Hooks

as funcionalidades do componente de class no componente funcional

useState

É um hook do React que permite que você adicione estado a um componente de função. O estado é um objeto que contém dados que podem ser atualizados durante a vida útil de um componente. Quando o estado é atualizado, o React re-renderiza o componente e exibe a versão atualizada na tela.

```
import React, { useState } from "react";
//importar o useState

export default function UseState() {
  const [contador, setContador] = useState(0);
  // uma array com dois valores 1º estado atual(0), 2º função de atualização do estado
  // state={contador: 0} => setContador = setState({contador: this.state.contador + 1})
```

Para usar o **useState**, você precisa importá-lo do React e chamar a função dentro do seu componente de função. A função **useState** retorna uma matriz com dois valores: o primeiro é o estado atual e o segundo é uma função que vai ser usada para atualizar o estado.

```
return (
  <div>
    <p>Contador: {contador}</p>
    <button onClick={() => setContador(contador + 1)}>Incrementar</button>
  </div>
);
```

contador é o estado atual e setContador é a função que pode ser usada para atualizar o estado. Ao clicar no botão "Incrementar", o setContador é chamado com o novo valor do contador que é a soma de contador + 1. Isso atualiza o estado e o React re-renderiza o componente com o novo valor do contador exibido na tela.

useRef

é um hook do React que permite criar uma referência a um elemento do DOM ou a um valor que persiste durante toda a vida útil do componente. É uma maneira de armazenar dados que não precisam ser refletidos no estado do componente, mas que precisam ser acessados de forma consistente em diferentes momentos do ciclo de vida do componente.

```
import React, { useRef } from 'react';

function focaNoInput() {
  const inputRef = useRef(null);

  function handleClick() {
    inputRef.current.focus();
  }

  return (
    <div>
      <input type="text" ref={inputRef} />
      <button onClick={handleClick}>Focar</button>
    </div>
  );
}
```

Para criar uma referência usando **useRef**, você precisa importá-lo do React e chamar a função dentro do seu componente de função. A função useRef retorna um objeto com uma propriedade **current** que pode ser usada para armazenar e acessar o valor atual da referência.

Neste exemplo, a função useRef é usada para criar uma referência a um elemento de input do DOM usando a propriedade **ref.**
A referência é inicializada com null e quando clicamos ele usa a referência para chamar a função focus.

useEffect

É um hook do React que nos permite lidar com efeitos, ou melhor dizendo, eventos ou atualizações na execução do nosso código. O useEffect pode ser considerado um hook de ciclo de vida, muito parecido com o componentDidMount, componentDidUpdate e componentWillUnMount.

```
import React, { useEffect } from 'react';

useEffect( ) {
    () =>{
       função a ser executada no useEffect
    }. [array de dependências]
}
  return (
    <div>
      <h1> Estamos aprendendo useEffect </h1>
    </div>
  );
}
```

Isso significa que, podemos executar funções acordo com o estado de vida do nosso component: quando ele é montado na página, quando ele sofre atualizações e quando ele é desmontado da página ou sua execução chega ao fim. Mas não se preocupe com isso agora, entenderemos melhor esse hook tão poderoso.

Entendendo o useEffect

Para funcionar, o useEffect possui dois argumentos: uma função de callback, obrigatória, onde dizemos ao useEffect o que fazer, passando uma instrução ou chamando uma função pré-criada, E a array que pode ou não receber uma dependência (que pode ser lida como condições para o useEffect ser executado).

```
import React, { useEffect, useState } from 'react';
export function App( ) {
const [state, useState] = useState(0)
useEffect( ) {
    () =>{
       document.body.style.backgroundColor =
"darkred":
.
    }. [state]
}
  return (
   <div>
    <h1>
      Estamos aprendendo useEffect
    </h1>
   </div>
  );
}
```

Quando a array de dependências está vazia, o useEffect será ativado assim que nosso componente for montado no navegador. Caso contrário, eu preciso dizer a ele quando deve ser executado, passando como dependência, geralmente um state, uma props, uma let, que quando sofre alguma alteração, ativa o efeito.