Semana 3 - Aula 4

Callback e Funções de Array

FUTURE4

Sumário

O que vamos ver hoje? 👀

- Vamos começar vendo um tipo específico de funções, chamadas de ***callback***

- Logo em seguida, veremos uma utilização prática delas em *arrays*

callback

Definição 📝

- Uma função é um trecho de código que desejamos **reutilizar**
- Pode receber uma ou mais variáveis como **entrada**
- Pode retornar uma **saída**

Definição 📝

```
1   function somar(a,b) {
2     return a + b
3   }
4   const resultado = somar(1,1)
5
```

FUTURE4

Definição 📝

- Em JS, **variáveis** podem assumir **valores** de **funções**

```
1   const somar = function(a, b){
2     return a + b
3   }
4   const resultado = somar(1,1)
5
```

```
1   const somar = (a, b) => {
2     return a + b
3   }
4   const resultado = somar(1,1)
5
```

Definição 📝

FUNÇÕES esperam receber **VARIÁVEIS**
como entradas

VARIÁVEIS podem ser **FUNÇÕES**

FUNÇÕES podem receber **FUNÇÕES**
como entradas

FUTURE4

Exemplo 🔍

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

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

FUTURE4

Exemplo 🔍

- No programa principal, só queremos mostrar o resultado se o número for par

- Vamos alterar **verificaPar** para receber uma outra variável

```
1   function verificaPar(numero, funcao2) {
2     if(numero % 2 === 0) {
3       const resultado = numero/2
4       funcao2(resultado)
5     }
6   }
7
```

FUTURE4

Exemplo 🔍

```
11  function verificaPar(numero, funcao2) {
12    if(numero % 2 === 0) {
13      const resultado = numero/2
14      funcao2(resultado)
15    }
16  }
17
18  const mostraResultado = (numero) => {
19    console.log("O Resultado é: ", numero)
20  }
21
22  verificaPar(20, mostraResultado)
23  verificaPar(21, mostraResultado)
24
```

EXECUTAR

FUTURE4

Exemplo 🔍

- Podemos simplificar e passar a ***arrow function*** diretamente, sem criar uma variável para isso

```
1   function verificaPar(numero, funcao2) {
2     if(numero % 2 === 0) {
3       const resultado = numero/2
4       funcao2(resultado)
5     }
6   }
7
8   verificaPar(20, (numero) => {
9     console.log("O Resultado é: ", numero)
10  })
11
```

Definição 📝

- As funções que são passadas como argumentos 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** muito importante

Exercício 1 - *callback*

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 ele for! Se não for, não faça nada.

- Use *callback* para imprimir no console

Pausa para relaxar 😴

5 min

FUTURE4

Big Picture

callback em *array*

Visão Geral 🧐

- Já sabemos iterar sobre arrays, usando os loops
  - *indexes* do *array*
  - operador *of*

- Vamos ver funções que facilitam a leitura e a análise de loops

Visão Geral 🧐

- Vamos ver as funções:
  - *forEach*
  - *map*
  - *filter*

Visão Geral 🧐

- Elas são funções dos **objetos** do tipo ***array***

```
1   const array = [1, 2, 3]
2
3   array.forEach()
4   array.map()
5   array.filter()
6
```

FUTURE4

Visão Geral 🧐

- Os ***input*** e ***output*** são diferentes, mas todas esperam receber um ***callback***

- Esta função é chamada cada vez que avançamos em uma **etapa** do *loop*

forEach

Definição 📝

- ***forEach*** significa "para cada"

- Utilização: Quando queremos iterar em cima das informações do array, para utilizá-las ou modificá-las

Definição 📝

- Output: Nada
- Input: Somente a função de ***callback***
  - Esta função tem três parâmetros:

    - *primeiro*: corresponde ao valor do elemento do array naquela etapa do *loop*

    - *segundo*: o valor do *index*

    - *terceiro*: é o *array* em si

Exemplo 🔍

```
112  const pokemons = [
113    { nome: "Bulbasaur",  tipo: "grama",  vida: 0 },
114    { nome: "Charmander", tipo: "fogo",   vida: 0 },
115    { nome: "Squirtle",   tipo: "água",   vida: 0 },
116  ]
117
118
119  pokemons.forEach((pokemon, index, array) => {
120    pokemon.vida = 1000
121  })
122
```

EXECUTAR

FUTURE4

Exercício 2 - *forEach*

Agora, vamos considerar que os *Pokemons* tenham outro atributo: "vidaMaxima", que é um número que corresponde ao valor máximo da vida dos *Pokemons*.

Faça um código que receba uma lista de *Pokemons* e altere o valor da vida deles para o valor máximo possível.

Exercício 2 - *forEach*

```
ENTRADA:
[ { nome: 'Bulbasaur', tipo: 'grama', vida: 0, vidaMaxima: 1000 },
  { nome: 'Charmander', tipo: 'fogo', vida: 0, vidaMaxima: 2000 },
  { nome: 'Squirtle', tipo: 'água', vida: 0, vidaMaxima: 3000 } ]

SAÍDA:
[ { nome: 'Bulbasaur', tipo: 'grama', vida: 1000, vidaMaxima: 1000 },
  { nome: 'Charmander', tipo: 'fogo', vida: 2000, vidaMaxima: 2000 },
  { nome: 'Squirtle', tipo: 'água', vida: 3000, vidaMaxima: 3000 } ]
```

Pausa para relaxar 😴

5 min

FUTURE4

map

Definição 📝

- ***map*** significa "mapear"

- Utilização: Quando queremos criar um **NOVO** *array* com elementos cujas informações dependem do original.

Definição 📝

- Output: Um novo array, com o tamanho igual ao do original

- Input: Somente a função de ***callback***

    - Esta função espera os mesmos argumentos que vimos no caso do *forEach*

    - Ela deve, obrigatoriamente, **retornar** algum valor

Exemplo 1 🔍

```
1   const pokemons = [
2     { nome: "Bulbasaur",  tipo: "grama",  vida: 0 },
3     { nome: "Charmander", tipo: "fogo",   vida: 0 },
4     { nome: "Squirtle",   tipo: "água",   vida: 0 }
5   ]
6
7   const nomeDosPokemons = pokemons.map((pokemon, index, array) => {
8     return pokemon.nome
9   })
10
```

EXECUTAR

FUTURE4

Exemplo 2 🔍

```
1   const pokemons = [
2     { nome: "Bulbasaur",  tipo: "grama",  vida: 0 },
3     { nome: "Charmander", tipo: "fogo",   vida: 0 },
4     { nome: "Squirtle",   tipo: "água",   vida: 0 }
5   ]
6
7   const nomeEVidaDosPokemons = pokemons.map((pokemon, index, array) => {
8     return {
9       nome: pokemon.nome,
10      vida: pokemon.vida
11    }
12  })
13
```

EXECUTAR

FUTURE4

Exercício 3 - *map*

Faça um programa que tenha uma lista de *numbers* (quaisquer valores) e transforme-a numa nova lista, contendo *strings* nesse formato:

"O elemento ${index do elemento} é ${valor}"

Exercício 3 - *map*

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

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

FUTURE4

Pausa para relaxar 😴

5 min

FUTURE4

filter

Definição 📝

- ***filter*** significa "filtrar"

- Utilização: Quando criar um **NOVO** *array*, retirando ou não alguns elementos

Definição 📝

- Output: Um novo array, com o tamanho igual ou menor do que o original

- Input: Somente a função de ***callback***

    - Esta função espera os mesmos argumentos que vimos no caso do *forEach/map*

    - Ela deve, obrigatoriamente, retornar algum **booleano** (*true* ou *false*)

Exemplo 1 🔍

```
1   const pokemons = [
2     { nome: "Bulbasaur",  tipo: "grama",  vida: 0 },
3     { nome: "Bellsprout",  tipo: "grama",  vida: 0 },
4     { nome: "Charmander", tipo: "fogo",   vida: 0 },
5     { nome: "Vulpix", tipo: "fogo",   vida: 0 },
6     { nome: "Squirtle",   tipo: "água",   vida: 0 },
7     { nome: "Psyduck",   tipo: "água",   vida: 0 }
8   ]
9
10  const soPokemonsDeGrama = pokemons.filter((pokemon, index, array) => {
11    if(pokemon.tipo === "grama"){
12      return true
13    }
14    return false
15  })
16
```

EXECUTAR

FUTURE4

Exemplo 2 🔍

```
1   const pokemons = [
2     { nome: "Bulbasaur",  tipo: "grama",  vida: 0 },
3     { nome: "Bellsprout",  tipo: "grama",  vida: 0 },
4     { nome: "Charmander", tipo: "fogo",   vida: 0 },
5     { nome: "Vulpix", tipo: "fogo",   vida: 0 },
6     { nome: "Squirtle",   tipo: "água",   vida: 0 },
7     { nome: "Psyduck",   tipo: "água",   vida: 0 }
8   ]
9
10  const pokemonsDeFogoEAgua = pokemons.filter( pokemon => {
11    if(pokemon.tipo === "água" || pokemon.tipo === "fogo") {
12      return true
13    }
14    return false
15  })
16
```

EXECUTAR

FUTURE4

Exemplo 3 🔍

```
1   const pokemonsDeFogoEAgua = pokemons.filter( pokemon => {
2     if(pokemon.tipo === "água" || pokemon.tipo === "fogo") {
3       return true
4     }
5     return false
6   })
7
```

```
1   const pokemonsDeFogoEAgua = pokemons.filter( pokemon => {
2     return pokemon.tipo === "água" || pokemon.tipo === "fogo"
3   })
4
```

```
1   const soPokemonsDeGrama = pokemons.filter((pokemon, index, array) => {
2     if(pokemon.tipo === "grama"){
3       return true
4     }
5     return false
6   })
7
```

```
1   const soPokemonsDeGrama = pokemons.filter((pokemon, index, array) => {
2     return pokemon.tipo === "grama"
3   })
4
```

FUTURE4

Exercício 4 - *filter*

Faça um programa que tenha uma lista de *numbers* (quaisquer valores) e crie 2 novos *arrays*:

a. Um só com os números maiores do que 10
b. Um só com os números pares

Exercício 4 - *filter*

```
ENTRADA:
[ 1, 2, 5, 8, 10, 11, 13, 15, 20 ]
SAÍDA:
Maior que 10:  [ 11, 13, 15, 20 ]
Par:  [ 2, 8, 10, 20 ]
```

FUTURE4

Resumo

Resumo 📓

- Funções podem receber **entradas** de quaisquer tipos e realizar suas operações
- Essas entradas podem até ser **outras funções**
- ***callback*** são funções passadas como **entradas** de outras funções e costumam ser usadas ao fim de uma etapa importante da sua execução

Resumo 📓

Função	Utilização	Retorna um *array*?	Tamanho do array
forEach	Ler ou utilizar os itens do *array*	Não	-
map	Criar um novo *array* com elementos modificados em relação ao original	Sim	Igual ao original
filter	Criar um novo *array* com alguns elementos do original	Sim	Igual ou menor que o original

Dúvidas?

FUTURE4

Obrigado!

FUTURE4