JavaScript Frameworks

Lecture 12

(Source: wikipedia, MIT OCW, and CS142 from Stanford University taught by Mendel Rosenblum)

Disclaimer

- The following represents my subjective view of the JavaScript frameworks landscape. It's still evolving, so some of what we learn may be out of date by the time you graduate ☺
- Unlike in the earlier lectures, we'll not have time to go in depth into any one technology or framework − we'll consider a broad overview
- Some of this is more "factual" than conceptual

JavaScript Frameworks

- jQuery
- Angular.js (also Backbone and Ember.js)
- React.js

jQuery

- One of the most widely used JavaScript frameworks (created by John Resig, circa '06)
- Include using <script src="jquery.js"></script>
- Makes it easier to navigate DOM, handle events and send AJAX messages
  - Does not attempt to superimpose higher level models on the application
  - Can interoperate with native JavaScript code as it's just another library (for the most part)
  - Provides cross-browser compatibility

Central Abstraction in jQuery

- $() to select DOM elements and/or handlers
- Can do whatever you want insider the $()

```
$(function () { // when the page has loaded
    $('img').on('click', function () {
        // handle click event on any img element
        …
    });
});
```

Cascading functions

- $ is just another JavaScript function, but it also returns the object on which it is invoked, so you can invoke further methods on it.

- This applies not only to the $ function, but to other jQuery functions as well as below:

```
$('div.test').add('p.quote').addClass('blue').slideDown('slow');
```

jQuery: Utility Functions

- Prefixed with a “$.” (dollar and dot)
- Examples: $.each, $.ajax

```
$.ajax({
  type: 'POST',
  url: '/process/submit.php',
  data: {
    name : 'John',
    location : 'Boston',
  },
}).done(function(msg) {
  alert('Data Saved: ' + msg);
}).fail(function(xmlHttpRequest, statusText,
errorThrown) {
  alert(‘Error’ + statusText);
}
```

jQuery: Closing thoughts

- Very useful framework for getting around some of JavaScript's ugliness in syntax
- But most of its popularity comes from cross-platform compatibility which is not really that important these days
- Design principles of Jquery can be incorporated without the whole shebang
  - Size and compatibility are other considerations

JavaScript Frameworks

- jQuery
- Angular.js (also Backbone and Ember.js)
- React.js

Angular.js

- Model-View-Controller framework for JavaScript (along with Ember.js, Backbone.js)

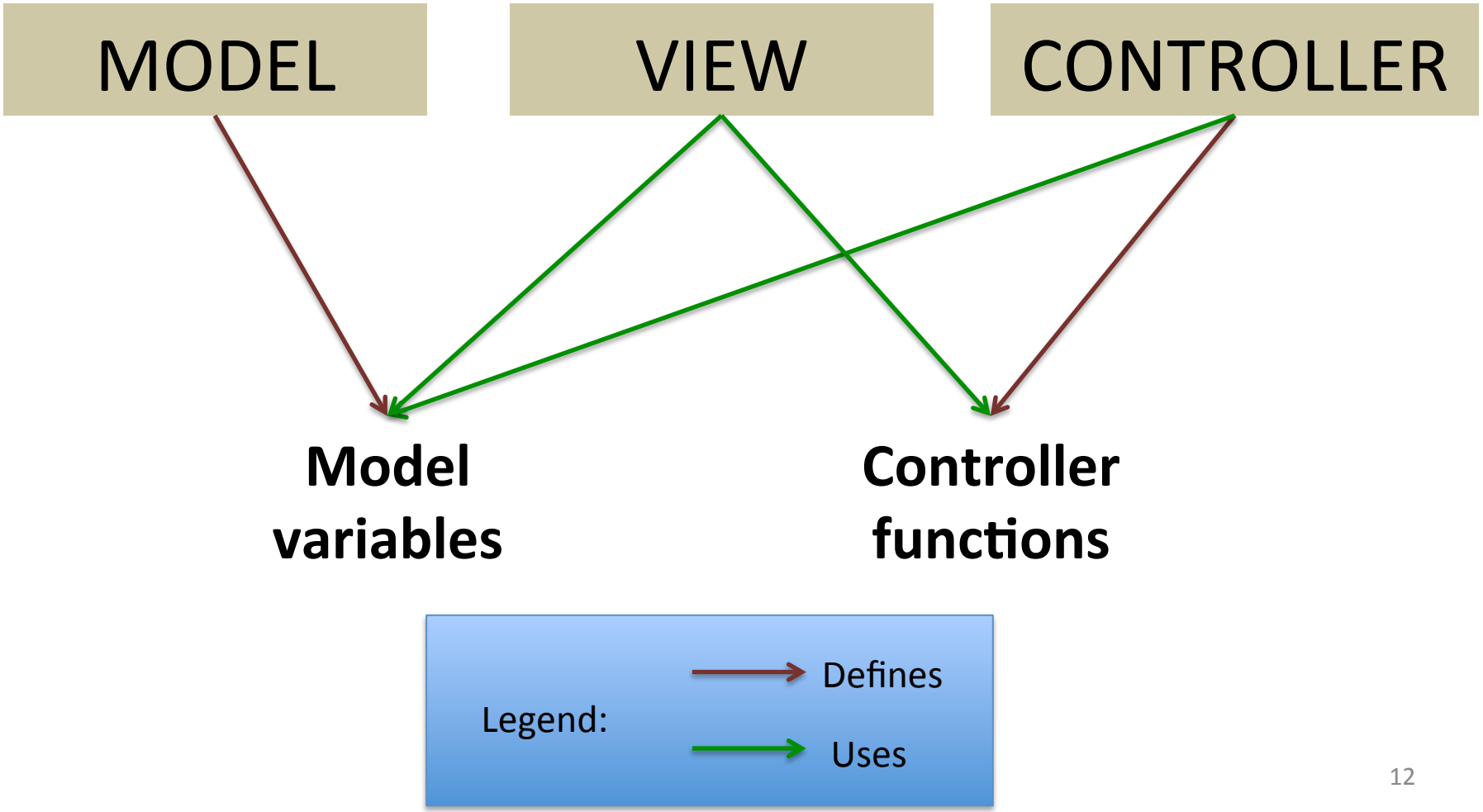
- Popularized by Google starting about 2010. Has overtaken many other rival frameworks

- Main idea: connect DOM to the JS code through a single declarative specification

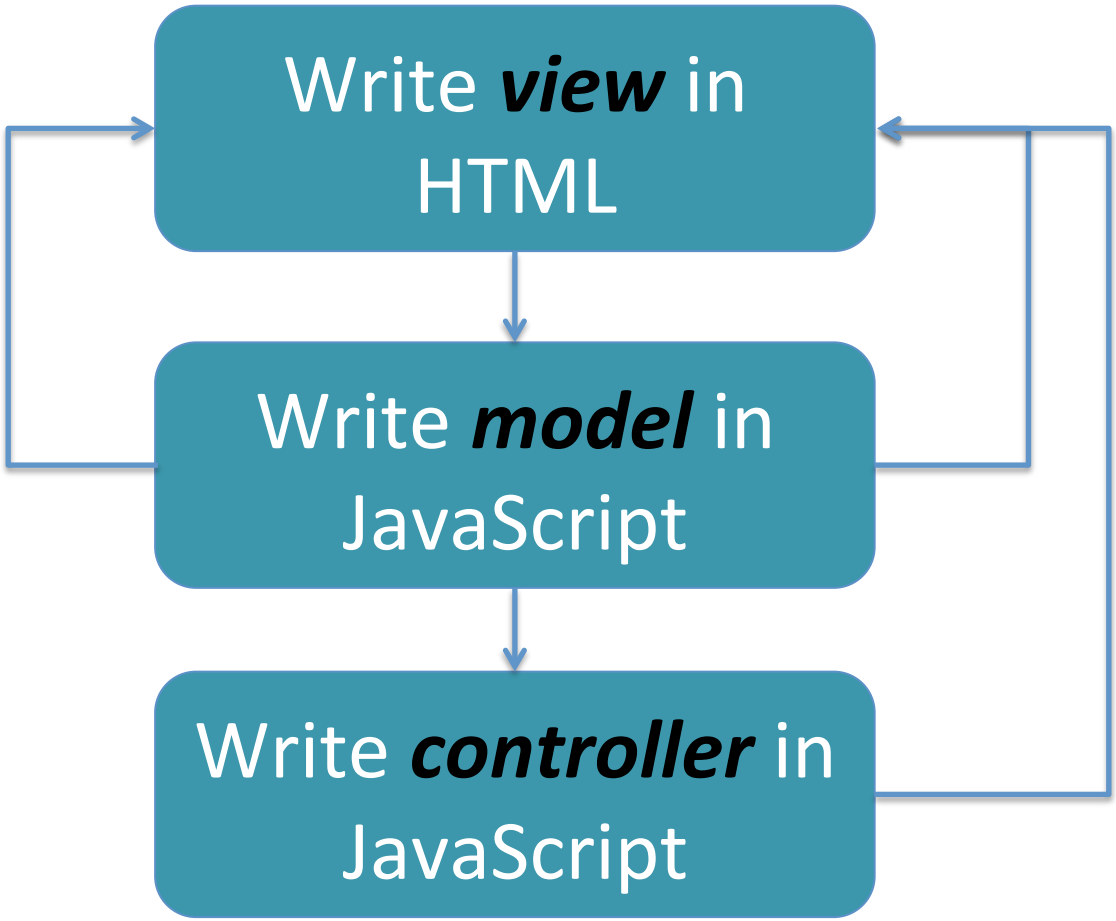
Model-View-Controller

MODEL	VIEW	CONTROLLER
Defines the data that will be used by the application. Data = ***model variables***	Template that defines *how* the data will be shown to the user	Defines functions that alter the data in model Functions = ***controller functions***

Model-View-Controller



AngularJS Workflow



Iterative process (it's not always immediately clear what model variables and controller functions to define)

Defining the View: Write a Template in HTML

```
<html ng-app="todoApp">
 <head>
  …
  <script type="text/javascript" src=”js/angular.js"></script>
  <script type="text/javascript" src="js/controller.js"></script>
 </head>
 <body ng-controller="TodoController">
  …
  <div>There are currently {{listSize}} items in the list</div>
  <div>
   <div ng-repeat="item in todoList">
    {{item}}
   </div>
  </div>
  <input type="text" ng-model="userInput" />
  <button>Add Item</button>
 </body>
</html>
```

Including the AngularJS Library

```
<html ng-app="todoApp">
 <head>
  …
  <script type="text/javascript" src=”js/angular.js"></script>
  <script type="text/javascript" src="js/controller.js"></script>
 </head>
 <body ng-controller="TodoController">
  …
  <div>There ar                    ze}} item
  <div>
   <div ng-repe                    ">
    {{item}}
   </div>
  </div>
  <input type="text" ng-model="userInput" />
  <button>Add Item</button>
 </body>
</html>
```

Option 1: Download AngularJS from angularjs.org and save into a file

Option 2: Type URL from Angular website directly

Indicating where the Angular Program is Rooted

```
<html ng-app="todoApp">
 <head>
  …
  <script type="text/javascript" src="js/angular.js"></script>
  <script type="text/javascript" src="todo.js"></script>
 </head>
 <body ng-controller="TodoController">
  …
  <div>{{listSize}} items</div>
  <div ng-repeat="item in todoList">
   {{item}}
  </div>
 </div>
 <input type="text" ng-model="userInput" />
 <button>Add Item</button>
 </body>
</html>
```

ng-app: A special Angular attribute that indicates where the Angular program is ***rooted***

Name of the application (will be defined in JS code later)

Indicating where the Angular Program is Rooted

```
<html ng-app="todoApp">
 <head>
  …
  <script type="text/javascript" src="js/angular.js"></script>
```

```
…
<div ng-app="otherApp">
    <div>
        Different Root
    </div>
</div>
…
```

Can place root in any tag

In this case, Angular program *only applies* to this tag and its descendants

```
  <button>Add Item</button>
 </body>
</html>
```

Indicating where the Angular Program is Rooted

```
<html ng-app="todoApp">
 <head>
  …
  <script type="text/javascript" src=”js/angular.js"></script>
  <script type="text/javascript" src="js/controller.js"></script>
 </head>
 <body ng-controller="TodoCo
  …
  <div>There are currently {                    list</div>
  <div>
   <div ng-repeat="item in t
    {{item}}
   </div>
  </div>
  <input type="text" ng-model="userInput" />
  <button>Add Item</button>
 </body>
</html>
```

ng-app in HTML tag means we want *entire* program to be Angular

ng-repeat

```
<html ng-app="todoApp">
 <head>
  …
  <scri                                t" src="
  <scri                                t" src="                    ipt>
 </head
 <body                                roller">
  …
  <div>There are currently {{listSize}} items in the list</div>
  <div>
   <div ng-repeat="item in todoList">
    {{item}}
   </div>
  </div>
  <input type="text" ng-model="userInput" />
  <button>Add Item</button>
 </body>
</html>
```

`ng-repeat`: A special Angular attribute (more on this later…)

Creates this div element for *each* item in the todoList.

Two-Way Data Binding: Curly Braces

```
<html ng-app="todoApp">
 <head>
  …
  <script type="text/javascript" src=”js/angular.js"></script>
  <script type="text/javascript" src="js/controller.js"></script>
 </head>
 <body ng-controller="TodoController">
  …
  <div>There are currently {{listSize}} items in the list</div>
  <div>
   <div ng-repeat="item in todoList">
    {{item}}
   </div>
  </div>
  <input type="text" ng-model="userInput" />
  <button>Add Item</button>
 </body>
</html>
```

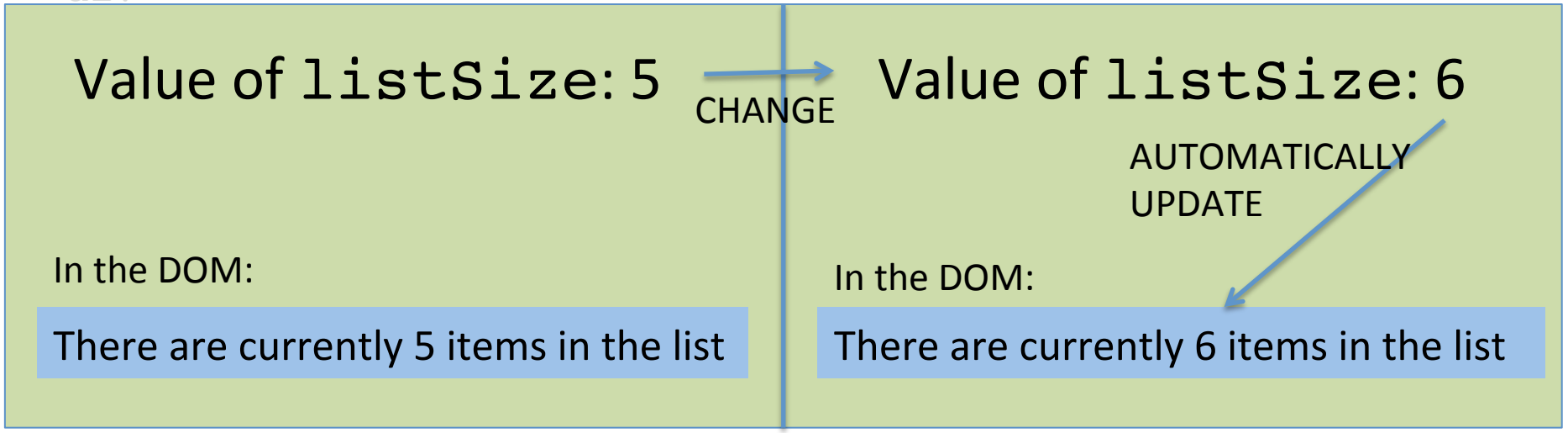
Two-Way Data Binding: Curly Braces

```
<html ng-app="todoApp">
 <head>
  …
  <script type="text/javascript" src=”js/angular.js"></script>
  <script type="text/javascript" src="js/controller.js"></script>
 </head>
 <body ng-controller="TodoController">
  …
  <div>There are currently {{listSize}} items in the list</div>
  <div>
   <div ng-repeat="item in todoList">
    {{item}}
   </div>
  </div>
  <input type="text" ng-model="userInpu
  <button>Add Item</button>
 </body>
</html>
```

e.g., evaluates value of `listSize` whenever changes are made to that variable

Two-Way Data Binding: Curly Braces

```
<html ng-app="todoApp">
 <head>
  …
  <script type="text/javascript" src=”js/angular.js"></script>
  <script type="text/javascript" src="js/controller.js"></script>
 </head>
 <body ng-controller="TodoController">
  …
  <div>There are currently {{listSize}} items in the list</div>
  <div>
```




```
</html>
```

Two-Way Data Binding: Curly Braces

```
<html ng-app="todoApp">
 <head>
  …
  <script type="text/javascript" src=”js/angular.js"></script>
  <script type="text/javascript" src="js/controller.js"></script>
 </head>
 <body ng-controller="TodoController">
  …
  <div>There are currently {{listSize}} items in the list</div>
  <div>
```

Two-Way Data Binding: Angular automatically updates the DOM whenever model variables change

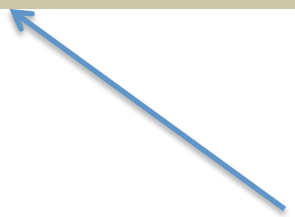
DOM is abstracted out from developer ➔ No DOM interactions!

```
</html>
```

Two-Way Data Binding: ng-model

```
<html ng-app="todoApp">
 <head>
  …
  <script type="text/javascript" src="js/angular.js"></script>
  <script type="text/javascript" src="js/controller.js"></script>
 </head>
 <body ng-controller="TodoController">
  …
  <div>Th                                    ms in the list</div>
  <div>
   <div r
    {{item}}
   </div>
  </div>
  <input type="text" ng-model="userInput" />
  <button>Add Item</button>
 </body>
</html>
```

`ng-model`: Value of `userInput` is bound to the value of the text input element



Two-Way Data Binding: ng-model

Text input element

Wake up

Value of `userInput` : Wake up

```
  </div>
  <input type="text" ng-model="userInput" />
  <button>Add Item</button>
 </body>
</html>
```

ng-controller

```
<html ng-app="todoApp">
 <head>
  …
  <script type="text/javascrip
  <script type="text/javascript" src="js/controller.js"></script>
 </head>
 <body ng-controller="TodoController">
  …
  <div>There are currently {{listSize}} items in the list</div>
  <div>
   <div ng
    {{item
   </div>
  </div>
  <input t
  <button>Add Item</button>
 </body>
</html>
```

Name shared by the model and controller (in this case, `TodoController`)

`ng-controller`: Indicates which HTML code portion the model and controller correspond to

Defining the Model: Setting Up the Module

```
var todoApp = angular.module("todoApp", []);
```

Creates the Angular module. This function takes two parameters.

Parameter 1: Name of Angular app (should be same as ng-app value!)

Parameter 2: Empty array (to indicate we're *creating* a new module)

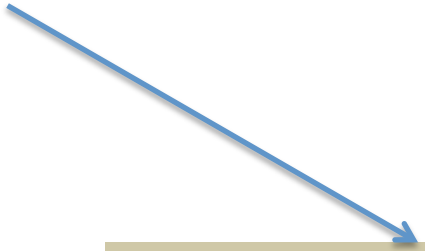
The model is defined in the JavaScript code

Setting up .controller()

```
var todoApp = angular.module("todoApp", []);

todoApp.controller('TodoController', function($scope) {

});
```

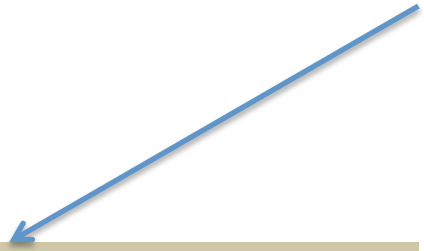


`.controller()` defines both the model and the controller

Setting up .controller()

```
var todoApp = angular.module("todoApp", []);

todoApp.controller('TodoController', function($scope) {
```



Parameter 1: Identifier shared by both controller and model (remember ng-controller?)

Parameter 2: Anonymous function with the *scope object* ($scope) as its argument.

```
});
```

We will define the model variables and controller functions using ***$scope***

Setting up .controller()

```
var todoApp = angular.module("todoApp", []);

todoApp.controller('TodoController', function($scope) {

});
```

Model variables and controller functions will be defined here using $scope

Initialize the Model Variables

```
var todoApp = angular.module("todoApp", []);

todoApp.controller('TodoController', function($scope) {









});
```

Initialize the Model Variables

```
var todoApp = angular.module("todoApp", []);

todoApp.controller('TodoController', function($scope) {
    $scope
```

Take the scope object

```
});
```

Initialize the Model Variables

```
var todoApp = angular.module("todoApp", []);

todoApp.controller('TodoController', function($scope) {
    $scope.listSize
});
```

Take the scope object

Write name of model variable to define (eg, `listSize`)

Initialize the Model Variables

```
var todoApp = angular.module("todoApp", []);

todoApp.controller('TodoController', function($scope) {
    $scope.listSize = 0;
});
```

Take the scope object

Write name of model variable to define (eg, `listSize`)

Initialize model variable with an assignment

Initialize the Model Variables

```
var todoApp = angular.module("todoApp", []);

todoApp.controller('TodoController', function($scope) {
    $scope.listSize = 0;
    $scope.userInput = “”;




});
```

Initialize the Model Variables

```
var todoApp = angular.module("todoApp", []);

todoApp.controller('TodoController', function($scope) {
    $scope.listSize = 0;
    $scope.userInput = “”;
    $scope.todoList = [];




});
```

Initialize the Model Variables

```
var todoApp = angular.module("todoApp", []);

todoApp.controller('TodoController', function($scope) {
    $scope.listSize = 0;
    $scope.userInput = “”;
    $scope.todoList = [];

});
```

These three lines define the *entire* model!

Defining the Controller: Write the Controller Functions

```
var todoApp = angular.module("todoApp", []);

todoApp.controller('TodoController', function($scope) {
    $scope.listSize = 0;
    $scope.userInput = “”;
    $scope.todoList = [];
```

Defining the controller functions is similar to defining model variables, but ***right-hand side of assignment is a function***

```
});
```

Defining the Controller: Write the Controller Functions

```
var todoApp = angular.module("todoApp", []);

todoApp.controller('TodoController', function($scope) {
    $scope.listSize = 0;
    $scope.userInput = “”;
    $scope.todoList = [];

    $scope.addItem =



});
```

Defining the Controller: Write the Controller Functions

```
var todoApp = angular.module("todoApp", []);

todoApp.controller('TodoController', function($scope) {
    $scope.listSize = 0;
    $scope.userInput = “”;
    $scope.todoList = [];

    $scope.addItem = function() {
        Modify model variables’ values here,
        where appropriate
    };
});
```

No need to manipulate the DOM!

Defining the Controller: Write the Controller Functions

```
var todoApp = angular.module("todoApp", []);

todoApp.controller('TodoController', function($scope) {
    $scope.listSize = 0;
    $scope.userInput = “”;
    $scope.todoList = [];

    $scope.addItem = function() {
        $scope.todoList.push($scope.userInput);

    };
});
```

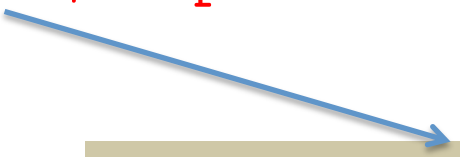
Push the value of the current user input to the todo list

Defining the Controller: Write the Controller Functions

```
var todoApp = angular.module("todoApp", []);

todoApp.controller('TodoController', function($scope) {
    $scope.listSize = 0;
    $scope.userInput = “”;
    $scope.todoList = [];

    $scope.addItem = function() {
        $scope.todoList.push($scope.userInput);
        $scope.listSize = $scope.listSize + 1;
    };
});
```



Increment the list size by 1

How to Add Controller Function to View?

```
…
<button ng-click="addItem()">Add Item</button>
…
```

```
$scope.addItem = function() {
    $scope.todoList.push($scope.userInput);
    $scope.listSize = $scope.listSize + 1;
};
});
```

Other attributes: `ng-keypress`, `ng-mouseenter`, etc.

The `ng-repeat` Attribute

```
<div ng-repeat="item in todoList">
    {{item}}
</div>
```

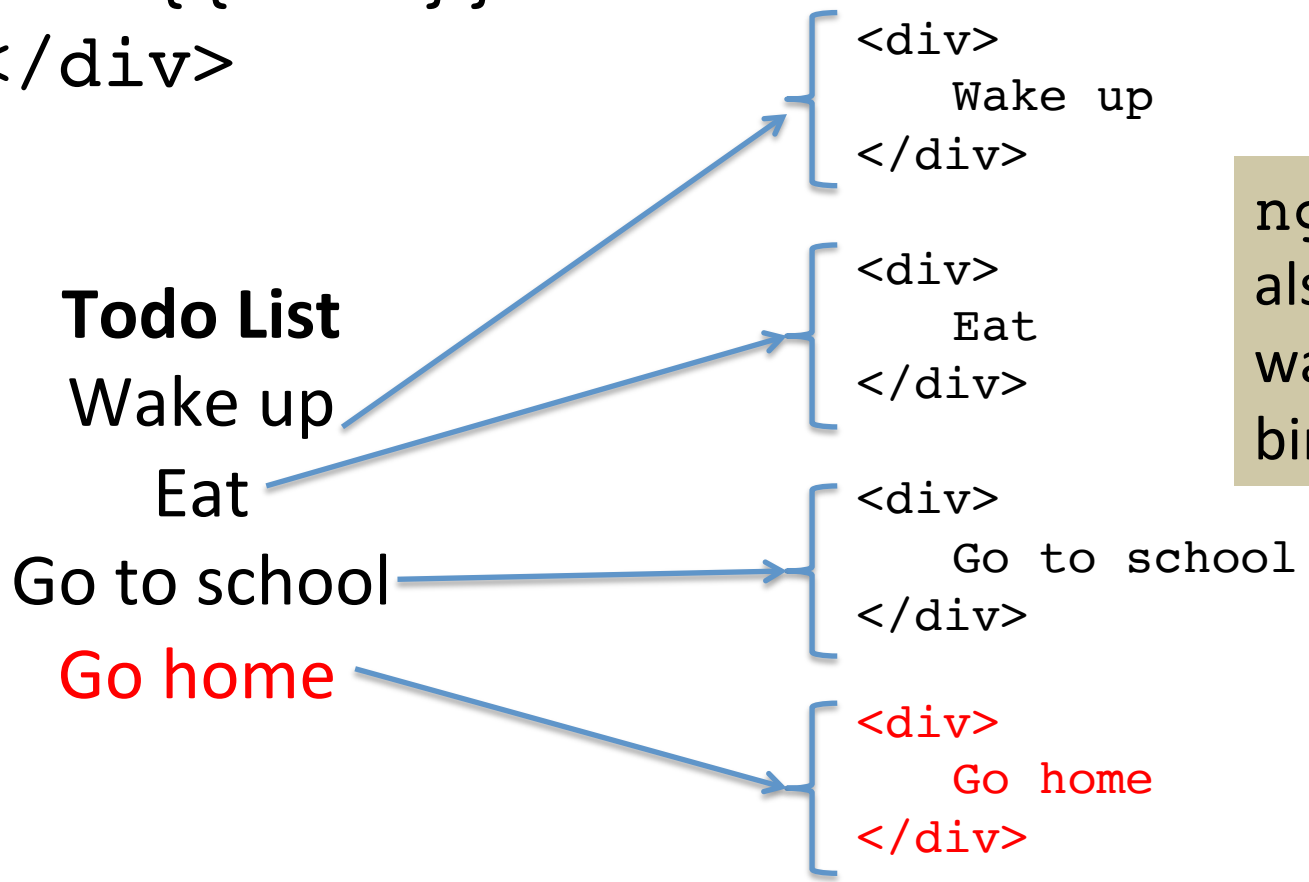
Value of `ng-repeat` is of the form **<alias> in <list>**

Name the programmer wants to use to refer to each item in list

The model variable containing the list

The ng-repeat Attribute

```
<div ng-repeat="item in todoList">
    {{item}}
</div>
```



ng-repeat also uses two-way data binding!

Angular.js drawbacks

- DOM access and update overhead is very high
- There's only one way of doing it − the Angular way. Interoperability with vanilla JS is poor.
- Large library to download for simple stuff
  - Cant' use just a subset of the framework
  - Cognitive overhead to learn new framework

JavaScript Frameworks

- jquery
- Angular.js (also Backbone and Ember.js)
- React.js

React.js: From Facebook

- JavaScript framework - Does view component only
- ● View declared in JavaScript (more accurately a lang translated to JavaScript)
  - ○ Angular: HTML with JavaScript embedded
  - ○ React: JavaScript with HTML embedded
- ● Basic building block: Components
  - Have a function render() that returns HTML-like structure
  - Accepts inputs (this.props)
  - Have an internal state (this.state)

- Components are reusable pieces composed to form vie

React: Virtual DOM

- React component render() functions results are places in a Virtual DOM
- ○ Highly optimized one-way binding process
  - Only components whose this.props or this.state change are updated
  - Much faster access than the real DOM
- React efficiently pushes the Virtual DOM to the Browser's DOM
- Only the parts of the Browser's DOM that change are updated

React.js

```
var CommentBox = React.createClass({displayName: 'CommentBox',
  render: function() {
    return (
      React.createElement('div', {className: "commentBox"},
        "Hello, world! I am a CommentBox."
      )
    );
  }
});
ReactDOM.render(
  React.createElement(CommentBox, null),
  document.getElementById('content')
);
```

React.js: JSX

```
var CommentBox = React.createClass({
  render: function() {
    return (
      <div className="commentBox">
        Hello, world! I am a CommentBox.
      </div>
    );
  }
});
ReactDOM.render(
  <CommentBox />,
  document.getElementById('content')
);
```

React.js: Pros and Cons

- Pros
  - Forces declarative style using JSX
  - Virtual DOM makes DOM manipulation cleaner
  - Faster than Angular.JS as it's just a library

- Cons
  - High cognitive cost in learning it
  - Doesn't impose a discipline on programmers
  - Doesn't play well with vanilla HTML

JavaScript Frameworks

- jquery
- Angular.js (also Backbone and Ember.js)
- React.js