ORIGAMID

OBJETOS

Array Iteração

[].forEach()

`[].forEach(callback(itemAtual, index, array))` a função de callback é executada para cada item da array. Ela possui três argumentos, itemAtual (valor do item da array), index (index do valor na array) e array (array original).

```
const carros = ['Ford', 'Fiat', 'Honda'];
carros.forEach(function(item, index, array) {
  console.log(item.toUpperCase());
});

// com Arrow Function
carros.forEach((item, index, array) => {
  console.log(item.toUpperCase());
});
```

> *O método sempre retorna undefined*

Arrow Function

```
const li = document.querySelectorAll('li');

li.forEach(i => i.classList.add('ativa'));

li.forEach(function(item) {
  item.classList.add('ativa');
});
```

Modificar a Array Original

O terceiro argumento do callback é uma referência direta e se modificado irá também modificar a array original.

```
const carros = ['Ford', 'Fiat', 'Honda'];
carros.forEach((item, index, array) => {
  array[index] = 'Carro ' + item;
});

carros; // ['Carro Ford', 'Carro Fiat', 'Carro Honda']
```

> *É melhor utilizarmos o map para isso*

[].map()

`[].map(callback(itemAtual, index, array))` funciona da mesma forma que o forEach(), porém ao invés de retornar undefined, retorna uma nova array com valores atualizados de acordo com o return de cada iteração.

```
const carros = ['Ford', 'Fiat', 'Honda'];
const newCarros = carros.map((item) => {
  return 'Carro ' + item;
});

carros; // ['Ford', 'Fiat', 'Honda']
newCarros; // ['Carro Ford', 'Carro Fiat', 'Carro Honda'];
```

Valor Retornado

Se não retornarmos nenhum valor durante a iteração utilizando map, o valor retornado como de qualquer função que não possui o return, será undefined.

```
const carros = ['Ford', 'Fiat', 'Honda'];
const newCarros = carros.map((item) => {
  const novoValor = 'Carro ' + item;
});

newCarros; // [undefined, undefined, undefined];
```

Arrow Function e [].map()

Uma Arrow Function de linha única e sem chaves irá retornar o valor após a fat arrow `=>`.

```
const numeros = [2, 4, 6, 8, 10, 12, 14];
const numerosX3 = numeros.map(n => n * 3);

numerosX3; // [6, 12, 18, 24, 30, 36, 42];
```

[].map() vs [].forEach()

Se o objetivo for modificar os valores da array atual, sempre utilize o map, pois assim uma nova array com os valores modificados é retornada e você pode imediatamente iterar novamente sobre estes valores.

```
const numeros = [2, 4, 6, 8, 10, 12, 14];
const numerosX3 = numeros.map(n => n * 3);

numerosX3; // [6, 12, 18, 24, 30, 36, 42];
```

[].map() com Objetos

Map pode ser muito útil para interagirmos com uma array de objetos, onde desejamos isolar um valor único de cada objeto.

```
const aulas = [
  {
    nome: 'HTML 1',
    min: 15
  },
  {
    nome: 'HTML 2',
    min: 10
  },
  {
    nome: 'CSS 1',
    min: 20
  },
  {
    nome: 'JS 1',
    min: 25
  },
]
```

```
// [15, 10, 20, 25];

const puxarNomes = aula => aula.nome;
const nomesAulas = aulas.map(puxarNomes);
// ['HTML 1', 'HTML 2', 'CSS 1', 'JS 1']
```

[].reduce()

`[].reduce(callback(acumulador, valorAtual, index, array), valorInicial)`
executa a função de callback para cada item da Array. Um valor especial existe nessa função de callback, ele é chamado de `acumulador`, mas é na verdade apenas o retorno da iteração anterior.

```
const aulas = [10, 25, 30];
const total1 = aulas.reduce((acumulador, atual) => {
  return acumulador + atual;
});
total1; // 65

const total2 = aulas.reduce((acc, cur) => acc + cur, 100);
total2; // 165
```

Reduce Passo a Passo 1

O primeiro parâmetro do callback é o valor do segundo argumento passado no reduce(callback, inicial) durante a primeira iteração. Nas iterações seguintes este valor passa a ser o retornado pela anterior.

```
const aulas = [10, 25, 30];

// 1
aulas.reduce((0, 10) => {
  return 0 + 10;
}, 0); // retorna 10

// 2
aulas.reduce((10, 25) => {
  return 10 + 25;
}, 0); // retorna 35

// 3
aulas.reduce((35, 30) => {
  return 35 + 30;
}, 0); // retorna 65
```

Reduce Passo a Passo 2

Se não definirmos o valor inicial do acumulador, ele irá `pular` a primeira iteração e começara a partir da segunda. Neste caso o valor do acumulador será o valor do item da primeira iteração.

```
const aulas = [10, 25, 30];

// 1
aulas.reduce((10, 25) => {
  return 10 + 25;
}) // retorna 35

// 2
aulas.reduce((35, 30) => {
  return 35 + 30;
}) // retorna 65
```

Maior Valor com [].reduce()

```
const numeros = [10, 25, 60, 5, 35, 10];

const maiorValor = numeros.reduce((anterior, atual) => {
  return anterior < atual ? atual : anterior;
});

maiorValor; // 60
```

Podemos retornar outros valores

```
const aulas = [
  {
    nome: 'HTML 1',
    min: 15
  },
  {
    nome: 'HTML 2',
    min: 10
  },
  {
    nome: 'CSS 1',
    min: 20
  },
  {
    nome: 'JS 1',
    min: 25
  },
]

const listaAulas = aulas.reduce((acumulador, atual, index) => {
```

```
}, {})
```

Passo a passo Reduce

Passo a passo do método reduce criando um Objeto.

```
// 1
aulas.reduce(({}, {nome: 'HTML 1', min: 15}, 0) => {
  {}[0] = 'HTML 1';
  return {0: 'HTML 1'};
}, {})

// 2
aulas.reduce(({0: 'HTML 1'}, {nome: 'HTML 2', min: 10}, 1) => {
  {0: 'HTML 1'}[1] = 'HTML 2';
  return {0: 'HTML 1', 1: 'HTML 2'};
}, {})

// 3
aulas.reduce(({0: 'HTML 1', 1: 'HTML 2'}, {nome: 'CSS 1', min:
20}, 2) => {
  {0: 'HTML 1', 1: 'HTML 2'}[2] = 'CSS 1';
  return {0: 'HTML 1', 1: 'HTML 2', 2: 'CSS 1'};
}, {})
```

```
aulas.reduce(({0: 'HTML 1', 1: 'HTML 2', 2: 'CSS 1'}, {nome:
'JS 1', min: 25}, 3) => {
  {0: 'HTML 1', 1: 'HTML 2', 2: 'CSS 1'}[3] = 'JS 1';
  return {0: 'HTML 1', 1: 'HTML 2', 2: 'CSS 1', 3: 'JS 1'};
}, {})
```

[].reduceRight()

Existe também o método `[].reduceRight()`, a diferença é que este começa a iterar da direita para a esquerda, enquanto o reduce itera da esquerda para a direita.

```
const frutas = ['Banana', 'Pêra', 'Uva'];

const frutasRight = frutas.reduceRight((acc, fruta) => acc + '
' + fruta);
const frutasLeft = frutas.reduce((acc, fruta) => acc + ' ' +
fruta);

frutasRight; // Uva Pêra Banana
frutasLeft; // Banana Pêra Uva
```

[].some()

`[].some()`, se pelo menos um return da iteração for truthy, ele retorna true.

```
const frutas = ['Banana', 'Pêra', 'Uva'];
const temUva = frutas.some((fruta) => {
  return fruta === 'Uva';
}); // true

function maiorQue100(numero) {
  return numero > 100;
}
const numeros = [0, 43, 22, 88, 101, 2];
const temMaior = numeros.some(maiorQue100); // true
```

[].every()

`[].every()`, se todos os returns das iterações forem truthy, o método irá retornar true. Se pelo menos um for falsy, ele irá retornar false.

```
const frutas = ['Banana', 'Pêra', 'Uva', ''];
// False pois pelo menos uma fruta
// está vazia '', o que é um valor falsy
const arraysCheias = frutas.every((fruta) => {
  return fruta; // false
});

const numeros = [6, 43, 22, 88, 101, 29];
const maiorQue3 = numeros.every(x => x > 3); // true
```

[].find() e [].findIndex()

`[].find()`, retorna o valor atual da primeira iteração que retornar um valor truthy. Já o `[].findIndex()`, ao invés de retornar o valor, retorna o index deste valor na array.

```
const frutas = ['Banana', 'Pêra', 'Uva', 'Maçã'];
const buscaUva = frutas.findIndex((fruta) => {
  return fruta === 'Uva';
}); // 2

const numeros = [6, 43, 22, 88, 101, 29];
const buscaMaior45 = numeros.find(x => x > 45); // 88
```

[].filter()

`[].filter()`, retorna uma array com a lista de valores que durante a sua iteração retornaram um valor truthy.

```
const frutas = ['Banana', undefined, null, '', 'Uva', 0,
'Maçã'];
const arrayLimpa = frutas.filter((fruta) => {
  return fruta;
}); // ['Banana', 'Uva', 'Maçã']

const numeros = [6, 43, 22, 88, 101, 29];
const buscaMaior45 = numeros.filter(x => x > 45); // [88, 101]
```

Filter em Objetos

```
const aulas = [
  {
    nome: 'HTML 1',
    min: 15
  },
  {
    nome: 'HTML 2',
    min: 10
  },
  {
    nome: 'CSS 1',
    min: 20
  },
  {
    nome: 'JS 1',
    min: 25
  },
]

const aulasMaiores = aulas.filter((aula) => {
```

```
// [{nome: 'CSS 1', min: 20}, {nome: 'JS 1', min: 25}]
```

Exercícios

```
<section class="curso">
  <h1>Web Design Completo</h1>
  <p>Este curso é para quem deseja entrar ou já está no mercado de criação de websites.</p>
  <span class="aulas">80</span>
  <span class="horas">22</span>
</section>
<section class="curso">
  <h1>WordPress Como CMS</h1>
  <p>No curso de WordPress Como CMS, você aprende do zero como pegar qualquer site em HTML e torná-lo totalmente gerenciável com a plataforma do WordPress.</p>
  <span class="aulas">46</span>
  <span class="horas">9</span>
</section>
<section class="curso">
  <h1>UI Design Avançado</h1>
  <p>Este é um curso avançado de User Interface Design.</p>
  <span class="aulas">55</span>
```

```
// Selecione cada curso e retorne uma array
// com objetos contendo o título, descricao,
// aulas e horas de cada curso


// Retorne uma lista com os
// números maiores que 100
const numeros = [3, 44, 333, 23, 122, 322, 33];


// Verifique se Baixo faz parte
// da lista de instrumentos e retorne true
const instrumentos = ['Guitarra', 'Baixo', 'Bateria',
'Teclado']


// Retorne o valor total das compras
const compras = [
  {
    item: 'Banana',
    preco: 'R$ 4,99'
```

```
    item: 'Ovo',
    preco: 'R$ 2,99'
  },
  {
    item: 'Carne',
    preco: 'R$ 25,49'
  },
  {
    item: 'Refrigerante',
    preco: 'R$ 5,35'
  },
  {
    item: 'Quejo',
    preco: 'R$ 10,60'
  }
]
```