

Callback e Funções de Arrays

Labenu_



O que vamos ver hoje?

- O que é Callback
- Funções de arrays
 - map
 - filter



Callback

Labenu_



Pequena revisão de Funções

- Uma função é um trecho de código **isolado** que desejamos **reutilizar**
- Pode receber (ou não) valores de **entrada** (inputs) que serão usadas dentro da função
- Pode retornar (ou não) uma **saída** (output / resposta), que deve ser **guardada** em uma variável para usarmos em outro momento do código



Pequena revisão de Funções

Declaração de função

```
1 function somaNumeros (num1, num2) {  
2     return num1 + num2  
3 }
```

Expressões de função

```
1 let somaNumeros = function(num1, num2) {  
2     return num1 + num2  
3 }
```

```
1 let somaNumeros = (num1, num2) => {  
2     return num1 + num2  
3 }
```



Callback 📞

FUNÇÕES esperam receber **VARIÁVEIS** como entrada

FUNÇÕES podem ser atribuídas à **VARIÁVEIS**

FUNÇÕES podem receber **FUNÇÕES** como entrada



Callback - Exemplo 📞

- A função abaixo verifica se um número é divisível por 2. Se ele for, realiza a divisão. Senão, não faz nada

```
1 const verificaPar = (numero) => {  
2  
3   if (numero % 2 === 0){  
4     const resultado = numero / 2  
5   }  
6 }
```



Callback - Exemplo 📞

- Caso o número seja par, queremos imprimir uma mensagem. Podemos criar uma função específica responsável por esta mensagem:

```
1 const imprimeMensagem = (valor) => {  
2     console.log("O resultado da sua conta é," valor)  
3 }
```



Callback - Exemplo 📞

- Para que a função seja executada apenas quando o número for par, podemos recebê-la como parâmetro na **verificaPar** e usar apenas neste caso

```
const verificaPar = (numero, imprimir) => {  
  if(numero % 2 === 0){  
    const resultado = numero/2  
    imprimir(resultado)  
  }  
}
```



Callback - Exemplo 📞

```
const verificaPar = (numero, imprimir) => {  
  if(numero % 2 === 0) {  
    const resultado = numero/2  
    imprimir(resultado)  
  }  
}  
  
const imprimeMensagem = () => {  
  console.log('O resultado da sua conta é ' + numero)  
}  
  
verificaPar(2, imprimeMensagem)
```

imprimeMensagem
é a nossa
função de callback!



Callback

- As funções que são passadas como parâmetro para outras funções são chamadas de **callback**
- **Callback** significa "ligar de volta"/"retribuir"
- Elas recebem este nome porque são usadas no **fim** da função principal ou **depois de uma etapa** importante dela



Callback - Simplificação 📞

- Podemos simplificar e passar a arrow function diretamente como parâmetro, sem criar variáveis

```
const verificaPar = (numero, imprimir) => {  
  if(numero % 2 === 0){  
    const resultado = numero/2  
    imprimir(resultado)  
  }  
}  
  
verificaPar(2, (valor) => {  
  console.log("O valor da sua conta é", valor)  
})
```

Vamos ver na prática! 🔬





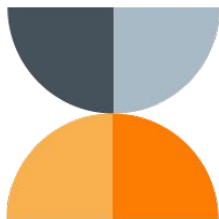
Exercício 1

- Faça um código que determine se um número é **ímpar**
- A ideia é que ele receba um **único número** e imprima no terminal "Sim, é ímpar" **somente** se o número for ímpar. Se não for, não faça **nada**
- Use uma função de **callback** para imprimir a mensagem no console



Pausa para relaxar 🧘

10 min



- Funções são trechos de código reutilizáveis
- Elas recebem variáveis como parâmetro
- Funções podem ser atribuídas à variáveis (chamamos de funções não nomeadas)
- Quando uma função é recebida como entrada de outra função, chamamos ela de **callback**



Funções de Array

Labenu_



Funções de Array

- Já sabemos iterar sobre arrays usando os loops
 - for, while, for...of
- Hoje vamos ver algumas funções específicas para arrays que facilitam fazer a **leitura** deles, bem como também fazer algumas **operações** nos itens



Funções de Array

- Apresentaremos 2 funções que **só podem ser usadas em arrays**:
 - `map()`
 - `filter()`
- Existem várias outras funções que você pode pesquisar **depois** que entender essas duas muito bem, pois serão **as mais importantes** para nós



Funções de Array

- Os outputs dessas funções são diferentes, mas todas elas esperam receber um **callback** como input
- Essa função de callback vai ser chamada **cada vez que avançamos em um elemento** do array
- Ou seja, essas funções também servem para **ler item por item** de um array e cada uma tem vantagens e usos diferentes!



map

Labenu_



map

- **Definição**: map significa "mapear"
- **Utilização**: quando queremos criar um **NOVO** array baseado em informações do array original
- **Output**: um novo array com tamanho **igual** ao tamanho do array original
 - Precisamos guardar esse array em algum lugar!



map

- **Input**: somente uma função de callback
- Esta função pode receber três parâmetros
 - primeiro: corresponde ao valor do elemento do array naquela etapa do loop (**item**)
 - segundo: o valor do índice daquele elemento (**index**)
 - terceiro: o array original em si (**array**)
- Esta função deve obrigatoriamente **retornar** algum valor



map - Sintaxe

```
const pokemons = [  
  { nome: "Bulbasaur", tipo: "grama" },  
  { nome: "Bellsprout", tipo: "grama" },  
  { nome: "Charmander", tipo: "fogo" },  
  { nome: "Vulpix", tipo: "fogo" },  
  { nome: "Squirtle", tipo: "água" },  
  { nome: "Psyduck", tipo: "água" },  
]  
  
const nomeDosPokemons = pokemons.map((pokemon, indice, array) => {  
  return pokemon.nome  
})
```



map - Sintaxe

```
const pokemons = [  
  { nome: "Bulbasaur", tipo: "grama" },  
  { nome: "Bellsprout", tipo: "grama" },  
  { nome: "Charmander", tipo: "fogo" },  
  { nome: "Vulpix", tipo: "fogo" },  
  { nome: "Squirtle", tipo: "água" },  
  { nome: "Psyduck", tipo: "água" },  
]  
  
const nomeDosPokemons = pokemons.map((pokemon, indice, array) => {  
  return pokemon.nome  
})
```

Vamos ver na prática! 





Exercício 2

- Faça um programa que tenha uma lista de numbers (quaisquer valores) e que transforme-a numa nova lista contendo strings
- O formato das strings deve ser:
O elemento $\${index}$ é $\${valor}$

ENTRADA:

```
[ 10, 11, 12, 13, 14, 15 ]
```

SAÍDA:

```
[ '0 elemento 0 é 10',  
  '0 elemento 1 é 11',  
  '0 elemento 2 é 12',  
  '0 elemento 3 é 13',  
  '0 elemento 4 é 14',  
  '0 elemento 5 é 15' ]
```



Pausa para relaxar 🐱

5 min



filter

Labenu_



filter 🚪

- **Definição**: filter significa "filtrar"
- **Utilização**: quando queremos criar um **NOVO** array retirando (ou não) alguns itens do array original
- **Output**: um novo array com tamanho **igual ou menor** ao tamanho do array original
 - Precisamos guardar esse array em algum lugar!



filter 🚪

- **Input**: somente uma função de callback
- Esta função pode receber três parâmetros
 - primeiro: corresponde ao valor do elemento do array naquela etapa do loop (**item**)
 - segundo: o valor do índice daquele elemento (**index**)
 - terceiro: o array original em si (**array**)
- Esta função deve **retornar** um booleano (true/false)



filter - Sintaxe



```
const pokemons = [  
  { nome: "Bulbasaur", tipo: "grama" },  
  { nome: "Bellsprout", tipo: "grama" },  
  { nome: "Charmander", tipo: "fogo" },  
  { nome: "Vulpix", tipo: "fogo" },  
  { nome: "Squirtle", tipo: "água" },  
  { nome: "Psyduck", tipo: "água" },  
]
```

```
const apenasPokemonsDeGrama = pokemons.filter((pokemon, indice, array) => {  
  return pokemon.tipo === "grama"  
})
```



filter - Sintaxe



```
const pokemons = [  
  { nome: "Bulbasaur", tipo: "grama" },  
  { nome: "Bellsprout", tipo: "grama" },  
  { nome: "Charmander", tipo: "fogo" },  
  { nome: "Vulpix", tipo: "fogo" },  
  { nome: "Squirtle", tipo: "água" },  
  { nome: "Psyduck", tipo: "água" },  
]
```

```
const apenasPokemonsDeGrama = pokemons.filter((pokemon, indice, array) => {  
  return pokemon.tipo === "grama"  
})
```

Vamos ver na prática!





Exercício 3

- Faça um programa que tenha uma lista de números (quaisquer valores) e crie dois novos arrays:
 - Um só com números maiores que 10
 - Outro só com números pares

ENTRADA:

[1, 2, 5, 8, 10, 11, 13, 15, 20]

SAÍDA:

Maior que 10: [11, 13, 15, 20]

Par: [2, 8, 10, 20]





Exercício 4

- Dado um array de produtos, onde cada produto é um objeto com nome, preço e categoria, retorne um novo array com **o nome** dos produtos da **categoria Limpeza**

```
const produtos = [  
  { nome: "Alface Lavada", categoria: "Hortifruti", preco: 2.5 },  
  { nome: "Guaraná 2l", categoria: "Bebidas", preco: 7.8 },  
  { nome: "Veja Multiuso", categoria: "Limpeza", preco: 12.6 },  
  { nome: "Dúzia de Banana", categoria: "Hortifruti", preco: 5.7 },  
  { nome: "Leite", categoria: "Bebidas", preco: 2.99 },  
  { nome: "Cândida", categoria: "Limpeza", preco: 3.30 },  
  { nome: "Detergente Ypê", categoria: "Limpeza", preco: 2.2 },  
  { nome: "Vinho Tinto", categoria: "Bebidas", preco: 55 },  
  { nome: "Berinjela kg", categoria: "Hortifruti", preco: 8.99 },  
  { nome: "Sabão em Pó", categoria: "Limpeza", preco: 10.80 }  
]
```



Resumo

Labenu_



Resumo

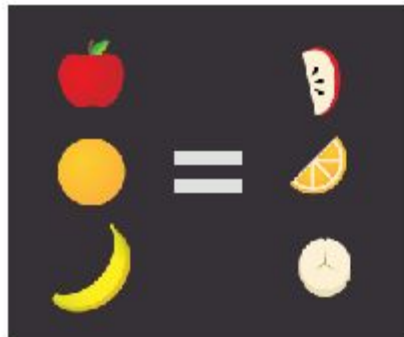


Função	Utilização	Retorna um <i>array</i> ?	Tamanho do array
<i>map</i>	Criar um novo <i>array</i> com elementos modificados em relação ao original	Sim	Igual ao original
<i>filter</i>	Criar um novo <i>array</i> com alguns elementos do original	Sim	Igual ou menor que o original

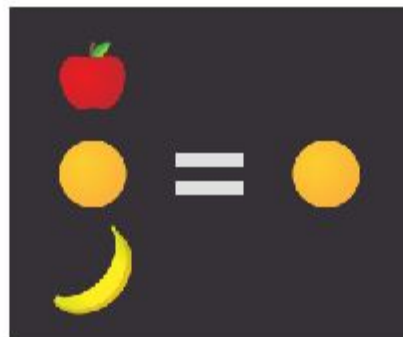


Resumo

`Array.map()`



`Array.filter()`



Dúvidas? 🧐

Labenu_





Obrigado(a)!