**RESOLUCIÓN EJERCICIOS MÓDULO 5**

**EJERCICIO 7: PALABRA A FRASE, LO UTILIZA CON MÉTODO REDUCE EN VEZ DE JOIN (PUEDE HACERSE CON CUALQUIERA)**

El parámetro “palabras” es un arreglo, y el método reduce es un método para arreglos. Va a hacer cierta operación, va a estar iterando sobre cierto arreglo y va a hacer alguna tarea en particular.

¿Qué haríamos entonces en este caso?

Vamos a definir una variable donde se guarde todo el proceso que vamos a hacer con el reduce. La vamos a llamar “frase” que es lo que tenemos que retornar, una frase o una oración que tenga todas las letras que están contenidas en el arreglo. Esta frase, se va a cargar con lo siguiente: Vamos a llamar a “palabras” que es nuestro arreglo y el método, se llama reduce. El reduce dentro de los paréntesis recibe (le tenemos que pasar por dentro) una función.

Ejemplo de explicación de clase: teníamos una variable llamada “números” que recibe un arreglo de números. Lo que hacía la función “sumatoria” en este caso, le pasábamos un acumulador y un elemento, y lo que hacíamos con el return era comenzar a acumular los valores sumados con cada uno de los elementos (recordemos que el elemento es como una palabra referencial, una variable muda que hace referencia a cada uno de los números que está dentro del arreglo de números, y al final, lo que retornaba era esa cantidad de hacer las sumas de todos los elementos que estaban dentro del arreglo). Y en la variable suma guardabamos (=) a números.reduce llamando a "sumatoria” (entre paréntesis) que es justamente lo que conocemos con **callback. Callback** es pasarle una función como argumento de otra función para que haga cierta tarea en particular y al final simplemente retornamos o mostramos por consola cierta información.

Este caso, se resuelve de la misma manera, solo que nuestro arreglo, es un arreglo de Strings.

Es decir, “palabras” tiene algo de este estilo [‘hola’, ‘adiós’, ‘que’, ‘tal’], y lo que queremos es devolver esta información, concatenada (unida): “hola adiós que tal”. Entonces reduce va a recibir una función de callback que va a tener un acumulador y un elemento (o un representante de los elementos que hacen parte del arreglo “palabras”. Entonces, ¿qué va a hacer mi función? Mi función va a ir palabra por palabra, cada uno de los elementos que tiene el arreglo palabras y va a estar haciendo una concatenación de la información, es decir, vamos a estar uniendo cada una de las palabras a través del acumulador. Es decir, mi función va a retornar el acumulador, y ese acumulador va a estar concatenando cada una de las palabras. Es decir, cuando encuentre la palabra “hola”, este hola va a tomar como las veces del acumulador, luego, pasaría a la siguiente palabra que sería “adiós”, entonces, vamos a estar concatenando “hola” + “adiós” (El signo + nos sirve para concatenar Strings). Entonces, cuando va palabra por palabra va teniendo en cuenta lo que ya estaba más la nueva información que le llega y esto lo une. Esto lo convierte en un solo string de esta manera “holaadios”, a la siguiente palabra (El acumulador va teniendo toda la información que le va llegando y cada vez que llegue una nueva palabra, simplemente se concatena”. Al final mi acumulador sería “holaadiosquetal”.

A la hora de retornar, se debería retornar el acumulador más un espacio entre comillas y sumando la palabra. Así en el resultado final, se tendrán las palabras separadas por un espacio entre cada una de ellas.

**EJERCICIO 8: ARRAY CONTIENE**

Método for: definimos donde queremos iniciar: variable i = 0 (la primera posición del arreglo) y vamos a ir hasta la longitud y aumentamos de a uno el valor en cada vuelta. Entonces vamos a empezar a preguntarnos: vamos a estar recorriendo todo el arreglo y queremos saber si este arreglo, contiene algún elemento. Entonces, vamos a empezar a recorrer todos los elementos que tiene el arreglo y si alguno de ellos retorna un número x y retorna true, es porque lo encontramos.

Vamos a preguntarnos, si el arreglo en la posición i (nos permite obtener el valor que se encuentra en esa posición) o sea, si vale 0 estaríamos en el primer elemento, si vale 1 estaríamos en el segundo elemento y así sucesivamente. Entonces si el arreglo en su posición i (cualquiera que sea) es igual al elemento (si algún momento coinciden) entonces retornamos true.

Si ponemos un else dentro del bucle for lo que hace es que ese else se va a ejecutar la primera vez que no haya esa coincidencia. Por ejemplo, supongamos que la i vale 0, estaremos parados en la primera posición, y le estamos preguntando, ¿array de 0 es igual al elemento? En caso de que el primer elemento del array no sea el mismo elemento que estamos buscando, se va a leer el return false dentro del if, pero va a ser un resultado falso porque el elemento que estamos buscando, puede estar en otra posición más adelante. (Siempre va fuera del for)

**EJERCICIO 9: ARRAY OF NUMBS**

Es una función que recibe un arreglo de números. En este caso, debemos sumar todos los elementos que están en el arreglo y retornar el resultado de esa suma. Entonces le damos esa instrucción en la función para que haga el proceso que necesitamos y retorne la suma al final.

Toma un arreglo con cierta información y lleva ese proceso de ir elemento por elemento y devolver algo concreto. Algo concreto de ese resultado o de esa funcionalidad.

Vamos a guardar en una variable el resultado del reduce, es decir, ponemos el nombre de la función punto reduce. Y recordemos que se puede hacer de dos maneras: con una función de callback o agregando la función dentro del método reduce. La función recibe el acumulador y un valor referencia (numero). Entonces mi reduce va a ir elemento por elemento, va a ir guardando en el acumulador la suma de esa búsqueda de ir elemento por elemento. Es decir, tenemos el acumulador más cada número que estamos utilizando, y esta información es la que estaríamos retornando.

El método reduce necesita un valor inicial que se le pasa después de la función (después del bloque de código, después de la llave). Si no ponemos un valor inicial, va a inicial directamente con el 0. Los paréntesis del método reduce cierran cuando termina la función.

Al final, simplemente retornamos la variable.

¿Cómo identificamos cuando necesitamos un reduce? La idea es que empezamos a hacer como un recorrido y al final, toda la información que teníamos inicialmente en el arreglo se convierte en una sola cosa concreta, que es justamente como la filosofía que tiene el reduce. Va a estar recorriendo arreglos y va a estar retornando una sola cosa concreta (hace esa reducción en algo concreto)

**EJERCICIO 10: PROMEDIO RESULTADO TEST**. Tenemos un arreglo de números (notas), y lo que tenemos que retornar es un total, es decir, el promedio de esas notas.

El reduce se va a encargar de sumar todas las notas: Declaramos la variable (=) parámetro.reduce( recibe la función que tiene el acumulador y las notas). Retorna la suma de todas las notas (acumulador + notas); (Recorremos todo el arreglo “resultados test” y vamos sumando todas las notas que tiene el arreglo de notas. Y se retorna la suma y se divide por las notas (parámetro.longitud – para saber la cantidad de elementos que tiene el arreglo)

El método reduce es un bucle encapuchado.

En caso de querer hacerlo con un **bucle for**:

Definimos la variable suma que inicie en cero, recorremos el bucle (iniciamos la variable en cero, vamos hasta donde la función “resultados test” nos lo permita (es decir, su longitud) y vamos a estar sumando al acumulador. Necesitamos encontrar cada uno de los elementos y sumarlos a la variable suma, es decir, la variable suma que es mi contador va a ser igual al valor que yo ya tenga más el valor que vaya encontrando en resultados test, posición i. Entonces en cada par de valor o elementos que vaya encontrando voy a estar reemplazando la variable suma con los valores anteriores. (La variable suma comienza con el valor 0, se le suma el primer elemento del arreglo y ahora, la variable suma, va a tomar el valor del resultado de la variable suma más el primer elemento. Ese nuevo valor va a sumarse con el siguiente valor del arreglo y así sucesivamente). Cuando se cierra el ciclo, retornamos el valor de suma dividido el parámetro.length.

**EJERCICIO 11: NUMERO MAS GRANDE**

Este arreglo de números, tiene valores y tendríamos que devolver el más grande. Nosotros estamos a ciegas entonces podemos suponer cual es el más grande y si encontramos otro más grande, ese sería mi nuevo más grande. Vamos a empezar a recorrer los demás elementos del arreglo y si encontramos un mayor a este mayor, entonces ese será nuestro nuevo mayor.

Vamos a suponer que el número mayor es el que se encuentra en la primera posición (puede que sí o puede que no). Declaramos una variable (=) parámetro[valor asignado(0)]. Declaramos una variable que vamos a estar reasignando cada vez que encontremos un número nuevo.

Llamamos al arreglo de números.forEach (el forEach necesita una función que va a estar recorriendo cada elemento que tiene el arreglo (numero (referencia de que son valores numéricos)) Mi función ¿qué va a hacer?. Se va a encargar de comparar los elementos con el mayor. Vamos a preguntarle si el número es mayor o igual que mayor entonces vamos a hacer una reasignación, es decir, si cada valor que va a tener mi arreglo, si en algún momento cumple la condición de que es más grande que el mayor que hemos definido. Lo que yo quiero hacer es decirle que mayor ya no sea el primer valor, sino que sea igual al número.

**EJERCICIO 12: MULTIPLICAR ARGUMENTOS**

Recordemos “arguments”, nos hace referencia a la cantidad de elementos que recibe una función y ese resultado la devuelve como una especie de arreglo.

Entonces, arguments va a estar recibiendo argumentos que vamos a estar multiplicando y lo que queremos saber es el producto que se está devolviendo. Independientemente de la cantidad de argumentos que reciba, devolver la multiplicación de esos valores.

Vamos a multiplicar todos los valores que tiene el arreglo arguments: Hay un caso particular, que si no me llega nada (no me pasan argumentos) no tengo que hacer ninguna multiplicación.

Yo quiero saber si argumentos tiene elementos para multiplicarlos o no, lo hacemos a través de la longitud; entonces si arguments.length > 0, lo que queremos es multiplicar los valores que tiene ese arreglo argumentos; y si no se cumple la primera condición retornamos 0.

En caso del **bucle for**:

Iniciamos la variable desde cero, recorremos hasta arguments.length, y le sumamos i++. Luego, vamos a estar multiplicando, queremos saber el resultado de una multiplicación. Para eso, podemos crear una variable por fuera que se llame producto (=) 1; Entonces dentro del for, por cada elemento que vayamos encontrando nuestra variable producto va a ser igual a la multiplicación de arguments en la posición i. Entonces vamos a estar recorriendo todos los elementos que tiene el arreglo argumentos y lo multiplicamos por el producto anterior que ya teníamos definido, y eso, lo guardamos en la variable producto. Y retornamos el producto

**EJERCICIO 13: CUENTO ELEMENTOS**

Debemos retornar la cantidad de elementos del arreglo cuyo valor sea mayor a 18: Definimos una variable total que va a ser igual a 0. Vamos a recorrer el arreglo, y cada vez que encontremos un valor mayor a 18, entonces sumamos una unidad a ese total y al final, retornamos el total indicando cuantos cumplen la condición.

Método forEach, vamos a estar pasándole una función que va a recibir un número, que va a esta recorriendo cada número y por cada valor va a preguntar, si el número es mayor que 18. Si esta condición se cumple, el total que teníamos le sumemos una unidad. Y al final, si no cumple la condición no pasa nada, porque no se suma. Y al final de todo, retornamos el total.

**EJERCICIO 14: DIA DE LA SEMANA**

Recibe un número de día. Condicional if y le preguntamos si número de día es igual a 1 o es igual a 7 simplemente retorno “es fin de semana” y caso contrario, si no se cumple esa condición, retornamos “es día laboral”.

**EJERCICIO 15: EMPIEZA CON NUEVE**

Recibe un número, debe retornar true si el número entero inicia con 9, caso contrario, false.

Hacemos una conversión. Utilizamos el método **toString** para convertir el número en un string. Guardamos eso en una variable (var string). En string, guardamos el número que ya se convirtió en un string. A mí me interesa saber que el primer elemento del string sea un 9. Entonces vamos a preguntarnos, si string en la posición 0 es exactamente el ‘9’ retorno true. Si no, retorno false.

**EJERCICIO 16: TODOS IGUALES**

Queremos ver si todos los elementos del array son iguales. Podemos hacerlo con un for, que vaya preguntando elemento por elemento si son iguales.

Primero, vamos a guardar en una variable el valor que tiene array en la posición número 0 (es como una especie de codificador). Y lo otro, es empezar a recorrer el arreglo, empezar a recorrer los demás valores y ver si son iguales al primer valor o no.

Iniciamos con un for, nuestra variable ahora va a comenzar desde el 1 (porque me interesa comparar a partir de la primera posición, no de la 0, porque en la 0 ya se que tengo cierto valor; quiero empezar a separar desde la 0 en adelante). La condición sigue siendo la misma, hasta donde me permita la longitud del arreglo y sumando el acumulador. Vamos a preguntarnos, si array en la posición i es diferente a primer valor, significa que los valores que siguen no son iguales, por lo que se retorna false. Caso contrario, retorna true al final del for.

**EJERCICIO 17: MESES DEL AÑO**

Recorremos el arreglo, si en algún momento, encontramos enero, marzo o noviembre, lo guardamos en un arreglo nuevo.

Método forEach. Creamos un arreglo nuevo que sea un arreglo vacío. Vamos a recorrer el arreglo con un forEach, le damos una función que va a ir mes tras mes, y vamos a preguntarle, si el mes es exactamente en el que estoy parado es enero, o si el mes en el que estoy parado es marzo o si el mes en el que estoy parado es noviembre, entonces lo agrego al arreglo vacío que definí en la variable, y vamos a guardar todos aquellos que cumplan la condición. Es decir, llamamos a meses(arreglo).push y agregamos al mes. Entonces cada vez que cumpla esta condición, se estarían agregando al arreglo vacío.

¿Cómo sabemos que el arreglo meses tiene los 3 meses en particular? A través de length. Entonces si arreglo.length es igual a 3 significa que encontramos los 3 meses en particular. Y retornamos el arreglo, pero si esa condición no se cumple, entonces simplemente retornamos el mensaje “No se encontraron los meses pedidos”

**EJERCICIO 18: TABLA DEL SEIS**

Definir una variable números que tenga los números del 1 al 10. El map necesita un arreglo inicial (el que se creó). El map va a devolver un arreglo con la misma cantidad de elementos.

Vamos a recorrer el arreglo de números, y por cada número que tengamos, lo multiplicamos por 6.

Guardamos en una variable tabla el resultado de llamar al arreglo números con el método map. El método map va a recibir una función por callback, y le vamos a decir a esa función que va a estar parándose sobre todos los números. Y el .map va a ir número por número y lo va a multiplicar por 6, va a generar el resultado del 0, 1, 2 .. Cada vez que vea un número, va a retornar en esa posición la multiplicación de numero por 6. (La primera posición de tabla va a ser la multiplicación del número por 6, pasa al siguiente número, que en este caso sería 1, toma ese número y lo multiplica por 6). Retorna al final, el arreglo tabla.

Bucle **for**: arreglo vacío. Método for, variable i = 0, vamos a decirle que vaya hasta i < 11 y sumamos el acumulador. A tabla, le estaríamos pusheando el resultado de la multiplicación. Multiplicamos el valor de la i por 6. Y se van sumando al arreglo que tenemos. Luego, retornamos la variable tabla.

**EJERCICIO 19: MAYOR A 100**

Arreglo de números. Recorrer y retornar un arreglo con valores de 101 a 200.

Variable filtro. Hacemos un forEach, donde le pasamos la función en la que va a ir elemento por elemento (número). Si número es mayor estricto que 100 entonces lo pusheamos. Llamamos a la variable.push(numero). Y luego, retornamos la variable.

**MÓDULO 6:**

**EJERCICIO 1: CREAR GATO**

Debemos crear un nuevo objeto con las propiedades “nombre” y “edad”. Ambas propiedades deben tener el valor correspondiente recibido por parámetro. Además, agregar una propiedad con el nombre “meow”. La propiedad “meow” será una función que retorne el string “meow!”. Retornar el objeto.

Si queremos crear un objeto, guardamos en una variable la información; utilizamos var (con el nombre que queramos) (=) y abrimos llaves indicando que el contenido va a ser un objeto que estamos guardando en esa variable. En este caso nos hablan de la propiedad nombre cuyo valor (primero la clave : después el valor) va a ser el nombre que nos llega como parámetro de la función; y la edad, que es la otra clave que nos pedía va a tener por valor, la edad que le llega a la función. Además, nos pide una propiedad que a la vez sea una función y que se llamara “meow”. Entonces colocamos esa propiedad y como es una función, vamos a decirle de un lado tenemos una función (parece que no recibe parámetros, porque no nos dice nada), y lo único que haría esta función “meow” es retornar este string. Y lo último que haríamos es retornar el objeto es decir mi función retorna (el nombre de la variable que colocamos al principio)

**EJERCICIO 2: NUEVO USUARIO**

Crea un nuevo objeto. Debe tener las propiedades: nombre, email y password, con sus respectivos valores y retornar el objeto.

Guardamos un usuario (en una variable) y el usuario va a tener estas 3 propiedades: la propiedad nombre, con su respectivo nombre. La propiedad email con su respectivo email que venga de la función, y la propiedad password, con su respectivo pasword. Lo último que haríamos, sería retornar el objeto.

Recordar (los parámetros que tenemos dentro de la función) son variables. No sabemos que valor pueden tener, entonces los tomamos estrictamente como una variable, no le ponemos comillas ni nada, simplemente hacemos la simulación de la información que viene desde la variable.

**EJERCICIO 3: AGREGAR PROPIEDAD**

Recibe un objeto por parámetro. Debes agregarle una propiedad con el nombre recibido por parámetro. Esta propiedad será igual al valor “null”. Retornar el objeto.

¿Qué maneras teníamos de asignar una propiedad a un objeto? Había dos maneras: dot-notation y bracket-notation. Dot notation no nos sirve mucho cuando estamos utilizando una variable concreta o cuando estamos tratando de almacenar un variable (va a haber problemas si la información que viene es a partir de una variable que ya está definida. Para eso, es mucha mejor solución la notación del bracket. La notación del punto es más literal, si yo quiero guardar una profesión, va a tomar exactamente el valor “profesión” y no el contenido que tenga la variable. Cuando tengamos que hacer uso de las variables, la notación por punto no nos sirve mucho porque es muy literal con la información. Entonces, lo hacemos con la notación de corchetes: objeto[propiedad], y retornamos el objeto.

Notación de corchetes: cuando el objeto sea una variable (cuando no puedo decir explícitamente que se llama “propiedad”) y notación por punto cuando es explícito el llamado de lo que estamos haciendo, de lo que estamos asignando o simplemente de lo que estamos tratando de modificar.

**EJERCICIO 4: INVOCAR MÉTODO**

El parámetro “método” es un string que coincide con el nombre de una propiedad del objeto recibido. Esta propiedad contiene una función en su interior. Debes invocarla/ejecutarla. No es necesario retornar nada. Sólo nos encargamos de hacer el llamado a esa función

Los métodos son valores o propiedades que cuyo valor es una función.

Seleccionamos el objeto para acceder a él y debemos acceder a él por parámetro a través de notación. En este caso, estaríamos accediendo al valor que tiene la propiedad, que es una función. Si queremos invocar la función, simplemente agregamos paréntesis.

\* Supongamos que objeto es esto: // objeto = {método: Function }

Un objeto que tiene una propiedad método, que tiene una función por dentro. Si yo quiero acceder a esa función, entonces, llamamos primero al objeto y luego, busco al método dentro del objeto, es decir, estoy accediendo al valor que tiene esa clave. Para invocar la función solamente ponemos los paréntesis para que esta función se llame, independientemente de lo que tenga esa función por dentro.

**EJERCICIO 5: MULTIPLICAR NÚMERO DESCONOCIDO POR CINCO**

El parámetro “objeto misterioso” posee una propiedad con el nombre “numero misterioso”. Debes multiplicar este número por 5 y retornar el resultado.

¿Cómo accedo al valor que tiene la propiedad “numero misterioso”? Primero llamo al objeto y luego accedo al numero misterioso a través de notación porque está literalmente tiene algo que se llama de esa manera. Accedemos al valor que tiene la propiedad y la multiplicamos por 5. Luego retornamos el resultado.

**EJERCICIO 6: ELIMINAR PROPIEDAD**

El parámetro “propiedad” es una propiedad del objeto que recibes. Debes eliminarla del objeto y retornarlo finalmente.

Agregamos la palabra clave para eliminar objetos. Queremos eliminar la propiedad, es una variable del objeto. Simplemente, accedemos al objeto y a la propiedad. Lo hacemos mediante notación y agregamos la propiedad. Finalmente, retornamos, el objeto.

**EJERCICIO 7: TIENE EMAIL**

Verifica si el “objeto usuario”, en su propiedad “email”, posee un valor definido. Es ese caso retornar true. Caso contrario, false.

Dado que tengo el objeto, y dado que tiene esa propiedad, verifica si el valor está definido para esa propiedad. No tenemos que verificar que la propiedad email exista, sino que nos están garantizando que existe, pero queremos ver que sucede con el valor que tiene definida esa propiedad. Ya sabemos que hay una propiedad literalmente llamada email.

Queremos ver si la propiedad tiene un valor definido, o si no hay un valor definido o si no existe la propiedad.

Preguntamos lo siguiente: si objeto usuario punto email retorna true. Si no, retorna false.

**EJERCICIO 8: TIENE PROPIEDAD**

Recibe un objeto y una propiedad. Verifica si el objeto recibido posee una propiedad con el mismo nombre que el parámetro “propiedad”. En ese caso retornar true. Caso contrario, false.

El método has own property lo que hace es averiguar, decirnos true o false, si un objeto tiene esa propiedad.

Si, llamamos al objeto y preguntamos si tiene esa propiedad, y es un método que tiene una función (), se va a ejecutar. Y el valor que le pasamos a la función para que se ejecute es la propiedad que queremos averiguar si existe o no dentro del objeto. Y esa variable en este caso, entonces, como es una variable utilizamos directamente, la palabra propiedad. Si esto es verdad, entonces retornamos true. En caso contrario, retornamos false.

**EJERCICIO 9: VERIFICAR PASSWORD**

Verifica si la propiedad “password” del “objeto usuario” coincide con el parámetro “password”. En ese caso retornar true. Caso contrario, false.

Recibe un objeto usuario y una password. El objeto ya tiene determinada una propiedad que se llama literalmente password, y lo que queremos ver es que si el valor que tiene esa propiedad coincide con la password que le llega a la función.

Si, tomamos el objeto usuario, notación, llamamos a la propiedad (así accedemos al valor que tiene la propiedad password). Lo que queremos verificar es si coincide con la password que tenemos como propiedad. Entonces, tenemos que ver si es igual a la propiedad que le está llegando por parámetro. Retornamos true, en caso contrario, retornamos false.

**EJERCICIO 10: ACTUALIZAR PASSWORD**

Reemplaza la contraseña guardada en la propiedad “password” del “objeto usuario”. La nueva contraseña la recibes por parámetro. Retornar el objeto.

Recibe un objeto usuario y una nueva password.

Tenemos que hacer una reasignación de una propiedad que está definida. Queremos modificar el valor que tiene la propiedad.

Tenemos que acceder al objeto, notación, password. (acá estamos teniendo un valor), le asignamos una nueva password. Luego, retornamos el objeto usuario.

**EJERCICIO 11: AGREGAR AMIGO**

Recibe un objeto usuario y un nuevo amigo. El parámetro “objeto usuario” tiene una propiedad llamada “amigos” igual a un arreglo. Deber agregar el “nuevo amigo” al final de este arreglo. Retornar el objeto.

Tenemos un objeto usuario, que es un objeto, que tiene una propiedad “amigos”, que tiene un arreglo //objetoUsuario = {amigos: []} Esta es la información que tenemos, y lo que queremos es hacer que este nuevo amigo se agregue al final del arreglo.

Llamamos al objeto, y accedemos a la propiedad “amigos” que tiene el objeto. Lo que tenemos guardado es un arreglo, como es un arreglo, podemos hacer uso de todos los métodos que tiene un arreglo. Al método push le pasamos el “nuevo amigo”. Luego, retornamos el objeto.

**EJERCICIO 12: PASAR USUARIO A PREMIUM**

Recibe un objeto muchos usuarios. El parámetro “objeto muchos usuarios” es un arreglo de objetos (usuarios). Cada usuario tiene una propiedad llamada “es premium”. Define esta propiedad de todos los usuarios como true. Retornar el arreglo

Tenemos un objeto muchos usuarios, nos dicen que esto es un objeto de arreglos.

//objetoMuchosUsuarios = [ {es premium: true}, {es premium: true}, {}, {}]

// usuario usuario

Cada usuario, o sea, cada uno de los objetos que tenemos definidos tiene una propiedad que se llama “es premium” y que tiene un valor. Vamos a definir esa propiedad de todos los usuarios como true.

Inicialmente, tenemos un arreglo, así que vamos a utilizar los métodos de un arreglo. Me interesa recorrer cada uno de los objetos que tiene un arreglo, uno por uno, ver la propiedad, es decir, meterme dentro del objeto y cambiar el valor que tenga, por true.

Llamamos al arreglo punto for each, que recibe una función y un parámetro que va a representar a los usuarios (simplemente es una identificación de cada elemento que tiene el arreglo). Y lo que queremos hacer es por cada usuario, vamos a la propiedad “es premium” y le asignamos el valor de true. Tenemos que hacer una asignación de esa propiedad. Entonces tomamos el usuario que va a estar representando cada objeto del arreglo notación es premium y le asignamos el valor de true. Y al final, retornamos el arreglo inicial.

**EJERCICIO 13: SUMAR LIKES DE USUARIOS**

Recibe un objeto usuario. El parámetro “objeto usuario” tiene una propiedad llamada “posts” que es un arreglo. Este arreglo contiene objetos (post). Cada post posee una propiedad llamada “likes”. Esta propiedad es un número. Debes sumar los likes de todos los posts y retornar el resultado.

//objetoUsuario = {posts: [{likes: #}, {likes: #}]}

// post post

Mi objeto es un objeto usuario que tiene una propiedad llamada posts y, lo que tenemos es un arreglo de objetos. Cada objeto que tenemos, es un post, y cada uno de esos objetos post, tiene una propiedad que se llama likes, y esa propiedad, va a tener un número. Lo que queremos hacer es recorrer este arreglo y contar cada uno de los likes que tiene cada post.

Declaramos una variable que puede iniciar con un valor concreto (0). Vamos a empezar a recorrer el arreglo donde están los posts, nos metemos a cada objeto y contamos la cantidad de valores o la cantidad numérica que tiene la propiedad likes. Primero accedemos al objeto usuario, estamos metidos en el objeto. Accedemos a la propiedad que se llama post (con notación). Una vez que estamos acá, y que esto es un arreglo, podemos utilizar cualquier método de los arreglos para recorrer cada objeto y obtener la información que tienen los likes. Por ejemplo, un for each, que va a recibir una función que va a estar moviéndose a través de cada post. (nombre x). La idea con cada post (nombre x), es acceder a la propiedad likes que tenemos para obtener el valor (la información que se está guardando en esa propiedad). Y lo que hacemos con esto, es decirle a nuestra variable inicial es que su valor va a ser igual a la acumulación, más lo que vayamos encontrando (total likes = total likes + post.likes)

Lo hacemos de esta manera porque esta variable (total likes) va a ir guardando lo que vaya encontrando en el primer post, lo guardamos, luego pasaría al siguiente post, busca la información, la acumula con lo que ya está guardado y la guarda en la nueva variable. Luego, retornamos la variable inicial.

**EJERCICIO 14: AGREGAR MÉTODO CALCULO DESCUENTO**

Recibe un “objeto producto”. Agrega una propiedad al “objeto producto” con el nombre “calcular precio descuento”. Esta propiedad debe ser una función que multiplique el precio del producto por el “porcentaje de descuento”. El “objeto producto” posee una propiedad “precio” y una propiedad “porcentaje de descuento”. Luego debes restar del precio total del producto ese valor de descuento. Retornar el precio final. Ejemplo: Precio 10, porcentaje de descuento 0.2, precio final 8.

//objetoProducto = {precio: $xxx, porcentajeDeDescuento: xxx%, calcularPrecioDescuento: función ()}

Tenemos que crear el método. Vamos a llamar al objeto, notación, (agregamos el nombre del método que queremos crear) calcular precio y estamos guardando una función, la cual no posee ningún parámetro. Y lo que queremos es que la función nos calcule el descuento utilizando el precio que tiene el producto. Vamos a decirle a la función que retorne el resultado del descuento, va a tomar el precio, y a ese precio, le va a restar la multiplicación del precio por el descuento. ¿Cómo calculamos un descuento? Al precio original le restamos su precio por el descuento que se está dando.

Si yo quisiera acceder al precio, independientemente del valor que tenga, podemos hacerlo, utilizando el this (va a hacer alusión al objeto desde el cual se haga el llamado a esa palabra, va a estar referenciando al objeto “objeto producto”, de manera que el this, se va a comportar como si fuera realmente el objeto y poder acceder a cualquiera de las propiedades que tenga definida el objeto producto.

Retornamos el objeto

**MÓDULO 7: EJERCICIO 1**

**EJERCICIO 1: CREAR USUARIO**

Crea una clase de ES6 o una función constructora llamada “Usuario”. El constructor de esta clase debe tener las propiedades: “usuario”, “nombre”, “email” y “password”. El valor de cada propiedad la recibirá por parámetro. Además, esta clase debe tener un método llamado “saluda”. Este método debe retornar el string: “Hola, mi nombre es: {{nombre}}”. El valor {{nombre}} debe ser el nombre definido para la instancia de la clase. Retornar la clase.

Función constructora: Creamos una clase o una función constructora, tienen la misma funcionalidad que es crear ese molde. Vamos a tener la plantilla que nos permita instanciar objetos, cuyos objetos van a tener valores o ciertas propiedades en común. Las clases nos permiten tener una plantilla de la cual saquemos varios objetos que tengan ciertas propiedades en común.

Creamos la función constructora que nos dicen que se llama usuario y le vamos a pasar las propiedades que nos pasan (usuario, nombre, email, password). Y con esta información, vamos a empezar a crear las propiedades de las instancias. Vamos a decirle a nuestra instancia, que la representamos con el this, lo que hace es representarnos el objeto que tiene la clase. Entonces, tendríamos un usuario, igual al valor que nosotros le pasemos a la instancia; un nombre, igual al nombre que recibe; un email igual al email que recibe y una password igual a la información que está recibiendo por ese parámetro.

Tenemos que crear un método llamado “saludar”. Los métodos de una clase son funciones, es una propiedad que contiene una función por dentro como valor. Nuestra instancia, va a tener una propiedad “saludar” que realmente es un método porque contiene una función () y esa función retorna un string. Pero tiene una condición, el nombre es aquel que le estemos pasando a nuestra instancia. (Se concatena el string con la instancia de nombre). Luego, retornamos la clase.

Clase: Utilizamos la palabra class, llamamos al usuario, llamamos al constructor que va a recibir los mismos parámetros. Llamamos a cada una de las propiedades con sus valores.

**EJERCICIO 2: AGREGAR MÉTODO PROTOTYPE**

Recibe “usuario”. Agrega un método al prototipo de “Usuario”. El método debe llamarse “saludar” y debe devolver el string “Hello world!”

Lo queremos agregar a través del prototype. Primero, escribimos la clase punto el prototipo punto el método que queremos agregar, esto lo igualamos a una función que no recibe parámetros y retorna el string.

**EJERCICIO 3: AGREGAR STRING INVERTIDA**

Agrega un método al prototipo de “String” que devuelva la misma cadena de caracteres, pero invertida. El método debe llamarse “reverse”. Necesitarás utilizar el objeto “this”.

String es la clase. Y esta clase, es la que tiene todos los Strings.

Vamos a crear un método. Queremos que los Strings tengan un método que les permita hacer como un reverso de la palabra. Queremos crear un método que reciba un string, que se ejecute y nos devuelva el string que teníamos inicialmente pero reversado.

Llamamos a la clase, al prototipo y al método reverse que va a ser una función (=). La función va a tomar el string (this, va a estar representando cualquier string que venga de la clase), y aplicamos el método para convertir el string en un arreglo, y luego reversamos. Seguido, el método join para unir toda la información en un string. El método, va a retornar, esa conversión.

Bucle for: Definimos una variable y le asignamos un string vacío. Vamos a recorrer el string usando un for, vamos a iterar y lo vamos a hacer al revés, de manera que cada letra que encontremos, la vamos guardando en la variable reversed. Y al final, solo retornamos reversed que nos traería todas las letras reversadas.

Empezamos con un for (declaramos la variable, pero empezamos desde arriba). El índice, va a ser igual a la longitud del this (es el que reemplaza el string).length, al que tendríamos que restarle una unidad (la longitud es una unidad más grande que el índice del arreglo o del string), y vamos a parar cuando el valor de i valga 0 (va a para cuando ya estemos en la primera posición del string), y lo que hacemos con el acumulador (que ya no va a ser acumulador), va a ir descendiendo. Va a ir bajando un número, el índice que tiene el for. Por cada letra que se vaya encontrando (reversed) lo vamos a concatenar con cada uno de los caracteres que tiene el string this (reversed + this[i]).

Empieza desde la última letra y va bajando hasta llegar a la primera, en la posición 0. Por cada posición vamos a estar concatenando a lo que ya tengamos, el valor que tenga el string en ese valor.

**EJERCICIO 2:**

**EJERCICIO 1:** **PERSONA**

Crea el constructor de la clase “Persona”. Debe tener las propiedades “nombre”, “apellido”, “edad” y “domicilio”. Debe tener un método llamado “detalle” que nos devuelva un objeto con las propiedades de la persona y sus valores.

Vamos a crear el constructor, porque en este caso, es una clase. El constructor recibe el nombre, el apellido, la edad y el domicilio. Los valores van a ser los valores que reciba a partir de las propiedades. Nombre, apellido, edad, domicilio.

Vamos a tener un método llamado “detalle”, que tiene una función, esta función devuelve/ va a retornar un objeto con las propiedades de la persona y sus valores. ¿Cómo retornamos un objeto? Con llaves. Y ese objeto, va a tener la particularidad, que el nombre del objeto corresponde al nombre que recibe el constructor, el apellido igual, la edad (hacemos referencia a las mismas características que definimos). El nombre, es lo que tenga guardado la variable this.nombre, apellido, edad, domicilio.

**EJERCICIO 2: CREAR INSTANCIA PERSONA**

Recibe “nombre, apellido, edad, domicilio”. En este ejercicio debes crear una instancia de la clase construida en el ejercicio anterior. Recibirás las propiedades por parámetro. Retornar la instancia creada.

Creamos una variable. Si queremos que venga de esa clase, utilizamos la palabra new indicando que es una instancia nueva de la clase Persona () con los parámetros dentro. Y finalmente, retornamos esa instancia, lo que estamos guardando en esa variable.

**EJERCICIO 3: AGREGAR MÉTODO**

La función agrega un método “datos” a la clase “Persona”. Este método toma la propiedad “nombre” y “edad”, y devuelve el string: Ejemplo: “Juan, 22 años”.

Como la clase Persona, ya está creada, utilizamos el prototype para crear un nuevo método. Que es una función. La función va a tener presente el nombre de la persona y la edad. Para retornar nombre concatenada con una coma, concatenada con la edad, y concatenada con la palabra años.

**MÓDULO 8:**

**EJERCICIO 1: MAYÚSCULA**

Recibe “nombre”. Esta función recibe un nombre (string). Debe devolver el mismo nombre, pero con la primera letra en mayúscula. Ejemplo: “mario” = “Mario”.

Hay un método que se llama **toUpperCase()**, este método nos permite convertir en mayúsculas las letras de un string. Convierte todas las letras en mayúsculas.

El método **slice()**, va a recibir dos valores. El primer lugar me dice desde donde va a ser el recorte y el segundo, hasta donde va a ser. En caso de no pasarle ningún valor, va a recortar desde el primer valor hasta el final. Por ejemplo: “hola”.slice(1) = “ola”

Entonces, tendríamos “nombre”. Tomamos el primer valor del primer índice [0], y le aplicamos el toUpperCase y lo demás es concatenarle el nombre con el recorte y le decimos que la haga desde la posición número 1. Y luego, se retorna esa concatenación.

**EJERCICIO 2: INVOCAR CALLBACK**

Recibe “cb”. Invoca/ejecuta el callback “cb”. No debes retornar nada

El cb representa una función.

Función y sus paréntesis. Como para ejecutar la función.

**EJERCICIO 3: OPERACIÓN MATEMÁTICA**

Recibe num1, num2, y cb. En este ejercicio recibirás dos números y un callback. El callback realiza una operación matemática, por lo que necesita de los dos números. Retorna el resultado del callback pasándole como parámetros los números.

La función va a estar retornando el llamado del callback con los dos valores que le llega a la función.

**EJERCICIO 4: SUMAR ARRAY**

Recibe un array de números y un cb. Recibes un arreglo de números y un callback. Suma todos los números del arreglo. Este resultado debes pasárselo como argumento al callback recibido. No debes retornar nada.

Debemos sumar todos los valores del array, y cuando tengamos el total, se lo pasamos al callback. Declaramos la variable = 0. Tomamos el arreglo y utilizamos el forEach, que va a ir elemento por elemento y vamos a estar sumándolo con los valores que tiene la variable. Le pasamos una función que contenga números, y lo que hacemos con cada número, es simplemente, a la variable sumarle ese número. Luego, al callback se lo ejecuta con la variable.

**EJERCICIO 5: FOR EACH**

Recibe un arreglo y un callback. Debes iterar sobre el arreglo, y por cada elemento ejecutar un callback. Debes pasarle el elemento como argumento al callback.

Vamos a estar recorriendo el arreglo, y por cada elemento que tenga el arreglo, vamos a estar llamando al callback, se lo vamos a estar pasando directamente para que el callback lo ejecute.

Declaramos el arreglo, con el bucle forEach que recibe una función, que se va a aplicar a cada elemento del arreglo. Y va a tomar cada elemento del arreglo y se lo va a pasar al callback, es decir, llamamos al callback con cada elemento.

**EJERCICIO 6: MAP**

Recibe un array y un callback. Debes iterar sobre el arreglo, y cada elemento pasárselo como argumento al callback. Tiene que guardar el resultado devuelto por el callback en cada elemento dentro de un nuevo arreglo. Retorna el nuevo arreglo.

Vamos a declarar una variable. Y vamos a llamar al map. Le pasamos una función, que va a tomar cada elemento, y el callback recibe cada elemento. Retorna en un arreglo nuevo el callback con el elemento como parámetro, con la misma longitud que el arreglo original.

Retorna la variable en la que guardamos los elementos.

**EJERCICIO 7: FILTER**

Recibe un array de Strings. Debes identificar todos los elementos del arreglo que comiencen con la letra “a”. Luego retorna un nuevo arreglo con estos elementos.

El arreglo va a tener Strings y vamos a retornar un arreglo nuevo que contenga todos aquellos elementos que comiencen con la letra “a”.

Definimos un arreglo vacío. Hacemos un for normal, comenzamos desde la posición 0 hasta la longitud que tenga el arreglo de los Strings y aumentamos el valor del iterador. Vamos a preguntarle, vamos a ir elemento por elemento del arreglo, y si alguno de estos en la primera posición tiene la letra “a” entonces los pusheamos al arreglo vacío. Si el arreglo en su posición i y a la posición 0, si es igual a la letra “a”, entonces el arreglo, la va a guardar mediante el push (arreglo en su posición i). Retornamos el arreglo vacío.

**MÓDULO 9:**

**EJERCICIO 1: DE OBJETO A ARRAY**

Recibe un objeto. Tendrás que crear un arreglo de arreglos. Cada elemento del arreglo padre será un nuevo arreglo que contendrá dos elementos. Estos elementos debe ser cada par clave:valor del objeto recibido. Ejemplo: {D: 1, B: 2, C: 3} --> [[‘D’, 1], [‘B’, 2], [‘C’, 3]]

Supongamos que nos dan el ejemplo de arreglo, tenemos la D como propiedad y el 1 como valor. La B como propiedad y el 2 como valor, y así sucesivamente. Tenemos que retornar un arreglo que contenga un arreglo con la D y con el 1, es decir, la clave con su valor; otro arreglo que tenga la B con el 2, la clave con su valor.

Cambiamos un objeto con propiedades a un arreglo con arreglos por dentro, y cada uno de los arreglos va a tener dos elementos, la clave y su valor.

Creamos un arreglo vacío, que es el arreglo, que va a estar conteniendo todos los otros arreglos. Nos interesa conocer las propiedades o claves que tenga cada objeto y a partir de ahí, encontrar su valor (el de cada clave).

El for in va a ir clave por clave a partir de todas las claves que tiene el objeto (entonces lo escribimos clave en el objeto “objeto”, que es el nombre que recibimos) y por cada una de esas claves o propiedad, queremos crear un arreglo nuevo en el cual se agregue tanto la clave como el valor que tiene guardada esa clave. Creamos una variable con el nombre x, y va a contener dos elementos, el primero es justamente la clave, o sea, cada una de estas propiedades que tendría el arreglo; y el segundo elemento que contendría el arreglo, sería el valor que tiene la clave. ¿Cómo accedo al valor que tiene esa clave? Primero, llamo al objeto y luego notación corchete porque la clave es una variable, y la llamamos dentro de corchete. Con esto vamos a estar recorriendo cada una de las claves o cada una de las propiedades que tiene el objeto y por cada una de ellas, vamos a crear un arreglo que contenga esa propiedad o esa clave en particular y un segundo elemento, que es el valor que tiene la clave en particular, en este caso, objeto de clave. Luego vamos a pushear al arreglo vacío y dentro () va a ir el nombre de la variable. Al final, retornamos el arreglo vacío.

**Si queremos recorrer un objeto, tenemos el for in. Lo que ponemos como clave es representativo, está representando cada una de las propiedades que tiene el objeto. Y objeto simplemente es el nombre del objeto que nos están pasando como parámetro de la función.**

**For in solamente para los objetos.**

**EJERCICIO 2: NUMBER OF CHARACTERS**

La función recibe un string. Debes recorrerlo y retornar un objeto donde cada propiedad es una de las letras del string, y su valor es la cantidad de veces que se repite en el string. Las letras deben estar en orden alfabético.

Tenemos que devolver un arreglo donde cada letra es una propiedad del objeto y su valor, es la cantidad de veces que aparece la letra en el string.

Vamos a crear un objeto y sea un objeto vacío. Vamos a recorrer el string, vamos a identificar las letras para agregarlas al objeto que queremos devolver. Lo primero, es crear el lugar donde vamos a guardar la información, lo otro es, por cada letra que vayamos teniendo, vamos a mirar como las agregamos al objeto. Empezamos a recorrer el string y empezamos a ver cuantas veces se está repitiendo para ver esos casos particulares. Vamos a crear un for que inicie desde 0, hasta su longitud y nos vamos a ir moviendo de uno en uno. Supongamos que primero encontramos una letra “a”, nuestro objeto está vacío, y a nosotros nos interesa guardar esas letras para poder seguir haciendo el conteo. Mi letra “a” va a ser mi primer elemento, el del índice 0. Vamos a agregar el índice al arreglo como una propiedad, simplemente estamos guardando el valor de la letra a dentro del objeto como una propiedad. Vamos a ir guardando, y cada vez que se repita una, no la agrego, sino que le agrego un valor a la unidad porque ya existe.

¿Cómo sé si un objeto ya tiene una propiedad definida? **hasOwnProperty**. Vamos a preguntarle al objeto si ya tiene esa letra en particular, si no la tiene, entonces la agregamos al objeto, pero si sí la tiene, entonces lo que hacemos es sumarle una unidad al valor que tiene la propiedad.

Creamos otra variable, que nos va a estar guardando cada uno de los caracteres que tiene el string [i]. Y acá vamos a llamar al objeto y le preguntamos si este, tiene esa variable ya dentro. Si la variable ya es parte de sus propiedades.

Si la letra ya está, si esa letra del string ya es parte del objeto, entonces no la voy a agregar, simplemente voy a sumarle una unidad al valor que tiene la letra dentro del objeto. Voy a acceder al valor que tiene la propiedad, o sea, el objeto corchetes variable, y al valor que tenga la variable le sumamos 1, indicando que ya estaba dentro y solamente le sumamos al contador. Pero si la letra no está, la agregamos al objeto. Llamamos al objeto, llamamos a la propiedad y lo igualamos a 1 porque encontramos el primero. Y al final, retornamos el objeto.

**Un objeto no está ordenado. No tiene posiciones**

**EJERCICIO 3: CAP TO FRONT**

Recibe un string con algunas letras en mayúscula y otras en minúscula. Debes enviar todas las letras en mayúscula al comienzo del string. Retornar el string.

Tenemos que tener en cuenta el orden en particular. Creamos una variable para letras mayúsculas que va a ser un string vacío y otro, para letras minúsculas que también va a ser un string vacío. Y lo que nosotros queremos hacer eso retornar las mayúsculas antes que las minúsculas. Es decir, vamos a estar recorriendo el string y por cada mayúscula que veamos, la ponemos en el arreglo vacío, lo mismo, con las minúsculas. Y al final, lo que va a retornar mi función es la concatenación de las variables.

Vamos a hacer un for, el cual vamos a iniciar desde 0, hasta la longitud del string y vamos a ir aumentando de uno en uno. Vamos a preguntarnos por las posiciones, es decir, si el string en la posición i, si lo que está en ese carácter es exactamente igual a string en su posición i.toUpperCase (si yo me paro en una letra en mayúsculas entonces lo agregamos a la variable correspondiente a las mayúsculas. Es decir, la variable mayúscula, con lo que sea que tenga, va a estar sumando el nuevo valor de la posición i (Mayus = Mayus + string[i]). El caso contrario, es que si una letra no cumple esa condición es que es una letra minúscula, entonces va en la variable minus y se la concatena con el string i (el igual que en Mayus).

**EJERCICIO 4: AS A MIRROR**

Recibe una frase. Tu tares es retornar un nuevo string en el que el orden de las palabras sea el mismo. La diferencia es que cada palabra estará escrita al inverso.

Creamos una función que se encargue de hacer esa inversión que necesito hacer. Esta función va a recibir un string. Y la función, lo que va a hacer con el string es simplemente reversar. Vamos a retornar el string que nos llega, le aplicamos el split, reversamos y luego, juntamos la información sin espacios. Esta función la vamos a utilizar como recurso para hacer esa reversa a cada elemento, a cada palabra que tiene el string. Entonces estaríamos aplicando la función a cada una de las palabras que tiene el string. ¿Cómo ejecuto una función para cada palabra que tiene el string? Creamos una variable y vamos a tomar la frase original que tiene la función y la vamos a esplitear. (Lo que nos interesa es que cuando se haga esa separación, la separe con el espacio). Ahora qué vamos a hacer con ese arreglo, dado que ya lo tengo convertido en un arreglo, vamos a llamar a la función. Creamos otra variable a la que le vamos a pasar una función al arreglo anterior, de manera que me devuelva otro arreglo, pero, ya con las palabras invertidas. Esta variable, la igualamos al arreglo que creamos en la variable anterior, le aplicamos el método map, y recibe una función, va a ir palabra por palabra. Y por cada palabra, vamos a llamar al método de la primera función que creamos y dentro vamos a escribir palabra. Y luego retornamos esta función para que quede reversada en un arreglo. Luego de esto, tomamos el último arreglo (de la última variable) y lo unimos en un único string, y esto es lo que tiene que retornar la función.

**LOS ARREGLOS TIENEN EL MÉTODO REVERSE, NO LOS STRINGS**

**EJERCICIO 5: CAPICUA**

Recibe un número. Si el número que recibes es capicúa debes retornar el string: “Es capicúa”. Caso contrario: “No es capicúa”.

Vamos a transformar el número en un string. Creamos una variable; convertimos el número que nos llega y le pasamos el método toString ejecutado. Luego, declaramos el reverso de ese string. Creamos otra variable, que va a tomar el string número (primera variable), lo espliteamos, lo reversamos y lo unimos. Luego, lo comparamos; si lo que tengo en la primera variable es exactamente igual a lo que tengo en la segunda variable, entonces retornamos “Es capicúa”, en caso de no ser “No es capicúa”.

**EJERCICIO 6: DELETE ABC**

Recibe un string. Tu tarea es eliminar las letras “a”, “b” y “c” del string recibido. Retorna el string sin estas letras.

Vamos a crear una variable que sea un string vacío, por cada vez que encontremos la letra “a”, “b” o “c” no las agregamos, agregamos todas las demás al string vacío y retornamos esa variable al final.

Luego del string vacío, vamos a recorrer el string original, vamos a ir letra por letra del string. Vamos a preguntarle, si el string en su posición i es diferente de la a, b y c la vamos a guardar en el string vacío concatenándola con el string i. Y finalmente, retorna el string vacío.

**EJERCICIO 7: SORT ARRAY**

Recibe un arreglo de Strings. Debe retornar un nuevo arreglo, pero con las palabras ordenadas en orden creciente a partir de la longitud de cada string.

Vamos a empezar a comparar longitudes. Necesitamos un for que inicie desde 0, que vaya uno antes de la longitud del parámetro que recibimos y luego aumentamos. Luego, vamos a definir otro for con una variable j, vamos a iniciar una posición después de la i, independiente de cual sea el valor de i le sumamos 1 unidad. (Siempre una posición adelante del iterador i). j va a ir hasta la longitud del parámetro.

Vamos a empezar a preguntarnos, si la longitud de el parámetro en la posición i es mayor a la longitud del parámetro en j, entonces vamos a decirle que lo que esté en la posición i, sea lo que tiene la posición j (asignamos ese valor). Antes de esto, escribimos una variable auxiliar que me permite conservar cierta información, le decimos que guarde la información que está en el arreglo en la posición i. Y si queremos hacer el cambio, cambiamos el orden, el arreglo en su posición j, su nuevo valor, va a ser el que tenga la variable auxiliar.

Luego, retornamos el arreglo.

**EJERCICIO 8: BUSCO INTERSECCIÓN**

Recibe dos arreglos de números. Debes retornar un nuevo arreglo en el que se guarden los elementos en común entre ambos arreglos. Si no tienen elementos en común, retornar un arreglo vacío. Los arreglos no necesariamente tienen la misma longitud.

Definimos una variable que va a ser un arreglo vacío. De manera que cada vez que encontremos dos valores iguales, lo pusheamos al arreglo. Creamos un for que arranque en 0, y se mueva en la longitud, y se acumule. Creamos otro for que arranque desde la posición 0, se mueva en su longitud y se acumule. Si el array 1 en su posición 1 es exactamente igual al array 2 en su posición j, significa que encontramos dos números igual, y lo que hacemos es pushear a nuestro arreglo vacío, el del array que queramos en su posición. Luego, retornamos el arreglo vacío.