

# PROGRAMAÇÃO WEBII

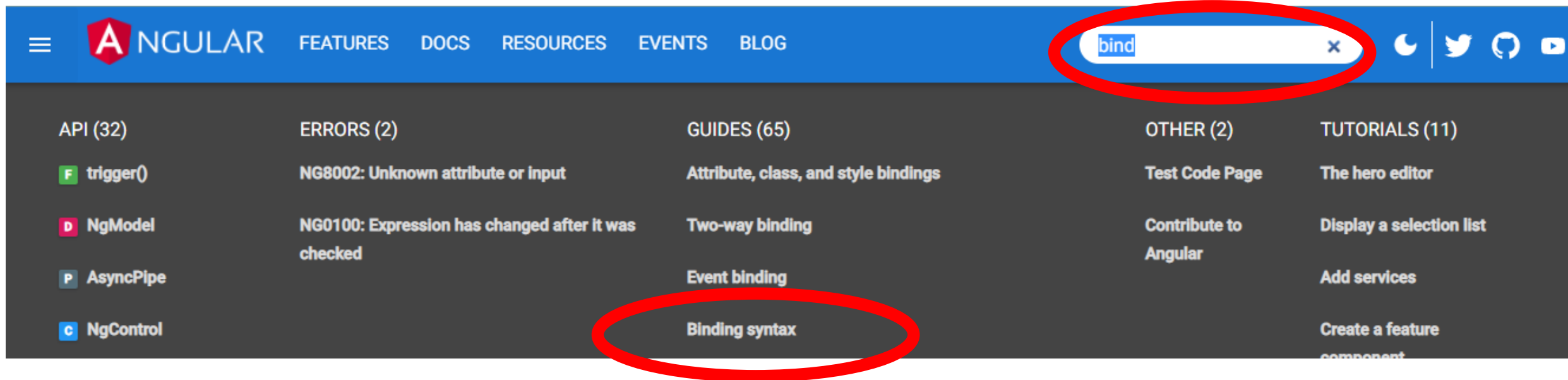
# 1

## Two way binding, funções e Event binding

Data bindings

# Two way binding, funções e Event binding

- Vamos aprender sobre o two-way data binding e o event binding.
- Na documentação do angular, digitar na pesquisa bind e selecionar bind syntax



# Two way binding, funções e Event binding

■ Clicar na opção Types of data binding

Attribute Class Style	<code>((expression))</code> <code>[target]="expression"</code> <code>bind-target="expression"</code>	source to view target
Event	<code>(target)="statement"</code> <code>on-target="statement"</code>	One-way from view target to data source
Two-way	<code>[(target)]="expression"</code> <code>bindon-target="expression"</code>	Two-way


Binding syntax

Data binding and HTML

HTML attributes and DOM properties

- **Types of data binding**

Binding types and targets



# Two way binding, funções e Event binding

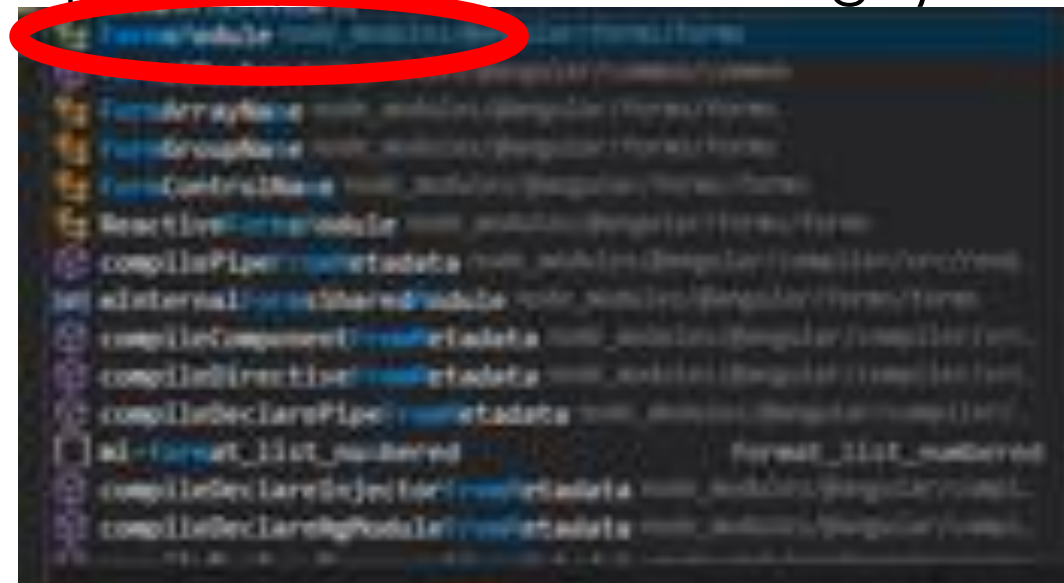
- Vamos abrir o VsCode
- Primeiro, vamos entender o que é o two way data binding
- Abra o arquivo cadastrar-produto.html
- Selecione o código:

```
8      <label for="nome">Nome do Produto: </label>
9      <input
10         type="text"
11         class="form-control"
12         placeholder="Digite o nome do Produto"
```

- É a possibilidade de digitarmos alguma coisa dentro do input e ele automaticamente enviar a informação que está sendo digitada dentro do input para o TypeScript

# Two way binding, funções e Event binding

- Para fazer isso, vamos abrir o arquivo `app.module.ts` e iremos precisar fazer um import
- Na última linha do import, inserir uma virgula e vamos digitar na próxima linha `FormsModule` (A extensão que instalamos já irá exibir o que precisaremos importar para o nosso código). Só selecionar a primeira opção:



# Two way binding, funções e Event binding

- Após isso, já irá realizar a importação do módulo no início do nosso código:

```
12 import { LOCALE_ID } from '@angular/core';  
13 import localePt from '@angular/common/locales/pt';  
14 import { registerLocaleData } from '@angular/common';  
15 import { FormsModule } from '@angular/forms';
```

# Two way binding, funções e Event binding

- Como atualizamos o `app.module.ts`, teremos que parar nosso servidor e começar novamente
  - `ng serve -o <ENTER>`



# Two way binding, funções e Event binding

- Por que importar o `FormsModule`?
  - ▶ Nós iremos utilizar a diretiva `[(ngModel)]` e o **ngModel** está no **FormsModule** agora.
- O **FormControl** faz parte da área de **Reactive Forms** do Angular. Este conceito é usado quando se quer controlar os campos de formulários, ou seja, conseguimos controlar de forma reativa qualquer coisa relativa ao campo e não somente ao valor do campo.
- Com o **ngModel** temos uma ligação do valor do campo com uma variável na programação do componente.

# Two way binding, funções e Event binding

## Falando do ngModel

1. ngModel em angular2 é válido apenas se o FormsModule estiver disponível como parte do seu AppModule.
2. `ng-model` está sintaticamente errado.
3. chaves quadradas `[..]` referem-se à ligação da propriedade.
4. chaves de círculo `(..)` referem-se à ligação de evento.
5. quando chaves quadradas e de círculo são colocadas juntas, como `[(..)]` refere-se ao encadernamento de duas vias, comumente chamado de caixa de banana.

# Two way binding, funções e Event binding

Caixa de banana:

- Neste tipo de binding (two way binding), temos a união dos dois tipos de binding em uma sintaxe conhecida como "banana in the box" ou `[(ngModel)]`, como podemos ver esta sintaxe realmente lembra duas bananas, representando os parenteses "()", e a caixa, representando os colchetes "[]", uma forma simples de lembrar a sintaxe é imaginar esta figura abaixo:



<http://blog.almeidapedro.com.br/post/2017/12/13/bindings-em-angular>

# Two way binding, funções e Event binding

- Agora, vamos abrir o nosso arquivo cadastrar-produto.component.ts e criar as nossas variáveis:
- Após o export, entre chaves, digitar:
  - nome: string;
  - validade: string;
  - preco: number;

```
nome: string = '';  
validade: string = '';  
preco: number = 0;
```

```
8 export class CadastrarProdutoComponent implements OnInit {  
9  
10     nome: string;  
11     validade: string;  
12     preco: number;  
13  
14     constructor() { }  
15  
16     ngOnInit(): void {  
17     }  
18 }
```

# Two way binding, funções e Event binding

- Agora, vamos abrir o nosso arquivo cadastrar-produto.component.html
- Após o comando placeholder="Digite o nome do Produto", para utilizarmos o ngModel, iremos precisar primeiro definir a propriedade name do nosso campo. O nosso input precisa ter um name, o name do nosso primeiro input será "nomeProduto", depois podemos utilizar o ngModel, entre colchetes e chaves e depois a nossa variável que acabamos de criar no ts, conforme segue abaixo:

name="nomeProduto" [(ngModel)]="nome"

<ALT> <SHIFT><F> para formatar o nossos código e vamos salvar

```
<input
  type="text"
  class="form-control"
  placeholder="Digite o nome do Produto"
  name="nomeProduto"
  [(ngModel)]="nome"
/>
```

# Two way binding, funções e Event binding

Vamos verificar como ficou nosso formulário, mas por enquanto nada mudou. Vamos fazer para os demais inputs do nosso formulário.

CRUD Angular Home Produtos ▾

## Cadastrar Produto

Nome do Produto:

Validade do Produto:



Preço do Produto:

[Voltar](#)

[Salvar](#)

# Two way binding, funções e Event binding

- Vamos para o campo validade
- Após o class = "form-control", digitar:  
name="validade" [(ngModel)] = "validade"

```
<input  
  type="date"  
  class="form-control"  
  name="validade"  
  [(ngModel)]="validade"  
>
```

# Two way binding, funções e Event binding

- Vamos para o campo preco
- Após o class = "form-control", digitar:  
name="preco" [(ngModel)] = "preco"

```
<input  
  type="number"  
  class="form-control"  
  placeholder="0.00"  
  name="preco"  
  [(ngModel)]="preco"  
>
```



# Two way binding, funções e Event binding

- Agora já estamos fazendo a ligação entre os input's do formulário html com as variáveis que temos no TypeScript, em tempo real isso está acontecendo.
- Para testar o nosso event binding que é o próximo tópico, teremos que criar uma função. Como criamos uma função no Angular?
- Vamos criar como se fosse uma variável, mas colocando parênteses para declarar que é uma função.
- Vamos abrir o nosso arquivo cadastrar-produto.component.ts

# Two way binding, funções e Event binding

- Antes do última chave }, vamos criar o método salvar e vamos chamar ele de salvarProduto(). Após o parênteses, temos que falar qual é o tipo de retorno desse método. Como esse método não terá retorno algum, iremos declarar como void e depois abre e fecha chaves.
- <ALT><SHIFT><F> para formatar
- Ficaria assim:

```
15     ngOnInit(): void {}  
16  
17     salvarProduto(): void {}  
18 }
```

# Two way binding, funções e Event binding

- Já criamos o método. O que o método irá fazer?
- Ele irá chamar o console.log para escrever o nome do produto, validade e preço. Vamos colocar também um alert para termos certeza que o código está funcionando, apenas para simular um aviso para o usuário., conforme abaixo. Vamos salvar e verificar nossa aplicação como ficou.

```
salvarProduto(): void {  
    console.log('Nome: ', this.nome);  
    console.log('Validade: ', this.validade);  
    console.log('Preço: ', this.preco);  
    alert('Salvo com sucesso!');  
}  
}
```

# Two way binding, funções e Event binding

- Vamos acionar o modo de desenvolvedor <F12>
- Vamos simular o cadastro de um produto e clicar no botão Salvar.
- Nada vai acontecer, porque precisamos ativar o event binding

## Cadastrar Produto

Nome do Produto:

Validade do Produto:



Preço do Produto:

[Voltar](#)

[Salvar](#)

# Two way binding, funções e Event binding

- Voltando no nosso código, vamos no arquivo cadastrar-produto.component.html
- No botão Salvar, após o float-right", vamos colocar parênteses para chamar o event binding (). Dentro do parênteses, vamos chamar o evento click e o evento de click irá chamar o método chamarProduto()  
(click)="salvarProduto()"
- Salvar e testar novamente

```
<button class="btn btn-success float-right" (click)="salvarProduto()">Salvar</button>
```

# Two way binding, funções e Event binding

Para testar, vamos digitar os dados novamente e clicar no botão Salvar

The screenshot shows a web browser at `localhost:4200/produtos/cadastrar`. The page title is "Cadastrar Produto". The form contains the following data:

- Nome do Produto: Curso de Angular
- Validade do Produto: 31/12/2021
- Preço do Produto: 53.34

A green "Salvar" button is at the bottom right. A red circle highlights a toast message that says "Salvo com sucesso!". Another red circle highlights the browser's developer console, which shows the following log:

Nome:	Validade:	Preço:
Curso de Angular	2021-12-31	53.34

The console log also shows the file path `cadastrar-produto.component.ts` for each field.

# Two way binding, funções e Event binding

- O que podemos fazer aqui, temporariamente é criarmos um card.
- Abra o arquivo cadastrar-produto.component.html, após o último `</div>`, digitar: `div.card<TAB>`
- Vamos utilizar a tag `<p></p>` para cada campo e fazer a interpolação da variável. Não vamos nem utilizar o pipe aqui, é mais para vocês verificarem como o data binding funciona

```
54     </div>
55     <div class="card">
56         <p>Nome: {{ nome }}</p>
57         <p>Validade: {{ validade }}</p>
58         <p>Preço: {{ preco }}</p>
59     </div>
```


# Two way binding, funções e Event binding

■ Vamos salvar e testar...

## Cadastrar Produto

Nome do Produto:

Validade do Produto:



Preço do Produto:

Voltar

Salvar

Nome:

Validade:

Preço: 0



# Two way binding, funções e Event binding

- Vamos só colocar uma margem top para melhorar a visualização abaixo:
- Após o card, digitar: mt-3
- Salvar e testar

```
<div class="card mt-3">  
  <p>Nome: {{ nome }}</p>  
  <p>Validade: {{ validade }}</p>  
  <p>Preço: {{ preco }}</p>  
</div>
```


# Two way binding, funções e Event binding

- O visual ficará um pouco melhor...

## Cadastrar Produto

Nome do Produto:

Validade do Produto:



Preço do Produto:

Voltar

Salvar

Nome:

Validade:

Preço: 0

# Two way binding, funções e Event binding

- Agora será possível entender melhor o conceito. O que você escrever no campo, automaticamente vai sendo exibido na tabela abaixo.
- O two way data binding permite que digitemos texto diretamente no nosso input e já converse automaticamente com o nosso TypeScript

Nome do Produto:

Validade do Produto:

Voltar

Nome: Curso de

Validade:

Preço: 0

## Recapitulando...

- Precisamos ir no app.module.ts e colocar o FormsModule na propriedade Imports

```
imports: [  
  BrowserModule,  
  AppRoutingModule,  
  FormsModule  
],
```

- Precisamos importar ele, porque não vem por padrão no nosso projeto.

```
import { FormsModule } from '@angular/forms';
```

- Após alterar o arquivo app.module.ts, devemos parar o servidor e executar novamente

## Recapitulando...

■ Agora você pode criar as suas variáveis dentro do seu componente

```
nome: string = '';  
validade: string = '';  
preco: number = 0;
```

■ Ir no seu input e colocar o ngModule, lembrando que o input precisa ter um name.

```
<input  
  type="date"  
  class="form-control"  
  name="validade"  
  [(ngModel)]="validade"  
>
```

## Recapitulando...

- Outra coisa que nós vimos foi o event binding, que irá permitir que chamemos os nossos métodos, nossas funções através de eventos.
- Então iremos colocar sempre parênteses, o nome do nosso evento e dentro das aspas que método queremos chamar quando aquele evento acontecer.

```
<button class="btn btn-success float-right" (click)="salvarProduto()">
```

