Programação recursiva em listas

PADRÃO 1: realizar uma operação de redução numérica em uma lista de valores.

[EXEMPLO] Escreva um programa para calcular a soma dos valores de uma lista. Ex: $\{3, 8, 20, 21, 34, 44\}$.

Formulação recursiva

$soma(\{\}) = 0$

$soma(\{a_1 ... a_n\}) = a_1 + soma(\{...a_n\})$

Implementação em Javascript

```
const soma = (lista) => {
    if (lista.length == 0) {return 0}
    else {
        const head = lista.slice(0,1)[0]
        const tail = lista.slice(1)
        return (head + soma(tail))
    }
}
```

Versão com uso do já conhecido operador ***spread*** para listas.

```
const soma = (lista) => {
    if (lista.length == 0) {return 0}
    else {
    const [x, ...xs] = lista;
    return x + soma(xs)
    }
}
```

Versão com teste acerca de *valor indefinido* e *operador condicional ternário*

```
const indef = x => typeof x == 'undefined'
const soma = ([x, ...xs]) => indef(x) ? 0 : x + soma(xs)
```

PADRÃO 2: retornar o elemento de uma lista que atenda a um determinado critério.

[EXEMPLO] Encontrar o último elemento de uma lista qualquer passada.

Formulação recursiva

$ultimo(\{\}) = vazia$

$ultimo(\{a_1\}) = a_1$

$ultimo(\{a_1 ... a_n\}) = ultimo(\{...a_n\})$

Uso de *função de interface* para tratar listas vazias

```
const ultimo = ([x, ...xs]) => {
    if (indef(x)) return 'Lista vazia'
    else return ultimoAux([x, ...xs])
}
const ultimoAux = ([x, ...xs]) => {
    if (xs.length == 0) return x
    else return ultimoAux(xs)
}
```

[EXEMPLO] Encontrar o maior elemento de uma lista.

Formulação recursiva

$maior(\{\}) = vazia$

$maior(\{a_1\}) = a_1$

$maior(\{a_1 ... a_n\}) = a_1, \text{ se } a_1 > maior(\{...a_n\})$

```
const maior = ([x,...xs]) => {
    if (indef(x)) {return 'Lista vazia'}
    else return maiorAux([x,...xs])
}
const maiorAux = ([x,...xs]) => {
    if (xs.length == 0) return x
    else {
        const maior = maiorAux([...xs])
        return (x > maior) ? x : maior
    }
}
```

PADRÃO 3: realizar operações sobre elementos de uma lista, gerando uma nova lista.

[EXEMPLO] Inverter a ordem dos elementos de uma lista.

Formulação recursiva

$inverte(\{\}) = \{\}$

$inverte(\{a_1 ... a_n\}) = \{inverte(\{...a_n\}), a_1\}$

```
const inverte = ([x, ...xs]) =>
    indef(x) ? [] : [...inverte(xs), x]
```

[EXEMPLO] Duplicar a presença de cada elemento de uma lista.

Formulação recursiva

$duplica(\{\}) = \{\}$

$duplica(\{a_1 ... a_n\}) = \{a_1, a_1, ...duplica(\{...a_n\})\}$

```
const indef = x => typeof x == 'undefined'
const duplica = ([x, ...xs]) =>
    indef(x) ? [] : [x,x,...duplica(xs)]
```

PADRÃO 4: verificar se uma lista possui um elemento que atenda a uma dada propriedade/característica.

[EXEMPLO] Verificar se uma lista possui um determinado elemento.

Formulação recursiva

$elem(e, \{\}) = F$

$elem(e, \{a_1 ... a_n\}) = (e = a_1) \vee elem(e, \{...a_n\})$

```
const elem = (e,[x,...xs]) => {
    if (indef(x)) {return false}
    else return (e===x) || elem(e,[...xs])
}
```

PADRÃO 5: verificar se a lista atende a uma determinada propriedade.

[EXEMPLO] Testar se uma string consiste num *palíndromo*.

Formulação recursiva

$$palindromo('') = T$$
$$palindromo('char1') = T$$
$$palindromo('char_1 ... meio ... char'_n) = (char_1 = char_n) \wedge palindromo('meio')$$

```
const palindromo = (str) => {
    if (str.length < 2) return true
    else {
        const primeiro = str.slice(0,1)
        const ultimo = str.slice(-1)
        const meio = str.slice(1,-1)
        return (primeiro===ultimo) && palindromo(meio)
    }
}
```

INTERFACE HTML/CSS para os exemplos

Interface para exercícios de recursividade em listas

Entrada:

Escolher operação:

- somar valores
- último valor
- maior valor
- inverter lista
- duplicar valores da lista
- contém elemento?
- é palíndromo?

executa

Saída: