

JavaScript, jQuery and Front-end frameworks

Ryan Morris
Develop Intelligence



Introductions



- Who am I?
- Who are you?
 - What do you do?
 - What do you hope to gain from this course?
 - What is your programming background?
 - JavaScript
 - HTML
 - CSS
 - jQuery
 - Angular
 - Node
 - Unit Testing

How the class works



- Informal
- Planned and “on demand” exercises
- There’s too much to cover, so some things are left out
- Stop me anytime for questions or more information
- We will hit the documentation as needed
- Daily outline is flexible, we’ll adjust as needed

Course overview



- Day 1: JavaScript
- Day 2: HTML and the DOM, jQuery intro
- Day 3: jQuery and jQuery Plugins
- Day 4: Whatever we didn't cover!
 - jQuery UI
 - Web APIs and frameworks
 - Intro to AngularJS and Node.js
 - Testing

Set up



- A browser with dev tools (
 - Chrome
 - Firefox
 - ie...)
- An editor
 - SublimeText
 - Notepad++
 - WebStorm
- Something to serve up files
 - Local server (for ajax)
 - Mongoose
 - Apache/XAMPP
- Our web editors
 - <http://jsfiddle.net/>
 - <http://jsbin.com/>

Resources



- Documentation
 - <http://devdocs.io>
 - <https://developer.mozilla.org/en-US/docs/Web>
 - <http://kapeli.com/dash> (Mac only)
- Compatibility checks
 - <http://caniuse.com>
- ES 5 compatibility table
 - <http://kangax.github.io/compat-table/es5/>
- jQuery APIs
 - <http://api.jquery.com/>
- JS Lint
 - <http://www.jslint.com/>

Remote workstations



- ◉ Can't use local or jsfiddle.net? Use a remote desktop
 - ◉ Connect to <https://protechra.webex.com> using Internet Explorer or Firefox.
 - ◉ Enter your user id and password
 - ◉ Id: **Remote_Access3XX** ← XX will be unique to you, and provided by instructor
 - ◉ Pass: **Pa\$\$w0rd** ← that's a zero in there
 - ◉ On the next screen you'll see a Computer3XX listed; connect to it and follow the instructions
 - ◉ When prompted for an access code:
 - ◉ **Pa\$\$w0rd**
- ◉ To adjust the view, click the “down” arrow in the lower right-hand corner and change it from “View”

Day 1: JavaScript 101-401



- ~~Intro/Setup~~
- JavaScript basics
 - Core data types, syntax, control flow
 - Data structures
 - Functions
 - Scope, closures and other patterns
 - Inheritance

Wizard check



- ◉ How well versed are you in basic programming concepts and JavaScript?
 - ◉ Function?
 - ◉ Objects and prototype?
- ◉ Try me out
 - ◉ <http://jsfiddle.net/mrmorris/a5v1p5by/>

JavaScript History



- ◉ Netscape wanted interactivity like HyperCard w/ Java in the name
- ◉ Designed & built in 10 days by Brendan Eich
- ◉ Combines influences from...
 - ◉ Java, “Because people like it”
 - ◉ Scheme, the actor model
 - ◉ Self, prototypal
- ◉ 1999 – ES3, in all browsers by 2011
- ◉ 2009 – ES5
- ◉ ES6 coming soon.... ES7 in progress, too

What it is



- ◉ ECMAScript (usually v5, soon v6)
- ◉ Interpreted
- ◉ Fully dynamic
- ◉ Single-threaded
- ◉ Unicode (UTF-16, to be exact)
- ◉ Prototype-based (vs Class based)
 - ◉ But... Multi-paradigm
- ◉ Slightly different across all implementations
- ◉ Loosely typed, but not un-typed
- ◉ Case-sensitive
- ◉ Kind of weird
- ◉ Has lots of bad, lots of good

Why JavaScript?



- ◉ Despite the shortcomings, it's pretty awesome
 - ◉ Very expressive
 - ◉ Very flexible (that multi-paradigm thing)
 - ◉ Lightweight
- ◉ The language of the web
 - ◉ The browser
 - ◉ Client-side frameworks
 - ◉ A server and command line services
 - ◉ Beginning to dominate the entire software stack
- ◉ Easy to learn partially, much harder to learn completely and sufficiently

Just the basics (Syntax)



- JavaScript falls in the C-family, so...
- White space and indentation generally doesn't matter
- Blocks are wrapped with curly braces { }
- Statements, expressions, constructs are terminated by semi-colon ;

First things first...



- Open up your browser console...
 - F12
 - Right click > Inspect
- Type in:
 - `console.log("Hello World!");`
- Hit enter

Browser Debugging



- ◉ Use browser dev tools to access its JavaScript console
 - ◉ The browser's "console" is a line interpreter and log
- ◉ All major browsers are converging to the same API for console debugging
- ◉ Can use it to set breakpoints
 - ◉ Let you see scoped variables and context
 - ◉ Can set a conditional break-point
- ◉ This is where we'll be working; follow along!
 - ◉ `console.log() => echo`

Core Data Types Overview



- Primitives
 - strings
 - numbers
 - booleans
 - null - lack of value
 - undefined – no value set, the default in JavaScript
- Objects
- Functions
 - “Invokable objects”
- Built-in objects
 - RegExp, Date, Error
 - String, Number, Boolean, Function, Array

Working with variables



- Initialize a new variable with var

- `var myEmptyVar; // undefined`
 - `var myVariableName = “string value”;`
 - `var my_other_var2 = 5;`

- `typeof` to check type

- Returns the value as a string
 - `typeof 5; // “number”`

Variable Naming Conventions



- Start with a letter, _ or \$
- Followed by 0 or more letters, digits, _ or \$
- Case-sensitive, mind you
- Start lowercase by convention (uppercase for constructor functions)
- No JavaScript keywords
- By convention, avoid leading with _ or \$

Undefined & Null



- ◉ There are two special values, null and undefined
- ◉ Little difference between the two and usually doesn't matter which you prefer (accident in the design of JavaScript)
- ◉ Variables declared without a value will start with undefined
- ◉ Can compare to null to see if a variable has a value
 - ◉ `null == undefined; // true`
 - `null == 0; // false`

Numbers



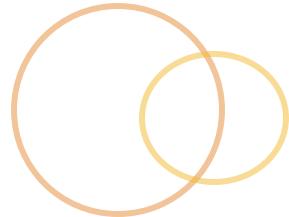
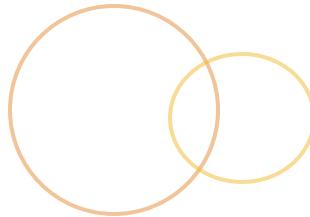
- ◉ All numbers are 64 bit floating point
 - ◉ Decimal can be included, so this is ok: 9.81
 - ◉ Scientific notation: 2.998e8
 - ◉ Hexadecimal: 0xFF; // 255
- ◉ Special numbers:
 - ◉ NaN
 - ◉ Usually the result of some bad math
 - ◉ JavaScript does consider this a “number”
 - ◉ Toxic... will destroy all subsequent math
 - ◉ `NaN != NaN; // Wha????`
 - ◉ Infinity
 - ◉ Division by 0

Numbers – Floating point math



- ◉ Floating point operations are not accurate
 - ◉ `0.1 + 0.2 == 0.3; // is false in js`
 - ◉ `0.1 + 0.2 // equals.... Try it out?`
- ◉ So what to do?
 - ◉ Use a special data type like Big Decimal
 - ◉ <https://github.com/dtrebbien/BigDecimal.js>
 - ◉ Round to a fixed decimal place
 - ◉ `myNum.toFixed(2); // returns string`
 - ◉ `parseFloat(myNum.toFixed(2)); // back to num`
 - ◉ Work only with integers

Strings



- Enclosed by “ or ‘ (just don’t mix them)
 - `var str = “My String”;`
- Escape with backslash (\)
 - `\n` is newline
- + Operator for concatenation
- Can utilize the + operator to convert a string to a number
 - `(+“42”); // will equate to the integer 42`

Type Coercion



- JavaScript is loosely typed
- Implicit Conversion
 - If a variable type is not what JavaScript expects, it will convert it on the fly – be careful.
- It usually does this in an unexpected way...
 - `8 * null -> 0`
 - `“5” - 1 -> 4`
 - `“5” + 1 -> 51`
- Much confusion
 - `[] + [] -> empty string`
 - `[] + {} -> [object Object]`
 - `{ } + [] -> 0`
 - `{ } + { } -> NaN`

{objects: “intro”}



- Objects are a dynamic collection of *properties* grouped together
- The object *literal*
 - `var emptyObject = {};`
 - `var myObject = {
 key: value,
 key2: value2
};`
- Set/Get properties
 - `myObject.key = 5;`
 - `myObject[“key”] = 5; // supports variable keys`
- Properties can reference functions (methods)
 - `var myObject = {
 key: value,
 methodName: function(){}
};`

Everything* is an object



- *most things
- Primitive literals are not objects
 - But do have Object counterparts (except null and undefined)
 - `var strPrimitive = "My String";`
`var strObject = new String("Hello");`
 - Temporarily coerced to object when used as object
 - So... we can access properties and invoke methods of objects, including primitives
 - `str.length;`
 - `str.toUpperCase();`
 - `"Hello".length;`

Arrays [Intro]



- Arrays are collections of data stored with sequential integers
- In JavaScript, the array is an object that behaves kinda like an array (array-like)
- Array literal...
 - `var emptyArray = [];`
 - `var myArray = [1,2,3,4];`
- Get/Set
 - `myArray[1]; // 2`
 - `myArray[1] = 20;`
- Arrays (an object) come with some extra properties and methods... like `length`
 - `myArray.length; // 4`

Recap: basic data types



- 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”
- 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

Comments



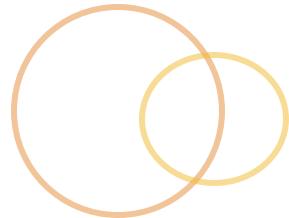
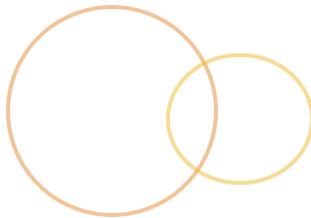
- Line comments with //
 - // This is a comment line
- Wrap block comments with /* */
 - /* Wrap comments */
 - /**
 * Document block style
 */

Expressions



- A fragment of code that produces a value
 - Arithmetic
 - String
 - Logical
 - Object
- Like...
 - 22 (the literal)
 - $1+1$
 - 2^{++}
 - $(x) + (52 + 3)$
 - $(x > 0)$

Operators



- Arithmetic
- Assignment
- Comparison
- Bitwise
- Logical
- String

typeof operator



- Takes one operand
 - `typeof variableName;`
- Returns a string
- Type gotchas
 - `object = object`
 - array = object**
 - `number = number`
 - `boolean = boolean`
 - `undefined = undefined`
 - `string = string`
 - `function = function`
 - but... `null = object`**
- <http://jsfiddle.net/mrmorris/DjBtL/>

Arithmetic operators

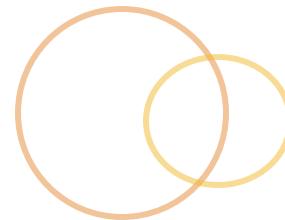
○ +

○ -

○ *

○ /

○ %



Shortcut operators



- ◉ `+=, *=, -=, /=, %=`
- ◉ `x++ (increment)`
- ◉ `++x`
- ◉ `x-- (decrement)`
- ◉ For your bag of tricks:
 - ◉ `(+x);`
 - ◉ Convert number to a string
 - ◉ `!!myVar;`
 - ◉ Double bang can convert any value to a boolean

Exercise: Get primitive



- ◉ Declare some variables, get to know basic data types
- ◉ Fork this
 - ◉ <http://jsfiddle.net/mrmorris/qpo9quc5/>

Bitwise Operators

○ &

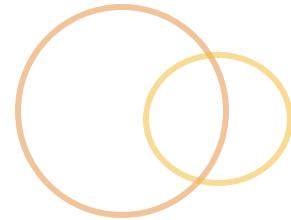
○ |

○ ^

○ >>

○ >>>

○ <<



Comparison Operators



- Equality

- ==
- !=

- Strict Equality

- ===
- !==
- 5 === 5; // true
- ({} === {}); // false

- Greater than/Less Than

- <
- >
- <=
- >=

Logical Operators



- ◉ **&&**

- ◉ If first operand is **truthy**, then result is second operand.
Otherwise result is first operand

- ◉ So...

```
return a && b; // is like...
if (a) {return b;} else {return a;}
```

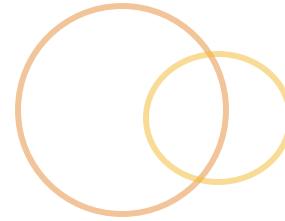
- ◉ **||**

- ◉ If first operand is **truthy**, then result is the first operand.
Otherwise result is the second operand.

- ◉ So...

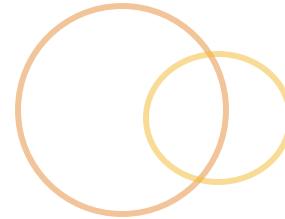
```
return a || a.member; // is like...
if (a) {return a;} else {return b;}
```

Unary Operators



- Take just one operand
 - `typeof`
 - `-` (negative)
 - `!` (negate)

Ternary Operators



- Expects three arguments
- (boolean) ? ifTrue : ifFalse;
 - `var shirt = (isWarm ? "tshirt" : "sweater");`

Statements



- ◉ **Expressions** are the *phrases* whereas **statements** are the *sentences*
 - ◉ A program is a list of statements
 - ◉ Most will require “;” to terminate, though not all.
Best to err on the side of caution and clarity.
 - ◉ **Blocks** are enclosed by { } braces
- ◉ **Control Structures**
 - ◉ Conditional
 - ◉ Switch
 - ◉ Loops

“var” statement



- Declare and initialize a variable in scope
 - Without “var” a variable will be created in the global scope
- No need for type, just a name and optional value
- When no value is given, the default value is undefined
- Can list multiple variables in one statement
 - `var myVar1,
 myVar2=3,
 myVar4;`

Conditional statements



- ◉ `if (expression) {...}`
- ◉ `if (expression) {`
 `...`
 `} else {`
 `...`
 `}`
- ◉ `if {} else if {} else {}`

Switch statements



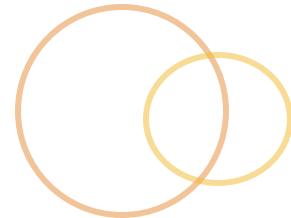
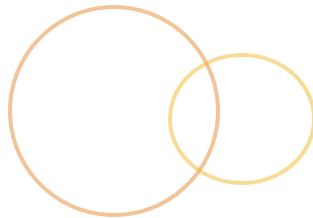
```
switch (expression) {  
    case val1:  
        // statements  
        break;  
  
    case default:  
        // statements  
        break;  
}
```

A word on Falsy vs Truthy



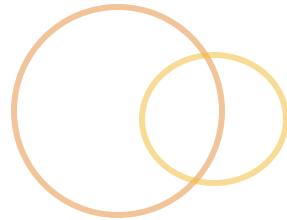
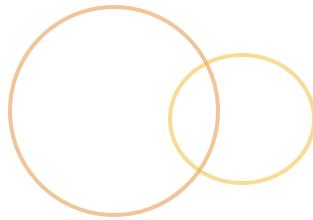
- These are **falsy**
 - `false`
 - `null`
 - `undefined`
 - `""`
 - `0`
 - `NaN`
- Everything else is **truthy**, including...
 - `{}`
 - `[]`
 - `"0"`
 - `"false"`

Loops



- for
- while
- do...while
- for...in
- ~~for each...in~~
- ~~with~~

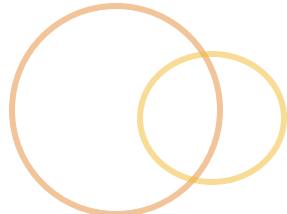
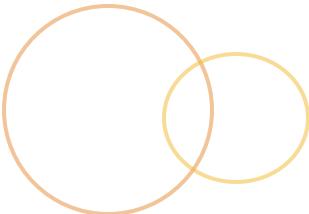
For loop



- for (var i=0; i<10; i++) {
 // executes 10 times...
}

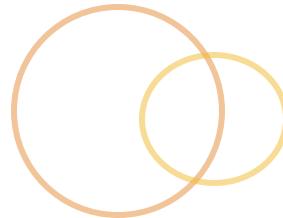
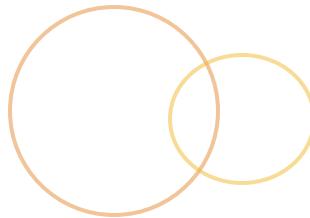


While



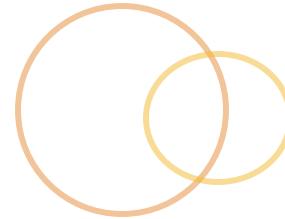
- ◉ `var i=0;`
`while (i<10) {`
 `// ...`
`}`
- ◉ `var i=0;`
`do {`
 `...`
`} while (i<10);`

For...in



- Can enumerate over *enumerable* properties of an object
 - `for (var propertyName in object) {
 // object[propertyName]
}`
 - Will include inherited properties as well, including stuff you probably don't want
 - Use `obj.hasOwnProperty(propertyName)`
- ~~For each...in~~
 - deprecated
 - Iterate over property values
 - `for each (var value in object) {}`

Breaking a loop



- Can break with break;
- Can skip to next iteration with continue;

Control Structures Recap



- Conditionals like `if` and `if-else`
- `Switch` statements
- Iterate (loop) with `while` and `for`
- Enumerate over an object with `for..in`
- Examples
 - <http://jsfiddle.net/mrmorris/GN7qL/>

Exercise – Control flow



- Get to know control flow and iteration statements
- We'll use some basic browser functions
 - `alert("A message!");`
 - `var response = prompt("Ask for a value!");`
 - `confirm("Ask user to say 'ok'");`
- Fork me
 - <http://jsfiddle.net/mrmorris/cxz2hta1/>

Functions



- ◉ Define reusable blocks of code, a new vocabulary
- ◉ “The best part of JavaScript”
- ◉ Function covers responsibility for
 - ◉ Methods
 - ◉ Classes
 - ◉ Constructors
 - ◉ Modules
- ◉ Functions can have their own methods
- ◉ They are *First Class Objects*

Defining a function



- ◉ Three ways
 - ◉ Function **declaration**
 - ◉ Function **expression**
 - ◉ with the `Function()` constructor
 - ◉ `new Function (arg1, arg2, ... argn, functionBody)`
- ◉ <http://jsfiddle.net/mrorris/N8vcg/>

Function Declaration/Statement



- ◉

```
function myFunctionName(param1... paramn)
{
    // function body
}
```
- ◉ When first token is function, it is a statement
- ◉ The function name is mandatory
- ◉ Cannot put a function statement in an if-block because of *hoisting*; it will become available to entire scope

Function Expressions



- Define an unnamed function and assigns it to a variable

- ```
var x = function <name>(parameters,...) {
 // body
}
```

- Name is optional in a function expression

- ```
// stores ref to anonymous function  
var funcRef = function() {};
```
 - ```
// stores ref to named function
var funcRef = function funcName() {};
```

# Function Invocation



- Invoke with ()
  - myFunctionName(argument1, argument2);
- Any number of arguments can be passed in, regardless of defined parameters
- All arguments are available in arguments variable in the function
- Missing arguments will be set as undefined

# Return statements



- `return <expression>;`
- Will return the result of the expression as a result of the function invocation, to the caller of the function
- Constructor functions will return this
- Careful with your line breaks...
  - ```
return
  x;
// Becomes
return;
  x;
```

Function pseudo-parameters



- ➊ Every function has access to special parameters upon invocation
 - ➊ arguments
 - ➊ this

arguments



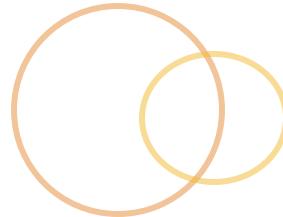
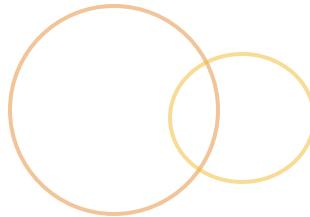
- Available upon invocation
- Includes all arguments passed to the function
- Array-like, but not an actual array
 - Only has length property
- “live”; can affect actual arguments
 - Should be treated as read-only
- To treat like an array, convert it to one...
 - ```
var arr =
 Array.prototype.slice.apply(arguments);
```
- <http://jsfiddle.net/mrmorris/2kpyB/>

# Built-in functions



- ◉ Bug the user, or debug you
  - ◉ `alert(msg);`
  - ◉ `confirm(msg)`
  - ◉ `prompt(msg, msg);`
- ◉ `eval(); // dangerous`
- ◉ Type/Value stuff
  - ◉ `isFinite()`
  - ◉ ~~`isNaN()`~~
  - ◉ `parseInt()`
  - ◉ `parseFloat()`
- ◉ `encodeURI()`, `decodeURI()`

# Scope!



- Scope refers to variable access and visibility in a piece of code at a given time
  - var declares a variable within the current scope
- Lexical Scoping
  - Scope is defined by where a var/function is declared at author time (can determine scope by reading it inline)
- Refer to scope at two levels, *global* and *local*
- No block scope, instead JavaScript has function scope
  - Functions are the only thing that can create a new scope
  - A variable declared (with var) in a function is visible only in that function and its inner functions. But not the other way around.

# Scope continued



- Example of basic scope

```
var a = 5;
function foo(b) {
 var c = 10;
 d = 15; // no var; set in the global scope

 function bar(e) {
 var c = 2; // does not reference c
 a = 12; // will reference a
 }
}
```

- Three scopes exists here
- Each inner scope has access to the outer, *but* the outer scopes cannot access inners
- ReferenceError indicates unfound in scope chain
- <http://jsfiddle.net/mrmorris/YJL9u/>

# Scope matters



- Avoid polluting the global scope
- Supports namespacing (modules) your code
- You can “hide” things by wrapping them in a function
- Closures are born out of using lexical scope
- We’ll see more of this later...

# No block scope...



- ◉ I lied

- ◉ eval() can cheat scoping, inserting itself into the scope
- ◉ In *strict mode* eval() creates its own scope
- ◉ “with” simulates some block scope, but not all – deprecated
- ◉ An exception’s “catch” block has its own scope
- ◉ And in ES6 – “let” establishes a newly scoped var...

# Hoisting – An experiment!



- What will the output be?

- `a = 2;`

- `var a;`

- `console.log(a); // ?`

- And this one?

- `console.log(a);`

- `var a = 2; // ?`

# Hoisting, continued



- *Hoisting* refers to when a variable declaration is lifted and moved to the top of its scope
  - ... only the declaration, not the assignment
- function *statements* are hoisted, too, so you can use them before actual declaration
- JavaScript essentially breaks a variable declaration into two statements
  - `var myVar=0, myOtherVar;`  
// is interpreted as  
`var myVar=undefined, myOtherVar=undefined;`  
`myVar=0;`
- <http://jsfiddle.net/mrmorris/dcvn6735/>

# Exercise: Scope



- Write two functions that share a non-global variable (they log it to the console).

# Functions recap



- ◉ Functions
  - ◉ can be defined with a name or anonymously
  - ◉ are first class objects
  - ◉ create their own scope
  - ◉ Inner scopes can access parent scopes, but not the other way around
- ◉ Declare variables at the top of your scope to avoid hoisting issues

# Back to Objects...



- ◉ Remember that everything is an object except null & undefined
- ◉ Even primitive literals have object wrappers
  - ◉ They remain primitive until used as objects, for performance reasons
- ◉ An object is a dynamic collection of properties
  - ◉ Properties can be functions
- ◉ `this` is a special keyword; inside an object method it refers to the object it resides in

# Creating an object literal



- Create an object literal with {}:

```
var myObjLiteral = {
 name: "Mr Object",
 age: 99,
 toString = function() {
 return this.name;
 }
};
```

- <http://jsfiddle.net/mrmorris/4dsLonat/>

# Object properties



- ◉ Four ways to access and set
  - ◉ Dot Syntax
    - ◉ `myObj.key;`
  - ◉ Square bracket Syntax
    - ◉ `myObj["key"];`
  - ◉ `Object.defineProperty(myObj, "key", desc)`
    - ◉ Takes a descriptor defining property properties
  - ◉ `Object.defineProperties(myObj, "key", desc)`
- ◉ Can delete a property with `delete`

# Object properties descriptors



- Object properties have (usually) hidden descriptors that affect its behavior
  - enumerable
  - configurable
  - Optional...
    - value
    - writable
    - get
    - Set
- `Object.defineProperty(obj, "key", { .. descriptors .. })`

# Object reflection



- `typeof`
- `in` operator
  - `“propertyName” in object`
  - Goes all the way up the chain, not just “own”
- `hasOwnProperty`
  - `myObj.hasOwnProperty(“propertyName”)`

# Object enumerating



- `for...in`
  - Enumerates over enumerable properties
  - And all inherited properties
  - Arbitrary order (not for arrays)
- `Object.keys(obj)`
  - Returns array of all “own”, enumerable properties
- `Object.getOwnPropertyNames(obj)`
  - Returns array of all own property names, including non-enumerable

# Reference



- ◉ Objects can be passed as arguments to functions (and returned)
- ◉ Objects are passed-by-reference, not value
  - ◉ So take care when passing an object into a function that modifies the object
- ◉ `==` operator compares object references, not values. Only true when objects are actually the same object in memory
- ◉ <http://jsfiddle.net/mrmorris/ZLW22/>

# Mutability



- All primitives in JavaScript are immutable
  - Using an assignment operator just creates a new instance of the primitive
  - Pass-by-value
  - Unless you used an object constructor for a primitive...
- Objects are mutable (and pass-by-reference)
  - Their values (properties) can change

# Exercise: Objects



- Create a “copy” function
- Fork me:
  - <http://jsfiddle.net/mrmorris/mLccst8c/>

# Built-in Objects



- String
- Number
- Boolean
- Function
- Array
- Date
- Math
- RegExp
- Error
- <http://jsfiddle.net/mrmorris/rrb67ev0/>

# String Object



- 16 bit unicode characters (UCS-2, not quite UTF-16)
- Single or double quotes are ok
- Similar strings are === equal
- ES5 – Supports multiline string literals w/ “\”  
“my string\  
can have multilines”

# String Properties



- ◉ Properties
  - ◉ length
- ◉ Methods (just a short list)
  - ◉ .charAt(i)
  - ◉ .concat()
  - ◉ .indexOf(needle)
  - ◉ .slice(iStart, iEnd)
  - ◉ .substr(iStart, length)
  - ◉ .replace(regex|substr, newSubStr|function)
  - ◉ .toLowerCase()
  - ◉ .trim()
- ◉ A word on generics (not always available)
  - ◉ String.toLowerCase(arg); // also, Arrays support generics

# Number Object



- 64-bit binary floating point based on IEEE-754
- Aka “double” in Java
- 102, 120.00, .0000000102
- Decimals are approximate – so be careful
  - // given a=0.1, b=0.2, c=0.3  
$$(a+b)+c \neq a+(b+c)$$

# Number Properties



- Properties

- MAX\_VALUE
- NaN
- Etc...

- Generic methods

- Number.isInteger()
- Number.isFinite()
- Number.parseFloat()
- Number.parseInt()

- Instance methods

- num.toString()
- num.toFixed()
- num.toExponential()

# Math Object



- Singleton
- Methods
  - abs, log, max, min, pow, sqrt, sin, floor, ceil, random...
- Constants
  - E
  - LOG2E
  - PI

# Date Object



- An instance of the Date object is used to represent a point in time
- Must be constructed...
  - `var myDate = new Date(); // current date`
  - `var myDate = new Date(ms);`
  - `var myDate = new Date(dateString);`
  - `var myDate = new Date(y,m,d,h,min,sec,ms);`
- Months start at 0, days start at 1
- Timestamps are unix time
  - `myDate.getTime();`

# Date Methods



## ◉ Generics

- ◉ `Date.now(); // current time in ms`
- ◉ `Date.UTC();`
- ◉ `Date.parse("March 7, 2014");`

## ◉ Instance methods

- ◉ `.getFullYear()`
- ◉ `.getMonth()`
- ◉ `.getDate()`
- ◉ `.getHours()`
- ◉ `.getMinutes()`
- ◉ `.getSeconds()`
- ◉ `.getYear()`
- ◉ `.setYear(yearValue)`

# Regular Expressions



- ◉ Patterns used to match character combinations in strings
- ◉ Very tough to understand but extremely powerful
  - ◉ `^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$`
- ◉ Creation
  - ◉ `// Constructed`  
`var re = new RegExp("abc");`
  
  - ◉ `// Literal`  
`var re = /abc/;`
- ◉ <http://jsfiddle.net/mrmorris/bs145x7r/>

# Regular Expression object



- Methods
  - `re.exec(str)`
  - `re.test(str)`
- Also used in string methods...
  - `str.match()`
  - `str.replace()`
  - `str.search()`
  - `str.split()`

# Regular Expressions matchers



- Escape with / backslash
- Use [] for sets, [01234] or [0-4] for a range
- Special character groups, ex: \d (digits) and \w (alphanumeric)
- + match at least one
- \* match 0 or more
- ^ invert
- ? Optional - /neighbou?r/
- {4} time occurrence
- {2,4} – at least twice, at most 4 times

# Regular Expression continued...



- ◉ /<sup>^</sup>...\$/ for start and end of a line
- ◉ | pipe for choices (pig|cow|chicken)
- ◉ i, g - modifiers
- ◉ () for grouping and remembering matches
  - ◉ /boo+(hoo+)+/i
  - ◉ When using () grouping, the matches in these groups will be included in the array
    - ◉ The whole match is the 1<sup>st</sup> element
    - ◉ The next element is match of first group, and so on
    - ◉ A non-matched group shows as undefined in the array
- ◉ Use a cheatsheet or <http://regexpal.com/>

# Arrays



- Arrays are objects that behave like arrays
  - Indexes of an array are converted to strings and set as object properties
- Use arrays when order of the data should be sequential
- Creating an array
  - // literal

```
var myArray = [1,2,3];
```
  - // constructor

```
var myArray = new Array(1,2,3);
```
- <http://jsfiddle.net/mrmorris/at6djrey/>

# Array properties



- Properties
  - `length`
- Methods
  - `Array.isArray()`; // ES5
    - Because `typeof arr` is “object”
  - `.sort()`
  - Search
    - `.indexOf(val)`
    - `.lastIndexOf(val)`
  - ...

# Array Modifications



- Insert
  - `.shift()`, `.push()`
- Remove
  - `.unshift()`, `.pop()`
- Combine
  - `.concat()`
- Extract
  - `.slice(starti, endi)`
  - `.splice(starti, removeCount [, newEl1, newEl2])`
    - From start, remove `removeCount`, then add `newEls`

# Array enumeration



- ◉ Use “for” not “for...in”, which doesn’t keep array keys in order
  - ◉ 

```
for (var i=0; i < myArray.length; i++) {
}
}
```
- ◉ `.forEach(callback[, thisArg])`
  - ◉ New in es5 (ie9+)
  - ◉ No way to stop or break a forEach loop
  - ◉ 

```
myArray.forEach(function(val, index, arr) {
});
```

# Array tests



- ◉ `arr.every(callback[, thisArg])`
  - ◉ checks If every element in an array passes a callback function
  - ◉ `myArray.every(function(val, index, arr) {  
 return (val>0); // evaluates to boolean  
});`
- ◉ `arr.some(callback[, thisArg])`
  - ◉ verify if at least one passes the test

# Array filter, map, reduce



- ◉ **.filter()**
  - ◉ Iterate over your array of items passing them to a function. Returning true from the function indicates the item should be retained.

```
myArray.filter(function(item) {
 return item!=2;
}); // removes items that don't equal 2
```
- ◉ **.map()**
  - ◉ Iterates over array, invoking a function on each value. The return value is the modified value of the item.
- ◉ <http://jsfiddle.net/mrmorris/pbwY3hy5/>

# Array reduce()



- ◉ .reduce()
  - ◉ Boils down a list of values into a single value.
  - ◉ Iterates over array, passing a *memo array* as it iterates through.
  - ◉ The callback takes previousValue, currentValue, index and array
    - ◉ [0,1,2,3,4].reduce(function(previousValue, currentValue, index, arr) {  
 return previousValue + currentValue;  
 }, optionalInitialValue);
  - ◉ previousValue starts at initial, then becomes the return value of each iteration.

# Constants



- Not frequently used
- Immutable while script thread is running
- Same rules as apply to variables, but keyword 'const' is used instead of 'var'
- They **\*are\*** scoped

# Errors



- <http://jsfiddle.net/mrmorris/L2tj2z70/>
- Errors in JS throw an Exception, at which point the code halts and looks for exception handling
- Exception handling
  - try, catch, throw
  - finally
- Error object thrown:
  - {  
    name: string,  
    message: string,  
    fileNumber: number,  
    lineNumber: number  
}

# Built-in Exceptions



- Error
- SyntaxError
- ReferenceError
- TypeError
- RangeError
- URIError
- EvalError

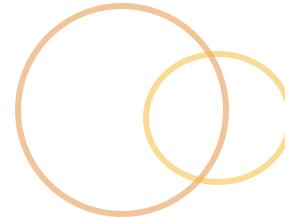
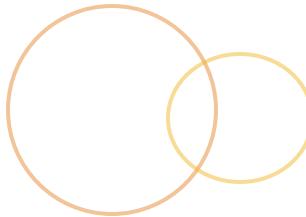
# The global object



- The scope in which all variables live by default
- The context of the application
- “window” when running in a browser



ES5...



- Ignore trailing commas
- No reserved word restrictions on prop names
- Built in getter/setter property settings
- Infinity, NaN and undefined are now constants
- JSON is now native
- Now has a “strict mode”
- Many new methods!
  - Function.prototype.bind()
  - String.prototype.trim()
  - Array.prototype.every(), filter, map, reduce
  - Object.keys()
  - Object.create()
  - Array.isArray()

# Strict Mode



- ◉ "use strict";
- ◉ It kills deprecated features from ECMAScript 3
- ◉ It catches some common coding bloopers, throwing exceptions.
- ◉ It disables features that are confusing or poorly thought out.
  - ◉ Ensures “eval” has its own scope
  - ◉ Does not auto-declare variables at the global level during a scope-chain lookup
- ◉ Can be set globally or within function block

# Object immutability



- ◉ `Object.freeze(obj)`
  - ◉ Freezes an object: other code can't delete or change any properties
  - ◉ <http://jsfiddle.net/mrmorris/auuzgaxr/>
- ◉ `Object.preventExtensions(obj)`
  - ◉ Prevents any extensions of an object
  - ◉ <http://jsfiddle.net/mrmorris/0mn20b5L/>
- ◉ `Object.seal(obj)`
  - ◉ Prevents other code from deleting properties of an object
  - ◉ <http://jsfiddle.net/mrmorris/jmqguegw/>
- ◉ `Object.defineProperty(obj, propName, definition)`
  - ◉ Defines (or updates) a property on an object
  - ◉ <http://jsfiddle.net/mrmorris/g6t0hywp/>

# JavaScript Best Practices



- Avoid polluting the global namespace
- Define variables at top of your scope
- Use === & !== strict comparison
- Use named function expressions to avoid accidental hoisting
- Avoid primitive object wrappers like Number() or String()
- CamelCase constructor functions
- Include implicit ;
- Always open and close blocks { }
- Indent

# Exercise – Core objects



- Reverse an array
  - <http://jsfiddle.net/mrmorris/zqfwy1xp/>
- Strings and Dates
  - <http://jsfiddle.net/mrmorris/vk2tg472/>
- Data grids in the console
  - <http://jsfiddle.net/mrmorris/0kptbv7p/>
- Don't forget devdocs.io!

# Function patterns



- Closures
- Immediately invoked functions
- Anonymous Functions
- Chaining
- Callbacks
- Currying

# Closures



- ◉ One of the most important features of JavaScript
- ◉ And often one of the most misunderstood & feared features
- ◉ But... they are *all around you* in JavaScript
- ◉ They happen when you write code that relies on lexical scope

# Closures, continued



- ◉ “When the context of an inner function includes the scope of the outer function and the inner function enjoys the context even after the parent function has returned”
- ◉ When a function is able to remember and access its lexical scope, even when executing outside its lexical scope.
- ◉ When an inner function *closes over* the variables of an outer function
- ◉ It retains state and scope after it completes execution
- ◉ And... the inner function is used outside the outer

# Closures are easy...



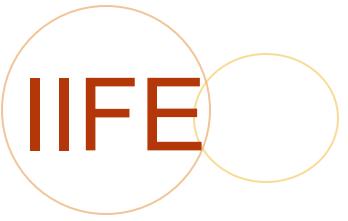
- Most basic example

```
○ function closeOverMe() {
 var a=1;
 return function() {
 alert(a);
 };
};
var witness = closeOverMe();
witness(); // closure witnessed!
```

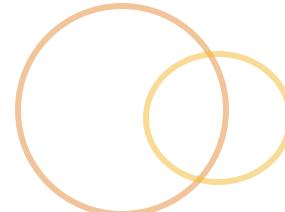
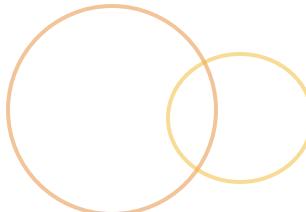
- With closures, functions have access to variables that were available in the scope where the function was created

- Examples

- <http://jsfiddle.net/mrmorris/974U2/>



IIFE



- ◉ Immediately Invoked Function Expressions
- ◉ Define the function and immediately invoke it
  - ◉ 

```
var tasksController = function() {
 // body
}(); // there it is!
```
- ◉ Anonymous? Sure...
  - ◉ 

```
(function(){
 ...
})();
```
  - ◉ 

```
(function(){
 ...
})(); // also works
```
- ◉ <http://jsfiddle.net/mrmorris/u5srzzgk/>

# Anonymous functions



- A function expression without a name
  - `var anon = function() {}`
- IIFE
- Powerful
  - Functions can be passed as arguments
  - Defined inline
- But...
  - difficult to test in isolation
  - Discourages code re-use

# Name your anonymous functions



- It can be a good idea to name your anonymous functions

- ```
(function myAnonFunc() {  
    // myAnonFunc();  
})();
```

- Why?

- “myAnonFunc” is scoped to the function inner so it can iterate on itself
- Easier to debug; errors reference the function name

Function callbacks



- When a function is provided as an argument as something to be invoked inline, or under specific circumstances (like an event)
- ```
function runCallback(callback) {
 // does things
 return callback();
}
```

# Timers



- Establish delay for function invocation
  - `setTimeout(func, delayInMs[, arg1, argn])`
    - `var timer = setTimeout(func, 500);`
  - Use `clearTimeout(timer)` to cancel
- Establish an interval for periodic invocation
  - `setInterval(func, ms)`
  - `clearInterval(timer)`
- Context will always be global for the callbacks
- <http://jsfiddle.net/mrmorris/s5g2moc6/>

# Callbacks and closures



- Careful with function expressions in loops
  - Can have scope issues
    - ```
for (var i=0; i<3; i++) {  
    setTimeout(function(){  
        console.log(i);  
    }, 1000*i);  
} // what will this output?
```
 - Instead, create an additional scope to maintain state for the inner function (expression)
 - Closures save the day
 - <http://jsfiddle.net/mrmorris/e8n62r3w/>

Function Chaining



- When a function returns a reference to its context, “this”, it can chain to other methods on the same object
- <http://jsfiddle.net/mrmorris/aA24r/>

Currying and Partials



- When you create a new function from an old function, modifying the arguments.
- <http://jsfiddle.net/mrmorris/aq5fL6xx/>

Lazy Function Definition



- ◉ A pattern for defining a function that may or may not be initialized... and avoids initializing when not used
- ◉ Assigns itself into the caller on invocation to avoid “immediate invocation” and save some memory
 - ◉

```
var digit_name = function(n){  
    var names = [...];  
    digit_name = function(n) {  
        return names[n];  
    }  
    return digit_name(n);  
}
```

Closure Conditional



- Better than the lazy function definition...

- ```
var digit_name = (function(){
 var names;
 return function(n) {
 if (!names) {
 names = [...];
 }
 return names[n];
 }
}());
```

# let Scope



- ES6 introduces “let”
  - Cousin to “var”, declare a variable in the scope of containing block, {}
  - ```
if (expression) {  
    var a = 1; // scoped to wrapping function  
    let b = 2; // scoped to the block  
} // woah.
```
- Careful, its not always obvious
 - Use an explicit block to make it clear
 - ```
if (expression) {
 {
 let b = 1;
 }
}
```

# let and hoisting



- It does not *hoist*

- {  
    console.log(b); // b does not exist yet  
    let b = 12;  
    console.log(b); // works fine  
}

# For let loops



- Using let with a for loop is possible in ES6
  - ```
for (let i=0; i<10; i++) {  
    // i is bound to a new scope each iteration  
    // getting its value reassigned  
    // at the end of the iteration  
}
```
 - The above is like...
 - ```
let i = 0;
for (j = 0; j<10; j++) {
 let i = j;
 // ...
}
```
- Why is this cool? ...

# Let loops and closures



- It's cool because now we don't need to manually create new scopes
  - ```
for (let i=0; i<3; i++) {  
    setTimeout(function(){  
        console.log(i);  
    }, 1000*i);  
}
```
- Remember the issue with functions in a loop?
 - Sorry! Won't actually work yet...
 - <http://jsfiddle.net/mrmorris/ssav4pfh/>

Function patterns recap



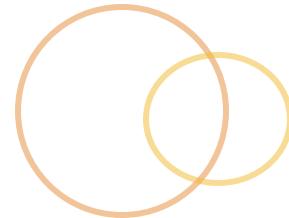
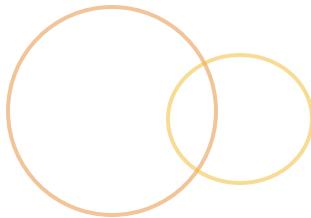
- ◉ Mind your scope!
 - ◉ In callbacks, in particular
- ◉ Closures create a persistent and private scope
- ◉ Functions are often passed around as callbacks
(due to the nature of the event loop...)
- ◉ Function expressions can be immediately invoked (and you'll see this a lot)

Exercise: Closures



- Month names function
 - <http://jsfiddle.net/mrmorris/y37qch2g/>

Day 2



- ◉ Any remaining set up issues?
- ◉ Any questions from day 1?
- ◉ Any comments/suggestions?
- ◉ For today:
 - ◉ JavaScript Core: Objects and Prototypes
 - ◉ HTML, CSS refresher
 - ◉ The DOM
 - ◉ Events
 - ◉ Native Ajax
 - ◉ jQuery basics

Warm-up Exercise



- ◉ Create an object that represents yourself
 - ◉ Give it a name (ex: Ryan)
 - ◉ A height? Age? Whatever
- ◉ You should be able to do stuff, like speak
 - ◉ Create a function on your object, “speak”, accepts a string argument and logs it to the console as:
 - ◉ “Ryan says: {the string}”
- ◉ Let’s assume you also have a trophy collection
 - ◉ Create an array property on your object that has a couple trophies inside it
 - ◉ Add a function that lets you view trophies:
 - ◉ `me.viewTrophy(i);` // will log the name of the trophy at i index
 - ◉ Add a function that lists all trophies, separated by commas
 - ◉ `me.listTrophies();` // -> “gold star, track, silver medal”
- ◉ How might we make it so that your trophies are protected from theft?
 - ◉ Hint: IFFE, Closures, and entering into modules...

Scope & Context



- ◉ We already discussed Scope
 - ◉ Determines visibility of variables
 - ◉ Lexical scope (write-time)
- ◉ There is also Context
 - ◉ Context refers to the location a function/method was invoked from
 - ◉ Sort of dynamic scope; defined at run-time
 - ◉ Context is referenced by “this”



“this” is the context



- ◉ this keyword is a reference to the “object of invocation”
 - ◉ Bound at invocation
 - ◉ Depends on the call-site of the function
- ◉ It...
 - ◉ allows a method to know what object it is concerned with
 - ◉ allows a single function object instance to service many functions/usages
 - ◉ is key to inheritance
 - ◉ gives methods access to their objects

“this” is determined by call-site



- this is *bound* at call-time to an object
- 1) Default binding
 - Global
- 2) Implicit binding
 - Object method
 - Warning: Inside an inner function of an object method it refers to the global object
- 3) Explicit binding
 - Set with .call() or .apply()
- 4) Hard binding
 - Defined by .bind()
- 5) “new” binding
 - When *constructing* a new object
- <http://jsfiddle.net/mrmorris/RUNS5/>

More explicit binding



- ◉ Context can be changed, which affects the value of this, via Function's call, apply and bind methods
 - ◉ `obj.foo(); // obj context`
`obj.foo.call(window); // window context`
- ◉ “bind” doesn’t execute, it returns a copy of the function with the context re-defined. The resulting function is a “bound function”
 - ◉ `var getX = module.getX;`
`boundGetX = getX.bind(module);`
- ◉ <http://jsfiddle.net/mrmorris/or7y5orn/>

Inheritance in JavaScript

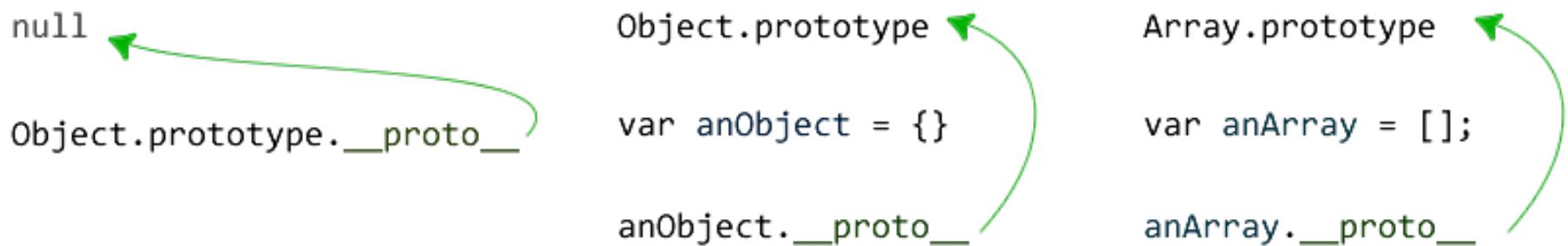


- ◉ Or.. How I learned to love the Prototype
- ◉ JavaScript is class-free, prototypal
- ◉ There is a way to simulate classical inheritance, but prototypal approach is more suited to JavaScript
- ◉ Prototype modal
 - ◉ Tends to be smaller
 - ◉ Less redundant
 - ◉ Can simulate classical inheritance as needed
 - ◉ More powerful

Prototypes (intro)

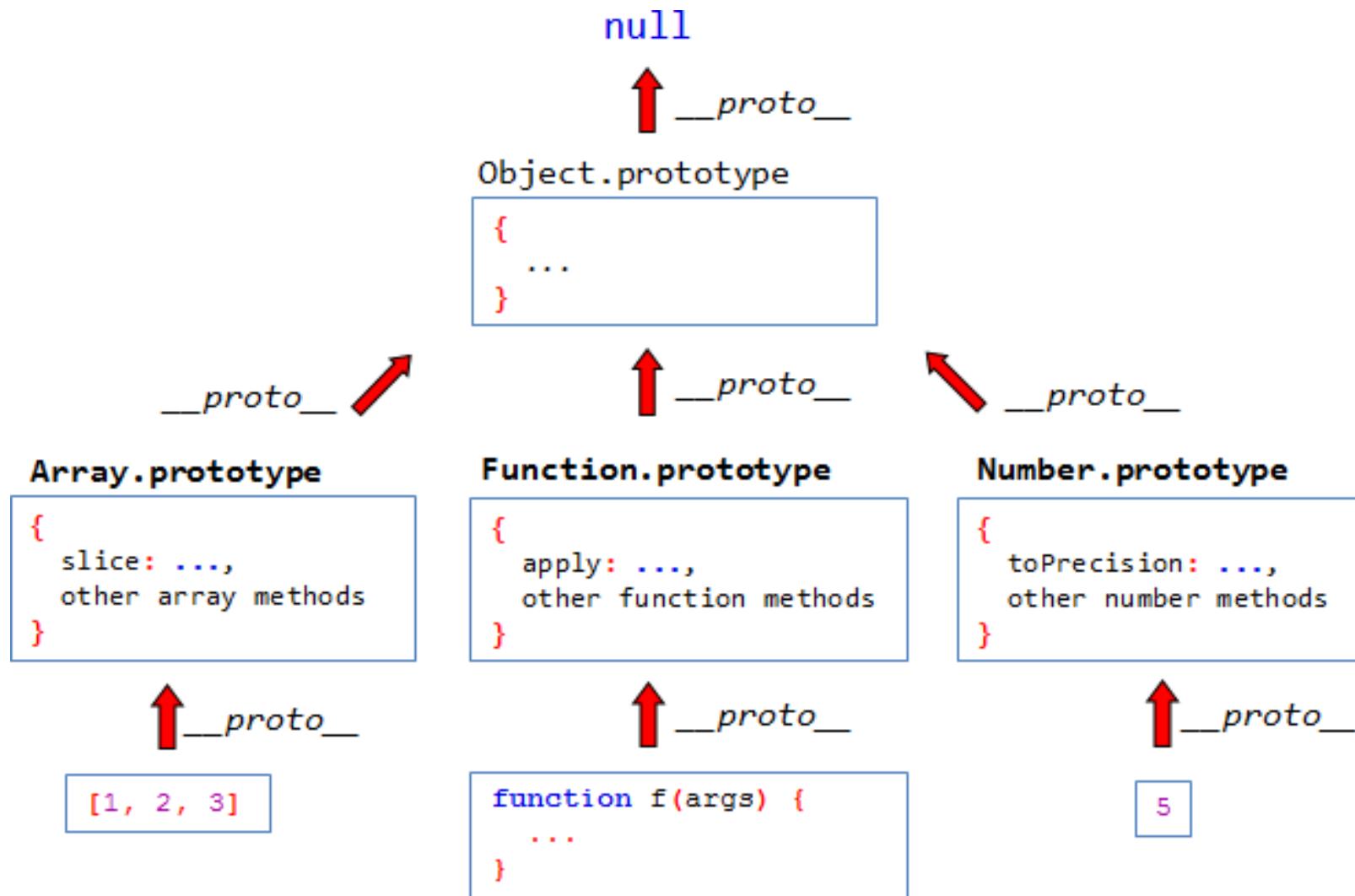


- Objects have an internal link to another object called its *prototype*
- That prototype has a prototype, and so on, up the *prototype chain*



- The chain goes up to **Object.prototype** and then **null** (the end of the chain)
- Objects **delegate** behavior to other objects through this prototype linkage

This is how objects inherit their stuff



Let's return to objects...



- There are three common ways to create new objects in JavaScript
 - Object Literal
 - `var newObj = {};`
 - `Object.create(prototypeObj)` - ES5
 - `var newObj = Object.create(Object.prototype);`
 - Constructor Function
 - `var newObj = new Object();`

Native Constructor Pattern



- Tends to simulate classical inheritance
- Relies on “new” keyword and “this”

```
○ function Car(color) {  
    this.color = color;  
    this.toString = function() {  
        return this.color + " car";  
    };  
};  
var toyota = new Car("red");
```

- It...
 - Creates a new object
 - Invokes the function with that object as the context
 - Sets the prototype of the new object to the Constructor's prototype
 - Returns the new object

Native Constructor continued



- Can make the Car instances more flexible by utilizing its prototype property

- ```
function Car(color) {
 this.color = color;
}
```

```
Car.prototype.toString = function() {
 return this.color + " car";
}
```

```
var toyota = new Car("red");
// toyota.__proto__ -> Car.prototype
```

# Augmenting



- The **prototype** behaves like a live template for objects that link to it in the chain
- (*In the Constructor Pattern*) you can continue to augment instances through its Constructor's prototype
  - `Car.prototype.drive = function(){}`

```
// now toyota will immediately have this
// available
```

```
toyota.drive();
```

# Own properties



- An object can have its “own” properties, referenced within the object itself
- It can also have properties it delegates up the prototype chain
- Continuing our Car example...
  - `toyota => {color: "red"}`
  - `toyota.red;` // is its own property (thanks to the constructor)
  - `toyota.drive();` // is delegated up the chain
  - `Car.prototype.tires = 4;` // now augment
  - `toyota.tires;` // works through delegation up the chain

# Prototype



- Every function in JavaScript has a `prototype` property, which is by default an empty object.
- Every object in JavaScript has a internal reference to it's prototype (`__proto__`)
  - Better said as, “A reference to the object’s prototype that it inherits from”
- Objects also have a `constructor` property
  - What constructor function made me?

# Prototypes continued...



- ◉ All objects inherit from `Object.prototype`
  - ◉ The `Function` object inherits from `Function.prototype`
- ◉ Prototype is like a fallback object of properties that an object can use when the property does not exist on the object itself
- ◉ When you request a property of an object, it checks the object, then its prototype, then the prototype's prototype, and so on...
  - ◉ The prototype chain
- ◉ This is the basis of inheritance in JavaScript

# Object creation inheritance



- ◉ Literal
  - ◉ `var obj = {};`
- ◉ Constructor function
  - ◉ `var obj = new Object();`
- ◉ `Object.create`
  - ◉ `var obj = Object.create(Object.prototype);`

# Pseudo-classical Inheritance



- ◉ Basically...
  - ◉ Create a Constructor Function
    - ◉ 

```
function Gizmo(id) {
 this.id = id;
}
```
  - ◉ Add methods and properties that will be inherited
    - ◉ 

```
Gizmo.prototype.toString = function() {
 return "gizmo " + this.id;
}
```
  - ◉ Allow inheritance via prototype
    - ◉ 

```
function Hoozit(id) {this.id=id;}
Hoozit.prototype = new Gizmo();
Hoozit.prototype.test = function(){...}
```

# Simulated Classical Inheritance



- Native implementation
  - <http://jsfiddle.net/mrmorris/2MRXh/>
- Douglas Crockford's implementation
  - <http://www.crockford.com/javascript/inheritance.html>
  - <http://jsfiddle.net/mrmorris/52Tkb/>
- John Resig's implementation
  - <http://ejohn.org/blog/simple-javascript-inheritance>
  - <http://jsfiddle.net/mrmorris/gxwur/>

# Prototypal Inheritance



- ◉ There are no “classes”, only objects
- ◉ All objects “inherit” methods and properties from other objects
- ◉ Objects are treated as dynamic templates
- ◉ How to do it
  - ◉ `Object.create(prototype); // ES5`
- ◉ Prototypal inheritance is arguably “better”
  - ◉ Allows for multi-inheritance (mixins)
  - ◉ Considerably simpler than classical patterns

# Object.create()



- Basis of “real” prototypal inheritance
- Creates an object with a specified prototype and optional properties
- `Object.create(prototype, properties);`
- What it is:
  - ```
function object(o) {  
    function F(){};  
    F.prototype = o;  
    return new F();  
}
```
- Car example
 - <http://jsfiddle.net/mrmorris/f0tsos80/>
- Advanced examples (if we want)
 - <http://jsfiddle.net/mrmorris/uc3k80k1/>

Parasitic Inheritance



- When a function is used to create an object, augment it, and return the augmented object as though it constructed it.
- <http://jsfiddle.net/mrmorris/c6y5agLx/>

Functional Inheritance



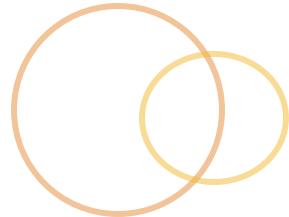
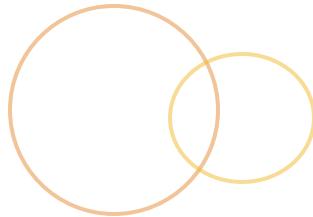
- ◉ A constructor-like function is responsible for creating an object w/ its properties.
- ◉ Other constructor-like functions can build on the original constructors.
- ◉ It simulates private members and supports shared secrets across related objects
- ◉ <http://jsfiddle.net/mrmorris/8pMy3/>

Super methods



- Can simulate a super method (or parent method) by capturing the method and making it available to the instance object
- Not really necessary, but doable
- ```
Function hoozit(id) {
 var secret={};
 var that=gizmo(id, secret);
 var super_toString = that.toString;
 //...
 return that;
};
hoozit.super_toString();
```

# Mix-ins



- Several approaches...
  - <http://jsfiddle.net/mrmorris/t4veurm5/>
- Extend objects
- Functional mix-ins
  - Use “call” and “apply” method
  - ```
function logged(){this.id=5;};  
function Item() {  
    logged.call(this); // mix-in  
}  
Item.prototype = Object.create(null);
```

introspection



- ◉ `instanceof`
 - ◉ Tells you whether an object was derived from a specific constructor
 - ◉ `anObj instanceof FooConstructor`
- ◉ `Object.isPrototypeOf`
 - ◉ In the prototype chain of `a`, does `Foo.prototype` appear
 - ◉ `Foo.isPrototypeOf(anObj)`
- ◉ `Object.getPrototypeOf()`
 - ◉ Returns the actual prototype (`__proto__`)
 - ◉ Must be an object in ES5, ES6 correctly returns `String.prototype` for “`astring`”

Objects and prototype recap



- Objects are dynamic
- Inheritance behaves like delegation/linking
 - You can use an object as a template for another object ad infinitum
- Object properties (and methods) are looked for up the **prototype chain**
- `proto` is a special property linking an object to its prototype
- `prototype` is a property you set on constructors
- **this** in an object method references the object
 - But it can change, depending on how you call the function

Inheritance w/ constructors



- MyFunction is treated as a constructor function
 - `var MyFunction = function() {}`
- Its prototype property is an object that “instances” will link to up the prototype chain
 - `MyFunction.prototype = {prop:val};`
- “Instances” of the MyFunction have `__proto__` set as MyFunction.prototype
 - `var newObj = new MyFunction();`
`// newObj.__proto__ -> MyFunction.prototype`
- Continue inheritance by linking constructors
 - `var MyChildFunction = function() {}`
`MyChildFunction.prototype = new MyFunction();`
`var newChildObj = new MyChildFunction();`

Inheritance w/out constructors



- An object is treated as a template (or delegate) for another object
 - `var topObj= {name: "Main"};`
- We use `Object.create()` to create objects that link up to their prototype
 - `var midObj= Object.create(topObj);`
`var botObj = Object.create(midObj);`
 - proto is a special property linking to each objects prototype
 - `botObj.__proto__ -> midObj`
`midObj.__proto__ -> topObj`
`topObj.__proto__ -> Object.prototype`

Delegation > inheritance



- JavaScript inheritance is more like “delegation” rather than classical inheritance which relies on copies
- Really is no need to use “new” or constructor functions
 - ```
var foo = {...};
var bar = Object.create(foo);
var b1 = Object.create(bar);
var b2 = Object.create(bar);
```

# Exercise: Inherit something



- Using the “native constructor” technique, create a series of objects that inherit from their predecessors
- Choose a subject where there are plenty of types to pick from:
  - Animal -> Mammal -> Horse -> Clydesdale
  - Vehicle -> Automobile -> SUV -> Jeep
  - Profession-> Engineering -> Software Architect
  - Etc.
- No need to make several of each tier, just one inheritance chain is enough
- Feel free to add properties and methods that seem appropriate
- Now try the same with Object.create() (instead of constructors)

And... a few other design patterns



- Modules
- Singletons
- Lazy function definition(s)

# Modules



- Help to organize logical units of functionality
- Several options for module implementation
  - Object literal notation
  - The Module pattern
  - AMD Modules
  - CommonJS modules
  - ECMAScript Harmony modules
- Prevent namespace clutter

# The module pattern



- Allows for “private” methods and functions
- Useful for singletons or our own global namespace
- Uses an anonymous closure to wrap private functionality and return the public interface
- <http://jsfiddle.net/mrmorris/8gwtvv99/>

# The module pattern



- var module= (function(){  
    var privateVar;  
    function privateFunction() {...}  
  
    return {  
        publicVar: “bla”,  
        publicMethod: function() {  
            // can access privates  
        }  
    };  
})(); // immediately invoke!

# Advanced Module Patterns



- Import: Can import libs by providing as arguments
  - ```
var myModule = (function($){  
    // use $ for jQuery  
}(jQuery));
```
- Augmentation: Use a module to augment another object
 - Loose augmentation
 - Tight augmentation
- Revealing module pattern
 - Uses all private members and a single object interface referencing privates

Singleton pattern



- Restricts instantiation to a single instance
- Stores a reference to itself and returns that if it exists
- <http://jsfiddle.net/mrmorris/qruLc51s/>

Exercise: Modules



- ◉ Let's revisit the "me" object you created earlier
- ◉ Set it up as a module so that no properties are publicly accessible
 - ◉ You'll use an IIFE
 - ◉ You'll return an object that acts like your public API
 - ◉ The object will include your getters and setters
- ◉ Make it so that:
 - ◉ Your trophy collection can not be removed or deleted-from
 - ◉ Your trophy collection can be added to (with a new method)
 - ◉ Your trophy collection can still be viewed (viewTrophy(i), listTrophies())
 - ◉ Every time someone views your trophy collection you count a view

Continuing JavaScript



- JavaScript: the Good Parts (Crockford)
 - Great re-introduction to the language and common pitfalls
- You Don't Know JS series (Simpson)
 - Look at JavaScript in a new light
 - <https://github.com/getify/You-Dont-Know-JS>
- Learning JavaScript Design Patterns (Osmani)
 - Thorough book about design patterns found in JavaScript
 - <http://addyosmani.com/resources/essentialjsdesignpatterns/book>

JavaScript and the Browser



○ Why JavaScript

- Interactivity, based on user or browser triggered events
- Load data into the page dynamically
- Built in business logic
- Single page applications (if that's your cup of tea)
- The web is the new application platform...

○ How it fits

- HTML for view data & ui structure
- CSS for presentation
- JavaScript for behavior
 - Though.. becoming central for business logic

Wizard check



- Write an HTML form from scratch?
- Style a form (or full page) from scratch?
- Manipulate elements in the page with just the DOM?
- Set up an event handler for form submissions?
Clicks?
- Know what events are and why they are important?

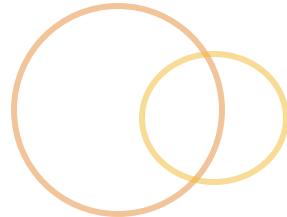
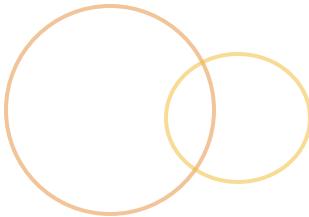
Browser debugging w/ Console



- ◉ “console” object
 - ◉ Typically on “window”, though doesn’t always exist
 - ◉ Methods
 - ◉ log
 - ◉ dir (lists all properties)
 - ◉ info
 - ◉ warn
 - ◉ error
 - ◉ table(object)
 - ◉ group(name); groupEnd();
 - ◉ assert(bool, message); // shows only if true
- ◉ debugger; // breakpoint
- ◉ <http://jsfiddle.net/mrmorris/X76Gq/>



HTML



- Hyper Text Markup Language
- 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 an html page



◎ The document

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

◎ <http://jsbin.com/retupe/edit?html>

Anatomy of an html element



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

HTML Elements refresher



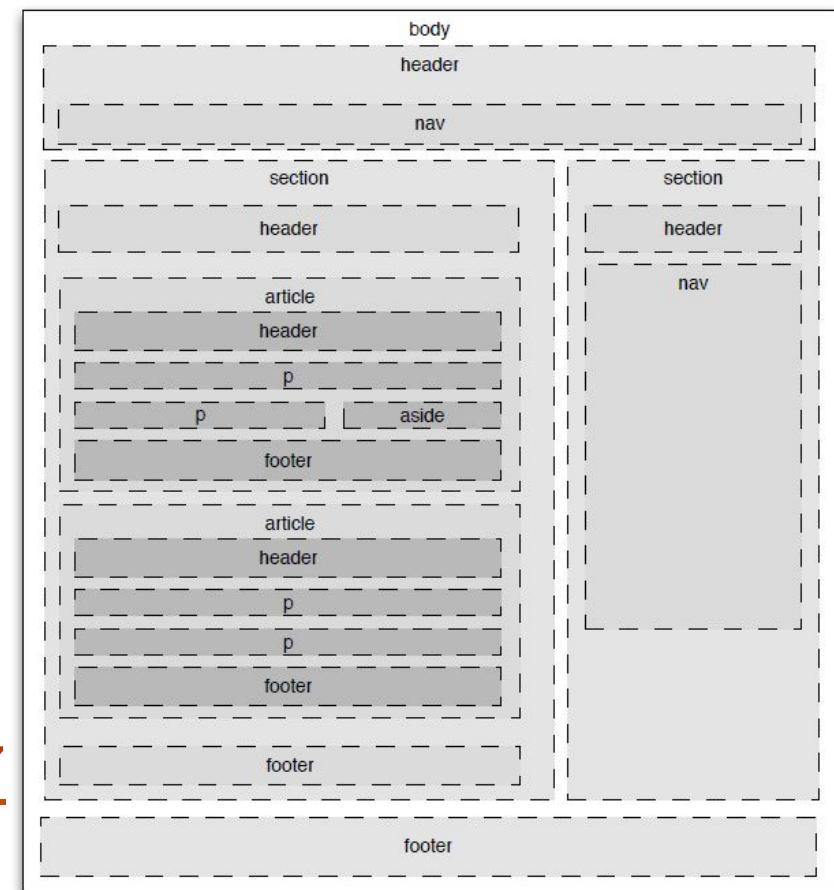
- Structure
 - <div>
 -
 - <table>
 - <tr>, <td>, <thead>, <tbody>
 - <form>
 - <fieldset>, <label>, <input>, <select>, <textarea>
 - And some new HTML5 semantic elements
- Content
 - <h1> through <h6>
 - <p>
 - or (with)
 - Text modifiers
 - ,
- See:
 - <https://developer.mozilla.org/en-US/docs/Web/HTML/Element>

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>

<http://jsfiddle.net/mrmorris/cb47>



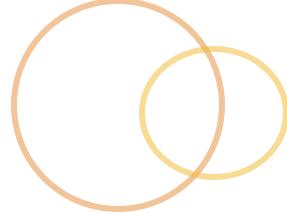
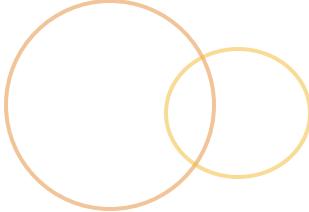
HTML5 Forms



- New input types
 - number
 - range
 - url
 - email
 - tel
 - color
 - Search
- New element
 - datalist
- New input attributes
 - required
 - autofocus
 - placeholder
 - List
- Built in validation
 - Use “novalidate” to turn off
- <http://jsfiddle.net/mrmorris/zh18vn4x/>



css



- Cascading Style Sheets
- Language for describing look and formatting of the document
- Separates presentation from content
- Define as external resource or inline
 - <link rel="stylesheet" type="text/css" href="theme.css">
 - <style type="text/css">
...style declarations...
</style>

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 {}` `<div id="header"></div>`

- By class

- `.main {}` `<div class="main"></div>`

- By attribute

- `div[name="user"] {}` `<div name="user"></div>`

- By relationship to other elements

- `li:nth-child(2) {}` ``

CSS Specificity



- ◉ Multiple selectors will override each other based on “specificity”
 - ◉ <div id=“main” class=“fancy”>
What color will I be?
</div>
 - ◉

```
#main{  
    color: #f90;  
}  
.fancy{  
    color:blue;  
}  
#main.fancy{  
    color:red;  
}
```
- ◉ Order of specificity: inline, id, psuedo-classes, attributes, class, type, universal
- ◉ !important

New Selectors & Classes



- Attribute Selectors
 - <http://jsfiddle.net/mrmorris/tp6t6skt>
- Sibling Selector
 - <http://jsfiddle.net/mrmorris/98jg21y3/>
- Pseudo-classes
 - Form inputs
 - <http://jsfiddle.net/mrmorris/nqsbj80o/>
 - Structural
 - nth/first/last/only
 - :empty
 - :not
 - <http://jsfiddle.net/mrmorris/ghddq4eu/>

Including JS on the page



- ◉ HTML Parser parses while script is downloaded then run
- ◉ Include at the bottom of </body>
 - ◉ It won't block
 - ◉ All DOM is loaded
- ◉ <script>...</script> blocks
- ◉ <script src="filename.js"></script>
- ◉ Inline on elements is possible "onclick"

How JavaScript loads (Performance)



- JS loading is blocking...
- Browser will download then interpret any JS it comes across before proceeding
- So... put scripts in files and at bottom of the body tag

The DOM



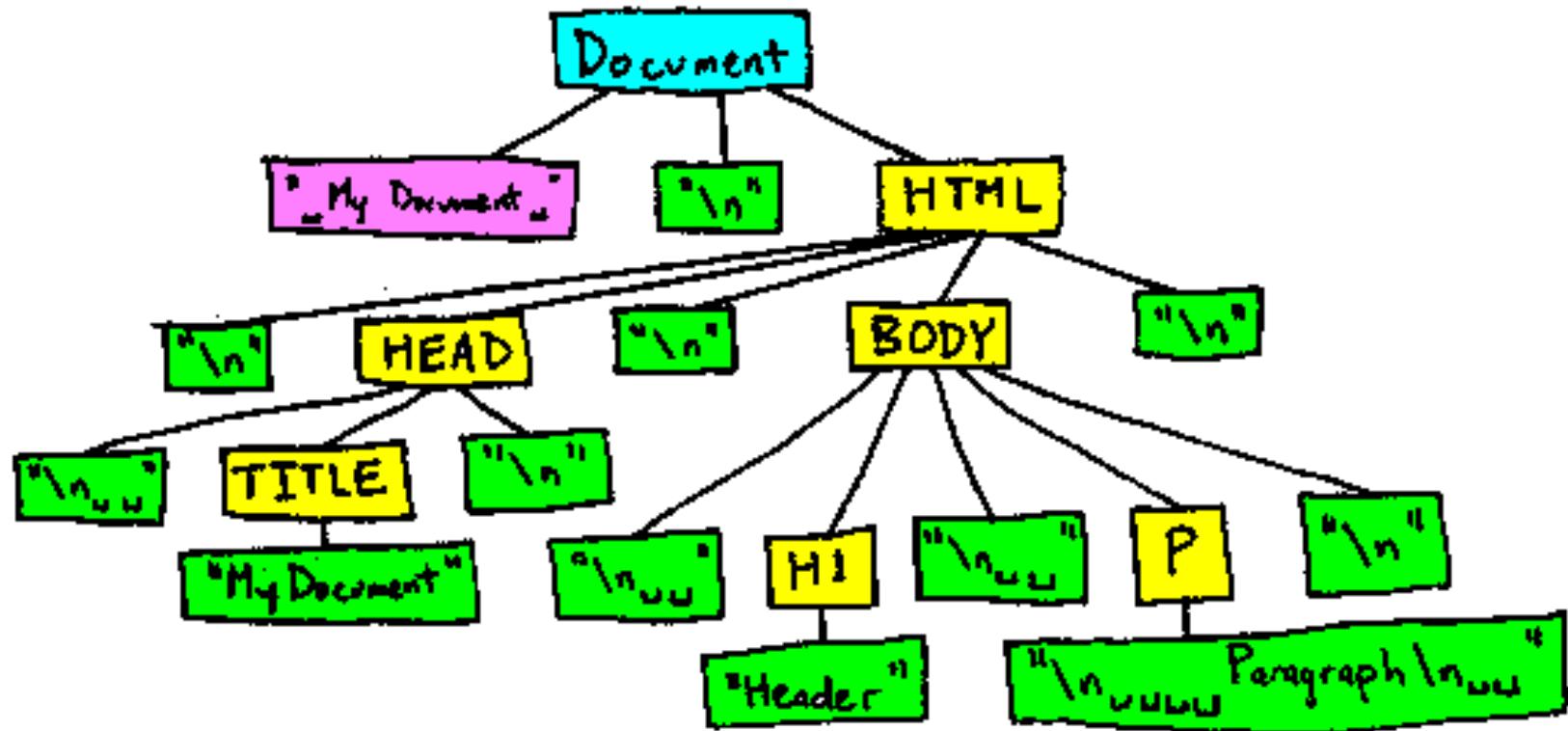
- Document Object Model
- What most people hate when they say they hate JavaScript
- The browser's API: interface it provides to JavaScript for manipulating the page
- Browser parses HTML and builds a model of the structure, then uses the model to draw it on the screen
- “Live” data structure

Document structure



- ◉ document object gives us access to the document
- ◉ 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



- <p id="name" class="hi">My text</p>
Maps to
- {

```
...  
childNodes: NodeList[1],  
className: "hi",  
innerHTML: "My text",  
id: "name",  
...  
}
```
- Attributes map **very loosely** to object properties so don't rely on it
- Lets check out some live examples...

Working with the DOM



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

DOM – Get your element(s)



- Start on document or a previously selected element (except getElementById)
- Returns a NodeList
 - `el.getElementsByTagName("a");`
 - `el.getElementsByClassName("className");`
 - `el.querySelectorAll("css selector");`
- Returns a single element
 - `document.getElementById("idname");`
 - `el.querySelector("css selector");`
- <http://jsfiddle.net/mrmorris/wcff257b/>

DOM - Traversing



- Move between nodes via their relationships
- Element node relationship properties
 - `.parentNode`
 - `.previousSibling`, `.nextSibling`
 - `.firstChild`, `.lastChild`
 - `.childNodes` // `NodeList`
- Mind the whitespace!
- <http://jsfiddle.net/mrmorris/dv13y28m/>

Looping through a NodeList



- ◉ **HTMLCollectionObject/NodeList**
 - ◉ 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

DOM – Node Content



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

DOM – Manipulation



- <http://jsfiddle.net/mrmorris/ktwdye0w/>
- Use el.innerHTML
- DOM manipulation
 - .createElement()
 - .createTextNode()
 - .appendChild()
 - .removeChild()
 - parentEl.insertBefore(newEl, beforeEl);
 - el.replaceChild(newNode, oldNode)
 - oldNode must be a child of “el”

DOM – Element Attributes



- <http://jsfiddle.net/mrmorris/duopdjdb/>
- Accessor methods
 - `el.getAttribute(name);`
 - `el.setAttribute(name, value);`
 - `el.hasAttribute(name);`
 - `el.removeAttribute(name);`
- As properties
 - `.href`
 - `.className`
 - `.id`
 - `.checked`

DOM – Styling/CSS



- Use element's “style” property
 - It's an object of style properties
 - .color = “black”;
 - .position
 - .top
 - .left
 - Some style names differ in JavaScript
 - Hyphens become camelCase
 - background-color => backgroundColor
 - Some names were keywords
 - float => cssFloat
 - <http://jsfiddle.net/mrmorris/hJwCj/>

DOM - Getting styles



- ◉ Can't accurately "get" styles on an element through its `style` property or attribute
- ◉ Must use `window.getComputedStyle()`
- ◉ `el.cssText` will return computed inline styles
- ◉ `classList` API
 - ◉ `el.classList.add("class")`;
 - ◉ `el.classList.remove("class")`;
 - ◉ `el.classList.toggle("class")`;
 - ◉ `el.classList.contains("class")`

Geometry of Elements



- Element size in px
 - `.offsetWidth`
 - `.offsetHeight`
- Element inner size, ignoring borders
 - `.clientWidth`
 - `.clientHeight`
- `.getBoundingClientRect()`
 - Returns object with top, bottom, left, right properties relative to top left of the page
- If you want it relative to document, add on current scroll position (globals)
 - `pageXOffset`, `pageYOffset`

DOM Performance



- ◉ Dealing w/ browser brings up a lot of performance issues
- ◉ Touching a node has a cost (especially in ie)
- ◉ Styling a bigger cost (it cascades)
 - ◉ Inserting nodes
 - ◉ Layout changes
 - ◉ Accessing css margins
 - ◉ Reflow
 - ◉ Repaint
- ◉ Accessing a nodeList has a cost

DOM basics - Recap



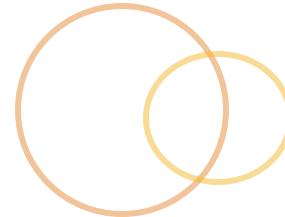
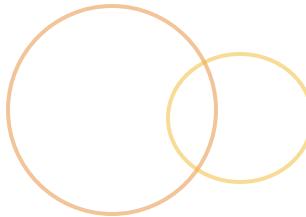
- ◉ The DOM is a model of the web page document.
 - ◉ It is a standardized convention.
- ◉ Browsers offer a JavaScript API to interact with the DOM
 - ◉ Can affect the page
- ◉ You can access, manipulate, create any content within the page
- ◉ jQuery will abstract much of the DOM API implementation nuances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

Exercises: Dom manipulation



- 1. 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. Make a function, “embolden”, that takes an element and makes it bold
 - ```
function embolden(element) {
 // makes the element bold
 // hint: style it OR wrap it in
}
```

# Events



- ◉ Browser's JavaScript engine has an event-driven, single-threaded, 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)
- DOM mutation (deprecated in favor of Mutation  
Observers in DOM4)

# Event handling (Basics)



- Select an element that relates to the event
- Indicate which event you want to listen to
- Define an event handling function to respond to the event when it occurs

# Event handling evolution



- Netscape
  - Used on<type>
  - Node[“onClick”] = function(){..}
- Microsoft
  - Node.attachEvent
  - Has global event object
- W3C
  - Node.addEventListener
    - All browsers by ie9+
    - body.addEventListener(“click”, function(){});

# Bind an event to an element



- Inline
- 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/>

# Context... in event handling



- ◉ JavaScript event handlers will be invoked in the context of the DOM node that triggered the event
  - ◉ So: `this === elementNode;`
- ◉ Careful with inner functions or callbacks, which may change context
  - ◉ Can store parent context
  - ◉ Or use “bind”

# Event handlers with arguments



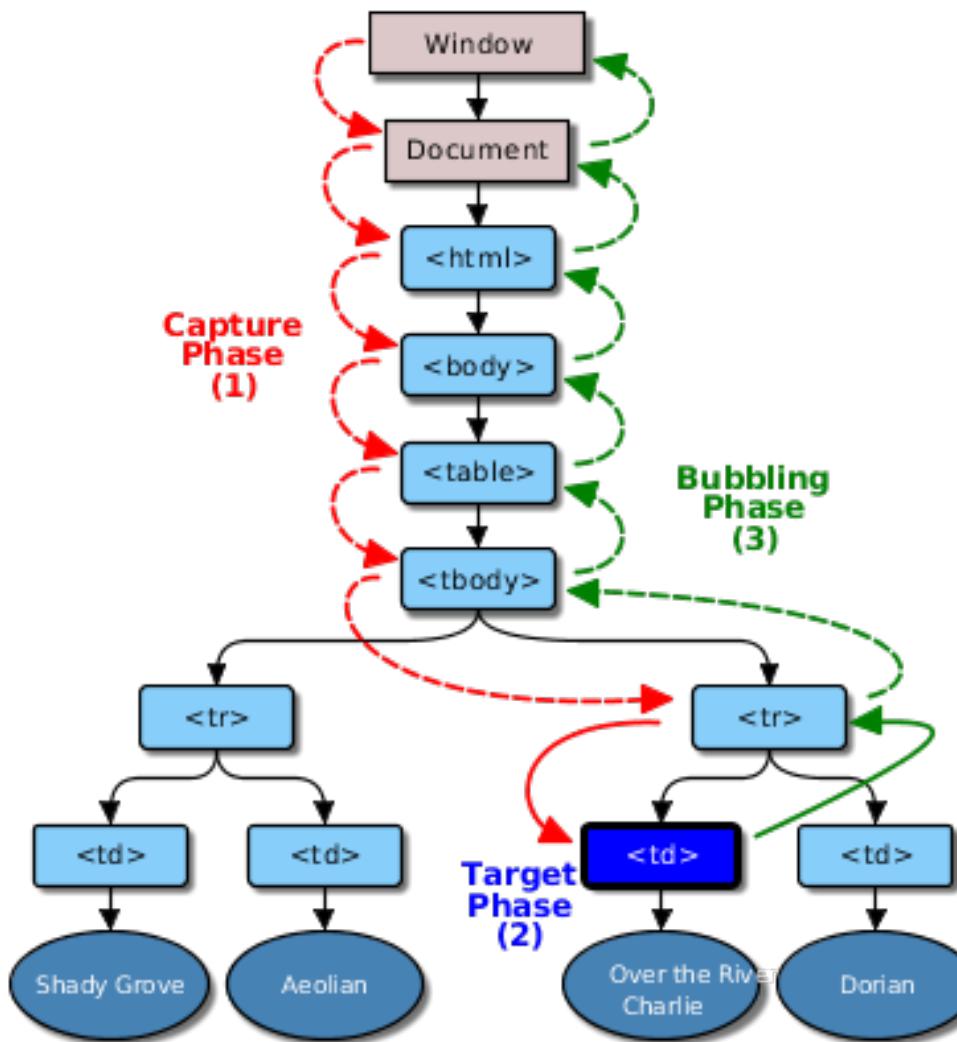
- ◉ This won't work like you expect
  - ◉ `element.addEventListener('blur', doSomething(5));`
- ◉ It is passing the result of a function invocation,  
not the function to-be-called as a callback
- ◉ Instead, wrap it in a function
  - ◉ `EI.addEventListener('blur', function() {  
 doSomething(5); // func needed an arg  
});`

# Event Propagation



- An event triggered on an element  $e/A$  is also triggered on all parent elements of  $e/A$
- Two models
  - Trickling, aka Capturing phase (Netscape)
  - Bubbling (MS)
- W3C decided to support both
  - Starts in capturing, then bubbling
  - Defaults to bubbling
- Bubbling supports Event Delegation
  - attach an event handler to a common parent of many nodes... and parent can determine source child and dispatch as needed.

# Event Propagation, continued



# Stopping events



- Cancel bubbling

- // ie8 and below

```
eventObject.cancelBubble = true;
```

```
// ie9+
```

```
if (eventObject.stopPropagation) {
 eventObject.stopPropagation();
}
```

- Prevent default browser behavior

- eventObject.preventDefault()

# 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)
- ◉ <http://jsfiddle.net/mrmorris/2gK7L/>

# That handler stuff...



- ```
.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 information



- Can determine the location of the mouse when the event occurred
 - Event.screenX
 - Event.screenY
 - Event.pageX
 - Event.pageY
 - Event.clientX
 - Event.clientY
- Key events include a “keyCode” property
- <http://jsfiddle.net/mrmorris/8htsexcg/>

Event Loop

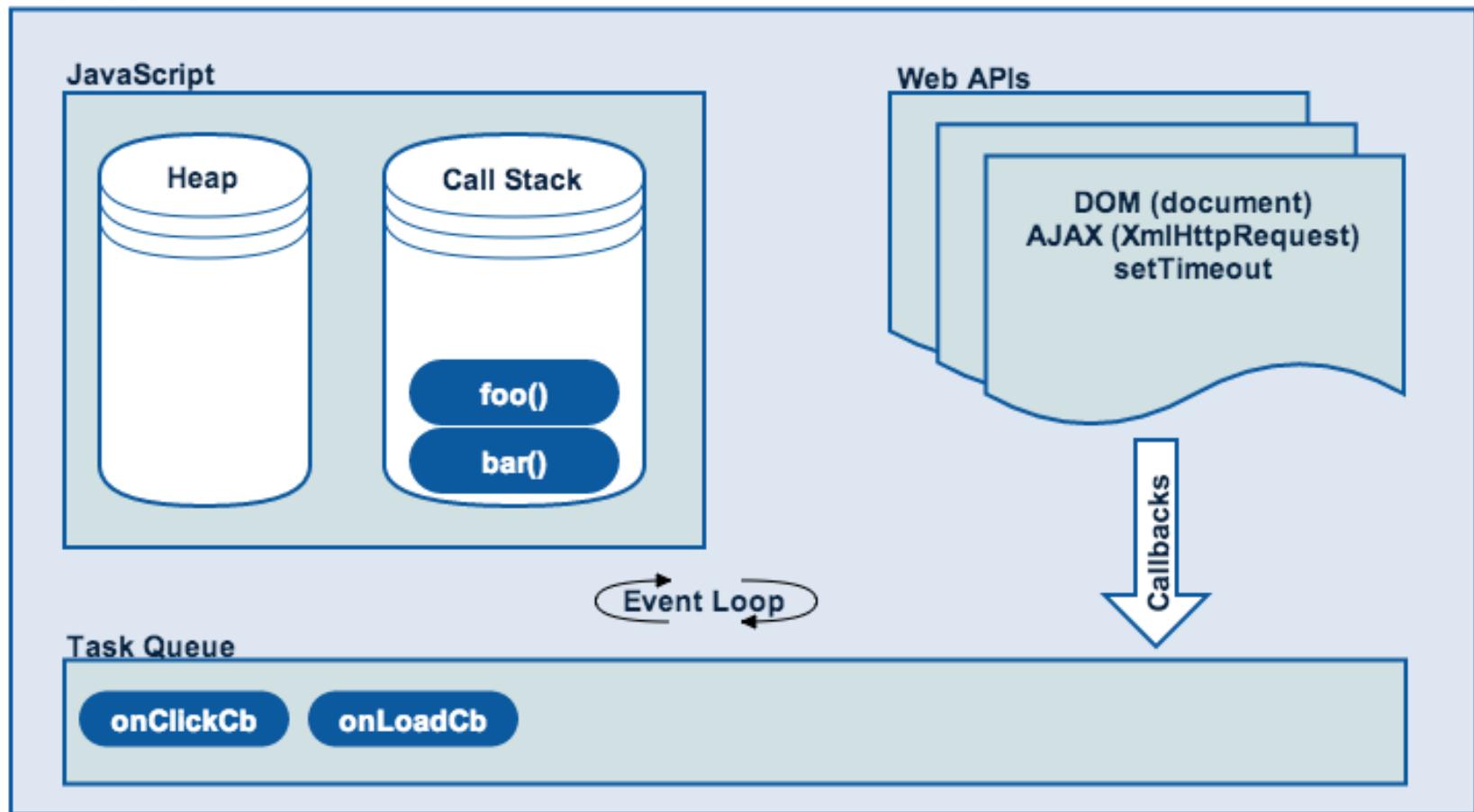


- Heart of the browser's JavaScript engine (one of the best parts)
- Allows asynchronous operations
 - Methods that get tacked into the queue are 'async'
- Each “turn” it returns a function from the Queue and runs it to completion (blocking)
 - Avoid blocking scripts...
 - alert, confirm, prompt, synchronous xhr/ajax
- For long running tasks...
 - *Eteration*: break task into multiple turns and call each with a setTimeout
 - Move the task into a separate process

The event loop



Browser



The event loop break down



- Single threaded stack
- Function calls are added to the stack
- On stack... “return” will pop the call off the stack and go to the next function down
- Each call is blocking until finished
- When stack is clear, anything (next one) in the Task Queue is popped off and put on the stack to run
- `setTimeout(0)` will push the task to the end of the main program by putting it in the task queue. Then the main program goes through the stack.

Event debouncing



- ◉ “debounce” events that fire rapidly like `mousemove`, `scroll`
 - ◉

```
textarea.addEventListener("keydown",  
  function(){  
    clearTimeout(timeout); // safe  
    timeout = setTimeout(function(){  
      // stopped typing...  
    }, 500);  
  });
```
- ◉ Respond to an event at intervals, displaying mouse coordinates periodically on `mousemove`
- ◉ <http://jsfiddle.net/mrmorris/a7GTG/>

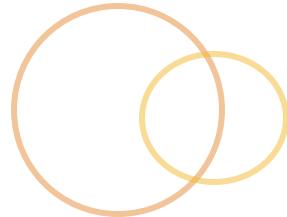
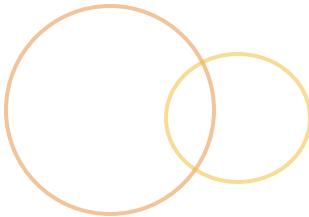
Events recap



- ◉ **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.



AJAX



- Interface through which browsers can make HTTP Requests
- Handled by the XMLHttpRequest object
- Introduced by Microsoft in the 90s for ie, taken from there...
- Why use it?
 - Non-blocking
 - Dynamic page content/interaction
 - Supports many formats
- Limitations
 - Same-origin policy
 - History management

Ajax – Sending a request



- ◉ Step by step
 - ◉ 1. Browser makes a request
 - ◉ 2. Server responds in xml/json/html
 - ◉ 3. Browser parses and processes response
- ◉ Create a request object and call its “open” and “send” methods
 - ◉

```
var req = new XMLHttpRequest();
req.open("GET", "url.json", false);
req.send(null);
```

Ajax – Handle the response



- ◉ “load” event will fire when response is received
- ◉

```
req.onload = function(e) {  
    if (req.status==200) {  
        console.log(req.responseText);  
    }  
}
```
- ◉

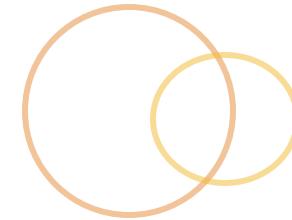
```
req.addEventListener("load", function(e) {  
    // also ok  
});
```
- ◉ An example... to the local servers!
 - ◉ <http://js-jquery.course/ajax/>

Ajax Data formats



Format	Summary	PROS	CONS
HTML	Easiest for content in page	<ul style="list-style-type: none">• Easy to parse• No need to process much	<ul style="list-style-type: none">• Server must produce the HTML• Data portability is limited• Limited to same domain
XML	Looks similar to HTML, more strict	<ul style="list-style-type: none">• Flexible and can handle complex structure• Processed using the DOM	<ul style="list-style-type: none">• Very verbose, lots of data• Lots of code needed to process result• Same domain only
JSON	Similar object literal syntax	<ul style="list-style-type: none">• concise! Small• Easy to use within JavaScript• Any domain, w/ JSONP or CORS	<ul style="list-style-type: none">• Syntax is strict• Can contain malicious content since it can be parsed as JavaScript

Ajax with HTML



- Easiest way to go
- Works with the DOM and styles
- Scripts will NOT run

Ajax with XML



- More work in processing the data to turn XML into HTML

```
var data = xhr.responseXML;
var events =
data.getElementsByTagName('event');

for (var i=0; i<events.length; i++) {
  var container = document.createElement('div');
  container.className = 'event';
  // create img node
  // append
}
```

- <http://js-jquery.course/ajax/index-xml.html>

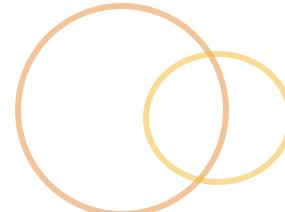
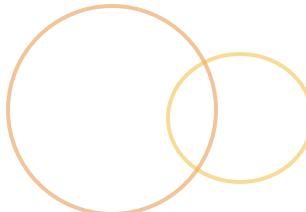
A word about JSON



- JavaScript Object Notation
- Used as a data storage and communication format
- Similar to JavaScript's object literal, with a few restrictions
 - Property names must be surrounded by double quotes
 - No function definitions, function calls or variables
- Methods
 - `JSON.stringify(object);`
 - `JSON.parse(string);`
- <http://js-jquery.course/ajax/data/data.json>



JSON



```
○ {
    "name": "Jason",
    "trophies": [
        "trophy1",
        "trophy2"
    ],
    "age": 40,
    "car": {
        "name": "toyota",
        "year": 1985
    }
}
```

Ajax with JSON



- It is sent and received as a string and will need to be de-serialized

```
var data = JSON.parse(xhr.responseText);
var newContent = "";

for (var i=0; i< data.length; i++) {
    newContent += '<div class="event">';
    newContent += '<img src=' + data[i].val+
"/>';
}
```

```
document.getElementById('content').innerHTML
= newContent;
```

- <http://js-jquery.course/ajax/index-json.html>

Ajax Continued...



- ◉ It is best to use an abstraction of XMLHttpRequest for
 - ◉ status and statusCode handling
 - ◉ Error handling
 - ◉ Callback registration (onreadystatechange vs onload)
 - ◉ Browser variations and fallbacks
 - ◉ Additional event handling
 - ◉ progress
 - ◉ load
 - ◉ error
 - ◉ abort
- ◉ Use a lib like jQuery....

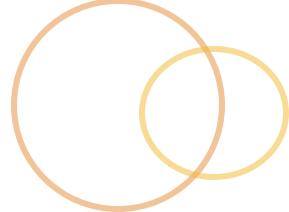
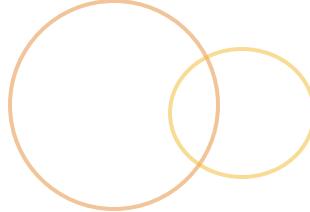
Cross-origin



- ◉ By default, ajax requests must be made on the same domain
- ◉ Alternatives to this are:
 - ◉ A proxy file on the server
 - ◉ JSON/p “Json with padding”
 - ◉ CORS (Cross-origin resource sharing), which involves new http headers between browser and server – ie10+



JSONp



- Browsers don't enforce same-origin policy on the `src` in script tags
- So...
 - We define a handling **callback** function
 - We dynamically add a **script** referencing an external `src`
 - We tell the script the **callback** to wrap the response in
 - Once script loads, the response is wrapped in our **callback**, which is invoked on load
- The callback function is expected to be defined on the origin
- <http://js-jquery.course/ajax/jsonp-example.html>

AJAX Recap



- ◉ A means for the browser to make additional requests without reloading the page
- ◉ Enables very **fast** and **dynamic** web pages
- ◉ Best with small, light transactions
- ◉ **JSON** is the data format of choice
- ◉ **Requests across domains** are possible but require jumping through some extra hoops (and your server must support it)

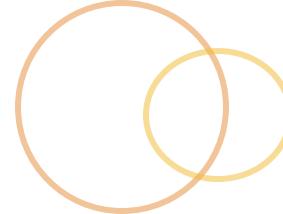
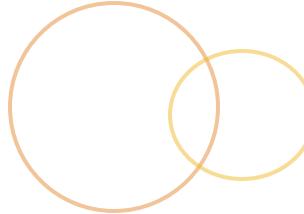
Exercise: JavaScript and the DOM



- ◉ 1) Create a tabbed UI that responds to click events and adding new tabs.
 - ◉ <http://jsfiddle.net/mrmorris/osq6fed3/>
- ◉ 2) Create a table-builder function that accepts JSON data and builds a table element with all the trimmings.
 - ◉ <http://jsfiddle.net/mrmorris/mnyn3y0t/>
 - ◉ Bonus: Use AJAX to load the data



jQuery



- ◉ “*write less, do more*”
- ◉ Abstracts away the ugly parts of the DOM and cross-browser support
- ◉ Handles...
 - ◉ Selecting elements with css selectors
 - ◉ Interacting with the DOM + content
 - ◉ Event handling
 - ◉ Ajax
- ◉ Doesn’t do anything pure JavaScript can’t do
- ◉ Chaining, feature detection

jQuery setup



- <http://jquery.com/>
- Include the script
 - 1.9+ -> supports ie 6, 7 and 8
 - 2.0+ -> drops support for older browsers
- \$ == jQuery
 - \$ is shorthand reference to jQuery object
- noConflict
 - Releases the \$ namespace
 - `jQuery.noConflict(); // use jQuery instead of $`
or
`$j = jQuery.noConflict();`
or
`(function($){...})(jQuery);`
 - If you have two copies of jQuery, passing “true” to noConflict will restore original
 - `jQExtracted = jQuery.noConflict(true);`

Getting started with jQuery()



- `jQuery()` or `$(())` is a function to fetch a DOM element as a jQuery object
 - `$(document);`
- It returns a *jQuery selection object*
 - `var myDocument = $(document);`
- You can then invoke jQuery methods on the element(s) you selected
 - `myDocument.hide();`
- And... you can chain most method invocations
 - `$("p.fancy").doSomething();`
- There are utility functions that don't depend on an element
 - We'll see those later

Running code on DOM ready



- `$(document).ready(fn);`
 - Executes `fn()` when the document's DOM is ready to execute JavaScript
 - `$(fn)` is equivalent
- `$(window).load(fn)`
 - Executes `fn()` when the document's DOM is ready, all resources are loaded (images, CSS, scripts) as well as the documents in any IFRAMES
- Most of the jsfiddle examples will do this for us...
- <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

CSS Selector refresher



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

Selecting, some examples



- `$("div")`
- `$("div#main ul")`
- `$("#main li.fancy")`
 - `$(".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)

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

Exercise: 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



- Practice selecting
 - <http://jsfiddle.net/mrmorris/5zto6n44/>
- Reverse selection mapping
 - <http://jsfiddle.net/mrmorris/7wgvnd8n/>
- Hint:
 - To style a table row/col with a background, you have to style the cell (td)

Add and filter selections



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

Exercise: Yup, we're still selecting!



- ◉ 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 – selecting native dom node



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

jQuery selection - recap

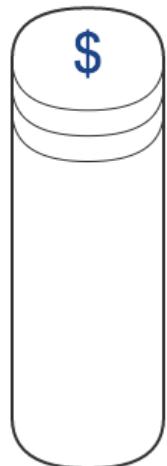


- `$(selector)` returns a jQuery selection object
 - array-like
 - `$("li"); // array of li's`
 - Element methods can be run directly on it
 - `$("#id").text("goodbye");`
 - These can be chained
 - `$("#id").text("goodbye").fadeOut()`
 - Implicitly iterates
 - `$("li").text("goodbye");`
 - Can store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

jQuery collection object

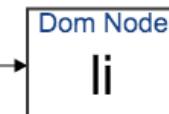
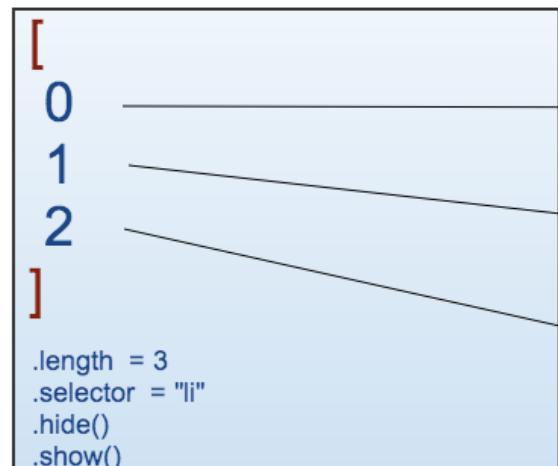


the jQuery Object



`$("li");`

>Returns



`$("li")[0];`

`$("li")[1];`

`$("li")[2];`

`$("li").hide();`

This will iterate over each result (as jquery objects)

jQuery looping



- ◉ jQuery selector will return array-like collection of matching elements
- ◉ Use `$(selector).each()` for iterating over elements
 - ◉ `$(li).each(function(i, el){
 // this === the dom node
 $(this).show();
});`
- ◉ Implicit iteration for jQuery collections
 - ◉ `$('li').html("updated");`
- ◉ Use `$.each()` for iterating over any collection
 - ◉ `$.each(collection, function(i, val){})`

jQuery chaining



- ◉ jQuery programming heavily uses “chaining”, which is a typical JavaScript function pattern
 - ◉ `$(“selector”).css({..}).fadeIn();`
- ◉ Most methods used to update the jQuery selection can be chained, since they return the jQuery element
- ◉ However... methods that retrieve information from the DOM or browser cannot be chained

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}{prepend}text{append}{after}**

jQuery element creation



- Step 1, Create the element

- `var newEl = $("<div>Hi</div>");`

- Step 2, 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

Selecting – Special cases



- ◉ iframes are special (aka “a pain”)
 - ◉ <iframe src="..."></iframe>
 - ◉ \$("iframe").contents().find(selector);
 - ◉ <iframe id="bla"></iframe>
 - ◉ \$("iframe#bla").contents().find(selector);
- ◉ Escaping characters
 - ◉ Sometimes a id name may use special characters, like “.”, so they will need to be escaped
 - ◉ Escape with “\\”
 - ◉ . Becomes \\.
- ◉ Put it together
 - ◉ <iframe id="bla.bla"></iframe>
 - ◉ \$("iframe#bla\\.bla").contents().find(selector);

Exercise: Selecting and manipulating



- ◉ Find the flags, take two
 - ◉ <http://jsfiddle.net/mrmorris/bkfxf6h2/>

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
 - `$('.li").css({
 "padding-left": "+=75",
 "color": "#ff9f9f9"
});`
 - `$('.li").css({
 paddingLeft: "+=75",
 color: "#ff9f9f9"
});`

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

Positioning Elements



- Use offset() and position() methods to determine and change position of elements in the page
- **.offset()**
 - Gets or sets coordinates of the element relative to the top left hand corner of the document
- **.position()**
 - Gets or sets coordinates of the element relative to any ancestor
- Both return an object with “top” and “left” properties, in px

jQuery - geometry



- Box dimensions

- `.height()`
- `.width()`
- `.innerHeight() // + padding`
- `.outerHeight() // + padding + border`
- `.outerHeight(true); // + padding + border + margin`
- And... width

- Remember your box model?

- [margin [border [padding [width] p] b] m]

Window and page dimensions



- Can use `height()` and `width()` methods to determine dimensions of the browser window and the document
 - `$(window).height();` // visible browser window
 - `$(document).height();` // full page height
- Can also get and set the position of scroll bars
 - `.scrollLeft()`
 - `.scrollTop();` // vertical position of the scroll bar for the first element in the selection. Can set it, too.

jQuery animations



- 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 animations continued...



- Custom animations
 - .delay()
 - `$(this).delay(700*index).fadeIn(700);`
 - Best for effects because it is limited – no way to cancel
 - .stop()
 - .animate()
- <http://jsfiddle.net/mrmorris/Q32FC/>

jQuery animations continued...



- `.animate()` for custom effects
 - Can animate most numeric css properties
 - Color can be animated with the “jquery-color” plugin
 - `.animate({
 opacity: 0.5
}, [speed, easing, completeCallback]);`

Effect queue



- Effects are stacked in a queue on an element
- Can stack animations in a queue
 - `$("#el").animate().animate().delay(1000).animate();`
- Can introduce functions to the queue
 - `$("#el").animate().animate().queue(function(next){
 // stuff then go to next item in queue
 $(this).dequeue();
 // or.. As of jquery 1.4
 next();
}).animate();`
- To clear the queue
 - `.stop(true, true); // clearQueue, jumpToEnd`

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 `$("<div></div>");`
- **Styling** is supported, as are some **light animations**

Exercise: Content manipulation



- Content manipulation, movin' fruit
 - <http://jsfiddle.net/mrmorris/geq2fyxv/>

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
 - ◉ 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) {  
    // handled only when the ul descendants  
    // match the given selector  
    // e.target is the childNode triggering  
});
```
- ◉ <http://jsfiddle.net/mrmorris/mn4G9/>

jQuery Event Object



- Passed to event handler functions
- Properties
 - type – type of event
 - which – button or key that was pressed
 - data – extra data passed in from handler
 - target – dom element that initiated the event
 - pageX – mouse position from viewport
 - pageY
 - Timestamp
- Methods
 - .preventDefault()
 - .stopPropagation()

Exercise: Events



- Event handling with jQuery
 - <http://jsfiddle.net/mrmorris/jjomkn54/>

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/>

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
- jQuery validation plugin

Exercise: Basic Events



- Still challenged by events? Try the click chaser:
 - <http://jsfiddle.net/mrmorris/ofh1cz0d/>
- Form events
 - <http://jsfiddle.net/mrmorris/tpyhgt42/>

jQuery Ajax



- Several ways to do this, but they are all shortcuts of `$.ajax()`, which gives you all the controls

```
○ $.ajax({  
    type: method (GET or POST)  
    url  
    data  
    success: callback  
    error: callback  
    complete: callback  
    timeout  
    dataType: (xml, json or html)  
    beforeSend: callback  
});
```

jQuery ajax shorthand



- `element.load()`
 - Easy!
 - Loads data directly into an element
 - Can target fragment elements in the response
 - `$('#content').load('bla.html#content');`
- `$.get(url[, data, callback, dataType]);`
- `$.post(url[, data, callback, dataType]);`
 - Serialize form data with `$("#my-form").serialize();`
- `$.getJSON(url[, data, callback]);`
- `$.getScript(url[, callback]);`

jQuery Ajax examples



- <http://js-jquery.course/ajax/jquery-index.html>
- More \$.ajax with form post
 - <http://jsfiddle.net/mrmorris/pj4e7jxv/>

jQuery – handling the response



- `$.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()`

# jQuery 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 doesn't really do it right



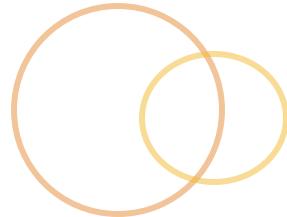
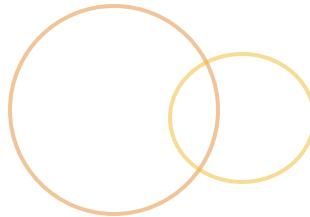
- The spec: <https://promisesaplus.com/>
  - Uses .then()
- jQuery doesn't quite follow it
- Could use some alternative libs like
  - Q <https://github.com/kriskowal/q>
  - RSVP <https://github.com/tildeio/rsvp.js>
  - When <https://github.com/cujojs/when>
- Var betterPromise = Q(jQueryPromise);

# Older jQuery ajax callbacks



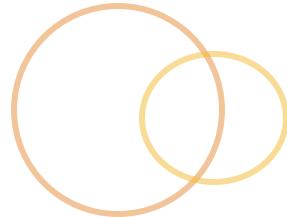
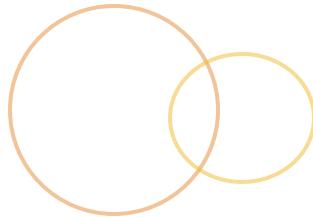
- Pre jQuery 1.8, ajax callbacks were handled by these (now deprecated) methods
  - `.success()`
  - `.error()`
  - `.complete()`

# Deferred



- Promise interface is based off of the `$.Deferred` object in jQuery
- Two key methods
  - `resolve`
  - `reject`
- Three events (for callbacks)
  - `done`
  - `fail`
  - `always`
- Allows you to establish a state that can be fulfilled (or failed) and for code to be executed based on that state once it changes

# Deferred



- ➊ var deferred = \$.Deferred();

```
deferred.done(function(result){
 console.log("I am: " + result);
});
```

```
deferred.resolve("done");
```

# Taking Deferred/Promises further



- `deferred.promise();`
  - Return a (restricted) promise interface
- `deferred.pipe(doneCB, failCB)`
  - return modified data or fail with `reject()`
- `deferred.when(prom1, prom2).done().fail().always();`
  - returns a new deferred from a set of promises
- `deferred.then(doneCB, failCB)`
  - Returns new promise after previous are satisfied

# 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()`

# Exercise – Ajax



- Fetching data through an API
- We'll be using this API:
  - <http://jsonplaceholder.typicode.com/>
  - <https://github.com/typicode/jsonplaceholder>
- Let's try one together
  - <http://jsfiddle.net/mrmorris/Ln8ecynw/>
  - \*Check out the network panel
- Create a todo list
  - <http://jsfiddle.net/mrmorris/1gtqsohv/>

# jQuery Utilities



- ◉ <http://jsfiddle.net/mrmorris/bmtqr5s6/>
- ◉ `$.each(array, callback(index, val))`
  - ◉ Iterates over array items and runs a callback on each
  - ◉ Also supports an object... in which case callback gets “property name” and “value of property”
  - ◉ “this” will be the value, wrapped in an object
  - ◉ Return “false” to break the loop
- ◉ `$(selector).each()` is used exclusively to iterate over a jQuery object
  - ◉ Made for DOM iteration
  - ◉ “this” is a DOM element
  - ◉ Callback gets the index
  - ◉ Should use implicit iteration

# jQuery Utilities



- ◉ 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)

# Extending jQuery



- ◉ Sometimes you want to make a reusable piece of functionality available throughout your code... or in other projects
- ◉ Plugins offer a way to extend jQuery object by extending the jQuery prototype
- ◉ Lots of plugins available already
  - ◉ <http://plugins.jquery.com/tag/jquery/>
  - ◉ <https://www.npmjs.org/browse/keyword/jquery-plugin>
- ◉ Just include the plugin src file(s) and follow its usage directions

# jQuery Plugin Authoring



- All methods on the jQuery object are plugins
- Plugins should be designed to work on a single element or a collection of elements
- Plugins modify the jQuery prototype (\$.fn), so all instances, now and future, will gain the plugin functionality
- Individual plugins should only take up one “slot” in the \$.fn namespace

# jQuery Plugin Authoring cont...



- Extend `$.fn` to make method available to all jQuery objects
  - `$.fn.greenify = function() {  
 this.css("color", "green");  
};  
$("a").greenify();`
- Make it chainable...
  - Return **this**
- Make it alias-proof
  - `(function($) {...}(jQuery)); // IIFE`
- <http://jsfiddle.net/mrmorris/X9b6G/>

# Advanced Plugins



- Should expose default plugin settings to allow for customizations by the user
  - Set and extend “options” in function
  - Set defaults externally
  - ```
var opts = $.extend({}, $.fn.hilight.defaults, options);
```



```
...
```

```
$.fn.hilight.defaults = opts;
```

wrap this in a closure
- An example
 - <http://jsfiddle.net/mrmorris/Z7HZL/>
- Can also allow extensibility within a plugin , with callbacks or methods that can be redefined

Plugins, best practices



- ◉ Keep your plugin self-contained to avoid cluttering the jQuery namespace
- ◉ To work with collections, use “each()”
 - ◉

```
return this.each(function(){  
    $(this); // it's a callback so "this" is the DOM node  
}); // returns the collection
```
- ◉ Accepting options – easiest with an object literal and extend for defaults
- ◉ Some plugin files will lead with “;” to prevent minification issues with other libs
- ◉ Boilerplate
 - ◉ <https://github.com/jquery-boilerplate/jquery-boilerplate/blob/master/src/jquery.boilerplate.js>

Exercise: Plugins



- ◉ Fix this broken plugin
 - ◉ <http://jsfiddle.net/mrmorris/2d0qr06f/>
 - ◉ What is broken and why?
 - ◉ What happens if jQuery is in noConflict mode?
 - ◉ What if we want to configure the plugin with options
(Like the ability to customize the character that is appended)

jQuery/DOM Best Practices



- Avoid changing the DOM as much as possible
 - But if you have to, use a document fragment or string
- Be specific on the right side of selectors
- Don't be overspecific
- Don't use the * selector, if possible
- Check for selection length before manipulating
- If performing a number of operations on an element, detach it from the DOM
- If changing the CSS for many selectors at once, create a new style tag and append it to HEAD
- Use native where it works
- Take note when you need to return an element, or not
- Avoid double wrapping \$(jQueryObjects)

Exercise: jQuery wrap up



- ◉ (Maybe we'll save this for last)
 - ◉ What is of interest?
 - ◉ [Adv] Make a slideshow plugin
 - ◉ [Int] Make a grid-builder plugin, given some data.json
 - ◉ [Int] A tabbed UI
 - ◉ [Beg] A tooltip UI element plugin
 - ◉ [Beg] A modal plugin

jQuery UI



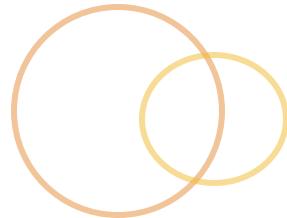
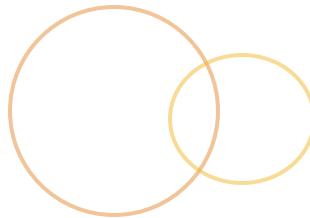
- <http://jqueryui.com/>
- “jQuery UI is a curated set of user interface interactions, effects, widgets, and themes built on top of the jQuery JavaScript Library”
 - Accordions
 - Date picker
 - Tabs...
- Extends jQuery, using jQuery conventions and introducing a few of its own

Getting set up with jQuery UI



- ◉ Download it
 - ◉ <http://jqueryui.com/download/>
 - ◉ Build your package
 - ◉ Core: Required
 - ◉ Interactions: Drag, Drop, Resize, Selectable, Sort
 - ◉ Widgets: UI Elements like Tabs, Dialog, Datepicker
 - ◉ Effects: Additional animation effects
 - ◉ Pick a theme, scope as needed
- ◉ Use a CDN
 - ◉ <https://code.jquery.com/ui/>
- ◉ Include the files in your document
 - ◉ jQuery UI js
 - ◉ The theme css
 - ◉ And... an images folder in same place as theme css file
- ◉ <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

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 walk through



- 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 Options



- So many!
- Let's just cruise the documentation
 - <http://jqueryui.com/demos/>

Widgets vs HTML5



- ◉ Native HTML5 components will conflict with Framework widgets (primarily form widgets)
- ◉ <http://jsfiddle.net/mrmorris/fb5r8x04/>
- ◉ Native...
 - ◉ Easy
 - ◉ Optimized per platform (Mobile)
 - ◉ No dependencies
 - ◉ But not easily styled or customized
 - ◉ Not all browsers support them yet
- ◉ So you have to pick one or the other, can't easily get both
- ◉ Modernizr shim is a stopgap
 - ◉ If (!Modernizr.inputtypes.date) { \$("#el").datepicker();}

Don't forget your 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”
 - ◉ `$("#my-element").on("widgetsomething", function() {
 console.log("Hi");
});`
 - ◉ or hook into the event callback
 - ◉ `$("#my-element").widgetName({
 something: function() {}
});`
 - ◉ Check the documentation
 - ◉ Dialog Events: <http://api.jqueryui.com/dialog/>

Interactions



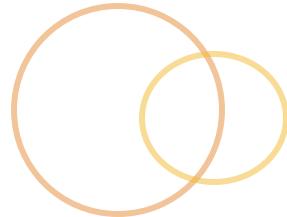
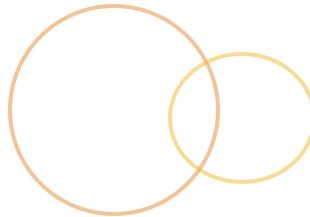
- ◉ “Set of mouse-based interactions to add rich interfaces and complex widgets”
- ◉ Similar to widgets, they have options, methods and events.
- ◉ The interactions
 - ◉ Draggable
 - ◉ Droppable
 - ◉ Resizable
 - ◉ Selectable
 - ◉ Sortable

Using Interactions



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

Effects



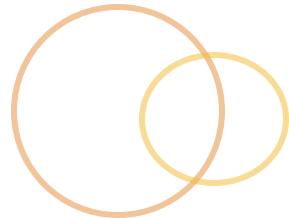
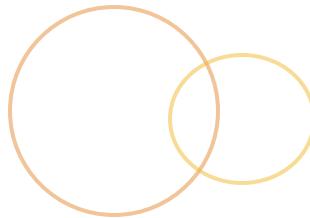
- Extends jQuery
 - New Effects
 - Better class animation support
- Not all styles can be animated
 - background-image
- Color animation is now built-in

Animating



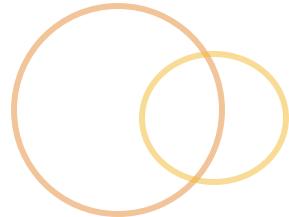
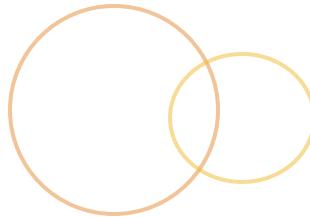
- Class animations
 - .addClass(class, duration, complete)
 - .removeClass(class, duration, complete)
 - .toggleClass(class, duration, complete)
 - .switchClass(removeClass, addClass, duration, cb)
 - Specificity matters!
- Display animations
 - .hide(effect, options, complete)
 - .show(effect, options, 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
 - etc.. <http://api.jqueryui.com/category/effects/>
- 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);`

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



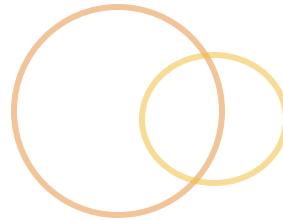
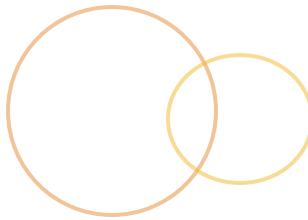
- **.hide**
  - (effect [, options] [, duration] [, complete])
  - (options)
    - effect
    - easing
    - duration
    - complete
    - queue
- **.addClass**
  - (className [, duration] [, easing] [, complete])
  - (className [, options])
    - duration
    - easing
    - complete
    - children
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

# Additional selectors



- ◉ **:data(key)**
  - ◉ Elements that have data stored in that key
- ◉ **:focusable**
  - ◉ Elements that support focus (input, select, textarea, button, object)
- ◉ **:tabbable**
  - ◉ Elements that support getting focus by tabindex

# 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

# Exercise – Widgets playground



- Set up a tabbed UI with 3 tabs
  - The **first tab** should display a paragraph with several **tooltips** utilized in the text
    - Hint: use **a** or **spans** to wrap text that should be tool-tipped
  - The **second tab** should display a single, centered (vertical and horizontal) **button** widget
    - Use **.position()** to center it
    - Clicking the button should trigger a **dialog** window that says hello
  - The **third tab** should display a **form** with a **date picker**
- Basic jQuery UI template
  - <http://jsfiddle.net/mrmorris/q7o0o4d3/>
- Need the libs locally? Use a CDN...
  - <https://code.jquery.com/jquery-1.11.1.min.js>
  - <https://code.jquery.com/ui/1.11.1/jquery-ui.min.js>
  - <https://code.jquery.com/ui/1.11.2/themes/smoothness/jquery-ui.css>
- Use the documentation
  - <http://jqueryui.com/demos/>

# Exercise - Animation



- Create a div that is centered on the screen
  - It needs a width and height specified
  - Use .position()
- Experiment with animating it
- Add 5 other divs with data-color attributes;  
clicking these will change the color of the original div
  - Or try a “color” html5 form control
  - Need some color values?  
[http://www.w3schools.com/cssref/css\\_colornames.asp](http://www.w3schools.com/cssref/css_colornames.asp)

# Theming



- ThemeRoller – a robust custom theme builder
  - <http://jqueryui.com/themeroller/>

# The Widget Factory



- Use `$.widget` to create new widgets or extend them
- Widgets vs jQuery plugins
  - Maintain state
  - Typically have life cycles
  - Can have their own events and react to changes
- Widget Factory defines how to...
  - Create & destroy widgets
  - Get & set options
  - Invoke Methods
  - Listen to events from the widget

# Let's make a widget



- ➊ Simple widget setup

- ➌ 

```
$.widget("ns.highlight", {
 options: {},
 _create: function() {
 // this.element
 // this.options
 }
});
$("#myel").highlight();
```

- ➌ Creates a jquery plugin on `$.fn` with widget boilerplate behavior

- ➌ <http://jsfiddle.net/mrmorris/1LbrqL88/>

# Built-in Widget methods



- `option(key, value)`
  - Getter and setter
- `_setOption: function(key, value)`
  - Can use this to react to option changes
  - `this._super(key, value)`
- `_setOptions: function(options)`
  - Can override or use to react to changes
  - `this._super(options);`
- `_destroy()`
  - Remove an instance, unbinds events, etc

# Chaining widget methods



- Get the widget instance

- // pre jQuery UI 1.11

```
var widg = $("#myel").data("ns-widgetname");
widg.method();
```

- // in jQuery UI 1.11+

```
var widg = $("#myel").widgetname("instance");
```

# Widget Extension



- Extend existing widgets
- Pass in original widget's constructor and the extensions as the last arg
  - ```
$.widget("custom.betterSpinner", $.ui.spinner, {  
    newMethod: function() {},  
    extendMethod: function() {  
        // this will call parent methods  
        return this._super(arg1, arg2);  
        // return this.superApply(arguments);  
    }  
});
```

HTML 5 APIs



- HTML5 APIs

- Constraint API (Forms)
- Geolocation
- File API
- Storage API
- Web Sockets
- Web Workers
- Media elements
- History API
- Canvas
- ClassList

HTML 5 APIs (in brief)



- New set of JavaScript apis that standardize how you make use of the new features
- There are a lot... mixed support across all browsers
- A few like...
 - Geolocation
 - localStorage
 - sessionStorage
 - History
- And many more.. <http://platform.html5.org/>

Feature detection



- ◉ Check for support of an api before using it
 - ◉ if (navigator.geolocation) {
... supported...
}
- ◉ Or... use a lib like Modernizr.js
 - ◉ if (Modernizr.geolocation) {
... supported...
}
- ◉ Shims can often get it working in most modernish browsers

HTML5 - ClassList



- Instead of
 - `$(‘body’).addClass(‘hasJs');`
 - Or `document.body.className += “ hasJs”;`
- You can utilize the `classList` object
 - `document.body.classList.add(‘hasJs');`
 - `document.body.classList.remove(‘hasJs');`
 - `document.body.classList.toggle(‘hasJs');`
- <http://jsfiddle.net/mrmorris/k67Lx/>

HTML5 Constraint API



- Adds validation to forms
- Not fully supported
 - [http://caniuse.com/#search=form validation](http://caniuse.com/#search=form%20validation)
- No replacement for server side validation!

HTML5 Constraint API



- Basic validation performed via “type” attribute
 - <input type="email" />
 - <input type="url" />
 - <input type="number" />
- By pattern
 - <input type="text" pattern="[0-9]{2}" />
 - 2 numbers...
- Require
 - <input type="text" required />
- Length
 - <input type="text" maxlength="9" />
 - <input type="number" min="1" max="10" />
- No validation
 - <form novalidate>

HTML Constraint API



- ◉ More new form field types
 - ◉ search
 - ◉ email
 - ◉ url
 - ◉ tel
 - ◉ number
 - ◉ range
 - ◉ date
 - ◉ month
 - ◉ week
 - ◉ time
 - ◉ datetime
 - ◉ datetime-local
 - ◉ color
- ◉ `el.setCustomValidity(message);`
 - ◉ Check if valid, set message if not
 - ◉ Set it to "" to indicate validity
- ◉ <http://jsfiddle.net/mrmorris/ybpvu578/>

History API



- Can interact and update a user's browser history
- Each tab/window keeps its own history, which is a list of pages visited
- Useful when loading ajax content, which doesn't affect history

History API



- Go forward (like clicking forward/back in browser)
 - `window.history.forward();`
- Go backward in history
 - `window.history.back();`
- Move to a specific point in history
 - `window.history.go(-1);`
- Length is the # of pages in history
 - `window.history.length;`
- Current state
 - `window.history.state`

History API



- HTML5 Introduces several new functions and an event (`window.onpopstate`) to manage history
- `window.history.pushState();`
 - `history.pushState({}, "page 2", "new-page.html");`
- `window.history.popState();`
- `window.history.replaceState();`
 - Modify current history object, versus adding
- The state object
 - Can be anything you want, it is useful to specifying parameters for handing history change events
- `window.onpopstate = function(event) {`
 `event.state;`
`}`
- <http://jsfiddle.net/mrmorris/R34rC/>

Storing Data



- Can use Cookies
 - 5kb max, each
 - Max 20 cookies per domain
 - Not very secure
- HTML5 Offline storage apis
 - **Web Storage**
 - ~~Web SQL DB~~
 - Indexed DB (aka Web Simple DB)

Web Storage



- Simple api for storing key/value pairs of data
- Two types, **local** and **session** – same api, different lifetime and scope
 - Local
 - Scoped to origin; permanent
 - Session
 - Scoped to origin and window/tab
- About 5 mb storage per origin
- Browser treats it like a cookie, or a cache, depending on the browser

Storage API



- Can use either the `localStorage` or `sessionStorage` objects
- Save an item with `setItem()`
- Retrieve a value with `getItem(key)`
- Object notation is supported
 - `localStorage.keyName;`
 - `localStorage.keyName=value;`
- Storing is synchronous, therefore it can slow things down
- Remove items with `removeItem(key)`
- Clear the storage with `clear()`
- Can listen to “storage” event

Storage API



- Basic

- <http://jsfiddle.net/mrmorris/j8x3nv9g/>

- Read more

- <http://diveintohtml5.info/storage.html>
 - <http://www.html5rocks.com/en/features/storage>

Geolocation



- ◉ Lets use share their location
- ◉ Form of long/latitude
- ◉ User can opt in/out of the request to share, or turn it off entirely
- ◉ “geolocation” object
 - ◉

```
If (navigator.geolocation) {  
    navigator.geolocation.getCurrentPosition(  
        successFunc(pos) {  
            pos.coords.latitude;  
            pos.coords.longitude  
        },  
        failFunc(posError){...}  
    );  
};
```

Geolocation API



- Geolocation Object
 - Request through here
 - Child of navigator
 - `getCurrentPosition(successCallback, failCallback);`
- Position Object
 - `Position.coords.latitude`
 - `Position.coords.longitude`
 - Accuracy (in meters)
 - Altitude
 - Heading, speed, timestamp
- PositionError Object
 - Code and a message

Geolocation Examples



- Basic
 - <http://jsfiddle.net/mrmorris/nqNxU/>
- Little more to it
 - <http://jsfiddle.net/mrmorris/s4xtduo2/>

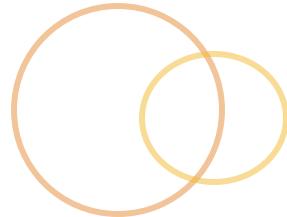
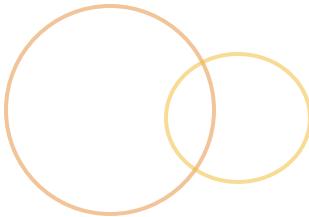
Drawing in the browser



- Browser has several ways to display graphics
- You can only get so far w/ DOM elements, css and transformations
- 1) Scalable Vector Graphics (SVG)
 - Drawing with shapes
 - Retain vector-ness therefore are scalable
 - Uses DOM elements to define shapes
- 2) Canvas
 - Single DOM element that encapsulates a picture
 - Has its own programming interface
 - Converts shapes to pixels (non-vector)
 - Requires a full redraw for changes
 - Purely scripted

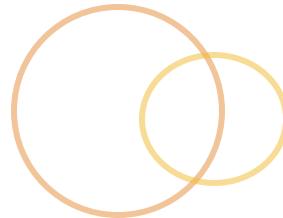
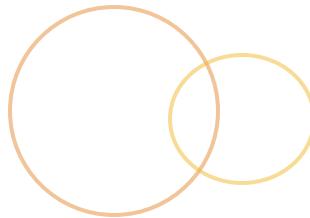


SVG



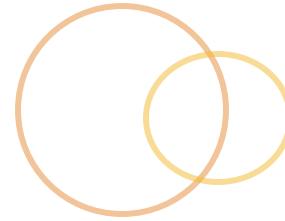
- ◉ Warning: I am not an artist
 - ◉ <http://jsfiddle.net/mrmorris/8G95p/>
 - ◉ <http://jsfiddle.net/mrmorris/efsa0mn5/>
- ◉ Has its own namespace and new elements
 - ◉ <svg xmlns="http://www.w3.org/2000/svg" width="150" height="150">
 <rect x="10" y="10" width="100" height="100"/>
 <circle cx="50" cy="50" r="40" stroke="black" stroke-width="3" fill="red" />
 Sorry, your browser does not support inline SVG.
 </svg>
- ◉ Read more
 - ◉ <https://developer.mozilla.org/en-US/docs/Web/SVG>

Canvas



- Basic stuff
 - <http://jsfiddle.net/mrmorris/21peeay2/>
- A new canvas is just an element and it starts empty
- To begin drawing, we first create a context, which is an object whose methods provide the drawing interface.
- 2 widely used “styles” of drawing
 - “2d” for flat
 - “webgl” for 3d through OpenGL interface
- <http://jsfiddle.net/mrmorris/cL6S5/>

Canvas Continued



- A shape can be either “filled” or “stroked”
 - `fillRect(x,y,w,h)`
 - `strokeRect()`
 - `clearRect(); // transparent rect`
- Setting a “`fillStyle`” changes the way shapes are filled
 - Can be a string for a color
- “`strokeStyle`” is similar
 - Color of the line
 - “`lineWidth`”

Canvas Drawing: Paths



- A path is a sequence of lines, which is done through side effects of lines
 - beginPath()
 - moveTo()
 - lineTo()
 - stroke()
 - Can also closePath() explicitly and fill();
- The path contains multiple shapes, each started by the “moveTo” – this creates a series of line segments
- Unclosed objects will automatically be closed when filled

Canvas Drawing: Curves



- A path may also contain curved lines
- quadraticCurveTo() draws a curve to a point using a control point and destination point
- bezierCurve() is similar, but uses two control points
- arcTo() can generate circle fragments
 - A circle is 4 arcTo calls.. Or just arc()
 - arc(centerX,centerY,radius,startAngle,endAngle)
- To prevent lines between objects, use moveTo() to start a new object

Canvas Drawing: Text



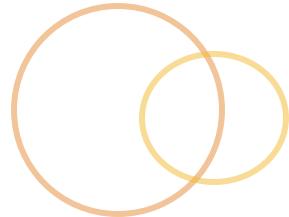
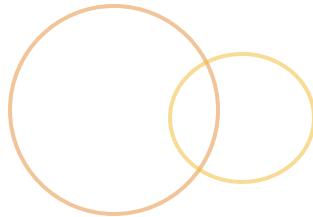
- ◉ `.fillText`
- ◉ `.strokeText`
- ◉ Coordinates indicate start of text baseline

Canvas Drawing: Images



- ◉ .drawImage() allows us to draw pixel data
 - ◉ drawImage(img, x, y);
- ◉ Either or from another canvas
- ◉ When hiding images to load you may need to register a load event
 - ◉ Var img = document.createElement("img");
img.src="img/hat.png";
img.addEventListener("load", function() {
 cx.drawImage(img, x, y);
});

Canvas



- Amazing stuff..

- <http://www.effectgames.com/demos/canvascycle/>
- <http://andrew-hoyer.com/experiments/cloth/>

- Even more awesome

- <http://phoboslab.org/crap/wolf3d/>
- <https://code.google.com/p/quake2-gwt-port/>

HTML5 – Video & Audio



- <http://jsfiddle.net/mrmorris/717rs3ct/>
- Native video playback element
 - <video src="url" controls></video>
- Can specify multiple fallback sources
 - <source src="bla" type="video/mp4,code=..." />
- Can jump to a specific time or set a media fragment
 - src="bla.mp4#t=10,20"
- Can specify a titles track using WebVTT format
 - <track src="bla.vtt" label="subtitles" kind="subtitles" srclang="en" default></track>
- Can css style it
 - filter:grayscale(100%); // makes it black and white

HTML5 Video



○ Other attributes

- Autoplay
- Preload
- Poster (preview image)
- Controls
- Height
- Width
- Loop
- Muted

HTML5 Video



- API (same with <audio>)
 - currentTime, volume, muted, currentSrc
 - Load(), play(), pause(), canPlayType()
- Events
 - canplaythrough, playing, ended
 - error, progress, waiting
 - Loadedmetadata
- Can interact with canvas elements
 - To take screenshots
- MediaElement.js
 - Excellent shim and extra features to media elements

HTML5 – Web Workers



- ◉ JavaScript is single-threaded, events can be blocking
- ◉ Use setTimeout, setInterval and AJAX to mimic concurrency
- ◉ Web Workers bring “threading”-like behavior to JS
- ◉ API gives spec to
 - ◉ Spawn background tasks (scripts)
 - ◉ Workers utilize thread-like message passing
 - ◉ Two types of workers, dedicated and shared

Web Workers continued...



- ◉ <http://jsfiddle.net/mrmorris/W3mBN/>
- ◉ <http://jsfiddle.net/mrmorris/hkgsxa3e/>
- ◉ Create a worker from a script, which is downloaded asynchronously
- ◉ postMessage() starts the worker
- ◉ Add a “message” listener to handle data
- ◉ Can close worker with .terminate()
- ◉ The Worker script...
 - ◉ Refers to “self” as itself
 - ◉ Listens to “message” event
 - ◉ self.postMessage() to send message back
 - ◉ Closes itself with self.close()
 - ◉ Can load further scripts with importScripts()
- ◉ More about Blob
 - ◉ <https://developer.mozilla.org/en-US/docs/Web/API/Blob>

HTML5 – Web Sockets



- Enable server to send data to the client... two way messaging enables much more dynamic applications
- Used to use long-polling or “COMET”
 - Keep a long connection open and re-open it as it gets closed
 - Added lots of overhead and occupied ports
- The spec defines an api for establishing “socket” connections between a browser and server... persistent connections so both parties can send and receive

Web Sockets



- Examples
 - Basic <http://jsfiddle.net/mrmorris/5zwbo1ft/>
 - Similar <http://jsfiddle.net/mrmorris/9bB79/>
- Create a connection with `WebSocket()` constructor
- Set event handlers for “open” and “error” and “message”
- Can send strings, Blob or ArrayBuffer for binary

HTML5 – File Reader API



- Lets applications asynchronously read the content of files from a user's computer
- File objects can be gotten through
 - `FileList` object, returned from an `<input>`
 - `DataTransfer` object (from Drag & Drop)
 - `HTMLCanvasElement`'s `mozGetAsFile()` api
- With this you can
 - Access (or display) a file info
 - Drag & drop files into the browser
 - Display live preview thumbnails
 - Handle ajax uploads w/ progress display
- <http://jsfiddle.net/mrmorris/uLzy57dg/>

...HTML5 APIs



- Drag & Drop API
 - <http://jsfiddle.net/mrmorris/94csfkpu/>
 - Read on:
https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Drag_and_drop
- Web Notifications API
 - <http://jsfiddle.net/mrmorris/tjmrz82k/>
 - Read on:
https://developer.mozilla.org/en-US/docs/Web/API/Notification/Using_Web_Notifications
- Web SQL API
 - <http://jsfiddle.net/mrmorris/5ks9u45g/>
 - Read on:
<http://html5doctor.com/introducing-web-sql-databases/>

HTML5 – Learn more



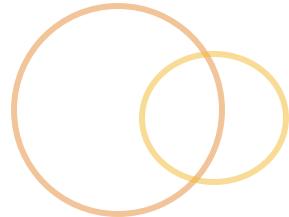
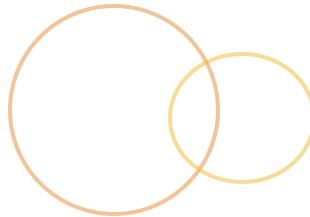
- Keep up to date
 - <http://www.html5rocks.com/en/>
 - <http://platform.html5.org/>

Google Maps



- ◉ Ability to embed maps in your app/pages and customize the look and information displayed
- ◉ Three options
 - ◉ Static maps
 - ◉ Embeddable maps
 - ◉ Interactive/Apps

Setup

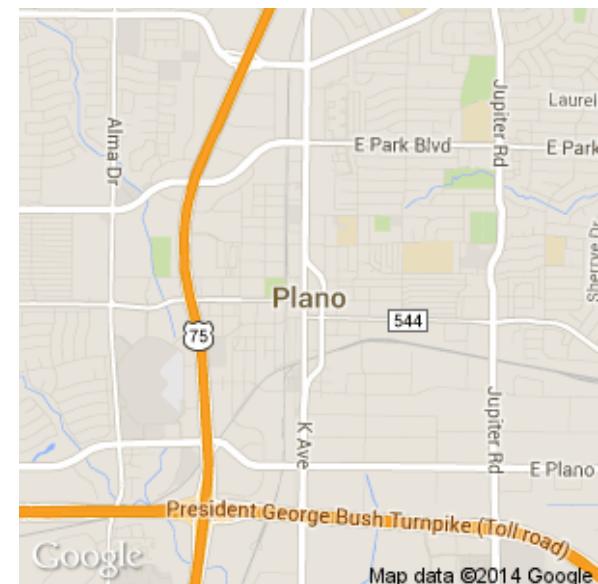


- Establish API access
 - <https://code.google.com/apis/console>
 - You can use mine... for now! ☺
 - AlzaSyARI5tt5D1UMaqg2vZ7-RiweQVh8aimcE
- Turn on the features in your api console
- And you're good to go
 - Mind your limits: 25k per day, or 1000 per hour per ip.
Roughly.

Option 1: Static Maps



- Embed a map as an image
- No JavaScript required
- Map options defined by query parameters
 - [https://maps.googleapis.com/maps/api/staticmap?
center=Plano,TX&zoom=13&size=300x300](https://maps.googleapis.com/maps/api/staticmap?center=Plano,TX&zoom=13&size=300x300)
- “key” param for api key
- Use it in an img tag



Static Maps Options



- [https://maps.googleapis.com/maps/api/staticmap?
parameters](https://maps.googleapis.com/maps/api/staticmap?parameters)
- Parameters
 - **center**
 - Latitude,longitude
 - Address string (url encoded)
 - **zoom**
 - 0 = earth, to 21
 - **size**
 - widthxheight (50x50)
 - **scale**
 - Numeric value increasing scale
 - **format**
 - png, gif or jpg
 - **maptype** (roadmap, satellite, terrain, hybrid)
 - markers, path, style, etc..

Option 2: Embed API



- Use an HTTP request to fetch a dynamic, interactive map
- Provides custom elements to visitor, based on their google account
- Embeds in an iframe
 - ```
<iframe
width="600"
height="450"
frameborder="0" style="border:0"
src="https://www.google.com/maps/embed/v1/place?
key=API_KEY&q=Space+Needle,Seattle+WA">
</iframe>
```

# Embed API parameters



- [https://www.google.com/maps/embed/v1/MODE?  
key=API\\_KEY&parameters](https://www.google.com/maps/embed/v1/MODE?key=API_KEY&parameters)
- Modes
  - place
    - Displays map pin at place or address
    - ?q=City+Hall,New+York,NY
  - directions
    - Path between two points
    - ?origin=PLACE&destination=PLACE&avoid=tolls|  
highways&units=metric
  - search
    - Result of a map search
    - ?q=my+search+string+in+place
  - view
    - Map with no directions or marker
    - ?center=long,lat&zoom=1&maptype=satellite

## Option 3: Google Maps API



- A robust javascript api for rendering google maps
- Recommend you declare your doctype as *html5*
- Based off the “google” object

# Basic Map Setup



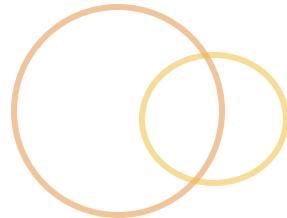
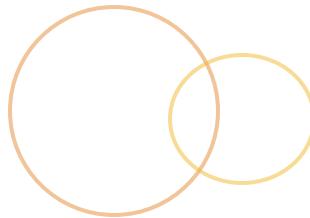
- Include the google maps api lib in your document
  - <script type="text/javascript" src="[https://maps.googleapis.com/maps/api/js?  
key=API\\_KEY](https://maps.googleapis.com/maps/api/js?key=API_KEY)"></script>
- Include a container div
  - <div id="map-canvas" style="width: 100%; height: 100%"></div>
  - Establishing height/width is important
- Then...

# Basic Map Setup, pt 2



- Define a config object for your map
  - ```
var mapOptions = {  
  center: {lat: -34.397, lng: 150.644},  
  zoom: 8  
}; // center and zoom are always required
```
- For each map, create an instance of Map
 - ```
var map = new
 google.maps.Map(document.getElementById("map-
 canvas"), mapOptions);
```
- <http://jsfiddle.net/mrmorris/1cLk13sj/>

# Markers



- Identifies a location on the map
- Support custom marker images, known as “icons”
- Use the `google.maps.Marker` constructor
  - ```
var marker = new google.maps.Marker({  
    position: myLatlng,  
    map: map, // from your map constructor  
    title: "Hello World!"  
});  
// Or use setMap()  
marker.setMap(map);
```
- <http://jsfiddle.net/mrmorris/1cLk13sj/>

Google Maps Api – More options



● Adjusting controls

- Just set control property to true/false in config object
 - zoomControl
 - panControl
 - scaleControl
 - mapTypeControl
 - streetViewControl
 - rotateControl
 - overviewMapControl
- Adjust appearance of controls with {control}Options
 - scaleControlOptions: {
 position: google.maps.ControlPosition.TOP_CENTER
 }
 - zoomControlOptions: {
 style: google.maps.ZoomControlStyle.SMALL
 }

Map methods



- `panTo(latlng)`
- `setCenter(latlng)`
- `setOptions(options)`
- `setZoom(zoom)`
- `setTilt(tilt)`
- **Getters**
 - `getZoom()`
 - `getTilt()`
 - `getCenter(); // LatLng return`

Map events



- Interactions with the map trigger events, which you can register listeners for and handle
 - <https://developers.google.com/maps/documentation/javascript/events>
- Use “google.maps.event.addListener”
 - `google.maps.event.addListener(map|marker, event, function(){...});`
- The map object
 - `zoom_changed, bounds_change, center_change, click, etc...`
- Markers
 - `click, dblclick, mouseup, mousedown, mouseover, mouseout`
- <http://jsfiddle.net/mrmorris/m7naxp6z/>

Styling the Map



- Define an object that includes sets of styles (stylers) for individual map features and their elements
 - {
 featureType: "road.local",
 elementType: "labels",
 stylers: [
 hue: "#00fee"
]
}
- But we're lazy
 - <http://gmaps-samples-v3.googlecode.com/svn/trunk/styledmaps/wizard/index.html>



Take it further



- ◉ Fancy demos
 - ◉ <http://www.morethanamap.com/>
- ◉ Try it out ourselves...

Exercise: Go google mappin'



- Create a page that has a button, “Track me”, that when clicked will get the user’s position and create a map, with a marker at the user’s location
 - Make sure you specify the height and width of your map canvas
 - Done? Use a special marker image
 - Too lazy? Here’s one:
[http://wiki.mabinogiworld.com/images/3/3f/
DevCAT_Hat.png](http://wiki.mabinogiworld.com/images/3/3f/DevCAT_Hat.png)
 - Bonus points for exploring the Places API to use alongside Google maps
 - Display nearby “sushi places” as markers

Designing for the web



- ◉ There are a lot of devices out there, with different capabilities and sizes, so what are we to do?
- ◉ Mobile-first
- ◉ Layout approaches
 - ◉ Static
 - ◉ Layout is at a fixed size, regardless of device
 - ◉ Fluid/Liquid
 - ◉ Scale the width of parts of the design relative to the window
 - ◉ Adaptive
 - ◉ Specific layouts for different viewport sizes
 - ◉ Media queries
 - ◉ Responsive
 - ◉ Layouts adapt to width but there are unique layouts for different viewport sizes
 - ◉ Combines adaptive and fluid
- ◉ <http://liquidaptive.com/>

CSS/Front-end Frameworks



- ◉ “Bootstrap” your front-end with css, grids, components/widgets, js libs, layout options and more...
 - ◉ Twitter’s Bootstrap
 - ◉ Foundation
 - ◉ 960 Grid
 - ◉ SemanticUI
 - ◉ UIKit
 - ◉ But wait, there’s more!
 - ◉ <http://usablica.github.io/front-end-frameworks/compare.html>

Media Queries



- Control which css is applied for different media, viewport size and orientations
 - <media type> and <expression(s)>
- Will apply the css if the media matches and the expressions evaluate to true
 - @media all and (orientation:landscape) {}
 - @media (min-width:200px and max-width: 600px) {}
- Complex queries with “not”, “and” and “only”
 - not and only require a media type
 - @media not all and (monochrome), print and (color) {}
 - “not” affects only first query statement
- min-, max- indicates “greater than or equal to” and “less than or equal to”
 - min-width:500px; /* if width is greater than or equal to 500 */

Media Queries continued



- Media Types
 - all, print, screen, speech (all others are deprecated)
- Expression types
 - color, monochrome
 - aspect-ratio
 - width, min/max-width (viewport, rendering area)
 - height, min/max-height
 - orientation (landscape or portrait)
 - ~~min/max-device-height (output device screen or page)~~
 - ~~min/max-device-width~~
 - min/max resolution (dpi, ppx)
- Read up:
 - <https://developer.mozilla.org/en-US/docs/Web/CSS/@media>
- Example:
 - <http://jsbin.com/kohuda/3/>

Viewport meta



- Mobile browsers will scale page to fit, usually requiring a zoom in...
- Mobile safari introduced viewport meta tag
 - <meta name="viewport" content="width=device-width, initial-scale=1">
 - "width" controls size of the viewport
 - pixels or "device-width"
- initial-scale controls the zoom level
- maximum-scale, minimum-scale and user-scalable ("yes" or "no") control zoom permissions
- Back to our example: <http://jsbin.com/kohuda/3/>

Detect the device



- Could check the user agent...
 - <http://detectmobilebrowsers.com/>
 - Script for all agent patterns
- Feature detection is recommended over device detection
 - Easier to support features versus many devices
 - Modernizr.touch

When targeting mobile



- ◉ Mind your bandwidth
- ◉ Mind your touch
 - ◉ Devices depend on tap, swipe, double tap
 - ◉ Browsers are different
 - ◉ Accuracy can also be a problem
- ◉ Mind your device size
 - ◉ Not everyone has big phones... yet
 - ◉ Or big resolution
- ◉ Mind your device
 - ◉ Not everyone uses an iPhone

Simulating Mobile (to test)



- Actual simulators
 - Opera Mobile Emulator
 - <http://www.opera.com/developer/mobile-emulator>
- BrowserStack
- Local network and use your phone
- Test in the browser
 - Just scale it down (quick and dirty)
 - Device mode in Chrome
 - <https://developer.chrome.com/devtools/docs/device-mode>

Exercise – Media queries



- Set up a simple page that responds to the width of the browser window
 - Add a nav list that sits horizontally when the screen is 500px or greater, and that stacks vertically when the screen is less than 500 px wide.
 - <http://jsfiddle.net/mrmorris/tj8r05jj/>
- Make sure you include the correct viewport meta (if not in jsFiddle)
- Test it in Chrome's simulator

Shims & polyfills



- ◉ A “shim” is a piece of code (or other technology) that fills a gap in existing functionality
- ◉ A “polyfill” is a collection of shims that make a browser act as if it had a missing feature (or group of features)
 - ◉ Sometimes referred to as “regressive enhancement”
- ◉ Fortunately, there’s a polyfill for pretty much all features:

<https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-browser-Polyfills>

Modernizr



- JavaScript library that detects HTML5 and CSS3 features in the user's browser
- Features are then reflected in the BODY's class attribute, e.g. 'draganddrop' or 'no-draganddrop'
 - Features can also be programmatically checked via property on the Modernizr object, e.g. Modernizr.draganddrop
- Automatically enables non-HTML5 compliant browsers to support HTML5 sectioning elements
- Has a built-in resource loader that enables fallbacks for when features are not detected on a browser
- Used by Twitter, Google, Microsoft and more
- <http://modernizr.com>

Using Modernizr



- Include the script
- Set default state of html to no javascript
 - <html class="no-js">
 - Modernizr, if run, will convert it to something like this:
 - <html class="js canvas canvastext geolocation rgba hsla no-multiplebgs borderimage borderradius boxshadow opacity...">
- Class fixes
 - .no-multiplebgs vs multiplebgs
- Javascript checks
 - if (Modernizr.geolocation) {}
- Use Modernizr.load to conditionally load shims
 - Modernizr.load({
 text: Modernizr.geolocation,
 yep: 'geo.js',
 nope: 'geo-polyfill.js'
}, 'runaftereverythingelse.js');

SASS and LESS



- Pre-processors for CSS
- Extend CSS with dynamic behavior such as variables, nesting, mixins, operations and functions
- Written in JavaScript
- Greatly speeds cross-browser development
- LESS
 - Written in JavaScript, installed through node
 - <http://lesscss.org>
- Sass
 - Ruby gem
 - <http://sass-lang.com/>

Twitter Bootstrap



- HTML5/CSS3 framework built by Twitter to standardize UI
- Great performance, and looks the same on desktop and handheld devices
- Built-in grid system, components, typography, form controls and jQuery-based behaviors
- Extendable via SASS/LESS, very large theming community
- Version 3 (latest) was written mobile first
- <http://jsbin.com/zabeko/1/edit?html,js,console,output>
- <http://getbootstrap.com/>

Give Bootstrap a go



- <http://getbootstrap.com/getting-started/>
- Drop in the bootstrap js and css file
 - They recommend using two shims, Respond.js for media queries and Html5 Shim for html5
- Doctype is html5, include the viewport meta
- Wrap everything in a “container” div

Exercise: Using Bootstrap



- Lets make this form a bootstrap form:
 - <http://jsbin.com/wexusa/1/edit>
 - Or fiddle: <http://jsfiddle.net/mrmorris/st1zddzz/>
- Use the docs
 - <http://getbootstrap.com/>
 - Ok, ok, hand holding:
 - <http://getbootstrap.com/css/#forms>
- Done?
 - Now set it up in a tabbed ui w/ Bootstrap
 - <http://getbootstrap.com/javascript/#tabs>

jQuery Mobile



- Mobile focused front-end component framework
- Built on jQuery and jQueryUI

jQuery Mobile Setup



- Get the files
 - <http://jquerymobile.com/>
 - Or CDN: <https://code.jquery.com/mobile/>
- Set up the html
 - Add jquery mobile and jquery scripts
 - Add jquery mobile theme css
- Set at least the data-role attribute with a value of page
 - <div data-role="page">
- jQuery Mobile also has a theme roller
 - <http://themeroller.jquerymobile.com/>

Basic page setup



- ◉ [http://jsbin.com/mihori/2/edit?
html,js,console,output](http://jsbin.com/mihori/2/edit?html,js,console,output)

jQuery UI vs jQuery Mobile



- ◉ Widgets are created automatically based on data attributes
- ◉ Typical elements are all styled automatically
- ◉ Supports multiple “pages” per document, hence the data-role=“page”

Page Events



- Events that run for every “page” being loaded/displayed (not just a full page refresh)
- **mobileinit**
 - When jQuery Mobile is loaded but before it has enhanced the page
 - Use this to modify global config options
- **pagecreate**
 - This event occurs after DOM is loaded and jQm has set up its widgets
 - `$(document).on("pagecreate", function(e) {});`
 - Prior to 2.14, pageinit was used
- **pageinit**
 - Careful of setting up event handlers here, since this is re-run for every page load, resulting in additional handlers
 - Use a “isHandlerAdded” var outside the init, set it to true in the init

Touch Events



- Modern browsers support some
 - Touchstart, touchend, touchmove, touchcancel
- jQuery Mobile adds gesture events to assist with some of the lower level events
 - tap, taphold, swipe, swipeleft, swiperight
- Virtual mouse events
 - Help to avoid having to duplicate touch and mouse events
 - vmouseover, vmousedown (touchstart/mousedown), vmousemove (touchmove, mousemove), vmouseup (touchend, mouseup), vclick (click), vmousecancel (touchcancel, mousecancel)
- orientationchange on window

jQuery Mobile Pages



- Page

- <div data-role="page"></div>
- Can have multiple, just give them ids and can link between them with href="#idname"

- Header

- <div data-role="header"></div>

- Content

- <div data-role="content"></div>

- Footer

- <div data-role="footer"></div>
- Use data-position="fixed" to lock to the bottom

Page Transitions



- Use ids to reference pages within content
- Default is “slide” but several others like pop, slideup, fade, flip
- Use data-transition and data-direction=“reverse”

External pages



- ◉ Uses ajax to load new content by default
 - ◉ Automatically handles adding scripts
- ◉ What about conflicting ids?
 - ◉ One page per document (suggested by jQM team)
 - ◉ Disable Ajax loading for that link
 - ◉
 - ◉ Or globally with ajaxEnabled setting
 - ◉ \$.mobile.ajaxEnable = false
- ◉ You can prefetch
 - ◉ Link
 - ◉ But its hard to test, adds overhead

Page setup



- ◉ Pages are quite flexible, but careful with ajax loading
- ◉ jQm will cache previously fetched pages
- ◉ Basic example w/ ajax
 - ◉ <http://js-jquery.course/jquery-mobile.html>

jQuery Mobile - Widgets



- Dialog
 - With buttons
- Popup
 - Lightweight alternative to dialogs with more programmatic control
 - Displayed when user clicks a control with data-rel="popup" href="#idofpopup"
- Buttons
 - Grouped
 - Icons
 - Inline
 - Events
- Navbars
- Collapsible content blocks
 - Sets (accordions)
- Form widgets
- ListView
 - ListView filter
 - Dividers
- Panels
- Let's check the demos: <http://demos.jquerymobile.com/1.4.4/>

Why jQuery Mobile?



- ◉ A quick (albeit basic) framework for getting a mobile app-like site up and running
- ◉ Ajax page loading is nifty
- ◉ Theming is handy
- ◉ Additional support for touch events (though this isn't limited to jQuery mobile)
- ◉ PhoneGap supported
 - ◉ So could port it into an app

PhoneGap?



- Packages up your javascript/html “app” into a WebKit UI View
 - Basically a chromeless browser
- Also adds apis for mobile specific functionality
- <http://phonegap.com/>

Exercise – Try out jQuery Mobile



- If we have time lets give it a run
- Set up a basic “mobile friendly app” with 3 pages
 - Try setting up one page with links as Buttons
 - <http://demos.jquerymobile.com/1.4.4/button-markup/>
 - Another page as links with a ListView
 - <http://demos.jquerymobile.com/1.4.4/listview/>
 - And the third as a navbar
 - <http://demos.jquerymobile.com/1.4.4/navbar/>
- Bonus: have one page set up to load via ajax
(use your local server)

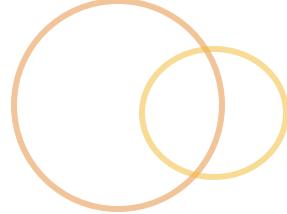
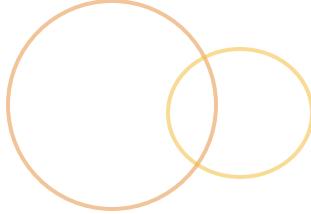
Testing



- Why test?
 - Reduces complexity with test-assisted design
 - Sanity, calm
 - Browser testing simplified
 - Self-documenting
- For bugs...
 - Write a test showing the bug
 - Fix the bug
 - Test remains as regression test
- Tests should be self-contained

The QUnit logo consists of the word "QUnit" in a bold, dark red sans-serif font, enclosed within two overlapping circles. The top circle is light orange and the bottom circle is yellow.

QUnit



- Powerful unit testing framework for JavaScript
- Built and used by jQuery* team
- Allows to test code, plugins, modules through assertions
- Extendable
 - JUnit Logger
 - Composite Test Harness
 - PhantomJS browser-less testing
- How you write your tests is up to you
 - TDD

Qunit – Getting Started



- <http://qunitjs.com/>
- Has an html file with css and a single js file – this is your browser-based test runner
- #qunit-fixture is for DOM fixtures used by tests and is reset after each test
 - <http://jsfiddle.net/mrmorris/A9G2L/>
- Terminology
 - Test Unit
 - Assertions

Qunit – The structure of a test



- `QUnit.test("hello world test", function(assert) {`

```
// verify your code units do what  
// they are supposed to do
```

```
    assert.ok(myCode(), "Passed!");
```

```
});
```

- Keep tests atomic
- Qunit may reorder tests while it runs them

Qunit – Testing User Actions



- Code that relies on a user interaction often has some anonymous handler function that needs to be invoked... need to simulate this user triggered event
- Use jQuery's “trigger()” to trigger an event
 - `$(el).trigger(eventName);`
- Use “triggerHandler()” to ONLY run the event’s handlers rather than simulate a native event
 - `$(el).triggerHandler(eventName);`
- Customizing the Event object
 - `var event = $.Event("keydown");
event.keyCode=9;
$(document).trigger(event);`

Qunit - Assertions



- Assertions are essential to any test
- Check expected results against actual
- Tell Qunit how it should work and Qunit confirms
- Qunit has 8 assertions
 - ok
 - equal (==)
 - deepEqual
 - notEqual
 - notDeepEqual
 - propEqual
 - notPropEqual
 - strictEqual
 - notStrictEqual
 - throws

Assertions continued...



- `ok(truthy[, message]);`
 - One argument required
 - If it evaluates to truthy, it passes
- `equal(actual, expected[, message]);`
 - Uses simple `==` comparison
 - Use `strictEqual()` for strict `==`
- `deepEqual(actual, expected [, message]);`
 - Can use `equal` in most cases, however for an object it will only check object identity (not all props)
 - Uses `==` strict
 - Compares object properties recursively

Qunit Fixtures



- ◉ When interacting with the DOM you can use the `#qunit-fixture` element
- ◉ It gets reset after each test()

Qunit – Setup/Teardown



- ◉ Qunit modules support “setup” and “teardown” methods, which are run before and after each test respectively
- ◉

```
Qunit.module("module", {  
    setup: function(assert){...},  
    tearDown: function(assert){...}  
});
```

Qunit - Modules



- Use “modules” to group tests together
 - `QUnit.module("my module");`
`//... tests...`
`QUnit.module("Another Module");`
- You can selectively run tests or modules using the test runner

Qunit – Mocking, Stubs and Spies



- Eventually you'll need to fake something
 - A library you do not want to be testing
 - Server requests, which will slow down tests and muddy data
 - Slow scripts or actions
- Can “fake” ajax requests
 - Set up your own helper method to return data
- Spies
 - Allow you to monitor a function
 - Can record arguments, return values, calls, context
 - Useful for callback testing
- Fakes
- Stubs
 - Functions with pre-programmed behavior
 - Typically to force a behavior for the test or to prevent something from happening (ajax) during a test

Qunit - Running



- Run tests in your browser (or multiple browsers)
- Can run console test runner
 - PhantomJS – headless browser
 - TestSwarm
- Run specific tests with “?filter”
 - Specific Modules “?filter=foo”
- Avoid global pollution
 - ?noglobals
 - Will enforce and fail tests if globals are created

Qunit - Testing



○ Synchronous callbacks

- When a callback has an assertion that is in danger of never being called, you can use “expect()” to define the number of assertions to expect to have been called.

○ Asynchronous callbacks

- Expect() falls short here
- Use asyncTest() (instead of test()) and call .start() when your codeblock is complete
- ```
Qunit.asyncTest("my test", function(assert) {
 expect(1);
 $("video").on("canplaythrough", function(){
 assert.ok(true);
 Qunit.start();
 });
});
```

# Qunit – Custom Assertions



- Extend “assert” object
- Basically functions that call Qunit.push(), which is how Qunit detects that an assertion has taken place and tracks the result
  - ```
Qunit.assert.contains = function(needle, haystack, msg){  
    var actual = haystack.indexOf(needle) >= 1;  
    Qunit.push(actual, actual, needle, msg);  
};
```

QUnit Core



- Expectations
 - <http://jsfiddle.net/mrmorris/5L0y5ewu/>
- Assertions
 - <http://jsfiddle.net/mrmorris/qhj8z22t/>
- Atomic tests
 - <http://jsfiddle.net/mrmorris/kntzL9b8/>
- Fixtures
 - <http://jsfiddle.net/mrmorris/58w0ksp0/>
- Modules
 - <http://jsfiddle.net/mrmorris/3oxy4s1t/>
- noglobals
 - <http://jsfiddle.net/mrmorris/6u169arw/>
- Callbacks
 - <http://jsfiddle.net/mrmorris/ygj4jL0j/>

QUnit Asynchronous Tests



- ◉ Bad
 - ◉ <http://jsfiddle.net/mrmorris/n8uqwf7w/>
- ◉ Good
 - ◉ <http://jsfiddle.net/mrmorris/hd1vtwtc/>
- ◉ Mocked
 - ◉ <http://jsfiddle.net/mrmorris/q2qep0sz/>

QUnit Lab



- <http://jsfiddle.net/mrmorris/2jwLm3v6/>
- Let's write tests that ensure that this plugin works correctly
- Tests should verify that the plugin is chainable, that it prepends the correct number to each item and that you can specify which number to start enumerating from

Alternatives to qUnit?



- Jasmine is a nice BDD test framework
 - <http://jasmine.github.io/2.0/introduction.html>
 - More expressive/descriptive
 - Behavior focused (versus code focused)

AngularJS & Node Introduction



AngularJS



- Client-side framework
 - “structural framework for dynamic web apps.”
 - MVC (or MVVM)
- Augments html using JavaScript
 - Doesn’t try to mask that it’s in html
 - Makes apps more expressive
- Why is it nice?
 - Separation of application logic, data and views
 - Built in ajax services and dependency injection
 - Excellent testing
 - Modular (becoming more so)

The beginnings of Angular



- Started in 2009 as commercial project, GetAngular
- Replicated an app that consisted of 17k lines of code and 6 months of work in 3 weeks...
- Google sponsored as AngularJS

AngularJS



- Why AngularJS?

- Open source, great community
- It does a lot for us, allowing us to focus on core application business logic
- Pure JavaScript and HTML
- Enhances the HTML
- Built-in testing practices
- Single-page app, if you're into it

Alternatives to Angular



- ◉ Plenty of new frameworks out there
- ◉ Ember.js
 - ◉ Depends on convention over configuration
 - ◉ Great router and data module
- ◉ Backbone.js
 - ◉ Lightweight, small footprint
 - ◉ Fewer concepts to grasp
 - ◉ No structure
 - ◉ more of a foundation for your own framework

How it fits in the stack



- ◉ Nothing special, just a JavaScript library to include
- ◉ Augments html, giving behavior, snippets and functionality to html fragments.
- ◉ A pattern for organizing your application using controllers, models and views
- ◉ A test suite for unit testing and end-to-end testing, if needed (command line test runners)
 - ◉ Karma
 - ◉ Protractor

Core Concepts



- Data-driven via data-binding
 - Keeping view data in sync with model data
- Declarative
 - Easier to see intent
 - Encapsulated; logic is contained within the directive
- Separation of concerns
 - MVC
- Dependency Injection
 - Attempts to simplify managing instances of objects
 - Makes testing much easier
 - Modularity
- Scope
- Extensible HTML
- Test first

Terminology of an Angular app



- Module(s)
- Controllers
- Directives
 - Augment the HTML
- Expressions
- Filters
- Services, Providers and Factories
- Data-binding (one and two way)
- Dependency Injection

Using Angular



- Get the files or use the CDN
 - Angular js <https://angularjs.org/>
 - Maybe bootstrap + jQuery
- Bootstrap the app space
 - <html ng-app></html>
- <http://jsfiddle.net/mrmorris/wek3r3fq/>

Our first Angular app



```
○ <!DOCTYPE html>
<html ng-app>
<head>
    <title>Simple app</title>
    <script src="angular.js"></script>
</head>
<body>
    <input ng-model="name" type="text"
placeholder="Your name">
    <h1>Hello {{ name }}</h1>
</body>
</html>
```

What's happening here



- <!DOCTYPE html>
<html **ng-app**> ← establish our *module*
 <head>
 <title>Simple app</title>
 <script src="angular.js"></script>
 </head>
 <body> Two-way binding "name" var
 <input ng-model="name" type="text"
placeholder="Your name">
 <h1>Hello {{ name }}</h1> ← expression
 </body>
</html>

Quick overview



- **Modules** define our application space
- Angular supports **data-binding**
- **Expressions**, wrapped by {{ }}, are JavaScript like snippets that also support data binding of variables
 - {{5 + 6}} -> 11
 - {{“Hello “ + name}} -> Hello Ryan (and “live”)
 - {{data.user.id}}
 - {{myFunction()}}
- ng-model and ng-app are **directives**, augmenting our html with behavior

Modules, encapsulate our app



- ◉ Modules are how we create a scope for our application
 - ◉ The primary way to define an App
 - ◉ App is not restricted to one module, however
- ◉ Group logical pieces of functionality into modules
- ◉ Makes it possible to share modules between apps
 - ◉ Easier to share code between apps
- ◉ A module can define its own controllers, services, factories and directives
- ◉ A module can depend on other modules as dependencies
 - ◉ Angular will automatically fetch dependencies and inject them

Define a module



- `var myApp = angular.module("myApp", []);`
 - Name + Dependencies array
 - Can fetch a module by excluding dependencies arg
- Then tell your app to utilize that module for a portion of your view (or all of it)
 - Use the `ng-app` **directive**
 - `<div ng-app="myApp">`
- <http://jsfiddle.net/mrmorris/ptuj67m0/>

Controllers



- ◉ Workhorse for majority of business logic and UI-oriented work
 - ◉ Fetching data from the server for the UI
 - ◉ Determine data to display
 - ◉ Presentation logic
 - ◉ User interactions
- ◉ Augment the views of our application
- ◉ It's a function that adds behavior to the **scope** of the view
 - ◉ Sets up initial state
 - ◉ Has custom behaviors for the scope

Controller setup



- Define a controller function and link it to the html
 - Use the ng-controller **directive** in the html
 - Define a controller function
 - // super basic

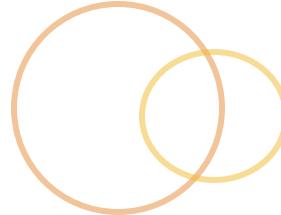
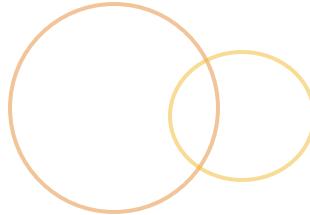
```
function MainController() {  
    this.name = "Jim";  
}
```
 - Naming convention: *CapitalNameController*
- Few ways to do this...
 - Define it as a standalone function
 - Define it on your module (better)
 - Use *Array notation* for dependencies (best)
 - `app.controller(Name, ['dep1', 'dep2', function(dep1, dep2) {}]);`
- <http://jsfiddle.net/mrmorris/41mbjxbo/>

Controller scope



- Controllers have their own “scope”
 - Provides the context for view expressions
 - The glue between the controller and view
 - The scope in a controller behaves like the “view model”
- Controllers inherit scope
 - Nested controllers inherit from their parent controllers
 - Root Scope
 - Controller Scope A
 - Nested Controller B
 - Directives
 - Controller Scope C
- Scope events
 - Scoped variables can be watched (\$watch) for changes
 - Can also have changes applied (\$apply) throughout the app
 - That's how binding works behind the scenes

\$scope



- ◉ **\$scope** object (**service**, in Angular)
 - ◉ It **is** the data model in Angular
 - ◉ Connection between view and html and the glue between the view and controller
 - ◉ Both controllers and directives have access to \$scope, but not eachother
 - ◉ *Must be injected to controllers*
- ◉ All data set in the \$scope object is automatically accessible to the view
- ◉ **\$scope** inherits from parent \$scope
 - ◉ Controllers inside controllers, up to...
- ◉ **\$rootScope**
 - ◉ Based on ng-app root
 - ◉ Parent of all \$scope objects
- ◉ **\$parent** references scope one up the tree
- ◉ <http://jsfiddle.net/mrmorris/bkw5e9hb/>

\$scope vs “controller as”



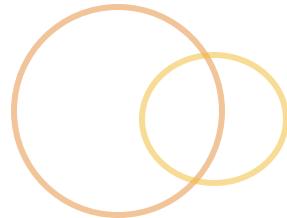
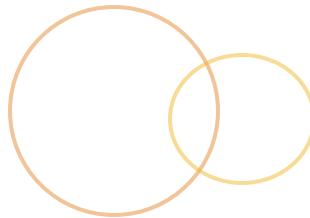
- Two approaches for managing scope
- \$scope (service-like)
 - Old way
 - <div ng-controller="MyCtrl">
 - app.controller("MyCtrl", ["\$scope", function(\$scope) {
 \$scope.name = "My Name";
}]);
- this (“controller as alias”)
 - New since 1.2
 - Binds controller to current \$scope
 - <div ng-controller="MyCtrl as ctrlAlias">
 - app.controller("MyCtrl", [function() {
 var self = this;
 self.name = "My Name";
}]);
 - Nested scopes are easily referenced
- <http://jsfiddle.net/mrmorris/vsj0uapd/>

Exercise – Basic controller



- A basic controller that allows a user to input their name and favorite color
 - Set up a function ON your controller that returns a value
 - Try invoking your function in an expression, too
 - Add a button element and give it an “ng-click” directive
 - `ng-click="anotherFunction()"`
 - `anotherFunction` should just log something to the console
- Start here:
 - <http://jsfiddle.net/mrmorris/bao9hrnb/>
 - No fiddling?
 - CDN:
<https://ajax.googleapis.com/ajax/libs/angularjs/1.2.26/angular.js>

ng-repeat



- A most powerful directive
 - ng-repeat="eachVar in arrayVar"
- Can repeat over an object's properties
 - ng-repeat="(keyVar, valVar) in objectVar"
- Helper variables in ng-repeat
 - \$first, \$middle, \$last, \$event, \$odd => bools for current item position
 - \$index => integer of current position
- ng-track
 - "eachVar in arrayVar track by eachVar.prop"
- <ng-repeat-start=""><ng-repeat-end>
- <http://jsfiddle.net/mrmorris/901pv7fp/>

Loading sequence



- Html loads, including script resources
- Angular bootstraps first ng-app
 - Angular loads the module
 - Handles directives and binds
 - Generates scopes for controllers and repeats
 - Then adds listeners to watch for bound variable changes
- ng-cloak
 - A directive that hides content until angular finishes bootstrapping
 - Prevents {{ }} from displaying until handled
 - Or use ng-bind instead of {{}} where possible

UI Directives



- ng-show, ng-hide
 - Hide or show an element based on the value of an expression being truthy or falsy
- ng-class
 - Evaluates class from result of expression
 - When result is an object, uses first truthy valued “key”
- Why not just class="{{expression}}"?
 - Because it would evaluate the expression once and you'll miss all the binding fun!

Data Binding



- ◉ The coolest!
- ◉ Establishes a dynamic connection between view data and model data
 - ◉ `{{ expression }}`
- ◉ As an expression value changes, so does the DOM
- ◉ The view becomes a live projection of the model state

Data Binding continued



- One-way binding
 - {{expression}}
 - ng-bind="expression"
- Two-way binding
 - <input ng-model="name"> Name: {{name}}
 - Input with ng-model="varname"
 - And an expression or usage {{varname}}
- Best practice: utilize binding on object properties, rather than single vars
 - user.name vs name
 - Clean scope and better inheritance

Data Binding optimization



- ◉ Angular monitors if a value has changed through its event loop, “dirty checking”
- ◉ Optimizes how it watches for bound data changes
 - ◉ UI events may trigger a “check”
 - ◉ Input changes
 - ◉ Ajax requests

Forms in Angular



- ng-bind to bind form data to controller
- ng-submit for submission handler
- Take a look
 - <http://jsfiddle.net/mrmorris/52yyL23x/>

Form validation and state



- Utilizes form and input “name” attributes
- Makes use of html5 input types/attributes
 - <form novalidate></form>
- By setting the correct attributes, input types and utilizing some additional validation directives, angular will track individual field validation and form validation
 - An object with validation issues
 - Classes set to indicate validation issues

Form validation toolbox



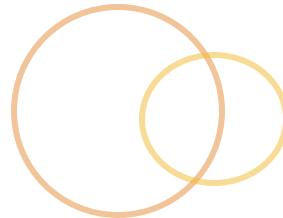
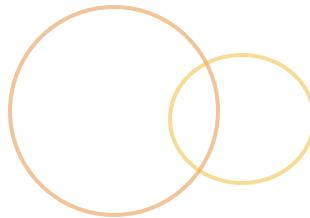
- Attributes
 - required, type="email", type="number", type="date", type="url"
- Validation directives
 - ng-required, ng-minlength, ng-maxlength, ng-pattern
- States on the form
 - \$invalid, \$valid, \$pristine, \$dirty, \$error
- Classes set on forms and inputs
 - ng-invalid, ng-valid, ng-pristine, ng-dirty
- Classes set on inputs
 - ng-valid-{rulename}, ng-invalid-{rulename}
 - ex: ng-valid-required, ng-invalid-required
- <http://jsfiddle.net/mrmorris/qctrvrLn/>

Additional form elements



- Nest forms with <ng-form name="aName">
 - Access as a sub-set in the main form
 - myForm.aName
 - Allows for grouping validation sets and sub-states
 - myForm.aName.\$invalid
- Checkboxes
 - ng-checked="expression"
 - ng-true-value, ng-false-value (if not a boolean)
- Can initialize an input value
 - ng-value="expression"
- Select lists
 - ng-options="keyVal as valVal for item in array"

Services



- Functions or objects that hold behavior or state across the app
- Shared singletons
 - Lazy loaded once, automatically injected
- Can be implemented as a *factory*, *service* or *provider*
- Good for...
 - Server calls, api usage, shared validation, shared data, shared business logic
- Built-in services
 - \$log, \$window, \$location, \$http

Dependency Injection



- Heavily used in Angular
 - Avoid re-instantiating objects
 - Avoid dependence on objects
- Injecting dependencies
 - `myApp.controller("MainCtrl", ["$scope", "$log",
function($scope, $log) {}]); // preferred`
 - `myApp.controller("MainCtrl", function($scope, $log) {});`
 - `myApp.controller("MainCtrl", ["$scope", "$log",
function(s, l){}]);`
 - Order matters...
- <http://jsfiddle.net/mrmorris/a60zhhdv/>

Creating a Service



- Typical reasons for a service
 - Reusable
 - Store shared data or state
 - Independent of the view
 - Integrates with a 3rd party service
 - A model factory or data cache
- App.factory('ServiceName', [function() {
 // privates
 return {
 method: function(){}
 };
}]);
- <http://jsfiddle.net/mrmorris/ofyh5huL/>

Factories, Services, Providers, oh my



- Three ways to define and register “services”
- Service
 - More OO-oriented, class-style
 - Return nothing, work on the instance (this)
- Factory
 - Functional style
 - Return functions and objects
- Provider
 - Allows for configuration of service before app loads
 - More advanced
- <http://jsfiddle.net/mrmorris/hrjgtuuv/>

Server communication



- \$http service
 - Or ngResource module for bootstrapping RESTful endpoints
- Uses Promises
 - ```
promise.then(successHandler(data) {
}, errorHandler(errData) {
});
```
  - Can chain “then” methods.
    - Use \$q.reject(data) to trigger error handler of next promise
    - Chained promises will wait for the promise result before proceeding to next promise
  - Non-blocking

# Using the \$http service



- <http://jsfiddle.net/mrmorris/zkxvwy97/>
  - <http://angular-test1.course/sample-http1.html>
- \$http methods
  - get(url, config)
  - post(url, data, config)
  - delete
  - head
  - put(url, data, config)
  - jsonp
  - \$http(config)
- Config
  - headers: object
  - method: string
  - url: string
  - withCredentials: boolean
  - transformRequest and transformResponse: functions
- Wrap it in a service (see sample2)
  - <http://jsfiddle.net/mrmorris/egc17923/>

# Filters



- Ability to process & format the data being displayed
- Applied to data in expressions (in the view) or directly on data in controllers (via function call)
  - Filter expression
    - “`data | filter: option`”
    - `{{ “My String” | lowercase }}` // displays “my string”
    - `{{ 55.25234 | number:2 }}` // displays 55.25
  - Filter as function in your controllers
    - Use \$filter service
    - `$filter(‘lowercase’)(‘My Word’);`
- Can stack filters by piping
  - `var | number:2 | currency`

# Built-in filters



- Formatting
  - uppercase, lowercase, number, currency, date, json
- Array filters
  - limitTo (negative to work from the end of the array)
  - orderBy
    - Order by a expression (string, function or array)
  - filter
    - Accepts a string as a matcher
    - Or an object to verify property matches
    - Or a function, returns false to drop item
- <http://jsfiddle.net/mrmorris/wg1rbpsy/>

# Using filters in your controller



- Inject '\$filter'
  - \$filter('filtername')(arg1, arg2);
- Inject '{nameOfFilter}Filter'
  - numberFilter(item, 2);

# Custom filters



- Create new filters
  - ```
myApp.filter('capitalize', [function() {  
  // privates, shared by single instance of this filter  
  return function (input, opt1, opt2) {  
    // return value to use  
  }  
}]);
```
- Return a filter function, which is what will be called when filtering
- Array filters (for ng-repeat) will need to iterate over array
 - ```
angular.forEach(items, function(item) {})
```
- <http://jsfiddle.net/mrmorris/wg1rbpsy/>

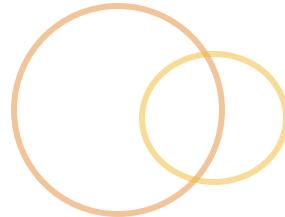
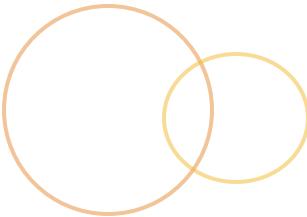
# Gotchas



- Directive attributes do not need {{ }} expression wrappers
  - `ng-show="{{expression}}"`
  - `ng-show="expression"`
- \$timeout and \$interval

The jqLite logo consists of the word "jqLite" in a bold, dark red sans-serif font. It is partially enclosed by two overlapping circles: a larger orange circle on the left and a smaller yellow circle on the right.

jqLite



- Minimal jQuery lib
- \$element or angular.element is an alias for \$/jQuery
  - angular.element(element); // ONLY html string or dom element
- Only common methods
  - addClass(), hasClass(), removeClass(), toggleClass()
  - html(), text()
  - after, append,
  - attr(), css(), removeAttr(), val()
  - eq(), find()
  - on(), off()
- And some angular-specific methods
  - .controller(), controller(name); // directive name
  - .scope()
  - .injector()
  - And a few more

# Can I still use jQuery?



- ◉ Yup, just load it before Angular.js
- ◉ But...
  - ◉ Its philosophy does not align with AngularJS
  - ◉ Angular encourages extension of HTML vocabulary, not traversal/manipulation of existing HTML
  - ◉ Only directives should be manipulating the DOM
  - ◉ Most event handling is taken care of

# Routing



- Routing allows us to take a single page application to the next step, loading different views based on the url
- ngRoute module
- Handles all the painful stuff like
  - Browser history management
  - Maintaining state
  - Loading view templates and the js along with it
- Uses hashbang urls
  - The hash fragment
  - Avoids additional full page requests

# Using ngRoute



- Get the module file
  - <https://code.angularjs.org/1.2.26/>
- Include it in your page
- Include it as a dependency in your module
  - `angular.module('myApp', ['ngRoute']);`
- Indicate which section of the page changes based on the route
  - `<div ng-view></div>`
    - Note: only one usage in the app
- Define your routes in the config using `$routeProvider` service
- Links will reference “#/{}url”

# Defining routes



- `$routeProvider.when(url, config)`
  - url is a url string or regex
  - config contains options about what to do when the route is used
- `$routeProvider.otherwise(config)`
  - What to do for a non matching route
  - `{redirectTo: '/'}`
- A super simple example
  - <http://angular-test1.course/sample-routing1.html>

# More routing “when” options



- url
  - String or regex
  - “/item/:id” indicates a param to pass to \$routeParams
- The option object
  - template -> html string
  - templateUrl -> html partial
  - controller
    - Can specify controller if not already defined in html partial
    - “MainCtrl as ctrl” is also ok
    - OR define the controller inline, passing the controller function and array syntax for DI
  - controllerAs
    - Alternative for a controller alias definition
  - redirectTo -> url string
  - resolve
    - Handle tasks before a route is loaded, ex: authentication

# But I don't like the hash!



- Use the \$locationProvider in config()
- Set html5 mode
  - \$locationProvider.html5Mode(true)
- Set hashPrefix
  - \$locationProvider.hashPrefix('!')
- Set base url in <head> for resource loading
  - <base href="/" />
- <http://angular-test1.course/sample-routing-html5mode.html>

# Directives



- Augmenting HTML
  - Re-using HTML snippets
  - Adding behavior to the page (ng-show/ng-hide)
  - UI Widgets
- Fall into two sets
  - Behavior modifiers
  - Reusable components

# Component-ish Directives



- ng-include
  - Result of an expression can be a path to an html snippet
  - <div ng-include="expression"></div>
  - <div ng-include="url/to/partial.html"></div>
- ng-switch
  - ng-switch="expression"
    - ng-switch-when="value"
    - ng-switch-default
  - Comments out markup, doesn't just hide it
- <http://angular-test1.course/sample-directives1.html>

# Create a directive



- Define a function that returns a directive config
  - ```
app.directive('myDirective', [function() {  
    return {  
        // directive config  
    }  
}]);
```
- Then we can use the directive in our html
 - `<div my-directive></div>`
- <http://jsfiddle.net/mrmorris/eet4f99c/>

Custom directive options



- template
- templateUrl
- restrict: AECM
 - attribute, element, class name, comment
 - ```
app.directive('myDirective', [function() {
 return {
 templateUrl: '/views/partial.html',
 restrict: 'AE'
 }
}]);
```
- can be used as an Attribute or Element
  - <div my-directive></div>
  - <my-directive></div>
- link: function(\$scope, \$element, \$attrs){}
- scope: boolean|object
  - by default it *is* parent's (false), but can set as true to get its own child scope, or an object representing a new scope
  - Object will create an isolated scope with ability to reference outside

# App structure



- ◉ Group by functionality or component, not by type
- ◉ Tests should mirror the app
- ◉ Examples
  - ◉ <https://github.com/mgechev/angularjs-style-guide#directory-structure>

# Bootstrapping



- Angular seed:
  - <https://github.com/angular/angular-seed>

# Chrome plugin



- Batarang chrome extension for debugging AngularJS apps
  - Model browsing (models bound to scopes)
  - Dependency graphing
  - Performance analysis

# Testing Angular apps



- You'll need node
- Karma
  - Test runner
  - Can utilize any framework
- Jasmine
  - Unit test framework
- Protractor
  - End-to-end testing

# Best practices



- Group files by feature, rather than type
- Group features into modules (not by type)
- Use array syntax for defining dependencies
  - Careful, order of dependencies matters
- Don't manipulate the DOM in your controllers
- Keep controllers trim
- Services should handle data and business logic (unless its temporary data)
- Service, Factory and Provider are effectively the same thing, just with different sugar
- Write tests!

# Angular 2.0?

- Big syntax changes
- No more jQlite



# Exercise – Angular apps away!



- Get to know Angular

- <http://jsfiddle.net/mrmorris/4Ldnpq2m/>
- Set up a module (ng-app)
- Set up a controller (ng-controller)
- Set up an array of objects (users, todo-items, books)
  - Adventurous? Use \$http service and an api endpoint from here: <http://jsonplaceholder.typicode.com/>
- Create a list or table for the objects (ng-repeat)
- Set up an input field to search on the items in the list or table (ng-model and filter on ng-repeat)
- For the brave:
  - A button that removes an item from the list/table
  - A form field for adding new items to the data

# Node.js – What is it?



- ◉ Platform built on Chrome's JavaScript runtime for fast, scalable network apps
  - ◉ Created by Ryan Dahl
  - ◉ Has a event-driven, non-blocking I/O model
- ◉ Allows you to run JavaScript outside of the browser (like your terminal)
- ◉ Comes with a JavaScript API to access the network and file system
- ◉ <http://nodejs.org/>

# Why use Node.js?



- Can modularize and share across the stack
- Heaps of utility, doesn't have to be a server
  - Unit testing
  - Automated build tasks
  - Command line tools
- Potential for JavaScript across the stack
  - Server (Node)
  - Services (Node)
  - Database (Mongo)
  - Data format (JSON)
  - Client-side (Angular, Backbone, Express, etc)
  - Test runner and framework
  - Entire build and deploy process while we're at it
- Can run many tasks at once, but only one operation at a time

# When to use Node



- Great for lightweight, high-traffic, low-cpu things
  - JSON APIs
  - Single page apps
  - Command line utilities
  - Streaming data (file uploads)
  - Two-way interaction between client and server like Chat rooms
  - Input queues, deferred actions
- Don't use it for
  - CPU intensive operations; it will block the thread
  - Long running, blocking operations

# Compared to traditional servers



- ◉ Traditional
  - ◉ Thread-based
  - ◉ Each request will spawn a thread, consuming memory
  - ◉ Each thread will wait for request to do its thing and blocks on several operations (i/o)
  - ◉ Need many threads to handle traffic
- ◉ Node.js
  - ◉ Event-driven
  - ◉ Runs on a single thread which can initiate many operations in parallel
    - ◉ but can only complete one (callback) at a time from the callback queue
  - ◉ Tasks are begun quickly, and only reported on when complete (and put in the processing queue)
  - ◉ Events and callbacks are key to this operation

# Exercise: Hello world



- Install node
  - <http://nodejs.org/download/>
  - Or use homebrew: brew install node
- Create a simple JavaScript file
  - console.log("Hello World");
- Tell node to run it
  - \$> node hello-world.js
- Much Wow.

# A basic server



- New file; server.js

- ```
var http = require("http");
http.createServer(function(request, response) {
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Hello World");
  response.end();
}).listen(8888);
```

- <http://angular-test1.course/node/server-basic.js>

- Run it: node server.js

- Head to <http://localhost:8888/>

Where might you use node



- A simple server
- A robust API, serving up JSON
- A command line script runner
- Web services, like queue managers
- A full-on web framework
 - <http://mean.io/#/>
 - MongoDB
 - Express
 - AngularJS
 - NodeJS

Node Package Manager



- Comes packaged with NodeJS installation
 - `npm install npm -g`
- <https://npmjs.org/>
- Some popular ones
 - Express – web dev framework
 - Socket.io – for websockets
 - Mongo – wrapper for mongo db api

package.json

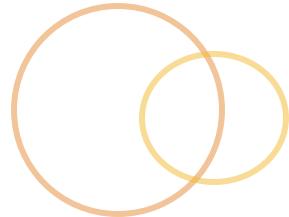
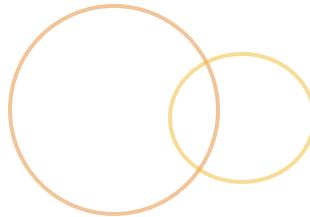


- ◉ Npm packages information and dependencies defined in package.json

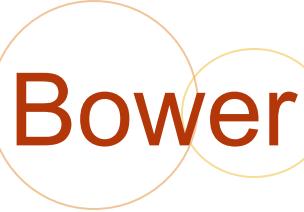
```
◉ {  
    "name": "node-api",  
    "description": "test api",  
    "version": "0.0.1",  
    "private": true,  
    "main": "server.js",  
    "dependencies": {  
        "express": "~4.0.0",  
        "body-parser": "1.0.2",  
        "jsonp-express": "0.0.1"  
    }  
}
```

- ◉ Npm install
- ◉ Can also list build processes, test runners, etc

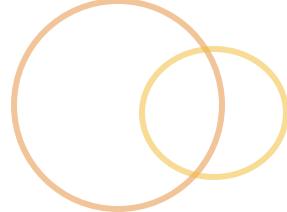
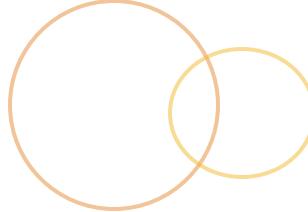
Express



- Framework for NodeJS
 - <http://expressjs.com/>
- Excellent for fast API development
- Essentially a series of middleware calls
 - Sits between request and response
- My example express server
 - ~/sites/ex-node-restful/server.js
 - And
 - ~/sites/ex-node-restful/proxy-server.js

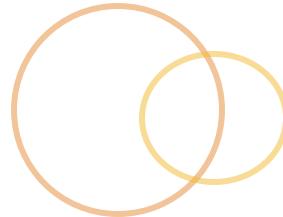
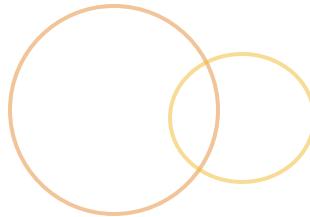


Bower



- Package manager for front-end web stuff
 - <http://bower.io/>
 - Installed via npm
- Can easily drop in libs like
 - Jquery
 - Bootstrap
 - Angular
 - Backbone
 - <http://bower.io/search/> etc...

GruntJS



- JavaScript task runner, built in Node
 - <http://gruntjs.com/>
 - Installed via npm
- Automate tasks like
 - Builds
 - Minification
 - SASS/LESS
 - Testing
 - Linting
- You define tasks that must be run, it goes and does them for you, reporting on success/fails

Angular + Express + Node

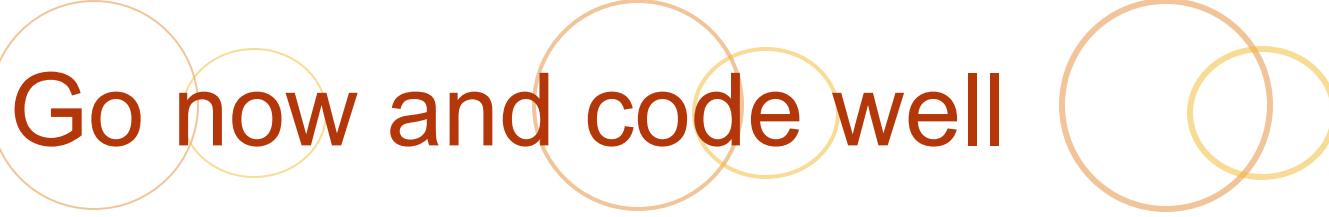


- ◉ Can stack
 - ◉ Angular as the client-facing app
 - ◉ Express as middleware/routing
 - ◉ Node as the backend serving up the content
- ◉ Angular + Express seed:
 - ◉ <https://github.com/btford/angular-express-seed>

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