1. Array.map(React Native)

```
//Array com números de 1 a 5
const numeros = [1, 2, 3, 4, 5];

//Estrutura de repetição que percorre todo o array e multiplica cada um dos reg
for(let i = 0; i < numeros.length; i++) {
    numeros[i] = numeros[i] * 2
}
console.log(numeros)
```

Código 1. Forma tradicional

O resultado do console pode ser visto na **Figura 1**.

Figura 1. Forma tradicional

```
//Array com números de 1 a 5
const numeros = [1, 2, 3, 4, 5];

//Função (arrow function) que percorre todo o array e multiplica cada um dos re
const dobro = numeros.map((numero) => numero * 2);

console.log(dobro);
```

Código 2. Forma simplificada

O resultado do console pode ser visto na **Figura 2**.

Figura 2. Forma simplificada

Nesse caso, o `Array.map` executa a multiplicação para cada um dos itens do array, sem a necessidade de acessar o array diretamente ou criar uma estrutura de repetição manualmente.

Para utilizar o map, chamamos a função map a partir do array desejado:

```
const numeros = [1, 2, 3, 4, 5]
const dobro = numeros.map((numero) => numero * 2)
```

Dentro do map passamos a função que será executada. Você pode passar o nome da função, ou seu código diretamente:

```
function dobrarNumero(numero) {
    return numero * 2;
}

const numeros = [1, 2, 3, 4, 5];
const dobro = numeros.map((numero) => dobrarNumero(numero))
console.log(dobro)
```

```
const numeros = [1, 2, 3, 4, 5]
const dobro = numeros.map((numero) => numero * 2)
```

O parâmetro passado para a função é o identificador de cada item do array:

```
const numeros = [1, 2, 3, 4, 5]
const dobro = numeros.map((numero) => numero * 2)
```

Nesse exemplo chamamos de numero, mas esse parâmetro pode ter qualquer nome.

Em seguida, através de uma arrow function, multiplicamos o parâmetro passado por 2, ou seja, multiplicamos cada item do array por 2.

```
const numeros = [1, 2, 3, 4, 5]
const dobro = numeros.map((numero) => numero * 2)
```

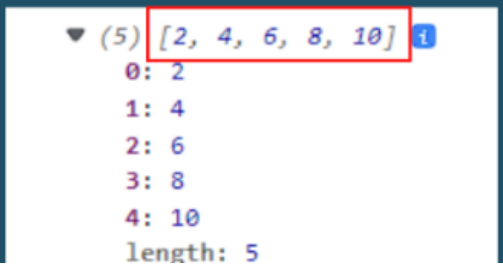
O map irá percorrer um a um cada item do array e executar o código escrito dentro do map.

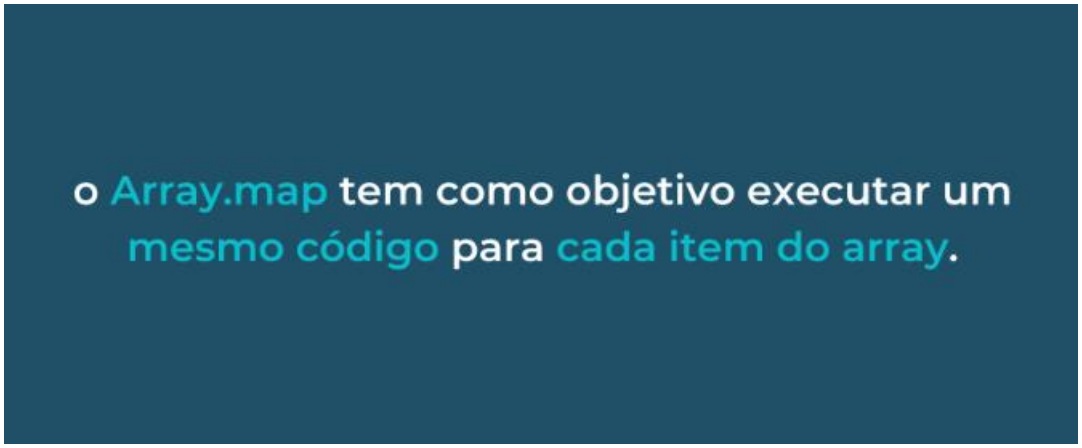
```
const numeros = [1, 2, 3, 4, 5]
const dobro = numeros.map((numero) => numero * 2)
```

1 * 2 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10

Se imprimirmos a constante dobro na tela, seu resultado será o array: [2, 4, 6, 8, 10]

```
const numeros = [1, 2, 3, 4, 5]
const dobro = numeros.map((numero) => numero * 2)
```





Por que é útil?

A utilidade do Array.map no React Native está em permitir ao desenvolvedor utilizar uma coleção de dados para exibir componentes na tela.

Isso faz com que o desenvolvedor não precise alterar o código caso um novo item tenha que ser exibido, bastando adicioná-lo à sua coleção de dados.

Código 4.

```
export default function ListaCarros() {
return (
        <ItemCarro fabricante="Nissan" modelo="Skyline-GTR"} />
        <ItemCarro fabricante="Mitsubishi" modelo="Lancer Evolution VIII"} />
        <ItemCarro fabricante="Audi" modelo="Skyline-GTR"} />
        <ItemCarro fabricante="Volkswagen" modelo="Golf GTi"} />
        <ItemCarro fabricante="Nissan" modelo="350z"} />
        <ItemCarro fabricante="Honda" modelo="Civic"} />
        <ItemCarro fabricante="Nissan" modelo="240SX"} />
    );
}
```

No **Código 5** vemos um código mais enxuto.

```
export default function ListaCarros() {
listacarros.map((carro, index) => {
return (
        <ItemCarro key= fabricante={carro.fabricante} modelo={carro.modelo} />
    );
})}
}
```

Código 5. Código enxuto

Array.map no React Native

Por que é útil?

Vimos que o Array.map consegue executar um mesmo código para múltiplos itens, mas qual sua utilidade dentro do React Native?

No React Native, o Array.map pode ser utilizado para repetir a exibição de um componente com base em uma coleção de dados:

```
const bancos = [
  {
    "codigo": "001",
    "nome": "Banco do Brasil S.A."
  },
  {
    "codigo": "003",
    "nome": "Banco da Amazônia S.A."
  },
  {
    "codigo": "004",
    "nome": "Banco do Nordeste do Brasil S.A."
  },
  {
    "codigo": "007",
    "nome": "Banco Nacional de Desenvolvimento Econômico e Social BNDES"
  },
```

Se fossemos fazer sem o Array.map,
o código ficaria assim:

```
<ScrollView>
  <ItemBanco codigo={listabancos[0].codigo} nome={listabancos[0].nome} />
  <ItemBanco codigo={listabancos[1].codigo} nome={listabancos[1].nome} />
  <ItemBanco codigo={listabancos[2].codigo} nome={listabancos[2].nome} />
  <ItemBanco codigo={listabancos[3].codigo} nome={listabancos[3].nome} />
</ScrollView>
```

Sem Array.Map

Note que tivemos que repetir o componente ItemBanco 4 vezes para exibir os 4 itens na tela.

Já com Array.map, podemos criar um código muito mais simples reaproveitando o componente Item.

```
listabancos.map((banco, index) => {
  return (
    <ItemBanco key={index} codigo={banco.codigo} nome={banco.nome} />
  );
})}
```

Com Array.Map

Se compararmos os dois códigos podemos reparar que com Array.map não temos que nos preocupar com o número de itens a ser exibido.

```
<ScrollView>
  <ItemBanco codigo={listabancos[0].codigo} nome={listabancos[0].nome} />
  <ItemBanco codigo={listabancos[1].codigo} nome={listabancos[1].nome} />
  <ItemBanco codigo={listabancos[2].codigo} nome={listabancos[2].nome} />
  <ItemBanco codigo={listabancos[3].codigo} nome={listabancos[3].nome} />
</ScrollView>
```

Sem Array.Map

```
listabancos.map((banco, index) => {
  return (
    <ItemBanco key={index} codigo={banco.codigo} nome={banco.nome} />
  );
})
```

Ou seja, ao adicionar um item na coleção, ele será automaticamente exibido:

```
const bancos = [
  {
    "codigo": "001",
    "nome": "Banco do Brasil S.A."
  },
  {
    "codigo": "003",
    "nome": "Banco da Amazônia S.A."
  },
  {
    "codigo": "004",
    "nome": "Banco do Nordeste do Brasil S.A."
  },
  {
    "codigo": "007",
    "nome": "Banco Nacional de Desenvolvimento Econômico e Social BNDES"
  },
  {
    "codigo": "010",
    "nome": "Credicoamo Crédito Rural Cooperativa"
  },
```

Com Array.map podemos exibir componentes de forma dinâmica.

Em resumo, se a coleção de dados mudar, a exibição também muda automaticamente.

```
const bancos = [
  {
    "codigo": "001",
    "nome": "Banco do Brasil S.A."
  },
  {
    "codigo": "003",
    "nome": "Banco da Amazônia S.A."
  },
  {
    "codigo": "004",
    "nome": "Banco do Nordeste do Brasil S.A."
  },
]
```

```
listabancos.map((banco, index) => {
    return (
        <ItemBanco key= codigo={banco.codigo} nome={banco.nome} />
    );
 })}
```

Código 7. Trecho do Componente ListaBancos usado na aula

2. Array.filter

Nesta aula iniciaremos relembrando o que vimos sobre `Array.filter` no módulo de JavaScript.

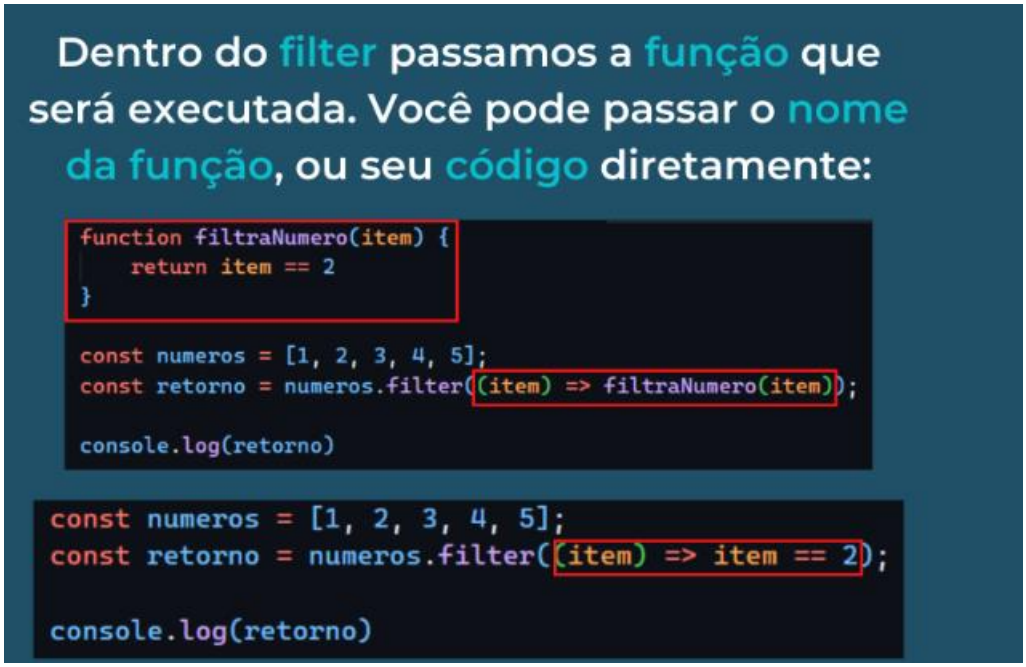
O objetivo é recordar como é o funcionamento do array.filter antes de irmos direto para sua sintaxe, uma vez que seu funcionamento é o mesmo no JavaScript e no React Native. Confira no flow:

Array.filter

Relembrando

Vimos no módulo de JavaScript que o Array.filter é uma função nativa de array, utilizada para filtrar conteúdo de uma coleção de dados.

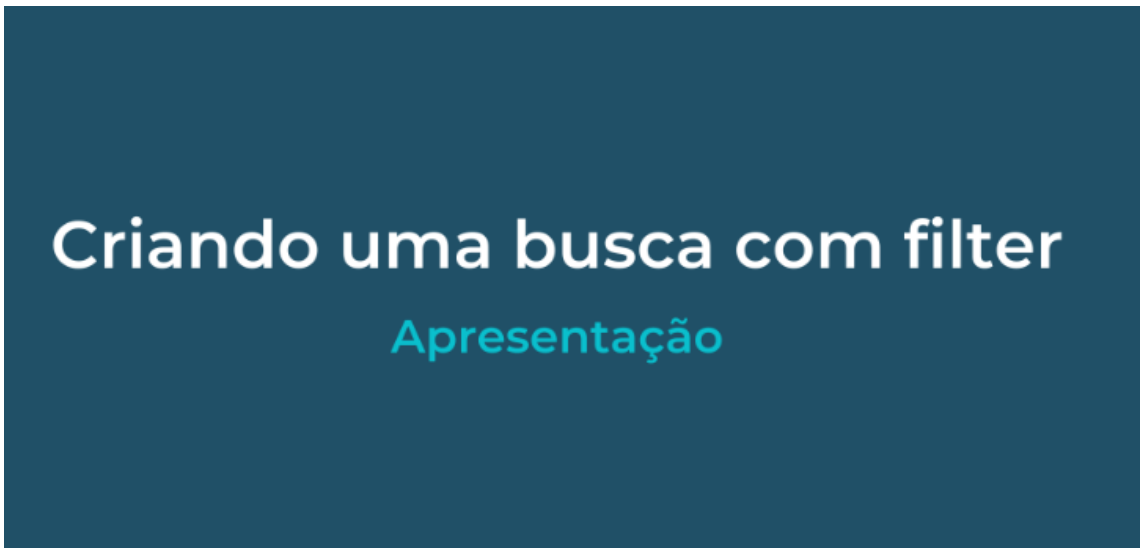
```
const numeros = [1, 2, 3, 4, 5];
const retorno = numeros.filter((item) => item == 2);

console.log(retorno)
```



3. Criando uma busca com Array.filter

O exemplo que usaremos é uma aplicação simples que lista diversos bancos e permite ao usuário fazer buscas pelo nome do banco, como mostra a Figura 1**.**

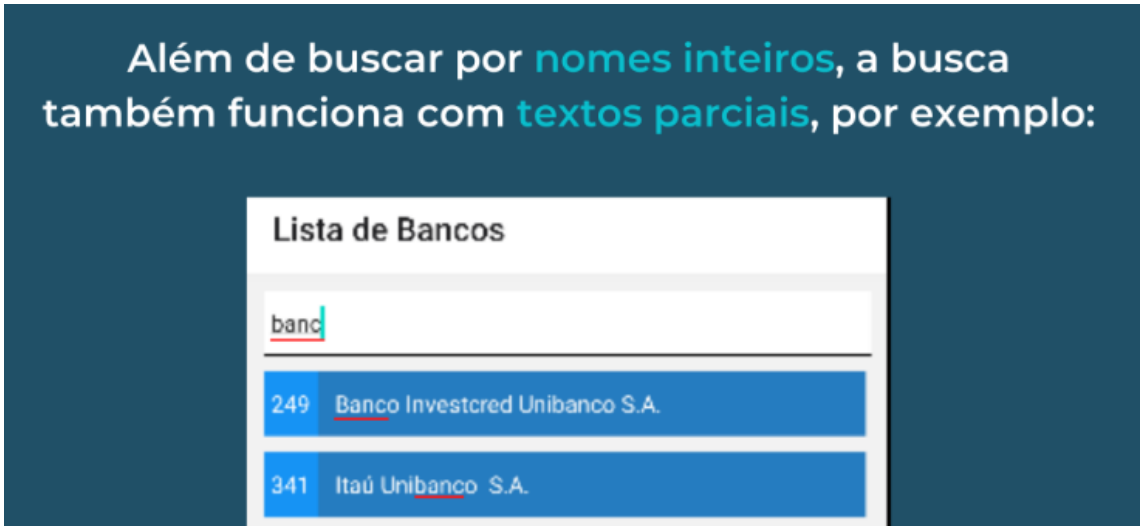
Figura 1. App de listagem de bancos



O exemplo desse flow é uma aplicação que lista e permite buscar por bancos.

O App lista diversos bancos a partir de uma coleção de dados e os exibe com uso do array.map:

O App conta também com um campo de busca, onde o usuário pode digitar o que quiser para buscar:




```
const bancos = [
  {
    codigo: "001",
    nome: "Banco do Brasil S.A."
  },
  {
    codigo: "003",
    nome: "Banco da Amazônia S.A."
  },
  {
    codigo: "004",
    nome: "Banco do Nordeste do Brasil S.A."
  },
  {
    codigo: "007",
    nome: "Banco Nacional de Desenvolvimento Econômico e Social BNDES"
  }
]
export default bancos;
```

Código 1. dados/bancos.js (conteúdo parcial)

```
import React from 'react'
import { View, Text } from 'react-native'
import styles from './styles'

export default function ItemBanco({codigo, nome}) {
    return (
        <View style={styles.Item}>
            <Text style={styles.Ano}></Text>
            <Text style={styles.Texto}></Text>
        </View>
    )
}
```

Código 2. ItemBanco/index.js

```jsx
import React, { useState } from "react";
import { ScrollView, View, TextInput } from 'react-native'

import ItemBanco from "../ItemBanco";

import styles from "./styles";
import bancos from "../../dados/bancos";

export default function ListaBancos() {

  const [textoBusca, setTextoBusca] = useState("");
  const [listaBancos, setListaBancos] = useState(bancos)

  function handleBusca(termoDigitado) {
    setTextoBusca(termoDigitado)

    const resultadoBusca = bancos.filter((banco) => banco.nome.toLowerCase().in
    setListaBancos(resultadoBusca)
  }

  return (
    <View style={styles.Container}>

      <TextInput
        style={styles.CampoBusca}
        placeholder="Busca"
        value=
        onChangeText={(termoDigitado) => handleBusca(termoDigitado)}
      />

      <ScrollView>
        {
          listaBancos.map((banco) => {
            return (
              <ItemBanco
                codigo={banco.codigo}
                nome={banco.nome}
                key={banco.codigo}
              />
            )})
        }
      </ScrollView>
    </View>
  );
}
```

Código 3. ListaBancos/index.js

agora que você já viu o código da aplicação, entenda no flow seguinte como ele funciona:

Criando uma busca com filter

Funcionamento do App em detalhes

O App inicia mostrando uma listagem de bancos.

Os dados dessa listagem vêm da importação de uma coleção de dados.

```
import bancos from "../../dados/bancos";
```

ListaBancos/index.js