








Lista 07 - Recursividade em listas

Para cada um dos problemas a seguir, **encontre uma fórmula recursiva** apropriada, que contemple **caso base** e **passo indutivo**. A seguir, implemente sua formulação recursiva na linguagem Javascript.

⚠ caso deseje, você pode optar pelo uso das representações `Array.prototype` e `String.prototype` na definição de funções para uso em listas e strings, respectivamente. Essa representação permite que a função criada seja utilizada **a partir da** lista (ou string) de interesse, tornando desnecessário passar essa lista (ou string) como argumento da função no momento de seu uso. Obs: nesse caso, não é possível utilizar a representação de *funções como expressões*.


-  Q1. Criar uma função recursiva chamada `busca` que busca o índice de um determinado item em uma lista **ORDENADA**. Se o item estiver presente, ele deve retornar o índice, caso contrário, deve retornar -1.
-  Q2. Criar uma função que encontre o **maior** valor numa lista de inteiros usando a recursividade. Considere que a lista possui pelo menos um elemento.
-  Q3. Criar uma função chamada `somaAninhado` que pega uma lista e devolve a soma de todos os itens. Atenção: o item de uma lista pode ser outra lista.
-  Q4. Função para retornar o número de `vogais` numa string. Considere que todas as letras estão minúsculas.
-  Q5. Criar uma função que transforma frases terminadas com múltiplos pontos de interrogação ? ou pontos de exclamação ! numa frase que termina apenas com um, sem alterar a pontuação no meio das frases. Ex:
`eliminaIntExc("O que é isso????") ---> "O que é isso?"`
-  Q6. Considere o problema de gerar o montante acumulado de casos de uma doença qualquer. Como entrada, existe uma sequência de valores onde cada termo representa o número de casos daquele dia. Ex: para a sequência 7, 3, 19, 5, 15, 10 seria gerada a sequência 7, 10, 29, 34, 49, 59.
-  Q7. Pesquise sobre o algoritmo de ordenação QUICKSORT e o implemente; complemente o template de solução a seguir.


Formulação recursiva


$qsort(\{\}) = \{\}$


$qsort(\{a_1 \dots a_n\}) = ???$


```
const indef = x => typeof x == 'undefined'
...
...
const qSort = ([x, ...xs]) => {
  if (indef(x)) {return []}
  else {
    return (???)
  }
}
```


 **Q8.** Escreva um programa para realizar uma **Busca Binária** em uma lista de elementos ordenados. Se o elemento existir, retorne sua posição na lista; caso contrário, retorne -1 (ou algum outro tipo de indicativo de inexistência). Exemplos: `[-4,0,3,7,11].buscabin(7) ---> 3`; `[-4,0,3,7,11].buscabin(1) ---> -1` (ou NaN) .


 **Q9.** Função para eliminar de uma lista todas as ocorrências de um dado elemento. Exemplo:
`[-4,0,3,7,11].elimina(7) ---> [-4,0,3,11]` .

 **Q10.** Defina uma função que, dada uma lista, retorne a (sub)lista contendo os elementos que ocorrem exatamente uma vez na lista original. Exemplos: `[4,2,1,3,2,3].unicos() ---> [4,1]` ;
`[1,4,4,2,1,3,2,3].unicos() ---> []`

 **Q11.** Uma string é uma substring de outra se os elementos da primeira ocorrem na segunda, na mesma ordem. Por exemplo, 'ship' é uma substring de 'Fish & Chips' , mas não de 'hippies' . Defina uma função que decida se uma string é uma substring de outra.

 **Q12.** A propriedade `length` de uma lista retorna o número de elementos presentes na lista. Por exemplo, a lista `[1, [2, 3]]` possui 2 elementos: o número 1 e a lista `[2, 3]` . Suponha que ao invés disso, queiramos saber o total de elementos não aninhados na lista. No exemplo anterior, então, teríamos 3 elementos: números 1 , 2 e 3 . Escreva uma função para realizar esta última contagem.

 **Q13.** Crie uma função que pega um número e retorna um dígito que é o resultado da soma de todos os dígitos do número de entrada. Quando a soma dos dígitos resultar em mais de um dígito, repita a soma até obter um único dígito. Exemplos: `digitosoma(123) → 6` , `digitosoma(999888777) → 72 → 9` ,
`digitosoma(999999999998) → 107 → 8` .

 **Q14.** Crie um programa para representar um joguinho de dados simples. O usuário escolhe quantas vezes ele deseja jogar um par de dados. A pontuação será a soma total de todos os lançamentos. Entretanto, se algum lançamento gerar um valor duplo, a pontuação total será zerada. Represente o lançamento de cada par de dados como uma lista com dois elementos e o conjunto total de lançamentos seria então uma lista de listas. Exemplos: `pontuacao([[5, 2], [4, 3], [5, 6]]) → 25` , `pontuacao([[5, 6], [1, 1], [6, 4], [6, 3]]) → 0` .



Q15. Implementar sua própria função de alta ordem `map(...)` .



Q16. Implementar sua própria função de alta ordem `filter(...)` .



Q17. Implementar sua própria função de alta ordem `reduce(...)` .