Datos compuestos

Listas

Listas

Una lista es un conjunto de elementos que tienen asociado un orden

Ejemplos:

- Lista de compras
- Lista de estudiantes
- Lista de actividades

Listas

Conjunto

Fila

Las listas tienen un orden dado. Pueden ser de cualquier longitud, pero deben tener un número finito de elementos

Creando listas

- Existe una lista especial que no tiene elementos. Se llama la lista **vacía**
- En JavaScript esta lista se denota como []
- Podemos crear una lista en DrRacket mediante:
  - cons(‘rojo’ []); construct a list with 1 element
- Podemos agregar mas elementos a una lista
  - cons (‘naranja’, cons (‘rojo’ [])); construct a list with 2 elements
  - cons (‘verde’, cons (‘naranja’, (cons ‘rojo’ []))); construct a list with 3 elements
  - cons (1, cons (2, cons (3 []))); construct a list with 3 elements
- Los elementos de la lista no necesariamente deben ser del mismo tipo
  - cons (true, cons (2, cons (‘rojo’ []))); construct a list with 3 elements

Operaciones básicas con listas

- Dada una lista de varios elementos hay 2 operaciones de selección básicos

  First:

  first(cons(1, cons(2, cons(3, []))))

  Retorna el primer elemento de la lista. En este caso 1

  Rest

  rest(cons(1, cons(2, cons(3, []))))

  Retorna los demás elementos de la lista, quitando el primero: cons (2, cons(3,[]))

Operaciones básicas con listas

isEmpty: ¿Es una lista vacía? ¿Hay más elementos?

Ejemplos:

```
isEmpty(cons(1, cons(2, cons(3, []))))

>false

isEmpty(first(cons(3, [])))

>false

isEmpty(rest(cons(3, [])))

>false
```

Operaciones básicas con listas

isList: ¿Es una lista? ¿Hay más elementos en la lista?

Ejemplos:

isList(cons(1, cons(2, cons(3, [])))))

>true

isList(first (cons(3, [])))

>false

isList(rest (cons(, [])))

>false

Pero afortunadamente en JavaScript...

Podemos simplemente crear una lista de la siguiente manera:

[1, 2, 3, 4, 5, 6, 7]

const my-list = [1, 2, ‘a’, ‘b’, true, false]

En algunos lenguajes funcionales la función **car es** equivalente a nuestro **first,** y la función **cdr** es equivalente a nuestro **rest**

Longitud de una lista

Con la función longitud podemos saber cuántos elementos hay en una lista.

```
function longitud(my-list) {
        if (isEmpty (my-list))
                return 0;
        return 1 + longitud( rest(my-list));
}
```

Problemas propuestos

1. Encuentre el mayor valor de una lista de números
2. Encuentre el promedio de los valores de la lista
3. Invierta el orden de una lista
4. Ordene de manera ascendente una lista
5. Genere la lista de los primeros ***n*** términos de la serie de Fibonacci
6. Dada una lista, eliminar todos los elementos que no sean números
7. Implemente una función que inserta un elemento ***x*** en la posición ***n*** de la lista, si **n** está entre **0** y el **(longitud lista).** No hace nada en caso contrario.
8. Dada una lista ordenada, implementar una función que retorna el índice ***n*** de dónde se encuentra un número ***x*** dado, si existe, o ***-(n + 1)*****,** donde ***n*** es la posición en la cual se debería insertar ***x*** para mantener la lista ordenada.
9. Implemente una función que inserta datos en una lista que siempre está ordenada.
10. Implemente una función que busca un elemento en una lista desordenada.
11. Implemente una función que elimina el elemento ***n*** de la lista

Crear un nuevo proyecto con npm

Para comenzar a trabajar en un proyecto de NodeJS, la forma mas sencilla es dejar que npm nos ayude a crear el archivo de configuración del paquete. Para esto hacemos los siguiente:

```
mkdir proyecto1

cd proyecto1

npm init
```

Luego respondemos las preguntas del asistente y veremos al final del proceso un archivo package.json en nuestra carpeta del proyecto

Crear un nuevo proyecto con npm

Al final deben tener un archivo como este:

```
{
  "name": "goo",
  "version": "1.0.0",
  "description": "Ejercicios listas y recursion",
  "main": "index.js",
  "scripts": {
    "test": "jest"
  },
  "author": "Andrés M. Castillo",
  "license": "MIT"
}
```

Crear un nuevo proyecto con npm

Agregamos estas dependencias a nuestro archivo package.json

```
"devDependencies": {
  "jest": "^24.5.0"
},
"dependencies": {
    "functional-light": "0.1.0"
 }
```

E instalamos las dependencias usando npm. Esto se hace usando el comando:

```
npm install
```

Crear un nuevo proyecto con npm

Ahora creamos nuestra carpeta src y dentro de ella creamos el archivo index.js y pegamos el ejemplo disponible en https://github.com/andcastillo/functional-light

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

console.log(cons('1',[])); // ['1']
console.log(cons('2', cons('1',[]))); // ['2', '1']
console.log(isList(cons('1',[]))); // TRUE
console.log(isList({length: false })); // false
console.log(isEmpty(cons('1',[]))); // false
console.log(isEmpty([])); // true
console.log(isEmpty(9)); // false
console.log(rest(cons(1, cons(2, [])))); // [2]
console.log(rest([])); // []
console.log(cons(1, [2, 3])); // []

const foo = cons(484, []);
console.log(cons('XX', foo))
  console.log(foo); // Debe imprimir [484]
```

```
(define (fib-serie n)
  (if (= n 0) (list 0)
      (if (= n 1) (list 1 0)
          (append (list (+ (first (fib-serie (- n 1)))
                           (first (rest (fib-serie (- n 1))))))
                  (fib-serie(- n 1)))
)))
```