Distilled JavaScript

Cooking on a MEAN stack

We're Osper.

We believe in financial freedom for young people.

We believe in building good habits early.

We believe in simplicity and safety.

We believe it's time for something new.

About me - Grahame ‘Frank’ Oakland

Programmer - recovering Java-holic

JavaScripter - {

```
  1:’Closure’,
  2:’AngularJS’,
  3:’ExtJS’,
  4:’YUI’,
  5:’jQuery’,
  6:’Vanilla DOM’
}
```

Now searching for the **MEAN**-ing of life?

About you

Where are you from?

What do you do?

Why are you here?

How **MEAN** are you?

Session 1 - firing up the still

JavaScript warm up

Loose coupling, patterns, and the best practices of JavaScript design

Tools of the master craftsman (node, grunt, bower, yeoman, qunit)

Project Outline - Choosing the blend - **tea.js**

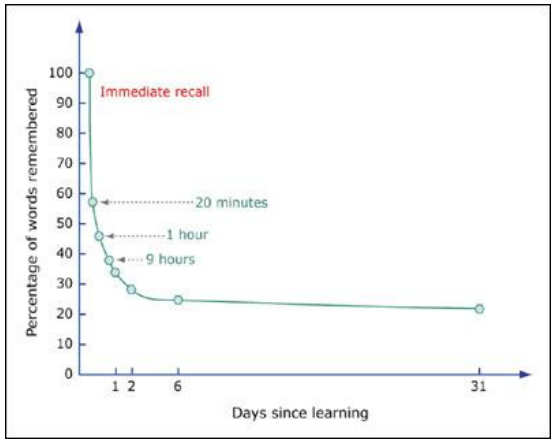
MongoDB - putting the **M** in MEAN

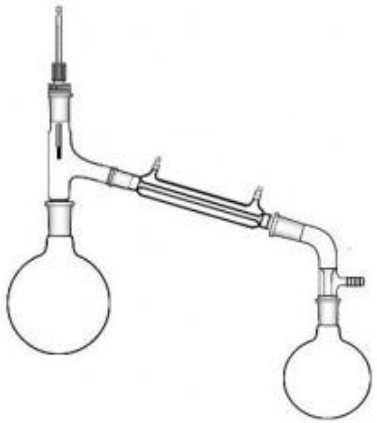
Why Distilled?

Lack of time - how can we learn new stuff when we have so much work on?

Lack of practice - how do we get over the initial learning hump and get in shape?

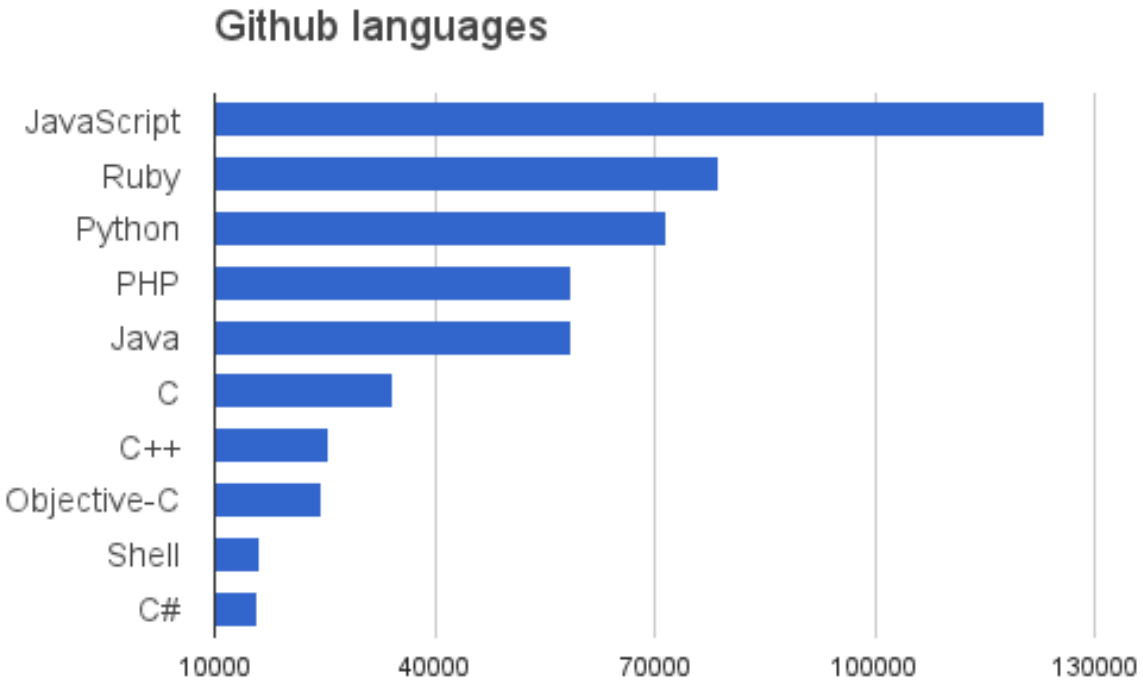
Lack of people - how can we connect with other people to learn with?





JS

Why JavaScript? - the most popular language on the planet today?



JS

But

not everyone agrees

The JavaScript Problem

1 The problem

The JavaScript problem is two-fold and can be described thus:

1. **JavaScript sucks.** The depths to which JavaScript sucks is well-documented and well-understood. Its main faults are: its lack of module system, weak-typing, verbose function syntax[1], late binding[2], which has led to the creation of various static analysis tools to alleviate this language flaw[3], but with limited success[4] (there is even a static type checker[5]), finicky equality/automatic conversion, `this` behaviour, and lack of static types.

1. **We need JavaScript.** Using it for what it is good for, i.e. providing a platform for browser development, but not using the language *per se*, is therefore desirable, and many are working to achieve this, in varying forms. There are various ways to do it, but we ought to opt for compiling an existing language, Haskell, to JavaScript, because we do not have time to learn or teach other people a new language, garner a new library set and a new type checker and all that Haskell implementations provide.

Exercise 1

Use JavaScript to fix the numbering problem in the Haskell Wiki page

```
git clone https://github.com/distillers/firing_up.git
```

The_JavaScript_problem.html

 JavaScript sucks. The depths to which JavaScript sucks is well-documented and well-understood. Its main faults are: its lack of module system, weak-typing, verbose function syntax[1], late binding[2], which has led to the creation of various static analysis tools to alleviate this language flaw[3], but with limited success[4] (there is even a static type checker[5]), finicky equality/automatic conversion, <code>this</code> behaviour, and lack of static types.

 We need JavaScript. Using it for what it is good for, i.e. providing a platform for browser development, but not using the language per se, is therefore desirable, and many are working to achieve this, in varying forms. There are various ways to do it, but we ought to opt for compiling an existing language, Haskell, to JavaScript, because we do not have time to learn or teach other people a new language, garner a new library set and a new type checker and all that Haskell implementations provide.


WEATHER REPORT
WEATHER REPORT

Callback pattern - making use of first class objects

```
// findNodes() that accepts a callback function to apply on each one
var findNodes = function (callback) {
    var i = 100,
        nodes = [],
        match;

    // check if callback is callable
    if (typeof callback !== "function") {
        callback = false;
    }
    while (i) {
        i -= 1;
        // complex logic to find match here…

        // now callback:
        if (callback) {
            callback(match);
        }
        nodes.push(match);
    }
    return nodes;
};
```

Exercise 2

Break our code into two separate parts (*search*) and (*modify*) and use the Callback pattern


return to forever
featuring
chick corea
hymn of the seventh galaxy
Polydor

Revealing module

```
var aRevealingModule  = (function() {

    var privateCounter  = 0;

    function privateFunction  () {
        privateCounter ++;
    }
    function publicFunction () {
        publicIncrement();
    }
    function publicIncrement () {
        privateFunction();
    }
    function publicGetCount () {
         return privateCounter;
    }
    // Reveal public pointers to  private functions and properties
    return {
        init: publicFunction,
        increment: publicIncrement,
        count: publicGetCount
    };
}());
```

Exercise 3

Create a simple module based on the *Revealing Module* pattern for our code and ensure it is in its own file


WASA
EXPRESS

QUnit - js unit test goodness

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>QUnit Example</title>
  <link rel="stylesheet" href="/resources/qunit.css">
</head>
<body>
  <div id="qunit"></div>
  <div id="qunit-fixture"></div>
  <script src="/resources/qunit.js"></script>
  <script src="/resources/tests.js"></script>
</body>
</html>
```

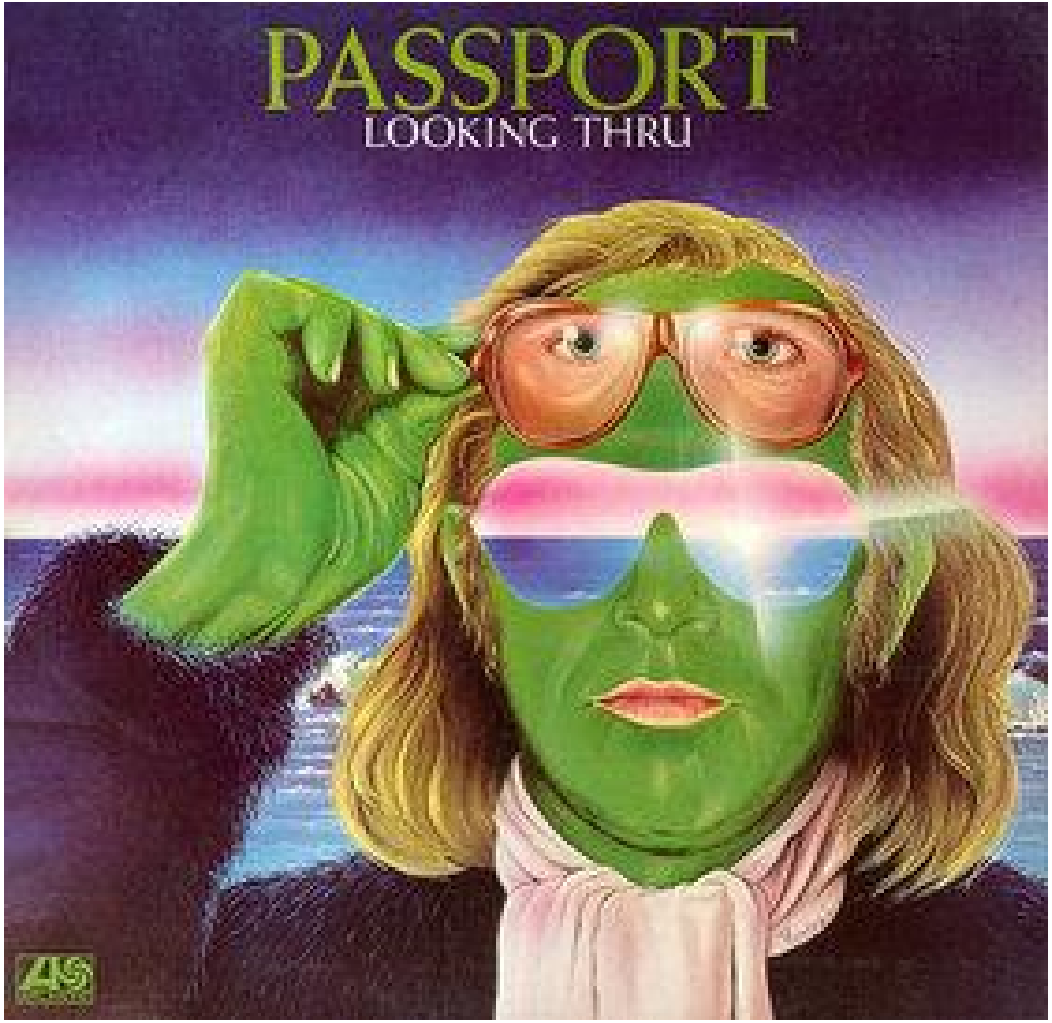
```
test("hello Qunit test world test", function() {
  ok(1 == "1", "Passed!" );
});

test("hello weird JavaScript ", function() {
  ok("0" == false, "Strange but true!" );
});
```

Exercise 4

Grab the QUnit library and create some simple *test.js* file for our module

"To use QUnit, you only need to include two QUnit files on your HTML page. QUnit consists of qunit.js, the test runner and testing framework, and qunit.css, which styles the test suite page to display test results…"


PASSPORT
LOOKING THRU

Web Framework Pattern - can you recognize this?

JS

Templates - separate messy markup from your JavaScript code

```
<script type="text/javascript"
    src="/scripts/handlebars-1.3.0.js" />
```

```
<script id="some-template" type="text/x-handlebars-template">
  <table>
    <thead>
      <th>Username</th>
      <th>Real Name</th>
      <th>Email</th>
    </thead>
    <tbody>
      {{#users}}
        <tr>
          <td>{{username}}</td>
          <td>{{firstName}} {{lastName}}</td>
          <td>{{email}}</td>
        </tr>
      {{/users}}
    </tbody>
  </table>
</script>
```

```
var source   = $("#some-template").html();
  var template = Handlebars.compile(source);
  var data = { users: [
      {username: "alan", firstName: "Alan", lastName: "Johnson", email: "alan@test.com" },
      {username: "allison", firstName: "Allison", lastName: "House", email: "allison@test.com" },
      {username: "ryan", firstName: "Ryan", lastName: "Carson", email: "ryan@test.com" }
    ]};
  $("#content-placeholder").html(template(data));
```

Exercise 5 - build your mini MVC

Use Handlebars templates to create your own mini client MVC using the Haskell content (model) to render the *The_JavaScript_Problem.html* page's 'problem' section using the *problems.json* model


ИГОРЬ
САВУШКИН
ПЕСНИ
ИЗ К/ФО
„СОЛНЦЕ,
СНОВА
СОЛНЦЕ"
СТИХИ
ЕВГ. ЕВТУШЕНКО
ИНСТРУМЕНТАЛЬНАЯ
МУЗЫКА

Part II

JavaScript - a freaky fusion

SELF: *"SELF offers a new paradigm for object-oriented languages that combines both simplicity and expressiveness. Its simplicity arises from realizing that classes and variables are not needed. Their elimination banishes the metaclass regress, dispels the illusory distinction between instantiation and subclassing, and allows for the blurring of the differences between objects, procedures, and closures…"*

- Ungar, D., and Smith, R. B. SELF: The Power of Simplicity

Scheme: *"Scheme is a statically scoped and properly tail-recursive dialect of the Lisp programming language invented by Guy Lewis Steele Jr. and Gerald Jay Sussman. It was designed to have an exceptionally clear and simple semantics and few different ways to form expressions. A wide variety of programming paradigms, including functional, imperative, and message passing styles, find convenient expression in Scheme."*

- Revised^{6} Report on the Algorithmic Language Scheme

JazzRock - another freaky fusion

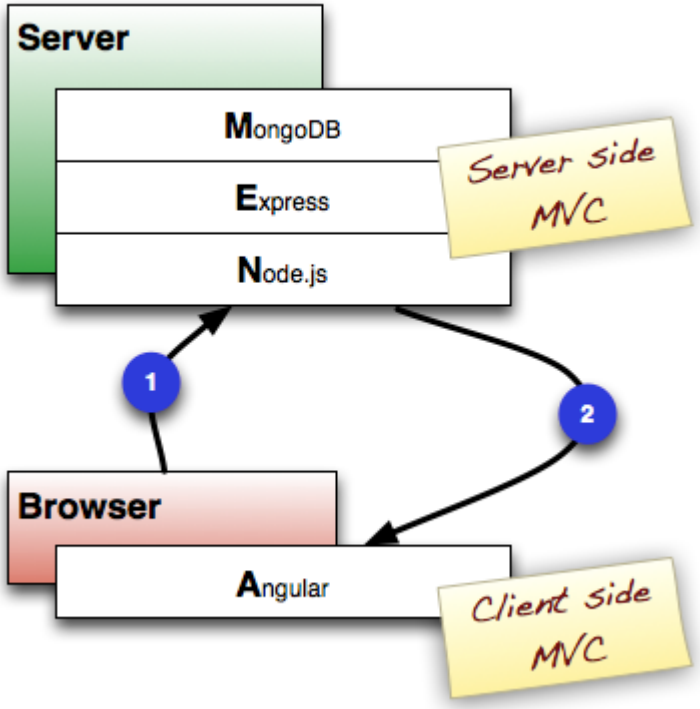
What is meant by **MEAN?**

MongoDB NoSQL document based data store

Express Web framework for reactive node apps

AngularJS Client side framework for MV-*whatever*

Node.js JavaScript platform built on V8



JS

Exercise 6

1. If not already done so, install **node** and ensure it works

2. Install the **qunit** and **jshint** plugins and run them for our mini MVC project

```
>jshint --h
Usage:
  jshint [OPTIONS] [ARGS]

Options:
  -c, --config STRING    Custom configuration file
      --reporter STRING  Custom reporter (<PATH>|jslint|checkstyle)
      --exclude STRING   Exclude files matching the given filename pattern
                         (same as .jshintignore)
      --exclude-path STRINGPass in a custom jshintignore file path
      --verbose          Show message codes
      --show-non-errors  Show additional data generated by jshint
  -e, --extra-ext STRING Comma-separated list of file extensions to use
                         (default is .js)
      --extract [STRING] Extract inline scripts contained in HTML
                         (auto|always|never, default to never)  (Default is never)
      --jslint-reporter  Use a jslint compatible reporter (DEPRECATED, use
                         --reporter=jslint instead)
      --checkstyle-reporter Use a CheckStyle compatible XML reporter
                            (DEPRECATED, use --reporter=checkstyle
                            instead)
  -v, --version          Display the current version
  -h, --help             Display help and usage details
```

Tools - get tooled up for rapid development

node - JavaScript platform built on V8 engine with a swish REPL

npm - The package manager for node that comes included

jshint - linter that will ensure our code is lovely and warm

tests - qunit - jQuery's own unit test framework, jasmine - story style

grunt - build tool

bower - package management

yeoman - create scaffolding for our projects

MEAN workflow tools

Grunt is used to build, preview and test your project, thanks to help from tasks curated by the Yeoman team and grunt-contrib.

Bower is used for dependency management, to download and manage your scripts, plugins or front-end packages.

Yeoman (yo) scaffolds out a new application, writing the Grunt config and grabbing relevant Grunt tasks and Bower dependencies

MEAN workflow tools - Grunt

Grunt is a task runner and is used to build, preview and test your project

```
module.exports = function(grunt) {
    grunt.initConfig({
        pkg: grunt.file.readJSON('package.json'),
        uglify: {
            // uglify task configuration ...
        }
    });
    // load the plugin that provides the 'uglify' task.
    grunt.loadNpmTasks('grunt-contrib-uglify');
    // log something
    grunt.log.write('Hello world!\n');
    // define default task(s).
    grunt.registerTask('default', ['uglify']);
};
```

Exercise 7

1. Install **grunt** using node package manager `npm -g install grunt`
2. Create the **dist**, **src** and **test** directories for the project
3. Install `grunt` locally for our project
3. Create the `package.json` and `Gruntfile.js` config files
4. Create tasks for linting (jshint), testing, (qunit) concatenating and minifying (uglify)
5. Get gruntin! `grunt`

MEAN workflow tools - Bower

Bower is a simple no fuss package manager for your projects (in effect shortcut for git)

```
Usage:

    bower <command> [<args>] [<options>]

Commands:

    cache                   Manage bower cache
    help                    Display help information about Bower
    home                    Opens a package homepage into your favorite browser
    info                    Info of a particular package
    init                    Interactively create a bower.json file
    install                 Install a package locally
    link                    Symlink a package folder
    list                    List local packages
    lookup                  Look up a package URL by name
    prune                   Removes local extraneous packages
    register                Register a package
    search                  Search for a package by name
    update                  Update a local package
    uninstall               Remove a local package
```

Exercise 8

1. Install **bower** using node package manager `npm -g install bower`
2. Verify it works `bower help`
3. Download some packages angular backbone etc
4. Use `bower init` to create a `bower.json` file
5. Now delete the previously packages - *of course you'll need to find them first!*
6. Run `bower install` and check it re-downloads the stuff in `bower.json`

MEAN workflow tools - Yeoman (yo)

Yeoman (yo) scaffolds out a new application, writing the Grunt config and grabbing relevant Grunt tasks and Bower dependencies

"Yeoman is an open source project which defines an opinionated stack for web application development. It includes a golden bundle of tools and frameworks, provided with documentation and authority…"

Exercise 9

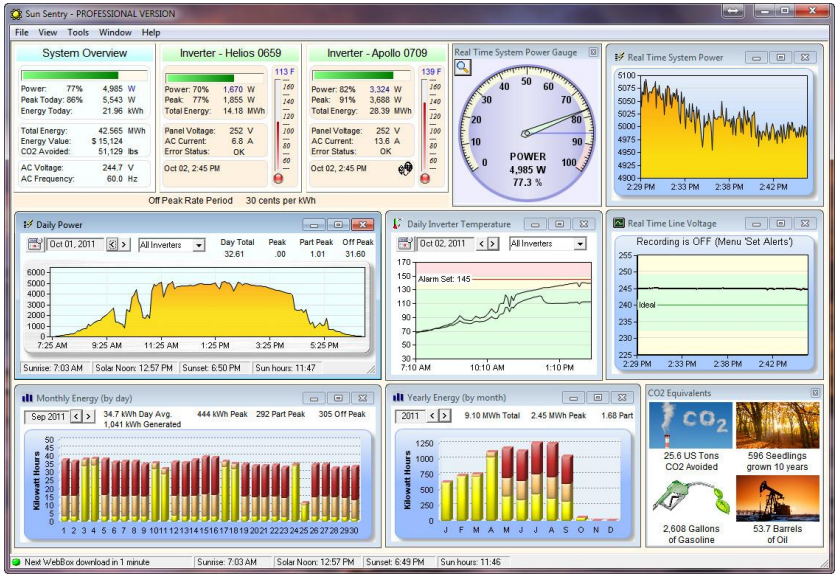
1. Install yeoman `npm -g install yo`
2. Install the MEAN generator `npm install -g generator-meanstack`
3. Run it `yo meanstack`
4. See what happens when you `grunt`

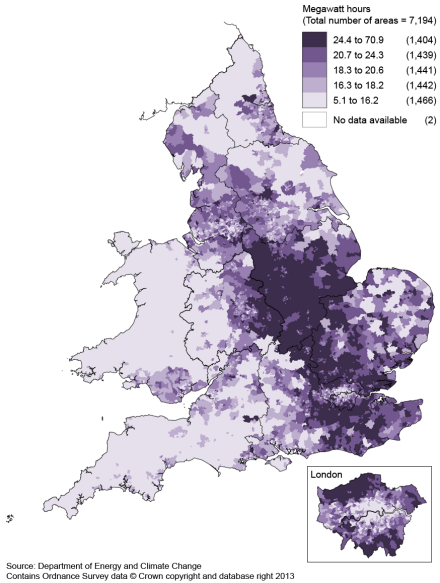
Have a brew

tea.js - how much does a cup of tea cost?

“So I have been thinking about getting something to track my energy bills, I'm a consumer only, no fancy pants solar cells I'm afraid. I would like to monitor gas, water and electricity. My gas meter has a pulse output and I plan on adding a water and electricity meter with the same.

...for example, I’d like to see exactly how much energy I use when I make a cup tea of tea and how this saving could make a difference if others are doing the same”





JS

tea.js - data logging of smart meters using Energy monitor

“Remember to only fill the kettle with as much water as you need. British people waste £91m in electricity when boiling kettles with too much water in them. You can save up to £10 a year…”

- periodically poll Energy monitor (~every 5mins)
- store the results for Water, Electricity and Gas
- push the updates to the tea application and display accordingly
- publish our savings to the street, town world!

JS

MEAN persistence - MongoDB

mongo is the NoSQL document data store which uses BSON binary JSON for its data

```
// Retrieve
var MongoClient = require('mongodb').MongoClient;

// Connect to the db
MongoClient.connect("mongodb://localhost:27017/exampleDb", function(err, db) {
  if(err) { return console.dir(err); }

  var collection = db.collection('test');
  var doc1 = {'hello':'doc1'};
  var doc2 = {'hello':'doc2'};
  var lotsOfDocs = [{'hello':'doc3'}, {'hello':'doc4'}];

  collection.insert(doc1);
  collection.insert(doc2, {w: 1}, function(err, result) {});
  collection.insert(lotsOfDocs, {w: 1}, function(err, result) {});
});
```

Exercise 9

1. Install **mongodb**
2. create a **tea** database
3. Connect to it
4. Add some test tables for **electricity**, **gas** and **water**

TEA - Swiss prog rock *'super group'* from 1973

Review About the course

Day 1: JavaScript distilled and putting the M in mean (mongodb)

Day 2: Views - Angular and Backbone

Day 2: Server side - Express and Node

Day 4: Node and project wrap up