Why JavaScript Frameworks?

- Complexity of managing DOM manipulation and data updates manually
- Well defined application architectures:
  - Model View Controller / Model View View Model / Model View Whatever
  - Binding of model and view: controllers, view models

Software Library

- Collection of implementations of behavior with a well-defined interface by which the behavior is invoked
- Reuse of behavior
- Modularity
- E.g., jQuery

https://en.wikipedia.org/wiki/Library_(computing)

Software Framework

- Abstraction in which software provides generic functionality that can be selectively changed by additional user-written code
- Universal, reusable environment that provides particular functionality as part of a larger software platform
- E.g., Angular, Ember, Backbone

https://en.wikipedia.org/wiki/Software_framework

Library vs Framework

- The following borrowed from AngularJS documentation makes the distinction clear:
  - **a library** - a collection of functions which are useful when writing web apps. Your code is in charge and it calls into the library when it sees fit. E.g., jQuery.
  - **frameworks** - a particular implementation of a web application, where your code fills in the details. The framework is in charge and it calls into your code when it needs something app specific. E.g., Angular, Ember, etc.

https://docs.angularjs.org/guide/introduction

Framework

- Hollywood Principle
  - Don’t call us, we’ll call you!
- Inversion of Control
- Imperative vs Declarative Programming

JavaScript Frameworks

- Single Page Application
  - Rich Internet Applications
- Model-View-Controller (MVC) / Model-View-ViewModel (MVVM) / Model-View-Whatever
  - Data binding, routing
- Scalable, Reusable, Maintanable JS code
- Test driven development

JavaScript Frameworks

- Angular
- Ember
- Backbone
- React
- Aurelia
- Meteor
- Polymer
- Knockout
- Vue
- Mercury