JavaScript

- Created by Brendan Eich
  - Netscape, now Chief Technology Officer at Mozilla Corporation

- Formalized as ECMAScript (ECMA-262)
  - Dates back to the battles Netscape (JavaScript) vs Microsoft (JScript)

- Multiparadigm Language
  - Functional
  - Object Oriented
  - Imperative

- Prototype-based

- Originally intended for client side, but now increasingly popular on server-side
Agenda

- https://docs.google.com/document/d/1517aC0-wFxxXSREusscwRuTGDEHTr6mrJqtkoOhNcM/edit
Online Editing

- JsFiddle
  - Recursive AngularJS Views: http://jsfiddle.net/eu81273/8LWUc/18/

- Plunkr
  - Service vs Factory vs Provider
    http://plnkr.co/edit/UDzRruJGVAHBkovf0OPT?p=preview

- http://nitrous.io/
  - Online Dev-Box (virtual machines with a web console)
Best Practices

- Warning highly subjective – constructive feedback welcome :)
Best Practices: Design and Refactoring

• Think in OOP!
  – Flexibility does not replace good software engineering.
  – OOP is actually restricting oneself to a well understood subset of possibilities

• Use a class framework
  – Prototypal inheritance is just cumbersome to write from scratch

• Getter and Setters?
  – I like them: Undefined function already on access

• Unless you are using JetBrain's Webstorm, think twice or thrice when designing an API
  – Avoid refactoring as much as possible
  – Refactoring in JavaScript sucks
Best Practices: Functions

Functions

- There is 2 ways to declare a function:
  - function myFunc() { … }
    - Run time
  - var myFunc = { … }
    - Parse time

- Detailed Differences:
  - See also:
Best Practices: Classes/Prototypes

- [http://www.phpied.com/3-ways-to-define-a-javascript-class/](http://www.phpied.com/3-ways-to-define-a-javascript-class/)

- There is prototypal inheritance, but ...

- Use a framework
  - I use [http://prototypejs.org/](http://prototypejs.org/), maybe there is a more popular one?
Best Practices: Namespaces

- Don't use a Namespace object
- Use requirejs
Best Practices: Arrays

- Never change the prototype
  - array.last

- Assume that someone else changed or might change the prototype of basic objects
  - Array.last: http://jsfiddle.net/L2UNg/

- Conclusion: Use http://underscorejs.org/ for dealing with objects and collections.
  - DON'T do it yourself, you are reinventing the wheel.
Best Practices: JSHint

- JavaScript code quality tool to detect potential errors and problems

- Fork from Douglas Crockford's JSLint
  - More configuration options to make it less hostile to your code :)


- `sudo npm -g install jshint`

- Put 'use strict' in your functions / modules
AngularJs

http://angularjs.org/
AngularJs: Trend

- http://www.google.com/trends/explore#q=angularjs
Overview

- Features
- Creating a Directive
Hello World in AngularJS

• Bad Example
  – [http://jsfiddle.net/zYHYS/](http://jsfiddle.net/zYHYS/)
  – Even official docs use(d?) this style, but it promotes bad style

• Good Example
  – [http://jsfiddle.net/fG9Pc/1/](http://jsfiddle.net/fG9Pc/1/)
Key Features

• Module System
  - angular.module(…)

• Dependency Injection
  - Angular.controller('MyCtrl', ['$scope', function($scope) { … }])

  Two Way Model-View Binding

• Custom Directives
  - <my-cool-widget></my-cool-widget>

• MVC
  - No “Model” class! Cost of “dirty checking”
Scope

- scope is an object that refers to the application model.
  - The scope references the model; the scope is NOT the model!
    - https://github.com/angular/angular.js/wiki/Understanding-Scopes

- It is an execution context for expressions.

- Scopes are arranged in hierarchical structure which mimic the DOM structure of the application.

- Scopes can watch expressions and propagate events.

- Source: http://docs.angularjs.org/guide/scope
Two Way Binding

- [http://jsfiddle.net/zYHYS/1/](http://jsfiddle.net/zYHYS/1/)
Custom Directives

- http://jsfiddle.net/CE9fa/1/
Angular @ Runtime

- $compile takes a String and returns a template function