TRYBE
Modulo II – Front-end

Bloco 8 – Higher Order Functions (HOFs)

Definição
HOF é uma função que recebe outra função como parâmetro ou que retorna uma função.

Uso
HOFs são funções poderosas:
- facilitando a manipulação e criação de arrays;
- deixando o código mais legível;
- substituindo a escritura manual inteira do for.

Sintax
```
array.hof((element, otherpossibleparameters) => {
executed function;
});
```

1) forEach, find, filter, some, every, sort

Array.forEach

Para **iterar** sobre todos os elementos de um array. Percorre o array dado e executa a função passada para cada um dos seus valores. O forEach **não retorna nenhum valor.**

```
const numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

const multipliesFor2 = (element) => {
  console.log(`${element} * 2: ${element * 2}`);
};

numbers.forEach(multipliesFor2);
```

Array.find

Para **encontrar o primeiro elemento** de um array que satisfaça a uma condição.
(Para pegar todos, é com filter.)

```
const array1 = [5, 12, 8, 130, 44];

const found = array1.find(element => element > 10);

console.log(found);
// expected output: 12
```

Array.filter

Igual ao find, mas **retorna todos os elementos** que satisfaçam a uma condição.
Ex, substituindo o ex acima com filter, daria //expected output: [12, 130, 44]

Prático para **filtrar** um array: ou seja remover elemento não desejado de um array.

Array.some e Array.every

Para **testar** se os elementos de um array satisfazem a uma condição. Retornam **boolean**.

A primeira retorna **true se ao menos um elemento** de um array satisfaz a uma condição.
A segunda retorna **true se todos os elementos** de um array satisfazem a uma condição.

```
function isBiggerThan10(element, index, array) {
  return element > 10;
}
[2, 5, 8, 1, 4].some(isBiggerThan10);  // false
[12, 5, 8, 1, 4].some(isBiggerThan10); // true
```

```
function isBigEnough(element, index, array) {
  return element >= 10;
}
[12, 5, 8, 130, 44].every(isBigEnough);   // false
[12, 54, 18, 130, 44].every(isBigEnough); // true
```

Array.sort

Para **ordenar** arrays de acordo com algum critério (numérico, alfabético, crescente, decrescente…).
Exemplos principais no site https://www.w3schools.com/jsref/jsref_sort.asp .

Ou seja: caso queira ordenar de forma alfabética, basta chamar `sort`, passando nenhuma função como parâmetro.

Copiar

```
const food = ['arroz', 'feijão', 'farofa', 'chocolate', 'doce de leite'];
food.sort();
console.log(food);
// [ 'arroz', 'chocolate', 'doce de leite', 'farofa', 'feijão' ]
```

Para ordenar de forma numérica crescente, é necessário passar a função a seguir:

Copiar

```
const points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){ return a - b });
console.log(points); // [1, 5, 10, 25, 40, 100]
```

Sobre todos

Todos podem receber mais do que um paramêtro, assim: *(element, indice, array)*

---

2) map e reduce

Array.map

Definição
O map aplica uma função sobre os elementos de um array e retorna um array novo, sem modificar o original. Retorna um array do mesmo tamanho.
Diferente do forEach que não retorna nada, que pode modificar array original e que é genérico.

Usos

→ **Retornar um array novo** de mesmo tamanho com alterações passadas, à partir de elementos de um array inicial.

```
const persons = [
  { firstName: "Maria", lastName: "Ferreira" },
  { firstName: "João", lastName: "Silva" },
  { firstName: "Antonio", lastName: "Cabral" },
];

const fullNames = persons.map((person) => `${person.firstName} ${person.lastName}`);

console.log(fullNames); // [ 'Maria Ferreira', 'João Silva', 'Antonio Cabral' ]
```

→ **Unir dois arrays** para criar um novo

```
const products = ['Arroz', 'Feijao', 'Alface', 'Tomate'];

const prices = [2.99, 3.99, 1.5, 2];

const updateProducts = (listProducts, listPrices) => {
  return listProducts.map((product, index) => (
    { [product]: listPrices[index] }
  ));
};

const listProducts = updateProducts(products, prices);
console.log(listProducts);
=> [
  { Arroz: 2.99 },
  { Feijao: 3.99 },
  { Alface: 1.5 },
  { Tomate: 2 },
]
```

→ **Pode ser combinado com outras HOFs** como filter e find, para retornar resultado desejado.

Array.reduce

Definição
Possui um parâmetro a mais, chamado de acumulador (acc.) que serve para acumular dentro de si os resultados das funções. O reduce **retorna apenas um valor,** que está no accumulador.
Dois paramêtros: o acumulador e o valor iterado.

Usos

→ Uso do **acumulador**

```
const numbers = [32, 15, 3, 2, -5, 56, 10];

const sumNumbers = numbers.reduce((result, number) => result + number);
console.log(sumNumbers); // 113
```

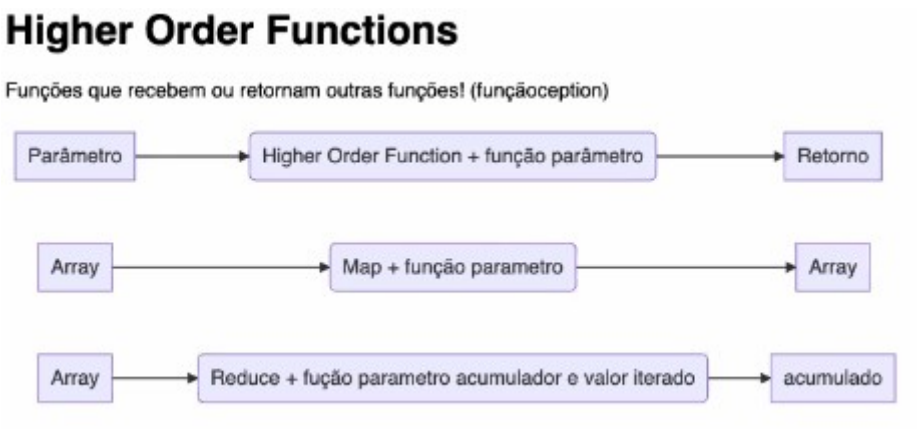
// O parâmetro `result` é o acumulador. Ele recebe, do `reduce`, o retorno da função a cada iteração.
// O retorno é salvo no primeiro parâmetro , result. É como fazer result = result + number;

→ Pode **passar um segundo parâmetro para o reduce**, logo após a função.

```
const sumNumbers = numbers.reduce(getSum, 10);
console.log(sumNumbers); // 123
```

// Ao adicionar esse segundo parâmetro você está colocando um valor inicial na variável results, ou seja, no acumulador. Caso nenhum parâmetro for passado, o seu valor inicial será o retorno da primeira iteração do reduce sobre o array.

Infografia resumindo HOFs, map e reduce



Dicas diversas

() serve para chamar a function, depois de métodos tipo toUpperCase

–

lembrar da escritura condicional ternary:

```
condition ? exprIfTrue : exprIfFalse
```

–

---

3) Js ES6 - spread operator, parâmetro rest, destructuring e mais

Aprendizagem de 7 features do Javascript ES6.

spread operator

Cria item novo à partir de item existente. **Espalha os valores de um objeto iterável,** como array (pode ser chamadostambém de vetor) e objetos.

```
const arr = [0,1,2]
const newArr = [3, ...arr]
```

//[3,0,1,2]

```
function soma(a,b,c){
  return a+b+c
}


console.log(soma(...arr))
```

parâmetro rest

Funcionalidade **oposta ao spread**: **os valores listados são agrupados em um array.**
Permite a passagem, em um único parâmetro, de um número indefinido de valores.

```
printPackageContents('cheese', 'eggs', 'milk', 'bread');
function printPackageContents(...items) {
  for (const item of items) {
    console.log(item);
  }
}
```

Sintax:

```
function functionname(...parameters)
{
statement;
}
```

Usos
Assim, pode criar functions com bastante paramêtros.
Esses paramêtros são armazenados num array que pode ser acessado depois desde dentro da function.
Com o rest, uma function pode ser chamada graças ao rest parameter com qualquer número de params, sem importar como foi definida.

object destructuring

Sintax
[{Variáveis}] = [Objeto];

Usos

- Para **extrair o valor de algum objeto.**
Permite pegar apenas o que quiser num objeto.

```
let options = {
  title: "Menu",
  width: 100,
  height: 200
};

let {title, width, height} = options;

alert(title);  // Menu
alert(width);  // 100
alert(height); // 200
```

- Também pode **modificar nomes de chaves,** usando "**:**"

Ou seja isso fala basicamente
"what : goes where"

```
let options = {
  title: "Menu",
  width: 100,
  height: 200
};

// { sourceProperty: targetVariable }
let {width: w, height: h, title} = options;

// width -> w
// height -> h
// title -> title

alert(title);  // Menu
alert(w);      // 100
alert(h);      // 200
```

- Finalmente, pode **adicionar propriedades que faltam com "="**

```
let options = {
  title: "Menu"
};

let {width = 100, height = 200, title} = options;

alert(title);  // Menu
alert(width);  // 100
alert(height); // 200
```

- Usos podem ser **combinados**

```
let {width: w = 100, height: h = 200, title} = options;
```

- O destruturador pode ser **integrado em functions.**

```
function imprimeDadosDoAluno({nome, idade, email})
```

array destructuring

Também aplica uma desestruturação, só que em arrays.

```
var first = someArray[0];
var second = someArray[1];
var third = someArray[2];
```

---------->

```
var [first, second, third] = someArray;
```

Pode **aninhar** com **[]**

```
var [foo, [[bar], baz]] = [1, [[2], 3]];
console.log(foo);
// 1
console.log(bar);
// 2
console.log(baz);
// 3
```

Pode **pular** com **,**

```
var [,,third] = ["foo", "bar", "baz"];
console.log(third);
// "baz"
```

Melhor artigo para todos os destructuring:

https://hacks.mozilla.org/2015/05/es6-in-depth-destructuring/

default destructuring

Default destructuring permite evitar o undefined e **fornecer um valor default** para quando a propriedade que quer destruturar não é definida
(ex: não existe em um objeto, ou valor não existe em um objeto).

```
const position2d = [1.0, 2.0]
const [x, y, z = 0] = position2d

console.log(z) // 0
```

```
var [missing = true] = [];
console.log(missing);
// true

var { message: msg = "Something went wrong" } = {};
console.log(msg);
// "Something went wrong"

var { x = 3 } = {};
console.log(x);
// 3
```

abbreviation object literal

Para **remover duplicações** nos objetos

```
{
  name: name,
  age: age
}
```

----->

```
{
  name,
  age
}
```

default params

Queremos que o **parâmetro tenha um valor predefinido**.

Sintax
function functionname (chosenname = 'defaultvalue') {}

```
function hello (name = 'World') {
  console.log(`Hello ${name}`);
}

hello('People');
hello();

// > Hello People
// > Hello World
```