Explicar los siguientes Conceptos:

1.1 Variables
1.2 Strings
1.3 Funciones (argumentos, return)
1.4 Declaraciones if
1.5 Valores booleanos (true, false)

Explicaciones

1.1 Variables.- Son espacios que se reservan y que le nombramos con identificadores para luego asignarles, una o varias veces, valores diferentes.

1.2 Strings.- Son valores que también se les conoce como cadenas de texto, a estos String se los debe encerrar entre comillas simples o dobles. Ej. 'Calle #18'

1.3 Funciones.- Creamos nuestras funciones para que agrupen líneas de código y procesen u operen con los valores de los argumentos que se les pasa al momento de llamarlas por su nombre. También es posible crearlas hasta para que devuelvan tras su finalización un determinado Valor.

1.4 Declaraciones if.- Se usa el if acompañado de una condición(decisión) para que realice algo o no o continue.

1.5 Valores booleanos (true, false).- Los valores booleanos hacen o ayudan si llevan o bien un valor true(verdadero) o un valor false(falso).

Explicar los siguientes Conceptos:

3.1a for
3.1b &&, ||, !

Explicaciones:

3.1a for.- La declaración for permite hacer repeticiones de las intrucciones o líneas de códigos que se escriban dentro de sus llaves, estas repeticiones se terminan cuando la condición da respuesta false.

3.1b &&, ||, !.- Estos símbolos trabajan con valores tipo boolean (true y false), es decir, los dos primeros símbolo entre valores booleanos y devuelve una respuesta booleana.
Ejemplo: true && false su respuesta es false;

Y el tercer símbolo un valor booleano a su derecha y asimismo devuelve otro valor booleano.
Ejemplo: !true y devuelve false.

Las respuestas, de los dos primeros símbolos se rigen de acuerdo a lo siguiente:
Supongamos que tengo del lado izquierdo del símbolo una expresión que le llamaremos P;
Y en el lado derecho una expresión que le llamaremos Q, entonces:

	P		Q	Respuesta	
Si	true	&&	true	true	
Si	true	&&	false	false	
Si	false	&&	true	false	
Si	false	&&	false	false	Observación: Solo es true cuando ambos son true

```
     P       ||       Q      Respuesta
Si  true     ||      true      true
Si  true     ||      false     true
Si  false    ||      true      true
Si  false    ||      false     false
```

Observación: Basta que uno de los operandos sea true para que la respuesta de true.
Importante: Solo da como respuesta false cuando ambos son false.

```
         P         !P          Respuesta
Si     true        !true         false
Si     false       !false        true
```

Observación: La negación del valor booleano da como respuesta el otro booleano.

```
         P         !!P          Respuesta
Si     true        !!true         true
Si     false       !!false        false
```

Observación: La negación de la negación del valor booleano da como respuesta el mismo valor.

En un archivo de texto separado que debes crear, escribe explicaciones de los siguientes conceptos
como si se lo estuvieras explicando a un niño de 12 años. Hacer esto te ayudará a descubrir rápidamente
cualquier agujero en tu comprensión.

Arrays.

Explicación:
Los Arrays.- Son objetos que se los puede nombrar con un apodo y se les puede pasar muchos valores distintos separados
por comas, dentro de corchetes, al momento de declararlos o despues a través de índices de posición o con sus funciones. Ejemplos:

Crear un arreglo de 3 elementos en su declaracion:

```
var lista = ['Ana',12,true];
```

Declarar un arreglo vacío o sin elementos:

```
var lista02 = [];
```

Al arreglo declarado, de apodo o nombre, lista02 añádale, en la posición 0, un valor string:

```
lista02[0] = 'OCP0120';
```

Al mismo arreglo anterior añádele, al final, otro valor string con la funcion push():

```
lista02.push('AMP0238');
```

Al arreglo, declarado primerito, de apodo lista añádale, al inicio, un valor numérico con la función unshift():

```
lista.unshift(73456);
```

Al arreglo, declarado primerito, de apodo lista añádale, en la posición 3, un arreglo vacío:

```
lista[3] = [];
```

1. En un archivo de texto separado que debes crear, escribe explicaciones de los siguientes conceptos como si se lo estuvieras explicando a un niño de 12 años. Hacer esto te ayudará a descubrir rápidamente cualquier agujero en tu comprensión.

    * Objetos
    * Propiedades
    * Métodos
    * Bucle for…in
    * Notación de puntos vs notación de corchetes

Explicaciones:

Objetos.- El objeto almacena un conjunto o colección de varios tipos de datos que están ligados a un texto(apodo) clave para poder extraer su valor grabado, a este par << 'texto clave':value >> se le conoce como propiedad. A esta colección de tipos se la hace dentro de llaves y a su vez se les declara con un nombre o etiqueta para identificarlo. Ejemplo:

```
let ecuador = {
      'D T': 'Alfaro' ,
      arqueros: ['Hernán Galindez', 'Moisés Ramírez', 'Alexander Domínguez'] ,
      delanteros: ['Valencia','Estrada', 'Reasco', 'Rodríguez']
};
```

Observación: La etiqueta del objeto es ecuador y tiene 3 claves que son 'D T' , 'arqueros' y 'delanteros'.

Propiedades.- Las propiedades de los objetos son cada uno de los pares << 'clave':valor >>, el valor puede ser de cualquier tipo, o sea string, number, boolean, null, undefined, puede hasta ser otro objeto y otras entidades. Por ejemplo del objeto anterior una propiedad es el par 'D T': 'Alfaro'

Métodos.- Mencionamos en el concepto anterior otras entidades y una de estas son los métodos que no son más que funciones expresadas dentro del objeto y tratadas igual como propiedades. Ejemplo de una funcion como una propiedad más del objeto, aumentemos al objeto denominado ecuador un valor función con clave 'tactica'.

```
ecuador['tactica'] = function nf() {  return ['uno', 'uno', 'cuatro', 'tres', 'dos'];  };
```

Bucle for…in.- Con este bucle podemos acceder a todos los elementos(propiedades) que agrupa el objeto, esto de acceder a todos los elementos se le conoce como recorrer el objeto completamente. Veamos como se lo usa:

```
for (let propiedad in ecuador) {

      console.log(propiedad + ' : ' + ecuador[propiedad]);
}
```

Notación de puntos vs notación de corchetes.- Son convenciones o formas creadas por los desarrolladores para acceder a las propiedades del objeto, con la notación punto, usamos el símbolo punto para separar y que clave del objeto escojo para operar con su valor; en cambio con la notación de corchetes usamos los corchetes, de abrir y cerrar, para encerrar la clave(en string) con la que voy a operar con su valor. Ejemplo:

Con Notación de punto accedo escribiendo así > ecuador.arqueros;

Con Notación de corchetes accedo escribiendo así > ecuador['arqueros'];

1. En un archivo de texto separado que debes crear, escribe explicaciones de los siguientes conceptos como si se lo estuvieras explicando a un niño de 12 años. Hacer esto te ayudará a descubrir rápidamente cualquier agujero en tu comprensión.
  * prototype
  * Constructors (de Clases)

Explicaciones:
Prototype.- Son simplemente mecanismos por los que los objetos de Javascript heredan(les dan o les pasan) propiedades de otros objetos de Javascript y también es posible lo de la herencia porque todos pertenecen al objeto prototype que viene implementado en javascript.

Ejemplo:
Vamos a construir un objeto function que hará de Construcctor de propiedades para otros, crearemos la función constructora Car de esta manera, le pusimos inciando con una C en mayúscula a Car por acuerdo entre desarrolladores:

```
function Car() {
  this.doors = 4;
  this.maxSpeed = 240;
  this.getSpeed = function gS() {
    return this.maxSpeed;
  }
}
```

Observación: function Car() { ... } es un objeto Prototype de javascript, por lo que le podemos adicionar una propiedad de la siguiente manera:

```
Car.prototype.getSpeed = function pgS() {
  return 120;
}
```

Ahora vamos a crear un objeto que herede estas propiedades creadas, lo hacemos escribiendo lo de acontinuación:

```
let carInstance = new Car();
```

Observación: Habrás notado que se ha generado un poliformismo, o sea existe una funcion getSpeed tanto en Car como en prototype, si hacemos una llamada en console.log de ese método primero lo llama si existiera en Car, en caso de no lo llama al de prototype, para este ejemplo mostraría por consola el valor de 240, para verificarlo escribamos:

```
console.log(carInstance.getSpeed());
```

Si quisiéramos llamar al getSpeed de prototype, los haríamos escribiendo:

```
console.log(carInstance.__proto__.getSpeed());
```

Constructors (de Clases).- Los construcctores de clases son justamente lo que hemos utilzado en el ejercicio anterior, o sea valiéndonos de la funcion constructora como también lo podemos hacer por intermedio del uso de la palabra reservada class,

Construyamos como quedaría la clase usando class:

```
class Car {
  constructor() {
    this.doors = 4;
    this.maxSpeed = 240;
  }
  getSpeed() {
    return this.maxSpeed;
  }
}
```

Ahora instanciemos esta Clase, lo hacemos igual con new, nos quedaría así:

```
let instanciaCar = new Car();
```

Llamemos al método getSpeed, lo haré a través de un console.log:

```
console.log(instanciaCar.getSpeed());
```

```
//Funcion constructora empieza la etiqueta con Mayúscula por convención
function Car(make, model, year) {
  this.make = make;
  this.model = model;
  this.year = year;
}

//Añadir otra propiedad a la función constructora
Car.prototype.color = 'color original';

const car1 = new Car('Eagle', 'Talon TSi', 1993);

console.log(car1.color);    // 'color original'

car1.color = 'black';
console.log(car1.color);    // 'black'

console.log(Object.getPrototypeOf(car1).color); // 'color original'

//Ver la propiedad superior color que cree
car1.__proto__;

//Ver la propiedad sobrecargada color que cree
car1.color;

//Solo el valor de la propiedad superior color
car1.__proto__.color;


//Otro ejemplo de herencia y poliformismo
function Persona(edad) {
  this.edad = edad;
}

Persona.prototype.getEdad = function nf() {
  return 'tiene: ' + this.edad + ' años';
};

var per1 = new Persona(22);
var per2 = new Persona(34);
var per3 = new Persona(2335);


//Hago poliformismo a objeto per3
per3['getEdad'] = function nf() {
  return this.edad/2;
};

per1.getEdad();
per2.getEdad();
per3.getEdad();
```

```
//Otra forma de asemejarse a una clase(coje de otro objeto que hace de plantilla), definiendo las propiedades en prototype

let Persona = {
  nombre: 'Por defecto',
  edad: 'Por defecto',
  getEdad: function pege() {
    return this.edad;
  }
  getNombre: function pngn() {
    return this.nombre;
  }
}

var juan = Object.create(Persona);
juan.__proto__;


//Ultima forma reciente
class Persona {
  constructor(nombre, apellido) {
    this.nombre = nombre;
    this.apellido = apellido;
  }

  getNombre() {
    return this.nombre;
  }

  saludar() {
    console.log('Hola!' + this.nombre);
  }
}

var toni = new Persona('Toni', 'Tralice');
toni.saludar();

//Herencia
class Empleado extends Persona {
  constructor (nombre, apellido, empleo, sueldo) {
    super(nombre, apellido);
    this.empleo = empleo;
    this.sueldo = sueldo;
  }
}

var toni = new Empleado('Toni', 'Tralice', 'Profesor', 100);
toni.saludar();
```

FUNCIONES CALLBACK
Cuando pasamos una función(objeto) como argumento y lo recibe el parámetro(objeto) de la otra función se la llama Cb(callback), es como que el parámetro es una referencia de la función-objeto que recibe.

Ejercicio:

```
//Declaramos y definimos, como expresión, la función objeto que hace un proceso.
let espacea = function espa(texto_p) {
  if (typeof(texto_p)==='string' || typeof(texto_p)==='number') {
      if (typeof(texto_p)==='number') { texto_p += ''; }  //si el parámetro es numérico lo transforma a string
      let vcespa='';
      for (let c=0; c<texto_p.length; c++) {
        if (c<texto_p.length-1) {
          vcespa += texto_p[c] + ' ';
        } else {
          vcespa += texto_p[c];
        }
      }
      return vcespa;
  } else {
    return 'No es un valor que se pueda espaciar';
  }
};

//Definimos la función que le da los insumos o es la que le pasa valores para que procese la otra función que declaramos y definimos.
function esteV(valor_p, cb_espa_p) {
  return (cb_espa_p(valor_p));
}

//Comprobamos si nos funciona el cb, con algunos valores
esteV('hola', espacea);
esteV(2335, espacea);
esteV(true, espacea);
esteV('a', espacea);
```

```
EJERCICIOS:
function colors(color) {
  //La función recibe un color. Devolver el string correspondiente:
  //En caso que el color recibido sea "blue", devuleve --> "This is blue"
  //En caso que el color recibido sea "red", devuleve --> "This is red"
  //En caso que el color recibido sea "green", devuleve --> "This is green"
  //En caso que el color recibido sea "orange", devuleve --> "This is orange"
  //Caso default: devuelve --> "Color not found"
  //Usar el statement Switch.
  switch (color) {
    case 'blue':
    case 'red':
    case 'green':
    case 'orange':
      return 'This is ' + color;
    default:
      return 'Color not found';
  }
}


function esPrimo(numero) {
  // Devuelve "true" si "numero" es primo
  // De lo contrario devuelve "falso"
  // Pista: un número primo solo es divisible por sí mismo y por 1
  // Pista 2: Puedes resolverlo usando un bucle `for`
  // Nota: Los números 0 y 1 NO son considerados números primos
  if (numero===1 || numero===0) {
    return false;
  } else {
    let Es=true;
    for (let divisor=2; divisor<Math.abs(numero); divisor++) {
      if (Math.abs(numero)%divisor===0) {
        Es=false;
        break;
      }
    }
    return Es;
  }
}


function agregarItemAlComienzoDelArray(array, elemento) {
  // Añade el "elemento" al comienzo del array
  // y devuelve el array
  // Pista: usa el método `.unshift`
  // Tu código:
  array.unshift(elemento);
  return array;
}
```

```
function numeroMasGrande(numeros) {
  // "numeros" debe ser una matriz de enteros (int/integers)
  // Devuelve el número más grande
  // Tu código:
  return Math.max(...numeros);
}


function mesesDelAño(array) {
  //Dado un array que contiene algunos meses del año desordenados, recorrer el array buscando los meses de
  // "Enero", "Marzo" y "Noviembre", guardarlo en nuevo array y retornarlo.
  //Si alguno de los meses no está, devolver: "No se encontraron los meses pedidos"
  // Tu código:
  let nuevoArray=[];
  for (let c_a=0; c_a<array.length; c_a++) {
          if (array[c_a]==='Enero' || array[c_a]==='Marzo' || array[c_a]==='Noviembre') {
              let yaEsta=false;
              for (let c_nA=0;c_nA<nuevoArray.length;c_nA++) {
                  if (array[c_a]===nuevoArray[c_nA]) {
                      yaEsta=true;
                      break;
                  }
              }
              if (yaEsta===false) {
                nuevoArray.push(array[c_a]);
              }
          }
  }

  if (nuevoArray.length===3) {
      return nuevoArray;
  } else {
      return 'No se encontraron los meses pedidos';
  }
}


function continueStatement(numero) {
  //Iterar en un bucle aumentando en 2 el numero recibido hasta un límite de 10 veces.
  //Guardar cada nuevo valor en un array.
  //Devolver el array
  //Cuando el número de iteraciones alcance el valor 5, no se suma en ese caso y se continua con la siguiente iteración
  //Pista: usá el statement 'continue'
  // Tu código:
  let array=[];
  let enDos=numero;
  let lim;
  for (lim=1; lim<=10; lim++) {
      if (lim===5) {
          continue;
      }
      enDos += 2;
      array.push(enDos);
  }
  return array;
}
```

```
function crearGato (nombre, edad) {
  // Crear un nuevo objeto con la propiedad "nombre" y el valor definido como el argumento "nombre".
  // Agrega una propiedad al objeto con el nombre "edad" y usa el valor definido en el argumento "edad"
  // Agrega un método (funcion) llamado "meow" que devuelva el string "Meow!"
  // Devuelve el objeto
  // Tu código:
  let gato_obj = { 'nombre':nombre };
  gato_obj['edad'] = edad;
  gato_obj['meow'] = function nf() {  return 'Meow!';  };

  return gato_obj;
}


function eliminarPropiedad (objeto, unaPropiedad) {
  // Elimina la propiedad de objeto cuyo nombre está pasado por el parametro unaPropiedad
  // tip: tenes que usar bracket notation
  // Devuelve el objeto
  // Tu código:
  delete objeto[unaPropiedad];
  return objeto;
}


function nuevoUsuario (nombre, email, password) {
  // Crea un nuevo objeto con las propiedades coincidiendo con los argumentos que se pasan a la función
  // Devuelve el objeto
  // Tu código:
  let usuario_obj = {};
  usuario_obj['nombre'] = nombre;
  usuario_obj['email'] = email;
  usuario_obj['password'] = password;
  return usuario_obj;
}


function tieneEmail (usuario) {
  // Devuelve "true" si el usuario tiene un valor definido para la propiedad "email"
  // De lo contrario, devuelve "false"
  // Tu código:

  //
  //Corrijo lo que se pide, se presta a una confusión el valor null -> SI es un valor que se puede colocar a una clave o sea
  //SI existe la clave pero según para el que realizó el ejercicio del test, NO existe.  Debió haber redactado de otra manera
  //para pasar el test sin hacer trampa.  Por decirles algo así debió redactar...
  //

  //... Devuelve true si en el objeto usuario existe la clave 'email' y false en caso de no existir(undefined) o de tener como valor el null.
  //Tú código:

  return typeof(usuario['email']) !== 'undefined' && usuario['email'] !== null;
}
```

```
function sumarLikesDeUsuario (usuario) {
  // "usuario" tiene una propiedad llamada "posts" que es un array
  // "posts" es un array de objetos "post"
  // Cada objeto "post" tiene una propiedad llamada "likes" que es un entero (int/integer)
  // Suma todos los likes de todos los objetos "post"
  // Devuelve la suma
  // Tu código:

  //usuario['posts'] es array
  //post['likes'] es number

  let totalLikes=0;
  for (let ck=0; ck<usuario['posts'].length; ck++) {
      totalLikes += usuario['posts'][ck]['likes'];
  }
  return totalLikes;
}


function agregarMetodoCalculoDescuento (producto) {
  // Agregar un método (función) al objeto "producto" llamado "calcularPrecioDescuento"
  // Este método debe multiplicar el "precio" del "producto" ("producto.precio" o "producto[precio]") y "porcentajeDeDescuento" para obtener el
descuento
  // El método resta el descuento del precio y devuelve el precio con descuento
  // Devuelve el objeto "producto" al final de la función
  // Ejemplo:
  // producto.precio -> 20
  // producto.porcentajeDeDescuento -> 0.2 (o simplemente ".2")
  // producto.calcularPrecioDescuento() -> 20 - (20 * 0.2)
  // Tu código:
  producto['calcularPrecioDescuento'] = function nf() {
      return this['precio'] - (this['precio']*this['porcentajeDeDescuento']);
  };
  return producto;
}
```

```
function crearUsuario() {
  // Crea una Clase de ES6 o una función constructor llamada "Usuario"
  // Debe aceptar un objeto "opciones" con las propiedades "usuario", "nombre", "email" y "password"
  // En el `contructor`, define el usuario, el nombre, el email y la contraseña
  // El `contructor` debe tener un método llamado "saludar" en su `prototype` que devuelva una string 'Hola, mi nombre es {{nombre}}'
  // {{nombre}} debe ser el nombre definido en cada instancia
  // Devuelve la clase
  // Tu código:
  class Usuario {
      constructor(objOpc_p) {
          this.usuario = objOpc_p['usuario'];
          this.nombre = objOpc_p['nombre'];
          this.email = objOpc_p['email'];
          this.password = objOpc_p['password'];
      }
  }
  Usuario.prototype.saludar = function ups() {
      return `Hola, mi nombre es ${this.nombre}`;
  };
  //let usuarioDeUsu = new Usuario({usuario:'001', nombre:'Adalberto', email:'adalsus@laei.com', password:'hy77667'});
  //console.log(usuarioDeUsu.saludar());
  return Usuario;
}


function agregarMetodoPrototype(Constructor) {
  // Agrega un método al Constructor del `prototype`
  // El método debe llamarse "saludar" y debe devolver la string "Hello World!"
  // Tu código:
  Constructor.prototype.saludar = function cps() {
      return 'Hello World!';
  };
  return Constructor;
}

function agregarStringInvertida() {
  // Agrega un método al prototype de String que devuelva la misma cadena de caracteres, pero invertida.
  // El método debe llamarse "reverse"
  // Ej: 'menem'.reverse() => menem
  // 'toni'.reverse() => 'inot'
  // Pista: Necesitarás usar "this" dentro de "reverse"
  String.prototype.reverse = function spr() {
      let texto = this;
      let alreves = '';
      for (let i = texto.length - 1; i >= 0; i--) {
        alreves += texto[i];
      }
      return alreves;
  };
  return String;
}
```

```
// ---------------------------------------------------------------------------//
  //Crea el constructor de la clase "Persona"
  //Debe tener las propiedades: "nombre", "apellido", "edad" y "domicilio"
  //Debe tener un método llamado "detalle" que nos devuelve un objeto con las propiedades de la persona y sus valores.
  //Ej: {
    //   Nombre: 'Juan',
    //   Apellido: 'Perez',
    //   Edad: 22,
    //   Domicilio: 'Saavedra 123'
    //  }

  class Persona {
    constructor(nombre, apellido, edad, domicilio) {
      // Crea el constructor:
      this['nombre'] = nombre;
      this['apellido'] = apellido;
      this['edad'] = edad;
      this['domicilio'] = domicilio;
    }

    detalle() {
          return {'Nombre':this['nombre'], 'Apellido':this['apellido'], 'Edad':this['edad'], 'Domicilio':this['domicilio']};
    };
}

function crearInstanciaPersona(nombre, apellido, edad, dir) {
  //Con esta función vamos a crear una nueva persona a partir de nuestro constructor de persona (creado en el ejercicio anterior)
  //Recibirá los valores "Juan", "Perez", 22, "Saavedra 123" para sus respectivas propiedades
  //Devolver la nueva persona creada
  let personaObj = new Persona(nombre, apellido, edad, dir);
  return personaObj;
}

function agregarMetodo() {
  //La función agrega un método "datos" a la clase Persona que toma el nombre y la edad de la persona y devuelve:
  //Ej: "Juan, 22 años"
  let Datos = {
      datos() {
          return `${this['nombre']}, ${this['edad']} años`;
      }
  };
  Object.setPrototypeOf(Persona.prototype, Datos);
  /*//Otra forma
  Persona.prototype.datos = function ppd() {
      return `${this['nombre']}, ${this['edad']} años`;
  };*/

  return Persona;
}
```

```
function mayuscula(nombre) {
  //La función recibe un nombre y debe devolver el mismo que recibe pero con su primer letra en mayúscula
  //ej: Recibe "mario" ----> Devuelve "Mario"
  //Tu código:

  //Como práctica lo resuelvo creando un callback
  let capital = function letraC(nom_p) {
      return `${nom_p['0'].toUpperCase()}${nom_p.slice(1)}`;
  };
  function eV(nombre_p, fcb_capi) {
      return fcb_capi(nombre_p);
  }
  return eV(nombre, capital);

  //Otra forma sin callback
  //return `${nombre['0'].toUpperCase()}${nombre.slice(1)}`;
}


function invocarCallback(cb) {
  // Invoca al callback `cb`
  //Tu código:
  return cb();
}

function operacionMatematica(n1, n2, cb) {
  //Vamos a recibir una función que realiza una operación matemática como callback junto con dos números.
  //Devolver el callback pasándole como argumentos los números recibidos.
  //Tu código:
  return cb(n1, n2);
}

function sumarArray(numeros, cb) {
  // Suma todos los números enteros (int/integers) de un array ("numeros")
  // Pasa el resultado a `cb`
  // No es necesario devolver nada
  //Tu código:
  let sumaTodos = numeros.reduce(function cb_r(vacu, va/*, i, cualArreglo*/){
      return vacu + va;
    },
    0 /*valor inicial de vacu*/
  );

  return cb(sumaTodos);
}
```

```
function forEach(array, cb) {
  // Itera sobre la matriz "array" y pasa los valores al callback uno por uno
  // Pista: Estarás invocando a `cb` varias veces (una por cada valor en la matriz)
  //Tu código:
  array.forEach(function cb_fE(elemento/*, indice*/){ return cb(elemento); });
}


function map(array, cb) {
  // Crea un nuevo array
  // Itera sobre cada valor en "array", pásalo a `cb` y luego ubicar el valor devuelto por `cb` en un nuevo array
  // El nuevo array debe tener la misma longitud que el array del argumento
  //Tu código:
  let nuevoArray = [];
  array.map(
            function cb_m(element/*,index, verArray*/) {
              nuevoArray.push(cb(element));
              return element;
            }/*,
            thisArg*/
  );
  return nuevoArray;
}

function filter(array) {
  //Filtrar todos los elementos del array que comiencen con la letra "a".
  //Devolver un nuevo array con los elementos que cumplen la condición
  //Tu código:
  return array.filter(  function cb_f(element) { return element['0']==='a'; } );
}


function deObjetoAmatriz(objeto){
  // Escribe una función que convierta un objeto en una matriz, donde cada elemento representa
  // un par clave-valor en forma de matriz.
  //Ejemplo:
  /*objeto({
      D: 1,
      B: 2,
      C: 3
    }) ➞ [["D", 1], ["B", 2], ["C", 3]]*/
  //Escribe tu código aquí
  let listaArrays = [];
  for (let clave in objeto) {
      let eArray = [];
      eArray.push(clave, objeto[clave]);
      listaArrays.push(eArray);
  }
  //Otra forma
  //let listaArrays = Object.entries(objeto);

  return listaArrays;
}
```

```
function numberOfCharacters(string) {
  //La función recibe un string. Recorre el srting y devuelve el caracter con el número de veces que aparece
  //en formato par clave-valor.
  //Ej: Recibe ---> "adsjfdsfsfjsdjfhacabcsbajda" || Devuelve ---> { a: 5, b: 2, c: 2, d: 4, f: 4, h:1, j: 4, s: 5 }
  //Escribe tu código aquí
 let result = {};
 for (let valor of string) {
     result[valor] = result.hasOwnProperty(valor)   ?   ++result[valor]   :   1;
 }
  return result;
}


function capToFront(s) {
  //Realiza una función que reciba como parámetro un string y mueva todas las letras mayúsculas
  //al principio de la palabra.
  //Ejemplo: soyHENRY -> HENRYsoy
  //Escribe tu código aquí
  let maymi = function fan(texto) {
      let une=['',''];

      //Con un for clásico
      /*for (let p=0; p<texto.length; p++) {
          if ( texto[p]===texto[p].toUpperCase() ) {
              une[0] += texto[p];
          } else {
              une[1] += texto[p];
          }
      }*/

      //Con forEach
      Array.from(texto).forEach( function cb(letra) {
         if ( letra===letra.toUpperCase() ) {
             une[0] += letra;
         } else {
             une[1] += letra;
         }
      } );

      return une;
  };

  return maymi(s)[0]+maymi(s)[1];
}
```

```
function asAmirror(str) {
  //La función recibe una frase.
  //Escribe una función que tome la frase recibida y la devuelva de modo tal que se pueda leer de izquierda a derecha
  //pero con cada una de sus palabras invertidas, como si fuera un espejo.
  //Ej: Recibe ---> "The Henry Challenge is close!" || Devuelve ---> "ehT yrneH egnellahC si !esolc"
  //Escribe tu código aquí

  let fr = function fn(frase) {
      let cpr = [];
      let cnvArray = frase.split(' ');
      cnvArray.forEach( function cb(word) {
          cpr.push(Array.from(word).reverse().join(''));
      } );
      return cpr.join(' ');
  };

  return fr(str);
}


function capicua(numero){
  //Escribe una función, la cual recibe un número y determina si es o no capicúa.
  //La misma debe retornar: "Es capicua" si el número se número que se lee igual de
  //izquierda a derecha que de derecha a izquierda. Caso contrario retorna "No es capicua"
  //Escribe tu código aquí

  let numr = function nf(valorn) {
    let snr = Array.from(valorn+'').reverse().join('');
    return parseInt(snr);
  };

  return numero===numr(numero)  ?  'Es capicua' : 'No es capicua';
}


function deleteAbc(cadena){
  //Define una función que elimine las letras "a", "b" y "c" de la cadena dada
  //y devuelva la versión modificada o la misma cadena, en caso de contener dichas letras.
  //Escribe tu código aquí

  let eli_abc = function nf(frase) {
      let resul='';
      Array.from(frase).forEach( function cb(letra) {
         if (letra!=='a' && letra!=='b' && letra!=='c') {
             resul += letra;
         }
     } );
     return resul;
  };

  return eli_abc(cadena);
}
```

```
function sortArray(arr) {
  //La función recibe una matriz de strings. Ordena la matriz en orden creciente de longitudes de cadena
  //Ej: Recibe ---> ["You", "are", "beautiful", "looking"] || Devuelve ---> [“You", "are", "looking", "beautiful"]
  //Escribe tu código aquí

  let arrO = Array.from(arr);
  let arrT = [];
  for (let valor of arrO) {
      arrT.push(valor.length);
  }

  let auxT, auxV;
  for ( let c in arrT ) {
      let cS = (parseInt(c)+1)+'';
      for ( cS in arrT ) {
          if (parseInt(cS) > parseInt(c)) {
              if (arrT[c]>arrT[cS]) {
                  auxT = arrT[c];
                  auxV = arrO[c];
                  arrT[c] = arrT[cS];
                  arrO[c] = arrO[cS];
                  arrT[cS] = auxT;
                  arrO[cS] = auxV;
              }
          }
      }
  }
  return arrO;
}
function buscoInterseccion(arreglo1, arreglo2){
  //Existen dos arrays, cada uno con 5 números. A partir de ello, escribir una función que permita
  //retornar un nuevo array con la intersección de ambos elementos. (Ej: [4,2,3] unión [1,3,4] = [3,4].
  //Si no tienen elementos en común, retornar un arreglo vacío.
  //Aclaración: los arreglos no necesariamente tienen la misma longitud
  //Escribe tu código aquí
  let ca1 = Array.from(arreglo1);
  let ca2 = Array.from(arreglo2);
  let ame, ama;
  if (ca1.length<=ca2.length) {
      ame = ca1;
      ama = ca2;
  } else {
      ame = ca2;
      ama =ca1;
  }
  let inters = [];
  for (let valor of ame) {
    if (ama.includes(valor)) {
        inters.push(valor);
        ama = ama.filter( (item) => item !== valor );
    }
  }
  inters.sort( (a,b) => { return a-b; } );  /*<<--Ordeno, uso la manera directa*/

  return inters;
}
```

```
function stringMasLarga(strings) {
  // La función llamada 'stringMasLarga', recibe como argumento un arreglo de strings llamado 'strings'
  // y debe devolver el string más largo que hay en el arreglo (Es decir el de mayor cantidad de caracteres)
  // Ej:
  // stringMasLarga(['hi', 'hello', 'ni hao', 'guten tag']); debe retornar 'guten tag'
  // stringMasLarga(['JavaScript', 'HTML', 'CSS']); debe retornar 'JavaScript'

  // Tu código aca
  let lenM=strings[0].length;
  let strM=strings[0];
  for (let p=1; p<strings.length;p++) {
      if (strings[p].length>lenM) {
          lenM=strings[p].length;
          strM=strings[p];
      }
  }
  return strM;
}


function buscarAmigo(amigos, nombre) {
  // La funcion llamada 'buscarAmigo' recibe como argumento un array llamado 'amigos' que contiene
  // en cada posición del arreglo un objeto que tiene como propiedades 'nombre' y 'edad'. También
  // recibe un string llamado 'nombre'.
  // Debe devolver el objeto cuya propiedad 'nombre' coincida con el string 'nombre' recibido por argumento.
  // Ej:
  //  var amigos = [{ nombre: 'toni', edad: 33 } , { nombre: 'Emi', edad: 25 }];
  //  buscarAmigo(amigos, 'toni') debe devolver { nombre: 'toni', edad: 33 };

  // Tu código aca:
  let  objEs = {};
  amigos.forEach( function cb(v_obj) {
      if (Object.keys(objEs).length===0) {
        if (v_obj['nombre']===nombre) {
            Object.assign(objEs, v_obj);
        }
      }
  } );

  return objEs;
}


function numeroSimetrico(num) {
  // La funcion llamada 'numeroSimetrico' recibe como argumento un numero entero 'num'
  // Esta devuelve true o false dependiendo de si el número es simétrico o no.
  // Un número es simétrico cuando es igual a su reverso.
  // Ej:
  // numeroSimetrico(11711) devuelve true

  // Tu código:
  let snum = num+'';
  let snum_rev = Array.from(snum).reverse().join('');
  return snum===snum_rev;
}
```

```
function pluck(array, propiedad) {
  // La función llamada 'pluck' recibe como argumento un array de objetos llamado 'array' y el nombre de una
  // propiedad.
  // La función debe devolver un nuevo arreglo con solo los valores dentro de la propiedad recibida
  // Ej:
  // var productos = [{ name: 'TV LCD', price: 100}, { name: 'Computadora', price: 500 }]
  // productos.pluck(productos, 'name') debería devolver ['TV LCD', 'Computadora']
  // Pista: es una buena oportunidad para usar map.

  // Tu código acá:
  let arrayValores = [];
  array.forEach( function cb(v_obj) {
      arrayValores.push( v_obj[propiedad] );
  } );

  return arrayValores;
}
```

```
function crearClasePersona() {
  class Persona {
    constructor(nombre, edad, hobbies, amigos) {
      // El constructor de la clase Persona recibe nombre (string), edad (integer), hobbies (array de strings), amigos (array de objetos)
      // Inicializar las propiedades de la persona con los valores recibidos como argumento

      // Tu código aca:
      this['nombre'] = nombre;
      this['edad'] = edad;
      this['hobbies'] = hobbies;
      this['amigos'] = amigos;
    }

    addFriend(nombre, edad) {
      // El método 'addFriend' recibe un string 'nombre' y un entero 'edad' y debe agregar un objeto:
      // { nombre: nombre, edad: edad} al arreglo de amigos de la persona.
      // No debe retornar nada.

      // Tu código aca:
      let newobj = {'nombre' : nombre, 'edad' : edad};
      this['amigos'].push(newobj);
    }

    addHobby(hobby) {
      // El método 'addHobby' recibe un string 'hobby' y debe agregarlo al arreglo de hobbies de la persona.
      // No debe retornar nada.

      // Tu código aca:
      let newstr = hobby;
      this['hobbies'].push(newstr);
    }
    getFriends() {
      // El método 'getFriends' debe retornar un arreglo con sólo los nombres del arreglo de amigos
      // de la persona.
      // Ej:
      // Suponiendo que la persona tiene estos amigos: [{nombre: 'martin', edad: 31},{nombre: 'toni', edad: 33}]
      // persona.getFriends() debería devolver ['martin', 'toni']

      // Tu código aca:
      let arrayAmis = [];
      this['amigos'].forEach( function cb(v_obj) {
          arrayAmis.push(v_obj['nombre']);
      } );

      return arrayAmis;
    }

    getHobbies() {
      // El método 'getHobbies' debe retornar un arreglo con los hobbies de la persona
      // Ej:
      // persona.getHobbies() debe devolver ['correr', 'dormir', 'nadar']

      // Tu código aca:
      return this['hobbies'];
    }
```

```
    getPromedioEdad() {
      // El método 'getPromedioEdad' debe retornar el promedio de edad de los amigos de una persona
      // Ej:
      // Si la persona tuviera estos amigos:
      // {
      //   amigos: [{
      //     nombre: 'toni',
      //     edad: 33,
      //   }, {
      //     nombre: 'Emi',
      //     edad: 25
      //   }]
      // }
      // persona.getPromedioEdad() debería devolver 29 ya que (33 + 25) / 2 = 29

      // Tu código aca:
      let edadesT = this['amigos'].reduce(  function cb(acumu, v_obj) {
          return acumu + v_obj['edad'];
      }, 0  );

      return edadesT/this['amigos'].length;
    }
  };

  return Persona;
}
```

```
function filtrar(funcion) {
  // Escribi una función filtrar en el prototipo de Arrays,
  // que recibe una funcion (callback) que devuelve true o false.
  // filtrar los elementos de ese arreglo en base al resultado de esa funcion
  // comparadora, devolver un nuevo arreglo con los elementos filtrados.
  // NO USAR LA FUNCION FILTER DE LOS ARREGLOS.
  // ej:
  // var productos = [{
  //   price: 100,
  //   name: 'tv'
  // }, {
  //   price: 50,
  //   name: 'phone'
  // }, {
  //   price: 30,
  //   name: 'lamp'
  // }]
  // productos.filtrar(function(p) {
  //   return p.price >= 50;
  // }) => [{price: 100, name:'tv'}]

  Array.prototype.filtrar = function nf(fcb) {
    let filtrados = [];
    for (let k in this) {
        if ( fcb(this[k]) ) {
           filtrados.push(this[k]);
        }
    }
    return filtrados;
  };

};
```