Components & Component-based Architecture

Components only control their own View and Data

- Never modify data or DOM outside their own scope
- Modifying creates side-effects that lead to chaos
- Therefore, Angular components always use isolate scope

Components have well-defined public API – Inputs and Outputs

- Inputs: use ‘<‘ and ‘@’ bindings only
- Never change property of passed in object or array
- Outputs: use ‘&’ for component event callbacks
- Pass data to callback through param map { key: val }

Components have well-defined lifecycle

- $onInit – controller initialization code
- $onChanges(changeObj) – called whenever one-way bindings are updated
  - changeObj.currentValue, changeObj.previousValue
- $postLink – similar to ‘link’ in directive
- $onDestroy – when scope is about to be destroyed

Application is a tree of components

- Entire application should be comprised of components
- Each one would have a well-defined input and output
- 2-way data binding is minimized as much as possible

Step 1: Register Component With Module

```
angular.module('App', [])
.component('myComponent', {
  templateUrl: 'template.html',
  controller: CompController,
  bindings: {
    prop1: '<',
    prop2: '@',
    onAction: '&'
  }
});
```

Normalized form.
In HTML,
use my-component

Step 1: Register Component With Module

```
angular.module('App', [])
.component('myComponent', {
  templateUrl: 'template.html',
  controller: CompController,
  bindings: {
    prop1: '<',
    prop2: '@',
    onAction: '&'
  }
});
```

Simple config object.
NOT a function.

Step 2: Configure Component

```
angular.module('App', [])
.component('myComponent', {
  templateUrl: 'template.html',
  controller: CompController,
  bindings: {
    prop1: '<',
    prop2: '@',
    onAction: '&'
  }
});
```

Almost always have a template

Step 2: Configure Component

```
angular.module('App', [])
.component('myComponent', {
  templateUrl: 'template.html',
  controller: CompController,
  bindings: {
    prop1: '<',
    prop2: '@',
    onAction: '&'
  }
});
```

Placed on scope with label '$ctrl'

Not required. Empty function provided automatically

Step 2: Configure Component

```
angular.module('App', [])
.component('myComponent', {
  templateUrl: 'template.html',
  controller: CompController,
  bindings: {
    prop1: '<',
    prop2: '@',
    onAction: '&'
  }
});
```

'bindings' object is the isolate scope param mapping definition

Step 3: Reference Props in Template

```
<div
  ng-click="$ctrl.onAction({myArg: 'val'})">
  {{ $ctrl.prop1.prop }} and {{ $ctrl.prop2 }}
</div>
```

Step 4: Use Component in HTML

```
<my-component
  prop1="val-1"
  prop2="@parentProp"
  on-action="parentFunction(myArg)">

  {{ $ctrl.prop1.prop }} and {{ $ctrl.prop2 }}

</my-component>
```

Summary

- Components encourage component-based architecture
  - But they don't enforce it 100%, so we must follow conventions
- Components should never modify data or DOM that doesn't belong to them
  - That's why it always has isolate scope and well-defined API
- Register component with .component('name', configObj)
- Provide controller only if you are adding extra functionality
  - Otherwise, Angular already provides an empty function for us