



Eventos e State

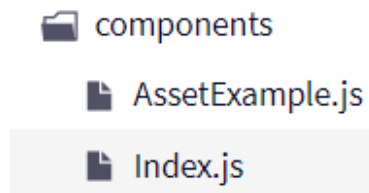
Ewerton J. Silva

ewerton.silva41@etec.sp.gov.br

Criando componente principal

Em um novo projeto dentro da pasta “components” crie um novo arquivo com o nome “Index.js” e insira o código apresentado ao lado.

Este componente vai retornar uma View com um Text dentro.

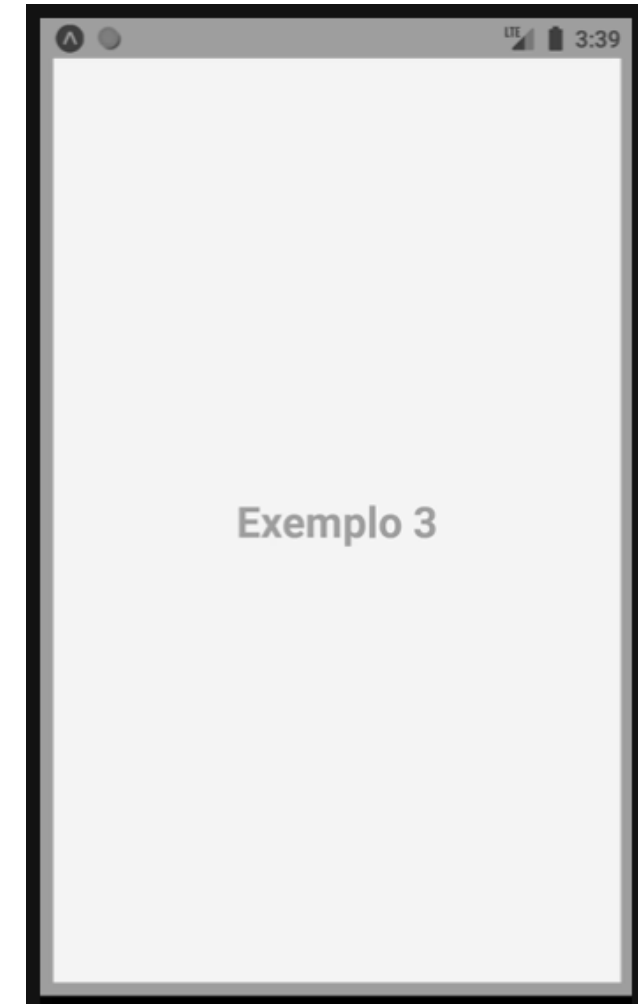


```
1  import * as React from 'react';
2  import { View, Text, StyleSheet } from 'react-native';
3
4  export default function Index() {
5    return(
6      <View style={styles.container} >
7        <Text style={styles.paragraph} >
8          Exemplo 3
9        </Text>
10     </View>
11   );
12 }
13
14 const styles = StyleSheet.create({
15   container: {
16     flex: 1,
17     justifyContent: 'center',
18     backgroundColor: '#F5F5F5',
19     padding: 8,
20   },
21   paragraph: {
22     margin: 24,
23     fontSize: 26,
24     fontWeight: 'bold',
25     textAlign: 'center',
26     color: '#9E9E9E',
27   },
28 });
```

Em “App.js”, vamos importar o componente “Index.js” e utilizá-lo dentro do “App.js”. Em nossas aulas iremos manter sempre o padrão dos slides 2 e 3 sempre que iniciarmos um novo projeto, limpando o código e chamando um componente criado para realizarmos nossos exemplos.

Deixe o código de “App.js” conforme o exemplo ao lado, aqui iremos retornar uma “View” com o componente criado anteriormente.

```
1  import * as React from 'react';
2  import { View, StyleSheet } from 'react-native';
3  import Constants from 'expo-constants';
4
5  import Index from './components/Index';
6
7  export default function App() {
8    return (
9      <View style={styles.container}>
10        <Index />
11      </View>
12    );
13  }
14
15  const styles = StyleSheet.create({
16    container: {
17      flex: 1,
18      justifyContent: 'center',
19      paddingTop: Constants.statusBarHeight,
20      backgroundColor: '#9E9E9E',
21      padding: 8,
22    },
23  });
```



Criando um botão

No componente “Index.js” logo abaixo do texto inserido anteriormente vamos colocar um botão.

A forma mais comum de uso de botões é através dos componentes

“TouchableOpacity” ou

“Touchablehighlight” em nosso exemplo iremos utilizar o

“TouchableOpacity”

<https://reactnative.dev/docs/touchablehighlight>
<https://reactnative.dev/docs/touchableopacity>

```
1  import * as React from 'react';
2  import { View, Text, StyleSheet, TouchableOpacity } from 'react-native';
3
4  export default function Index() {
5    return(
6      <View style={styles.container} >
7        <Text style={styles.paragraph} >
8          Exemplo 3
9        </Text>
10
11        <TouchableOpacity
12          onPress={() => alert('Olá, mundo!')}
13          style={styles.button}
14        >
15          <Text style={styles.textButton} >
16            Diga "Olá, mundo!"
17          </Text>
18        </TouchableOpacity>
19
20      </View>
21    );
22  }
```



TouchableOpacity

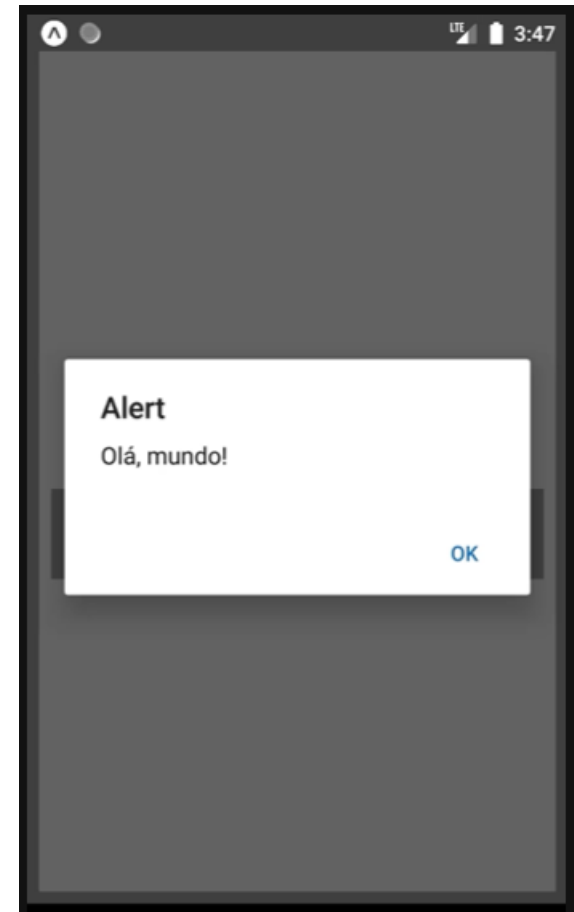
Este componente funcionar como um container e permite que tudo que está dentro dele responda a um determinado evento, em nosso exemplo ao declara-lo definimos duas propriedades, o evento “onPress” que reponde através de uma arrow function sempre que clicarmos sobre algo que faz parte deste container e a estilização que será aplicada.

Dentro do componente inserimos um “Text” que serve para apresentar o texto, aqui poderíamos inserir imagens, ícones...

```
11      <TouchableOpacity
12      →  onPress={() => alert('Olá, mundo!')}
13      →  style={styles.button}
14      >
15        <Text style={styles.textButton} >
16          Diga "Olá, mundo!"
17        </Text>
18      </TouchableOpacity>
```

Insira o código para estilizar os componentes que irão representar o botão e teste nosso exemplo.


```
24  const styles = StyleSheet.create({
25    container: {
26      flex: 1,
27      justifyContent: 'center',
28      backgroundColor: '#F5F5F5',
29      padding: 8,
30    },
31    paragraph: {
32      margin: 24,
33      fontSize: 26,
34      fontWeight: 'bold',
35      textAlign: 'center',
36      color: '#9E9E9E',
37    },
38    button: {
39      backgroundColor: '#9E9E9E',
40      height: 60,
41      justifyContent: 'center',
42    },
43    textButton: {
44      fontSize: 20,
45      color: '#fff',
46      textAlign: 'center',
47    }
48  });
```



O que é uma arrow function?

Vamos começar entendendo o porque estamos utilizando este recurso, na imagem da esquerda temos a forma correta de executar uma ação ao clicar sobre o “TouchableOpacity”, a imagem do lado direito representa a forma geralmente utilizada em outras linguagens de programação, onde no evento precisamos chamar apenas o comando a ser executado.

Teste o comando da imagem a direita e veja se o aplicativo devolve uma mensagem sempre que clicarmos no botão?



```
<TouchableOpacity
  onPress={() => alert('Olá, mundo!')}
  style={styles.button}
>
```



```
<TouchableOpacity
  onPress={alert('Olá, mundo!')}
  style={styles.button}
>
```

Renderização de tela

A programação em React Native é baseada em componentes funcionais, o que significa que todo componente pode ser renderizado através do retorno de uma função, assim a tela só será atualizada (exibindo novos valores) caso seja necessário retornar algum valor novo, isso é feito para otimizar o aplicativo!

Dessa forma quando queremos executar uma ação em um evento devemos inserir nele a chamada de uma função e não a execução de um comando, pois se executarmos um comando diretamente, ele só será acionado quando ocorrer uma nova renderização de tela.



```
export default function Index() {  
  return(  
    <View style={styles.container} >  
      <Text style={styles.paragraph} >  
        Exemplo 3  
      </Text>  
    </View>  
  );  
}
```


Uso de funções

Quando não utilizamos função para execução foi possível identificar que a mensagem aparece apenas na renderização e o botão não fazia nada quando clicávamos nele.

Observe nas três imagens abaixo as formas possíveis de executar funções em um evento, a primeira é a forma tradicional, semelhante ao que fazemos quando criamos um componente, a segunda é uma notação resumida que só deve ser utilizada se o retorno da função couber em uma única linha de código, dessa forma não é necessário o uso do comando “return”.

```
<TouchableOpacity
  onPress={
    function Mensagem() {
      return(
        alert('Olá, mundo!')
      );
    }
  }
  style={styles.button}
>
  <Text style={styles.textButton} >
    Diga "Olá, mundo!"
  </Text>
</TouchableOpacity>
```

```
<TouchableOpacity
  onPress={
    function Mensagem() {
      alert('Olá, mundo!');
    }
  }
  style={styles.button}
>
  <Text style={styles.textButton} >
    Diga "Olá, mundo!"
  </Text>
</TouchableOpacity>
```

Sintaxe de funções

Funções são blocos de código que permitem programar instruções que possam ser chamadas através de seu nome, ou funções não nomeadas são chamadas de funções anônimas. Em JavaScript funções podem ser utilizadas para executar determinadas instruções com ou sem o retorno de valores.

```
function [nome]([parametro1[, parametro2[, ..., parametroN]]]) {  
    declarações  
}
```

Exemplo de sintaxe de uma função normal

Arrow Functions

São funções anônimas com chamada simplificada, podemos utilizá-la sempre que necessário a execução de uma função ou comando através de um evento.

```
<TouchableOpacity  
  onPress={() => alert('Olá, mundo!')}  
  style={styles.button}  
>  
  <Text style={styles.textButton} >  
    Diga "Olá, mundo!"  
  </Text>  
</TouchableOpacity>
```



https://www.w3schools.com/js/js_arrow_function.asp

Função anônima X arrow function

```
function(parametro) {  
    return argumentos  
}
```

Exemplo de sintaxe de uma função anônima

```
(parametro1, parametro2, ..., parametroN) => expressão
```

Exemplo de sintaxe de uma arrow function com parâmetro

```
() => expressão
```

Exemplo de sintaxe de uma arrow function sem parâmetro

Antes de prosseguirmos confira se o “TouchableOpacity” está como na imagem abaixo, onde no evento “onPress” utilizamos uma “Arrow Function” para executar a instrução quando clicamos no botão.

```
<TouchableOpacity
  onPress={() => alert('Olá, mundo!')}
  style={styles.button}
>
  <Text style={styles.textButton} >
    Diga "Olá, mundo!"
  </Text>
</TouchableOpacity>
```

Criando um contador

Agora vamos criar uma variável e em seguida uma função que execute o comando que vai incrementar um número sempre que ela for executada, observe que nesta função não utilizamos a palavra “return”, pois ela não vai apresentar nenhum resultado, apenas modificar algo que já existe!

```
1  import * as React from 'react';
2  import { View, Text, StyleSheet, TouchableOpacity } from 'react-native';
3
4  export default function Index() {
5
6      → var numero = 0;
7
8      {
9          function AddNumber() {
10             numero++;
11         }
12
13         return(
```

A parte referente ao contador será renderizada dentro da “View” do componente “Index.js”, logo após o botão criado para exibir “Olá, mundo!”.

Iremos agrupar tudo dentro de uma “View”, onde teremos um “Text” que vai apresentar o valor da variável, observe que para exibir o valor da variável o nome dela deve estar entre “{}”, caso contrário seria apresentada a palavra “numero” na tela.

Com o “TouchableOpacity” vamos definir um botão que no evento “onPress” executa a função “AddNumber” esta chamada com “()” após o nome, pois não passamos parâmetro em sua chamada.

```
12 return(  
13   <View style={styles.container} >  
14     <Text style={styles.paragraph} >  
15       Exemplo 3  
16     </Text>  
17  
18     <TouchableOpacity  
19       onPress={() => alert('Olá, mundo!')}  
20       style={styles.button}  
21     >  
22       <Text style={styles.textButton} >  
23         Diga "Olá, mundo!"  
24       </Text>  
25     </TouchableOpacity>  
26  
27     <View style={styles.counter}>  
28       <Text style={styles.textCounter}>{numero}</Text>  
29       <TouchableOpacity  
30         onPress={() => AddNumber()}  
31         style={styles.button}  
32       >  
33         <Text style={styles.textButton} >  
34           + 1  
35         </Text>  
36       </TouchableOpacity>  
37     </View>  
38   </View>  
39 </View>  
40 );
```

Não se esqueça de aplicar os estilos em cada um dos componentes que inserimos referente ao exemplo do contador, teste o programa e veja o que acontece...



```
43 const styles = StyleSheet.create({
44   container: {
45     flex: 1,
46     justifyContent: 'center',
47     backgroundColor: '#F5F5F5',
48     padding: 8,
49   },
50   paragraph: {
51     margin: 24,
52     fontSize: 26,
53     fontWeight: 'bold',
54     textAlign: 'center',
55     color: '#9E9E9E',
56   },
57   button: {
58     backgroundColor: '#9E9E9E',
59     height: 60,
60     justifyContent: 'center',
61   },
62   textButton: {
63     fontSize: 20,
64     color: 'fff',
65     textAlign: 'center',
66   },
67   counter: {
68     borderWidth: 4,
69     borderColor: '#9E9E9E',
70     padding: 8,
71     marginTop: 8,
72     justifyContent: 'center',
73   },
74   textCounter: {
75     fontSize: 32,
76     color: '#424242',
77     textAlign: 'center',
78     fontWeight: 'bold',
79     padding: 8,
80   }
81 });
```


...não funcionou! Funcionou sim, acontece que só não é possível visualizar seu funcionamento, pois em nenhum momento a tela foi renderizada, nossa função de incrementar número não retorna um valor que é apresentado em tela, ela apenas incrementa a variável.

É possível verificar se a função está correta ao acrescentar um comando que apresente o valor da variável, poderíamos utilizar o “alert”, mas vou mostrar outra possibilidade o “console.log”.

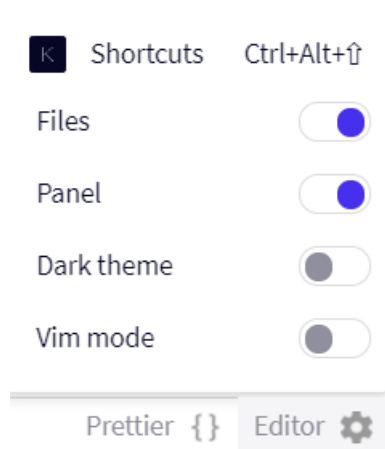
Adicione o comando apontado abaixo na função!

```
8      function AddNumber() {  
9          numero++;  
10     → console.log(numero);  
11     }
```



Mensagens no console ajudam muito o programador a entender o fluxo de dados em um aplicativo, para visualizar as mensagens de console, é preciso ativar sua visualização na tela.

Para isso clique em “Editor” e ative a opção “Panel”



Agora sempre que clicamos no botão “+ 1” a mensagem retornada pelo console será apresentada em “LOGS”, em nosso exemplo é exibido o valor da variável “numero”.



PROBLEMS LOGS

Android SDK built for x86: 1
Android SDK built for x86: 2
Android SDK built for x86: 3


State

Bom, sabemos que a função funciona, porém o valor não aparece em tela!

Para que seja possível visualizar os valores atualizados será necessário utilizar um recurso que permita identificar quando a variável muda seu valor, forçando uma renderização.

Este recurso chama-se “State”, para utiliza-lo devemos importa-lo conforme apontado na linha 1.

```
1  import React, { useState } from 'react';  
2  import { View, Text, StyleSheet, TouchableOpacity } from 'react-native';
```



State

Após importar o “useState” devemos utiliza-lo como se fosse uma variável, o detalhe é que definimos um “State” como uma constante que é um array de duas posições, sendo uma delas utilizada para acessar o valor armazenado e outra para modificar o valor armazenado, acrescentado a instrução “set” ao nome do primeiro valor declarado, estas dentro de um “[]” separadas por uma “,”.

Também devemos definir um valor inicial, utilizando um sinal de “=” seguido do comando “useState” com o valor inicial entre “()”.

```
1  import React, { useState } from 'react';
2  import { View, Text, StyleSheet, TouchableOpacity } from 'react-native';
3
4  export default function Index() {
5
6      const [numero, setNumero] = useState(0);
```



Hooks

Na verdade para resolver este problema existe duas soluções, uma para quem programa utilizando “Classes” e outra para quem programa apenas “Componentes Funcionais”, bom nós estamos no segundo caso, não vamos trabalhar com classes, pois desde 2018 existe uma forma mais simples de desenvolver com React Native e é desta forma que estamos aprendendo!

O “State” é uma das funcionalidades acrescentada junto com um grupo de outros recursos que permite você use diversos recursos do React sem escrever uma classe, tudo adicionado a partir da versão 16.8.

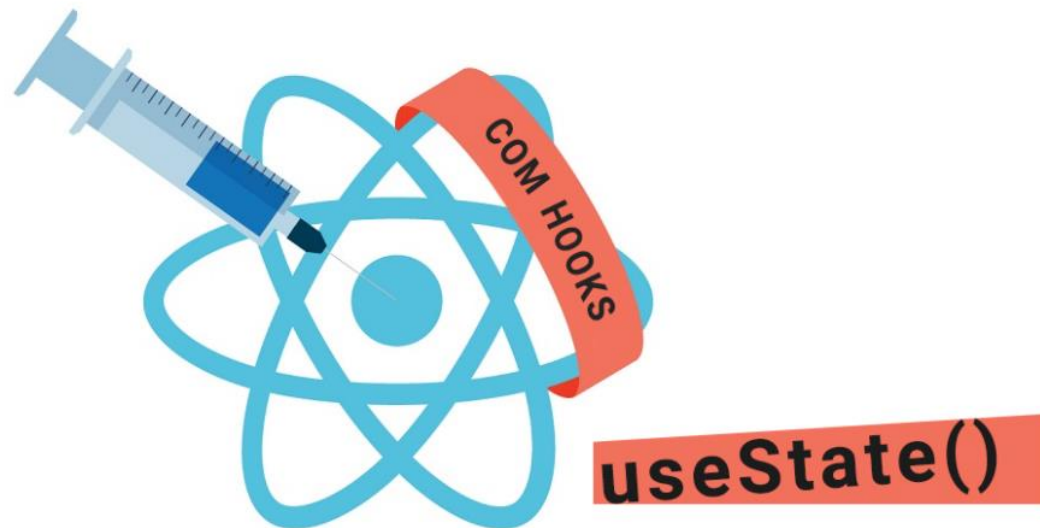
<https://pt-br.reactjs.org/docs/hooks-intro.html>

<https://blog.rocketseat.com.br/react-hooks/>

Hooks

Saiba mais sobre componentes funcionais e classes lendo os textos:

- [Componentes React: componentes de classe e funcional sem estado](#)
- [React Native Basics: Componentes Funcionais vs Classes](#)



<https://sujeitoprogramador.com/usando-o-hook-de-estados-usestate/>

Manipulando um State

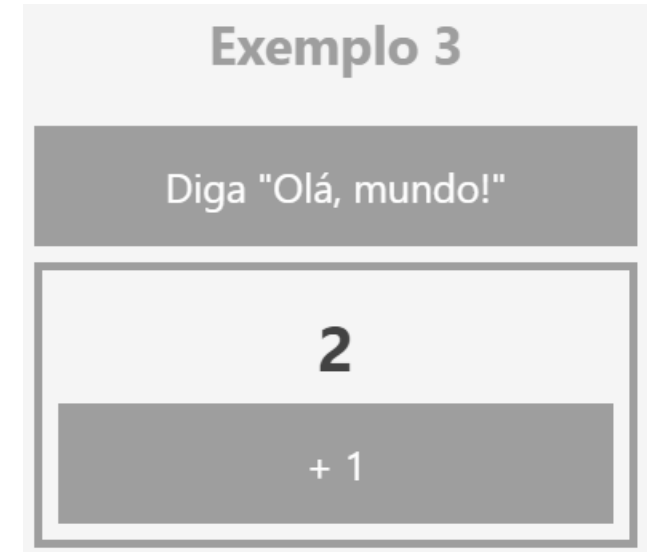
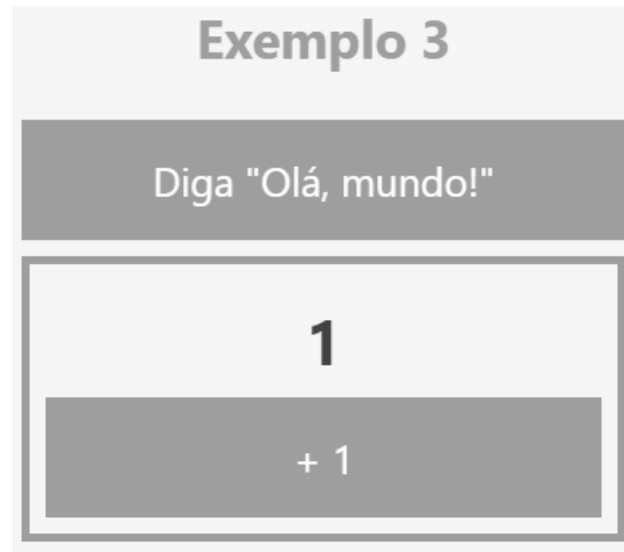
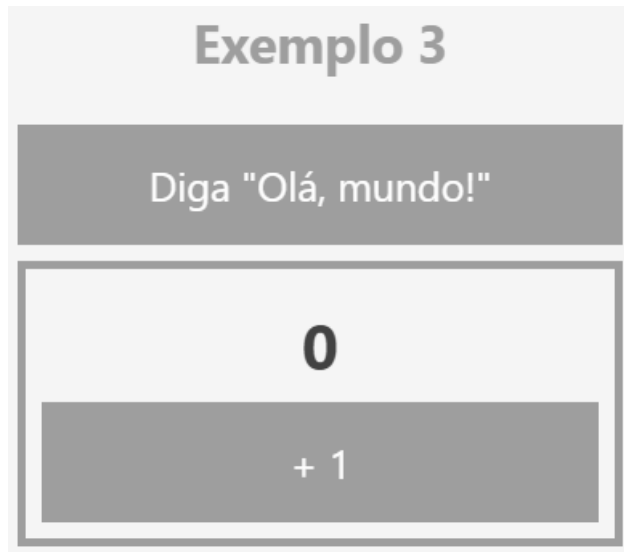
Diferente de uma variável comum, quando alteramos o valor de um “State” forçamos a renderização da tela, assim o valor incrementado sempre será exibido, desde que utilizemos o recurso da forma correta.

Na função criada para alterar o valor do “State” utilizamos o comando “set” + “NomeState” e entre “()” passamos o novo valor.

```
1  import React, { useState } from 'react';
2  import { View, Text, StyleSheet, TouchableOpacity } from 'react-native';
3
4  export default function Index() {
5
6      const [numero, setNumero] = useState(0);
7
8      function AddNumber() {
9          setNumero(numero + 1);
10     }
11
12     return(
```

Aqui o comando de incremento não funciona, pois ele tentaria alterar o valor de “numero” que por ser uma constante não aceita alteração.

Teste o programa



Atividade

- Inserir botão para zerar contador
- Botão para decrementar
- Manter o incrementar
- Compartilhe o link e o print da tela em um arquivo de texto.



Referências

- <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Fun%C3%A7%C3%B5es>
- <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/function>
- [https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Functions/Arrow functions](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Functions/Arrow_functions)
- <https://reactnative.dev/docs/touchablehighlight>
- <https://reactnative.dev/docs/touchableopacity>
- [https://www.w3schools.com/js/js arrow function.asp](https://www.w3schools.com/js/js_arrow_function.asp)
- <https://pt-br.reactjs.org/docs/faq-functions.html>
- <https://pt.stackoverflow.com/questions/9936/como-funcionam-fun%C3%A7%C3%B5es-an%C3%B4nimas>
- <https://blog.rocketseat.com.br/react-hooks/>
- <https://pt-br.reactjs.org/docs/hooks-intro.html>