Estados e Imutabilidade

Labenu_

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

Vamos ver na prática! 

Imutabilidade no React

Labenu_

Motivação 💡

- No React, não podemos alterar valores da mesma forma que costumávamos fazer no Javascript

```
let nome = "Labenu"
nome = "Lua"
```

- A única forma de alterarmos valores de variáveis é através de **estados**

Entendendo a imutabilidade na prática

- Para entender estado, vamos ver a implementação de um **contador** em React
- Variável para guardar o valor de um contador
- Número que mostra o valor do contador
- Botão que ao ser clicado, soma 1 no valor atual

Vamos ver na prática! 

Entendendo a imutabilidade na prática

```
function App() {
  let contador = 0

  const incrementarContador = () => {
    contador = contador + 1
    console.log(contador)
  }

  return (
    <div className="App">
      <h1>{contador}</h1>
      <button onClick={incrementarContador}>Incrementar</button>

    </div>
  );
}

export default App;
```

Estados

Labenu_

Estados e a Reatividade

- São as únicas variáveis que o React monitora
- Colocamos em variáveis de estados, todos os **valores que podem mudar** com o tempo
- Quando algum desses valores muda, o React renderiza a tela novamente

useState( )

Labenu_

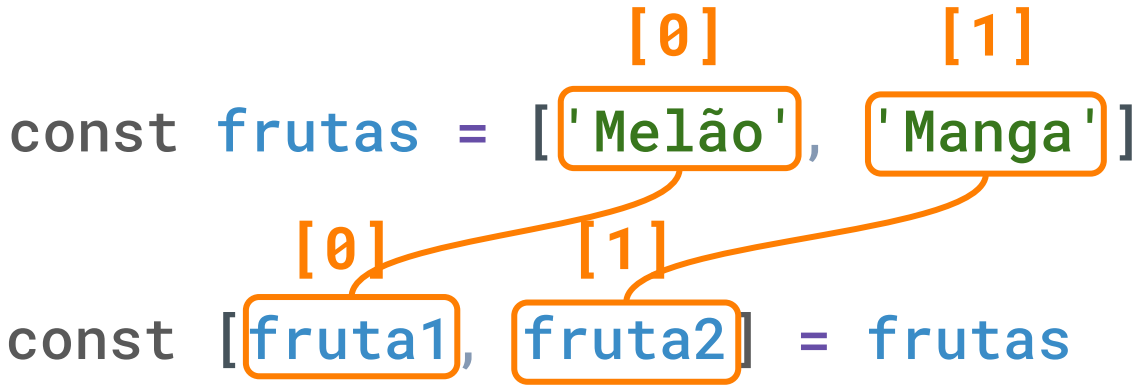
Hooks 

- useState é um hook que permite a criação de variáveis de estados em nossas aplicações

- Possibilita criarmos uma **variável de estado** em um componente funcional e permite **atualizá-la** por meio de uma função

Relembrando desestruturação

- Em Javascript, podemos utilizar destruturação, que é **alocar dentro de uma nova variável cada valor do array**
- Cada variável receberá o valor correspondente ao seu index



Relembrando retornos de funções

- Quando queremos enviar para fora de uma função mais de uma variável, podemos **retornar um array**

- Podemos acessar os valores do retorno através das posições do array

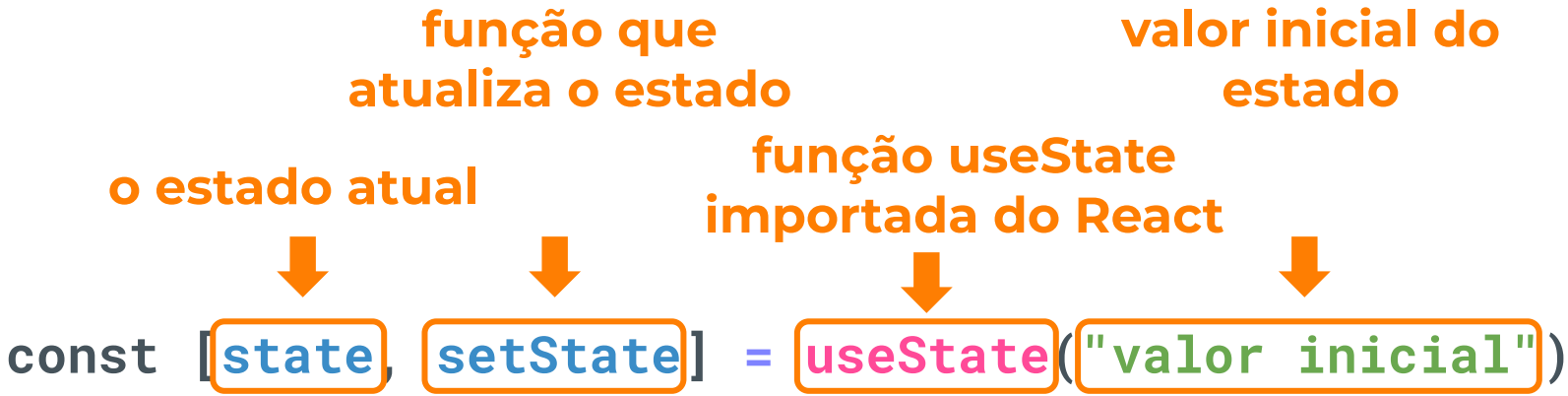
- Ou fazer uso da **desestruturação**, uma sintaxe mais simples

Sintaxe useState( ) 🗄️

- Precisamos desestruturar essa função. Ela retorna duas variáveis:

Sintaxe useState( ) 🗄️

- Precisamos desestruturar essa função. Ela retorna duas variáveis:



useState() 🗄️

- Para usar o useState, precisamos saber como: **Declarar**, **Utilizar** e **Atualizar** uma variável de estado!

1. Declarando useState( ) 🗄️

- Para **declarar** uma variável de estado, usamos a função **`useState( )`**. Ela deve ser importada do React, entre **`{chaves}`**, no topo do arquivo

```
import React, {useState} from 'react'
```

- Dentro do componente, declarar o estado:

```
const [fruta, setFruta] = useState('morango')
```

2. Atualizando useState( )

```
const [fruta, setFruta] = useState( 🍓 )
```

3. Usando uma variável de estado

- A variável de estado é usada como qualquer outra variável (não precisa de `this.state`)

- Podemos usar ela no meio do JSX, passar como props para outros componentes, etc...

```
return <div>
          <h2>Minha fruta favorita é {fruta}</h2>
       </div>
```

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	`const [state, setState] = useState("valor inicial")`
Ler estado	`{state}`
Atualizar estado	`setState()`

Exercício 1

- **Crie um estado** para guardar o valor inicial de um contador

- **Utilize** ele no JSX, altere o valor inicial e note o comportamento refletido na tela

- Faça um botão que a cada clique o valor do contador **atualize** para + 1

Inputs Controlados

Labenu_

Lidando com Inputs ⌨️

- Podemos dizer que o input do HTML já possui um estado próprio (atributo value)

- 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

```
const [inputName, setInputName] = useState("")
const [inputEmail, setInputEmail] = useState("")
```

- Essa propriedade **define** o valor (value) do input

```
<input value={inputName}/>

<input value={inputEmail}/>
```

Controlando o Input ⌨️

- Toda vez que o input **muda**, o estado deve ser **atualizado**

- Isso é feito através do evento onChange( )

```
<input
    value={inputName}
    onChange={handleInputName}
    />
```

Função onChange ⌨️

A função que passamos para o `onChange` recebe um `event` como parâmetro

```
handleInputName = (event) => {
    setInputName(event.target.value)
}
```

O `event` é um objeto que possui o valor (value) do input, basta acessar por `event.target.value`

Vamos ver na prática! 🔬

Resumo

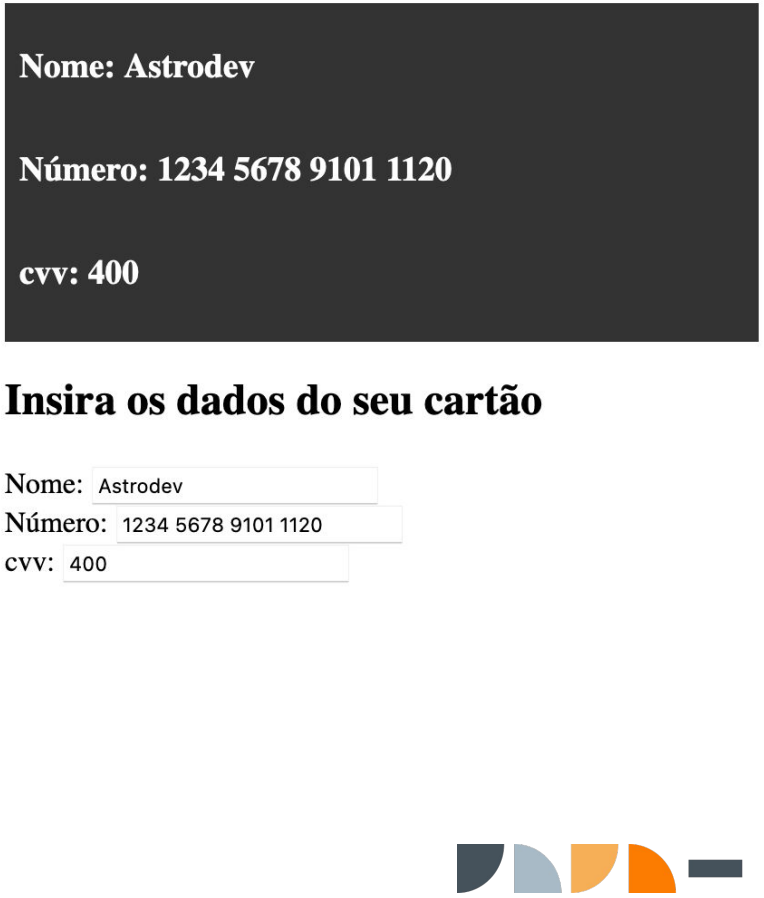
- Inputs controlados são inputs que possuem um estado

- 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`

Exercício 2

- Crie 3 inputs para inserir as informações de cartão de crédito(nome, número e código de segurança)

- Mostre na tela as informações digitadas pelo usuário



Dúvidas? 🧐

Labenu_