* A **module** contains the different components of an AngularJS app
* A **controller** manages the app's data
* An **expression** displays values on the page
* A **filter** formats the value of an expression

1. A user visits the AngularJS app.
2. The *view* presents the app's data through the use of *expressions*, *filters*, and *directives*. Directives bind new behavior HTML elements.
3. A user clicks an element in the view. If the element has a directive, AngularJS runs the function.
4. The function in the *controller* updates the state of the data.
5. The view automatically changes and displays the updated data. The page doesn't need to reload at any point.

Then, in **index.html** we added <body ng-app="myApp">. The ng-app is called a *directive*. It tells AngularJS that the myApp module will live within the <body> element, termed the application's *scope*. In other words, we used the ng-app directive to define the application scope.

So far this is our typical workflow when making an AngularJS app:

1. Create a module, and use ng-app in the view to define the application scope.
2. Create a controller, and use ng-controller in the view to define the controller scope.
3. Add data to $scope in the controller so they can be displayed with expressions in the view. any properties attached to $scope become available to use in the view.
4. AngularJS gets the value of product.price.
5. It sends this number into the currency filter. The pipe symbol (|) takes the output on the left and "pipes" it to the right.
6. The filter outputs a formatted currency with the dollar sign and the correct decimal places.

Then in the view, we added <div ng-repeat="product in products">. Like ng-app and ng-controller, the ng-repeat is a directive. It loops through an array and displays each element.

In the view inside <div class="col-md-6">, replace

<img src="img/the-book-of-trees.jpg">

with

<img ng-src="{{ product.cover }}">

The ng-src is a directive that sets the <img>element's src to a property in the controller.

We've used a few directives so far - ng-app, ng-controller, ng-repeat, and ng-src. What can we generalize about directives?

Directives bind behavior to HTML elements. When the app runs, AngularJS walks through each HTML element looking for directives. When it finds one, AngularJS triggers that behavior (like attaching a scope or looping through an array).

In the view modify <p class="likes"> to look like this:

<p class="likes" ng-click="plusOne($index)">

1. ng-click is a directive. When <p class="likes"> is clicked, ng-click tells AngularJS to run the plusOne() function in the controller.
2. The plusOne() function gets the index of the product that was clicked, and then adds one to that product's likes property.

Notice that the plusOne() doesn't interact with the view at all; it just updates the controller. Any change made to the controller shows up in the view.

<div class="rating">

<p class="likes" ng-click="plusOne($index)">+ {{ product.likes }} </p>

<p class="dislikes" ng-click="minusOne($index)">- {{ product.dislikes }} </p>

</div>

$scope.plusOne = function(index) {

$scope.products[index].likes += 1;

};

$scope.minusOne = function(index) {

$scope.products[index].dislikes += 1;

};

**Reusability.** Directives let you create self-contained units of functionality. We could easily plug in this directive into another AngularJS app and avoid writing a lot of repetitive HTML.

We can use Angular's built-in directives together with custom directives to create more readable apps.

<div class="card" ng-repeat="app in apps">

<app-info info="app"></app-info>

</div>

Directives are a powerful way to create self-contained, interactive components. Unlike jQuery which adds interactivity as a layer on top of HTML, AngularJS treats interactivity as a native component of HTML.

But what happens when the data contains hundreds of items, or if it's constantly changing like weather or financial data? Hardcoding data into a controller won't work anymore.

A better solution is to read the live data from a server. We can do this by creating a service.

But what happens when the app grows and needs to display more information? Stuffing more code to a single view will quickly make things messy.

A better solution is to use multiple templates that display different pieces of data based on the URL that the user is visiting. We can do this with Angular's application routes.

Why are routes useful? Instead of filling a single view with more code than needed, routes let us map URLs to self-contained controllers and templates.

* Directives are a way to make standalone UI components, like <app-info>
* Services are a way to make standalone communication logic, like forecast which fetches weather data from a server
* Routes are a way to manage apps containing more views.