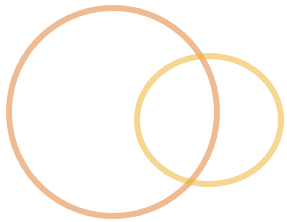
```
for (var i = 1; i <= 5; i++) {
    setTimeout(function() {
        console.log(i);
    }, i * 1000);
}

// logs out?
```

Solution: Callbacks & Async

```
for (var i = 1; i <= 5; i++) {
     (function(j){
         setTimeout(function() {
          console.log(i);
        }, i * 1000);
       })(i); // IIFE retain scope
} // outputs: 1, 2, 3, 4, 5
```

Exercise: Objects



- What is going on here?

```
var x = {
 color: “magenta”
}
x.name = “Bob”;
var y = {};

for (var prop in x) {
  if (x.hasOwnProperty(prop)) {
    y[prop] = x[prop];
  }
}
```

Exercise: Functions and Context

- What is going on here?

```
var x = {color: “magenta”}
var y = {color: “orange”}

var z = function() {
 console.log(“My color is”, this.color);
}

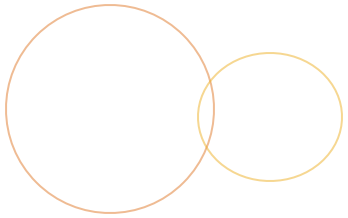
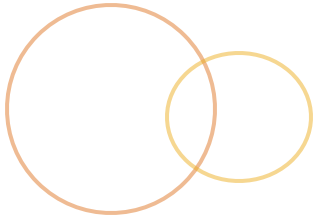
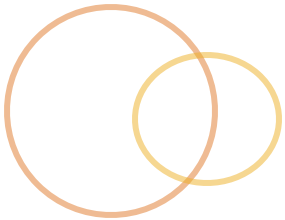
x.log = y.log = z;

x.log(); // ?
y.log(); // ?
z(); // ?… for bonus points
```

Exercise: Warm-up

- **Getting to know jsfiddle + console**
  - Take note of the iframe
- **JavaScript Basics**

  Try to complete as much as possible
  - Fork: http://jsfiddle.net/mrmorris/a5v1p5by/

refresher

HTML

Wizard check

- OK with basic HTML?
- Can write a page in full?
- Write a **<form>** and all necessary input controls?
- Understand the difference between ** ** and ****?
- Understand the usage of **attributes** on elements
- When to use **id** versus **class**?

HTML

- **H**yper**T**ext **M**arkup **L**anguage
- Browsers allow support for all sorts of errors – html is very error tolerant
- Structure of the UI and "view data"
- Tree of element nodes
- HTML5
  - Rich feature set
  - Semantic
  - Cross-device compatibility
  - Easier!

Anatomy of a page

```
<!doctype html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        …document info and includes…
    </head>
    <body>
        <h1>Hello World!</h1>
    </body>
</html>
```

Anatomy of an element

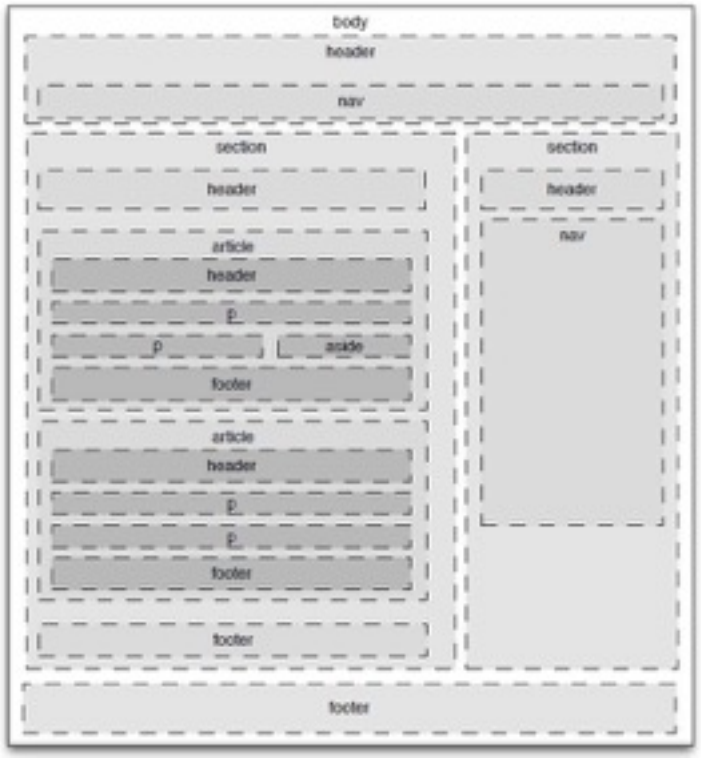
- <element attributeName="attributeValue">
  Content of element
  </element>
- Block vs inline
  -
  -
- Self closing elements
  - <input type="text" name="username" />

HTML Elements refresher

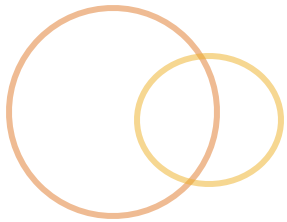
- **Structure**
  -
  -
  - <table>
    - <tr>, <td>, <thead>, <tbody>
  - <form>
    - <fieldset>, <label>, <input>, <select>, <textarea>
- **Content**
  - <h1> through <h6>
  - <p>
  - <ol> or <ul> (with<li>)
- **Text modifiers**
  - ,
- A list of elements:
  - https://developer.mozilla.org/en-US/docs/Web/HTML/Element

HTML5 Semantic Elements

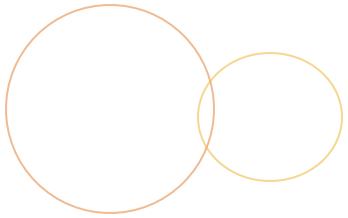
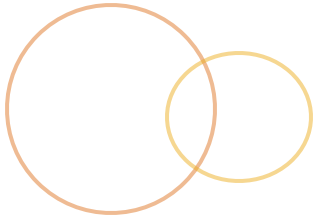
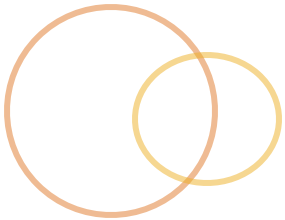
- Designed to degrade gracefully on non-HTML5 browsers
- Define an outline and semantic hints for a document
  - <header>
  - <footer>
  - <nav>
  - <main>
  - <section>
  - <article>
  - <aside>
  - <figure>, <figcaption>
  - <time>
  - <mark>
  - <details>, <summary>
- Examples
  - http://jsfiddle.net/mrmorris/L398weeh/



Before we move on…



- Any questions on HTML?

refresher

CSS

Wizard check

- OK with basic CSS selectors?
- Style a page in full?
- Select an element using CSS?
- Understand specificity?
- Got a few special pseudo-selectors under your belt?

Cascading Style Sheets

- Language for describing the look and formatting of the document
- Separates presentation from content

```
<!-- external resource -->
<link rel="stylesheet" type="text/css"
href="theme.css">


<!-- inline block -->
<style type=“text/css”>
     span {color: red;}
</style>


<!-- inline -->
<span style="color:red">RED</span>
```

Anatomy of a css declaration

```
selectors {
    /* declaration block */
    property: value;
    property: value;
    property: val1 val2 val3 val4;
}
div {
    color: #f90;
    border: 1px solid #000;
    padding: 10px;
    margin: 5px 10px 3px 2px;
}
```

CSS Selectors

- By element
  - h1 {color:#f90;} <h1></h1>
- By id
  - #header {}
- By class
  - .main {}
- By attribute
  - div[name="user"] {}
- By relationship to other elements
  - li:nth-child(2) {} <ul><li></li>***<li></li>***</ul>
  - p span {} ****
  - p > span {} ****

CSS Specificity

- Selectors apply styles based on its **specificity**
  - inline, id, pseudo-classes, attributes, class, type, universal
- **`!important`** allows you to override

html:

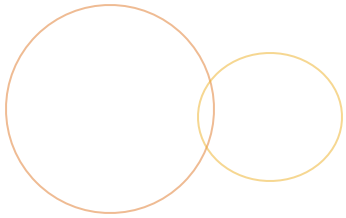
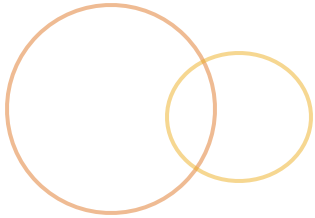
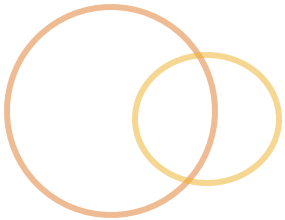
```
<div id="main" class="fancy">
     What color will I be?
</div>
```

css:

```
#main{
   color: orange;
}
.fancy{
   color: blue;
}
#main.fancy{
   color: red;
}
```

More CSS Selectors

Selector	Name/Description	Example
*	Star selector	*
#id	By id	#my-id
elementname	By element name	ul
.class	By class	.my-class-name
element subElement	By hierarchy	div p
element:pseudo	:hover, :checked, etc..	a:hover
element + nextEl	Adjacent sibling	div + div.special
element > child	Directly descendant	div > p
element ~ sibling	Any sibling	div ~ div
element[attribute]	Has an attribute	a[title]
element[attribute=val]	Attribute equals	input[type=checkbox]
element:nth-child(i)	Nth child	li:nth-child(5)
:first-child, :last-child	Order as child	ul li:first-child

mini-module

LOADING JS IN THE BROWSER

Block and inline

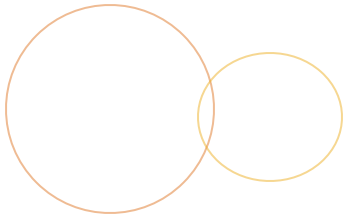
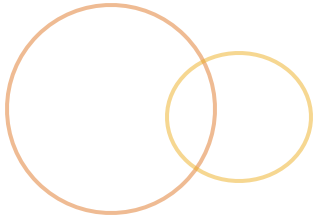
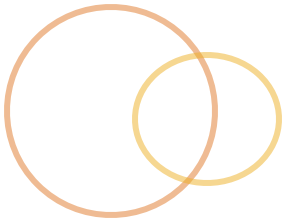
- Script blocks
  - <script>…</script>
- Script resources
  - <script src="filename.js"></script>

Scripts are blocking

- Browse loads resources top down
- Browser will wait on js+css downloads
- DOM is not parsed until scripts are loaded
- So…
  - Defer your <script> load
  - Include at the bottom of `</body>`
    - It won't block & the DOM is loaded
  - Or leverage the DOMContentLoaded (ie9+) events

Resource order matters

```
<html>
  <head>
      <!– meta –>
      <!– essential scripts? –>
      <!– essential css/above-the-fold –>
  </head>
 <body>
      <!– all your html –>
      <!– non-essential css –>
      <!– scripts –>
 </body>
</html>
```

module

THE DOM

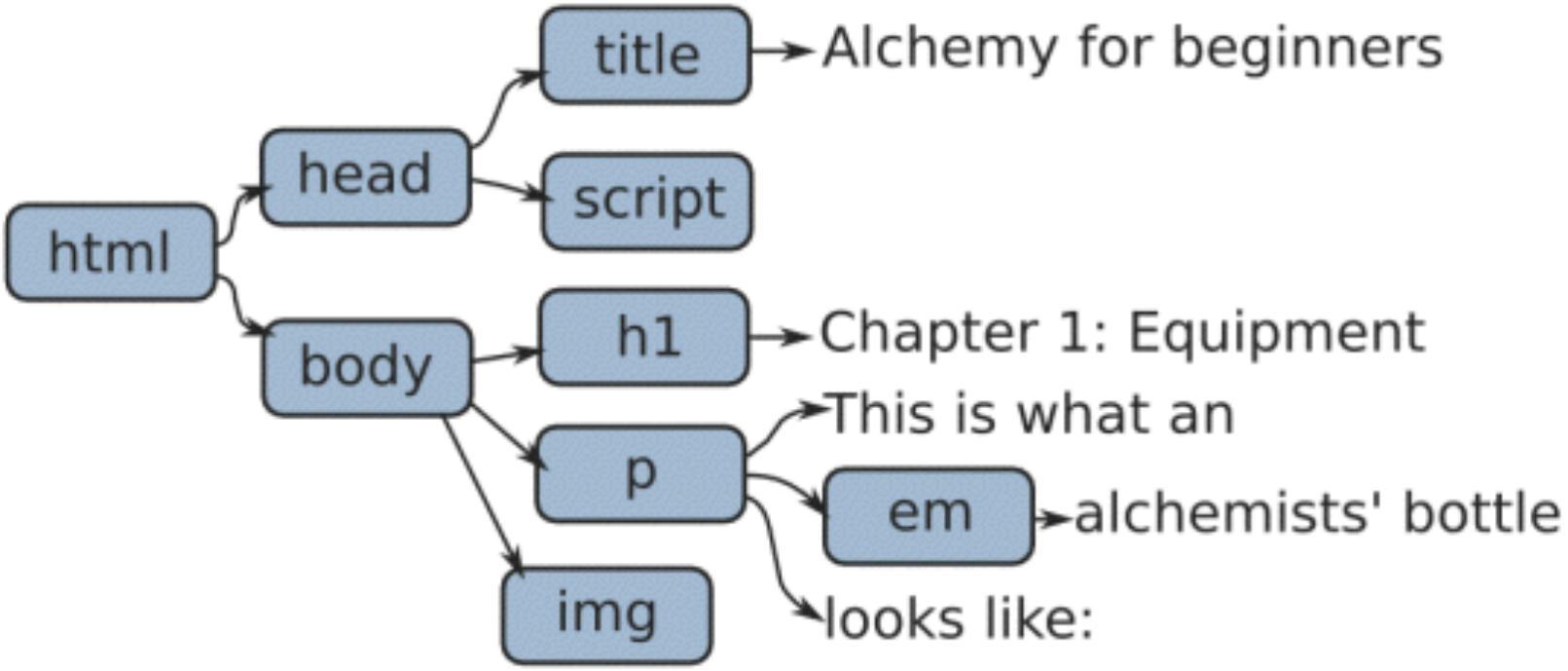
The DOM

- **D**ocument **O**bject **M**odel
- What most people hate when they say they hate JavaScript
- The browser's API: interface it provides to JavaScript for manipulating the page
- Browser parses HTML and builds a model of the structure, then uses the model to draw it on the screen
- "Live" data structure

DOM Structure

- Global **`document`** variable gives us programmatic access to the DOM
- It's a tree-like structure
- Each node represents an **element** in the page, or **attribute**, or **content** of an element
- Relationships between nodes allow traversal
- Each DOM node has a **`nodeType`** property, which contains a code for the type of element…
  - 1 – regular element
  - 3 – text

Document Structure



Document Nodes

- HTML like:

```
<p id="name" class="hi">My text</p>
```

- Maps to an object like:

```
{
    …
    childNodes: NodeList[1],
    id: “name”
     className: "hi",
    innerHTML: "My text",
    id: "name",
    …
}
```

- HTML attributes map **very loosely** to object properties

Working with the DOM

- Access the element(s)
  - Select one
  - Select many
  - Traverse
- Work with the element(s)
  - Text
  - Html
  - Attributes

Accessing individual elements

- Starting at **`document`** or a previously selected element

```
.getElementById(“main");
// returns first element with given id
// <div id=“main”>Hi</div>
.querySelector("p span");
// returns first matching css selector
// <p><span>Me!</span><span>Not!</span></p>
```

http://jsfiddle.net/mrmorris/wcff257b/

Accessing element lists

- Starting at **`document`** or a previously selected element

```
.getElementsByTagName("a");
// all <a> elements
.getElementsByClassName("fancy");
// all elements with specified class
// <span class="fancy"></span>
.querySelectorAll("p span");
// all elements that match the css selector
// <p><span>Me!</span><span>Me!</span></p>
```

Node Objects

- Single Element
  - **`HtmlElement`**
- Inherit properties / interfaces
  - **`HtmlElement`** `-> Element -> Node -> EventTarget`
- Collections
  - **`HTMLCollection/NodeList`**
    - Browsers may return one or the other
    - An array-like object containing a collection of DOM elements
    - The query is re-run each time the object is accessed, including the **length** property

Node Types

- Nodes can be of different types, we are mostly concerned with element nodes…
  - `anElement.nodeType`

```
// 1 = Element
// 3 = Text node
// 8 = Comment node
// 9 = Document node
```

Node Content

- Text node content
  - `textNode.nodeValue`
- Element node content
  - `el.textContent`
  - `el.innerText`
  - `el.innerHTML`

Node Attributes

- Accessor methods
  - `.getAttribute("title");`
  - `el.setAttribute("title", "Hat");`
  - `el.hasAttribute("title");`
  - `el.removeAttribute("title");`
- As properties
  - `.href`
  - `.className`
  - `.id`
  - `.checked`

http://jsfiddle.net/mrmorris/duopdjdb/

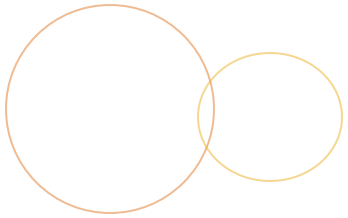
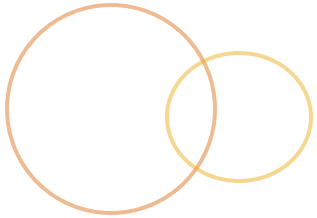
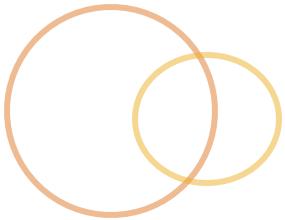
Traversal

- Move between nodes via their relationships
- Element node relationship properties
  - `.parentNode`
  - `.previousSibling, .nextSibling`
  - `.firstChild, .lastChild`
  - `.childNodes // NodeList`
- But… mind the whitespace!

http://jsfiddle.net/mrmorris/dv13y28m/

Modern Element Traversal

- Old traversal methods get tripped up by text-node, line break/whitespace
- New methods avoid that
  - Supported in ie9+
- From an element node
  - `.children`
  - `.firstElementChild, .lastElementChild`
  - `.childElementCount`
  - `.previousElementSibling`
  - `.nextElementSibling`

module

DOM > CREATION

Adding content

1. Create the container node
   - Insert additional content node(s)
   - Insert text node(s) if working with text
2. Determine *which pre-existing* node you can use to insert the *new* node
3. Insert it into the DOM (append, prepend, insert, replace)

http://jsfiddle.net/mrmorris/ktwdye0w/

Creating new nodes

- **`document.createElement`**`(“div")`
  - creates and returns a new node without inserting it into the DOM
- **`document.createTextNode`**`("foo bar”)`
  - creates and returns a new text node with given content

Set element content

- el.**textContent**
  - text content of node and all children
- el.**innerHTML**
  - html content of node and all children
- el.**nodeValue**
  - text, comment, attribute node values
- el.**value**
  - form input values

Adding nodes to the tree

```
// given this set up
var parentEl = document.getElementById("users"),
    existingChild = parentEl.firstElementChild,
    newChild = document.createElement("li");

parentEl.appendChild(newChild);
// appends child to the end of parentEl.childNodes

parentEl.insertBefore(newChild, existingChild);
// inserts newChild in parent.childNodes
// just before the existing child node existingChild
```

Moving and removing nodes

- Tree is “live”
  - Select and insert will move the element
  - Remove will detach it immediately

```
// given this set up
var parentEl = document.getElementById(“users”),
    existingChild = parentEl.firstElementChild,
    newChild = document.createElement(“li”);

parentEl.replaceChild(newChild, existingChild);
// removes existingChild from parent.childNodes
// and inserts newChild in its place

parentEl.removeChild(existingChild);
// removes existingChild from parentEl.childNodes
```

Styling elements

- Use element's "`style`" property
  - It's an object of style properties

```
el.style.color = "black";
el.style.marginLeft = "50px";
```

- Some style names differ in JavaScript
  - Hyphens become camelCase
    - `background-color => backgroundColor`
  - Some names were keywords
    - float => cssFloat

http://jsfiddle.net/mrmorris/hJwCj/

ClassList API

- Ability to get, set and toggle classes on element(s)
  - `el.classList.add(“class”);`
  - `el.classList.remove(“class”);`
  - `el.classList.toggle(“class”);`
  - `el.classList.contains(“class”)`

DOM Performance

- Interacting with DOM brings up performance issues
  - Searching the tree
  - Accessing a node
  - Iterating over a collection
  - Styling an element (cascades) and other redraws
    - Inserting nodes
    - Layout changes
    - Accessing css margins
    - Reflow/Repaint
- **When dealing with an element, store a reference rather than re-selecting**
- **When adding nodes, try to reduce the amount of insertions**

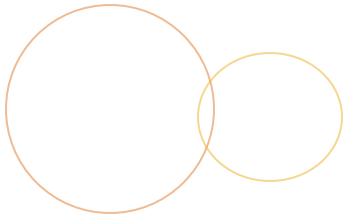
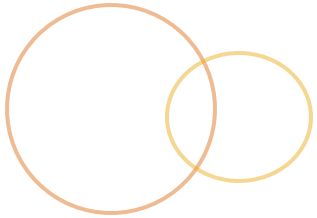
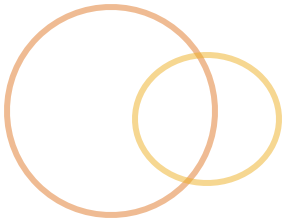
DOM basics - Recap

- The **DOM** is a model of the web page document.
  - It is a standardized convention.
- Browsers offer a **JavaScript API** to interact with the DOM
  - Can affect the page
- You can access, manipulate, create any content within the page
- jQuery will abstract much of the DOM API implementation nuances away, but it is still a good set of tools to have under your belt
  - **document.getElementById()**
  - **el.querySelector()**
  - **el.querySelectorAll()**
- Pay attention to DOM performance issues

Exercises: Dom manipulation

1. **Find the Flags**
   Using your special DOM hunting and walking skills, find the 3 “FLAG” elements in the content and move them to the “#bucket” element
   - http://jsfiddle.net/mrmorris/97ukrors/
2. **Embolden**
   Make a function, “embolden”, that takes an element and makes it appear bold. Write the HTML to test it.

```
function embolden(element) {
  // hint:
  // style it with fontWeight
  // OR wrap it in <strong>
}
```

module

EVENTS

Event-driven

- JavaScript engine has an single-threaded, event-driven, asynchronous programming model
  - As things happen
    - User clicks
    - Page completes loading
    - Form is submitting
  - Events are fired
    - Click
    - Load
    - submit
  - Which can trigger handler functions that are listening for these events

So many events…

- **UI**
  - load, unload, error, resize, scroll
- **Keyboard**
  - keydown, keyup, keypress
- **Mouse**
  - click, dblclick, mousedown, mousemove mouseup mouseover, mouseout
- **Focus**
  - focus, blur
- **Form**
  - input, change, submit, reset, select, cut, copy, paste

Basic Event Handling

1. Select an element
   - The element that triggers the event
   - or element that event passes through
2. Determine which event you want to listen for
3. Define an *event handling function* to respond to the event when it occurs

Event Handling

- Use the **addEventListener** method to register a function to be called when an event is triggered
  - ie9+

```
var el = document.getElementById("main");

el.addEventListener("click", function(event) {
      console.log(
            "event triggered on:",
            event.target
      );
}, false); // useCapture default:false
```

Handler options

- Inline
  - `<p onclick="function(){}"><p>`
- Traditional DOM event handlers
  - `el.onclick = function(){}`
- Event listeners (ie9+)
  - `el.addEventListener(event, function [, flow]);`
  - `el.removeEventListener(event, function);`
  - `el.attachEvent(); // ie8- only`
- Handlers are passed an "`event`" object
  - event object can have different properties depending on the event (ex: "`which`" for key pressed)
- http://jsfiddle.net/mrmorris/YAnBV/

Event handler context

- Functions are called in the context of the DOM element

```
el.addEventListener("click", myHandler);

function myHandler(event) {
  this; // equivalent to el
  event.target; // what triggered the event
  event.currentTarget; // where handler is bound
}
```

Gotcha: handlers with args

- This won't work like you expect

```
element.addEventListener('blur', doSomething(5));
```

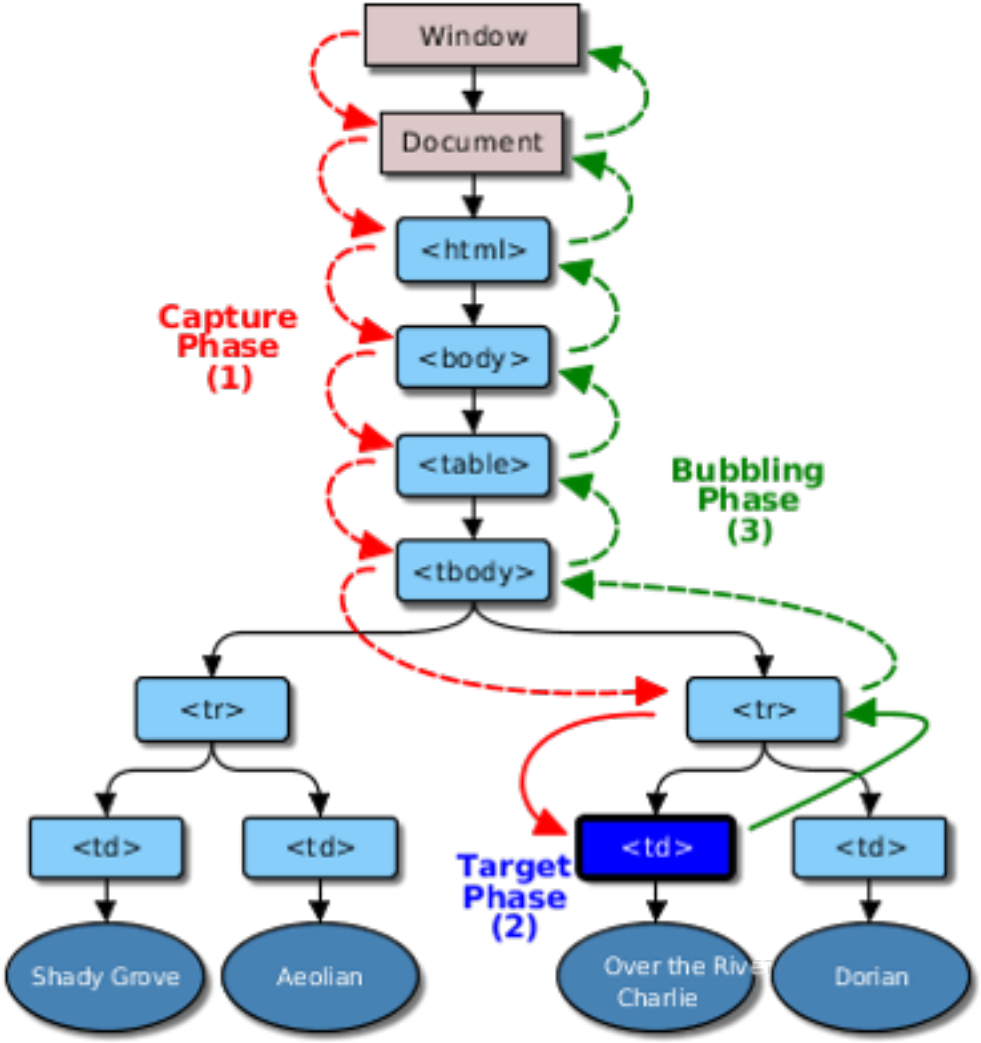
- Instead…
  - Wrap it in another function

```
el.addEventListener('blur', function() {
    doSomething(5);
});
```

  - Or use Function.bind()

```
el.addEventListener(
 'blur',
  doSomething.bind(el, 5)
);
```

Event Propagation



Event Propagation

- An event triggered on an element is also triggered on all “ancestor” elements
- Two models
  - Trickling, aka Capturing (Netscape)
  - Bubbling (MS)
- Event handlers can affect propagation

```
// no further propagation
event.stopPropagation();

// no browser default behavior
event.preventDefault();

// no further handlers
event.stopImmediatePropagation();
```

The event object

- Handlers are passed event object with lots of info about the event/user
  - Event.screenX
  - Event.screenY
  - Event.pageX
  - Event.pageY
  - Event.clientX
  - Event.clientY
- Key events include a "`keyCode`" property
- http://jsfiddle.net/mrmorris/8htsexcg/

Complete example

```
var el = document.getElementById('some-id');
el.addEventListener('click', function(event) {

        // "this" represents the element
        // handling the event
        this.style.color: "#ff9900";

        // "target" represents the element
        // that triggered
        event.target.style.color: "#ff9900";

        // you can stop default browser behavior
        event.preventDefault();

        // or you can stop the event from bubbling
        event.stopPropagation();

});
```

Event Delegation

- When a parent element is responsible for handling an event that bubbles up from its children
  - Allows new child content to be added and support the same event
  - Fewer handlers registered, fewer callbacks, reduced chance for memory leaks
- Relies on some event object properties
  - `target`, which references the originating node of the event
  - `currentTarget` property refers to the element currently handling the event (where the handler is registered)

Example: Event Delegation

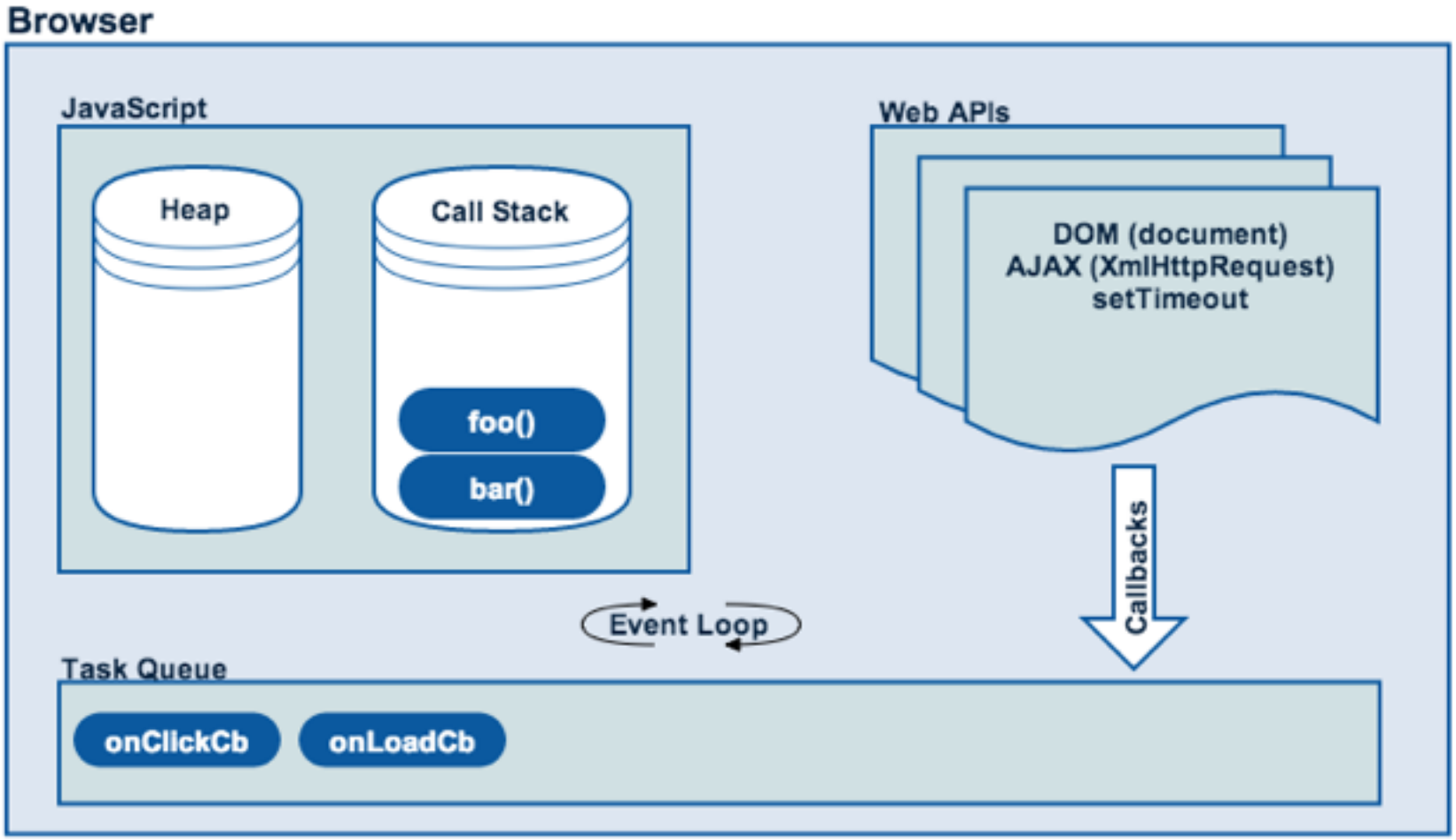
```
document
   .querySelector('ul')
   .addEventListener("click", myLiHandler);

function myLiHandler(event) {
   if (e.target && e.target.nodeName == "LI") {
      console.log(
         e.target.innerHTML, " was clicked!"
      );
   }
}
```

Event Loop

- The true power of JavaScript
- Allows asynchronous operations
  - Methods that get tacked into the queue are 'async'
- Each tick it returns a function from the Queue and runs it to completion (blocking)
  - Avoid blocking scripts…
    - alert, confirm, prompt, synchronous XHR

The event loop



Don’t block the event loop

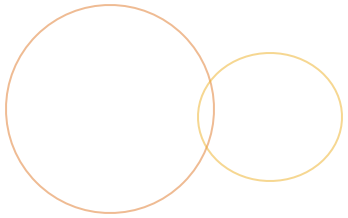
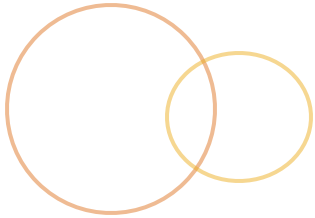
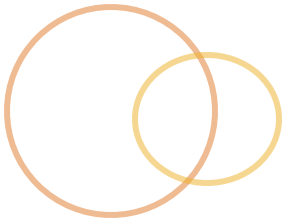
- Although event-loop is “never blocking”
- JS is still single-threaded
- All functions “run to completion”
  - Included messages in the queue
- For “run later” functionality…
  - **callbacks**
  - **Promises**
  - **Async/await** [ES6]
- For long running tasks…
  - **Eteration**: break task into multiple turns and call each with a setTimeout
  - **WebWorkers**: Move the task into a separate process

Events recap

- Events are notifications that bubble up from different sources in the page
  - Usually through a user doing something
  - Or some content in the page doing something
- Event delegation allows you to register a single handler to handle many (child) nodes' events
- The browser event loop is powerful but it is single-threaded so a long-running process can halt all interactions in the page.

Exercise - Event Handlers

- **Click Counter**
  Set up click handlers that count clicks and display information in the page.
  - https://jsfiddle.net/mrmorris/4e2wravL/

module

JQUERY

jQuery

- "*write less, do more*"
- A **utility lib** that abstracts the ugly parts of the DOM and cross-browser support
  - Selecting elements with css selectors
  - Interacting with the DOM + content
  - Event handling
  - Ajax
- Been around for 10+ years
- Doesn't do anything pure JavaScript can't do

jQuery versions

- 1.x
  - Most browser support
- 2.x
  - Dropped support for ie8
- 3.x
  - 3.x-compat, maximal browser support
    - includes ie8
  - 3.x
    - Modern/leading-edge browsers
  - 3-slim
    - Super trim, excludes ajax, effects, deprecated stuff

jQuery setup

- http://jquery.com/
- Include the script
  - Download or CDN
  - https://code.jquery.com/jquery-3.2.1.min.js
- Access the jQuery interface
  - `$(“#elementId);`
  - `jQuery(“#elementId”);`
  - `$.each([1,2,3], fn);`

noConflict

- `$.noConflict();`
  - Releases the $ namespace and restores any previous $ value

```
jQuery.noConflict();
jQuery(“div p”).hide();

var j = $.noConflict()();
j(“my-id”).hide();

// if you loaded two versions of jQuery
// completes releases second version
var v2 = $.noConflict(true);
```

Getting started with jQuery()

- Select element(s)
  - `$(document);`
- It returns a *jQuery selection object*
  - `var myDocument = $(document);`
- You can then invoke jQuery methods on the element(s) you selected
  - `myDocument.hide();`
- And… you can chain most method invokations
  - `$("p.fancy").doSomething();`
- There are also utility functions that don't depend on an element
  - $.merge([1,2,3], [4,5,6]);

Running code when DOM is ready

- These are all equivalent in jQuery
  - `$(handler)`
  - `$(document).ready(handler)`
  - `$("document").ready(handler)`
  - `$("img").ready(handler)`
  - `$().ready(handler)`
  - ~~`$(document).on("ready",handler)`~~ (deprecated)
- jQuery 3.x recommends:
  - **`$(handler)`**
- http://jsfiddle.net/mrmorris/y5C3L/

selecting in the DOM

- `$(cssSelector);`
  - returns a jQuery object, which represents the selection, and is a collection of the matched element(s)
  - Array-like, can access elements as though its an array
- Can set a context to select from
  - `$(cssSelector, contextSelector);`
  - contextSelector can be a css selector string, a DOM node or a jQuery object.
- You can include multiple selectors by comma delimiting them within the selection string
  - $("selector1, selector2")
- The selection is an object reference, not a copy
- Selection is *not* cached

Selecting, some examples

- `$("div")`
- `$("div#main ul")`
  - `$("ul", "div#main);`
- `$("#main li.fancy")`
  - `$(".fancy", "#main")`
- `$(".fancy")`
- `$("li:first")`
- `$("li.fancy, li.extravagant")`
- `$("input[type='text']")`

jQuery (CSS3) Selectors

- More Examples
  - http://jsfiddle.net/mrmorris/28h69/
- Use css 3 selectors
  - `*`
  - `elementname`
  - `#id`
  - `.class`
  - `selector1, selector2` (union)
- Hierarchy
  - `ancestor descendent` (descendent, ex: “li a”)
  - `parent > child` (direct child selector)
  - `previous + next` (adjacent sibling)
  - `previous ~ siblings` (sibling selector)

Selectors continued

- By content
  - `:contains('text')`
  - `:empty` – has no children
  - `:has(selector)` – checks children
  - `:not(selector)`
- Child filters
  - `:nth-child(an+b)`
    - (cycle size * counter + offset)
    - `nth-child(2)// second child`
    - `nth-child(2n+1) => 1, 3, 5…`
    - `nth-child(3n+2) => 2, 5, 8, 11…`
  - `:first-child`
  - `:last-child`
  - `:only-child`
  - `:nth-of-type(an+b)`
    - `Every nth child of same type`

Finding a specific item by order

- Absolute position
  - `:first`
  - `:last`
  - `:even`
  - `:odd`
- Each item in a jQuery object is given a 0-based index number, which can be used to filter the selection
  - `.eq(i)`
    - `$("selector").eq(3); // the 4th item`
  - `.lt(i)`
  - `.gt(i)`
  - `.first(), last()`

Finding elements by state

- Attribute filters
  - `[attribute]` – simply has it
  - `[attribute='value']` – or !=, ^= begins with
- Form filters
  - `:input`
  - `:text`
  - `:checked`
- Visibility
  - `:hidden`
  - `:visible`
- State
  - `:animated`

Explore selectors

- Take a gander, let's get familiar
  - An interactive selector map:
  - http://www.w3schools.com/jquery/trysel.asp
- Browse a site with the inspector
- Take some time
  - Selectors and being able to get the right element(s) is key to mastering jQuery

Exercise: Selector basics

- **Selection Basics**

  Try some basic selection approaches

  - http://jsfiddle.net/mrmorris/5zto6n44/
  - Hint:

    To style a table row/col with a background, you have to style the cell (td)

- **Selection the other way around**

  Describe what is being selected

  - http://jsfiddle.net/mrmorris/7wgvnd8n/

Add and filter selections

- You can add and filter elements to or from a jQuery selection
  - `$el.add(selector);`
    - `$('ul').add('p').hide();`
  - `$el.filter(selector)`
    - Filters a **selection** for sub-set of matching elements within
  - `$el.find(selector)`
    - Find **descendants** of elements in matched collection that match the second selector
    - Similar to $(selector, context);
  - `$el.not(selector), $el.has(selector)`
    - Similar to :not() and :has() css selectors, but not as performant

Exercise: Selecting continued

- **Manipulating your selection**
  - http://jsfiddle.net/mrmorris/adx1t106/

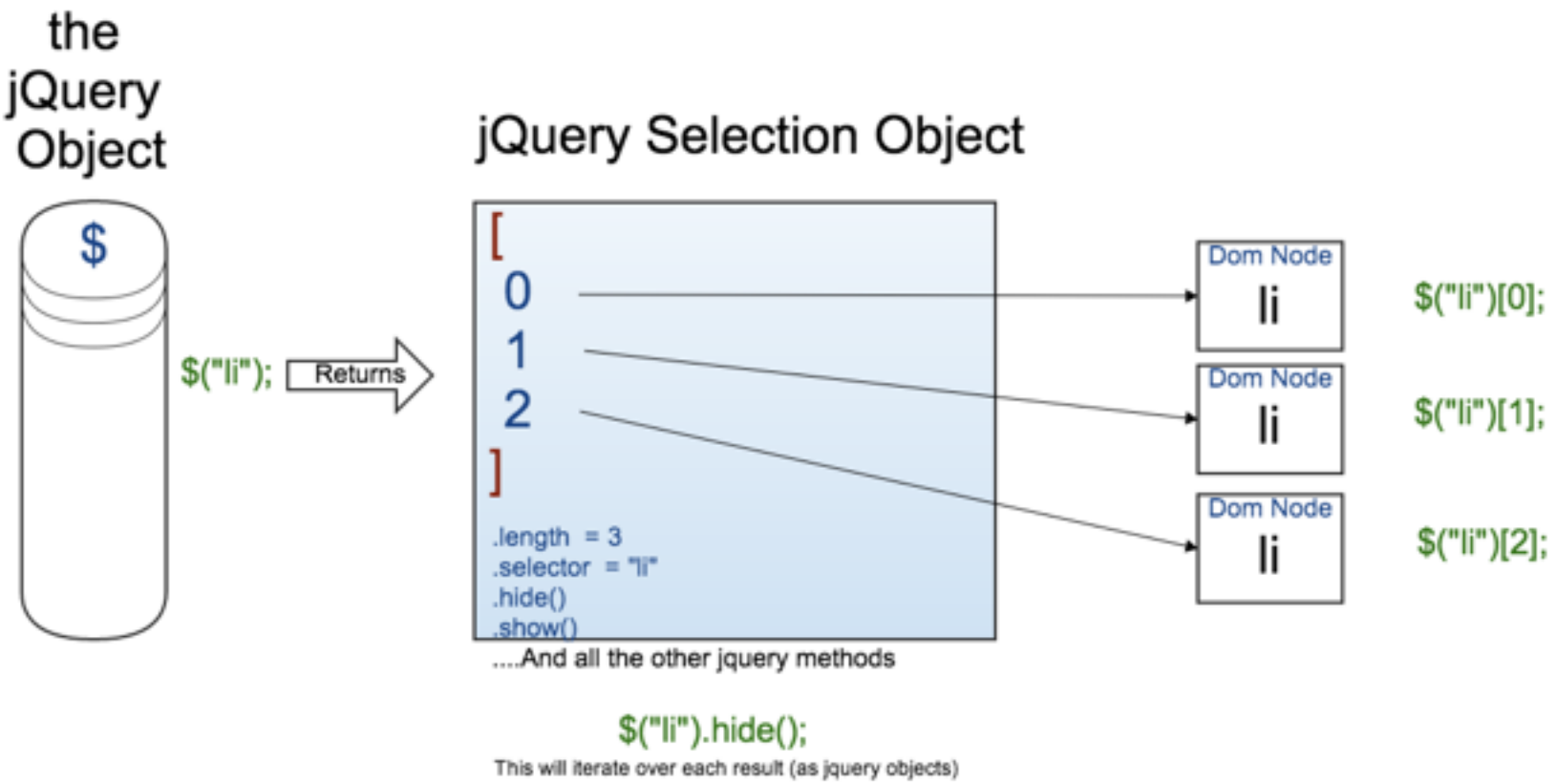
DOM traversal

- Three main traversal types
  - Parents
    - `.parent(), .parents(), .parentsUntil()`
    - `.closest()`
  - Children
    - `.children()`
    - `.find()`
  - Siblings
    - `.prev(), .prevAll(), .prevUntil()`
    - `.next(), .nextAll(), .nextUntil()`
    - `.siblings()`
- http://jsfiddle.net/mrmorris/3gY46/

jQuery selection - recap

- $(selector) returns a jQuery selection object
  - array-like
    - $(“li”); // array of li’s
  - Element methods can be run directly on it
    - $(“#id”).text(“goodbye”);
  - These can be chained
    - $(“#id”).text(“goodbye”).fadeOut()
  - Implicitly iterates
    - $(“li”).text(“goodbye”);
  - Can store the result and use all jquery element methods
    - var items = $(“li”);
      items.text(“goodbye”);

jQuery collection object



jQuery – selecting native dom node

- To access the native DOM element
  - `$("#foo")[0];`
  - `$("#foo").get(0);`
  - `$(".class").get(); // returns array of all`

jQuery looping

- `$(selector).each()`
  - for iterating over **jQuery *element collections***
    - ```
      $('li').each(function(i, el){
          // this === the dom node
          $(this).show();
      });
      ```
  - Implicit iteration for jQuery collections
    - `$('li').html("updated");`
- `$.each()`
  - for iterating over ***an array***
    - `$.each([1,2,3], function(i, val){})`
      ```

jQuery method chaining

- Typical pattern supported by jQuery
- Looks like:

```
$(“li”).css({color: ‘red’}).fadeIn();
$(“#nav li”)
   .removeClass(‘active’)
   .filter(‘.fourth’)
   .addClass(‘active’);
```

- How is this done?

jQuery element manipulation

- `.text()`
  - Returns escaped string
- `.html()`
- `.replaceWith()`
- `.remove()`
  - Drops element and all descendants
- `.detach()`
  - Removes and returns it, keeping a copy + event handlers for re-insertion
- `.empty()`
  - Removes child nodes
- `.unwrap()`
  - Removes parent of matched set, leaving matched elements
- `.clone()`
  - Create and returns copy of element + descendants
  - “true” to include event handlers
- http://jsfiddle.net/mrmorris/dyznkz48/

jQuery element insertion

- Methods
  - .before()
  - .after()
  - .prepend()
  - .append()
- These guys act a little differently
  - .prependTo()
  - .appendTo()
- **{before}**<li>**{prepend}**text**{append}**</li>**{after}**

jQuery element creation

- Create the element
  - `var newEl =  $("<div>Hi</div>");`
- Then Insert
  - `$("body").prepend(newEl);`
- Insertion methods accept an optional callback to perform dynamic insertions
  - ```
    $("div a").append(function() {
      return " (" + this.href + ")";
    });
    ```
    ```

Element Attributes

- `.attr(attributeName);`
  - `.attr(attributeName, value);`
- `.removeAttr(attributeName);`
- `.addClass(classNames);`
- `.removeClass(classNames);`
- `.css()`
- `.val()`
- `.isNumeric()`
- `.prop()`
  - Object ***property***, not necessarily attribute

Element styling

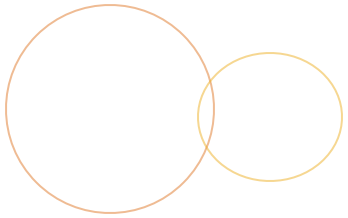
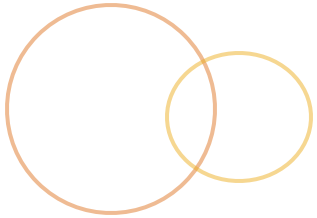
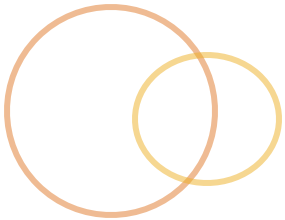
- .css() used to retrieve and set css properties
  - $('li').css('background-color'); // get
  - $('li').css('background-color', '#f9f9f9'); // set
- Can use an object of properties to set multiple
  - $('li").css({
    "padding-left": "+=75",
    "color": "#ff9f9"
    });
  - $('li").css({
    paddingLeft: "+=75",
    color: "#ff9f9"
    });

Element styling, continued

- jQuery is very generous in its flexibility
  - .css(‘background-color’, ‘red’);
    .css(‘backgroundColor’, ‘red’);
    .css({
      ‘background-color’: ‘red’
    });
    .css({
      backgroundColor: ‘red’
    });
  - These will all work just fine

Exercise: Selecting and manipulating

- **Find the flags, take two**
  Try finding and moving the flag elements again, but this time with jQuery helping us out
  - http://jsfiddle.net/mrmorris/bkfxf6h2/

module

JQUERY EFFECTS

jQuery animation methods

- Some basic functions
  - .show()
    - show(duration)
    - show(duration, easing)
    - show(duration, easing, callback)
    - show(options)
  - .hide()
  - .toggle()
  - .fadeIn()
  - .fadeOut()
  - .fadeTo()
  - .fadeToggle()
  - .slideUp()
  - .slideDown()
  - .slideToggle()

jQuery animate()

- .animate() for custom effects
  - .animate(props, options)
    - Options include duration, easing, queue, step (fn), complete (fn)…
  - Can animate most *numeric* css properties
    - Excludes color, background-color, for example
    - Color can be animated with the jQuery.Color plugin
  - .animate({
    opacity: 0.5
    }[, speed, easing, completeCallback]);

http://jsfiddle.net/mrmorris/Q32FC/

Sequence of animations

- Chain animations to queue them in a sequence

```
$("#el").slideUp('fast').delay(1000).fadeOut('slow');
```

- Introduce delays mid-queue
  - .delay(duration)
    - No way to cancel
    - Doesn't affect no-args show() or hide()
- Can introduce functions to the queue

```
$("#el").animate().animate().queue(function(next){
      // stuff then go to next item in queue
      $(this).dequeue();
      // or.. As of jquery 1.4
    next();
}).animate();
```

Effects queue

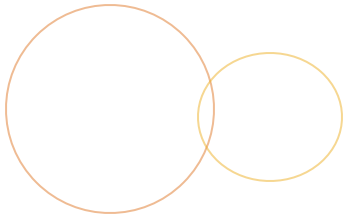
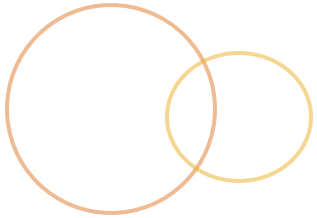
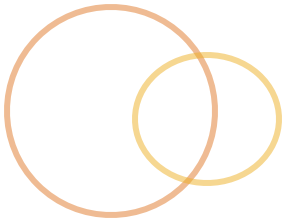
- “fx” is the default
- .stop();
  - Clear the queue
  - .stop(true, true); // clearQueue, jumpToEnd
- .finish()
  - stop animation, remove all queued, complete all animations
- .queue()
  - Get queued (array of fns) or manipulate a queue
- .dequeue()
  - Execute next function in the queue

jQuery basics - recap

- Working with jQuery revolves around its built-in selection method
  - $(“selectors”)
- Which returns a jQuery object, referencing a collection of DOM elements
- You can invoke methods (chained, if you like) on the result of that method, allowing you to manipulate and get information about the element
  - $(“selectors”).hide().show().hide();
- Creation of an element is as simple as passing markup to the jQuery method $(“ ”);
- Styling is supported, as are some light animations

Exercise: jQuery Basics

- **Fruit Basket**
  Use jQuery to add, replace and move “fruit” elements around the page
  - http://jsfiddle.net/mrmorris/geq2fyxv/

module

JQUERY EVENT HANDLING

jQuery Event Handling

- http://jsfiddle.net/mrmorris/rzk79p88/
- `.on(eventName, callback);`
  - .on(eventName, selector, callback);
  - .on(eventName, selector, data, callback);
- `.off(eventName, callback);`
  - .off(eventName, selector, callback)
  - .off(eventName, “*”); // all handlers
- `one(eventName, callback) {}`
  - Calls the handler only once
  - Similar to “on” with “off” within the handler

Older jQuery Event Handling

- Older versions of jQuery used (pre 1.7)
  - .click
  - .hover
  - .submit
- .bind() and .unbind()
  - are available but on/off is preferred.

jQuery – Event triggering

- .trigger(eventType [, extraParameters])
  - Trigger the event or eventType on an element
  - Bubbles up the tree
  - $(“#el”).trigger(“click”);
- .triggerHandler(eventType [, extraParameters])
  - Trigger only the handler
  - Only affects first matched element
  - No bubbling or default browser handling
  - Returns value of last handler executed

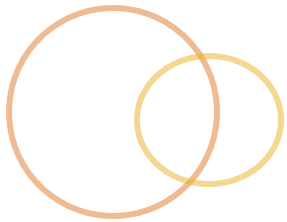
jQuery – Event Delegation

- When you delegate event handling to the parent element
  - A single parent element will fire for any event from the children
- Why?
  - Reduces number of event handlers being registered
  - Fewer memory leaks
  - “Live” behavior
- Delegation is *not* just a jQuery thing

```
$('ul').on('click', 'li.act', function(e) {
     // executed when the ul descendants
     // match the given selector
     // ie: "When li.act is clicked"
    e.target; // childNode triggering
});
```

- http://jsfiddle.net/mrmorris/mn4G9/

jQuery Event Object



- Passed to event handler functions, lots of data about the event
- Same event methods
  - .preventDefault()
  - .stopPropagation()
- Can pass it additional event info

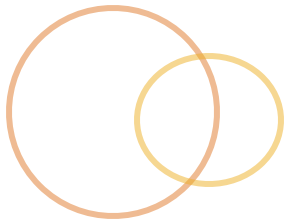
```
$(el).on('click', {customProp: customVal}, hndl);

function hndl(e) {
  console.log(e.data.ryan);
}
```

jQuery forms

- Selectors (css)
  - :button, :checkbox, :checked, :disabled, :enabled, :focus, :text, :file, :image, :input, :password, :radio, :reset, :selected, :submit
  - Not great performers.. So scope on the form element or filter()
- Methods
  - inputEl.val(), inputEl.prop(), .is(“:checked”)
- Getting the data
  - $(form).serialize(); // string
  - $(form).serializeArray(); // array of objects
- Not all inputs are equal
  - Checkboxes are not present in data when unchecked
  - Use **el.prop(‘checked’)** to determine if checked

jQuery form events



- Field events
  - blur, focus
  - change
    - After a field change and field is blurred
  - input
    - HTML5; when text input changes
  - select
    - When text in an input is selected
- Form events
  - submit, reset
- Return false to prevent actual POST
- http://jsfiddle.net/mrmorris/p2n9G/

jQuery Custom Events

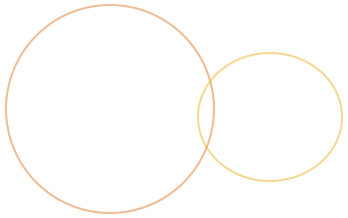
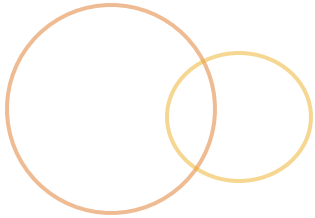
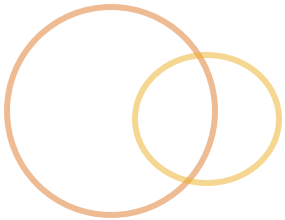
- Trigger and listen for your own events
  - `$(el).trigger('topic:label');`
  - `$(otherEl).on('topic:label');`
- Events propagate normally
- Allows you to focus on target of behavior, rather than the element(s) that trigger it

```
// after user removes last row in table
$tableEl.trigger('grid:empty');

// a site message
$messageEl.on('grid:empty', function() {
  $(this).html('There is no data').fadeIn('slow');
}
```

Exercise: jQuery Event Handling

- **Click chaser**
  [Easy] Basic event handling
  - http://jsfiddle.net/mrmorris/ofh1cz0d/
- **Image Grid**
  [Interm.] Event handling of a grid of images, including the need to set up delegation
  - http://jsfiddle.net/mrmorris/jjomkn54/
- **Issue Tracker**
  [Adv.] Set up an issue submission form using form events, including some basic validation.
  - http://jsfiddle.net/mrmorris/tpyhgt42/

module

JQUERY AJAX

jQuery Ajax

- Several ways to do this, but they are all shortcuts of **$.ajax()**

```
var jxhr = $.ajax({
    type: string (GET or POST)
    url: string
    data: mixed (converted to query str)
    success: function
    error: function
    complete: function
    timeout: number
    dataType: string (xml, json or html)
    beforeSend: function
}); // returns jQuery xhr object
```

jQuery – handling the response

- Callbacks (deprecated in jQuery 3.0)
  - beforeSend, dataFilter, success, error, complete
- or…jQuery XHR implements Promise interface
  - `var prom = $.ajax({…});`

```
prom.done(function(response){…});
prom.fail(function(){…});
prom.always(function(){…});
prom.then(doneFn, failFn);
```

- These promise methods can be chained
  - `$.ajax().done().fail().always()`

jQuery ajax shorthand

- element.load()
  - Loads data directly into an element
  - Can target fragment elements in the response
    - $('#content').load('bla.html#content');
- $.get(settings);
- $.post(settings);
- $.getJSON(settings);
- $.getScript(settings);

jQuery Ajax examples

- More $.ajax with form post
  - http://jsfiddle.net/mrmorris/pj4e7jxv/
- Example with CORS API
  - http://jsfiddle.net/mrmorris/5vwcx7zp/

Promises

- Ajax requests are returning a promise
  - Actually a “jqXHR” object that implements the *Promise interface*
- Promises have a lifecycle
  - Unfulfilled
  - Fulfilled
  - Failed
- In jQuery, the Promise is based off the $.Deferred object

The advantages of a promise

- You can:
  - add multiple success/failure callbacks
  - add callbacks even after the Promise lifecycle is complete
- Use the behavior of Deferred objects
  - Like delay a callback until multiple promises are complete
  - Or pipe result data
- The result of an asynchronous operation(s) can be treated as a first class object
- A solution to “callback hell”
  - Think of it like async pathways

jQuery Ajax review

```
$.ajax({
      type: 'GET', // or 'POST', 'DELETE',
      data: {},
      success: callback
      error: callback
      complete: callback
      dataType: 'json', // 'json','html'
});
```

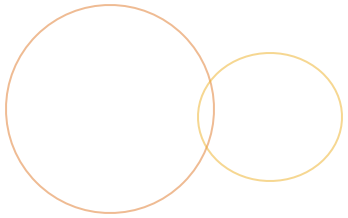
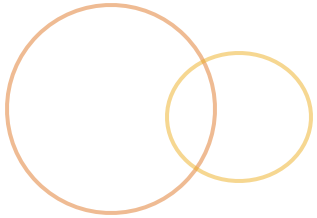
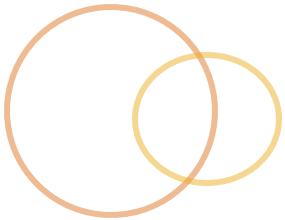
- $.ajax (and shortcuts) method immediately (synchronously) returns a Promise object
  - ```
    var prom = $.ajax({…});
    prom.done(function(response){…});
    prom.fail(function(){…});
    prom.always(function(){…});
    ```
- These promise methods can be chained
  - `prom.done().fail().always()`
    ```

Anatomy of an Ajax request

```
var prom = $.ajax({
        type: 'GET',
        url: 'http://some.api.com/data.json',
        dataType: 'json',
        data: {}
});
prom.done(function(response, status, prom) {
        // process your response data
});
prom.fail(function(prom, status, error) {
        // handle the error
});
prom.always(function(response, status, error) {
        // wrap up after done or fail
});
// combined done/error
prom.then(doneCallback, failCallback);
```

Exercise – jQuery Ajax

- We'll be using this API:
  - http://jsonplaceholder.typicode.com/
  - https://github.com/typicode/jsonplaceholder
- **Photo Grid**
  Working together lets complete a dynamic photo grid
  - http://jsfiddle.net/mrmorris/Ln8ecynw/
  - Hint: Check out the network panel
- **Todo List**
  Use ajax to load content from an API to build up a todo list
  - http://jsfiddle.net/mrmorris/1gtqsohv/

module

JQUERY UTILITIES

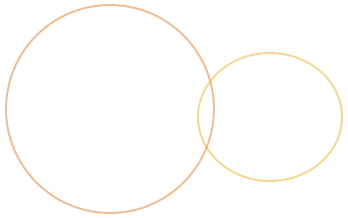
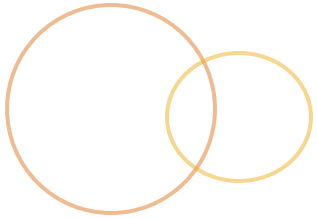
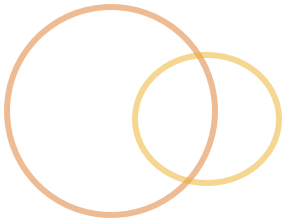
jQuery Iteration

- $.each(array|object, callback(index, val))
  - Iterates and run a callback on each element in array
  - Object enumeration (prop, val)
  - Return “false” to break the loop
- $.map(array|object, callback)
- $.grep(array|object, callback); // filter
- $(selector).each()
  - is used exclusively to iterate over a jQuery object
  - “this” is a DOM element

http://jsfiddle.net/mrmorris/bmtqr5s6/

jQuery Extras

- Accessed through $/jQuery
  - .trim()
  - .isNumeric()
  - .isArray()
  - .parseJSON(string)
  - .parseXML(string)
  - .parseHTML(string)
  - .merge(firstArray, secondArray)
    - This merges into the first
  - .extend(target, obj1, obj2)
    - Can set first arg to true for “deep”
- el.data(key, value)
- el.removeData(key)

module

JQUERY UI INTRODUCTION

jQuery UI

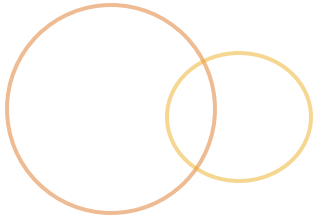
- http://jqueryui.com/
- “jQuery UI is a curated set of user interface **interactions**, **effects**, **widgets**, and **themes** built on top of the jQuery JavaScript Library”
  - Accordions
  - Date picker
  - Tabs, etc…
- Extends jQuery, using jQuery conventions and introducing a few of its own

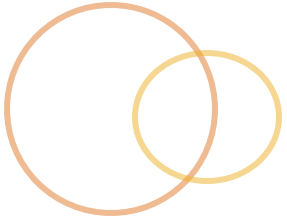
Getting set up with jQuery UI

- You must already have jQuery.js included in the page
- Download jqueryui script+css or use a CDN
  - http://jqueryui.com/download/
  - https://code.jquery.com/ui/
- Customize the package and/or theme
- Include the ui script + theme css in your page

http://jsbin.com/yicex/1/edit?html,js,console,output

Widgets





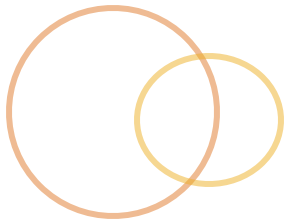
- Accordion
- Autocomplete
- Form widgets:
  - Button
  - Datepicker
  - Selectmenu
  - Slider input
- Dialog
- Menu
- Progressbar
- Spinner
- Tabs
- Tooltip

http://jqueryui.com/accordion/

Using widgets

- Match the HTML structure
- Initialize on an element
  - $("#my-element").widgetName();
- Set Options
  - $("#my-element").widgetName({
    color: "blue",
    times: 5
    });
  - Can pass multiple options objects, which are merged
  - Can override defaults via widgets prototype
    - $.ui.widgetName.prototype.options.times = 0;
- Use Methods
  - Query state or perform actions on the widget
  - $("#my-element").widgetName("methodName", arg1);

Widgets examples



- Button and Dialog
  - http://jsfiddle.net/mrmorris/5qvo4zy3/
- Accordion and Tabs
  - http://jsfiddle.net/mrmorris/c89p22ks/
- Form components
  - http://jsfiddle.net/mrmorris/2Lq5fh4z/
- Menus
  - http://jsfiddle.net/mrmorris/gxt2582m/
- Tooltips
  - http://jsfiddle.net/mrmorris/z8f5kx6o/

Widget events

- Widgets trigger events
  - All widgets have a “create” event upon instantiation
  - Can have custom events
  - To handle them
    - Just bind to the event, in this ex: “something”

```
$(“#my-element”).on(“wigdet**something**”, function() {
    console.log(“Hi”);
});
```

    - or hook into the event callback

```
$(“#my-element”).widgetName({
    **something**: function() {}
});
```

- Check the documentation
  - Dialog Events: http://api.jqueryui.com/dialog/

Interactions

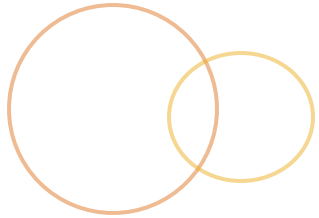
- “Set of mouse-based interactions to add rich interfaces and complex widgets”
- Similar to widgets, they have options, methods and events.
- The interactions:
  - Draggable
  - Droppable
  - Resizable
  - Selectable
  - Sortable

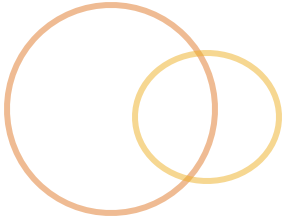
http://jqueryui.com/draggable/

Interactions Examples

- Drag and Drop
  - http://jsfiddle.net/mrmorris/ko7kxs80/
- Sortable
  - http://jqueryui.com/sortable/
- Selectable
  - http://jqueryui.com/selectable/
- Resizable
  - http://jqueryui.com/resizable/

Effects



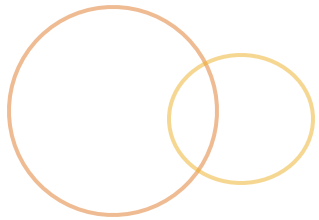


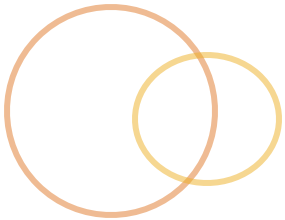
- Extends jQuery
  - New Effects
  - Better class animation support
- Not all styles can be animated
  - background-image
- Color animation is now built-in

Animating

- Class animations
  - .addClass(class, duration, complete)
  - .removeClass(class, duration, complete)
  - .toggleClass(class, duration, complete)
  - .switchClass(removeClass, addClass, duration, cb)
  - Specificity matters!
- Display animations
  - .hide(effect, options, complete)
  - .show(effect, options, complete)
  - .toggle(effect, options, complete)
- These methods are not new, just extended
- http://jsfiddle.net/mrmorris/hyannf0d/

Effects





- Some animations will use effects
  - fade
  - highlight
  - bounce
  - scale
- Effects can have options, like “distance” and “times” for the “bounce” effect
- Can apply an effect directly to an element
  - .effect(effect, options, duration, complete);

http://api.jqueryui.com/category/effects/

Easing

- The easing equation for our animations
- Controls speed of the animation over time
- “linear” is boring
  - “swing” -> jquery default
  - “easeInQuad”
  - “easeOutQuad”
  - And many more…
- http://api.jqueryui.com/easings/
- http://easings.net/

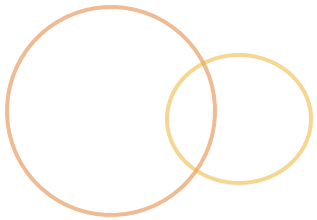
Animating

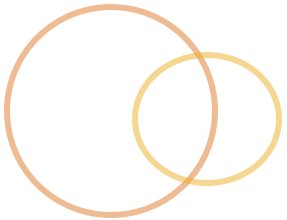
- Animate specific styles
  - .animate({
    color: “green”,
    backgroundColor: “#f90”,
    borderLeftColor: rgb(20, 20, 20)
    });
- Extends jQuery core .animate to support color animations
- Better to just use class-based animations unless specificity is an issue

Animation method arguments

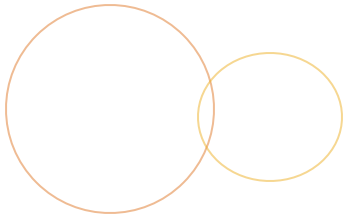
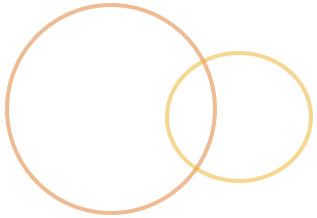
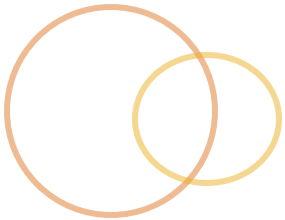
- .hide
  - (effect [, options] [, duration] [, complete])
  - (options)
    - effect
    - easing
    - duration
    - complete
    - queue
- .addClass
  - (className [, duration] [, easing] [, complete])
  - (className [, options])
    - duration
    - easing
    - complete
    - children
    - queue

Position





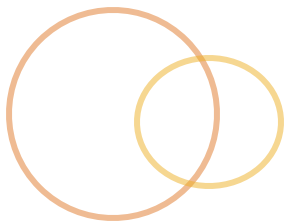
- Helper to position elements relative to window, document, other elements, or the mouse
  - // position my center at the center of targetElement
    $(“#element”).position({
    my: “center”,
    at: “center”,
    of: “#targetElement”,
    collision: “flip”
    });
  - Positions: “horizontal vertical”
    - Horizontals: left, center, right
    - Verticals: top, center, bottom
    - “right” => “right center”
    - Offsets are ok: “right+10 top-25%”
  - Collisions: flip, fit, flipfit, none

module

FORMS AND VALIDATION

HTML5 Forms



- New form control types
- Validation options built into HTML

Form Validation - Constraints

- With HTML5 you get new input types and their input verification
  - url, tel, email, number, date, etc…
- New attributes
  - required (binary)
  - pattern (regex or string match)
  - min, max, step, maxlength
  - Form “novalidate” attribute
- New events
  - “invalid” on inputs

http://jsfiddle.net/mrmorris/zh18vn4x/

HTML5 Forms continued

- new **css pseudo-selectors**
  - :valid, :invalid, :required, :optional
  - :in-range, :out-of-range
- new **element attributes**
  - novalidate
  - required
  - pattern, min, max, maxlength, step
- new **input types**
  - email
  - url
  - number
  - range
  - password
  - color
  - month
  - search
  - tel
  - date
  - time
  - week

http://jsfiddle.net/mrmorris/0pekaacb/

Downsides

- Client-side validation is easy to hack/skip
- Browser's are inconsistent in how they implement
- Error messages can't be customized (w/out js)
- Error messages are tied to user's locale
- Business logic embedded in markup

- so… you can use the HTML5 Constraint Validation API to programmatically pull it off

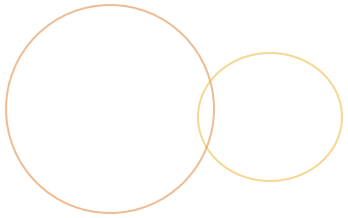
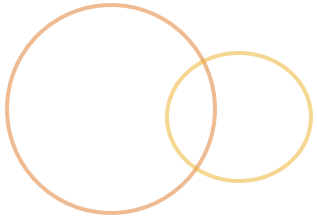
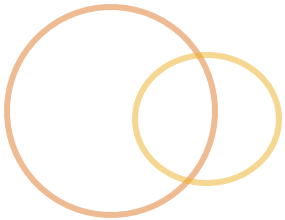
Constrain Validation

- element.setCustomValidity(msg);
- element.validity
  - a **ValidState** object
    - customError
    - patternMismatch
    - valid
    - valueMissing
    - willValidate
- element.checkValidity()

https://jsfiddle.net/mrmorris/guLh155u/

Exercise: JavaScript and the DOM

- **Tabbed UI**
  Create a tabbed UI that responds to click events and adding new tabs.
  - http://jsfiddle.net/mrmorris/osq6fed3/
- **Table Builder**
  Create a table-builder function that accepts JSON data and builds a table element with all the trimmings.
  - http://jsfiddle.net/mrmorris/mnyn3y0t/

the end is hear

WRAPPING UP

jQuery/DOM Best Practices

- **Avoid changing the DOM** as much as possible
  - But if you have to, use a document fragment or string
  - Check for selection length before manipulating
- **Be specific** when selecting
  - Use “context” or “filter”
  - Don’t be *over*-specific or use the * selector
- **Store your selection**(s)
- **Prefer native** over libs (where it makes sense)
- **Name your variables well**
  - $jQuerySelectedThing
  - titleElement
- **Avoid double wrapping** $(jQueryObjects)

Going beyond

- AJAX
- Modules
- jQuery toolkits
  - Help with modules
  - Minify and compile
  - Transpile
- HTML5 Apis
  - Web Workers
  - Sockets
- JS in the server
  - NodeJS

Stay sharp

- Solve small challenges for kata
  - http://www.codewars.com/
- Code interactively
  - http://www.codecademy.com/
- Share your code and get feedback
  - http://jsfiddle.net
- Free e-book
  - http://eloquentjavascript.net/
- Re-introduction to JavaScript
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript

Go now and code well

- That's a wrap!
  - What did you enjoy learning about the most?
  - What is your key takeaway?
  - What do you wish we did differently?
- Any other comments, questions, suggestions?
- Feel free to contact me at mr.morris@gmail.com or my eerily silent twitter **@mrmorris**