Listas e Registros

É possível fazer coleções de dados em linguagens de programação como o JavaScript. As coleções que utilizaremos aqui são as listas (arrays) e registros. Também serão vistas operações as quais é possível realizar com essas coleções de dados.

Listas

As listas são coleções ordenadas de dados. Esses dados podem ser de qualquer tipo, incluindo outras coleções de dados

```
const nomes = ['João','Maria','José','Silvano','Raphael']
```

A estrutura acima é uma lista que contém diferentes `strings`. e cada item da lista pode ser acessado por meio de `nomes[x]`, sendo `x` o índice (sendo 0 o primeiro item) que representa cada item.

```
const nomes = ['João','Maria','José','Silvano','Raphael']
console.log(nomes[0])
```

O código acima retorna:

```
'João'
```

Pode existir listas dentro de listas, como no exemplo abaixo

```
const multiplos = [[3,1],[6,3,1],[9,6,3,1],[12,9,6,3,1]]
```

Operações com Listas

Podemos fazer 3 operações com listas:

Mapeamento

Uma operação de mapeamento cria uma nova lista com o mesmo número de elementos. Esses elementos são modificados de acordo com uma função ou expressão definida

```
const numeros = [2,3,5,8,10,12]
const dobroLista = (lista) => lista.map((x) => x * 2)
console.log(dobrolista(numeros))
```

O código acima retorna

```
[4,6,10,16,20,24]
```

Para entender o código anterior, é necessário apontar algumas coisas:

- Na expressão `lista.map((x) => x * 2)`, x representa o elemento a ser operado na lista. Ou seja, para o caso do 2 (primeiro elemento da lista), a operação realizada é `(2)= 2 * 2`, que retorna 4
- A quantidade de elementos na lista **NÃO** é modificada
- São geradas novas listas contendo os novos valores do mapeamento.

Filtragem

As operações de filtragem retomam uma nova lista com elementos cuja função ou expressão tenha condição satisfeita. Logo, as expressões utilizadas na operação são de valor Booleano (Verdadeiro ou Falso)

```
const nomes = ['João','Maria','José','Silvano','Raphael', 'Júlio']
const filtroLetra = (lista) => lista.filter((x) => x[0] == 'J')
console.log(filtroLetra(nomes))
```

A operação acima **filtra** os nomes cuja inicial é a letra 'J', então retorna a lista abaixo:

```
['João','José','Júlio']
```

Perceba que: * Assim como nas expressões de mapeamento, o x na expressão de filtragem representa os elementos a serem operados na lista. Assim, caso o valor lógico do item corresponda ao valor declarado na expressão, o item é incluso na nova lista. Caso contrário, o item é ignorado. * Caso haja apenas um item na lista, ainda assim será retornada uma nova lista com o item que satisfaz a condição. * Caso nenhum item da lista a ser filtrada satisfaça a condição, será retomada uma lista vazia (`[]`)

Filtragem

As operações de filtragem são operações que reduzem todos os elementos da lista a um único valor baseado na função ou expressão utilizada. Também se faz uso de um **acumulador**, que aqui, soma e retoma o valor desejado ao realizar a operação de redução (como a soma de todos os itens ou a concatenação de strings)

```
const listaString = ['The', 'Quick', 'Brown', 'Fox', 'Jumps', 'Over',
'The', 'Lazy', 'Dog']
const concat = (acc, x) => `${acc} ${x}`
const reducaoConcat = (lista) => lista.reduce(concat,'')

console.log(reducaoConcat(listaFrase))
```

A operação acima concatena os itens da lista (todos de tipo `string`) em apenas uma `string` com um espaço entre cada item. Ela retorna

```
'The Quick Brown Fox Jumps Over The Lazy Dog'
```

Perceba que: * O `acc` "junta" os itens de cada lista até formar a frase final. * O `acc` também é operado e alterado diretamente pela operação de redução * Dentro da

declaração da redução, o que vem após a vírgula (no exemplo acima, `''`) é o valor inicial do acumulador.

As três operações conceituadas podem ser usadas em conjunto para fazer diversas operações com listas, possibilitando a ampla manipulação desses valores.

Registros

Os registros são estruturas de coleção de dados que permitem o agrupamento de informações de diversos tipos ou aspectos de um mesmo objeto.

```
const livro = {
    nome: 'Quinze Dias',
    autor: 'Vitor Martins',
    ano: 2017,
    ISBN: 8525063150,
    temas: ['Romance', 'Literatura Brasileira', 'Jovem-Adulto']
}
```

O exemplo acima é um registro de um livro. Os atributos da lista podem ser acessados da seguinte forma:

```
//Ex.: nome
console.log(livro.nome)
console.log(livro.ISBN)
```

Isso retorna

```
'Quinze Dias'
8525063150
```

É possível criar listas de registros

```
const livros = [
    {
    nome: "Quinze Dias",
    autor: "Vitor Martins",
    ano: 2017,
    ISBN: 8525063150,
    temas: ['Romance', 'Literatura Brasileira']},
    {
    nome: 'O Vampiro que Descobriu o Brasil',
    autor: 'Ivan Jaf',
    ano: 1999,
    ISBN: 9788508111176,
    temas: ['Ficção','História do Brasil', 'Literatura Brasileira']}
    ]
```

Para acessar os itens dos registros, pode-se utilizar-se da seguinte sintaxe

```
console.log(livros[1].nome)
```

O comando acima retoma:

```
'O Vampiro que Descobriu o Brasil'
```

Também é possível, quando relevante, utilizar registros dentro de outros registros

```
const pessoa = {
    nome: 'Fernando',
    idade: 24,
    medioConcluido: true,
    endereco: {
        rua: 'Etelvino de Souza',
        numero: 44,
        bairro: 'Jabotiana',
        cidade: 'Aracaju',
        estado: 'Sergipe'
    }
}
```

Para acessar as informações de registros dentro de registros, utiliza-se

```
console.log(pessoa.endereco.rua)
```

O código retorna

```
'Etelvino de Souza'
```

Imutabilidade de Listas e Registros

O paradigma funcional considera que todos os objetos são **imutáveis** (por isso a declaração com uso de `const`), e por isso não podem ser modificados após declarados... Exceto listas e registros.

Listas e registros, mesmo com a declaração usando `const`, podem ter seus valores modificados, como no exemplo abaixo

```
const lista1 = [1,4,3,10,6]
const lista2 = lista1.sort((a,b) => a-b)
console.log(lista2)
console.log(lista1)
```

Ao contrário do que se espera, ambas as listas foram operadas.

```
[1,3,4,6,10] //Lista2
[1,3,4,6,10] //Lista1
```

Isso ocorre porque `lista2` não é uma nova lista, e sim um novo "nome" para a `lista1`. Então a lista pode ser chamada pelos dois "nomes" atribuidos a ela. Isso prova que, mesmo que a estrutura da lista em si seja imutável, o conteúdo pode ser modificado.

Para evitar que isso aconteça, é possível fazer o seguinte:

Fazer uma cópia da lista utilizando parâmetros Spread `[...x]`

```
const lista1 = [1,4,3,10,6]
const lista2 = [...lista1].sort((a,b) => a-b)
console.log(lista2)
console.log(lista1)
```

Com o uso do parâmetro **Spread**, uma nova lista é criada e operada sem interferências na original.

```
[1,3,4,6,10] //Lista2
[1,4,3,10,6] //Lista1
```

Utilizar o método `Object.freeze()`

O recurso `Object.freeze()` congela os itens das listas (e dos registros), retornando a lista não modificada, mesmo que se tenha associado outro nome e modificado ela. Isso garante o princípio de imutabilidade para listas requerido pelo paradigma funcional. Para acessar a lista e criar uma nova lista com ela, utiliza-se o método anterior para copiá-la.

```
const lista1 = Object.freeze([1,4,3,10,6])
const lista2 = lista1.sort() //retorna a lista original
```

Uso de operações de mapeamento, filtragem e redução, etc

```
const lista1 = [1,4,3,10,6]
const lista2 = lista1.map((x) => x**2)
console.log(lista2)
console.log(lista1)
```

As operações com listas retomam sempre uma nova lista com os novos valores

```
[1,16,9,100,36] //Lista2
[1,4,3,10,6] //Lista1
```

Outra operação que cria uma nova lista é o `slice()`, que cria uma nova lista a partir de uma seleção de um ponto inicial e um ponto final em qualquer índice da lista.

```
const lista1 = ['Eu', 'amo', 'pizza','de','queijo']
const lista2 = lista1.slice(2,5)
console.log(lista2)
console.log(lista1)
```

O exemplo acima retorna:

```
[ 'pizza', 'de', 'queijo' ]
[ 'Eu', 'amo', 'pizza', 'de', 'queijo' ]
```

Registros

No caso de registros, as opções de imutabilidade se assemelham com as listas. Aqui temos o `Object.freeze()` e o `{...x}` (Spread). Note que aqui, o parâmetro Spread utiliza chaves, indicando que um registro está sendo copiado.

```
const cachorro = Object.freeze({
    nome: 'Luma',
    raça: 'Poodle',
    idade: 6,
    cor: 'preto',
    peso: 7.2
})

const cachorro2 = {...cachorro} //Spread

cachorro2.nome = 'Doki'
cachorro2.peso = 8.1

console.log(cachorro2)
console.log(cachorro)
```

O código retorna

```
{ nome: 'Doki', 'raça': 'Poodle', idade: 6, cor: 'preto', peso: 8.1 }
{ nome: 'Luma', 'raça': 'Poodle', idade: 6, cor: 'preto', peso: 7.2 }
```