**CLOSURES**

**Closures** => Funciones q retornan funciones. Un ***closure*** es una función que retorna otra función, que hace uso de los argumentos de la función creadora (padre). Lo que crea, genera un nuevo contexto de ejecución que está enlazado al contexto de ejecución del padre. Mas generalmente, una Closure es una función que devuelve funciones, sea una función, un objeto con funciones o un array de funciones que mantiene un contexto de ejecución con algunas variables que son heredadas.

function BinarioADecimal(num) {

  // tu codigo aca

  let n = num.toString();

  let sum = 0;

  for (let i = 0; i < n.length; i++) {

    sum += n[n.length-1-i] \* Math.pow(2,i);

  }

  return sum;

}

function DecimalABinario(num) {

  // tu codigo aca

  let bina = [];

  let n = num;

  while (n > 0) {

    bina.unshift(n % 2)

    n = Math.floor(n / 2);

  }

  let numBina = bina.toString();

  return numBina.replace(/,/g, "");

}

function counter() {

  /\*

  Ejercicio 1

  La función counter debe retornar otra función. Esta función retornada debe actuar como un contador, retornando un valor numérico que empieza en 1 e incrementa con cada invocación.

  Ejemplo:

  const nuevoContador = counter()

  nuevoContador()     // 1

  nuevoContador()     // 2

  nuevoContador()     // 3

  const otroContador = counter()

  otroContador()      // 1

  otroContador()      // 2

  otroContador()      // 3

   \*/

  let count = 0;

  return function () {

    count++;

    return count;

  }

}

function cacheFunction(cb) {

  /\*

  Ejercicio 2

  Tu tarea aquí es lograr, mediante un closure, que cacheFunction actúe como una memoria caché para el callback que recibe por parámetro (cb); es decir, que "recuerde" el resultado de cada operación que hace, de manera que, al realizar una operación por segunda vez, se pueda obtener el resultado de esa "memoria" sin tener que efectuar otra vez cálculos que ya se hicieron anteriormente.

  cacheFunction debe retornar una función. Esta función debe aceptar un argumento (arg) e invocar a cb con ese argumento; hecho eso, debe guardar el argumento junto con el resultado de la invocación (tip: usá un objeto donde cada propiedad sea el argumento, y su valor el resultado de la correspondiente invocación a cb) de manera que, la próxima vez que reciba el mismo argumento, no sea necesario volver a invocar a cb, porque el resultado estará guardado en la "memoria caché".

  Ejemplo:

  function square(n){

    return n \* n

  }

  const squareCache = cacheFunction(square)

  squareCache(5)    // invocará a square(5), almacenará el resultado y lo retornará

  squareCache(5)    // no volverá a invocar a square, simplemente buscará en la caché cuál es el resultado de square(5) y lo retornará (tip: si usaste un objeto, podés usar hasOwnProperty)

  \*/

  var cacheObj = {};

  return function(arg) {

    if(cacheObj.hasOwnProperty(arg)) return cacheObj[arg];

    cacheObj[arg]=cb(arg);

    return cacheObj[arg];

    }

}

// Bind

var instructor = {

  nombre: "Franco",

  edad: 25,

};

var alumno = {

  nombre: "Juan",

  curso: "FullStack",

};

function getNombre() {

  return this.nombre;

}

/\*

  Ejercicio 3

  IMPORTANTE: no modificar el código de arriba (variables instructor y alumno, y función getNombre)

  Usando el método bind() guardar, en las dos variables declaradas a continuación, dos funciones que actúen como getNombre pero retornen el nombre del instructor y del alumno, respectivamente.

\*/

let getNombreInstructor = getNombre.bind(instructor);

let getNombreAlumno = getNombre.bind(alumno);

/\*

  Ejercicio 4

  Sin modificar la función crearCadena, usar bind para guardar, en las tres variables declaradas a continuación, tres funciones que retornen una cadena (string) y el delimitador especificado (asteriscos, guiones, guiones bajos, respectivamente). Las funciones obtenidas deberían recibir solamente un argumento - la cadena de texto - ya que los otros argumentos habrán sido "bindeados".

\*/

function crearCadena(delimitadorIzquierda, delimitadorDerecha, cadena) {

  return delimitadorIzquierda + cadena + delimitadorDerecha;

}

let textoAsteriscos = crearCadena.bind(null,"\*","\*");

let textoGuiones = crearCadena.bind(null,"-","-");

let textoUnderscore = crearCadena.bind(null,"\_","\_");

// EJERCICIO 1

// Implementar la funcion 'exponencial' que recibe un parametro entero 'exp'

// y retorna una una funcion, nos referiremos a esta ultima como funcion hija,

// y a 'exponencial' como la funcion padre, la funcion hija debe de recibir

// un parametro y retornar dicho parametro elevado al parametro 'exp' de

// la funcion padre original 'exponencial'

// Ejemplo:

// > var sqrt = exponencial(2);

// > sqrt(2);

// < 4

// > sqrt(3);

// < 9

// > sqrt(4);

// < 16

function exponencial(exp) {

    return function (n) {

        return Math.pow(n,exp)

    }

}