**RECURSIÓN**

Definir las funciones recursivas nFactorial y nFibonacci.

nFactorial(n) debe retornar el factorial de n sabiendo que, siendo n un número natural, su factorial (representado como n!) es el producto de n por todos los números naturales menores que él y mayores a 0. Ejemplo: 5! = 5 \* 4 \* 3 \* 2 \* 1

nFibonacci(n) debe retornar el enésimo número de la secuencia de Fibonacci, tomando al 0 y al 1, respectivamente, como primer y segundo elementos de la misma, y sabiendo que cualquier elemento que se agregue a esta secuencia será el resultado de la suma del último elemento y el anterior.

Ejemplo: nFibonacci(7) retornará 13, ya que 13 es el dígito que está en la posición 7 de la secuencia.

Secuencia:  0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

Como ejercicio adicional y completamente opcional, al terminar de resolver este problema pueden intentar definir funciones que logren los mismos resultados pero de manera iterativa.

\*/

function nFactorial(val) {

  if (val <= 1) return 1;

  return val \* nFactorial(val - 1);

}

function nFibonacci(n) {

  if (n <= 1) return n;

  return  nFibonacci(n - 1) + nFibonacci(n - 2);

}

Implementar la clase Queue, sabiendo que es una estructura de tipo FIFO, donde el primer elemento que ingresa es el primero que se quita. Definir los siguientes métodos:

  - enqueue: agrega un valor respetando el orden.

  - dequeue: remueve un valor respetando el orden. Retorna undefined cuando la queue está vacía.

  - size: retorna el tamaño (cantidad de elementos) de la queue.

Pueden utilizar class o función constructora.

\*/

function Queue() {

  this.datos = [];

}

Queue.prototype.enqueue = function (val) {

  return this.datos.unshift(val)

}

Queue.prototype.dequeue = function (val) {

  return this.datos.pop(val)

}

Queue.prototype.size = function () {

  return this.datos.length

}