

# Componentes de Classe

Labenu\_



# O que vamos ver hoje?

- Revisão rápida
- Componentes de classe
- Reatividade e Estado
- Inputs Controlados



# Relembrando...

- Componente é **uma função que retorna um JSX**
- Temos 3 regras para componentes em React:
  - Precisamos **importar** o React no topo
  - **Primeira letra** do nome deve ser **maiúscula**
  - Regra do **pai único**



# Relembrando...

```
1 import React from 'react' ←
2
3 export function MeuComponente() {
4   return (
5     <div>
6       <h1>Meu 1º componente!</h1>
7       <p>Esse é meu 1º componente</p>
8     </div>
9   )
10 }
```

- Importar o React no topo
- Primeira letra maiúscula
- Regra do pai único





# Componentes de Classe

Labenu\_



# Novo tipo de Componente ✨

- Agora vamos ter um novo tipo de componente
- Ele é definido usando **classes** ao invés de funções
- Mais código 
- Possui algumas **funcionalidades extras!** 



# Classes ✨

- Classe é uma **estrutura de dados** presente no Javascript e em muitas outras linguagens
- Não precisamos entender como elas funcionam a fundo agora, **apenas como usá-las**
- O formatinho é sempre o mesmo, só copiar e com o tempo decorar :)



# Nova Sintaxe

## Componente Funcional

```
1 import React from 'react'
2
3 export function MeuComponente() {
4   return (
5     <div>
6       <h1>Meu 1º componente!</h1>
7       <p>Esse é meu 1º componente</p>
8     </div>
9   )
10 }
```

## Componente de Classe

```
1 import React from 'react'
2
3 export class MeuComponente extends React.Component {
4   render() {
5     return(
6       <div>
7         <h1>Meu 1º componente!</h1>
8         <p>Esse é meu 1º componente</p>
9       </div>
10    )
11  }
12 }
```





# render()

- `render()` é um método da classe
  - **Método** é o nome dado para toda **função** declarada em uma classe
- É equivalente à função que antes representava todo o componente funcional
- Não recebe nenhum parâmetro ⇒ `render()`  
↓



# Props

- Não precisam mais ser "**declaradas**" em nenhum lugar, a própria classe garante que teremos elas
- Acessamos as props usando a sintaxe `this.props`
- De resto, é tudo igual!



# Props

## Componente Funcional

```
1 import React from 'react'
2
3 export function MeuComponente(props) {
4   return (
5     <div>
6       <h1>Meu 1º componente!</h1>
7       <p>Esse é meu 1º componente</p>
8       <p>{props.texto}</p>
9     </div>
10   )
11 }
```

## Componente de Classe

```
1 import React from 'react'
2
3 export class MeuComponente extends React.Component {
4   render() {
5     return(
6       <div>
7         <h1>Meu 1º componente!</h1>
8         <p>Esse é meu 1º componente</p>
9         <p>{this.props.texto}</p>
10       </div>
11     )
12   }
13 }
```



# Funções de Evento - Funcional

- Antes, qualquer função extra necessária era declarada no corpo da função

```
1 import React from 'react'
2
3 export function MeuComponente(props) {
4   const onClickBotao = () => {
5     console.log('Clicou!')
6   }
7
8   return <div>
9     <h1>Meu primeiro componente!</h1>
10    <p>Este é o meu primeiro componente React</p>
11    <p>{props.texto}</p>
12    <button onClick={onClickBotao}>Clique!</button>
13  </div>
14 }
```



# Funções de Evento - Classe

- Agora, devemos declarar no escopo da classe ("ao lado" do método render)
- Nenhuma keyword (como `const` ou `function`) é necessária
- Para referenciá-la, precisamos usar a keyword `this`



# Funções de Evento - Classe

```
1 import React from 'react'
2
3 export class MeuComponente extends React.Component {
4   onClickBotao = () => {
5     console.log('Clicou!')
6   }
7
8   render() {
9     return <div>
10       <h1>Meu primeiro componente!</h1>
11       <p>Este é o meu primeiro componente React</p>
12       <button onClick={this.onClickBotao}>Clique!</button>
13     </div>
14   }
15 }
```





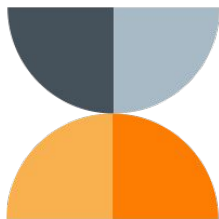
# Exercício 1

- Transformar o componente CardVideo do Labetube em um componente de classe



# Pausa para relaxar 🤪

10 min



- Componente de classe é um novo tipo de componente com **mais funcionalidades**
- Possui algumas diferenças de sintaxe:
  - `extends React.Component`
  - Recebe parâmetro `props` automaticamente
  - JSX é retornado no método `render()`
  - Funções de evento no mesmo nível que o `render()` => uso: `this.nomeDaFuncao()`





# Estado

Labenu\_



# Motivação

- Já vimos como criar componentes em React
- Podemos passar **propriedades** (**props**) para eles de forma que possam ser **reutilizados** em diferentes lugares com valores diferentes
- Mas, por hora, eles são **completamente estáticos**! Ou seja, não conseguimos mudar o que está na tela de forma dinâmica.



# Solução: Estados! 🎉

- Conceito **muito importante** do React
- **Objeto** que guarda propriedades que **mudam com o tempo**
- **Reativo!** ⇒ React **atualiza a tela** quando alguma das propriedades muda



# Estudo de Caso: Contador

- Para entender estado, vamos ver a implementação de um **contador** em React
- Dois botões:
  - Soma
  - Subtrai
- Número que mostra o valor do contador



# Estudo de Caso: Contador

```
1 import React from "react";
2
3 export class Contador extends React.Component {
4   render() {
5     return (
6       <div>
7         <p>0</p>
8         <button>Subtrai</button>
9         <button>Soma</button>
10      </div>
11    );
12  }
13 }
```



# Declarando um Estado

- Um estado pode ser declarado com uma variável chamada **state** na classe
- Ela deve ser um **objeto**
- Guarda os dados que **mudam** na tela
- Valores passados na declaração representam o **estado inicial**



# Declarando um Estado

```
1 import React from "react";
2
3 export class Contador extends React.Component {
4   state = {
5     valorContador: 0
6   }
7
8   render() {
9     return (
10       <div>
11         <p>0</p>
12         <button>Subtrai</button>
13         <button>Soma</button>
14       </div>
15     );
16   }
17 }
```



Declaração do estado



Nome de uma propriedade



Valor inicial da propriedade



# Lendo um Estado

- Podemos ler o estado da mesma forma que lemos as props: `this.state`
- Ele é um **objeto** com todas as propriedades que foram declaradas
- Inicialmente, terá o **valor que foi definido** na classe





# Lendo um Estado

```
1 import React from "react";
2
3 export class Contador extends React.Component {
4   state = {
5     valorContador: 0
6   }
7
8   render() {
9     return (
10       <div>
11         <p>{this.state.valorContador}</p>
12         <button>Subtrai</button>
13         <button>Soma</button>
14       </div>
15     );
16   }
17 }
```

Diagram illustrating the initial state of the component:

- An arrow points from the initial value `0` in the `state` object (line 5) to the text `Inicialmente: 0`.
- Another arrow points from the text `Inicialmente: 0` to the `render()` method (line 8), indicating that the state is used in the rendering process.



# Atualizando um Estado

- Não podemos alterar o estado diretamente
  - Exemplo **errado**: `this.state.contador = 10`
- É preciso fazer isso por meio de uma função especial: `this.setState()`
- Essa função recebe um **objeto com as propriedades** que serão atualizadas e **seus novos valores**



# Atualizando um Estado

- O componente é atualizado imediatamente
- `this.state` passa a ter o novo valor definido
- **Interface reflete atualização**  $\Rightarrow$  método `render()`, que cuida do nosso layout, roda novamente
- Assim, o `render()` **roda de novo toda vez que um estado é atualizado**, de maneira otimizada

Vamos ver na prática! 



# Atualizando o Estado

```
1 import React from "react";
2
3 export class Contador extends React.Component {
4   state = {
5     valorContador: 0
6   }
7
8   onClickSoma = () => {
9     const valorAtual = this.state.valorContador
10    const proximoValor = valorAtual + 1
11
12    this.setState({valorContador: proximoValor})
13  }
14
15  onClickSubtrai = () => {
16    this.setState({valorContador: this.state.valorContador - 1})
17  }
18
19  render() {
20    return (
21      <div>
22        <p>{this.state.valorContador}</p>
23        <button onClick={this.onClickSubtrai}>Subtrai</button>
24        <button onClick={this.onClickSoma}>Soma</button>
25      </div>
26    );
27  }
28 }
```

- Vamos começar olhando para a função **onClickSoma**



# Atualizando o Estado

```
1 import React from "react";
2
3 export class Contador extends React.Component {
4   state = {
5     valorContador: 0
6   }
7
8   onClickSoma = () => {
9     const valorAtual = this.state.valorContador
10    const proximoValor = valorAtual + 1
11
12    this.setState({valorContador: proximoValor})
13  }
14
15  onClickSubtrai = () => {
16    this.setState({valorContador: this.state.valorContador - 1})
17  }
18
19  render() {
20    return (
21      <div>
22        <p>{this.state.valorContador}</p>
23        <button onClick={this.onClickSubtrai}>Subtrai</button>
24        <button onClick={this.onClickSoma}>Soma</button>
25      </div>
26    );
27  }
28 }
```

- Vamos começar olhando para a função **onClickSoma**
- Começamos pegando o valor atual do contador



# Atualizando o Estado

```
1 import React from "react";
2
3 export class Contador extends React.Component {
4   state = {
5     valorContador: 0
6   }
7
8   onClickSoma = () => {
9     const valorAtual = this.state.valorContador
10    const proximoValor = valorAtual + 1
11
12    this.setState({valorContador: proximoValor})
13  }
14
15  onClickSubtrai = () => {
16    this.setState({valorContador: this.state.valorContador - 1})
17  }
18
19  render() {
20    return (
21      <div>
22        <p>{this.state.valorContador}</p>
23        <button onClick={this.onClickSubtrai}>Subtrai</button>
24        <button onClick={this.onClickSoma}>Soma</button>
25      </div>
26    );
27  }
28 }
```

- Vamos começar olhando para a função **onClickSoma**
- Começamos pegando o valor atual do contador
- Então, calculamos o próximo valor. Nesse caso, o próximo valor é o valor atual somado a 1



# Atualizando o Estado

```
1 import React from "react";
2
3 export class Contador extends React.Component {
4   state = {
5     valorContador: 0
6   }
7
8   onClickSoma = () => {
9     const valorAtual = this.state.valorContador
10    const proximoValor = valorAtual + 1
11
12    this.setState({valorContador: proximoValor})
13  }
14
15  onClickSubtrai = () => {
16    this.setState({valorContador: this.state.valorContador - 1})
17  }
18
19  render() {
20    return (
21      <div>
22        <p>{this.state.valorContador}</p>
23        <button onClick={this.onClickSubtrai}>Subtrai</button>
24        <button onClick={this.onClickSoma}>Soma</button>
25      </div>
26    );
27  }
28 }
```

- Vamos começar olhando para a função **onClickSoma**
- Começamos pegando o valor atual do contador
- Então, calculamos o próximo valor. Nesse caso, o próximo valor é o valor atual somado a 1
- Por fim, definimos o novo estado com a função **this.setState**



# Atualizando o Estado

```
1 import React from "react";
2
3 export class Contador extends React.Component {
4   state = {
5     valorContador: 0
6   }
7
8   onClickSoma = () => {
9     const valorAtual = this.state.valorContador
10    const proximoValor = valorAtual + 1
11
12    this.setState({valorContador: proximoValor})
13  }
14
15  onClickSubtrai = () => {
16    this.setState({valorContador: this.state.valorContador - 1})
17  }
18
19  render() {
20    return (
21      <div>
22        <p>{this.state.valorContador}</p>
23        <button onClick={this.onClickSubtrai}>Subtrai</button>
24        <button onClick={this.onClickSoma}>Soma</button>
25      </div>
26    );
27  }
28 }
```

- Ela recebe um **objeto**





# Atualizando o Estado

```
1 import React from "react";
2
3 export class Contador extends React.Component {
4   state = {
5     valorContador: 0
6   }
7
8   onClickSoma = () => {
9     const valorAtual = this.state.valorContador
10    const proximoValor = valorAtual + 1
11
12    this.setState({valorContador: proximoValor})
13  }
14
15  onClickSubtrai = () => {
16    this.setState({valorContador: this.state.valorContador - 1})
17  }
18
19  render() {
20    return (
21      <div>
22        <p>{this.state.valorContador}</p>
23        <button onClick={this.onClickSubtrai}>Subtrai</button>
24        <button onClick={this.onClickSoma}>Soma</button>
25      </div>
26    );
27  }
28 }
```

- Ela recebe um **objeto**
- Passamos a **propriedade** que queremos atualizar, e seu **novo valor**



# Atualizando o Estado

```
1 import React from "react";
2
3 export class Contador extends React.Component {
4   state = {
5     valorContador: 0
6   }
7
8   onClickSoma = () => {
9     const valorAtual = this.state.valorContador
10    const proximoValor = valorAtual + 1
11
12    this.setState({valorContador: proximoValor})
13  }
14
15  onClickSubtrai = () => {
16    this.setState({valorContador: this.state.valorContador - 1})
17  }
18
19  render() {
20    return (
21      <div>
22        <p>{this.state.valorContador}</p>
23        <button onClick={this.onClickSubtrai}>Subtrai</button>
24        <button onClick={this.onClickSoma}>Soma</button>
25      </div>
26    );
27  }
28 }
```

- Valor do contador é atualizado
- Agora, ele vale o que foi passado na função **this.setState**
- No caso da primeira execução da função onClickSoma, o novo valor seria **1**



# Atualizando o Estado

```
1 import React from "react";
2
3 export class Contador extends React.Component {
4   state = {
5     valorContador: 0
6   }
7
8   onClickSoma = () => {
9     const valorAtual = this.state.valorContador
10    const proximoValor = valorAtual + 1
11
12    this.setState({valorContador: proximoValor})
13  }
14
15  onClickSubtrai = () => {
16    this.setState({valorContador: this.state.valorContador - 1})
17  }
18
19  render() {
20    return (
21      <div>
22        <p>{this.state.valorContador}</p>
23        <button onClick={this.onClickSubtrai}>Subtrai</button>
24        <button onClick={this.onClickSoma}>Soma</button>
25      </div>
26    );
27  }
28 }
```

- A função **onClickSubtrai** segue a mesma lógica, de uma forma mais resumida



# Atualizando o Estado

```
1 import React from "react";
2
3 export class Contador extends React.Component {
4   state = {
5     valorContador: 0
6   }
7
8   onClickSoma = () => {
9     const valorAtual = this.state.valorContador
10    const proximoValor = valorAtual + 1
11
12    this.setState({valorContador: proximoValor})
13  }
14
15  onClickSubtrai = () => {
16    this.setState({valorContador: this.state.valorContador - 1})
17  }
18
19  render() {
20    return (
21      <div>
22        <p>{this.state.valorContador}</p>
23        <button onClick={this.onClickSubtrai}>Subtrai</button>
24        <button onClick={this.onClickSoma}>Soma</button>
25      </div>
26    );
27  }
28 }
```

- A função **onClickSubtrai** segue a mesma lógica, de uma forma mais resumida
- Passamos o próximo valor do estado, já calculando com base no valor atual



# Atualizando o Estado

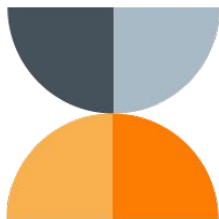
```
1 import React from "react";
2
3 export class Contador extends React.Component {
4   state = {
5     valorContador: 0
6   }
7
8   onClickSoma = () => {
9     const valorAtual = this.state.valorContador
10    const proximoValor = valorAtual + 1
11
12    this.setState({valorContador: proximoValor})
13  }
14
15  onClickSubtrai = () => {
16    this.setState({valorContador: this.state.valorContador - 1})
17  }
18
19  render() {
20    return (
21      <div>
22        <p>{this.state.valorContador}</p>
23        <button onClick={this.onClickSubtrai}>Subtrai</button>
24        <button onClick={this.onClickSoma}>Soma</button>
25      </div>
26    );
27  }
28 }
```

- A função **onClickSubtrai** segue a mesma lógica, de uma forma mais resumida
- Passamos o próximo valor do estado, já calculando com base no valor atual
- Da mesma forma, temos que passar também a propriedade sendo atualizada



# Pausa para relaxar 🤪

10 min



- Para criar um estado, declaramos um objeto chamado `state` no topo da classe
- Para ler o estado, usamos `this.state`
- Para atualizar o estado, usamos `this.setState()`



# Inputs Controlados

Labenu\_



# Lidando com Inputs

- Podemos dizer que o input do HTML possui um estado próprio
- Para trabalhar com ele corretamente no React, queremos que o **nosso componente controle o estado do input**
- Essa técnica é chamada de **inputs controlados**





# Controlando o Input

- Cada input será representado por uma propriedade do estado da classe
- Essa propriedade **define** o valor (**value**) do input
- Toda vez que o input **muda**, o estado deve ser **atualizado**
- Isso é feito através do atributo **onChange**



# Função onChange

- A função que passamos para o **onChange** recebe um **event** como parâmetro
- O **event** possui o valor do input dentro de **event.target.value**
- Pegamos esse valor e colocamos na propriedade do **estado** referente àquele input

Vamos ver na prática! 



# Inputs Controlados

```
1 import React from "react";
2
3 export class FormUsuario extends React.Component {
4   state = {
5     valorNome: '',
6     valorEmail: ''
7   }
8
9   onChangeNome = (event) => {
10     this.setState({valorNome: event.target.value})
11   }
12
13   onChangeEmail = (event) => {
14     this.setState({valorEmail: event.target.value})
15   }
16
17   render() {
18     return (
19       <div>
20         <input name='nome' onChange={this.onChangeNome} value={this.state.valorNome} />
21         <input name='email' onChange={this.onChangeEmail} value={this.state.valorEmail} />
22       </div>
23     );
24   }
25 }
```

- Definindo o estado inicial. Como os inputs começam vazios, as duas propriedades começam como strings vazias



# Inputs Controlados

```
1 import React from "react";
2
3 export class FormUsuario extends React.Component {
4   state = {
5     valorNome: '',
6     valorEmail: ''
7   }
8
9   onChangeNome = (event) => {
10     this.setState({valorNome: event.target.value})
11   }
12
13   onChangeEmail = (event) => {
14     this.setState({valorEmail: event.target.value})
15   }
16
17   render() {
18     return (
19       <div>
20         <input name='nome' onChange={this.onChangeNome} value={this.state.valorNome} />
21         <input name='email' onChange={this.onChangeEmail} value={this.state.valorEmail} />
22       </div>
23     );
24   }
25 }
```

- Definindo o estado inicial. Como os inputs começam vazios, as duas propriedades começam como strings vazias
- Valores são passados para o atributo **value** dos inputs



# Inputs Controlados

```
1 import React from "react";
2
3 export class FormUsuario extends React.Component {
4   state = {
5     valorNome: '',
6     valorEmail: ''
7   }
8
9   onChangeNome = (event) => {
10     this.setState({valorNome: event.target.value})
11   }
12
13   onChangeEmail = (event) => {
14     this.setState({valorEmail: event.target.value})
15   }
16
17   render() {
18     return (
19       <div>
20         <input name='nome' onChange={this.onChangeNome} value={this.state.valorNome} />
21         <input name='email' onChange={this.onChangeEmail} value={this.state.valorEmail} />
22       </div>
23     );
24   }
25 }
```

- Definindo funções de atualização do estado



# Inputs Controlados

```
1 import React from "react";
2
3 export class FormUsuario extends React.Component {
4   state = {
5     valorNome: '',
6     valorEmail: ''
7   }
8
9   onChangeNome = (event) => {
10     this.setState({valorNome: event.target.value})
11   }
12
13   onChangeEmail = (event) => {
14     this.setState({valorEmail: event.target.value})
15   }
16
17   render() {
18     return (
19       <div>
20         <input name='nome' onChange={this.onChangeNome} value={this.state.valorNome} />
21         <input name='email' onChange={this.onChangeEmail} value={this.state.valorEmail} />
22       </div>
23     );
24   }
25 }
```

- Definindo funções de atualização do estado
- Elas são passadas para o atributo **onChange** dos inputs



# Inputs Controlados

```
1 import React from "react";
2
3 export class FormUsuario extends React.Component {
4   state = {
5     valorNome: '',
6     valorEmail: ''
7   }
8
9   onChangeNome = (event) => {
10     this.setState({valorNome: event.target.value})
11   }
12
13   onChangeEmail = (event) => {
14     this.setState({valorEmail: event.target.value})
15   }
16
17   render() {
18     return (
19       <div>
20         <input name='nome' onChange={this.onChangeNome} value={this.state.valorNome} />
21         <input name='email' onChange={this.onChangeEmail} value={this.state.valorEmail} />
22       </div>
23     );
24   }
25 }
```

- Definindo funções de atualização do estado
- Elas são passadas para o atributo **onChange** dos inputs
- Recebem o **event** como parâmetro



# Inputs Controlados

```
1 import React from "react";
2
3 export class FormUsuario extends React.Component {
4   state = {
5     valorNome: '',
6     valorEmail: ''
7   }
8
9   onChangeNome = (event) => {
10     this.setState({valorNome: event.target.value})
11   }
12
13   onChangeEmail = (event) => {
14     this.setState({valorEmail: event.target.value})
15   }
16
17   render() {
18     return (
19       <div>
20         <input name='nome' onChange={this.onChangeNome} value={this.state.valorNome} />
21         <input name='email' onChange={this.onChangeEmail} value={this.state.valorEmail} />
22       </div>
23     );
24   }
25 }
```

- Definindo funções de atualização do estado
- Elas são passadas para o atributo **onChange** dos inputs
- Recebem o **event** como parâmetro
- Atualizam as **propriedades** de estado relevantes
- Com os **valores** vindo do event







## Exercício 2

- Crie 3 inputs para inserir as informações de cartão de crédito(nome, número e código de segurança).
- Imprima na tela as informações digitadas pelo usuário.

**Nome: Astrodev**

**Número: 1234 5678 9101 1120**

**cvv: 400**

### Insira os dados do seu cartão

Nome:

Número:

cvv:





## Exercício 2

- Crie 3 inputs e 3 elementos de cabeçalho (h1, h2 e h3)
- Vincule o conteúdo do input com o do cabeçalho de tamanho correspondente

**Texto do Input: aaa**

**Texto do Input: bbb**

**Texto do Input: ccc**





## Exercício 3

- Crie um novo botão que, ao ser clicado, deve imprimir o valor dos inputs no console e limpar o valor de todos os campos



# Resumo

Labenu\_



# Resumo

- Componente de classe é um novo tipo de componente com **mais funcionalidades**
- Possui algumas diferenças de sintaxe:
  - `extends React.Component`
  - Recebe parâmetro `props` automaticamente
  - JSX é retornado no método `render()`
  - Funções de evento no mesmo nível que o `render()` => uso: `this.nomeDaFuncao()`



# Resumo



## Componente Funcional

```
1 import React from 'react'
2
3 export function MeuComponente(props) {
4   return (
5     <div>
6       <h1>Meu 1º componente!</h1>
7       <p>Esse é meu 1º componente</p>
8       <p>{props.texto}</p>
9     </div>
10   )
11 }
```

## Componente de Classe

```
1 import React from 'react'
2
3 export class MeuComponente extends React.Component {
4   render() {
5     return(
6       <div>
7         <h1>Meu 1º componente!</h1>
8         <p>Esse é meu 1º componente</p>
9         <p>{this.props.texto}</p>
10       </div>
11     )
12   }
13 }
```



# Resumo

- Estados são objetos onde podemos guardar dados que mudam na nossa aplicação
- Quando o dado muda, a tela é atualizada (ou seja, a função render roda novamente)
  - Criar estado: objeto `state = {}`
  - Ler estado: `this.state`
  - Atualizar estado: `this.setState()`



# Resumo



```
1 import React from "react";
2
3 export class Contador extends React.Component {
4   state = {
5     valorContador: 0
6   }
7
8   onClickSoma = () => {
9     const valorAtual = this.state.valorContador
10    const proximoValor = valorAtual + 1
11
12    this.setState({valorContador: proximoValor})
13  }
14
15  onClickSubtrai = () => {
16    this.setState({valorContador: this.state.valorContador - 1})
17  }
18
19  render() {
20    return (
21      <div>
22        <p>{this.state.valorContador}</p>
23        <button onClick={this.onClickSubtrai}>Subtrai</button>
24        <button onClick={this.onClickSoma}>Soma</button>
25      </div>
26    );
27  }
28 }
```





# Resumo

- Inputs controlados são inputs que possuem um estado
- Passamos **dois parâmetros** para o elemento `<input/>`:
  - `value`: valor do estado onde queremos guardar o dado
  - `onChange`: função de atualização desse estado
    - Nessa função, recebemos um parâmetro `event`
    - Usamos: `event.target.value`



# Dúvidas? 🤔

Labenu\_





Obrigado(a)!