Funções de array

Vamos ver algumas funções que podem te ajudar muito na hora de tratar dados no JavaScript. Saber trabalhar com funções de Arrays pode poupar muito tempo seu e te ajudar a criar lógicas mais simples e descomplicadas, então vamos ver agora algumas funções de Array para te ajudar no desenvolvimento das suas aplicações.

Push

O método push adiciona elementos dentro de array
OBS: quando temos uma funcionalidade atrelada a um objeto chamamos ela de método.

```javascript
const usuarios = ['Julio', 'Vitor', 'Micael', 'Pedro']
usuarios.push('Flavio')

console.log(usuarios)
// Resultado: ['Julio', 'Vitor', 'Micael', 'Pedro', 'Flavio']
```

Resgatar ou pesquisar

FILTER

Podemos usar o método filter para procurar valores e retornar eles, veja esse exemplo:

Retornando um valor diferente:

```javascript
const usuarios = ['Joao', 'Maria', 'Jose']

const usuariosNovos = usuarios.filter(usuario => {
    return usuario !== 'Joao'
})
```

```
console.log(usuariosNovos)
// Resultado: ['Maria', 'Jose']
```

Retornando um valor específico:

```javascript
const usuarios = ['Joao', 'Maria', 'Jose']

const usuarioEspecifico = usuarios.filter(usuario => {
    return usuario === 'Joao'
})

console.log(usuarioEspecifico)
// Resultado: ['Joao']
```

Você pode usar esse tipo de método para exibir dados por determinado ID ou excluir um dado por determinado ID, por exemplo.

FIND

Podemos usar o método Find com uma finalidade parecida com os exemplos que usamos no filter, porém o find tem um comportamento diferente do filter, veja esse exemplo:

```javascript
const usuarios = ['Joao', 'Maria', 'Jose']

const usuariosNovos = usuarios.find(usuario => {
    return usuario !== 'Joao'
})

console.log(usuariosNovos)
// Resultado: Maria
```

O Find para a sua execução assim que ele acha um valor de acordo com a lógica que pedimos, pedimos para que fosse retornado um valor diferente de 'Joao' e assim que ele achou esse valor ele encerrou sua execução.

Podemos usar o método Find da mesma forma do segundo exemplo do método Filter, trocando apenas o Filter por Find.

```javascript
const usuarios = ['Joao', 'Maria', 'Jose']

const usuarioEspecifico = usuarios.find(usuario => {
```

```
    return usuario === 'Joao'
})

console.log(usuarioEspecifico)
// Resultado: Joao
```

INCLUDES

O Includes é um método que podemos usar para verificar se um valor existe dentro do array, veja esse exemplo:

```javascript
const usuarios = ['Joao', 'Maria', 'Jose']

console.log(usuarios.includes('Maria'))
// Resultado: true
```

Você pode definir inclusive a partir de que posição você quer fazer essa verificação, por exemplo:

```javascript
const usuarios = ['Joao', 'Maria', 'Jose']

console.log(usuarios.includes('Maria'), 2)
// Resultado: false
```

O resultado é falso pois o usuário 'Maria' se encontra na posição 1 do array, e a partir da posição 2 não existe nenhum usuário 'Maria'.

Update

SPREAD

Com o operador Spread podemos adicionar todos os valores existentes em um array dentro de outro e adicionar ainda um novo valor, essa técnica é muito utilizada em Apps de Todo List, veja esse exemplo:

```javascript
let tarefas = ['Cozinhar', 'Comprar leite', 'Lavar o carro']

// suponha que a tarefa venha de um input do usuário
const novaTarefa = inputTarefa
```

```
tarefas = [...tarefas, novaTarefa]
console.log(tarefas)
// Resultado: Todas as tarefas do array + tarefa adicionada pelo usuário
```

SLICE

Com o Slice podemos recortar elementos do nosso Array, veja esse exemplo:

```javascript
const usuarios = ['Joao', 'Maria', 'Jose']

const usuariosAtualizados = usuarios.slice(1)

console.log(usuariosAtualizados)
// Resultado: ['Maria', 'Jose']
```

Nós removemos uma posição do Array a partir do início.

Podemos ainda remover valores de trás para frente, fazemos isso da seguinte forma:

```javascript
const usuarios = ['Joao', 'Maria', 'Jose']

const usuariosAtualizados = usuarios.slice(0, -1)

console.log(usuariosAtualizados)
// Resultado: ['Joao', 'Maria']
```

Para que você internalize bem o uso do Slice é interessante que você faça vários testes diferentes para entender todo o funcionamento dele.

Remove

POP

Podemos facilmente remover o último elemento de um Array usando o método Pop, veja esse exemplo:

```javascript
const usuarios = ['Joao', 'Maria', ''Jose']

usuarios.pop()
console.log(usuarios)
// Resultado: ['Joao', 'Maria']
```

Transform

MAP

Com o método MAP nós conseguimos fazer alterações em todo nosso Array, veja esse exemplo:

```javascript
const numeros = [1, 2, 3, 4]

const numeroUm = numeros.map(numero => {
    return 1
})

console.log(numeros)
console.log(numeroUm)

// Resultado:
// [1, 2, 3, 4]
// [1, 1, 1, 1]
```

REVERSE

Como o próprio nome diz podemos usar o Reverse para reverter os valores de Arrays, veja esse exemplo:

```javascript
const usuarios = ['Joao', 'Maria', 'Jose']
console.log(usuarios.reverse())
// Resultado: ['Jose', 'Maria', 'Joao']
```

Porém, é importante saber que fazendo dessa forma nós estamos alterando o Array original, para criarmos um novo Array utilizando o Reverse fazemos assim:

```javascript
const usuarios = ['Joao', 'Maria', 'Jose']
const usuariosReverse = [...usuarios].reverse()

console.log(usuarios)
console.log(usuariosReverse)

// Resultado:
// ['Joao', 'Maria', 'Jose']
// ['Jose', 'Maria', 'Joao']
```

Loops

FOREACH

Apesar de vários métodos apresentados aqui também serem Loops, o ForEach tem um comportamento diferente deles, nós usamos o ForEach para percorrer cada item do nosso Array, veja esse exemplo:

```javascript
const usuarios = ['Joao', 'Maria', 'Jose']

usuarios.forEach(usuario => {
    console.log(usuario)
})

// Resultado:
// Joao
// Maria
// Jose
```

Podemos usar o ForEach para percorrer cada item do nosso Array e executar alguma função para tratar cada item.