HENRY

**DÍA 4**

**ESTRUCTURACIÓN DE DATOS I**

**RECURSIÓN**

**Ejercicio 1:**

Funcion que cuenta los elementos dentro de una Array, incluyendo las sub arrays.

var miArr=[1,2,3,4,[5,6]];

function n(arr) {

let count=0;

for(const element of arr) {

if (Array.isArray(element)) {

**count += n(element);**

} else {

count++;

}

}

return count

}

console.log(n(miArr));

**Ejercicio 2:**

Función que divide un número por 2 las veces que pueda hasta llegar a cero. (Caso base, 0 o 1 son 0)

function div(val) {

let count=0;

if (val < 2) return 0;

const newVal = Math.floor(val/2);

count++;

**count += div(newVal);**

return count

}

console.log(div(10));

**ESTRUCTURAS DE DATOS PARTE I**

Estructura de datos nos referimos a cómo se organizan los datos cuando programamos. En un cache quiero almacenar datos de tipo clave – valor.

Tipos de estructuras: Arreglos, Sets (Arreglo que no repite valores)

Stacks (Pilas) => LIFO (Last in, First Out) (push-pop)

Queue (Colas) => FIFO (First in, First Out) (push-shift)

Colas con el métodos de array

Function Queue() {

this.data = []; //Crea una clase

}

Queue.prototype.enqueue = function(val) { //Metodos

this.data.unshift(val)

}

Queue.prototype.dequeue = function(val) { //Metodos

this.data.pop(val)

}

Queue.prototype.size = function(val) { //Metodos

this.data.length

}

HOMEWORK

/\*

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) {

let resultado=0;

if (val <= 1) return 1;

resultado += val \* (val-1);

resultado \*= nFactorial(val-2)

return resultado

}

function nFibonacci(n) {

let miFibo = 0

if (n <= 0) return 0;

if (n === 1) return 1;

miFibo = nFibonacci(n - 1) + nFibonacci(n - 2);

return miFibo;

}

/\*

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

}