# Unidad 3. Estructuras definidas por el usuario en JavaScript.
# Boletín de actividades

## Funciones.

Una función es un conjunto de sentencias que se ejecutan en bloque. Se pueden entender como un subprograma que el desarrollador puede llevar a cabo cuando lo necesite. Las funciones constituyen una parte esencial del lenguaje JavaScript y son la base de la programación funcional.

Este es un ejemplo de *declaración de una función*:

```
function decirHola() {
    //Aquí empieza el cuerpo de la función.
    console.log("Hello world!");
    console.log("¡Hola mundo!");
    //Aquí acaba el cuerpo de la función.
}

decirHola();
```

La declaración de la función `decirHola` no ejecuta el cuerpo de la función. Para *llamar* una función (también llamado *correr*, *ejecutar* o *invocar*) se debe usar su nombre seguido de paréntesis.

### Valores de retorno.

La palabra reservada `return` en el cuerpo de una función hará que ésta finalice inmediatamente y se devuelva el valor especificado. En JavaScript la llamada a una función trabaja como una expresión y por tanto siempre se resolverá a un determinado valor. Si no se llama específicamente a `return`, se devolverá el valor `undefined`.

```
function obtenerSaludo() {
    return "Hello world!";
}

obtenerSaludo();    //Hello world!
```

### Invocar versus referenciar.

Es importante entender la diferencia entre invocar a una función o referenciarla. En JavaScript las funciones son objetos y se pueden referenciar para pasarlas a otras funciones o asignarlas a variables.

Si se indican los paréntesis, la función se está invocando. Si no se indican los paréntesis, la función se está referenciando.

```
obtenerSaludo();    //Hello world!
obtenerSaludo;      //function obtenerSaludo()
```

## Argumentos.

El método principal para pasar información a una función son los argumentos, también llamados parámetros. La siguiente función recibe 2 parámetros:

```
function avg(a, b) {
    return (a + b) / 2;
}
```

Cuando una función se invoca, los argumentos reciben un valor. Hay que tener en cuenta que los argumentos sólo existen dentro del cuerpo de la función incluso si hay variables fuera con el mismo nombre:

```
const a = 5, b = 10;
avg(a, b);
```

## Actividad 1. Argumentos por defecto.

En ES6 es posible definir valores por defecto para los argumentos de una función. Prueba el siguiente ejemplo y observa los valores de salida en la invocación de las funciones:

```
function f(a, b = "default", c = 3) {
    return `${a} - ${b} - ${c}`;
}

f(5, 6, 7);
f(5, 6);
f(5);
f();
```

## Actividad 2. Número indeterminado de argumentos.

En ES6 es posible pasar a una función un número no determinado de argumentos gracias al operador de propagación (*spread operator*): "…". Prueba el siguiente ejemplo:

```
function anadirPrefijo(prefijo, ...palabras) {
    const palabrasConPrefijo = [];
    for (let i = 0; i < palabras.length; i++) {
        palabrasConPrefijo[i] = prefijo + palabras[i];
    }
    return palabrasConPrefijo;
}

anadirPrefijo("con", "verso", "vexo", "cavo");
```

Nota que tanto el operador de propagación como los argumentos por defecto siempre deben ser los últimos, en caso contrario JavaScript no será capaz de distinguir el resto.

## Funciones anónimas.

Las funciones anónimas son aquellas que no tienen identificador. Las siguientes son funciones equivalentes:

```
//Función convencional
function obtenerSaludo() {
    return "Hello world!";
}
//Función anónima
function () {
    return "Hello world!";
}
```

Su uso dependerá del contexto, si la función se va a usar más tarde se le asignará un identificador. Si la función sólo se va a pasar a otra función podemos declararla como una función anónima.

```
//Funciones equivalentes.
function decirHola() {
    consola.log("¡Hola mundo!");
}

boton.addEventListener("click", decirHola);

boton.addEventListener("click", function () {
    console.log("¡Hola Mundo!");
});
```

### Actividad 1. Transforma a funciones anónimas.

Reescribe estas funciones como funciones anónimas:

a)
```
function obtenerSaludo() {
    return “¡Hola!”;
}
```

b)
```
function obtenerSaludoConNombre(nombre) {
    return `Hola ${nombre}`;
}
```

c)
```
function sumar(a, b) {
    return a + b;
}
```

### Actividad 2. Funciones que reciben funciones.

En JavaScript hay numerosos ejemplos de funciones que necesitan otras funciones para llevar a cabo su tarea. Las funciones anónimas permiten simplificar y mejorar la legibilidad del código.

a) Utiliza la función `setTimeOut` para mostrar un saludo por consola una vez transcurridos 5 segundos.

## Notación flecha.

ES6 ha introducido una nueva sintáxis llamada *notación flecha* que permiten crear *funciones flecha*. El término flecha hace referencia al signo “=>”.

Las funciones flecha son funciones anónimas con una sintaxis simplificada para declarar más rápidamente una función. Una función flecha sigue las siguientes reglas:

- Se puede omitir la palabra reservada `function`.
- Si la función tiene un único argumento, se pueden omitir los paréntesis.
- Si el cuerpo de la función tiene una única expresión, se pueden omitir las llaves y la palabra reservada `return`.

```
//Función anónima.
const decirHola2 = function () {
    return "¡Hola mundo!";
}
//Notación flecha equivalente.
const decirHola3 = () => "!Hola Mundo!";
```

## Actividad 1. Transforma a funciones flecha.

Reescribe estas funciones anónimas utilizando la notación flecha:

a)
```
const obtenerSaludo = function() {
      return “¡Hola!”;
}
```

b)
```
const obtenerSaludoConNombre = function(nombre) {
      return `Hola ${nombre}`;
}
```

c)
```
const sumar = function(a, b) {
      return a + b;
}
```

## Actividad 2. Funciones que reciben funciones.

Reescribe el ejercicio de mostrar un saludo por consola transcurridos 5 segundos utilizando la notación flecha.

## Funciones puras.

Las funciones puras son más fáciles de testear, más fáciles de entender y más portables. Además son la base de la programación funcional. Siempre deberíamos crear este tipo de funciones.

Para que una función se considere pura debe cumplir dos condiciones:

1. La primera es que siempre **devuelva la misma salida para la misma entrada**.

2. La segunda es que **no debe tener efectos colaterales**. Esto quiere decir que la invocación de la función no debe alterar el estado del programa.

### Actividad 1. Identificación de funciones puras.

Indica si las siguientes funciones son puras y por qué:

Función 1:

```
function esAnioBisiesto() {
    const oAnio = new Date().getFullYear();
    if (oAnio % 4 !== 0) return false;
    else if (oAnio % 100 != 0) return true;
    else if (oAnio % 400 != 0) return false;
    else return true;
}
```

Función 2:

```
const aColores = ['red', 'orange', 'yellow', 'green', 'blue'];
let iIndiceColor = -1;

function obtenerSiguienteColor() {
    if (++iIndiceColor >= aColores.length) iIndiceColor = 0;
    return aColores[iIndiceColor];
}
```

### Actividad 2. Creación de funciones puras.

Reescribe las funciones anteriores como funciones puras.

## Más ejercicios de funciones.

### Actividad 1. Números pares/impares.

a) Crea una función que devuelva la palabra “par” o “impar” dependiendo del número que reciba como argumento.

b) Muestra 500 números aleatorios del 1 al 10.000 indicando al lado de ellos si el número es par o impar.

### Actividad 2. Dibuja una tabla HTML.

Crea una función que permita dibujar una tabla HTML. La función recibirá como parámetros el número de filas y columnas de la tabla. Por defecto la función creará una tabla de 10 filas y 4 columnas.

### Actividad 3. Saluda a la clase.

Crea una función que reciba todos los nombres de los alumnos de una clase y los salude por consola.

## Objetos

En JavaScript es mejor pensar en los objetos como contenedores de propiedades. Una propiedad es un par clave:valor. JavaScript permite crear objetos de 2 formas diferentes:

1. Definiendo y creando un único objeto.

2. Definiendo un tipo de objeto (clase) y luego creando distintos objetos a partir de él.

**1. Creando un único objeto.**
La creación de un único objeto se puede hacer usando el objeto Object:

```
const oPersona = new Object();
oPersona.nombre = "Pedro";
oPersona.apellidos = "Sánchez";
oPersona.edad = 50;
oPersona.colorOjos = "azul";
oPersona.nombreCompleto = function () {
    return `${this.nombre} ${this.apellidos}`;
};
```

o utilizando la notación literal:

```
const oPersona = {
    nombre: "Pedro",
    apellidos: "Sánchez",
    edad: 50,
    colorOjos: "azul",
    nombreCompleto: function () {
        return `${this.nombre} ${this.apellidos}`;
    }
};
```

Por mantenibilidad **utiliza siempre la notación literal**.

**2. Creando un tipo de objeto (clase).**

Hasta la versión ES5 no existía una notación para crear clases como tal. Las funciones suplieron esta característica para permitir al usuario crear tipos personalizados:

```
function Persona(nombreP, apellidosP, edadP, colorOjosP) {
    this.nombre = nombreP;
    this.apellidos = apellidosP;
    this.edad = edadP;
    this.colorOjos = colorOjosP;
    this.nombreCompleto = function () {
        return `${this.nombre} ${this.apellidos}`
    };
}

const oMiPadre = new Persona("Juan", "Sánchez", 70, "marrón");
const oMiMadre = new Persona("María", "Campos", 72, "verde");
```

ES6 introduce una nueva forma de declarar tipos de objetos (clases) más similar a la de otros lenguajes orientados a objetos:

```
class Persona {
    constructor(nombreP, apellidosP, edadP, colorOjosP) {
        this.nombre = nombreP;
        this.apellidos = apellidosP;
        this.edad = edadP;
        this.colorOjos = colorOjosP;
    }

    nombreCompleto() {
        return `${this.nombre} ${this.apellidos}`;
    }
}

const oMiPadre = new Persona("Juan", "Sánchez", 70, "marrón");
const oMiMadre = new Persona("María", "Campos", 72, "verde");
```

Por mantenibilidad **utiliza siempre la sintáxis ES6**.

## Actividad 1. Objetos de tipo Coche.

Realiza los siguientes apartados tomando como referencia el artículo JavaScript Classes de W3Schools.

**a) Creación inicial de clase e instancias.**
a.1) Crea una clase Coche con los atributos fabricante, modelo, marchas y marchaActual y con el método cambiarMarcha para cambiar la marchaActual. Ten en cuenta:

- Para instanciar un objecto Coche sólo es necesario indicar el fabricante y el modelo.
- Las marchas pueden ser P (Parado), 1, 2, 3, 4, 5, 6 y R (marcha atrás).
- Al inicializar el objeto la marcha actual será parado.
- En el método cambiarMarcha, si la marcha a la que se pretende cambiar no existe se debe lanzar una excepción.

a.2) Crea dos instancias de la clase Coche: un Tesla ModelS y un Mazda 3i. Cambia la marcha del Tesla a primera y la del Mazda a marcha atrás.

a.3) Muestra por consola el fabricante, el modelo y la marcha en la que se encuentran ambos coches.

**b) Métodos get y set.**
Añade métodos get y set a la clase para cada uno de sus atributos.

**c) Métodos estáticos.**
Añade un método estático a la clase Coche para determinar si dos coches son iguales. Dos coches se consideran iguales si tienen el mismo fabricante y modelo.

**d) Herencia.**
d.1) Crea una clase padre Vehículo con un único atributo pasajeros. Añade un método get y set para el atributo.

d.2) Haz que Coche herede de Vehículo.

d.3) Crea un nuevo Coche, añádele 3 pasajeros y muestra el número de pasajeros por consola.

**f) Representación textual de objetos.**
Redefine el método toString para obtener una cadena con el siguiente formato al imprimir un objeto Coche: fabricante :::: modelo :::: pasajeros

## Actividad 2. Objetos de tipo Punto.

Queremos crear objetos que representen puntos en un plano, se necesita:

a) Que tenga dos propiedades x e y que servirán para representar las coordenadas del punto. Al instanciar un punto se le pasarán dos números, si lo que recibe en alguna coordenada no es un número la coordenada se colocará a 0.

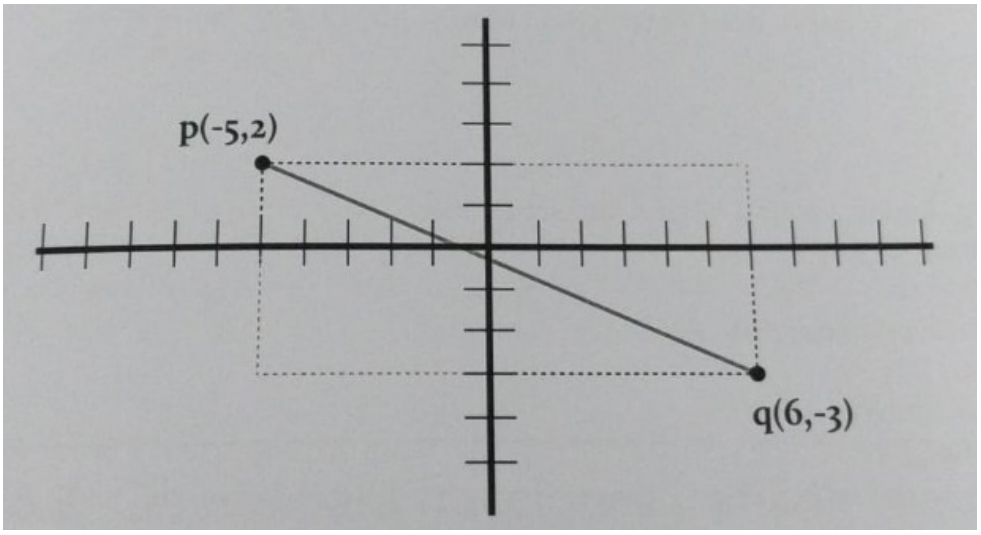
b) Añade un método *cambiar* al que se le pasan dos números y permite modificar las coordenadas del punto.

c) Añade un método *copia* que devuelva una copia del objeto.

d) Añade un método *iguales* que recibe un segundo punto y nos dice si ambos puntos son iguales.

e) Añade un método *suma* que reciba un segundo punto y devuelva un tercer punto resultado de sumar las coordenadas de los dos anteriores.

f) Añade un método *obtenerDistancia* que reciba un segundo punto y devuelva la distancia entre ambos puntos. Para hacer esta operación ten en cuenta:




Para calcular la distancia se aplicará el Teorema de Pitágoras, la distancia se obtiene con la fórmula:

$$\sqrt{distHoriz^2 + distVert^2} = \sqrt{abs(6-(-5))^2 + abs(-3-2)^2} = \sqrt{11^2+5^2} = \sqrt{146} = 12{,}08$$

g) Redefine el método toString para que se imprima la cadena (-5, 2) cuando se necesite una representación textual del objeto.

h) Crea dos objetos Punto que representen los puntos p y q mostrados en la imagen. Prueba que los métodos funcionen correctamente.

## Actividad 3. Objetos de tipo Ordenador y Portátil.

Queremos crear objetos que representen ordenadores, sus atributos son:

- Marca, un texto.
- Modelo, un texto.
- Memoria RAM, un número que indica los Gb de capacidad.
- Capacidad de disco duro, un número que indica los Gb de capacidad.
- Pulgadas de pantalla, un número que indica las pulgadas de la pantalla.

Se necesitan los siguientes métodos:

- toString, para obtener una representación textual de los ordenadores.

Se debe tener en cuenta que por defecto se tomarán los valores 17 pulgadas, 512Gb de disco duro y 4Gb de RAM.

Crea otro tipo de objeto que represente ordenadores portátiles que tendrán todos los atributos y métodos de los ordenadores y añaden un atributo autonomía, un número que indica el número de horas. Por defecto la autonomía será de 4 horas, 12 pulgadas y 256Gb de disco duro.

## Arrays

Los arrays son una de las características más potentes del lenguaje, se utilizan en todos los proyectos que impliquen la gestión de colecciones de datos.

Los arrays en JavaScript tienen un comportamiento particular. Ten en cuenta las siguientes consideraciones:

- Representan una colección de datos ordenada por índices numéricos.
- Los datos almacenados pueden ser heterogéneos, no tienen por qué ser todos del mismo tipo.
- IMPORTANTE: Si se usa un índice mayor de la longitud del array, éste crecerá automáticamente hasta el índice asignado rellenando los datos intermedios con el valor `undefined`.

### Actividad 1. Introducción.

Realiza las siguientes actividades de introducción para la declaración, acceso y manipulación de arrays:

**a) Arrays unidimensionales**: Declara un array unidimensional e inicialízalo con los valores 100, 200, 300, 400 y 500. A continuación imprímelo por consola con el siguiente formato:

*El array tiene en la posición 0 el valor 1.*
*El array tiene en la posición 1 el valor 2.*
...

Realiza la operación anterior utilizando un bucle `forEach` (MDN) y un bucle `for...of` (MDN)

**b) Array bidimensionales**: Genera un array bidimensional de 5 x 5 posiciones e inicialízalo con los siguientes valores:

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
| 1 | 2 | 3 | 4 | 5 |
| 1 | 2 | 3 | 4 | 5 |
| 1 | 2 | 3 | 4 | 5 |
| 1 | 2 | 3 | 4 | 5 |

**c) Arrays paralelos**: El usar arrays para almacenar información, facilita una gran versatilidad en las aplicaciones, a la hora de realizar búsquedas de elementos dentro del array. Pero en algunos casos, podría ser muy útil hacer las búsquedas en varios arrays a la vez, de tal forma que podamos almacenar diferentes tipos de información, en arrays que estén sincronizados.

Cuando tenemos dos o más arrays, que utilizan el mismo índice para referirse a términos homólogos, se denominan arrays paralelos.

Por ejemplo consideremos los siguientes arrays:
Usando estos tres arrays de forma sincronizada, y haciendo referencia a un mismo índice (por ejemplo índice=3), podríamos saber que Benjamín es profesor de Redes y tiene 26 alumnos en clase.

```
let profesores = ["Cristina","Catalina","Vieites","Benjamin"];
let asignaturas = ["Seguridad","Bases de Datos","Sistemas Informáticos","Redes"];
let alumnos = [24,17,28,26];
```

Veamos un ejemplo de código que recorrería todos los profesores que tengamos en el array imprimiendo la información sobre la asignatura que imparten y cuantos alumnos tienen en clase:

```
let profesores = ["Cristina","Catalina","Vieites","Benjamin"];
let asignaturas=["Seguridad","Bases de Datos","Sistemas Informáticos","Redes"];
let alumnos=[24,17,28,26];

function imprimeDatos() {
    for (i=0;i<profesores.length;i++) {
        console.log(`${profesores[i]} del módulo de ${asignaturas[i]}, tiene ${alumnos[i]} alumnos en clase.`);
    }
}

imprimeDatos();
```

Para que los arrays paralelos sean homogéneos, éstos tendrán que tener la misma longitud, ya que de esta forma se mantendrá la consistencia de la estructura lógica creada.
Entre las ventajas del uso de arrays paralelos tenemos:

- Se pueden usar en lenguajes que soporten solamente arrays, como tipos primitivos y no objetos o registros (como puede ser JavaScript).
- Son fáciles de entender y utilizar.
- Pueden ahorrar una gran cantidad de espacio, en algunos casos evitando complicaciones de sincronización.
- El recorrido secuencial de cada posición del array, es extremadamente rápido en las máquinas actuales.

Implementa una aplicación que utilizando arrays paralelos permita al usuario introducir el número de un mes y le devuelva el nombre del mes y si se trata de un mes lectivo.

### Actividad 2. Métodos nativos.

Conocer los métodos nativos que ofrece JavaScript para manipular arrays es fundamental para dominar esta estructura de datos. Descarga esta aplicación e implementa los ejercicios propuestos.

Para implementar las funciones especificadas puedes consultar la JavaScript Array Reference de W3Schools.

### Actividad 3. Loteria primitiva.

Implementa una aplicación que muestre 10 combinaciones para jugar a la lotería primitiva. Una combinación tiene 6 números del 1 al 49. Los números no se pueden repetir en una combinación.

Para ello:

- Genera un array de 49 elementos que contengan números del 1 al 49.
- Mezcla los elementos del array.
- Extrae un array de los 6 primeros elementos para obtener una combinación.

### Actividad 4. Tablero del juego Caracol preso.

Implementa el tablero del juego *El Caracol Preso*. Este juego consiste en ayudar a un caracol a escapar de la parcela electrificada en la que se encuentra encerrado. En esta actividad no tienes que implementar el juego completo sino generar el tablero que usará.

Este es un tablero de 10x10 posiciones:

```
O O O O O O O O O O
O X X X X X X X X O
O X X X X X X X X O
O X X X X X X X X O
O X X X X o X X X O
O X X X X X X X X O
O X X X X X X X X O
O X X X X X X X X O
O X X X X X X X X O
O O O O I I O O O O
```

Ubica adecuadamente los elementos del juego en las posiciones que correspondan:

- Carácter “O”. Simboliza la valla electrificada, se sitúa en la periferia de la parcela.
- Carácter “X”. Simboliza todas las posiciones a las que se puede mover el caracol. Se ubica en todas las posiciones internas de la parcela.
- Carácter “o”. Simboliza el caracol. Debe situarse en una posición aleatoria dentro de las posiciones internas de la parcela.
- Carácter “II”. Simboliza la salida. Debe colocarse en una posición aleatoria de la fila inicial o final exceptuando las esquinas.

### Actividad 5. Baraja de cartas.

A partir de este código, realiza las siguientes tareas:

Apartado A. Clase Carta.

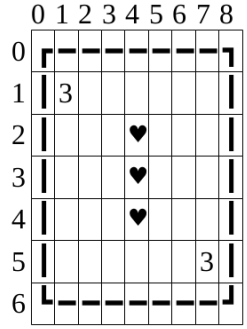
Implementa una clase **`Carta`** que represente las cartas de una baraja inglesa.

La clase tiene 3 atributos:

1. **`sSimbolo`**. Sus posibles valores son `A`, `2`, `3`, `4`, `5`, `6`, `7`, `J`, `Q` y `K`.

2. **`sPalo`**. Sus posibles valores son `diamantes`, `picas`, `corazones` y `treboles`.

3. **`aCarta`**. Array de 9 x 7 posiciones que representa una carta. Ejemplo para el `3` de `corazones`:




Cada posición almacena el código Unicode asociado al caracter. Por ejemplo, en las posiciones centrales se guarda el código `0x2665` que representa el caracter ♥

Añade además:

a) Un **`constructor`** que reciba el símbolo y el palo de la carta.

b) Métodos **`get`** y **`set`** para los atributos. Si se reciben un símbolo o palo no válido, se lanzará una excepción.

c) 4 métodos que inicializan y rellenan el array `aCarta`:

   1. **`inicializarCarta`**. Crea un nuevo array bidimensional de tamaño 9x7 e inicializa todas sus posiciones con un espacio en blanco.

   2. **`incluirLineas`**. Añade al array las esquinas y los bordes.

   3. **`incluirSimbolo`**. Añade al array el símbolo de la carta.

   4. **`incluirPalo`**. Añade al array el palo de la carta.

d) El método **`toString`** que devuelve una representación textual del array.

La siguiente tabla muestra donde colocar el palo en función del símbolo de la carta.

| Símbolo | Posición del palo |
|---|---|
| A, J, Q o K | [3, 4] |
| 2 | [3, 2] [3, 6] |
| 3 | [3, 2] [3, 4] [3, 6] |
| 4 | [2, 2] [2, 6] [4, 2] [4, 6] |
| 5 | [2, 2] [2, 6] [3, 4] [4, 2] [4, 6] |

| | |
|---|---|
| 6 | [2, 2] [2, 4] [2, 6] [4, 2] [4, 4] [4, 6] |
| 7 | [2, 2] [2, 4] [2, 6] [3, 4] [4, 2] [4, 4] [4, 6] |

Apartado B. Montones

a) Crea una baraja de cartas inglesa utilizando la clase `Carta`. La baraja tiene 40 cartas, 10 de cada palo.

b) Baraja las cartas utilizando el algoritmo de Fisher-Yates.

**Arrays. Métodos *forEach*, *map*, *filter*, *find*, *findIndex*, *reduce*, some, every, sort, findLast, findLastIndex, etcétera**

Dado el siguiente array de objetos literales, implementa las actividades que se proponen a continuación:

```
const coches = [
    {marca: 'BMW',  modelo: 'Serie 3', annio: 2012, precio: 30000, puertas: 4, color: 'Blanco', transmision: 'automatico'},
    {marca: 'Audi', modelo: 'A4',  annio: 2018, precio: 40000, puertas: 4, color: 'Negro', transmision: 'automatico'},
    {marca: 'Ford', modelo: 'Mustang',  annio: 2015, precio: 20000, puertas: 2, color: 'Blanco', transmision: 'automatico'},
    {marca: 'Audi', modelo: 'A6',  annio: 2010, precio: 35000, puertas: 4, color: 'Negro', transmision: 'automatico'},
    {marca: 'BMW', modelo: 'Serie 5',  annio: 2016, precio: 70000, puertas: 4, color: 'Rojo', transmision: 'automatico'},
    {marca: 'Mercedes Benz', modelo: 'Clase C',  annio: 2015, precio: 25000, puertas: 4, color: 'Blanco', transmision: 'automatico'},
    {marca: 'Chevrolet', modelo: 'Camaro',  annio: 2018, precio: 60000, puertas: 2, color: 'Rojo', transmision: 'manual'},
    {marca: 'Ford', modelo: 'Mustang',  annio: 2019, precio: 80000, puertas: 2, color: 'Rojo', transmision: 'manual'},
    {marca: 'Dodge', modelo: 'Challenger',  annio: 2017, precio: 40000, puertas: 4, color: 'Blanco', transmision: 'automatico'},
    {marca: 'Audi', modelo: 'A3',  annio: 2017, precio: 55000, puertas: 2, color: 'Negro', transmision: 'manual'},
    {marca: 'Dodge', modelo: 'Challenger',  annio: 2012, precio: 25000, puertas: 2, color: 'Rojo', transmision: 'manual'},
    {marca: 'Mercedes Benz', modelo: 'Clase C',  annio: 2018, precio: 45000, puertas: 4, color: 'Azul', transmision: 'automatico'},
    {marca: 'BMW', modelo: 'Serie 5',  annio: 2019, precio: 90000, puertas: 4, color: 'Blanco', transmision: 'automatico'},
    {marca: 'Ford', modelo: 'Mustang',  annio: 2017, precio: 60000, puertas: 2, color: 'Negro', transmision: 'manual'},
    {marca: 'Dodge', modelo: 'Challenger',  annio: 2015, precio: 35000, puertas: 2, color: 'Azul', transmision: 'automatico'},
    {marca: 'BMW', modelo: 'Serie 3',  annio: 2018, precio: 50000, puertas: 4, color: 'Blanco', transmision: 'automatico'},
    {marca: 'BMW', modelo: 'Serie 5',  annio: 2017, precio: 80000, puertas: 4, color: 'Negro', transmision: 'automatico'},
    {marca: 'Mercedes Benz', modelo: 'Clase C',  annio: 2018, precio: 40000, puertas: 4, color: 'Blanco', transmision: 'automatico'},
    {marca: 'Audi', modelo: 'A4',  annio: 2016, precio: 30000, puertas: 4, color: 'Azul', transmision: 'automatico'}
];
```

**Actividad 1.** Imprime por consola la marca y el modelo de todos los coches del array.

**Actividad 2.** Devuelve el primer coche de color rojo que aparece en el array.

**Actividad 3.** Devuelve true si todos los coches tienen menos de 10 años.

**Actividad 4.** Devuelve true si algún coche es más caro de 70000 euros o es "Mercedes Benz" y cuesta menos de 40000 euros

**Actividad 5.** Devuelve el índice del primer coche que cumple que es un Dodge con 2 puertas.

**Actividad 6.** Devuelve un array que contenga cadenas de caracteres mostrando el texto: “N.º puertas: X, color: Y” para todos los coches del array "coches".

**Actividad 7.** Devuelve un array con todos los coches que tienen transmisión automática, cuestan menos de 60000 euros y de color negro e imprime por pantalla todos los datos de los coches que cumplen las condiciones.

**Actividad 8.** Imprime por consola la marca, el año y precio de todos los coches cuya marca contiene una "o" (mayúscula o minúscula), ordenados por el año (menor a mayor).