1. Crear una función que determine si al menos 3 parámetros (1 a 3) están siendo enviados que devuelva en un array los valores de estos 3 parámetros caso contrario, si no tuviera parámetros, que retorne el valor ‘null’. Ej: someFunction(3, {prop1: true}, false) -> [3, {prop1: true}, false]; someFunction() -> null
2. Crear una función que al enviar un tipo de dato Array u Objeto como parámetro guarde un nuevo valor (de cualquier tipo de dato) y muestre el resultado del parámetro (variable) enviado después de que la función haya sido invocada *(utilizar variables globales, el concepto de JS pasa por referencia Array y objetos)*. Ej: someFunction(varArr); (donde varArr = [1,2,3] es un array) -> [1,2,3,**1**]; someFunction (varObj) (donde varObj = {prop1: 1} es un objeto) -> {prop1: 1, **a: true**}
3. Crear una función que devuelva el resultado de una variable de tipo objeto de una función externa después que ha sido agregada 3 propiedades dentro de una función interna *(crear una variable de tipo objeto en la función externa e incrementar sus propiedades dentro de la función interna, invocar la función interna dentro de la funcion externa y mostrar el resultado. utilizar definición de scope)* someFunction() -> {a: 1, b: true, c: ‘How hard is to scope!’, d: [1,2,3] };
4. Crear una función que dada una variable de tipo número, está definida y con valor en una función externa realice operaciones matemáticas dentro de varias funciones internas, es decir, en la 1ra función interna que se multiplique por el doble del mismo número y se guarde en la misma variable, en la 2da función interna que se divida el mismo número con el número 2 y se guarde en la misma variable, y por último en la 3ra función interna que se reste el mismo número por el número 1, se guarde en la misma variable e imprima el resultado *(utilizar el concepto de scope y encadenamiento de scopes)*. Ej:  
     
   var init = initFunc(); //Invoca la función externa e inicializa la variable número digamos que es a = 8

var first = init(); //Invoca la primera función, realizando la primera operación matemática en la variable: a = (8\*8) = 64

var second = first(); //Invoca la segunda función, realizando la segunda operación

en la variable: a = (64/2) = 32

var third = second(); //Invoca la tercera función, realizando la tercera y última operación en la variable ‘a’ y muestra el resultado: a = 31

1. Crear una función que dado un número pasado como parámetro, devuelve en un array la multiplicación de ese mismo número con un número incrementado en uno (como una tabla de multiplicación). *(utilizar el concepto de Closure, dentro de la función externa iniciar una variable i = 1 y un vector v = []; dentro de la función interna realice la operación de multiplicación, guarde en el vector ‘v’ el resultado e incremente el valor de i en 1, por último que devuelva el vector en la función interna y la función externa retorna la función interna)* Ej:   
   var init = initProd(6);  
   init.mult() -> [6];  
   init.mult() -> [6, 12];  
   init.mult() -> [6, 12, 18];
2. Crear una función que simule un contador de números comenzando desde el cero hasta N veces, donde N es el número de veces que ha sido llamado una función interna. *(utilizar el concepto de Closure, iniciar una variable count = 0 dentro de la función externa y dentro de la función interna incrementa esta variable ‘count’ en uno y retorne ese incremento, al final que la función externa retorne la función interna)*. Ej: var make = makeCounter() //inicializa la variable count en la función externa make.doCount(); //1, doCount es el Closure ó función interna. make.doCount(); //2
3. Crear una función que reciba como parámetro un vector que contiene 4 elementos (Ej: [1, ‘IIFE’, 12 > 12, {prop1: 100}]), se declare un ciclo “for” o “while” para iterar sobre este vector y utilice el método **setTimeout**, que por cada segundo guarde el valor de cada elemento del vector dentro de un objeto vacío ({0: 1, 1: ‘IIFE’, 2: false, 3: {prop1: 100}}, utilizar el **índice del vector como nombre de propiedad del objeto**) y muestre el **valor** que se está insertando en el objeto por cada segundo transcurrido y por último mostrar el objeto después de haber realizado el ciclo. *(utilizar el concepto de IIFE, declarar un objeto vacío dentro de la función externa; en el ciclo, invocar y pasar como parámetro el índice del ciclo en el IIFE; en la función del método setTimeout, guardar el* ***índice como llave y valor (el elemento del vector)*** *en el objeto vacío y mostrar el valor)*Ej:

anIIFEExample([1, ‘IIFE’, 12 > 12, {prop1: 100}]) -> {0: 1, 1: ‘IIFE’, 2: false, 3: {prop1: 100}}

//Y por cada segundo transcurrido, muestre el valor de cada elemento del vector

1 //el primer valor

‘IIFE’ //el segundo valor

false, //el tercer valor