[**Habilitando o Angular em nossa página**](https://cursos.alura.com.br/course/angularjs-mvc/section/2/exercise/2)

Vejamos as seguintes afirmações sobre Angular:

1) Angular é um script como outro qualquer, por isso precisa ser importado em nossa página através da tag script.

2) A diretiva ng-app carrega e importa o módulo que desejamos carregar.

3) A diretiva ng-app possui como valor o nome do módulo que desejamos carregar assim que nossa página for carregada.

4) Um módulo pode ter de 0 a N dependências.

Com base nessas afirmações, podemos dizer que:

* 2 e 4 são falsas
* 1 e 3 são verdadeiras
* 3 e 4 são falsas
* **1,3 e 4 são verdadeiras**

**Resposta do instrutor**

Angular é um script como outro qualquer, por isso precisa ser importado em nossa página através da tagscript, mas isso não é suficiente. Precisamos criar um módulo (outro script) e importá-lo em nossa página. Este módulo é apelidado de principal, porque é o primeiro que será carregado pelo Angular e que pode ou não carregar outros a partir dele, isto é, suas dependências. Importar o script do módulo também não é suficiente, por quê? Se tivéssemos carregado vários scripts de módulos em nossa página, como o Angular saberia qual inicializar primeiro? É por isso que precisamos usar a diretiva ng-app, que possui como valor o nome do módulo que desejamos carregar assim que nossa página for carregada.

**Criação de módulos**

Temos:

1) angular.module('alurapic', []);

2) angular.module('alurapic');

3) angular.module('alurapic', ['calopsitaStyle']);

4) module('alurapic', []);

No que diz respeito à criação de módulos, podemos afirmar que:

* Apenas 4 está errada
* **2 e 4 estão erradas**
* 2 e 3 estão corretas
* Apenas 3 e 4 estão erradas

**Resposta do instrutor**

Criamos um módulo através do objeto global angular. Ele recebe dois parâmetros: o primeiro é o nome do módulo e o segundo um array com suas dependências. Por mais que o o módulo não tenha dependência, precisamos passar o array vazio como parâmetro. Sua omissão indica para o Angular que desejamos acessar um módulo já criado.

**Angular Expression**

Temos as seguintes declarações sobre Angular Expression (AE):

1) Angular Expression serve para abrir lacunas em nossas views, que por sua vez passam a funcionar como templates.

2) Angular Expression possui a sintaxe {nomeDaPropriedade}.

3) Controllers podem fornecer dados para uma Angular Expression.

4) Uma Angular Expression não avaliada tem como valor null.

* Todas as afirmativas são verdadeiras
* **1 e 3 estão corretas**
* Apenas 4 está incorreta
* 1, 2 e 3 estão corretas

**Resposta do instrutor**

Usamos Angular Expressions (AE's) para abrir lacunas em views. Essa ideia de termos lacunas que precisam ser preenchidas remete à ideia de templates, que nada mais são do que um modelo com um monte de coisa pronta, mas que possui lacunas que precisam ser preenchidas para que fique completo.

A sintaxe de uma AE é {{nomeDaPropriedade}}. Mas quem fornece os dados para que essas "lacunas" sejam tapadas? Essa é a responsabilidade de um Controller no mundo MVC e também no mundo Angular. Caso não exista nenhum controller ou caso o controller não forneça os dados de que a AE precisa, Angular não emite um erro, simplesmente considera seu valor uma String em branco, como "".

**Controller do Angular**

Vejamos os seguintes códigos:

1) angular.module('contasAhReceber').controller(function($scope) { });

2) angular.module('contabilidade', []).controller('LancamentosController', function() { });

3) angular.module('estoque').controller('Provisao', function($scope) { });

4) angular.controller('CaixaCtrl', function($scope) { });

Sobre a criação de Controller em Angular, podemos afirmar sobre as alternativas apresentadas que:

* 1 e 3 estão corretas
* Apenas 4 está errada
* 2 e 4 estão erradas
* **2 e 3 estão corretas**

**Resposta do instrutor**

**Um controller pode ser criado com um módulo ou mais tarde adicionado em um módulo já existente. Por exemplo:**

**// criando com um módulo  
angular.module('contabilidade', []).controller('LancamentosController', function() { });**

**A função controller recebe como primeiro parâmetro o nome do controller e como segundo a função que o define, que pode ou não receber via injeção de dependência o $scope. Lembre-se que é através dele que disponibilizamos dados para a view, adicionando propriedades dinamicamente.**

**Vejamos agora um exemplo criando um controller para um módulo já existente, que está perfeitamente correto:**

**angular.module('estoque').controller('Provisao', function($scope) { });**

**No exemplo anterior, como não passamos o array [] como segundo parâmetro da função module, significa que estamos criando um controller para o módulo estoque que já foi criado. Controllers podem ser criados ao mesmo tempo em que criamos módulos ou posteriormente para um módulo já criado, sendo assim, podemos criar perfeitamente um controller para um módulo já existente e foi isso que fizemos no capítulo:**

**angular.module('alurapic').controller('FotosController', function($scope) {  
  
 $scope.foto = {  
 titulo : 'Leão',  
 url : 'http://www.fundosanimais.com/Minis/leoes.jpg'  
 };  
  
});**

**Criamos o controller FotosController para o módulo alurapic já existente.**

**Controller e seu escopo**

**Temos a seguinte marcação em nossa view:**

**<!-- public/index.html -->  
<!DOCTYPE html>  
<html lang="pt-br" ng-app="alurapic">  
 <head>  
 <!-- código omitido -->  
 </head>  
 <body ng-controller="FotosController">  
 <div class="container">  
 <h1 class="text-center">Alurapic</h1>  
  
 <img class="img-responsive center-block" src="{{foto.url}}" alt="{{foto.titulo}}">  
  
 </div><!-- fim container -->  
 </body>  
</html>**

**O controller FotosController foi associado ao elemento body e qualquer um de seus elementos filhos terão acesso a tudo o que foi adicionado no escopo do controller. Qual das opções abaixo disponibiliza corretamente foto no escopo do controller:**

* **angular.module('alurapic').controller('FotosController', function($scope) {  
   var foto = {  
   titulo : 'Leão',  
   url: 'http://fotosdoalem.com/leao.jpg'  
   };  
  });**
* **angular.module('alurapic').controller('FotosController', function($scope) {  
   $scope = {  
   titulo : 'Leão',  
   url: 'http://fotosdoalem.com/leao.jpg'  
   };  
  });**
* **angular.module('alurapic').controller('FotosController', function($scope) {  
   $scope.foto = {  
   titulo : 'Leão',  
   url: 'http://fotosdoalem.com/leao.jpg'  
   };  
  });**
* **angular.module('alurapic').controller('FotosController', function($scope) {  
   foto = {  
   titulo : 'Leão',  
   url: 'http://fotosdoalem.com/leao.jpg'  
   };  
  });**

**Resposta do instrutor**

**Quando criamos um controller, não basta declararmos variáveis com o mesmo nome das Angular Expressions dentro deles. A comunicação é feita através de $scope, um objeto que é a ligação entre o controller e a view. Qualquer propriedade adicionada neste objeto estará disponível na view e acessível através de AEs. Porém, caso o elemento que use a AE não contenha a diretiva ng-controller ou não seja filho de quem a contenha, a AE não será resolvida.**

**"Test your might!"**

**Temos as seguintes afirmativas:**

**1 - Data binding significa associação de dado, isto é, podemos associar um dado à view através de Angular Expression e qualquer mudança nesse dado refletirá na sua apresentação na view.**

**2 - Uma Angular Expression (AE) é somente leitura, isto é, alterações dos dados na view não se propaga para o model.**

**3 - Um model no Angular pode ser um objeto ou qualquer tipo literal do JavaScript, como String ou Boolean.**

**4 - Angular não é uma solução que se coaduna com a prática de Progressive Enhancement no que diz respeito a criação de páginas web.**

**Podemos afirmar que:**

* Apenas a 4 está errada.
* **Todas as afirmativas estão corretas**
* Apenas a 3 está errada.
* 4 e 3 estão erradas

**Resposta do instrutor**

**Uma Angular Expression (AE) é somente leitura, isto é, nossa view lê a informação no escopo do controller, mas não é capaz de atualizá-la. Este tipo de associação é chamado de one-way data binding, porque a informação flui do controller para a view e não contrário. Aliás, esse dado associado à view pode ser qualquer tipo literal do JavaScript, inclusive um objeto.**

**Angular não se coaduna com a prática do Progressive Enhancement, no qual criamos uma página usando um simples HTML e CSS sem JavaScript para depois então adicionar melhorias. Se por acaso alguma dessas melhorias deixarem de funcionar, não comprometerá o acesso ao conteúdo por parte do usuário. No caso do Angular, se o JavaScript estiver desligado ou se o dispositivo no qual nossa aplicação é carregada não o suporta, nada da aplicação funcionará.**

**Failed to parse SourceMap**

**Se no Chrome, durante o treinamento o console do navegador exibir a mensagem**

**Failed to parse SourceMap:** [**http://localhost:3000/js/lib/angular.min.js.map**](http://localhost:3000/js/lib/angular.min.js.map)

**Não se preocupe, você não fez nada de errado. As últimas versões do Chrome podem exibir essa mensagem quando arquivos minificados forem carregados. No caso do projeto do Angular que você baixou, esses arquivos não existem. Aliás, para que servem esses arquivos? Quando recebemos uma mensagem de erro do um script minificado, muitas vezes temos dificuldade em analisar a linha que disparou o erro porque o processo de minificação altera e trunca muitos nomes. O que o source map faz é permitir que seja usado o arquivo minificado, mas que a mensagem do debugger do navegador exiba algo mais legível. Como não somos mantenedores do projeto do Angular, a ausência desse arquivo não nos causará problemas.**