

# **Projeto e Desenvolvimento de Software II**

**Prof. Carlos Eduardo de Carvalho Dantas**

**carloseduardodantas@iftm.edu.br**

# **Parte I – Técnicas e Ferramentas para Desenvolvimento de Sistemas**

# AGENDA

---

## 1. Desenvolvimento Front-End

- Exemplos Angular

# Atividades e Exercícios Angular

---

- 1) Criar um novo projeto Angular chamado ExerciciosAngular

```
PS C:\Users\carlo> ng new ExerciciosAngular  
As a forewarning, we are moving the CLI npm p
```

# Atividades e Exercícios Angular

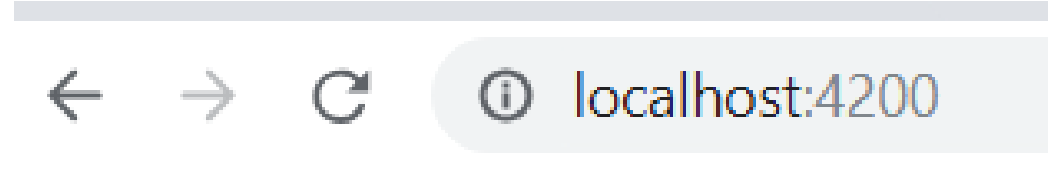
- Exemplo 1) A variável **nomes** se trata de um atributo da classe AppComponent, que é um componente. Essa variável pode ser usada no template html para que os dados sejam mostrados

```
TS app.component.ts x
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9
10     nomes: string[] = ['joão', 'maria', 'josé', 'pedro', 'felipe', 'carlos'];
11
12  }
13
```

```
TS app.component.ts  <> app.component.html x
1  <ul>
2    <li *ngFor="let nome of nomes">
3      {{nome}}
4    </li>
5  </ul>
```

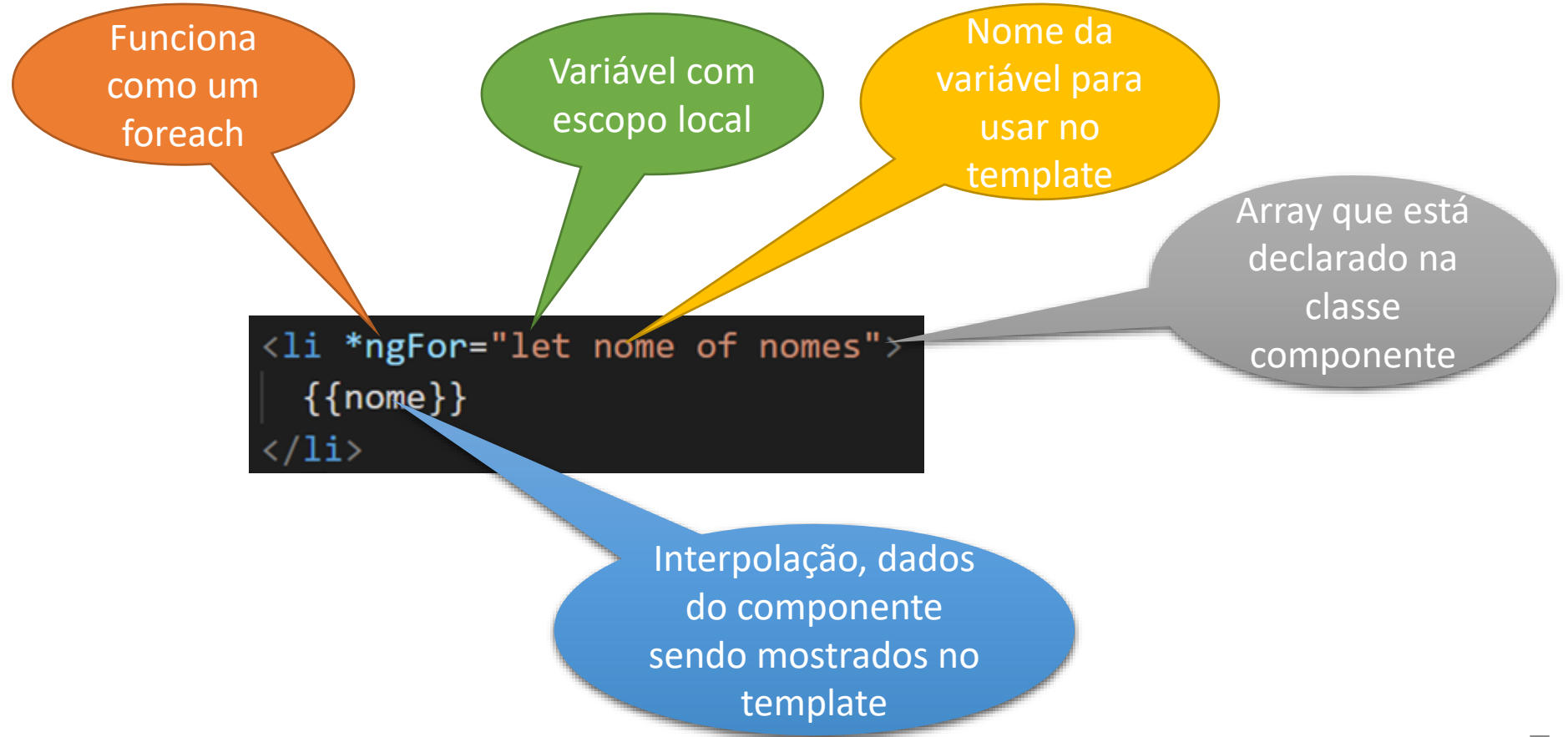
# Atividades e Exercícios Angular

## - Exemplo 1



- joão
- maria
- josé
- pedro
- felipe
- carlos

# Atividades e Exercícios Angular



# Atividades e Exercícios Angular

- Exemplo 2) É possível mostrar também a posição de cada elemento

```
ts app.component.ts  <> app.component.html ●  
1  <ul *ngFor="let nome of nomes, let i= index">  
2    <li>nome da pessoa é: {{nome}}, está na posição {{i + 1}} </li>  
3  </ul>
```

← → ↻ ⓘ localhost:4200

- nome da pessoa é: joão, está na posição 1
- nome da pessoa é: maria, está na posição 2
- nome da pessoa é: josé, está na posição 3
- nome da pessoa é: pedro, está na posição 4
- nome da pessoa é: felipe, está na posição 5
- nome da pessoa é: carlos, está na posição 6



# Atividades e Exercícios Angular

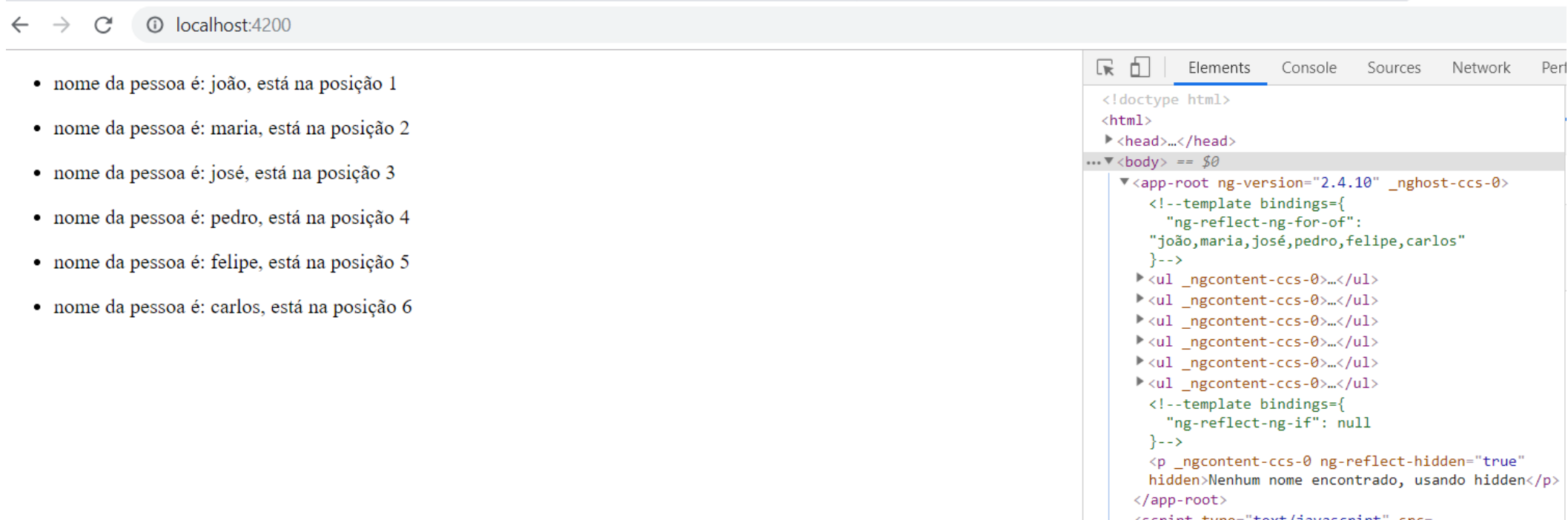
- Exemplo 3) Usando `*ngIf` e `[hidden]`. Ambos são usados para esconder determinado elemento (poderia ser uma div inteira) de acordo com uma condição booleana.

```
TS app.component.ts  <> app.component.html x
```

```
1  <ul *ngFor="let nome of nomes, let i= index">
2  |   <li>nome da pessoa é: {{nome}}, está na posição {{i + 1}} </li>
3  | </ul>
4  | <p *ngIf="nomes.length == 0">Nenhum nome encontrado, usando *ngIf</p>
5  |
6  | <p [hidden]="nomes.length > 0">Nenhum nome encontrado, usando hidden</p>
```

# Atividades e Exercícios Angular

- Exemplo 3) \*ngIf adiciona ou retira do DOM, o que é melhor para segurança, mas pior para desempenho.



The screenshot shows a web browser at localhost:4200 displaying a list of names and their positions. The list is as follows:

- nome da pessoa é: joão, está na posição 1
- nome da pessoa é: maria, está na posição 2
- nome da pessoa é: josé, está na posição 3
- nome da pessoa é: pedro, está na posição 4
- nome da pessoa é: felipe, está na posição 5
- nome da pessoa é: carlos, está na posição 6

The developer console shows the following HTML structure:

```
<!doctype html>
<html>
  <head>...</head>
  <body> == $0
    <app-root ng-version="2.4.10" _ngghost-ccs-0>
      <!--template bindings={
        "ng-reflect-ng-for-of":
        "joão,maria,josé,pedro,felipe,carlos"
      }-->
      <ul _ngcontent-ccs-0>...</ul>
      <ul _ngcontent-ccs-0>...</ul>
      <ul _ngcontent-ccs-0>...</ul>
      <ul _ngcontent-ccs-0>...</ul>
      <ul _ngcontent-ccs-0>...</ul>
      <ul _ngcontent-ccs-0>...</ul>
      <!--template bindings={
        "ng-reflect-ng-if": null
      }-->
      <p _ngcontent-ccs-0 ng-reflect-hidden="true"
        hidden>Nenhum nome encontrado, usando hidden</p>
    </app-root>
  </body>
</html>
```

# Atividades e Exercícios Angular

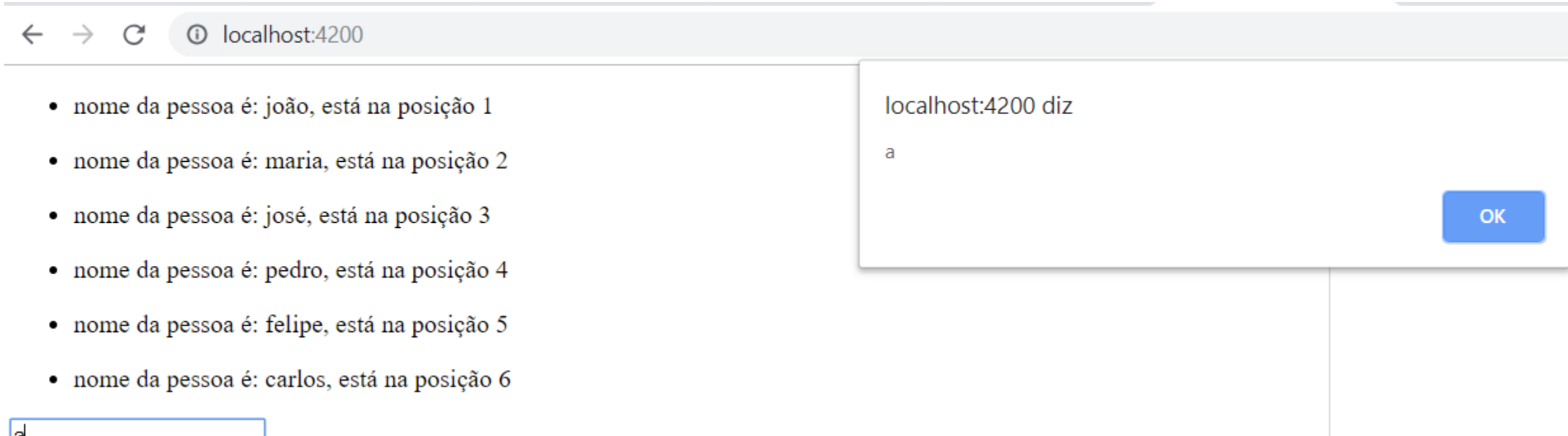
- Exemplo 4) Buscar por algum nome em específico

```
<input #box placeholder="digite um nome" (keyup)="buscar(box.value)"/>
```

```
export class AppComponent {  
  nomes: string[] = ['joão', 'maria', 'josé', 'pedro', 'felipe', 'carlos'];  
  
  buscar(valor: string) {  
    alert(valor)  
  }  
}
```

# Atividades e Exercícios Angular

## - Exemplo 4) Buscar por algum nome em específico



The screenshot shows a web browser window with the address bar displaying 'localhost:4200'. The main content area contains a list of six items, each representing a person's name and their position. Below the list is an input field. A modal dialog box is open, displaying the text 'localhost:4200 diz' followed by the letter 'a' on a new line. The dialog has an 'OK' button in the bottom right corner.

- nome da pessoa é: joão, está na posição 1
- nome da pessoa é: maria, está na posição 2
- nome da pessoa é: josé, está na posição 3
- nome da pessoa é: pedro, está na posição 4
- nome da pessoa é: felipe, está na posição 5
- nome da pessoa é: carlos, está na posição 6

localhost:4200 diz  
a

OK

# Atividades e Exercícios Angular

- Exemplo 4) Buscar por algum nome em específico

Variável de  
referência do  
template

Evento  
keyup, tecla  
pressionada

```
<input #box placeholder="digite um nome" (keyup)="buscar(box.value)"/>
```

Chamando o método na classe  
componente, usando a variável  
de referência para obter o valor  
do input

# Atividades e Exercícios Angular

- Exemplo 5) Implementando a busca
  - **Método 1** – iteração básica e verbosa.

```
<ul *ngFor="let nomeFiltro of nomesFiltro">  
  <li>{{nomeFiltro}}</li>  
</ul>
```

```
nomesFiltro: string[];  
  
buscar(valor: string) {  
  this.nomesFiltro = [];  
  
  //método 1  
  for (var i = 0; i < this.nomes.length; i++) {  
    if (this.nomes[i].toLowerCase().includes(valor.toLowerCase())) {  
      this.nomesFiltro.push(this.nomes[i]);  
    }  
  }  
}
```

Função para  
verificar se uma  
string é substring  
de outra

Passar a  
string para  
minúsculo

Adiciona elemento  
no final do array

# Atividades e Exercícios Angular

- Exemplo 5) Implementando a busca
  - **Método 2** – usando forEach, método interno do array.

```
//método 2
let temp = [];
this.nomes.forEach(buscarItem);
function buscarItem(nome) {
  if (nome.toLowerCase().includes(valor.toLowerCase())) {
    temp.push(nome);
  }
}
this.nomesFiltro = temp;
```

A própria função forEach já irá injetar cada nome como argumento para a função buscarItem

O argumento da função forEach é outra função que irá processar cada item do array

# Atividades e Exercícios Angular

- Exemplo 5) Implementando a busca
  - **Método 2** – a função que é chamada para processar cada elemento da lista também é chamada de função de callback.

```
var nomes = ['maria', 'josé', 'joão'];
```

```
nomes.forEach(function(nome){  
    console.log(nome);  
});
```



1º Iteração: nome = 'maria'  
2º Iteração: nome = 'josé'  
3º Iteração: nome = 'joão'



# Atividades e Exercícios Angular

- Exemplo 5) Implementando a busca
  - **Método 3** – usando o método filter.

```
//método 3  
this.nomesFiltro = this.nomes.filter(function (nome) {  
    return nome.toLowerCase().includes(valor.toLowerCase());  
});
```

O método filter é usado quando precisa filtrar os elementos da lista de acordo com algum critério

O argumento da função filter é outra função que irá processar cada item do array. No método 2 a função foi declarada explicitamente. No método 3, se trata de uma função anônima

# Atividades e Exercícios Angular

- Exemplo 5) Implementando a busca
  - **Método 4** – usando o método filter com arrow functions.

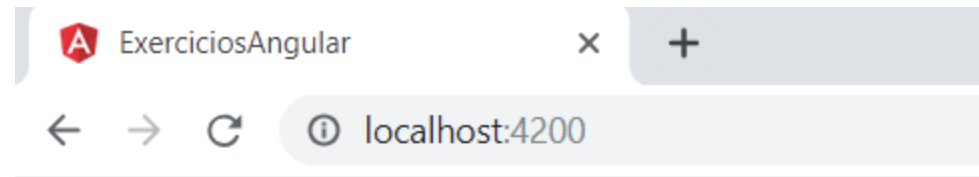
```
//método 4  
this.nomesFiltro = this.nomes.filter(  
  (nome) => nome.toLowerCase().includes(valor.toLowerCase()));
```

Possui exatamente a mesma semântica do método 3. Só diminui a verbosidade do código.

Já se sabe que o método filter irá receber cada nome da lista, e retornar booleano. Logo, a arrow function dispensa a formalidade das declarações

# Atividades e Exercícios Angular

## - Exemplo 5) Implementando a busca



- nome da pessoa é: joão, está na posição 1
- nome da pessoa é: maria, está na posição 2
- nome da pessoa é: josé, está na posição 3
- nome da pessoa é: pedro, está na posição 4
- nome da pessoa é: felipe, está na posição 5
- nome da pessoa é: carlos, está na posição 6

- carlos

# Atividades e Exercícios Angular

---

- É muito comum usar laços de repetição nos algoritmos. O problema é que com a abordagem dos métodos 1 e 2, é impossível saber qual o objetivo do corpo da iteração sem ver a sua implementação.
  - Buscar elemento?
  - listar todos?
  - Excluir?
  - Ordenar?
- Os métodos auxiliares visam explicitar qual operação se deseja realizar com base na sua sintaxe.

# Atividades e Exercícios Angular

---

- **Filter** – filtrar a lista de acordo com algum critério
- **Map** – modificar os elementos do array.
- **Find** – encontrar algum item específico
- **Every** – retorna verdadeiro se todos os itens do array respeitam alguma condição
- **Some** – retorna verdadeiro se pelo menos 1 item do array respeita alguma condição
- **Reduce** – Pegar todos os valores do array e condensar em um único elemento.

# Atividades e Exercícios Angular

## - Exemplo 6) Somando valores com reduce

Cria uma lista de "struct". Não é bem um objeto typescript, são apenas dados

```
14 | pessoas: any = [  
15 |   {id: 1, nome: 'joao', salario: 5000},  
16 |   {id: 2, nome: 'maria', salario: 1000},  
17 |   {id: 3, nome: 'josé', salario: 2000},  
18 |   {id: 4, nome: 'pedro', salario: 3000},  
19 |   {id: 5, nome: 'felipe', salario: 10000},  
20 |   {id: 6, nome: 'carlos', salario: 800},  
21 | ]  
22 |  
23 | getValorTotal(): Number {  
24 |   return this.pessoas.reduce(  
25 |     (soma, pessoa) => soma + pessoa.salario, 0);  
26 | }  
27 |  
14 | <p>Salario total: {{getValorTotal()}}</p>
```

Inicializa a variável soma com 0

A função reduce tem como primeiro argumento a variável de "consolidação acumulada". O segundo argumento é o elemento corrente do vetor

# Atividades e Exercícios Angular

- Exemplo 7) Buscando um elemento com find

```
buscarId(id) {  
  return this.pessoas.find(pessoa => pessoa.id == id);  
}
```

Cada elemento da lista é uma pessoa

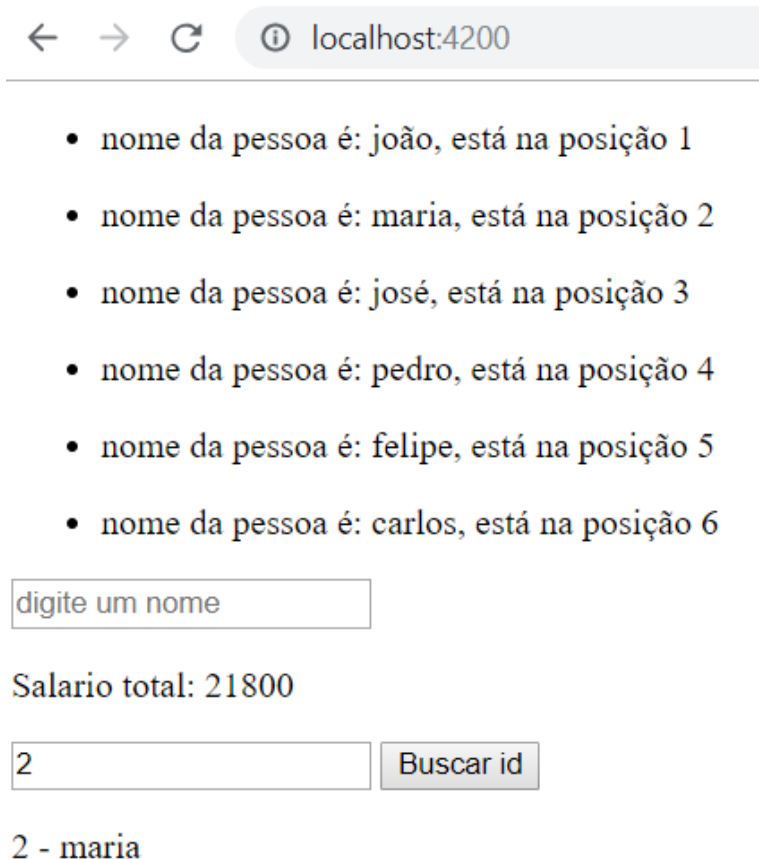
Só pode retornar uma pessoa, já que id é chave.

Usa variáveis de referência para o input e para o retorno da função

```
16 <input #parametroid placeholder="digite um id"/>  
17 <button (click)="retorno = buscarId(parametroid.value)">Buscar id</button>  
18 <p *ngIf="retorno != undefined">{{retorno.id}} - {{retorno.nome}}</p>
```

# Atividades e Exercícios Angular

## - Exemplo 7) Buscando um elemento com find



← → ↻ ⓘ localhost:4200

- nome da pessoa é: joão, está na posição 1
- nome da pessoa é: maria, está na posição 2
- nome da pessoa é: josé, está na posição 3
- nome da pessoa é: pedro, está na posição 4
- nome da pessoa é: felipe, está na posição 5
- nome da pessoa é: carlos, está na posição 6

digite um nome

Salario total: 21800

2

2 - maria



# Atividades e Exercícios Angular

## Exemplo 8) Aplicando map

Não existe  
retorno  
em map  
pois a  
alteração  
será  
aplicada  
na própria  
lista

```
28     aumentarSalario(percentual) {  
29         this.pessoas.map(pessoa =>  
30             pessoa.salario += pessoa.salario * percentual/100)  
31     }
```

```
20     <br>  
21     <input #percentual placeholder="digite o percentual"/>  
22     <button (click)="aumentarSalario(percentual.value)">Aumentar salario</button>  
23
```

# Atividades e Exercícios Angular

## - Exemplo 9) Aplicando every

```
verificaSalario(valor: number) {  
  return this.pessoas.every(pessoa => pessoa.salario > valor);  
}
```

Retorna booleano. Verifica se todo mundo ganha mais que o valor informado.

```
<p>{{verificaSalario(500)?'Todo mundo ganha mais que 500':  
'nem todo mundo ganha mais que 500'}}</p>
```

Usando operador ternário. Não se pode imprimir true ou false para o usuário

# Atividades e Exercícios Angular

## - Exemplo 10) Aplicando some

```
23 buscaCampos(criterio: string) {  
24     return this.pessoas.filter((pessoa) =>  
25         Object.keys(pessoa).some  
26         (chave => pessoa[chave].toString().includes(criterio)));  
}
```

Usa a combinação filter + some. Filter para filtrar por pessoas, e some para verificar os campos de cada pessoa

Retorna todas as chaves de pessoa: [id, nome, salario]

Verifica se o valor de algum dos campos de pessoa possui o critério como substring

# Atividades e Exercícios Angular

## - Exemplo 10) Aplicando some

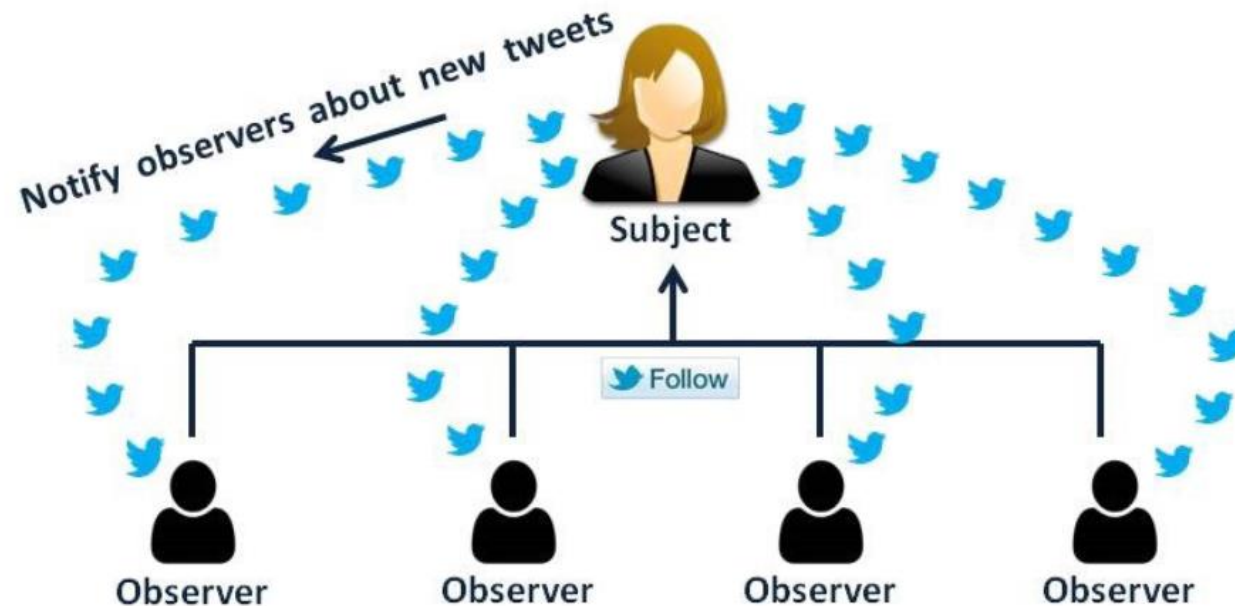
```
24 <br>
25 <input #criterio placeholder="digite um criterio de busca"
26 (keyup)="resposta = buscaCampos(criterio.value)"/>
27
28 <ul *ngFor="let r of resposta">
29   <li>{{r.id}} - {{r.nome}} - {{r.salario}}</li>
30 </ul>
```

- 1 - joao - 5000
- 2 - maria - 1000
- 5 - felipe - 10000

# Atividades e Exercícios Angular

- Observables - é uma coleção que funciona de forma unidirecional, ou seja, ele emite notificações sempre que ocorre uma mudança em um de seus itens e a partir disto pode-se executar uma ação.

## Observer Design Pattern



# Atividades e Exercícios Angular

---

- Observables estimulam a programação reativa (está incluído no pacote RxJS (Reactive Extensions), usado no Angular.
  - <https://github.com/Reactive-Extensions/RxJS>
  - <http://reactivex.io/>
- Programação reativa é um paradigma de programação orientado a fluxo de dados e propagação de mudança.
  - O Observer estimula a propagação de mudança, já que toda vez em que ocorre a mudança de estado, observadores são notificados.
  - Fluxo de dados ocorre porque os dados enviados podem ser controlados automaticamente, sem controle de estado.

# Atividades e Exercícios Angular

- As atualizações ocorrem automaticamente na medida em que novos dados surgem – programação assíncrona.

## Observables

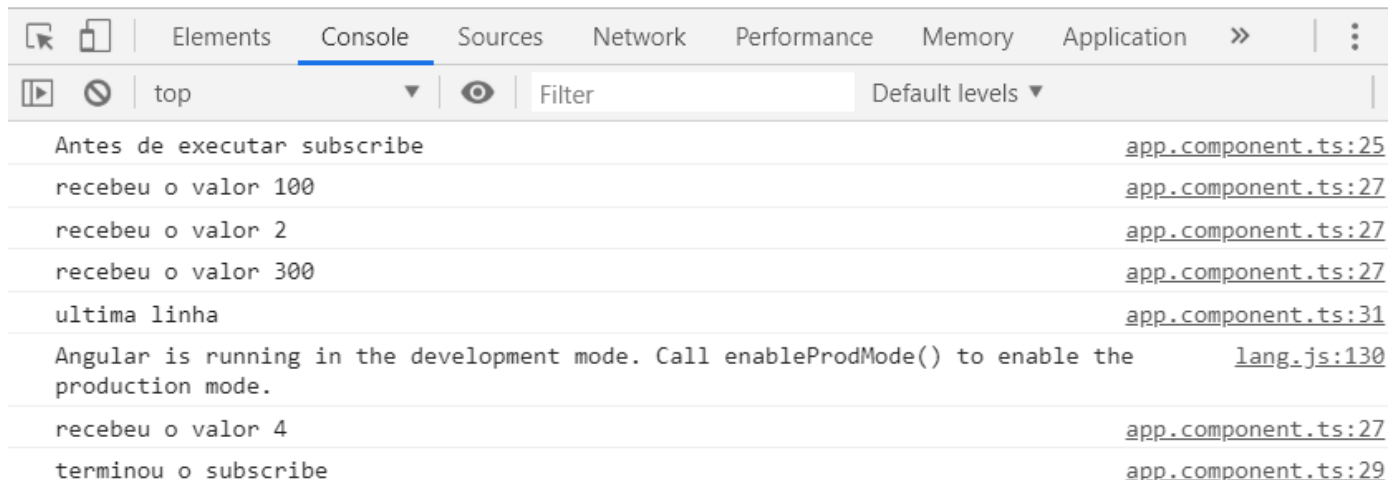


# Atividades e Exercícios Angular

## - Exemplo 11) Usando observables

```
ngOnInit() {  
  const observable = new Observable(subscriber => {  
    subscriber.next(100);  
    subscriber.next(2);  
    subscriber.next(300);  
    setTimeout(() => {  
      subscriber.next(4);  
      subscriber.complete();  
    }, 1000);  
  });  
});
```

```
console.log('Antes de executar subscribe');  
observable.subscribe({  
  next(x) { console.log('recebeu o valor ' + x); },  
  error(err) { console.error('Erro: ' + err); },  
  complete() { console.log('terminou o subscribe'); }  
});  
console.log('ultima linha');  
}
```





# Atividades e Exercícios Angular

## - Exemplo 11) Usando observables

```
ngOnInit() {  
  const observable = new Observable(subscriber => {  
    subscriber.next(100);  
    subscriber.next(2);  
    subscriber.next(300);  
    setTimeout(() => {  
      subscriber.next(4);  
      subscriber.complete();  
    }, 1000);  
  });  
}
```

setTimeout  
enviará a  
mensagem  
depois do  
tempo  
determinado  
, isto é, 1000  
milissegundo  
s.

A função subscriber  
define como obter  
valores e  
mensagens  
publicados no  
observer. Essa  
função é executada  
apenas quando o  
método subscribe  
for executado.

A cada chamada next,  
um novo valor é  
colocado no fluxo de  
dados

# Atividades e Exercícios Angular

## - Exemplo 11) Usando observables

```
console.log('Antes de executar subscribe');  
observable.subscribe({  
  next(x) { console.log('recebeu o valor ' + x); },  
  error(err) { console.error('Erro: ' + err); },  
  complete() { console.log('terminou o subscribe'); }  
});  
console.log('ultima linha');  
}
```

Quando o método subscribe é executado, recebe um conjunto de valores do observer, seja síncrono ou assíncrono.

A última linha executa antes do valor 4 exatamente pelo fato de subscribe ser assíncrono.

# Atividades e Exercícios Angular

## - Exemplo 12) Manipulando lista com observables

```
TS app.component.ts • app.component.html
8  })
9  export class AppComponent implements OnInit {
10     observable: Observable<string>;
11
12     nomes: Array<string> = [];
13
14     ngOnInit() {
15         this.observable = new Observable(subscriber => {
16             setInterval(() => {
17                 subscriber.next(this.makeid(5));
18             }, 10000);
19         });
20         let lista: Array<string> = [];
21         this.observable.subscribe({
22             next(x) { lista.push(x); },
23             error(err) { alert('ocorreu um erro '+err); }
24         });
25         this.nomes = lista;
```

Cada vez que o observable produzir algum conteúdo, a lista chamada nomes irá receber este conteúdo

A cada 10 segundos irá produzir uma string, usando o método makeid.

# Atividades e Exercícios Angular

## - Exemplo 12) Manipulando lista com observables

Para ficar mais interessante, a lista de nomes também receberá input do usuário

```
29 enviar(valor: string) {  
30     this.nomes.push(valor);  
31 }  
32  
33 makeid(length) {  
34     var text = "";  
35     var possible = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";  
36     for (var i = 0; i < length; i++)  
37         text += possible.charAt(Math.floor(Math.random() * possible.length));  
38     return text;  
39 }  
40  
41 }  
42
```

Método construído só para gerar caracteres aleatoriamente

# Atividades e Exercícios Angular

## - Exemplo 12) Manipulando lista com observables

```
TS app.component.ts • app.component.html x
1 <input placeholder="digite um nome" #campo>
2 <button (click)="enviar(campo.value)">Enviar</button>
3 <ul *ngFor="let r of nomes">
4   <li>{{r}}</li>
5 </ul>
```

Teste x

localhost:4200

fernando Enviar

- carlos
- p2FR9
- fernando
- 5KhjO
- jIRwx
- xa8Uk
- Xw4wu
- BTvkP

# REFERÊNCIAS

---

- GUEDES, Thiago. Crie aplicações com Angular. Casa do Código, 2018.
- GRONER, Loiane. Curso Angular. Disponível em: <https://www.youtube.com/watch?v=tPOMG0D57S0&list=PLGxZ4Rq3BOBoSRcKWEdQACbUCNWLczg2G>. Acesso em 02/04/2019.
- <https://medium.com/tableless/entendendo-rxjs-observable-com-angular-6f607a9a6a00>