learning spike

Modular JavaScript

Ryan Morris

@mrmorris

Introductions

- About me…
- About you…
  - Name?
  - What do you do here?
  - What is your programming background?
    - Any front-end?
    - 😍, 😡 or 😭 JavaScript?
  - What do you hope to gain from this course?

How the class works

- **Lecture & labs**
- **Informal**
- **Lots of research/docs**
- ***I am your guide***

What we'll cover

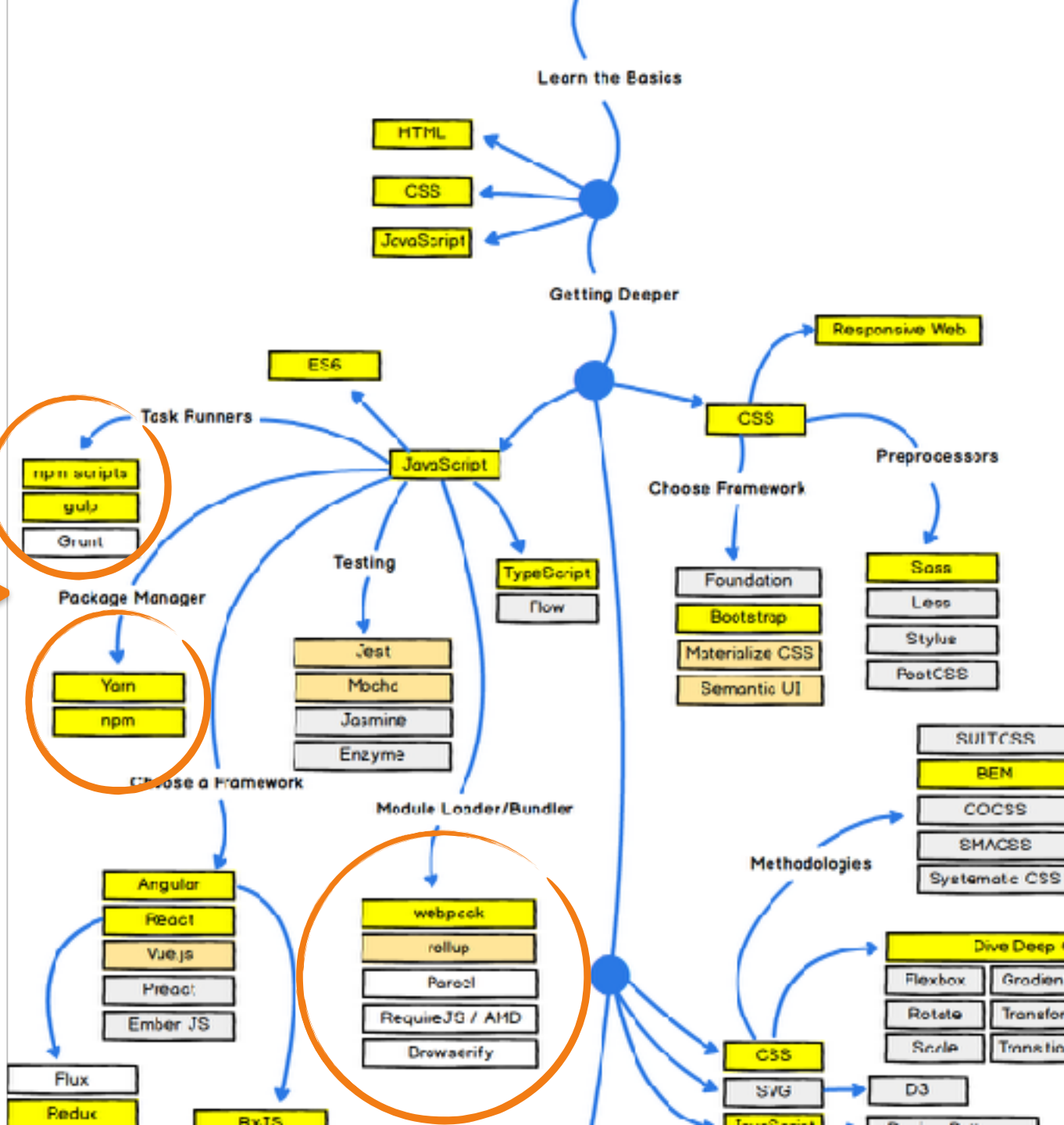
- A little review (js, html, css, dom)
- ES6 ~ ESNext
- JS Tooling
  - npm, grunt and beyond
- Module Design
- Module Patterns
  - AMD, UMD, CommonJS, ES6

I wasn't planning to cover

- XHR
- OO
- App Design

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

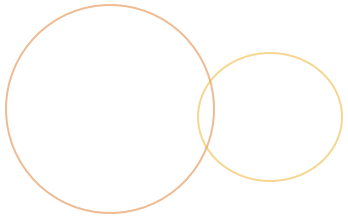
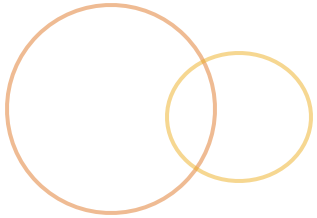
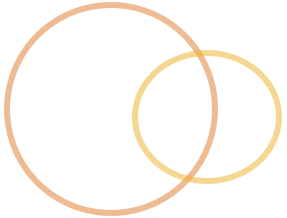
Here we go!


Learn the Basics
HTML
CSS
JavaScript
Getting Deeper
ES6
Task Runners
npm scripts
gulp
Grunt
JavaScript
Package Manager
Yarn
npm
Testing
Jest
Mocha
Jasmine
Enzyme
TypeScript
Flow
Choose a Framework
Angular
React
Vue.js
Preact
Ember JS
Flux
Redux
Module Loader/Bundler
webpack
rollup
Parcel
RequireJS / AMD
Browserify
CSS
Responsive Web
Choose Framework
Foundation
Bootstrap
Materialize CSS
Semantic UI
Preprocessors
Sass
Less
Stylus
PostCSS
Methodologies
SUITCSS
BEM
COCSS
SMACSS
Systematic CSS
CSS
SVG
D3

Modular JS — Set up

- You need:
  - A browser with dev tools
    - Open your browser and hit `F12` or `alt/opt/⌥ -⌘-i`
  - Node & NPM
    - https://nodejs.org/en/
  - Download the repository
    - https://github.com/rm-training/modular-js

👆 Everyone OK with the above?


refresher…?

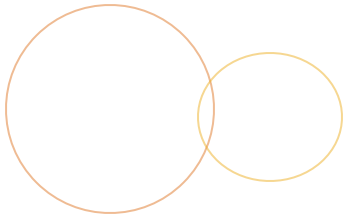
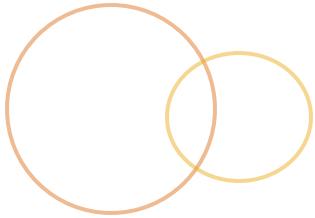
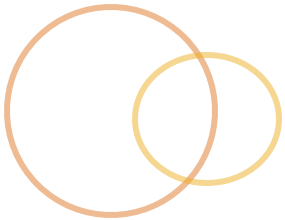
DEBUGGING

Browser Debugging

- Use browser dev tools to access its JavaScript console
  - The browser's "`console`" is a REPL
  - log output for testing
- Can also use dev tools to:
  - set breakpoints & debug js
  - view network requests
  - view memory usage
  - inspect html + css

The `console` object

- Console api
  - `console.log(); // echo/print/output`
  - `console.assert(); // test`
  - `debugger; // breakpoint`
- Gotchas
  - Console methods are asynchronous
    - They may not run in the order you expect
  - They are not available in every browser
- Seeing a bug/issue?
  - Clear your console of old errors
  - Check where the error happened


refresher

JAVASCRIPT

What is JavaScript?

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

Basic constructs

- **We should be OK with:**
  - variables
  - data structures like arrays or maps
  - if-else statements
  - for and while loops
  - functions

Core concepts

- **We should be OK on:**
  - Data Types
    - Objects, Functions, Arrays
  - Coercion
  - Scope
  - Hoisting
  - Context (`this`)
  - Closures & IIFEs

If we're not OK on a topic here we *should* dive into it

Refresher - Data Types

- There are 5 primitives (string, number, boolean, null, undefined) and then Objects
  - Functions are a callable Object
  - Objects are maps of properties referencing data
  - Arrays are for sequential data
- Declare variables with "`var`"
  - **Function scope**
  - Block scope in ES6 with "`let`" and "`const`"
- 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

Refresher - Type Coercion

- If a variable type is not what JavaScript expects, it will convert it on the fly, based upon the context

```
var x = “ryan”; // a literal
“ryan”.length; // is coerced to a… ?


+"42"; // 42
"Name: " + 42; // "Name: 42"
1 + "3"; // 4;
“1” + 3; // 13;
```

- Truthy vs Falsy is coercion in action

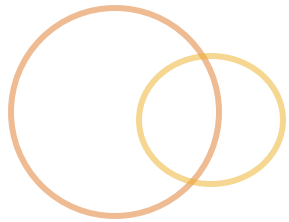
```
null; // false
“false”; // true
[]; // true
```

Refresher - What scope?

- What are the scopes here?

```
var a = 5;
function foo(b) {
  var c = 10;
  d = 15; // where is d?

  function bar(e) {
    var c = 2; // which c?
    a = 12; // which a?
  }
}
```

What scope, pt 2?



- What are the scopes here?

```
var a = 5;
function foo(b) {
  var c = 10;
  d = 15; // where is d?

  if (d < 5) {
    var c = 2; // which c?
  }
}
```

Exercise: Hoisting (pt 1 of 3)

- What will the output be?

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

 console.log(x); // what will the output be?
 return x;
}

foo();
```

Exercise: Hoisting (pt 1 of 3)

This…

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

 console.log(x);
 return x;
}
foo();
```

Becomes…

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

 console.log(x); // 42
 return x;
}
foo();
```

Exercise: Hoisting (pt 2 of 3)

- And this?

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

Exercise: Hoisting (pt 2 of 3)

This…

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

Becomes…

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

Exercise: Hoisting (pt 3 of 3)

- And finally

```
foo(); // ?
bar(); // ?


function foo() {
 console.log("Foo!");
}


var bar = function(){
 console.log("Bar!");
}
```

Exercise: Hoisting (pt 3 of 3)

This…

```
foo();
bar();


function foo() {
 console.log("Foo!");
}


var bar = function(){
 console.log("Bar!");
}
```

Becomes…

```
var bar;
function foo() {
 console.log("Foo!");
}

foo(); // Foo!
bar(); // TypeError


bar = function(){
 console.log("Bar!");
}
```

Exercise: Callbacks & Async

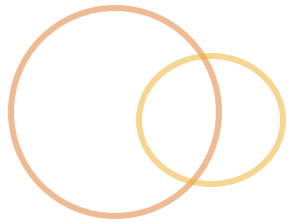
- What does this code do?

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

// what does this log out?
```

Solution: Callbacks & Async

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

Exercise: Objects



- What is going on here?

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

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

Exercise: Functions and Context

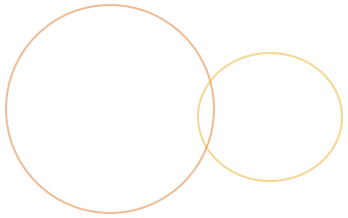
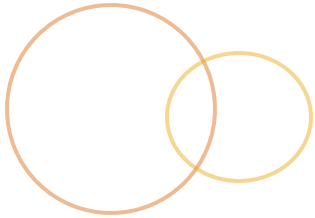
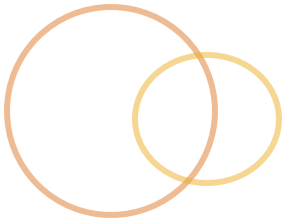
- What is going on here?

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

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

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

Core JS concepts

- **All good?**
  - Data Types - *primitives and objects*
  - Coercion - *embrace it*
  - Scope - *function scope, it is lexical*
  - Hoisting - *it happens*
  - Object - *objects are everywhere*
  - Function declaration vs expression
  - Context - *it is dynamic*

If we're not OK on a topic here we *should* dive into it

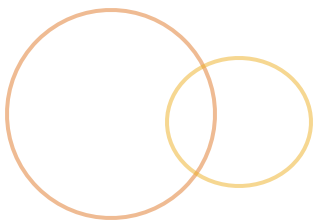
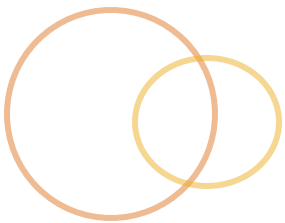

refresher

HTML

Wizard check

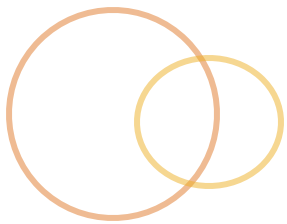
- OK with basic HTML?
- Can write a page in full?
- Write a **<form>** and all necessary input controls?
- Understand the difference between ** ** and ****?
- Understand the usage of **attributes** on elements
- When to use **id** versus **class**?

HTML


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

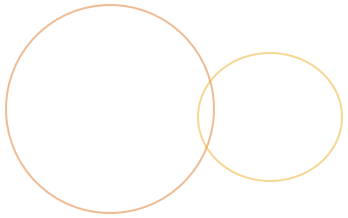
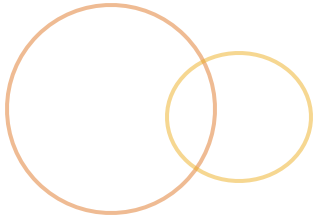
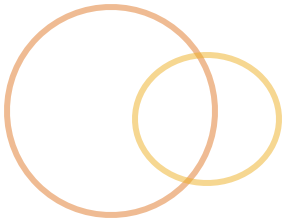
Anatomy of a page



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

Anatomy of an element

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


refresher

CSS

Wizard check

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

Cascading Style Sheets

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

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


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


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

Anatomy of a css declaration

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

CSS Selectors

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

CSS Specificity

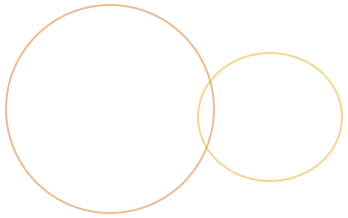
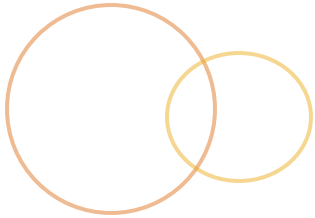
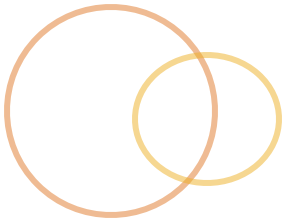
- Selectors apply styles based on its **specificity**
  - inline, id, pseudo-classes, attributes, class, type, universal
- **`!important`** allows you to override

html:

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

css:

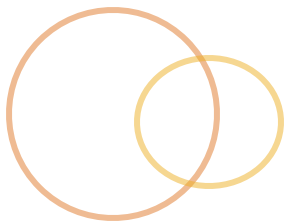
```
#main{
   color: orange;
}
.fancy{
   color: blue;
}
#main.fancy{
   color: red;
}
```


refresher

THE DOM

The DOM Refresher

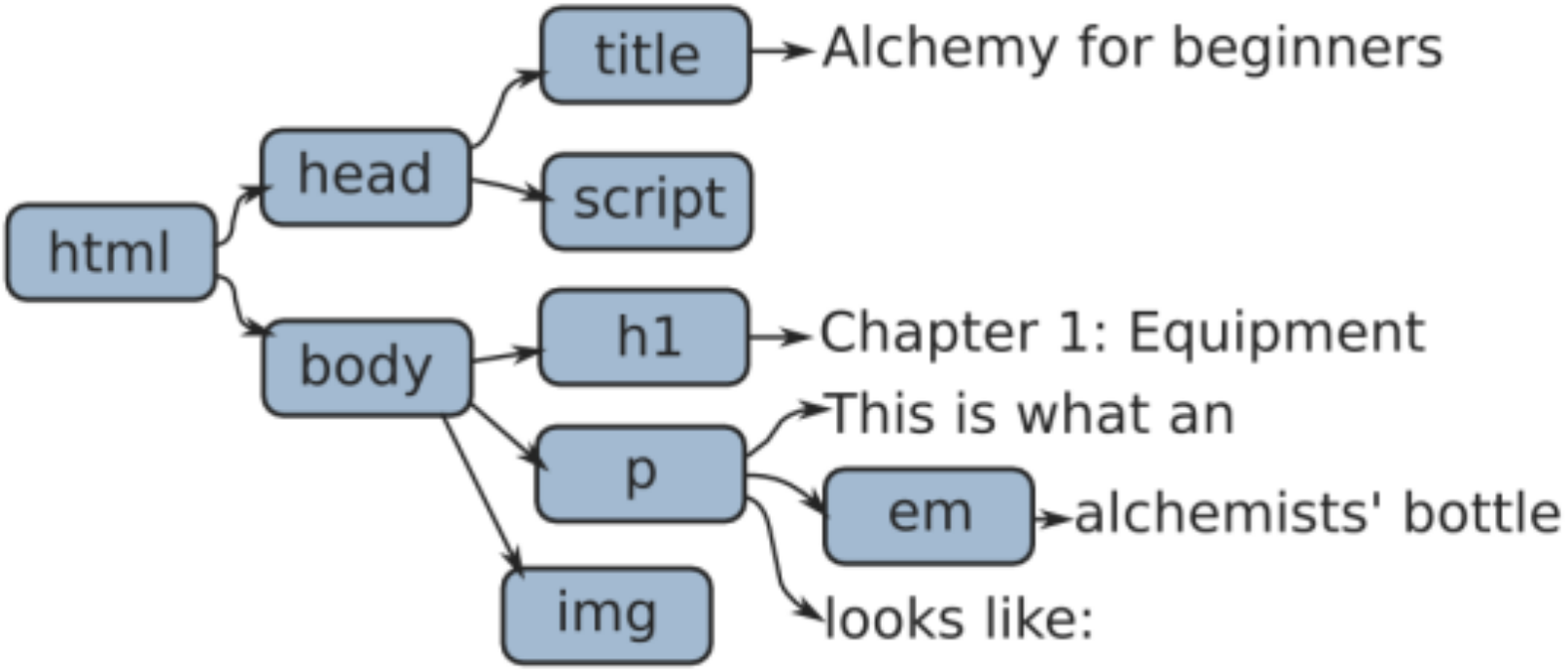


- How does everyone feel about using native DOM methods
  - `document.getElementById()`
  - `#querySelector()`
  - `#querySelectorAll()`
  - `#addEventListener()`

DOM Structure

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

Document Structure



Accessing elements

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

```
document.getElementById(“main");
// returns first element with given id
// <div id=“main”>Hi</div>
```

```
.querySelector("p span");
// returns first matching css selector
// <p><span>Me!</span><span>Not!</span></p>
```

```
.querySelectorAll("p span");
// all elements that match the css selector
// <p><span>Me!</span><span>Me!</span></p>
```

Element Traversal

- Avoid's text-node issues
- Supported in ie9+
- From an element node
  - `.children`
  - `.firstElementChild, .lastElementChild`
  - `.childElementCount`
  - `.previousElementSibling`
  - `.nextElementSibling`

Creating new nodes

- **`document.createElement`**`("div")`
  - creates and returns a new node without inserting it into the DOM
- **`document.createTextNode`**`("foo bar")`
  - creates and returns a new text node with given content
- Or edit the element content directly
  - `elementVar.`**`innerHTML`**` = '<span>hi</span>';`
  - `elementVar.`**`innerText`**` = 'hi';`

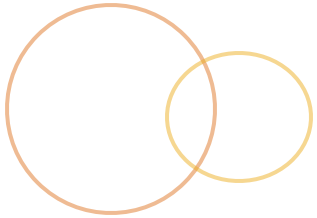
Adding nodes to the tree

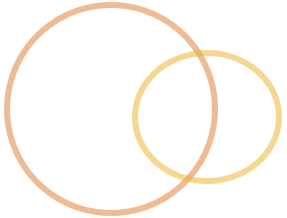
```
// given this set up
var parent = document.getElementById("users"),
    existingChild = parent.firstElementChild,
    newChild = document.createElement("li");

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

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

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

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

Element Attributes

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

Events





- Single-threaded, asynchronous event model
- Events fire and trigger registered handler functions
- Events can be click, page ready, focus, submit, etc

Event Handling

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

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

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

// not onClick properties
```

Event handler context

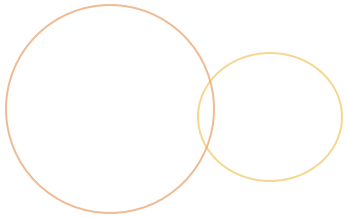
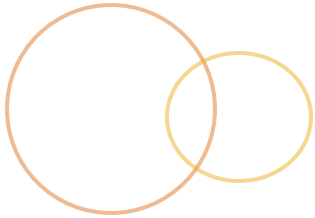
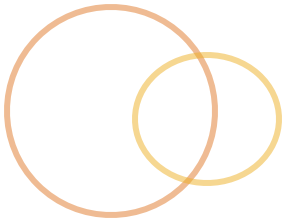
- Functions are called in the context of the DOM element

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

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

Event Propagation

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

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

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

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

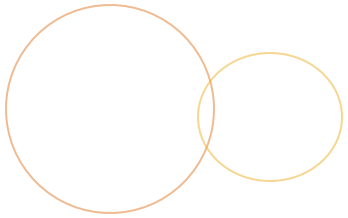
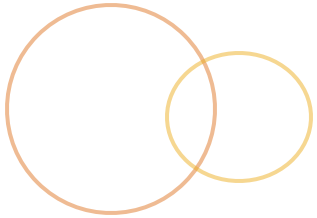
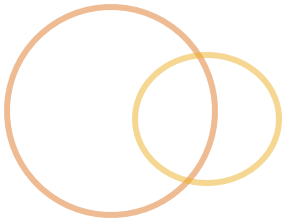

refresher

WARM UP

Warm Up

- **[dom + js] Input History**
  - http://jsfiddle.net/mrmorris/t2wazjmg/
  - *Or* locally in the `/warmup` folder in the repository
    - Run the server with: `npm start`

Solutions:
Input History: http://jsfiddle.net/mrmorris/0hvt7d9e/


modern javascript

INTRO TO ES6+

JavaScript Versions

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

`ES{n} <=> EcmaScript 20{n-1}`

How JS is progressing

- **Yearly** releases
- **Syntax** & **feature** improvements
- **Organizing** modules within the core language
  - `Number.isInteger()` **vs** `isFinite()`
- Entirely **backwards compatible**
  - There are exceptions when using **strict mode**
- *“Paving the cowpaths”*

How *you* are progressing

- **Target** by browser(s) or environment
- Target a **version** be using
  - ES3 (ouch)
  - ES5
  - ES6+
- Use **modernizr** and/or feature checking
- Strict mode to **opt in to modern** standards
- **Transpile** ES6+ code to ES5

Strict Mode

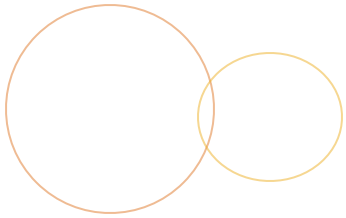
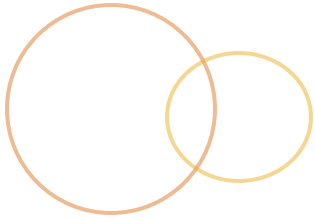
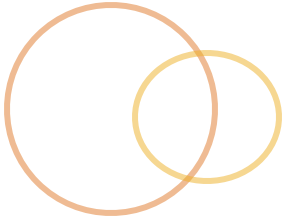
- **Opt in** to modern standards
- Silent issues become **error messages**
- Prevents accidental access to global (through `this`)
- Most modern JS will be *strict*

```
“use strict”;

function () {
  “use strict”;
}
```

ES6 Features Hit List

- Block scope
- Template literals
- Function argument defaults
- New array methods
- Arrow functions
- Object shorthand
- Rest and Spread
- Destructuring
- The class syntax
- Built-in module support

There is more!
Set & Map
Symbols
Iterators
Generators
Observables
Promises


es6

THE HITS

Block Scope

- JS is traditionally **function scoped**
- `let` and `const` bind within a **block's scope**

```
let x = 5;

if (x == 5) {
  let x = 10;
  let y = 12;
}

console.log(x); // 5
console.log(y); // Undefined
```

Block Scope

- Doesn't hoist
- Can't be redeclared
- `let` is mutable
- `const` is immutable

```
const me = {name: 'Ryan'};
const x = 10;
let y = 12;

x = 50; // TypeError
me.name = 'Jim'; // ok
me = {}; // TypeError
```

Template Literals

- Define strings that have inline variable replacement
- Can use any expression

```
let name = 'Ryan';
console.log(`${name} is ${5 + 2} years old`);
// Ryan is 7 years old
```

- Multiline with \

```
var myString = ‘bla’
  + ‘bla’;
  + ‘bla’;
```

```
let myString = `bla\
bla\
bla`;
```

Function Argument Defaults

- Can define default values for function arguments

```
function callMe(x, y, z) {
  x = x || 1;
  y = y || true;
  z = z || 'bla';
}
```

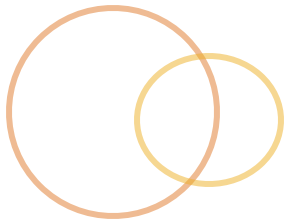
```
function callMe(x = 1, y = true, z = 'bla') {
 // …
}
```

Function Argument Defaults

- What happens when we pass in `undefined`?

```
function callMe(x = 1, y = true, z = 'bla') {
 console.log(x);
}

callMe(undefined); // ?
```

Function Defaults



- You can use expressions

```
function bla (val = 2 + 2) {}
function bla (val = someVal()) {}
```

- And even reference other variables in the signature (but the variable must be declared first)

```
// val can be used after it is declared
function bla (val, other = val + 2) {}
```

New array methods

- .forEach()
- .map()
  - Return a new array with same # of elements but new values
- .filter()
  - Return a new array with subset of original values
- .reduce()
  - Return a single value based on all values of an array
- .find()
  - First matching element that passes a test function
- .every()
  - True if every element passes a test function
- .some()
  - True if at least one element passes at test function

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

New object methods

- Object.assign()
- Object.keys()
- Object.entries (ES8)
  - array of [prop, val] array pairs
- Object.values (ES8)
  - Return array matching key order

Arrow Functions

- The “fat arrow”

```
() => 'fat';
```

- Super short syntax
  - () is optional if there is one argument
  - {…} is optional when it’s a one-liner
  - Implicit return

```
const adder = x => x+1;
```

```
const combiner = (x, y) => {
  return x + y;
};
```

Arrow Functions

- But it's mostly about ***lexical context***
  - **`this`** is bound to the outer function's context
  - you can't bind() an arrow function
  - no arguments of it's own
  - no .prototype

```
const test = () => {
  console.log(this); // Window
}

me.test = test;
me.test(); // still Window
```

Arrow Functions

```
// takes previously annoying stuff like:
document.addEventListener('click', function(){
  console.log(this); // #document
});

// and makes it like:
document.addEventListener('click', () => {
  console.log(this); // Window
});
```

Arrow Functions - Gotchas

```
const person = {
  trophies: ['blue', 'gold'],
  name: 'Ryan'
}

person.listTrophies = function() {
  this.trophies.forEach(function(el) {
    console.log(this.name, 'got', el);
  });
}

person.listTrophies();
// Ryan got blue
// Ryan got gold
```

Where can we swap in arrow functions?

Arrow Functions - Gotchas

```
const person = {
  trophies: ['blue', 'gold'],
  name: 'Ryan'
}

person.listTrophies = function() {
  this.trophies.forEach((el) => {
    console.log(this.name, 'got', el);
  });
}

person.listTrophies();
// Ryan got blue
// Ryan got gold
```

Where can we swap in arrow functions?

Just this one

Arrow Functions context

- Look at the outer function to determine an arrow function's context…

```
const arrowMaker = function() {
  return () => console.log(this);
}
// "arrowMaker's this was window"
arrowMaker()(); // Window

const me = {};
me.arrowMaker = arrowMaker;
// "arrowMaker's this was me"
me.arrowMaker()(); // me

// "arrowMaker's this was window"
me.arrow = arrowMaker();
me.arrow(); // Window
```

Object Shorthand

- Define object properties in shorthand notation

```
var color = 'red';
var me = {
      x: 5,
      color: color
}
```

```
var color = 'red';
var me = {
      x: 5,
      color
}
```

Object Shorthand w/ Functions

- There is also a shorthand function declarations

```
var me = {
  x: 5,
  toString: function () {}
}
```

```
var me = {
  x: 5,
  toString() {}
}
```

Object Dynamic Projects

- We can also write *dynamic* properties inline

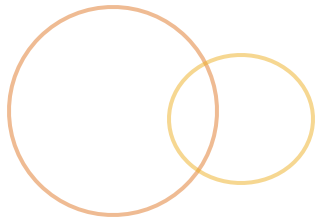
```
var me = {
      x: 5,
      ['prop_' + 42]: 42,
      "my func"() {}
}

me.prop_42; // 42
me["my func"](); // ok
```

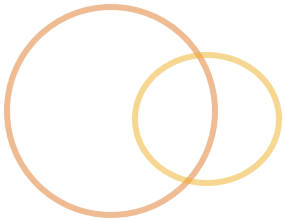
Rest

- A collector for remaining values

```
function addAll(a, b, c, …others) {
   console.log(others); // [4,5,6]
}

addAll(1,2,3,4,5,6);
```

Spread

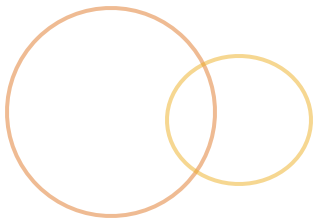
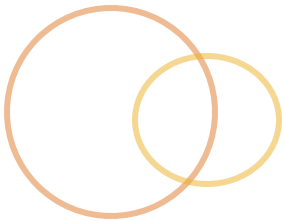




- Expands an *Iterable* in place
  - Iterables include strings, arrays - but not objects

```
// a string
[…"astring"]
// [a,s,t]

// an array
const myArray = [3,4];
const mergedArray = [1,2, …myArray, 5];
// [1,2,3,4,5]
```

Spread


- Spread for a shallow copy of an array

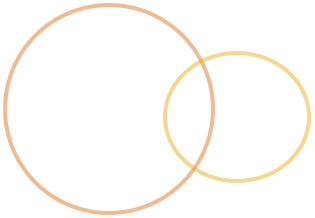
```
var array = ['red', 'blue', 'green']
var copyOfArray = [...array]
```

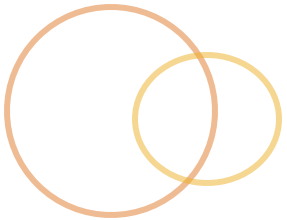
Spread

- Spread to merge

```
const defaultColors = ['red', 'blue', ‘green'];
const userDefinedColors = ['yellow', ‘orange'];

const mergedColors = [
    …defaultColors,
    …userDefinedColors
];
```

Spread





- Spread within a function call

```
function sum(x, y, z) {
  return x + y + z;
}

const values = [1,2,3];

sum(…values); // 6
```

Spread Objects [ES8]

- You can spread objects into other objects
  - copies own enumerable
  - Just not into arrays, unless it is Iterable

```
// shallow clone
const objClone = { ...obj };

// extend objects
const newObj = {…obj, z: 3};
```

Destructuring

- Extraction of elements from an iterable

```
let [a, b] = [1,2,3];
a; // 1
b; // 2

var [,,c] = [1,2,3];
c; // 3
```

Array Destructuring

- You can destructure with spread

```
let [a, …b] = [1,2,3];
```

- And set defaults

```
let [a, b, c=5] = [1,2,3];
```

- And handle nested arrays

```
let [a, b, [c, d]] = [1,2,[3,4]];
```

Object Destructuring

- Caveat is that property names should match

```
var {a, b} = {a: 1, b: 3};
a; // 1
b; // 3

var {d, e} = {a: 1, b: 3};
d; // d :(
e; // e :(
```

- But you can specify an **alias**

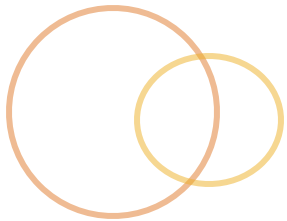
```
var {a:c, b} = {a: 1, b: 3};
c; // 1
b; // 3
```

Destructuring in function arguments

- You will also see destructuring in function args

```
function rad([a,b,c]) {
  console.log(a);
  console.log(b);
  console.log(c);
}

rad([1,2,3]);

rad(1); // error – not an iterable
```

Class syntax



- Syntax sugar on top of prototype

```
function Cat (breed) {
 this.breed = breed;
}

var calico = new Cat('calico');
```

```
class Cat {
  constructor(breed) {
      this.breed = breed;
  }
}

var calico = new Cat('calico');
```

Class is still prototypal

```
function Cat (breed) {
 this.breed = breed;
}

Cat.prototype.speak = function() {
  console.log('Meow');
}

var calico = new Cat('cal
calico.speak(); // Meow
```

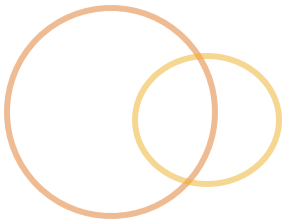
```
class Cat {
  constructor(breed) {
    this.breed = breed;
  }
  speak() {
    console.log('Meow');
  }
}

var calico = new Cat('calico');
calico.speak();
```

Defining methods

- Only methods can be defined on the class

```
class Cat {
  constructor(breed) {
    this.breed = breed;
  }
  speak() {
    console.log('Meow');
  }
}

Cat.prototype.legs = 4;

var calico = new Cat('calico');
```

Extending Classes



```
class RobotCat extends Cat {
  constructor() {
    super('unknown');
  }
  simulateSpeak() {
    console.log('Start simulation:');
    super.speak();
  }
}

var robot = new RobotCat();
```

Getters (and setters)

```
class RobotCat extends Cat {
 get tail() {
    return '~~~~~';
 }
}

var robot = new RobotCat();
robot.tail; // ~~~~~
```

Static methods

```
class Cat {
 static catCall() {
   console.log('COMMENCE MEOW!');
 }
}

var kitty = new Cat();
kitty.catCall(); // TypeError
Cat.catCall(); // COMMENCE MEOW!
```

Class caveats

- **Can't redeclare** a class in the same block scope
- They are **not hoisted**
- Class declaration is **scoped to the block**
- You can also use an expression

```
// class expression
const Dog = class {}

// class declaration
class Dog {}
```

Async and Await

- Write code that **looks synchronous** but **behaves asynchronously**
- Easier to read, write and maintain
- Depends on Promises (which are now native)

```
async function loadUser(id) {
  const user = await fetchUserData(id);
  const account = await fetchAccount(user);
  const items = await fetchItems(user);
  return Object.assign({},user,account,items);
}
```

ES6 Exercises

Using ES6 syntax and features, convert and/or solve the exercises in the `/es6` folder of the repo

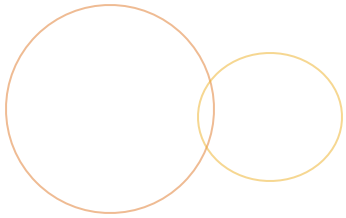
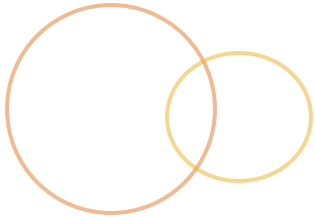
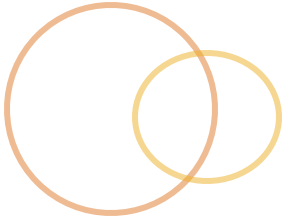
```
$ ls
arrays.js          classes-1.js     functions.js     spread.js
block-scope-1.js   classes-2.js     rest.js
block-scope-2.js   fat-arrows.js    solutions
ryan at Ryans-MacBook-Pro in ~/repos/training/modular-js/es6 on master
$
```

Run (and test) your code with node

```
$ node arrays.js
6937
ryan at Ryans-MacBook-Pro in
```

When in doubt, log to the console

```
console.log('help!');
```


modern javascript

USING ES6+ TODAY

But can you run it?

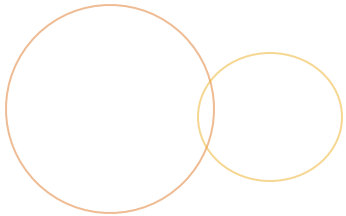
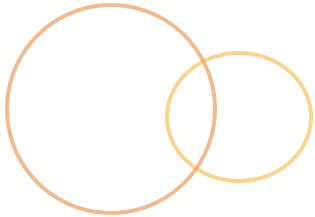
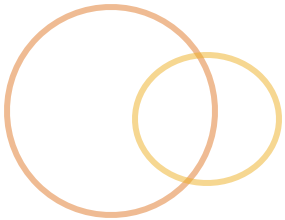
- Browser support is incremental and growing
  - Check caniuse.com
- Most likely you'll want to **transpile**
  - Convert ES6+ into ES5 or below
- There are a few tools that do this:
  - Babel
    - https://babeljs.io/
  - Typescript?
  - Coffeescript?

Intro to Babel

- **Babel** is a JavaScript compiler
- Write in future versions of ES and have it compile down to another version (ES3 or 5)
- Let's check it out:
  - https://babeljs.io/

How can we start using Babel?

- Install it with NPM
  - NPM?…


managing packages

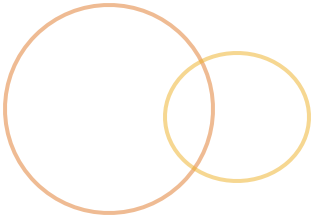
NPM

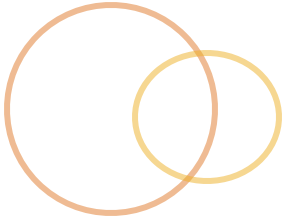
Neigh Purr Moo

- Node ~~Package Manager~~ Packaged Modules
  - https://www.npmjs.com/
- Command line tool to install, manage and publish modules to a registry
- Also a basic task runner
- Bundled with NodeJS

```
$> npm --version
5.6.0
```

NPM init

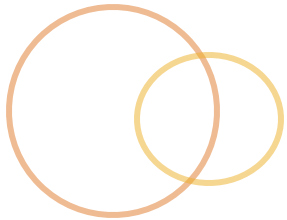




- Initialize a new project at a root folder
  - Sets up a **package.json** file to track meta and dependencies

```
npm init
```

npm install



- Used to install all dependencies

```
npm install
```

- And to install new dependencies

```
npm install jquery
```

```
npm install jquery --save
```

```
npm install jquery --save-dev
```

http-server

- A node package that runs a basic server locally
  - We can this for our project(s) moving forward

```
npm install http-server

# then
npx http-server [path] [options]
```

https://www.npmjs.com/package/http-server

Running package binaries

- **npx** can be used to execute installed binaries

```
npx install http-server
npx http-server
```

- *Or* install it globally (not always ideal)

```
npm install http-server -g
```

- *Or* you can seek out the binary in the node_module/ folder

```
./node_modules/http-server/bin/http-server
```

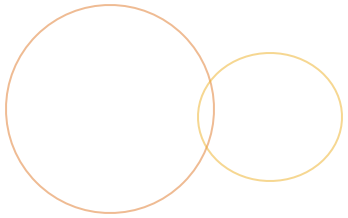
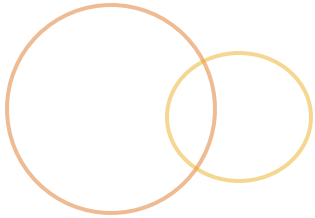
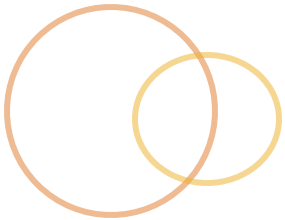
npm install

- Installs in node_modules/
- npm install -g
- npm uninstall
- npm update
- npm install jquery @3
  - @latest
  - @2.5.0

Lab - Initialize a project

We're going to initialize a project using npm and install a package (or two)

- Use `npm init` to initialize your project
  - Start in a new folder
- Use `npm` to install `jquery` and `http-server`
  - You should not install these globally
  - Make sure to save them as dev dependencies
- Check `package.json`
- Uninstall `jquery`
- Use `npx` to run `http-server`
- All done?
  - Add a `public/` folder with an `index.html` file and run the `http-server` with `public/` as the server root


tooling

BABEL

So, Babel

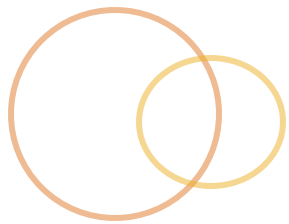
- Our compiler/transpiler
- We'll install with `npm`

```
npm install --save-dev babel-cli
npm install --save-dev babel-preset-env
```

- Then configure it in the `.babelrc` file at the root of our project

```
{
  "presets": ["env"]
}
```

Babel configuration



- Babel will check `.babelrc`
- Specifies which plugins we want to enable
  - ex: specify what module system we'll use
- And which environments we're targeting
  - ex: node or target browsers

```
{
  "presets": [
    [“env", {
      "targets": {
        “browsers": [
          "last 2 versions", "safari >= 7”
        ]
      }
```

Babel env preset

- Out of the box Babel won't really do anything
  - Either specify all the plugins you want so that transformations are applied
  - *Or* use a preset
- The '`env`' preset includes everything
- Other presets include
  - react
  - flow

Configuring babel modules

- Babel expects CommonJS modules to be used
  - We aren't using any modules (yet) so we have to ask Babel not to wrap it as a module
  - This also means that ***file order matters*** :/

```
{
  "presets": [
    [“env", {
      “modules”: false
    }]
  ]
}
```

Running Babel

- We can run babel against files or directories
  - It outputs to stdout by default

```
babel file.js
bable file.js --out-file output.js
```

- You can compile directories

```
babel src --out-dir lib
babel src --out-file concat.js
```

- And ask it to watch

```
bable file.js --watch --out-file output.js
```

Beyond Babel

- *Ideally* we won't have to re-run babel every time we want to transpire our project
- We *could* add it to npm

package.json

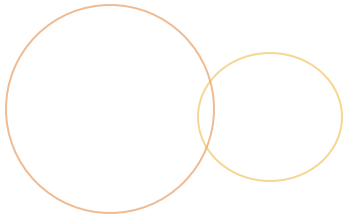
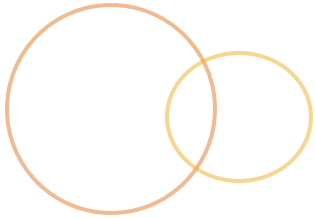
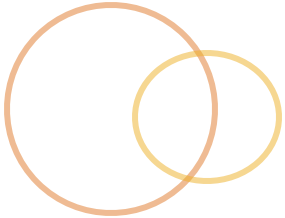
```
{
  "scripts": {
    "build": "npx babel src -d lib"
  }
}
```

- *Or* we can go into the world of task runners…

Lab - Transpile with Babel

Install Babel using NPM in your ES6 exercise folder

- Use `npm install` to install `babel-cli`
  - You'll also need `babel-preset-env`
  - Save these as dev dependencies
- Run one of your exercises through Babel and output in the command line
  - Do you see any changes?
  - You'll need to set up your `.babelrc` file
- Transpile your `es6` folder and output it to a new folder, '`build/`'
  - `npx babel <inDir> -d <outDir>`
  - Inspect the output


stepping back

FRONT-END TOOLING

Tooling

- Front-end development has evolved quite a bit
  - More responsibility
  - Server-side no longer manages everything
- Tooling enables us to manage our workflow

Lots of repetitive tasks to do

- **Analyze** code for quality
  - *Linting*
- **Optimize** JS, css and images for production
  - *Concat*, *Minify* and *Mangle*
- **Test** our app
- **Process** templates
- **Transpile** and/or compile for broader support
- Make sure **dependencies** are handled

we need a system to automate all this

The world of JS tooling

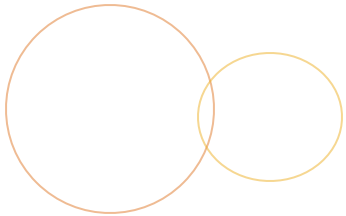
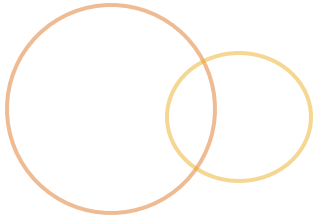
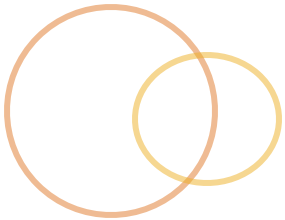
- Package Managers
  - npm
  - yarn
  - ~~bower~~
- Task runners
  - Grunt
  - Gulp
- Bundlers/Builders
  - Brunch
  - Webpack
  - Browserify
  - Rollup
  - Babel (getting there)
  - RequireJS

There's a lot of overlap throughout these tools

Dependency Management
Transpilers
Compilers
Optimizers
Some do more…

Our toolkit for today

- Package Managers
  - **npm**
- Task runners
  - **Grunt**
- Bundlers/Builds
  - Not much for today but eventually..
  - RequireJS *and/or* Webpack


task running

GRUNT

Grunt Exercise Set up

We'll be working from a pre-made app - Time 'till Weekend

Grab it from my repository:
https://github.com/rm-training/modular-js/releases/tag/timer-app-start-point

Or just from the main master branch
Start in the folder: `ttw-app-base`

Solutions
You can see the end-state by checking out this tag:
timer-app-no-modules-COMPLETE

Or grabbing this release:
https://github.com/rm-training/modular-js/releases/tag/timer-app-no-modules-COMPLETE

Intro to Grunt

- A task runner for JavaScript built in JavaScript
- Installed in two pieces via `npm`
  - `grunt-cli` (version agnostic grunt runner)
  - `grunt` (task runner)
- `Gruntfile` for configuration of tasks and flows
- *Lots* of plugins available

https://gruntjs.com/

Installing Grunt

- Install the cli

```
npm install grunt-cli -g
```

globally, if you prefer

- Then at the project level install grunt

```
npm install grunt --save-dev
```

- Finally, create a Gruntfile.js

```
touch Gruntfile.js
```

Configuration with Gruntfile.js

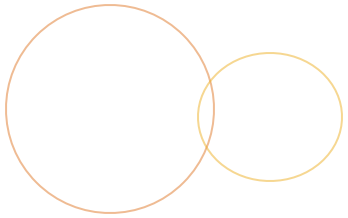
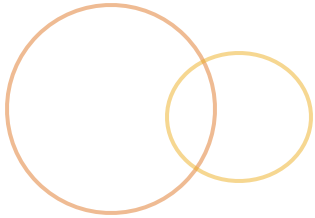
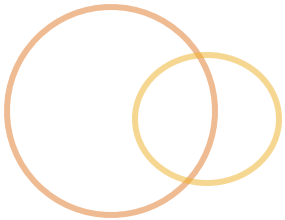
- Configure our tasks and workflows

Gruntfile.js

```
module.exports = function(grunt) {
      grunt.initConfig({
        // task configuration
      });
      grunt.registerTask('default', []);
}
```

- This is what is run when we run `grunt`

```
$ npx grunt
```


our first grunt task

LINTING

Linting

- Check code against a standard
  - Is the **quality** OK?
  - Does it meets **code style** requirements?
  - Are there **errors** or **pitfalls**?
- Four options:
  - JS Lint
  - JS Hint
  - JSCS
  - ES Lint

Pre-step: Organize our source files

- If we're operating on our files we'll first need to reorganize our scripts a bit
  - Put *inline .js* into `/public/js/*.js` files

Step 1 - Install the task

- Install via NPM

```
npm install grunt-contrib-jshint --save-dev
```

- And tell Grunt to load it

Gruntfile.js

```
module.exports = function(grunt) {

    grunt.loadNpmTasks('grunt-contrib-jshint');

}
```

Step 2 - Configure the task

Gruntfile.js

```
module.exports = function(grunt) {
    const config = {
      jshint: {
        options: {
            esversion: 6,
        },
        src: ['Gruntfile.js', 'public/js/*.js']
      }
    }

    grunt.loadNpmTasks('grunt-contrib-jshint');

    grunt.initConfig(config);
}
```

Step 3 - Run the task

command line

```
grunt jshint

# or
npx grunt jshint

# or
./node_modules/grunt/bin/grunt jshint
```

That's it!

Setting a default

- Grunt typically expects you to set up a ‘default’ task alias to be run any time you run just `grunt`

```
grunt.registerTask('default', ['jshint']);
```

Register tasks

- Use `registerTask` to set up basic **custom** tasks or to create task **aliases** and **workflows**

```
grunt.registerTask('foo', ‘Task description.',
function() {
  grunt.log.writeln(‘Running!');
});

grunt.registerTask(‘default’, [‘foo’]);
grunt.registerTask(‘build’, [‘foo’, ‘bar’]);
```

Grunt verbose

- If you're trying to figure out what Grunt is up to
  - Debug a Grunt error
  - Figure out why a task is failing to run
  - See more details about the files it is affecting

```
npx grunt -v
```

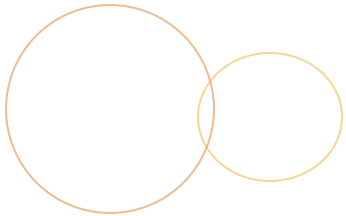
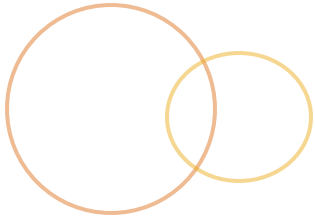
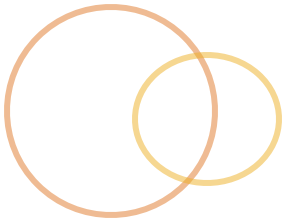
Lab - Linting with Grunt

Use jshint to de-lint the app code

- Move the javascript into standalone file(s)
- Install `grunt-cli`, `grunt` and the `jshint` plugin
- Set up the your jshint task to lint the js files
- Use Grunt to run the task. Clean up your code

```
$ npx grunt jshint
Running "jshint:src" (jshint) task
>> 1 file lint free.

Done.
```

- Then make it your default task
  - Running `npx grunt` should run jshint


grunt

OVERVIEW

Gruntfile.js

- Our config file for the project
- We'll use it for
  - Config individual tasks
  - Load tasks
  - Load or define custom tasks
  - Create workflows

Loading tasks

- Load tasks you've installed with NPM

```
grunt.loadNpmTasks('task-name');
```

- Or you can load your own task files from a folder

```
// load all tasks from within a folder
grunt.loadTasks('tasks');
```

The config object

- Usually consists of:
  - **tasks**
  - **options** for the task
  - **targets**
    - options for the target(s)
  - **file** matchers at the target *or* task level
    - src
    - dest
    - *or* a files object
    - *or* a files array
- Then we run them

```
{
  uglify: {
      options: {…},
      dev: {
        src: ['in.js'],
        dest: ['out.js']
       },
      prod: {
        options: {…},
        files: […]
      }
  }
}
```

cli

```
grunt uglify
grunt uglify:prod
```

File patterns - Compact Format

- Grunt supports several different file patterns for specifying target and output files
  - https://gruntjs.com/configuring-tasks#files

```
{
  src: 'in.js',
  dest: 'out.js'
}
```

```
{
  src: ['in1.js', 'in2.js'],
  dest: 'out.js'
}
```

```
{
  src: ['*.js'],
  dest: 'out.js'
}
```

```
{
  src: ['public/js/**/*.js'],
  dest: 'out.js'
}
```

Files Array or Object

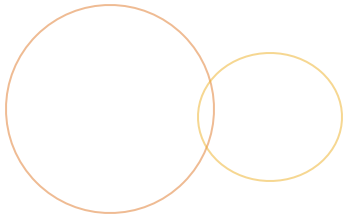
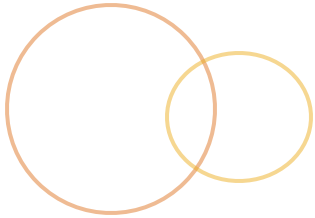
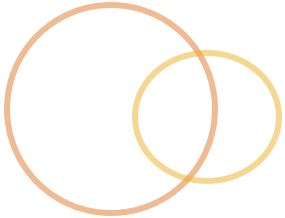
- File object format

```
files: {
      'dest.js': ['file1', 'file2'],
      'dest2.js': ['file3']
}
```

- When working on a set of files that you do not want to output as a single file

```
files: [{
      expand: true,
      cwd: 'public/js/vendor',
      src: ['**/*.js'],
      dest: 'generated/js/vendor/'
}]
```

https://gruntjs.com/configuring-tasks#building-the-files-object-dynamically


grunt

ORGANIZING OUR WORKFLOW

Organizing your workflow

- Consider each step that you'll want to take
  - lint it
  - run it through babel
  - concat
- These will be organized as tasks in Grunt
  - `grunt lint`
  - `grunt babel`
  - `grunt concat`
- And workflows (aliases)
  - `grunt build # lint, babel and concat`

Workflow Overview

- ~~Check code quality with **linting**~~
- **transpile** with babel
- **concatify** & minify the javascript
- use a **css precompiler** to convert sass/less
- possibly modify my **html** or **templates**
- And let's **watch** our files and do all the above any time a file changes

Target Planning

- Where will we output processed files?
- How many **targets** should we consider?
  - local *development* vs *staging* vs *production*
  - *dist* vs *generated*
  - *public* vs *build*

```
grunt build:generated
grunt build:dist

grunt minify:generated
grunt minify:dist
```

Our “DIST” Workflow

```
/public/
    index.html
    scripts/
        app.js
        logger.js
    styles/
        app.scss
    vendor/
        jquery.js
    images/
        icon.png
```

```
/generated/
    index.html
    scripts/
        app.js
        logger.js
    styles/
        app.css
    vendor/
        jquery.js
    images/
        icon.png
```

```
/dist/
    index.html
    scripts/
        all.min.js

    styles/
        app.css
    vendor/
        all.min.js
    images/
        icon.png
```

Where you work

Raw files
ES6+

Where you test

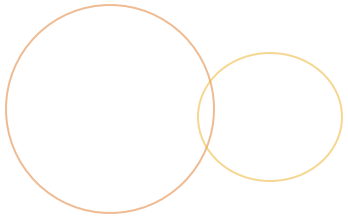
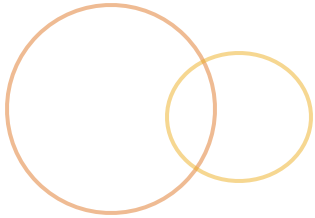
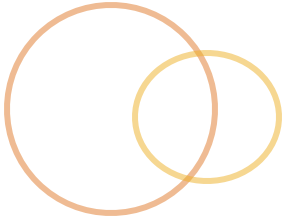
Compiled files
ES5-

What you deploy

Built files
Minified & Mangled
Not easy to debug

And later… Workflow Perks

- Start up a **server**(s)?
  - *I may use NPM for this*
- Re-run tasks **when files change**
- **Reload** my browser any time files change
- **Clean** up logs and/or build files
- Refactor our configuration
  - **auto-load** grunt plugins for us
  - reduce duplication of file lists


grunt

BABEL WITH GRUNT

Babel: our glorious transpiler

- Grunt has a babel task
- It will use our `.babelrc` config file
  - `Gruntfile.js` options will override
- Expects the following to be installed:
  - **babel-cli** or **babel-core, babel-preset-env**
- To get started we'll just need to install the grunt plugin, load it and configure it

```
npm install --save-dev grunt-babel
```

https://github.com/babel/grunt-babel

Configuring Babel in Grunt

- Set up options and our in/out files

in Gruntfile.js config object

```
babel: {
    build: {
      src: 'testfile.js',
      dest: 'testfile-out.js'
    }
};
```

This is a **multitask** and expects a target.

You can label the target whatever you like.

- Don't forget to load the task

```
grunt.loadNpmTasks('grunt-babel');
```

Work with a bunch of files

Don’t forget to:
install babel &
set up a .babelrc file

in Gruntfile.js config object

```
babel: {
    build: {
      files: [{
        expand: true, // enables most dyn. stuff
        cwd: 'js/', // must be a string!
        src: ['*.js'],
        dest: 'generated/js'
      }]
    }
}
```

Other things: source maps

- Can enable **sourcemap** output with babel

in Gruntfile.js config object

```
babel: {
    options: {
      sourceMap: true
    },
    …
```

- We should probably inline it
  - A value of "`inline`" will inline it :p

Sourcemaps

- **Maps compiled code back to the source**
  - *Very useful* when debugging across environments
- Most modern browsers understand them
- Common concerns
  - Does it mean larger downloads?
    Nope, browser will not load unless you dev tool!
  - Will it expose our work?
    Your work is already exposed!
  - Is it insecure?
    Security through obscurity is not security!

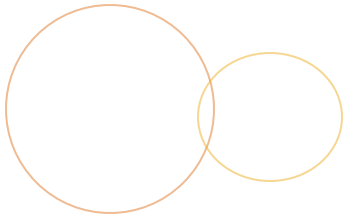
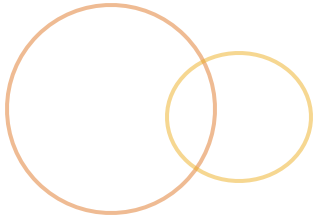
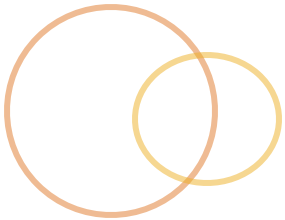
Lab - Babel with Grunt

Use babel to compile our javascript

- Install grunt-babel
  - Don’t forget babel-core, babel-preset-env and the .babelrc file
- Set up the your babel task
  - Are your files organized for this?
  - What will your output target be?
    I’ll be using ‘`generated`’
- Use Grunt to run it

```
$ npx grunt babel
Running "babel:generated" (babel) task

Done.
```

- Add it to your default task
  - Running `npx grunt` should run jshint *and* babel


grunt

GRUNT COPY

Grunt Copy

- Dealing with generated output can be tricky
- Grunt Copy helps with moving assets around

```
/public/
    index.html
    scripts/
        app.js
        logger.js
    styles/
        app.scss
    vendor/
        jquery.js
    images/
        icon.png
```

Where you work

```
/generated/
    index.html
    scripts/
        app.js
        logger.js
    styles/
        app.css
    vendor/
        jquery.js
    images/
        icon.png
```

Where you test

```
/dist/
    index.html
    scripts/
        all.min.js

    styles/
        app.css
    vendor/
        all.min.js
    images/
        icon.png
```

What you deploy

https://github.com/gruntjs/grunt-contrib-copy

Grunt Copy

Install it

```
npm install --save-dev grunt-contrib-copy
```

Configure it

in Gruntfile.js config object

```
copy: {
  build: {
     files: [{
        expand: true,
        cwd: 'public/',
        src: ['**/*', '!js/*'],
        dest: 'generated/'
     }]
  }
}
```

Don't forget to load it with loadNpmTasks()

We can ignore folders or files as well

Building Workflows

- We can tie tasks together with aliases

in Gruntfile.js

```
grunt.registerTask('build', [
  ‘lint’,
  ’babel',
  'copy'
]);
```

- Running an alias

cli

```
grunt build

# see it listed
grunt -h
```

Then update my server

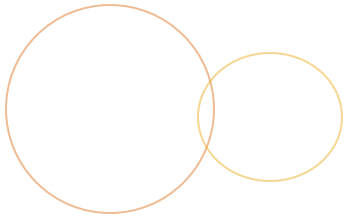
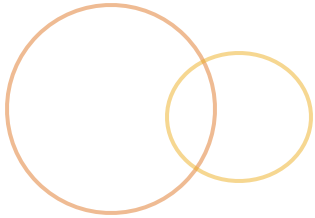
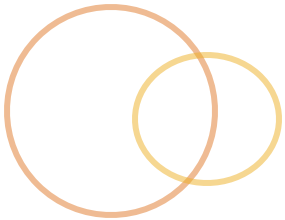
- I have a local server set up with **http-server**
  - We *could* get Grunt to do this for us
- But… for now:
  - I'll change my server's app root to `/generated`

Lab - Copy with Grunt

Use copy to help run our server

- Install grunt-contrib-copy
- Configure the copy task
  - Copy *all* files but your js that is to be compiled
  - Copy to `/generated` (or /build)
- Use Grunt to run it
- Add it to your default task
- Update npm “start” to run the server against `/generated`
- Run “`npm start`” and check the site

```
$ npx grunt
Running "jshint:src" (jshint) task
>> 3 files lint free.

Running "babel:generated" (babel) task

Running "copy:generated" (copy) task
Created 7 directories, copied 23 files

Done.
```


grunt

GRUNT CLEAN

Cleaning house

- Ignore generated output in your repo history
- We should also clean it up occasionally

```
rm -Rf generated
```

- But then again, then there's **grunt clean**

https://github.com/gruntjs/grunt-contrib-clean

Grunt Clean Installation

- Install it

```
npm install --save-dev grunt-contrib-clean
```

- Configure it

in Gruntfile.js config object

```
clean: {
  generated: ['generated']
}
```

Don't forget to load it with loadNpmTasks()

- Run it

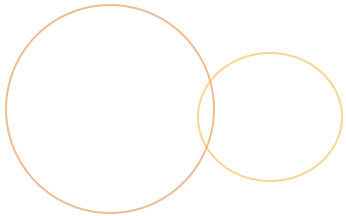
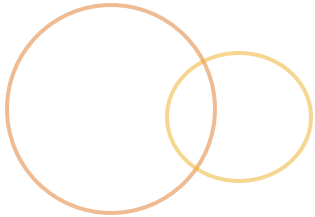
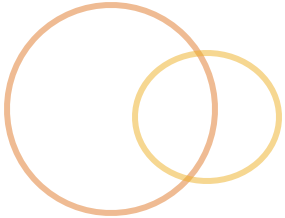
```
grunt clean:generated
```

Should *probably* clean each time we 'generate'

Lab - Clean with Grunt

Use clean to remove generated output

- Install grunt-contrib-clean
- Configure the clean task
  - It should remove `/generated`
  - Configure ‘clean’ to run each time you re-generate / generated
- Use Grunt to run it


grunt

GRUNT WATCH

Grunt Watch

- Running grunt for every edit is *tedious*
- **Grunt watch** will make grunt run indefinitely
  - Watches files and folders
  - Changes will trigger task(s)

https://github.com/gruntjs/grunt-contrib-watch

Grunt Watch Installation

- Install it

```
npm install --save-dev grunt-contrib-watch
```

- Configure it

in Gruntfile.js config object

Don't forget to load it with loadNpmTasks()

```
watch: {
  files: ['js/**/*.js'], // watch these
  tasks: ['build'], // on changes, run this
  options: {
    spawn: false, // faster but error-prone
  },
}
```

Live Reload

- Get our browser to reload for file changes
- Grunt watch supports live reload out of the box
- Live reload vs Hot reload

Live Reload Configuration

- Configure the task

```
options: {
  livereload: 35729,
  spawn: false
}
```

- Then configure your browser
  - Either a <script>

```
<script src="//localhost:35729/livereload.js"></script>
```

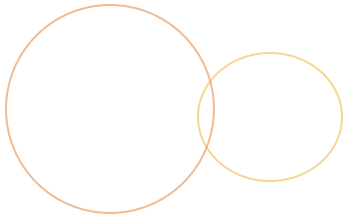
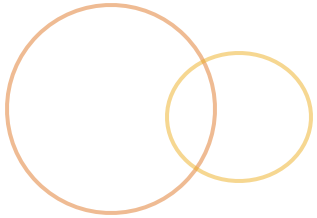
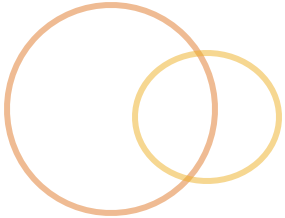
  - Or a browser extension
  - Or roll your own server…

Lab - Watch & Live Reload

Use watch to monitor file changes

- Install grunt-contrib-watch
- Configure the clean task
  - Reorganize and define aliases as needed
- Set up live reload
  - Add the <script> tag to index.html
- Run the grunt task that *includes* watch

```
$ npx grunt working
Running "jshint:src" (jshint) task
>> 3 files lint free.

Running "babel:generated" (babel) task

Running "copy:generated" (copy) task
Created 7 directories, copied 23 files

Running "watch" task
Waiting...
```


grunt

REFACTORING THE GRUNTFILE

Load NPM tasks automatically

- Want to avoid loading explicitly loading tasks?

```
loadNpmTasks('grunt-contrib-copy');  // 🙄
loadNpmTasks('grunt-contrib-clean'); // 😐
loadNpmTasks('grunt-contrib-watch'); // 😭
```

- A couple options:
  - matchdep
  - load-grunt-tasks

https://www.npmjs.com/package/load-grunt-tasks

Load Grunt Tasks installation

- Install it

```
npm install --save-dev load-grunt-tasks
```

- Require it in the `Gruntfile.js` *function*

in Gruntfile.js (not the config object)

```
require('load-grunt-tasks')(grunt);
```

- Remove any old `loadNpmTask()` calls

Organize app files

- We have lots of ***file patterns and lists*** in our config
- It may make sense at some point to refactor those into **shared properties** within the config
  - **Careful**: file order sometimes matters
    - Such as during concatenation
    - *But* not once we get into modules!
- **Grunt templates** will allow us to dynamically use properties throughout configs

```
{
  src: ['<%= files.js %>']
}
```

Grunt Templates

Gruntfile.js config

```
{
 files: {
  js: [
    'js/a.js',
    'js/b.js'
  ]
 },
 concat: {
  src: ['<%= files.js %>']
 }
}
```

This is an arbitrary property we added to track some files

We can reference it throughout the config object with templates

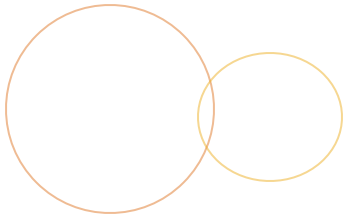
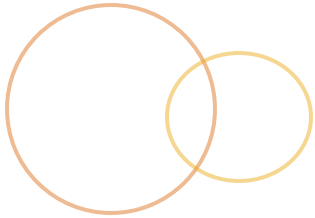
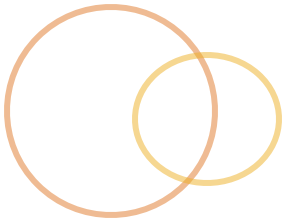
Is interpreted as:

```
…
 concat: {
  src: [
    'js/a.js',
    'js/b.js'
  ]
 }
…
```

Grunt Methods

- There are also grunt methods we can use in templates

```
{
 uglify: {
  options: {
    banner: '/*! <%= pkg.name %> <%=
grunt.template.today("yyyy-mm-dd") %> */\n'
  }
 }
}
```


grunt

UGLIFY

Concat, Minify and Uglify

- Optimize user experience by reducing the *number* and *size* of the files they need to download
  - **Bundling** js & css assets to reduce # of requests
  - **Minifying** text assets to trim white space, comments
  - **Uglifying** (or mangling) to reduce variable names
- HTTP/2?
  - Supports *multiplex* requests
  - *But*… Moderate bundling still shows benefits
- A few options out there
  - grunt-contrib-concat
  - uglify
  - bundlers and transpilers often build it in

Uglify

- We'll use **UglifyJS** @3
  - Supports ES6+ (now)
  - Generate source maps
  - Minifies & mangles
- Caveats
  - Concatenation (without modules) requires files are **combined in order**
  - We are probably **minifying compiled code**, not our original ES6+ files

https://github.com/gruntjs/grunt-contrib-uglify

Configuration

- Install it

```
npm install grunt-contrib-uglify --save-dev
```

- Configure it

in Gruntfile.js config object

```
uglify: {
  dist: {
    files: {
     'dist/js/all.min.js':
        ['generated/js/**/*.js']
    }
  }
}
```

Uglifying from generated/ to dist/

We'll probably also need to add a new copy task for dist

Source maps & mangling

```
uglify: {
  options: {
    sourceMap: {
      includeSources: true
    },
    mangle: {
      reserved: ['jQuery', 'Backbone']
    }
  },
  dist: {
    files: {
     'dist/js/all.min.js':
       ['generated/js/**/*.js']
    }
}
```

Inline sourcemaps?
Probably not in prod

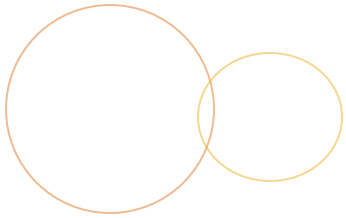
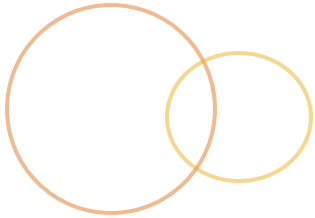
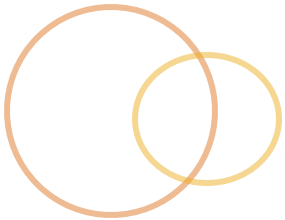
You can also just set
`mangle: true`

Mangle won't touch
global variable names

Let's test dist from our server

- Set up a new npm script to run the server against `/dist`
  - It's not working, what gives?
  - Our html file(s) are still pointing to the old scripts

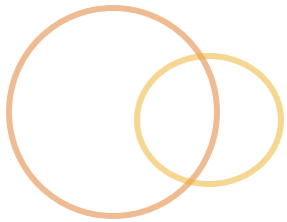
No lab yet — next Grunt Task!


grunt

TRANSFORMING HTML

Transforming HTML



- We may want to:
  - Pre-process templates
  - Handle HTML includes
  - Replace development-only content
  - Swap out dev or /generated <script> tags with our built <script> tag
- A handful of options
  - grunt-targethtml
  - grunt-preprocess
  - **grunt-processhtml**

https://www.npmjs.com/package/grunt-processhtml

Grunt ProcessHTML

- Install it

```
npm install --save-dev grunt-processhtml
```

- Configure it

in Gruntfile.js config object

```
processhtml: {
    build: {
     files: {
          'dist/index.html':
             ['public/index.html']
     }
  },
},
```

Using ProcessHTML

- Looks for HTML comments to replace

```
<!-- build:css css/all.min.css -->
<link rel="stylesheet" href="css/bootstrap.css">
<!-- /build -->
```

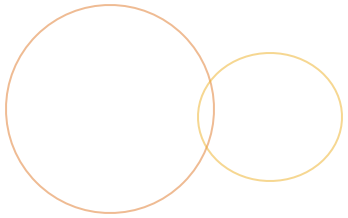
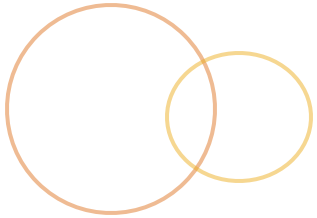
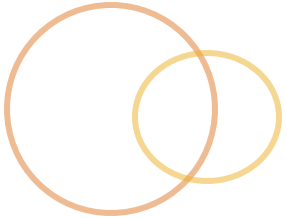
```
<!-- build:js js/all.min.js -->
<script src="js/app.js"></script>
<script src="js/stuff.js"></script>
<!-- /build -->
```

```
<!-- build:remove -->
<script src="//localhost/livereload.js"></script>
<!-- /build -->
```

Lab - Uglify and Transform

Prepare a distribution build and test it through your local server

- Install grunt-contrib-uglify and grunt-processhtml
- Configure the uglification first:
  - Uglify from /generated to /dist
  - Bundle app code and vendor code separately
  - You *may* need to ignore some files and specify files in a particular order (or not)
  - You will also need to 'clean' the /dist folder before new builds
- Test the distribution build with your server
  - Add a new npm script so you can "`npm run dist`"
- Then configure processhtml
  - Swap out the vendor and app script tags when you build "dist"
  - Remove the livereload tag, too
- Test the distribution server 


grunt

GOING FURTHER IN GRUNT

Recap

- Front-enders are *closer to the build system*
- We can *test and fix* build issues earlier
- *Front-end concerns* are better incorporated
  - Optimized images!
  - Optimized JS & CSS
  - Superset languages

Lots of plugins out there

- **template** processers, like *handlebar*
- **wget** - download resources and put them in your bundle
- **assemble** - carve up or build html from partials
- **newer** - trigger tasks only when a file is newer than the last version and/or task run
- **optimize** your images
- **css** minification and preprocessing
- auto-**open** your browser
- auto-**reload** your browser
- And of course custom tasks

Alternatives to grunt

- gulp
- npm
- brunch.io

Gulp - task runner

- More program than config
- Node-centric, uses pipes & file streams

```
npm install --save-dev gulp-babel
```

then in your gulpfile.js

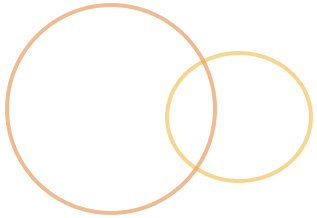
```
var gulp = require('gulp'),
 babel = require('gulp-babel')

gulp.task('build', function () {
  return gulp.src('src/app.js')
    .pipe(babel())
    .pipe(gulp.dest('build'))
});
```

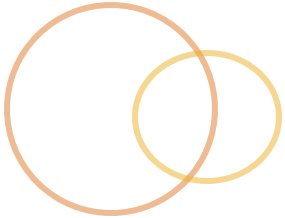
Gulp Example

```
var gulp = require('gulp');
var uglify = require('gulp-uglify');

gulp.task('scripts', function() {
 gulp.src(['public/js/**/*.js'])
    .pipe(uglify())
    .pipe(gulp.dest('build/js'));
});

// The default task (called when you run `gulp`)
gulp.task('default', function() {
  gulp.run('scripts');

  // Watch files and run tasks if they change
  gulp.watch('public/js/**', function(event) {
    gulp.run('scripts');
  });
});
```

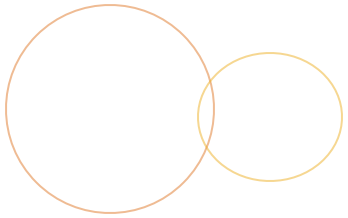
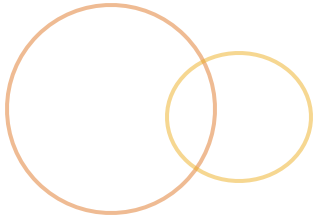
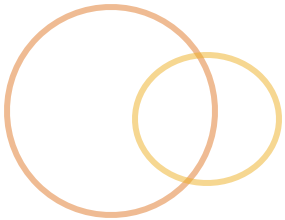
End… of Day 1

Day 2





- Writing Modules
- Handling Dependancies
- Bundling


modules

GO MODULAR

Issues with our application

- **Global** variables
- No **namespace**
- Nothing is very **reusable**/sharable
- **Unprotected** scope(s)
- **Dependencies** are not obvious
- Load **order matters** :/
- Not easy to **follow**
- As the app grows it will become painful to **maintain**

Go go complexity

- As an app grows, so does it's complexity

  "Big ball of mud"

- Modules attempt to solve these issues

  "Updating a chapter of a book should not require an update to every other chapter"

- Break down and organize functionality into smaller pieces that can be loaded as needed, re-used, shared, etc.

Why Modules

- Smaller surface area than *entire* app
- Define dependancies
- Define a public interface
- Encourage and support:
  - Encapsulation
  - Maintainability
  - Namespacing
  - Reusability
  - Debugging & Testing

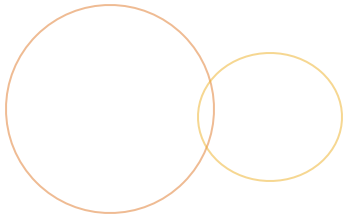
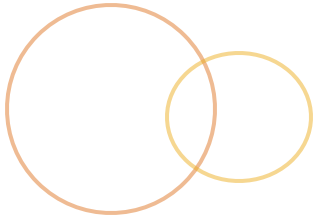
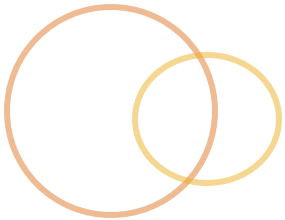
Module attributes

"Well-written *modules* provide solid **abstractions** and **encapsulation** boundaries, so that each module has a **coherent design** and a **clear purpose** within the overall application."

-webpack

Designing Modules

- Keep it simple
  - If all you need is an array, use an array
  - If all you need is a function, use a function
- Don’t over design up front
- *Think:*
  - Do I need to re-use this? Where.
  - How will I interface with this?
    - What methods does it have?
    - Is it static or will I need lots of copies?
  - Where are there dependencies I can remove?


modules

MODULES IN JS

Modules in JS

- Traditionally JS had no built-in module syntax
  - Scripts were loaded in sequence by the browser
  - Shared global space was abused
- Dependencies were handled via documentation
- Variations of IIFE were used to create modules
  - Closures create private scope & state
  - Objects acted as interface

Pattern: Object Namespace

- An object literal tracks our app-specific data
- Pros:
  - Some organization
  - Avoids global pollution
- Cons:
  - Entirely public
  - Not easily maintained
  - Limited reuse

```
const App = {
     name: 'My Mega App!',
     googleAnalyticsId: 'badifjeiweffwef',
     logger: function() {},
     logFormatted: function() {}
}
```

Pattern: IIFE

- Immediately Invoked Function Expression
- Creates a local scope that is “private”

```
(function() {
  const a = 1;
  console.log('This is private', a);
})();
```

The IIFE as a function

- We can export a global variable

```
const app = (function() {
  const a = 1;
  return function() {
    // run some other app code
  }
})();

app();
```

The IIFE as an interface

- Export an object to act as an interface

```
const app = (function() {
  const a = 1; // private
  function reload() {};

  return {
    init: function() {},
    stop: function() {},
    reload: reload
  }
)();

app.init();
app.stop();
```

This is the “revealing” pattern

Dependencies in IIFE modules

- Is no great way to pull this off
- But… we can help our module:
  - Be more self-contained
  - *Indicate* dependencies

```
// assuming we've linked to
// jquery.js and moment.js up above
const app = (function($, moment) {

  // can use $ inside
  // can use moment inside

)(jQuery, moment); // pass in their globals
```

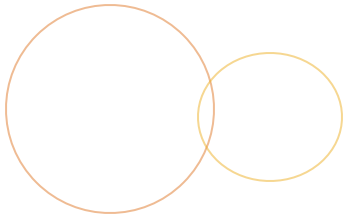
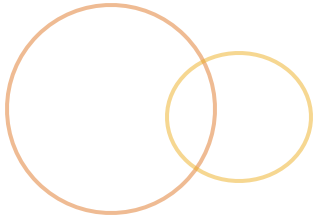
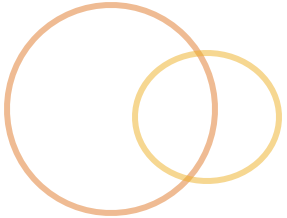
Downsides of the IIFE

- Still depends on a global variable(s)
- No real dependency management
  - Load order still matters
- Limited reuse
  - Only one value is “exported”
- Hard to analyze (by static analyzers)
- Hard to test in isolation

Lab - IIFE Modules

Update our JS (Timer App) to use IIFE modules

- Wrap logger.js as a module with a public interface
  - Update app.js's usage of logger functions
- Wrap app.js as a module
  - Does it need an interface?
  - Pass dependencies in rather than accessing globals
- Test it
  - `npx grunt`
  - `npm start`
- Can you see any other opportunities to organize?


modules

MODERN MODULES

Dependencies

- IIFE won't cut it
- We'll need a real module system
  - AMD
  - UMD
  - CommonJS
  - ES6

AMD (requirejs)

- Asynchronous Module Definition
- Designed for the web
- Requires a loader library
  - Dojo Toolkit
  - RequireJS
- Relies on two functions to help **define** modules and **require** dependencies

AMD Example

inside greeter.js

```
define(['./logger'], function() {
  return {
    hello: function() {
      console.log('hello');
    },
    goodbye: function() {
      console.log('goodbye');
    }
  };
});
```

Dependencies

inside app.js

```
require(['greeter', 'jquery'], function(greeter, $) {
  greeter.hello();
});
```

Dependencies

You can nest require() calls and conditionally require()

Downsides to AMD

- One module per file
- A lot of boilerplate
- Dependency on the loader system
- Hard to analyze
- Asynchronous loading may not be ideal
  - Transpile/compile step is suggested

CommonJS

- Designed for server-side JS in Node
- *Synchronously* loads dependencies
- Requires node
  - You *can* write in CommonJS
    then transpile it to be web-friendly
- Modules are
  - defined in separate files with
    an `exports` object specifying their interface
  - loaded via a `require()` function

CommonJS Example

inside modules/greeter.js

```
function hello () {
  console.log('hello');
}
function goodbye () {
  console.log('goodbye');
}

module.exports.hello = hello;
module.exports.goodbye = goodbye;
```

inside app.js

```
const greeter = require('./modules/greeter');
const $ = require('./modules/jquery');

greeter.hello();
greeter.goodbye();
```

Dependencies **can** be loaded programmatically

Downsides to CommonJS

- One module per file
- Hard to analyze
- Not appropriate for the web (synchronous)
  - You can transpile it:
    - Webpack
    - Browserify

UMD

- Universal Module Definition
- An attempt to marry AMD, CommonJS with a fallback to global variables
  - RequireJS
  - NodeJS
  - Etc…
- You can find quite a few web-friendly modules defined in this format

UMD Example

```
(function (root, factory) {
    if (typeof define === 'function' && define.amd) {
        // AMD
        define(['jquery'], factory);
    } else if (typeof exports === 'object') {
        // Node, CommonJS-like
        module.exports = factory(require('jquery'));
    } else {
        // Browser globals (root is window)
        root.returnExports = factory(root.jQuery);
    }
}(this, function ($) {
    //    methods
    function myFunc(){};

    //    exposed public method
    return myFunc;
}));
```

ES2015 Modules

- Native modules!
- Supports both synchronous and asynchronous loading
- Two new directives
  - `export` defines an available interface
  - `import` loads a module's interface
- Can't dynamically export and load, however
  - Supports static analysis
  - There are proposals to support it

ES2015 Module Example

inside modules/greeter.js

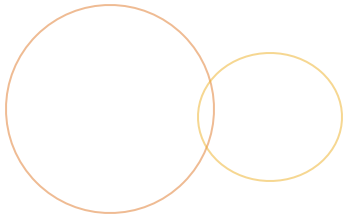
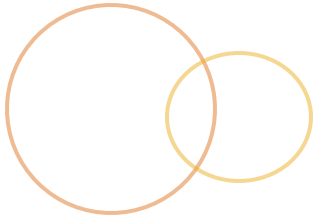
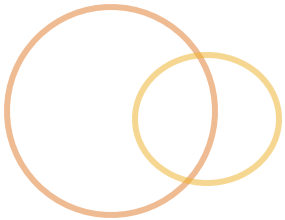
```
function hello () {
  console.log('hello');
}
function goodbye () {
  console.log('goodbye');
}

export hello;
export goodbye;
```

inside app.js

```
import { hello, goodbye } from 'modules/greeter';
import { * as $ } from 'modules/jquery';

hello();
goodbye();
```

Downsides to ES Modules

- Not yet widely supported
- Still need to transpile it
  - Webpack
  - Browserify
  - Rollup
- **though there is some limited web support in modern browsers*


modular js

AMD

AMD (RequireJS)

- Asynchronous Module Definition
- Built for the web
- RequireJS is our web-side loader and cli bundler
- Pros:
  - Easy to get going
  - Async loading
  - Lazy loading dependencies

Using RequireJS

- There will be two ways we use it
  - As a **dynamic module loader** on the web

```
<script src="require.js"></script>
```

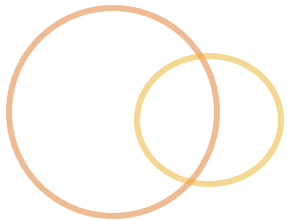
  - As a **module bundler** for final distribution

```
npm install requirejs
npx r.js -o app.config.js
```

AMD Loading on the web

1. Include the require.js script
2. Set up the configuration
3. Convert your modules to AMD format
4. Then define an app entry point

Configure it



```
require.config({
  // load modules by id from here
  // all module paths will be relative to here
  baseUrl: "scripts",
  paths: {
    // unless a module id begins with vendor
    vendor: '../vendor',
    // or the id is jquery
    jquery: '../vendor/jquery-1.3/jquery.min'
  },
  // give up loading a module after 15 seconds
  waitSeconds: 15
});
```

All the options:
https://github.com/requirejs/r.js/blob/master/build/example.build.js

Define() our modules

- Use `define()` to define modules for re-use
  - Won't be automatically invoked

```
// {string}                 module id (optional)
// {array}                  dependencies (optional)
// {function|object}        the module
define('logger', ['vendor/moment'], function(moment) {

  // I return my interface (optional)
  return {
    logFormatted: function () {}
  }

});
```

Require() our modules

- Use require() to load dependencies
  - Does not define a module for re-use
  - Typically your top level JS

```
// {array}                  dependencies (optional)
// {function}     a callback (optional)
require(['app', 'logger'], function(app, logger) {

  // Runs after dependencies are loaded

  require('a', function(a) {
    // I can programmatically require modules inline
  });

});
```

Define an entry point

- Load the initial dependencies
  - This *can* be in a separate script file, too

```
<script>
   require( ["app", "logger"],
    function(app, logger) {
        // called after dependencies are loaded
        // this call is optional
    }
  );
</script>
```

- Or just set the `data-main` attribute

```
<script src="js/require.js" data-main"js/main"></script>
```

base url

initial module

Bundling AMD

- Set up a config file for the optimizer

```
({
  baseUrl: 'public/scripts',
  paths: {},
  // tell RequireJS not to uglify our ES6
  optimize: 'none',
  // Entry point for the modules
  name: 'app',
  // output file
  out: 'generated/scripts/r-build.js'
});
```

- Run it

```
npx r.js -o requirejs.config.js
```

Grunt RequireJS Installation

- Install it

```
npm install --save-dev grunt-contrib-requirejs
```

- Configure it

in Gruntfile.js config object

You'll need to put your config & main require() in a separate script

We have to include the AMD loader

Which step of our workflow should this happen?

```
options: {
  optimize: 'uglify',
  baseUrl: 'generated/scripts',
  paths: {
    requireLib: '../vendor/require'
  },
  mainConfigFile: 'generated/scripts/main.js',
  name: 'main',
  out: 'dist/scripts/optimized.min.js',
  generateSourceMaps: true,
  includes: ['requireLib']
```

Include the loader?

- Optimize outputs files as AMD modules
  - You still need the require.js loader :/
  - *Or* you could wrap with almond (a loader shim)
    - https://github.com/requirejs/almond

AMD

- Pros:
  - **Asynchronous** loading (better startup times).
  - **Circular** dependencies are supported.
  - **Compatibility** for require and exports.
  - **Dependency** management fully integrated.
  - Modules can be split in multiple files if necessary.
- CONS
  - Slightly more **complex syntax**.
  - **Loader libraries are required** unless transpiled.
  - Hard to analyze for static code analyzers.

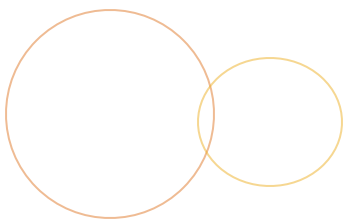
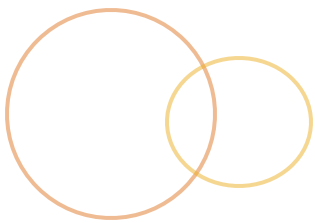
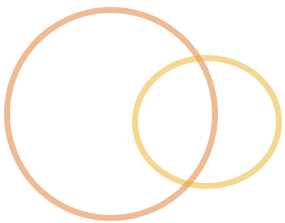
Lab - AMD Modules

Update our JS (Timer App) to use AMD modules

- Download the require script and include it in your app
- Define a configuration for require.js
- Wrap your modules as AMD modules
- Determine your entry point
- Test it in the /public or /generated builds
  - Don’t bother optimizing yet
- All done?
  - Try setting up bundling with grunt-contrib-requirejs

Define modules

- Each file is treated as a separate module
  - Local vars are private
- Export
  - defined on the exports object (or module.exports)
  - module.exports = {}
- Import
  - require('module');
- require.main === module
  - if node is running the module directly

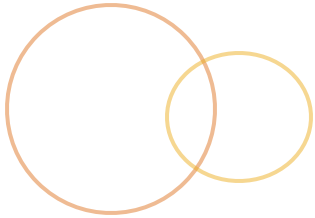




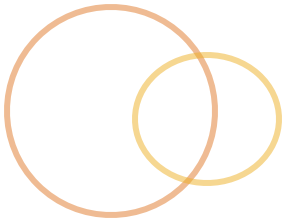

- Modules are loaded synchronously
- Modules are loaded only once
- require(‘/bla’) is absolute
- require(‘./bla’) is relative to the file calling require()
- require(‘bla’) is core or in node_modules

Module scope

- __dirname
- __filename
  - same as path.dirname(__filename)
- exports
- module
- require()

Patterns

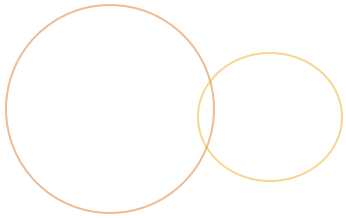
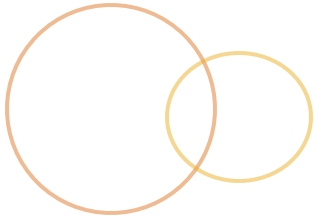
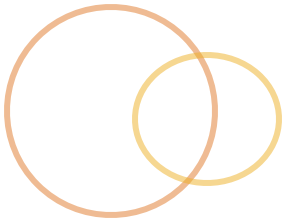




- module.exports.foo = {}
- require(‘./bla’).foo

Combos

- Webpack + babel
- Grunt + babel
- RequireJS + babel
- These guys are all doing just about the same thing
  - W/G/R is handling the module bundling
  - While Babel will transpile
- Babel can also transform modules now, too

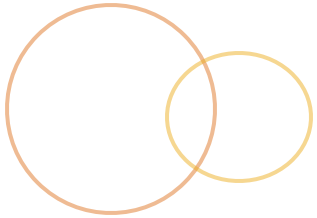

modular js

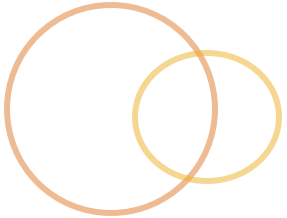
ES2015 MODULES

ES6, ES2015, Harmony

- Native JS Modules!
- Pros:
  - Supports both synchronous and asynchronous loading
  - Simple syntax
  - Easy to analyze statically
  - Circular dependencies supported
- Cons:
  - Still not supported everywhere.

Export





- Module are typically organized by file
- We'll use export to define interface(s)
- Exports are bound, aka "Named Exports"
- Can only export from the top level

testModule.js

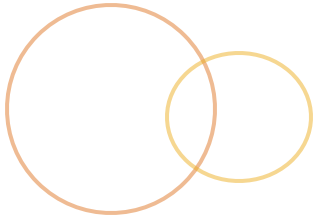
```
const foo = {};

export default 1;
export foo as default;
export foo;
export { hi: 'bar' } as hiBar;
export { foo as roo, bar };
```

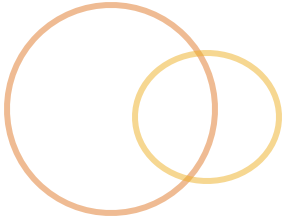
Named Exports

- When you export from a module you are exporting a binding - not just the value
  - If the module is used by other parts of the app, you are all using the *same named binding*
- Exports are exported as `const`

```
export let foo = 'bar';
export let bad = 'bla';

// what happens to exported foo?
setTimeout(() => foo = 'baz', 500);
```

Import





- We'll load module exports with `import`
- Can only be at the top level
- Imports are hoisted

```
// imports the default binding
import _ from 'lodash';

// import named "map" and "reduce"
// into local variables "map" and "reduce"
import {map, reduce} from 'lodash';

// alias an import
import {default as _, map} from 'lodash';
import {map as mapper} from 'lodash';

// import everything
import * as _ from 'lodash';
```

Read only imports

- You cannot re-bind an imported variable

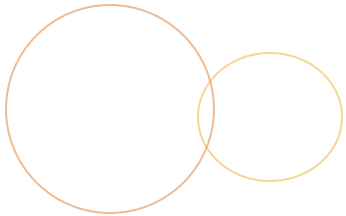
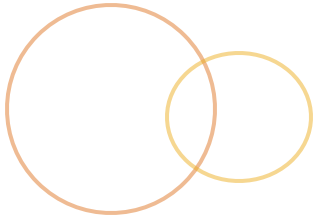
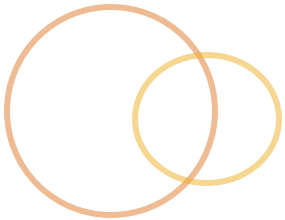
```
//------ lib.js ------
export let counter = 3;
export function incCounter() {
    counter++;
}

//------ main1.js ------
import { counter, incCounter } from './lib';

// The imported value `counter` is live
console.log(counter); // 3
incCounter();
console.log(counter); // 4

// The imported value can't be changed
counter++; // TypeError
```

Running with ES6 Today

- Some very light support in latest browsers
  - https://jakearchibald.com/2017/es-modules-in-browsers/

```
<script type="module" src="app.js">
```

- Most likely you'll need to transpile
  - From ES6 to AMD/UMD and then using a module loader
  - From ES6 to an ES5 friendly bundle
    - **Webpack**


bundlers

WEBPACK

Webpack intro

- Static module bundler
  - Recursively builds a dep. graph
  - Packages them into modules or a bundle
  - Understands all the JS module formats
- It understand JS natively but can be extended to transform your css/sass, images, html
- Involves **Loaders**
  - Specifies how to process a file type
- And **plugins**
  - Multi-purpose transformers

Set up webpack

- Install it

```
npm install webpack --save-dev
npm install webpack-cli --save-dev
```

- Determine an entry point (app.js?)
  - Imports dependences and start up the app
- Run webpack against the entry point

```
npx webpack public/js/app.js --output generated/js/bundle.js
```

Configuring

- Customize through a config file
  - Defaults to `webpack.config.js`

```
npx webpack --config webpack.config.js
```

webpack.config.js

```
const path = require('path');

module.exports = {
  entry: './public/js/app.js',
  output: {
    filename: 'bundle.js',
    path: path.resolve(__dirname, 'generated/js')
  }
};
```

Sourcemaps

- Use the **devtool** option
  - https://webpack.js.org/configuration/devtool/
- Lot's of options depending on the speed and output quality (ie: production) you want
  - eval
  - source-map
  - inline-source-map

```
…
devtool: 'cheap-eval-source-map'
…
```

Imports

- We can start using nom to manage JS dependencies

```
npm install moment --save;
```

```
import moment from 'moment';
```

- Ditch moment.js from our vendors folder

Grunt Webpack

- Install it

```
npm install --save-dev grunt-webpack
```

- Configure it

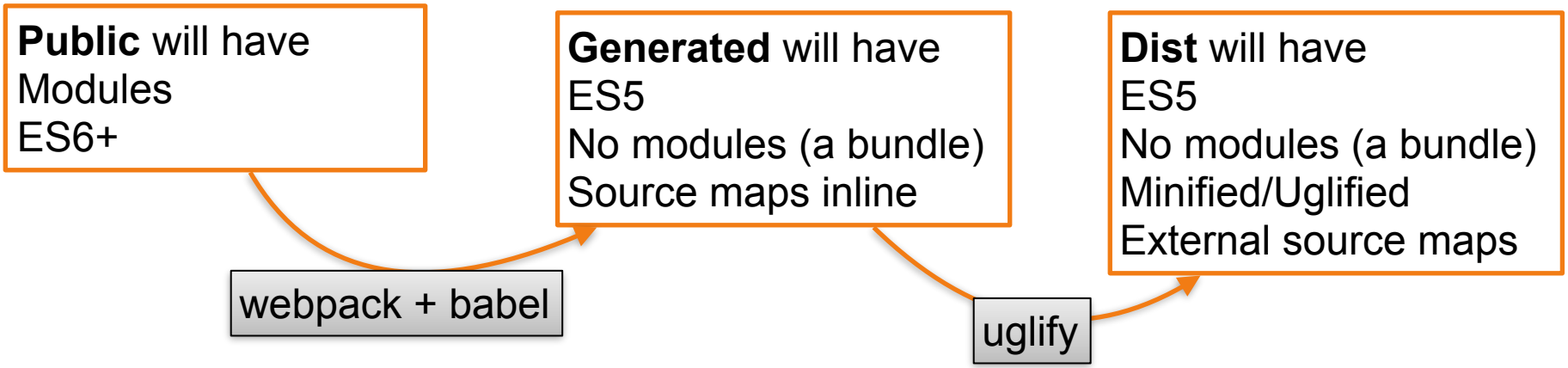
in Gruntfile.js config object

don't forget…
`const path = require('path');`
… web pack wants the path to be absolute

```
webpack: {
  generated: {
    mode: 'development',
    entry: './public/js/app.js',
    output: {
      filename: 'all.js',
      path: path.resolve(__dirname, 'generated/js')
    },
    devtool: 'inline-source-map',
    watch: false // or true
 }
}
```

Our workflow

- When should I be webpacking?
  - I can't run modules in the browser without it
- When should I babelify?
  - I want to test against ES5 code
  - Also: webpack has a *babel plugin*, hm….



Babel Loader

- Install

```
npm install babel-loader babel-core --save-dev
```

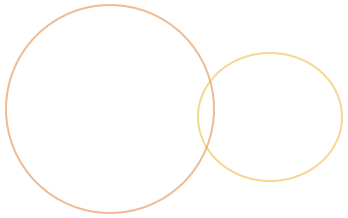
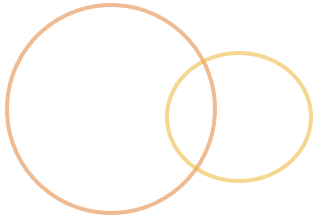
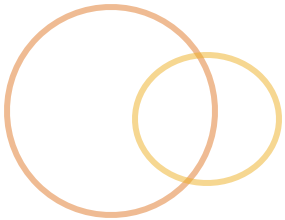
- Configure it in grunt or webpack.config.js

```
module: { // config for all modules
  rules: [{
     test: /\.js$/, // which files do I affect
     loader: 'babel-loader', // and which loader
     query: { // params to my loader
        presets: ['env']
     }
 }]
}
```

Lab - Webpack in Harmony

Update our JS (Timer App) to use ES6 modules and webpack (through grunt) for bundling

- Install webpack, webpack-cli and the grunt-webpack task
- Configure webpack through the Gruntfile.js
  - *If you prefer: You can start by configuring and running JUST web pack for testing purposes*
- Convert the code to use ES6 import & exports
- Drop moment.js in favor of **`npm install moment –save`**
- Get grunt+webpack outputting a bundle in your generated folder — turn off babel for this
  - Test out the generated app
- All done?
  - Try setting up babel-loader so that ES6 is transpiled


modular js

PARCEL

Parcel Intro

- *Another* bundler
- Can use index.html as your entry
- Understands js, css, html out of the box
  - Fewer need for customizing plugins
  - Uses transformers like babel automatically
- Built in server for hot reload
- Tree-shaking

```
npx parcel public/index.html
```

https://parceljs.org/

Build and Watch with Parcel

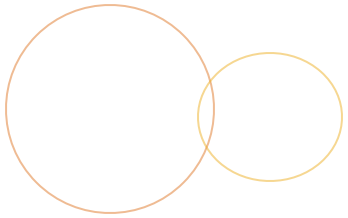
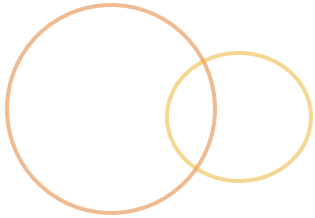
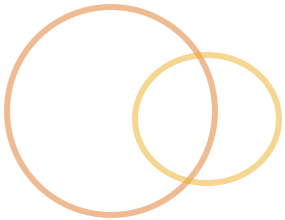
- Automatically transforms w/ Babel, PostCSS, etc
  - Just install them and set up *rc config files
- `watch`
  - Skip the server - just update on changes

```
parcel watch index.html
```

- `build`
  - Compile the output and minifies with uglify-es

```
parcel build index.html --out-dir dist
```

https://github.com/rm-training/modular-js/tree/timer-app-parcel-bundler


going beyond

MODULES

Wrapping up

- **Build tools?**
  - Be opinionated
  - Explore and test
  - Don’t embed yourself in *one* tool
    - Favor tools that incorporate long-living plugins
  - Don’t over complicate stuff…
    - Get it working and step away
    - If it slows you down, remove it
- **ES6?**
  - Use it!
- **Modules?**
  - Use ES6 Modules - node & web support is on the way

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**

Final Lab - To Do App

Build a simple “to do” web application using modern JavaScript and ES6 modules

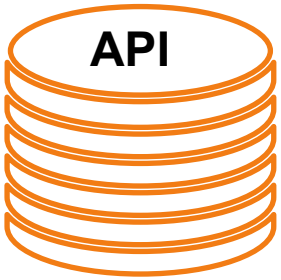
- Start from scratch or use my shell app
  - https://github.com/rm-training/modular-js/releases/tag/todo-app-start-point
- NPM install any external modules you like:
  - jquery, moment, http-server, json-server, handlebars
- **Let’s design** our workflow
  - Use Grunt as a task runner
  - Use any bundler you want
    - Webpack, Parcel, Rollup
  - We’ll need a local server and an API
- **Let’s design** our app
  - OO? Static Objects?
  - TodoList & TodoItems

Final Lab - System Design

Local Server can be:
http-server
Parcel
Your own server IYW

JS handles all
Data GET, POST
and DELETEs

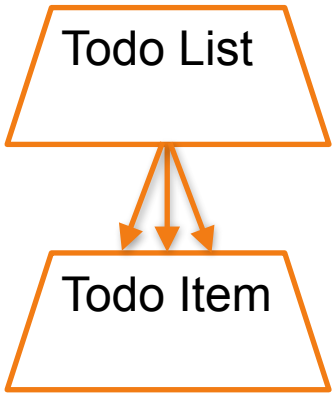
The API can be…
A json-server
localStorage faking it
A live test server




```
GET        /todo
POST       /todo
GET        /todo/{n}
POST/PUT   /todo/{n}
DELETE     /todo/{n}
```

Final Lab - App Design

Domain Design



Manages related items
Gets all items
Render all items?

Manages item state
Add and Editing of Items
Renders itself?

UI Elements

A form where user can submit todos
A list of todos displayed to the user
Individual todo items in the list

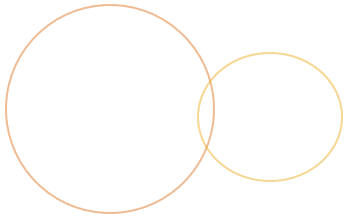
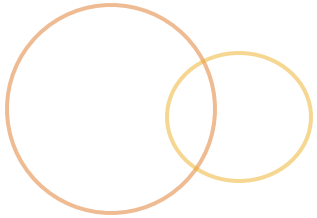
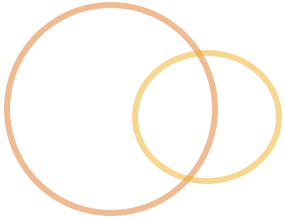
Supported Functionality

User can add new items to the list
User can complete items in the list
User can delete items from the list
User can un-complete items in the list

Supported Behaviors/Stories

When a user submits the form it should **validate**
When a user submits a valid form then a new, incomplete **item is displayed** in the list
When a user completes an item it **changes** to completed
When a user deletes an item it is **removed** from the list
When a user un-completes an item it **changes** to incomplete

Bonus: Items support a due date
When an item is past due it can't be completed

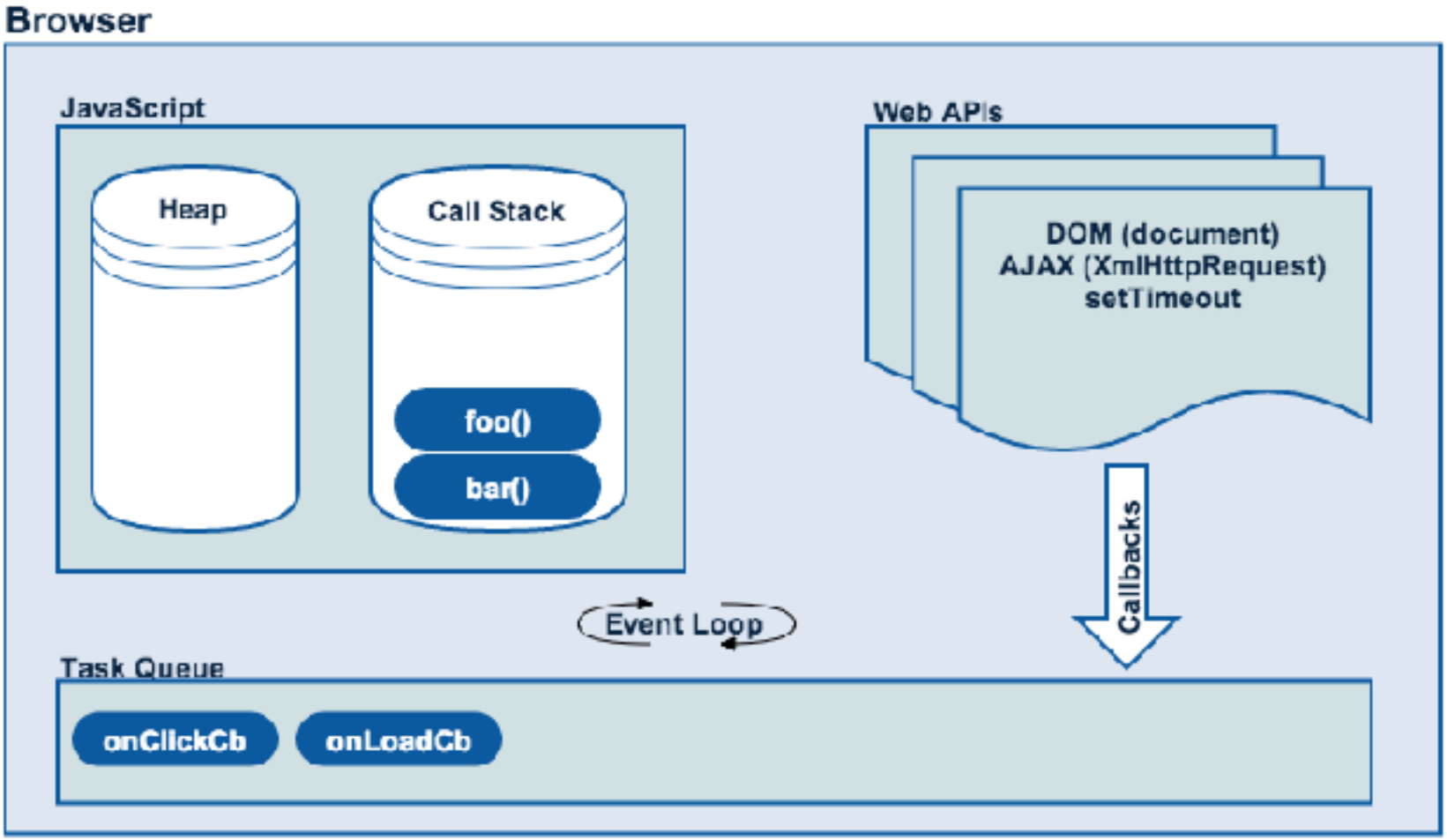





module

ASYNCHRONOUS PROGRAMMING

Single-threaded JavaScript

- Does everyone know the event-loop?



Being Asynchronous

- Because JavaScript cannot do more than one thing at a time…
  - Callbacks
  - Promises
  - [ES6] `async` and `await`
  - Observables

Callback Pattern

- A function passed to another function as a parameter
  - …so that it can be invoked later by the calling function.
- Aren't asynchronous on their own
  - …but we tend to use them for such things
  - ex: event handling, ajax handling, file operations, etc

```
function callLater(fn) {
  // do some async work
  return fn();
}

callLater(function() {
  console.log("I'm done!");
});
```

Callback Context

- **`this`** inside a callback may change, be careful

```
setTimeout(function() {
   console.log("I was called later");
}, 1000);

$('a').on('click', function() {
   console.log(this); // ?
});
```

The Downside to Callbacks

- Can become deeply nested and not easy to reason
- There is no guarantee that the callback will be invoked when you expect, if at all

```
// callback hell
async1(function(err, result1) {
    async2(function(err, result2) {
        async3(function(err, result3) {
            async4(function(err, result4) {
                /*…*/
            });
        });
    });
});
```

Promises

- A **Promise** represents a proxy for a value not necessarily known when the promise is created
  - They represent the *promise of future value*
- **Benefits:**
  - Guarantees that callbacks are invoked
  - Composable (can be chained)
  - Immutable (one-way latch)
  - You can continue to use them after resolved
  - They are objects you can pass around
- **Bummers:**
  - ES6+
  - No `.finally()`

Making a Promise

- Construct a Promise to represent a future value
  - Constructor expects a single argument, which is a function that has two arguments, **`fulfill`** and **`reject`**
- Attach handlers using **`then`** method
  - The handler consumes the later-value when it's ready
  - And handles errors, too

```
var promise1 = new Promise(function(fulfill, reject) {
        async1(function(err, data) {
                if (err) {
                        reject(err);
                } else {
                        fulfill(data);
                }
        });
});
promise.then(onFulfilled, onRejected);
```

Promises Terminology

- Specification: https://promisesaplus.com
  - **pending** – the action is not fulfilled or rejected
  - **fulfilled** – the action succeeded
  - **rejected** – the action failed
  - **settled** – the action is fulfilled or rejected

```
var p = new Promise(
  function(resolve, reject){
  ...
  if(something)
     resolve({});
  else{
     reject(new Error());
  }
})
```

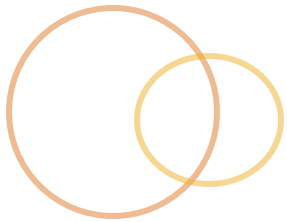
```
p.then(
  function(data){
    ...
  },
  function(err){
    ...
  }
);
```

Promise Errors

- `fulfill()` and `reject()` don't explicitly return from the constructor
- Handle errors thrown
  - Use the reject/error handler argument in `then()`
  - ES6 Promises also support a `.catch()` callback, which will do the same thing.

```
var promise1 = new Promise(function(fulfill, reject) {
     setTimeout(function() {
         reject("Something went wrong!");
     }, 1000:
});


promise1.then(null, function(error){
     console.log('Something went wrong', error);
});


prom1.catch(function(err) {
    console.log(err);
});
```

Chaining Promises



- **`.then()`** wraps any return value as a new Promise
  - …can chain them
  - you can specify a *new* promise to return
  - in this way you can have a waterfall of operations dependent on the previous completing

```
var promise1 = new Promise(function(fulfill, reject) {
     setTimeout(function() {
         fulfill(5);
     }, 1000:
});

promise1.then(function(data){
    console.log(data); // 5
    return data + 2; // returns a new promise
}).then(function(data) {
    console.log(data); // ?
}).catch(function(err) {
    console.log(err);
});
```

Fixing callback hell

- Remember this? Let's see what that would look like if we wrapped each async operation in a promise

```
async1(function(err, result1) {
    async2(function(err, result2) {
        async3(function(err, result3) {
        });
    });
});
```

Promised Land

- If each of our async functions returned a promise object, we could do this:

```
promise1
    .then(promise2)
    .then(promise3)
    .catch(function(err) {
        // deal with thrown error
    });
```

Promise breaking

- What is wrong with the below promise sequence?

```
fetchResult(query)
    .then(function(result) {
        // this is an async operation
        asyncRequest(result.id);
    })
    .then(function(newData) {
        console.log(newData);
    });
    .catch(function(error) {
        console.error(error);
    });
```

Composing Promises

- `Promise.all([…])`
  - Returns a promise that resolves when all promises passed in are resolved or at the first rejection
  - Fulfilled value is an array of all returned promise values
- `Promise.race([…])`
  - Returns a promise that resolves when any one promise is fulfilled or rejected

Composing Promises Example

```
var p1 = Promise.resolve(3);
var p2 = 1337;
var p3 = new Promise(function(resolve, reject) {
    setTimeout(resolve, 1000);
});

Promise.all([p1,p2,p3]).then(function(data) {
    console.log(values); // ?
});

Promise.any([p1,p2,p3]).then(function(data) {
    console.log(data); // ?
});
```

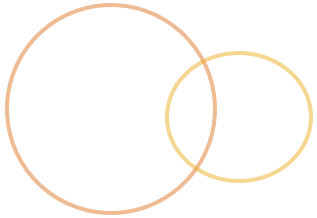
Async and await [ES6]

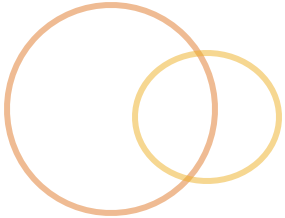
- Two new keywords allow us to write asynchronous code that looks and feels synchronous
- **`async function`**
  - Defines an asynchronous Function that can **yield flow of control** back to the caller
  - The function immediately returns a Promise that will be resolved when the function returns a value or rejected when it has an error
    - The function is resolved with any return value
    - Errors with any error thrown
- **`await`**
  - Informs code *within* an async function to yield/wait for an *internal* Promise to resolve before proceeding

From this…

```
function getAndRenderArtists() {
  var artists;
  Ajax.get(“/api/artists/1”)
   .then(function(data){
      artists = data;
      return Ajax.get(“albums”);
    })
    .then(function(data){
      artists.albums = data;
      View.set("artist", artist);
    })
    .catch(function(err){});
}
```

… to this





```
async function getAndRenderArtists() {
  var artist = await Ajax.get("/api/artists/1");
  artist.albums = await Ajax.get(
    “/api/artists/1/albums"
  );
  View.set("artist", artist);
}

var rendered = getAndRenderArtists();
rendered.then(function(response) {
  console.log(‘Page is loaded’);
});
```

Exercise - Promises

- **callLater**
  A function that sets up a waterfall of promises
  - Fork: https://jsfiddle.net/mrmorris/kp4gqp69/
- **Ajax with Promises**
  Set up an Ajax utility object that makes ajax requests and returns a promise
  - Fork: https://jsfiddle.net/mrmorris/5yzby96w/

Solutions:
callLater - https://jsfiddle.net/mrmorris/sLbmmq4g/
Ajax with Promises - https://jsfiddle.net/mrmorris/oa1jbgr3/

Going beyond

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

Stay sharp

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

Go now and code well

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