Universidad del Valle

Práctica 6

Fecha límite de entrega: 21 de abril de 2021

Desarrolle los siguientes ejercicios los cuales se deben tener en cuenta: listas, listas de listas, condicionales y recursividad utilizando el paradigma funcional y lenguaje de programación JavaScript. Usted debe enviar el código fuente y pasar los test a través de la plataforma INGInious M-IDEA (http://ingin.ddns.net/courselist). Puede apoyarse de la herramienta repl.it (https://repl.it/~)

Documento de repaso
https://docs.google.com/document/d/1kWkNKLZ9dak7lEcAwWuF0Eq4uah1AJ4UhzOO9Oz0G_U/edit?usp=sharing

IMPORTANTE

La recursión en una lista es similar a la recursión numérica, pero a diferencia en lugar de reducir el número en cada llamado recursivo (en cada paso) se debe tratar el resto de la lista (reducir la lista en cada paso) hasta que tengamos una lista vacía. Implemente en JavaScript los siguientes desafíos y ponga prueba o adquiera los conocimientos para realizar funciones recursivas con listas.

Como el lenguaje de curso (JavaScript) no incorpora el tipo de datos listas, es necesario incluir la librería functional-light.

const {cons, first, rest, isEmpty, isList, length} = require('functional-light');

Los ejemplos de entrada e invocación se presentan con vectores, pero la implementación de todas las funciones deben considerar listas de acuerdo a lo visto en clase.

Recuerde no usar .concat() .reverse() y cualquier otro método que simplifique la operación entre listas

1 Encuentre el mayor valor de una lista de números

Desarrolle la función que encuentre el mayor de los valores numéricos en una lista

código/solución 1

```
function mayor(lista) {
    if (isEmpty(lista)) {
        return "null";
    } else if (length(lista) == 1) {
        return first(lista)
    } else {
        return Math.max(first(lista), mayor(rest(lista)))
    }
}
```

código/solución 2

```
function mayorL(lista) {
    if (isEmpty(lista)) return null;
    if (isEmpty(rest(lista))) return first(lista);
    else if (first(lista) > mayorL(rest(lista))) return first(lista);
    else return mayorL(rest(lista));
}
```

Comprueba las salidas

Entradas

```
mayorL([]);
mayorL([1,6,2]);
mayorL([-9]);
mayorL([1,6,20,3,-6, 8, 9]);
```

Salidas

```
null
?
-9
?
```

2 Encuentre el promedio de los valores de la lista

Desarrolle la función que encuentre el promedio de los valores numéricos en una lista

código/solución

```
function suma(lista) {
    if (isEmpty(lista)) {
        return (0)
    } else {
        return (first(lista) + suma(rest(lista)))
    }
}

function promedio(lista) {
    if (isEmpty(lista)) {
        return ("null")
    } else {
        return (suma(lista) / length(lista))
    }
}
```

Comprueba las salidas

Entradas

```
promedio([]);
promedio([5]);
promedio([0, 1, 8]);
promedio([10,2,3,1,4,4,4]);
```

Salidas

```
null
?
3
?
```

Universidad del Valle

3 Concatenar 2 listas

Desarrolle la función que concatene dos listas

Entradas

```
concat([],[]);
concat([],[1, 2]);
concat([3,5], []);
concat([1,2,true],[“FDP”,3,8,2,7]);
```

Salidas

```
[]
?
[3,5]
?
```

4 Invierta el orden de una lista

Desarrolle la función que invierta el orden de los elementos de una lista

Entradas

```
invertir([]);
invertir([1]);
invertir([4,1,2,7,4,1]);
invertir([1,2,1]);
```

Salidas

```
[]
?
?
[1,2,1]
```

5 Genere la lista de los primeros n términos de la serie de Fibonacci

Desarrolle la función que genere una lista con los n primeros n de la serie fibonacci

Entradas

```
fibolist(0);
fibolist(1);
fibolist(2);
fibolist(3);
fibolist(6);
```

Salidas

```
[]
[0]
?
[ 0, 1, 1 ]
[ 0, 1, 1, 2, 3, 5 ]
```

6 Copia una lista de varios niveles de manera recursiva

Desarrolle la función que copia una lista de varios niveles de manera recursiva.

Entradas

```
copy([]);
copy([1,2,3]);
copy([1, 2, [3, 4], [1]]);
copy([[9]]);
```

Salidas

```
[]
?
?
[[9]]
```

7 Eliminar todos los elementos que no sean números

Desarrolle la función que elimine todos los elementos que no sean numéricos

Entradas

```
soloNum([]);
soloNum(['a']);
soloNum(['a', 1]);
soloNum([6, 'b']);
soloNum([6, 'b', 1, 2, true, 'b', false, -2]);
```

Salidas

```
?
[]
?
[6]
?
```

8 Insertar un elemento x en la posición n de la lista

Desarrolle la función que inserta un elemento elem en la posición x de la lista. Inserta el elemento si x >= 0 y x <= longitud de la lista. Ayuda: los argumentos son elem, posición, lista

Entradas

```
insertX(7,0,[1,2,3])
insertX(7,1,[1,2,3])
insertX(8,6,[1,0,3,5,9,3])
insertX(8,7,[1,0,3,5,9,3])
insertX(8,-1,[1,0,3,5,9,3])
insertX(7,0,[])
```

Salidas

```
[7, 1, 2, 3]
?
?
?
?
[7]
```

9 Cambia el elemento que está en la posición n de la lista, por el nuevo valor x

Desarrolle la función que reemplace el elemento de la posición x de la lista por el elemento elem. Reemplaza el elemento si x > 0 y x < longitud de la lista. Ayuda: los argumentos son posición, lista, elem

Entradas

```
replaceX(0,[1,2,3],7)
replaceX(1,[1,2,3],7)
replaceX(5,[1,0,3,5,9,3], 8)
replaceX(-1,[1,0,3,5,9,3], 8)
replaceX(0,[], 7)
```

Salidas

```
[7, 2, 3]
?
?
?
[]
```

10 Retorna el índice n de dónde se encuentra un número x

Implementar una función que retorne el índice n, donde se encuentra el elemento elem si existe. La lista debe estar ordenada y si el elemento no existe retorna -(n + 1), donde n es la posición en la cual se debería insertar elem para mantener la lista ordenada.

Entradas

```
lookupx([1,2,3,4], 2);
lookupx([1,2,3,4], 3.5);
lookupx([], 1.1);
lookupx([2,4,6,10], 10);
lookupx([2,4,6,10], 11);
```

Salidas

```
1
?
?
3
?
```

Universidad del Valle

11 Inserta datos en una lista que siempre está ordenada de manera ascendente

Desarrolle la función que inserta datos en una lista que siempre está ordenada de manera ascendente

Entradas

```
insertInOrder([], -2.3);
insertInOrder([-2, 3, 5, 5, 6 ], 4);
insertInOrder([-2, 3, 5, 5, 6 ], 3);
insertInOrder([-2, 3, 5, 5, 6 ], 8);
insertInOrder([-2, 3, 5, 5, 6 ], -3);
insertInOrder([-2, 3, 5, 5, 6 ], -2);
insertInOrder([-2, 3, 5, 5, 6 ], 5.2);
```

Salidas

```
[-2.3]
?
?
?
?
?
[-2, 3, 5, 5, 5.2, 6]
```

12 busca un elemento en una lista desordenada

Desarrolle la función que retorna el índice de un elemento en una lista., retorna -1 en caso de no encontrarlo

Entradas

```
lookUp([["3",4],5,3,4], 3)
lookUp([1,2,3,4], 3.5)
lookUp(["a"], 1.1)
lookUp([11,25, 1, 6,10], 6)
```

Salidas

```
2
?
?
?
```

13 elimina el elemento n de la lista

Desarrolle la función que elimina el elemento n (índice) de la lista.

Entradas

```
delete([1,2,3], 7);
delete([1,2,3], 1);
delete([1,0,3,5,9,3], 5);
delete([1,0,3,5,9,3], 0);
delete([], 0);
delete([2, 4], -1);
```

Salidas

```
[1, 2, 3]
?
[ 1, 0, 3, 5, 9 ]
[ 0, 3, 5, 9, 3 ]
[]
?
```

14 Ordene de manera ascendente una lista

Desarrolle la función que ordene de manera ascendente una lista. Si tiene sublistas, las ordena también

Entradas

```
sortaL([3,5,3,4])
sortaL([1,2,3,4])
sortaL([[4,3,2,1], 8])
sortaL([])
```

Salidas

```
?
[ 1, 2, 3, 4 ]
[ [ 1, 2, 3, 4 ], 8 ]
?
```

15 Busca todos los elementos de una lista que cumplen una cierta condición (extra)

Desarrolle la función que implemente una función que busca todos los elementos de una lista que cumplen una cierta condición, por ejemplo, los números que sean pares. La función debe retornar una lista con los elementos encontrados.

Entradas

```
filter([1, 3, 4, 6, 0, 1], (a) => a > 3);
filter([“ab”, “acv”, “gggg”, “cds”], (a) => a.l
```

Salidas

```
[4, 6]
?
```

16 Función data a todos los elementos de una lista(map) (extra)

Desarrolle la función que aplica una función data a todos los elementos de una lista(map). Por ejemplo, la función debe ser capaz de elevar todos los elementos de la lista al cuadrado.

Entradas

```
map([1, 2, 3, 4], (x) => x * x);
map([{a: “María”}, {a: “Carlos”}], (v) => “Hola
```

Salidas

```
[1, 4, 9, 16]
?
```

17 función que aplica una función a una lista genérica (extra)

Desarrolle la función que aplica una función a una lista genérica, y agrega todos los valores de la lista para producir una única salida

Entradas

```
reduce([1, 2, 3, 4, 5], (val, acum) => acum + v
reduce([1, 2, 3, 4, 5], (val, acum) => acum * v
reduce([1, 7, 3, 14, 5], (val, acum) => Math.ma
reduce([1, 7, 3, 14, 5], (val, acum) => Math.mi
reduce([{a: “María”}, {a: “Carlos”}], (val, acu
```

Salidas

```
15
120
?
1
?
```