Initialization

- One should try and place the <script> tag including the angular.js and related angular scripts at the bottom of the page to improve the app load time. This is because the HTML loading would then not be blocked by loading of angular.js and related scripts.

Expressions

- Complex javascript code should be made as a method in the controller (added as a method to the $scope) and then, should be called from the view. This is unlike putting the complex Javascript code as Angular expressions, right, in the view.

Controllers

- With real applications, one should avoid creating controllers in the global scope. Rather, one should use ".controller" method of the Angular Module to work with $scope object. Take a look at the sample code below illustrating this point:

Controller defined in the Global Scope:

```
function HelloController( $scope ) {
$scope.name = "Guest";
}
```

Controller defined using ".controller" method **(Recommended way)**

```
var helloApp = angular.module( "helloApp", [] );
helloApp.controller( "HelloController", function($scope){
$scope.name = "Guest";
} );
```

- The controllers should only be used to setup initial state of the $scope object and add one or more behaviour to this object. One should avoid using controllers to do some of the following:
  - Manipulate DOM,
  - Format input,
  - Filter output,
  - Share code or state across different controllers
  - Manage the life-cycle of other components (for example, to create service instances)
- A controller should contain only the business logic needed for a single view. The functionality should rather be moved to services and these services should be injected into the controllers using dependency injection.
- The recommended way to declare the controller function is to use the array notation such as following because it protects against minification.

Controller instantiation without array notation

```
var helloApp = angular.module( "helloApp", [] );
helloApp.controller( "HelloController", function($scope){
```

```
$scope.name = "Guest";
} );
```

Controller instantiation with array notation (**Recommended way**)

```
var helloApp = angular.module( "helloApp", [] );
helloApp.controller( "HelloController", ['$scope', function($scope){
$scope.name = "Guest";
}]);
```

- While writing **unit tests for controllers**, one of the recommended ways is to **inject $rootScope & $controller**. Take a look at the sample unit tests on this page: http://hello-angularjs.appspot.com/angularjs-unit-test-code-example-1. The following is the sample code representing injection of $rootScope and $controller objects.

  ```
  beforeEach(
  inject(
  function( $rootScope, $controller ){
  scopeMock = $rootScope.$new();
  $controller( 'CompanyCtrl', {$scope: scopeMock} );
  }
  )
  );
  ```

Directives

- One should prefer using the dash-delimited format (e.g. ng-model for ngModel). While working with an HTML validating tool, one could instead use the data-prefixed version (e.g. data-ng-model for ngModel).
- Prefer using directives via tag name and attributes over comment and class names. Doing so generally makes it easier to determine what directives a given element matches.
- While creating directives, it is recommended to prefix your own directive names to avoid collisions with future standard.

Template

- With large templates (HTML content with Angular directives) within an HTML file, it is recommended to break it apart into its own HTML file and load it with the templateUrl option.

Dependency Injection

The preferred way of injecting the dependencies is by passing the dependency to the constructor function rather than using one of the following other ways. In this way, the responsibility of creating the dependency object lies with other objects or function. This is straight forward for those who have worked with one or more dependency injection framework in the past. However, this could be useful information for the beginners. Following are other ways of doing dependency injection in Angular:

- Create it using the new operator.

- Look for it in a well-known place, also known as a global singleton.
- Ask a registry (also known as service registry) for it.

Unit Testing

- As mentioned above, one should try and avoid using controller methods for DOM manipulation. That would bring views code inside the methods and make it difficult to test.
- One may want to use Jasmine/Karma combination for doing controller methods testing.
- Protractor framework can be used for E2E testing, as recommended. Read more on Angular page for E2E testing.

Fuente: https://dzone.com/articles/angularjs-coding-best