Ejercicio 1

Vamos a intentar familiarizarnos con la Programación Orientada a Objetos en Javascript mediante este ejercicio.

Usando la sintaxis de ECMAScript, escribe una clase Punto que representa un punto en un espacio de dos dimensiones. Un punto tiene propiedades x e y, que recibirá como parámetros del método constructor. También tiene un método suma, que toma como parámetro otro punto y devuelve la suma de ambos, es decir, un nuevo punto cuyo valor "x" sea la suma de los valores "x" del objeto actual y del que le llega como parámetro y cuyo valor "y" sea también la suma de valores de propiedades "y".

Al ejecutar tu código con la siguiente instrucción:

```
console.log(new Punto(1, 2).suma(new Punto(2, 1)))
```

El resultado por pantalla debe ser:

```
Punto{x: 3, y: 3}
```

Ejercicio 2

Reescribe el siguiente código para que use la sintaxis ECMAScript (*class*, *constructor*, etc...) en lugar de manipular directamente prototipos.

"Locutor" es una clase que expone el método "dice". Al llamarlo, muestra por pantalla el texto recibido como parámetro junto con el nombre del locutor.

"Feriante" es un subtipo de Locutor que "grita" el texto recibido (convirtiéndolo a mayúsculas).

```
function Locutor(nombre, verbo) {
    this.nombre = nombre
    this.verbo = verbo || "dice"
}
Locutor.prototype.dice = function(text) {
    console.log(this.nombre + " " + this.verbo + " '" + text + "'")
}
function Feriante(nombre) {
    Locutor.call(this, nombre, "grita")
}
Feriante.prototype = Object.create(Locutor.prototype)
Feriante.prototype.dice = function(text) {
    Locutor.prototype.dice.call(this, text.toUpperCase())
```

```
}
new Feriante("Mr. Crecepelo").dice("Me lo quitan de las manos")
```

Por pantalla se espera la siguiente cadena:

```
Mr. Crecepelo grita 'ME LO QUITAN DE LAS MANOS'
```

Nota: fíjate que, para simular herencia con prototipos, en lugar de:

```
Feriante.prototype = new Locutor();
```

se ha usado una construcción similar:

```
Feriante.prototype = Object.create(Locutor.prototype);
```

Fíjate también que se nos “ha olvidado” cambiar el constructor de Feriante…

```
let keitel = new Feriante("Sr. Lobo");
keitel.constructor.name
--> Locutor (!!!)
```

(en la solución ECMAScript esto no debería ocurrir...)

Ejercicio 3

Queremos reescribir la clase Punto usando prototipos (es decir, el sistema de clases sin ECMAScript) en lugar de sintaxis ECMAScript (imagínate que tenemos que implementarlo usando un navegador que no soporta ECMAScript).

```
[tu código aquí]
let punto = new Punto(1, 2).suma(new Punto(2, 1));
console.log(punto.x, punto.y); // Se espera: 3 3
console.log(punto instanceof Punto); // Se espera: true
console.log(punto.constructor.name); // Se espera Punto
```

Ejercicio 4

Añade un método sig al objeto literal contador (recuerda la introducción a las clases en Javascript). Este método debe devolver el valor actual de *cont e* incrementarlo en una unidad. Usa el operador ++.

```
var contador = {
```

```
    cont: 0
    [tu código aquí]
}
console.log(contador.sig()) // → 0
console.log(contador.sig()) // → 1
console.log(contador.sig()) // → 2
```

Ejercicio 5

map, *filter* y *reduce* son tres métodos[1] de la clase array muy usados en programación funcional con JavaScript. Puedes ver su uso básico con ejemplos aquí: https://medium.com/poka-techblog/simplify-your-javascript-use-map-reduce-and-filter-bd02c593cc2d

En este ejercicios queremos usar filter() y reduce() para obtener la solución al siguiente problema:

Dado un objeto literal *almacen* con los siguientes elementos:

```
const almacen = [
    {tipo: "lavadora", valor: 5000},
    {tipo: "lavadora", valor: 650},
    {tipo: "vaso", valor: 10},
    {tipo: "armario", valor: 1200},
    {tipo: "lavadora", valor: 77}
]
let totalValorLavadoras = tu código aquí;
console.log (totalValorLavadoras); // se espera 5727
```

Haz uso de *filter* y *reduce* para obtener el precio total de las lavadoras del almacén.

Ejercicio 6

Otra de las novedades de ECMAScript más apreciadas es la función flecha (*arrow function*), =>. Refactoriza el siguiente código para hacer uso de la función *arrow* (=>)

```
almacen.forEach(function(item) {
     console.log(item.valor);
});
```

Usa el objeto *almacen* del ejercicio anterior.

---

[1] Métodos o funciones de orden superior. Una función de orden-superior es una función que toma como argumento otra función, o que devuelve una función como resultado.

Ejercicio 7

Refactoriza el ejercicio 5 para hacer uso de la función *arrow*.

Nota: usa el nombre de variable *item* en el parámetro de la función *arrow* para *filter*.

Ejercicio 8

Queremos refactorizar la siguiente función que encapsula el código necesario para mantener un array ordenado. El constructor recibe una función comparador que toma dos parámetros (a,b) y devuelve un número entero negativo si a es menor que b (según el comparador), 0 si son iguales, y un número entero positivo si b es menor que a.[2]

```
function ArrayOrdenado(comparador) {
    this.comparador = comparador
    this.contenido = []
}
ArrayOrdenado.prototype.findPos = function(elt) {
    for (var i = 0; i < this.contenido.length; i++) {
        if (this.comparador (elt, this.contenido [i]) < 0) break
    }
    return i
}
ArrayOrdenado.prototype.insert = function(elt) {
    this.contenido.splice(this.findPos(elt), 0, elt)
}
var ordenado = new ArrayOrdenado(function(a, b) { return a - b });
ordenado.insert(5);
ordenado.insert(1);
ordenado.insert(2);
ordenado.insert(4);
ordenado.insert(3);
console.log("array:", ordenado.contenido);
// array: [1, 2, 3, 4, 5]
```

El objetivo es:

- convertir el código a ECMAScript 6+
- refactorizar el bucle *for* (eliminarlo) para que en su lugar se haga uso del método *findIndex* (lo que evita además la función *indexOf*). El método *findIndex* ofrecido por la clase *Array*, toma como parámetro una función X y devuelve el índice del primer elemento del array para el que la función X devuelve true. Por ejemplo:
  [1,2,3].findIndex(x => x > 1) devuelve 1
- usar funciones *arrow* => siempre que sea posible.

---

[2] Este método de comparación tal vez te suene, porque es exactamente igual que el método compareTo() de Java: https://docs.oracle.com/javase/7/docs/api/java/lang/Comparable.html#compareTo(T)