wifi: actioninnovationcenter
pass: demianborba

ANGULAR BOOTCAMP

@jornadaadobe

OS SERVIÇOS DO ANGULAR

service

$anchorScroll
$animate
$cacheFactory
$compile
$controller
$document
$exceptionHandler
$filter
$http
$httpBackend
$interpolate
$interval
$locale
$location
$log
$parse
$q
$rootElement
$rootScope
$sce
$sceDelegate
$templateCache
$templateRequest
$timeout
$window

Angular services are substitutable objects that are wired together using dependency injection (DI). You can use services to organize and share code across your app.

Angular services are:

- Lazily instantiated – Angular only instantiates a service when an application component depends on it.
- Singletons – Each component dependent on a service gets a reference to the single instance generated by the service factory.

Angular offers several useful services (like `$http`), but for most applications you'll also want to create your own.

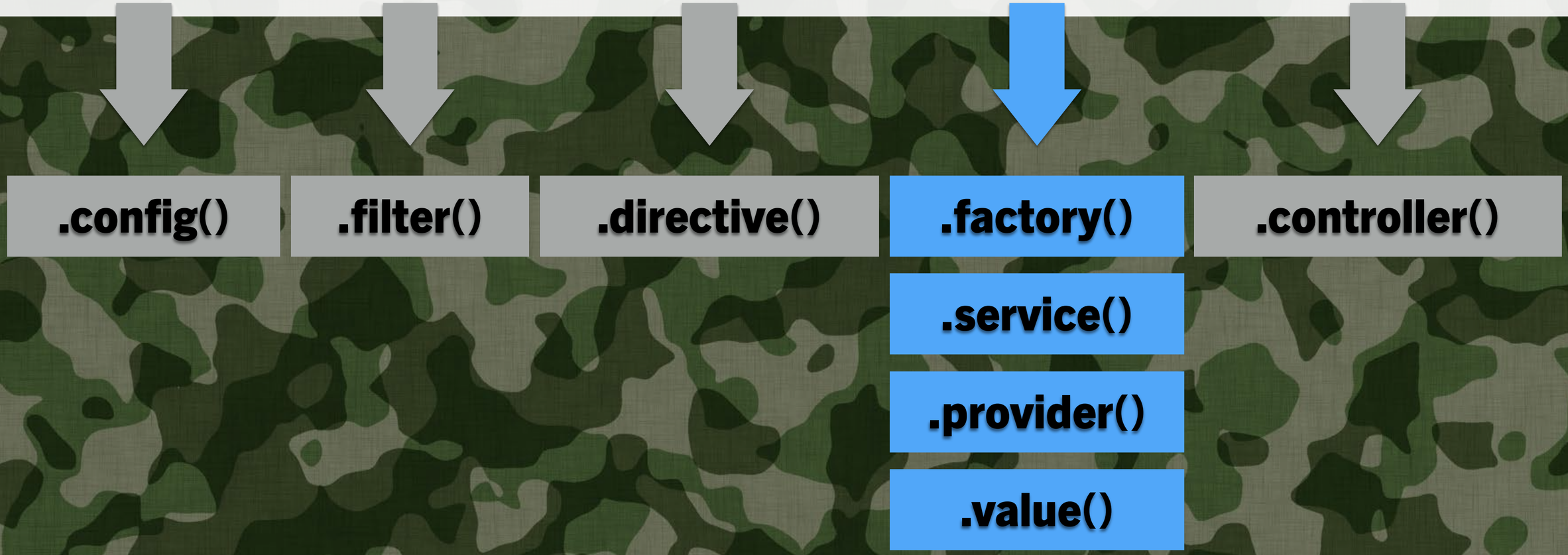
> **Note:** Like other core Angular identifiers, built-in services always start with `$` (e.g. `$http`).

CUSTOM SERVICES

- .factory()
- .service()
- .provider()
- .value()

MODULE

```
angular.module('myApp', []);
```



```
'use strict';

angular.module('wannaBuy')
  .factory('wbItems', function() {

    var items = [{name:'A'},{name:'B'},{name:'C'}];

    function getItems() {
      return items;
    };

    return {
      getItems: getItems
    };
  });
```

#exercício

CRIANDO UM CUSTOM SERVICE

DATA FOR CONTROLLERS

COMPREENDENDO O $broadcast

```
// firing an event downwards
$scope.$broadcast('myCustomEvent', {
  someProp: 'Sending you an Object!' // send whatever you want
});

// listen for the event in the relevant $scope
$scope.$on('myCustomEvent', function (event, data) {
  console.log(data); // 'Data to send'
});
```

#exercício

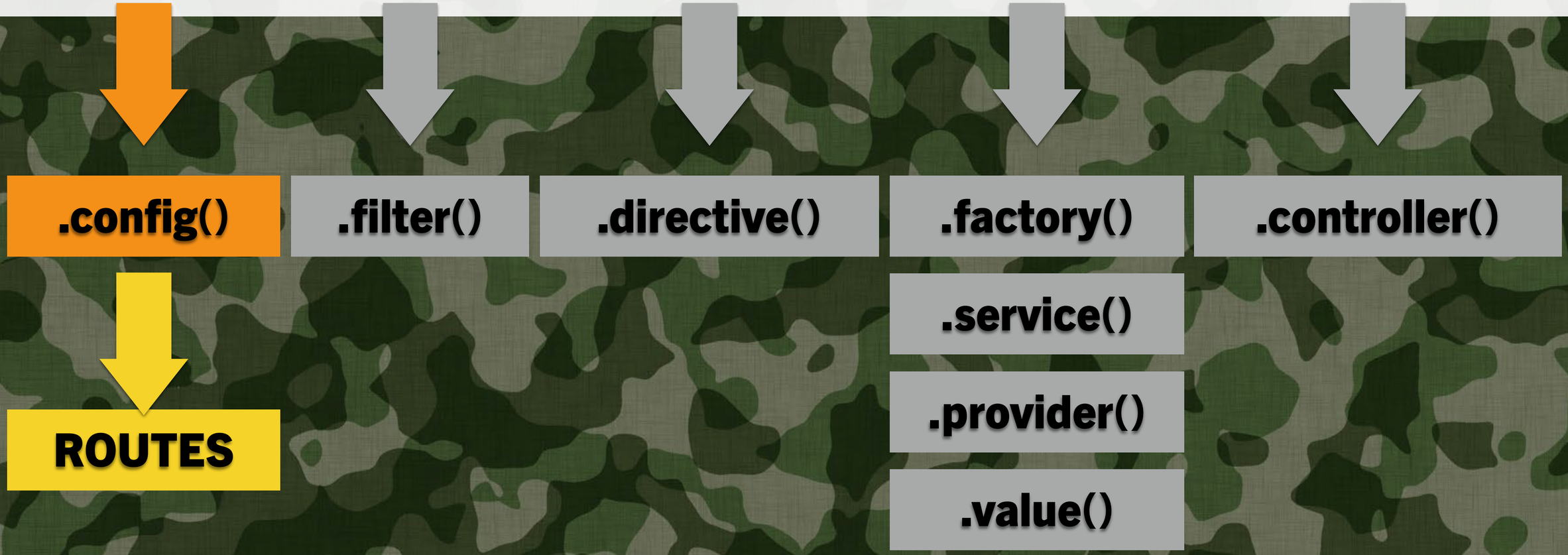
DISPARANDO EVENTOS

$broadcast

ROTEAMENTO NO ANGULAR

MODULE

```
angular.module('myApp', []);
```



```
angular.module('routingApp', ['ngRoute'])
  .config(['$routeProvider', function ($routeProvider) {
    $routeProvider
      .when('/pagina1', {
        controller: 'Pagina1Ctrl',
        templateUrl: 'partials/pagina1.html'
      })
      .when('/pagina1/:nome', {
        controller: 'Pagina1Ctrl',
        templateUrl: 'partials/pagina1.html'
      })
      .when('/pagina2', {
        controller: 'Pagina2Ctrl',
        templateUrl: 'partials/pagina2.html'
      })
      .when('/pagina3', {
        controller: 'Pagina3Ctrl',
        templateUrl: 'partials/pagina3.html'
      })
      .otherwise('/pagina1');
  }]);
```

#exercício

CRIANDO UM APP COM ROTAS

ROUTING

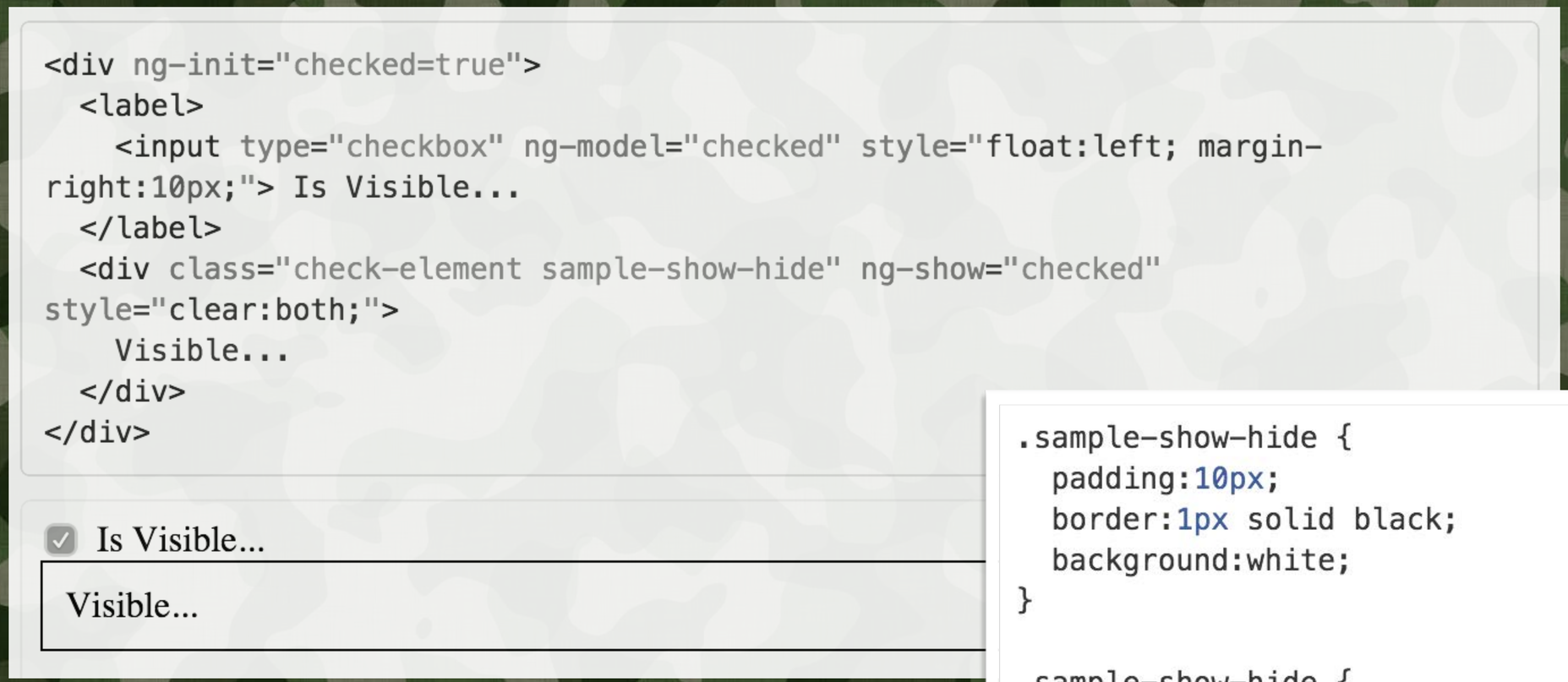
ANIMANDO ITENS COM O ngAnimate

```
<script src="angular.js">
<script src="angular-animate.js">
```

```
bower install angular-animate@X.Y.Z
```

```
angular.module('app', ['ngAnimate']);
```

```
<div ng-init="checked=true">
  <label>
    <input type="checkbox" ng-model="checked" style="float:left; margin-
right:10px;"> Is Visible...
  </label>
  <div class="check-element sample-show-hide" ng-show="checked"
style="clear:both;">
    Visible...
  </div>
</div>
```




```
.sample-show-hide {
  padding:10px;
  border:1px solid black;
  background:white;
}

.sample-show-hide {
  -webkit-transition:all linear 0.5s;
  transition:all linear 0.5s;
}

.sample-show-hide.ng-hide {
  opacity:0;
}
```

```
<p>
  <input type="button" value="set" ng-click="myCssVar='css-class'">
  <input type="button" value="clear" ng-click="myCssVar=''">
  <br>
  <span ng-class="myCssVar">CSS-Animated Text</span>
</p>
```

set clear

CSS-Animated Text

```
.css-class-add, .css-class-remove {
  -webkit-transition:all cubic-bezier(0.250, 0.460, 0.450, 0.940) 0.5s;
  -moz-transition:all cubic-bezier(0.250, 0.460, 0.450, 0.940) 0.5s;
  -o-transition:all cubic-bezier(0.250, 0.460, 0.450, 0.940) 0.5s;
  transition:all cubic-bezier(0.250, 0.460, 0.450, 0.940) 0.5s;
}

.css-class,
.css-class-add.css-class-add-active {
  color: red;
  font-size:3em;
}

.css-class-remove.css-class-remove-active {
  font-size:1.0em;
  color:black;
}
```

set clear

CSS-Animated T

```
<div ng-repeat="item in items" class="repeated-item">
  {{ item.id }}
</div>
```

```
/*
  We're using CSS transitions for when
  the enter and move events are triggered
  for the element that has the .repeated-item
  class
*/
.repeated-item.ng-enter, .repeated-item.ng-move {
  -webkit-transition:0.5s linear all;
  -moz-transition:0.5s linear all;
  -o-transition:0.5s linear all;
  transition:0.5s linear all;
  opacity:0;
}

/*
 The ng-enter-active and ng-move-active
 are where the transition destination properties
 are set so that the animation knows what to
 animate.
*/
.repeated-item.ng-enter.ng-enter-active,
.repeated-item.ng-move.ng-move-active {
  opacity:1;
}
```

```
/*
  We're using CSS keyframe animations for when
  the leave event is triggered for the element
  that has the .repeated-item class
*/
.repeated-item.ng-leave {
  -webkit-animation:0.5s my_animation;
  -moz-animation:0.5s my_animation;
  -o-animation:0.5s my_animation;
  animation:0.5s my_animation;
}

@keyframes my_animation {
  from { opacity:1; }
  to { opacity:0; }
}

/*
  Unfortunately each browser vendor requires
  its own definition of keyframe animation code...
*/
@-webkit-keyframes my_animation {
  from { opacity:1; }
  to { opacity:0; }
}

@-moz-keyframes my_animation {
  from { opacity:1; }
  to { opacity:0; }
}

@-o-keyframes my_animation {
  from { opacity:1; }
  to { opacity:0; }
}
```

Which directives support animations?

A handful of common AngularJS directives support and trigger animation hooks whenever any major event occurs during its life cycle. The table below explains in detail which animation events are triggered

Directive	Supported Animations
ngRepeat	enter, leave, and move
ngView	enter and leave
ngInclude	enter and leave
ngSwitch	enter and leave
ngIf	enter and leave
ngClass or	add and remove
ngShow & ngHide	add and remove (the ng-hide class value)

CSS Staggering Animations

A Staggering animation is a collection of animations that are issued with a slight delay in between each successive operation resulting in a curtain-like effect. The ngAnimate module (versions >=1.2) supports staggering animations and the stagger effect can be performed by creating a **ng-EVENT-stagger** CSS class and attaching that class to the base CSS class used for the animation. The style property expected within the stagger class can either be a **transition-delay** or an **animation-delay** property (or both if your animation contains both transitions and keyframe animations).

```
.my-animation.ng-enter {
  /* standard transition code */
  -webkit-transition: 1s linear all;
  transition: 1s linear all;
  opacity:0;
}
.my-animation.ng-enter-stagger {
  /* this will have a 100ms delay between each successive leave animation */
  -webkit-transition-delay: 0.1s;
  transition-delay: 0.1s;

  /* in case the stagger doesn't work then these two values
   must be set to 0 to avoid an accidental CSS inheritance */
  -webkit-transition-duration: 0s;
  transition-duration: 0s;
}
.my-animation.ng-enter.ng-enter-active {
  /* standard transition styles */
  opacity:1;
}
```

#exercício

APP COM ROTAS E ANIMAÇÃO

ROUTING WITH ANIMATIONS

CUSTOM FILTERS

CORE FILTERS

filter

currency
date
filter
json
limitTo
lowercase
number
orderBy
uppercase

Filters can be applied to expressions in view templates using the following syntax:

```
{{ expression | filter }}
```

E.g. the markup `{{ 12 | currency }}` formats the number 12 as a currency using the `currency` filter. The resulting value is `$12.00`.

Filters can be applied to the result of another filter. This is called "chaining" and uses the following syntax:

```
{{ expression | filter1 | filter2 | ... }}
```

Filters may have arguments. The syntax for this is

```
{{ expression | filter:argument1:argument2:... }}
```

E.g. the markup `{{ 1234 | number:2 }}` formats the number 1234 with 2 decimal points using the `number` filter. The resulting value is `1,234.00`.

CUSTOM FILTER

```
="task.text | isDone:task.status">
```

```
'use strict'

angular.module('myApp')
    .filter('isDone', function(){
        return function(value, status){
            return value + '<strong class="red"> status: ' + status + '</strong>';
        }
});

// filtro
// retorna uma funcao
// que retorna um valor
```

#exercício

CRIANDO UM FILTRO

CUSTOM FILTER

VALIDANDO FORMULÁRIOS

We can prevent form submission in an invalid state by adding `ng-disabled` to the Submit button:

```
<button ng-disabled='!addUserForm.$valid'>Submit</button>
```

Finally, we might want the controller to tell the user she's been successfully added. Our final template would look like:

```
<h1>Sign Up</h1>
<form name='addUserForm' ng-controller="AddUserController">
  <div ng-show='message'>{{message}}</div>
  <div>First name: <input name='firstName' ng-model='user.first' required></div>
  <div>Last name: <input ng-model='user.last' required></div>
  <div>Email: <input type='email' ng-model='user.email' required></div>
  <div>Age: <input type='number'
                   ng-model='user.age'
                   ng-maxlength='3'
                   ng-min='1'></div>
  <div><button ng-click='addUser()'
               ng-disabled='!addUserForm.$valid'>Submit</button>
</ng-form>
```

with controller:

```
function AddUserController($scope) {
  $scope.message = '';

  $scope.addUser = function () {
    // TODO for the reader: actually save user to database...
    $scope.message = 'Thanks, ' + $scope.user.first + ', we added you!';
  };
}
```

```
App</title></head>
<y ng-controller="MainCtrl as ctrl">
<form ng-submit="ctrl.submit()" name="myForm">
<input type="text"
ng-model="ctrl.user.username"
required
ng-minlength="4">
<input type="password"
ng-model="ctrl.user.password"
required>
<input type="submit"
value="Submit"
ng-disabled="myForm.$invalid">
</form>
<script
src="https://ai
</script
```

ngMessages

- directive in module ngMessages

```
angular.module('ngMessagesExample', ['ngMessages']);
```

```
<form name="myForm">
  <label>Enter your name:</label>
  <input type="text"
         name="myName"
         ng-model="name"
         ng-minlength="5"
         ng-maxlength="20"
         required />

  <pre>myForm.myName.$error = {{ myForm.myName.$error | json }}</pre>

  <div ng-messages="myForm.myName.$error" style="color:maroon">
    <div ng-message="required">You did not enter a field</div>
    <div ng-message="minlength">Your field is too short</div>
    <div ng-message="maxlength">Your field is too long</div>
  </div>
</form>
```

DEMO & DOCS

https://docs.angularjs.org/api/ngMessages/directive/ngMessages

DEBOUNCE *(ESPERA)*

```
<input ng-model="data"
       ng-model-options="{debounce: 500}">
```

debounced

123456

123456

regular

BLUR *(SAI DO CAMPO)*

```
<input ng-model="data"
       ng-model-options="{updateOn:'blur'}">
```

updateOn: blur

abcdef

abcdef

regular

abcdef

abcdef

DEMO & DOCS

https://docs.angularjs.org/api/ng/directive/ngModelOptions

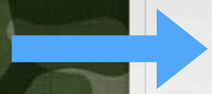
COMPREENDENDO O $emit

```
// firing an event upwards
$scope.$emit('myCustomEvent', 'Data to send');

// firing an event downwards
$scope.$broadcast('myCustomEvent', {
  someProp: 'Sending you an Object!' // send whatever you want
});

// listen for the event in the relevant $scope
$scope.$on('myCustomEvent', function (event, data) {
  console.log(data); // 'Data to send'
});
```

```
$on: function(name, listener) {
  var namedListeners = this.$$listeners[name];
  if (!namedListeners) {
    this.$$listeners[name] = namedListeners = [];
  }
  namedListeners.push(listener);

  var current = this;
  do {
    if (!current.$$listenerCount[name]) {
      current.$$listenerCount[name] = 0;
    }
    current.$$listenerCount[name]++;
  } while ((current = current.$parent));

  var self = this;
  return function() {
    namedListeners[indexOf(namedListeners, listener)] = null;
    decrementListenerCount(self, 1, name);
  };
}
```

```
// subscribes...
var myListener = $scope.$on('child', function (event, data) {
  // do something
});

// unsubscribes...
// this would probably sit in a callback or something
myListener();
```

```
app.controller('ParentCtrl',
  function ParentCtrl ($scope) {

  // $rootScope $on
  var myListener = $rootScope.$on('child', function (event, data) {
    //
  });

  // $scope $destroy
  $scope.$on('$destroy', myListener);

});
```



```
$scope.$on('myCustomEvent', function (event, data) {
  event.stopPropagation();
});
```

#exercício

DISPARANDO EVENTOS

$broadcast + $emit

#exercício

CARREGANDO UM JSON EXTERNO

LOADING JSON

CONHECENDO O $http

```
$http.get('api/user', {params: {id: '5'}
}).success(function(data, status, headers, config) {
 // Do something successful.
}).error(function(data, status, headers, config) {
 // Handle the error
});
```

```
var postData = {text: 'long blob of text'};
  // The next line gets appended to the URL as params
  // so it would become a post request to /api/user?id=5
  var config = {params: {id: '5'}};
  $http.post('api/user', postData, config
).success(function(data, status, headers, config) {
  // Do something successful
}).error(function(data, status, headers, config) {
  // Handle the error
});
```

```
$http({
  method: string,
  url: string,
  params: object,
  data: string or object,
  headers: object,
  transformRequest: function transform(data, headersGetter) or
                    an array of functions,
  transformResponse: function transform(data, headersGetter) or
                     an array of functions,
  cache: boolean or Cache object,
  timeout: number,
  withCredentials: boolean
});
```

```
$http({
    method: 'GET',
    url: 'https://api.flickr.com/services/rest',
    params: {
        method: 'flickr.photos.search',
        api_key: 'ed16856166bf2d96bc8fa297f249dda3',
        text: $scope.valorBusca,
        format: 'json',
        nojsoncallback: 1
    }
}).success(function (data) {
        $scope.buscando = false;
        console.log(data);
        $scope.resultado = data;
}).error(function (error) {
        $scope.buscando = false;
        console.log(error);
});
```

#exercício

CRIANDO UM APP COM $http

BUSCADOR FLICKR

MAIS SOBRE O $http

```
angular.module('MyApp',[]).
  config(function($httpProvider) {
    // Remove the default AngularJS X-Request-With header
    delete $httpProvider.default.headers.common['X-Requested-With'];
    // Set DO NOT TRACK for all Get requests
    $httpProvider.default.headers.get['DNT'] = '1';
 });
```

```
$http.get('api/user', {
   // Set the Authorization header. In an actual app, you would get the auth
   // token from a service
   headers: {'Authorization': 'Basic Qzsda231231'},
   params: {id: 5}
}).success(function() { // Handle success });
```

```
$http.get('http://server/myapi', {
   cache: true
}).success(function() { // Handle success });
```

This enables the cache, and AngularJS stores the response from the server. The next time a request is made for the same URL, AngularJS returns the response from the cache. The cache is also smart, so even if you make multiple simultaneous requests for the same URL, only one request is made to the server and the response is used to fulfill all the requests.

AngularJS applies some basic transformations on all requests and responses made through its `$http` service. These include:

Request transformations

If the data property of the requested `config` object contains an object, serialize it into JSON format.

Response transformations

If an XSRF prefix is detected, strip it. If a JSON response is detected, deserialize it using a JSON parser.

When would we use these? Let us assume that we have a server which is more attuned to the jQuery way of doing things. It would expect our POST data to come in the form `key1=val1&key2=val2` (that is, a string), instead of the JSON form of `{key1: val1, key2: val2}`. While we could make this change at every request, or add a `transformRequest` call individually, for the purpose of this example, we are going to add a general `transformRequest`, so that for all outgoing calls, this transformation from JSON form to a string happens. Here's how we would do this:

```
var module = angular.module('myApp');

module.config(function ($httpProvider) {
    $httpProvider.defaults.transformRequest = function(data) {
        // We are using jQuery's param method to convert our
        // JSON data into the string form
         return $.param(data);
    };
});
```

```
myAppModule.factory('CreditCard', ['$http', function($http) {
  var baseUrl = '/user/123/card';
  return {
    get: function(cardId) {
      return $http.get(baseUrl + '/' + cardId);
    },
    save: function(card) {
      var url = card.id ? baseUrl + '/' + card.id : baseUrl;
      return $http.post(url, card);


    },
    query: function() {
      return $http.get(baseUrl);
    },
    charge: function(card) {
      return $http.post(baseUrl + '/' + card.id, card, {params: {charge: true}});
    }
  };
}]);
```

CONHECENDO O $resource

```
myAppModule.factory('CreditCard', ['$resource', function($resource) {
  return $resource('/user/:userId/card/:cardId',
        {userId: 123, cardId: '@id'},
        {charge: {method:'POST', params:{charge:true}, isArray:false});
}]);
```

Resource Function	Method	URL	Expected Return
CreditCard.get({id: 11})	GET	*/user/123/card/11*	Single JSON
CreditCard.save({}, ccard)	POST	*/user/123/card with post data "ccard"*	Single JSON
CreditCard.save({id: 11}, ccard)	POST	*/user/123/card/11 with post data "ccard"*	Single JSON
CreditCard.query()	GET	*/user/123/card*	JSON Array
CreditCard.remove({id: 11})	DELETE	*/user/123/card/11*	Single JSON
CreditCard.delete({id: 11})	DELETE	*/user/123/card/11*	Single JSON

No Callbacks! (Unless You Really Want Them)

PROMESSAS (PROMISES)

```
var deferred = $q.defer();

var fetchUser = function() {
  // After async calls, call deferred.resolve with the response value
  deferred.resolve(user);

  // In case of error, call
  deferred.reject('Reason for failure');
}
```

```
angular.module('promisesApp', [])
    .controller('PromisesCtrl', ['$scope', 'UserService', function($scope, UserService){

        $scope.buscarDados = function() {
            UserService.buscarUsuarios().then(function(dados){
                alert('sucesso: ' + dados);
            }, function(){
                alert('erro');
            });
        };

    }])
    .factory('UserService', ['$q', function($q){

        function buscarUsuarios () {
            var retorno = $q.defer();
            setTimeout(function(){
                retorno.resolve('resposta do backend');
                // retorno.reject();
            }, 3000);
            return retorno.promise;
        }

        return {
            buscarUsuarios: buscarUsuarios
        }
    }]);
```

`var deferred = $q.defer();` - creates a new deferred

`deferred.resolve(value);` - resolves a deferred with a value

`deferred.reject(reason);` - rejects a deferred with a reason

`var promise = deferred.promise;` - gets a promise from deferred

`promise.then(success, failure);` - assigns callbacks for success (resolve) and failure (reject)

`promise.catch(failure);` - assigns failure callback (equals to `promise.then(null, failure)`)

`promise.finally(always);` - assign a callback to be called both on success or failure

`var promise = $q.reject(reason);` - returns rejected promise with a reason

`var promise = $q.when(valueOrPromise);` - wraps value or other implementation of thenable promise with AngularJS promise

`var promise = $q.all(promisesArr);` - returns a promise that will be resolved only when all promises in `promisesArr` are resolved

Up to here, when we used promises, we assigned both success and failure callbacks. But, there is also a way to assign only success or only failure functions:

Assign only success or failure callback to promise

```
promise.then(function() {
    console.log('Assign only success callback to promise');
});

promise.catch(function() {
    console.log('Assign only failure callback to promise');
    // This is a shorthand for `promise.then(null, errorCallback)`
});
```

In case we want to perform the same operation both on fulfillment or rejection of a promise, we can use `promise.finally()`:

Using finally

```
promise.finally(function() {
    console.log('Assign a function that will be invoked both upon success and
});
```

This is equivalent to:

```
var callback = function() {
    console.log('Assign a function that will be invoked both upon success and
};
promise.then(callback, callback);
```

Sometimes we need to return a rejected promise. Instead of creating a new promise and rejecting it, we can use $q.reject(reason). `$q.reject(reason)` returns a rejected promise with a reason. Example:

$q.reject(reason) example

```
var promise = async().then(function(value) {
        if (isSatisfied(value)) {
            return value;
        } else {
            return $q.reject('value is not satisfied');
        }
    }, function(reason) {
        if (canRecovered(reason)) {
            return newPromiseOrValue;
        } else {
            return $q.reject(reason);
        }
    });
```

Sometimes we need to perform several asynchronous operations, no matter the order, and to be notified when they all done. $q.all(promisesArr) can help us with that. Assume we have `N` methods that return promises: `async1(), ..., asyncN()`. The following code will log `done` only when all operations are resolved successfully:

$q.all(promisesArr) example

```
var allPromise = $q.all([
    async1(),
    async2(),
    ....
    ....
    asyncN()
]);

allPromise.then(function(values) {
    var value1 = values[0],
        value2 = values[1],
        ....
        ....
        valueN = values[N];

        console.log('done');
```

#exercício

USANDO PROMESSAS

PROMISES IN ANGULAR

CUSTOM DIRECTIVES

```
var myModule = angular.module(...);

myModule.directive('namespaceDirectiveName', function factory(injectables) {
 var directiveDefinitionObject = {
   restrict: string,
   priority: number,
   template: string,
   templateUrl: string,
   replace: bool,
   transclude: bool,
   scope: bool or object,
   controller: function controllerConstructor($scope,
                                              $element,
                                              $attrs,
                                              $transclude),
   require: string,
   link: function postLink(scope, iElement, iAttrs) { ... },
   compile: function compile(tElement, tAttrs, transclude) {
     return {
       pre: function preLink(scope, iElement, iAttrs, controller) { ... },
       post: function postLink(scope, iElement, iAttrs, controller) { ... }
     }
   }
 };
 return directiveDefinitionObject;
});
```

Property	Purpose
restrict	Declare how directive can be used in a template as an element, attribute, class, comment, or any combination.
priority	Set the order of execution in the template relative to other directives on the element.
template	Specify an inline template as a string. Not used if you're specifying your template as a URL.
templateUrl	Specify the template to be loaded by URL. This is not used if you've specified an inline template as a string.
replace	If true, replace the current element. If false or unspecified, append this directive to the current element.
transclude	Lets you move the original children of a directive to a location inside the new template.
scope	Create a new scope for this directive rather than inheriting the parent scope.
controller	Create a controller which publishes an API for communicating across directives.
require	Require that another directive be present for this directive to function correctly.
link	Programmatically modify resulting DOM element instances, add event listeners, and set up data binding.
compile	Programmatically modify the DOM template for features across copies of a directive, as when used in ng-repeat. Your compile function can also return link functions to modify the resulting element instances.

restrict

```
angular.module('app', [])
        .directive('bomDiaMaceio', [function(){
            return {
                restrict: 'E',
                template: '<h1>Bom dia Maceió</h1>'
            }
        }]);


// restrict: 'E' element    ->       <bom-dia-maceio></bom-dia-maceio>

// restrict: 'A' attribute  ->       <div bom-dia-maceio></div>

// restrict: 'C' class      ->       <div class="bom-dia-maceio"></div>

// restrict: 'M' comment    ->       <!-- directive:bom-dia-maceio -->
```

```
var appModule = angular.module('app', []);
appModule.directive('hello', function() {
  return {
    restrict: 'E',
    template: '<div>Hi there</div>',
    replace: true
  };
});
```

replace

We'll use it in a page like so:

```
<html lang='en' ng-app='app'>
...
<body>
  <hello></hello>
</body>
...
```

Loading it into a browser, we see "Hi there."

If you were to view the page source, you'd still see the `<hello></hello>` on the page, but if you inspected the generated source (in Chrome, right-click on *Hi there* and select *Inspect Element*), you would see:

```
<body>
  <div>Hi there</div>
</body>
```

The `<hello></hello>` was replaced by the `<div>` from the template.

transclusion

We could change our example to use transclusion:

```
appModule.directive('hello', function() {
  return {
    template: '<div>Hi there <span ng-transclude></span></div>',
    transclude: true
  };
});
```

applying it as:

```
<div hello>Bob</div>
```

We would see: “Hi there Bob.”

scope

1. The **existing scope** from your directive's DOM element.
2. A **new scope** you create that inherits from your enclosing controller's scope. Here, you'll have the ability to read all the values in the scopes above this one in the tree. This scope will be shared with any other directives on your DOM element that request this kind of scope and can be used to communicate with them.
3. An **isolate scope** that inherits no model properties from its parent. You'll want to use this option when you need to isolate the operation of this directive from the parent scope when creating reusable components.

You can create these scope configurations with the following syntax:

Scope Type	Syntax
existing scope	`scope: false` (this is the default if unspecified)
new scope	`scope: true`
isolate scope	`scope: { /* attribute names and binding style */ }`

scope

The syntax without aliases is in the following form:

```
scope: { attributeName1: 'BINDING_STRATEGY',
         attributeName2: 'BINDING_STRATEGY', …
       }
```

With aliases, the form is:

```
scope: { attributeAlias: 'BINDING_STRATEGY' + 'templateAttributeName',
         …
       }
```

The binding strategies are defined by symbols in Table 6-4:

Table 6-4. Binding strategies

Symbol	Meaning
@	Pass this attribute as a string. You can also data bind values from enclosing scopes by using interpolation {{}} in the attribute value.
=	Data bind this property with a property in your directive's parent scope.
&	Pass in a function from the parent scope to be called later.

scope

```
scope: false <- herda do pai
scope: {} <- escopo isolado
    '@' <- binding top down (se mudar muda no pai, muda a diretiva, somente) uma via
    '=' <- 2 way data binding
    '&' <- serve para executar um método do controller do pai
scope: true <- escopo novo não isolado
```

#exercício

CRIANDO UMA DIRETIVA

CUSTOM DIRECTIVES

#exercício

3 WAY DATA BINDING

FIREBASE SAMPLE

TESTING

KARMA = UNIT RUNNER

- Pode pré-processar arquivos (ex: CoffeScript to JS)
- Ativa/usa navegadores para rodar os testes
- Diz ao navegador qual arquivo carregar
- O navegador manda os resultados de volta ao Karma
- Karma reporta as respostas pelo Terminal

MOCHA

- Organiza os testes em Suites (describe)
- Define uma descrição e um callback para cada teste (it)
- Provê hooks de ciclo de vida para cada teste (beforeEach)
- Roda suites e testes
- Captura exceções e mapeia os testes

CHAI

- Assertion API
- Estilos variados (assert, expect, should)
- throw se a assertion falha
- Mocha coleta as exceptions e reporta de volta ao Karma

ANGULAR MOCKS

- Sobre-escreve core services com mocks como o $httpBackend
- Decorates core services como $timeout.flush
- Simplifica a integração Mocha/Jasmine

#exercício

CRIANDO UMA SUITE DE TESTES

ANGULAR KARMA

MELHORES PRÁTICAS

```
templates/
    _login.html
    _feed.html
app/
    app.js
    controllers/
        LoginController.js
        FeedController.js
    directives/
        FeedEntryDirective.js
    services/
        LoginService.js
        FeedService.js
    filters/
        CapatalizeFilter.js
```

```
app/
    app.js
    Feed/
        _feed.html
        FeedController.js
        FeedEntryDirective.js
        FeedService.js
    Login/
        _login.html
        LoginController.js
        LoginService.js
    Shared/
        CapatalizeFilter.js
```

DESATIVAR DEBUG DATA

Tools like Protractor and Batarang need this information to run, but you can disable this in production for a significant performance boost with:

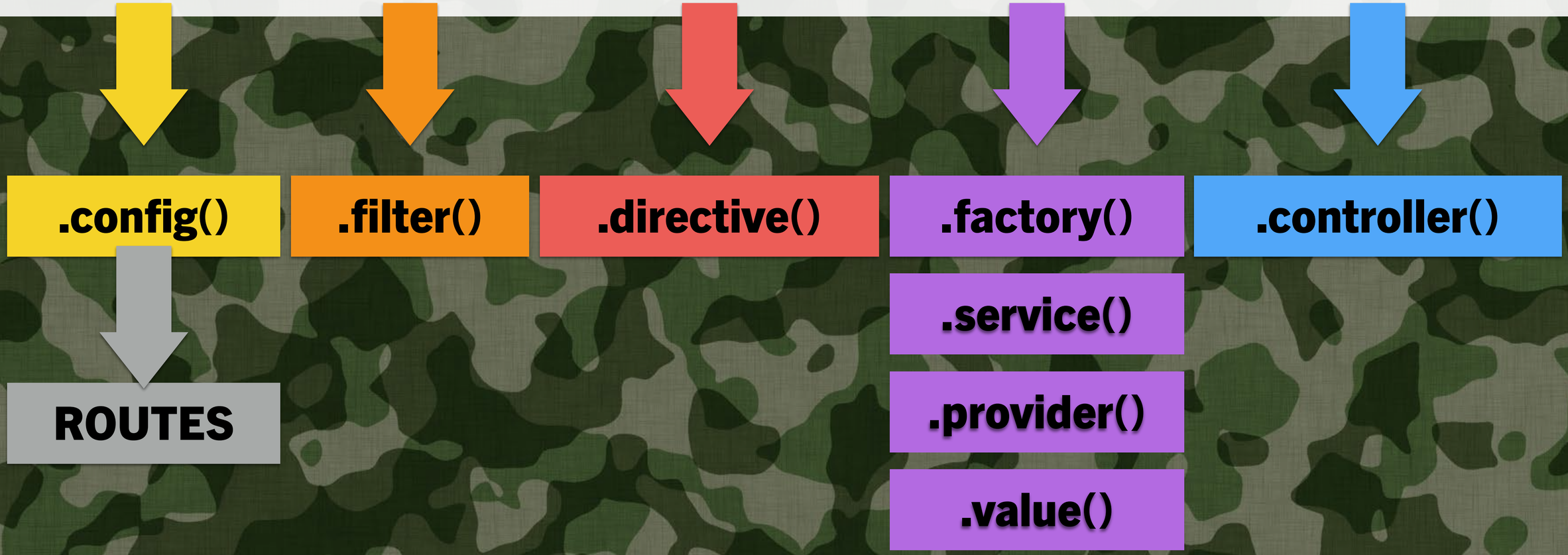
```
myApp.config(['$compileProvider', function ($compileProvider) {
  $compileProvider.debugInfoEnabled(false);
}]);
```

If you wish to debug an application with this information then you should open up a debug console in the browser then call this method directly in this console:

```
angular.reloadWithDebugInfo();
```

MODULE

```
angular.module('myApp', []);
```



@demianborba

OBRIGADO