ALICIA VÁZQUEZ

# JavaScript: Programación funcional y Arrays

Alicia Vázquez
@aliciaFPInf

# Índice



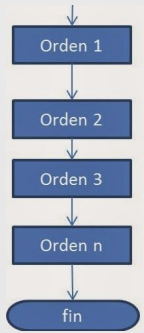
# 01

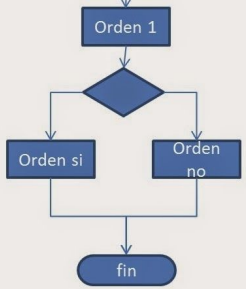
# Funciones

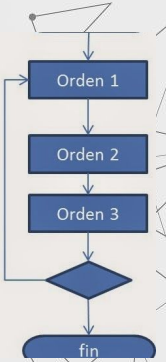
Declaración, expresión.

# Paradigma de la programación estructurada.

- En la programación estructurada un problema (programa) se divide en subproblemas más simples.
- El uso de **funciones** (subrutinas) permite reutilizar código.
- El uso de funciones permite dar **sentido semántico** al código, puesto que este debe explicarse solo eligiendo un buen nombre de función.
- La idea fundamental es que los programas son un **conjunto de bloques** que encajan entre sí sobre la premisa de que cada bloque tiene un solo punto de entrada y un solo punto de salida. La salida de uno será la entrada del siguiente.



# FUNCTION

Se declaran con **function** y se les pasan los *parámetros* entre paréntesis. La función puede devolver un valor usando *return* (si no tiene return es como si devolviera undefined).

```
// Declaración de la función y paso de parámetros
function nombreDeLaFuncion(parametro1, parametro2, …) {
   return nombreVariable;
}

// Ejecución
var resultado = nombreDeLaFuncion(parametro1, parametro2, …);
```

Puede usarse una función antes de haberla declarado por el comportamiento de Javascript llamado **hoisting**: el navegador primero carga todas las funciones y mueve las declaraciones de las variables al principio y luego ejecuta el código.

# Parámetros y argumentos (enlace)

A las funciones se les pueden pasar o no parámetros, que no son más que variables que existirán **sólo dentro** de dicha función, con el valor pasado desde la ejecución.

Los argumentos son los valores que pasamos al invocar la función.

**`function nombreFuncion (<parametros>){ <instrucciones> }`**

**`nombreFuncion(<argumentos>)`**

```
// Declaración de la función y paso de parametros
function tablaMultiplicar(tabla, hasta) {
   for (i = 0; i <= hasta; i++) console.log(tabla, "x", i, "=", tabla * i);
}

// Ejecución
tablaMultiplicar(2, 10); // Tabla del 2
tablaMultiplicar(5, 10); // Tabla del 5
```

# RETURN

Si es una variable global no hace falta pasarla, pero hay que saber que se verá **afecta** por lo que le hagamos en la función, esto es una práctica poco recomendable, perdemos el control de lo que ocurra con ella.

```
//Variable global
var resultado = 0;

function multiplica (a,b){
   resultado = a*b;
}

//fuera de la función
multiplica(3,7);
console.log("El resultado de es ", resultado);
```

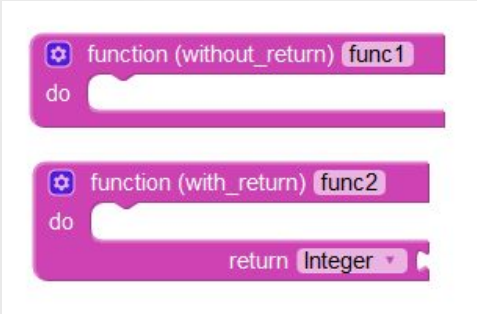
```
function multiplica (a,b){
   return a*b;
}

//fuera de la función
resultado=multiplica(8,9);
console.log("Return: El resultado de es ",
resultado);
```

# FUNCIONES, PROCEDIMIENTOS Y MÉTODOS.

Una **función** recibe o no parámetros pero **siempre devuelve un resultado**.
Llamaremos **procedimientos** a las funciones que **no devuelven nada**, no hay return.

```
// Declaración de un procedimiento y paso de parámetros
function nombreDelProcedimiento(parametro1, parametro2, …) {
}
// Ejecución
nombreDelProcedimiento(parametro1, parametro2, …);
```




En la **POO** los **métodos** son aquellas funciones y/o procedimientos que pertenecen a una clase, serán los métodos propios de cada clase.

# PARÁMETROS

Si se llama una función con menos parámetros de los declarados, el valor de los parámetros no pasados será *undefined*:

```
function potencia(base, exponente) {
   console.log(base);              // muestra 4
   console.log(exponente);         // muestra undefined
   let valor=1;
   for (let i=1; i<=exponente; i++) {
     valor=valor*base;
   }
   return valor;
}
potencia(4);     // devolverá 1 ya que no se ejecuta el for
```

Podemos dar un **valor por defecto** a los parámetros por si no los pasan asignándoles el valor al definirlos:

```
function potencia(base, exponente= 2) {
   console.log(base);              // muestra 4
   console.log(exponente);         // muestra 2 la primera vez y 5 la segunda
   let valor=1;
   for (let i=1; i<=exponente; i++) {
     valor=valor*base;
   }
   return valor;
}
console.log(potencia( 4));          // mostrará 16 (4^2)
console.log(potencia( 4,5));        // mostrará 1024 (4^5)
```

# REGLA DE LOS PARÁMETROS (enlace)

- No especificamos el tipo de dato → JS es no tipado.
- No se verifican los tipos de argumentos → JS es no tipado.
- No se comprueban el número de argumentos recibidos.
  - Sí **faltan** y se necesitan saltará una **excepción**, o devolverá *null*, *undefined*; dependerá de las instrucciones. → El programador deberá controlar esto.
  - Si hay **exceso** de argumentos todos pasarán al objeto **arguments** que está en todas las funciones.

```
//FALTA DE PARÁMETROS
function suma (a,b){
  //if(b === undefined){b=0;
}}
  return a + b;
var sum = suma(4);
alert(sum); //NaN
```

```
//EXCESO DE PARÁMETROS
function valores (){
    alert(arguments.length); //9
    for(let x of arguments){
        console.log(typeof(x),x); //number 2 number 4 ...
     }
}
       valores(2,4,6,8,'nueve','29/02/2024',[5,3,1]);
```

**OJO!!!!! En JS no hay sobrecarga de operadores (funciones)**

# Ejercicios funciones

1. Crea una función llamada **saludar** que tenga un parámetro obligatorio, *nombre*, y un parámetro opcional, *saludo*, que por defecto sea "Hola". La función debe devolver un saludo personalizado utilizando el nombre proporcionado y, el saludo adicional.
2. Crea una función llamada **calcular** que reciba dos números como parámetros obligatorios, *num1* y *num2*, y un parámetro opcional, *operacion*, que represente la operación matemática a realizar (por defecto, suma). La función debe devolver el resultado de la operación especificada entre los dos números.
3. Crea una función llamada **calcularPromedio** que no tenga definido el número de argumentos que representen calificaciones y que calcule su promedio. La función debe ignorar cualquier argumento adicional que se proporcione.
4. Crea una función llamada **concatenarNombres** que tome un número indefinido de argumentos que representen nombres y los concatene en una sola cadena separada por comas. La función debe ignorar cualquier argumento adicional que se proporcione.

# 02

# Funciones anónimas

Declaración, expresión.

# FUNCIÓN ANÓNIMA (enlace)

## ¿Qué es una función anónima?

Es simplemente una función que **no tiene nombre**. En lugar de crearla con un nombre fijo, la defines justo donde la vas a usar.

Usamos funciones con nombre cuando queremos reutilizar el código. Usamos funciones anónimas cuando queremos aislar el código para que no moleste a nadie más ni deje rastro en la memoria.

## ¿Cómo se usa?

• Guardándola en una variable: Como la función no tiene nombre, usas el nombre de la variable para llamarla.

• Pasándola como argumento: Se la entregas a otra función para que esta última la use.

```
 var multiply = function(x, y) {
  return x * y;
};
console.log(multiply(2,3));
```

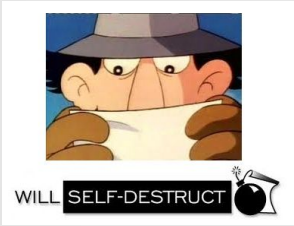
```
my_element.addEventListener("click", function (e) {
  console.log(this.className);
  console.log(e.currentTarget === this);
});
```

# ¿Para qué sirven?

• **Encapsular**: Sirven para agrupar código que solo necesitas en un lugar específico, sin "ensuciar" el resto de tu programa con nombres de funciones que no volverás a usar.

• **Ejecución inmediata (Immediately Invoked Function Expression IIFE):** Si necesitas que un bloque de código se ejecute en el mismo instante en que se declara, lo encierras entre paréntesis y lo activas de inmediato.

• **No deja rastro**: Si le das un nombre a una función, ese nombre ocupa un lugar en la "agenda" (memoria) del navegador, aunque nunca más vuelvas a llamar a esa función. Si es anónima y autoejecutada, es como un mensaje que se autodestruye tras leerse.



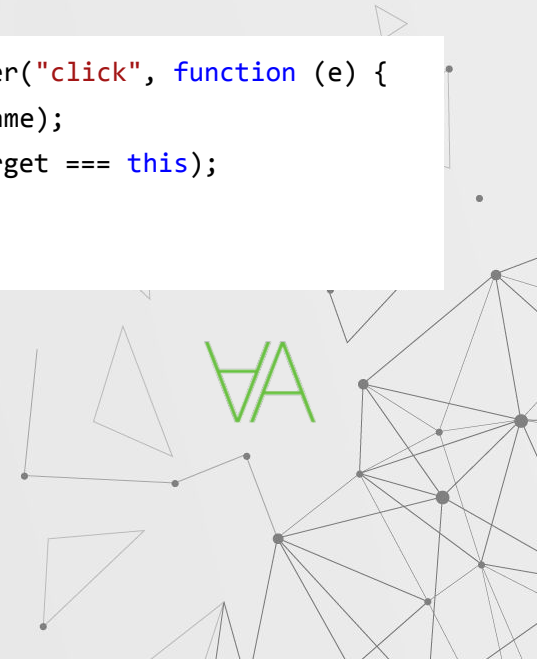

```
(function () {
   console.log("Ejecutando función anónima");
   return "Genial!!";
})();
//Así sería un ejemplo con un parámetro:
(function (edad) {
  console.log("Ejecutando función anónima");
  return "Genial!!. Tu edad " + edad;
})(30);
```

# Callbacks: "Llámame luego" (enlace)

Un **callback** es una función que pasas como "mensaje" a otra. Se usa mucho en eventos: por ejemplo, le dices a un botón que ejecute una función específica solo cuando alguien haga clic en él.

```
document.form1.colorButton.onclick = function(event){
  setBGColor();
};
```

```
my_element.addEventListener("click", function (e) {
  console.log(this.className);
  console.log(e.currentTarget === this);
});
```



# FUNCIÓN FLECHA - Arrow function (enlace)

La sintaxis de funciones en forma de flecha se ha creado simplemente para ahorrar líneas de código y ahorrar tiempo. Las **Arrow function** son funciones anónimas que usan la sintaxis de flecha (**=>**). Esta sintaxis está muy extendida en JS y veréis que la mayoría del código contiene arrow function.

Usaremos esta sintaxis en funciones muy específicas y pequeñas, que puedan leerse en una sola línea.

En otros lenguajes de programación son denominadas: **Funciones Lambda**

```
//Función normal
function pulgadaACm(pulgadas) {
    return pulgadas * 2.54;
};
//Arrow function
const pulgadaACm = (pulgadas) => pulgadas * 2.54;
```

# FUNCIÓN FLECHA - Arrow function (enlace)

Vemos paso a paso cómo transformar una declaración de función clásica o normal a una **Arrow Function.**

```
//tenemos una función normal
function pulgadaACm(pulgadas) {
    return pulgadas * 2.54;
};
//la transformamos a anónima como expresión
const pulgadaACm = function (pulgadas) {
    return pulgadas * 2.54;
};
//le quitamos la palabra reservada function y le ponemos una flecha
const pulgadaACm = (pulgadas) => {
    return pulgadas * 2.54;
};
//ahora transformamos el return en un implícito
const pulgadaACm = (pulgadas) => pulgadas * 2.54;
//Si sólo hay un parámetro podemos ahorrarnos el paréntesis también
const pulgadaACm = pulgadas => pulgadas * 2.54;
```

# Ejercicios funciones

De los ejercicios anteriores puede reescribir algo de código usando funciones flecha o anónimas?

# 03

# Arrays

# ARRAY

- El Array es un objeto por definición, está compuesta por una lista de elementos. Accedemos a la lista mediante un **índice**, empezando en **0**.

```
//Array (object)
let paises = ['España', 'Mexico', 'Colombia', 'Argentina'];
      console.log(
         typeof paises,
         paises[0],
         typeof(paises[0])
   );
let empty = []; // Array vacío (0 elementos)
console.log(empty.length); // 0

let mixto = ["a", 5, true]; // Array mixto (string, number, boolean)
```

# ARRAY

Herencia de otros lenguajes también podemos dar de alta usando la palabra **new**, pero no es aconsejable

```
// Forma tradicional (no se suele usar en Javascript)
const arreglo = new Array("a", "b", "c");    // Array con 3 elementos
const arreglo = new Array(3);                // Array vacío de tamaño 3 [undefined,undefined,undefined]

// Mediante literales (notación preferida)
const arreglo = ["a", "b", "c"];  // Array con 3 elementos
const arreglo = [];               // Array vacío (0 elementos)
const arreglo = ["a", 5, true];   // Array mixto (String, Number, Boolean)


//const arreglo = new Array(3); no confundir con const arreglo = [3]; que en este caso es un array de un elemento con valor 3. OJO CON NÚMERO DE ARGUMENTOS Q SE ENVIA AL CONSTRUCTOR!!!
```

# ARRAY acceso

Para acceder a los elementos de un array podemos hacerlo de varias maneras. Al ser una agrupación de elementos **SIEMPRE** necesitaremos un **bucle** si queremos recorrerlo.

| | |
|---|---|
| `arreglo[pos]` | Operador que devuelve (o modifica) el elemento número pos del array. |
| `arreglo.at(pos)` | Método que devuelve el elemento en la posición pos. Números negativos en orden inverso. |
| `arreglo.length` | Propiedad con el número de elementos que contiene el array. (`const lastItem = letters.length - 1;` ) |

# ARRAY acceso

```
const arreglo = ["a", "b", "c"];
arreglo.length;   // 3
arreglo[0];       // 'a'
arreglo[2];       // 'c'
arreglo[5];       // undefined

const arreglo =  ["a", "b", "c"];
arreglo[1] = "Z";  // Devuelve "Z" y modifica arreglo a ["a", "Z", "c"]
arreglo[3] = "D";  // Devuelve "D" y modifica arreglo a ["a", "Z", "c", "D"]
arreglo[5] = "A";  // Devuelve "A" y modifica arreglo a ["a", "Z", "c", "D", undefined, "A"]
```

# ARRAY añadir/eliminar

Existen métodos que **modifican** el array directamente. Son métodos propios de la clase Array.

| | |
|---|---|
| `number = arreglo.push(e1, e2, e3...)` | Añade uno o varios elementos **al final** del array. Devuelve el tamaño del array. |
| `elemento = arreglo.pop()` | Elimina el **último** elemento del array. Devuelve el elemento eliminado. |
| `number = arreglo.unshift(e1, e2, e3...)` | Añade uno o varios elementos **al inicio** del array. Devuelve el tamaño del array. |
| `elemento = arreglo.shift()` | Elimina el **primer** elemento del array. Devuelve el elemento eliminado. |
| `arreglo.splice(posicion, cantidad, (opcional) nuevo)` | El "multiusos" (borra, añade o reemplaza en cualquier posición). |

# ARRAY añadir/eliminar

```
const array = ["a", "b", "c"]; // Array inicial

array.push("d"); // Devuelve 4. Ahora array = ['a', 'b', 'c', 'd']
array.pop(); // Devuelve 'd'. Ahora array = ['a', 'b', 'c']

array.unshift("Z"); // Devuelve 4. Ahora array = ['Z', 'a', 'b', 'c']
array.shift(); // Devuelve 'Z'. Ahora array = ['a', 'b', 'c']

// --- Operando con SPLICE (el cirujano) ---
// Eliminar: Desde la posición 1, quita 1 elemento
array.splice(1, 1); // Devuelve ['b']. Ahora array = ['a', 'c']

// Insertar: En la posición 1, no quites nada e inserta 'x' e 'y'
array.splice(1, 0, "x", "y"); // Devuelve []. Ahora array = ['a', 'x', 'y', 'c']

// Reemplazar: En la posición 3, quita 1 y pon 'Z'
array.splice(3, 1, "Z");// Devuelve ['c']. Ahora array = ['a', 'x', 'y', 'Z']
```

# ARRAY metodos

El Array tiene sus propios métodos que nos permite manipularlo, modificando el original.

| | |
|---|---|
| `sort()` | Ordena alfabéticamente el Array [0..9 →A..Z→a..z]. |
| `reverse()` | Ordena al revés, [z..a→Z..A → 0..9] |

```
const fruits = ["Banana", "1Orange", "Apple", "mango"];
//Ordenamos
fruits.sort();
printArray(fruits, "Ordenado: ");

// Orden al revés
fruits.reverse();
printArray(fruits, "Al revés: ");
```

# ARRAY bucle de recorrido FOR..OF / FOR.. IN

Para recorrer un array podemos usar el **FOR.. OF** cogiendo directamente el elemento sin necesidad de un iterador

`for ( let element of array)` ⟶ *mientras haya elementos dentro del arreglo*

Pero también podemos seguir usando **FOR .. IN** (como el for más simple) o **WHILE**, tomando la variable como el iterador.

Es más interesante el **FOR.. OF**, pero si necesitas tener control del iterador, entonces mejor **FOR .. IN** o **WHILE**

```
//JOIN
function printArrayOf(array, text) {
    for (let x of array) {
        text += x + ",";
    }
    console.log("FOR OF: " + text);
}
```

```
//JOIN
function printArrayIn(array, text) {
    for (let i in array) {
        text += array[i] + ",";
    }
    console.log("FOR IN: " + text);
}
```

# Convertir a ARRAY, unir a String

Podemos pasar de String o NodeList a un Array y de un Array a un String. Estas funcionalidades son muy interesantes para hacer según qué manipulaciones, puesto que los métodos que ofrece la clase array son más completos.

| | |
|---|---|
| `Array.from(String o NodeList del DOM , tipo de Dato)` | Devuelve, si es posible, un array con los elementos separados. En caso contrario nos devolverá un error. |
| `[...String]`<br>`[...NodeList]` | |
| `string.split(separador)` | Devuelve un array, con el número de elementos que salga según el separador indicado. |
| `arreglo.join(separador);` | Devuelve un string con todos los elementos del array separados por el separador. |

# Convertir a ARRAY, unir a String

```
const text = "12345";                                  // es un String
const arreglo = Array.from(text);                      // ["1", "2", "3", "4", "5"]
const arreglo = [...text];                             // ["1", "2", "3", "4", "5"] son equivalentes.

const divs = document.querySelectorAll("div"); // son NodeList del DOM
const elementos = Array.from(divs);                    // [div, div, div]
const elementos = [...divs];                           // [div, div, div] son equivalentes.

const text = "12345";
const numbers = Array.from(text, Number);                                  // [1, 2, 3, 4, 5]

const arreglo = ["a", "b", "c"];
// Une elementos del array por el separador indicado
arreglo.join("->");        // Devuelve string 'a->b->c'
arreglo.join(".");         // Devuelve string 'a.b.c'

// Separa elementos del string por el separador indicado
"a.b.c".split(".");        // Devuelve ['a', 'b', 'c']
"5-4-3-2-1".split("-");    // Devuelve ['5', '4', '3', '2', '1']
```

# Ejercicios Array:

**Problema 1:**
Dada una lista de números, escribe una función en JavaScript que devuelva la suma de todos los números pares en la lista. La función deberá iterar sobre cada número en la lista, comprobar si el número es par y, si es así, añadirlo a la suma total.

```
sumarPares([2,4,5,6,7,8,3,1,10,4])
```

**Problema 2:**
Recibimos un array con los datos del alumno, pero debemos ordenarlo. En lugar de tener al final el nombre del alumno lo queremos al principio, antes del apellido y queremos añadir al final la nota media. De tal manera que:

```
ordenDatos(["Rodriguez", "8", 9, '5',4, 'Clara']) → Clara,Rodriguez,8,9,5,4,6.5
```

**Problema 3:**
Partiendo de un array con palabras aleatorias, pero nuestro programa no acepta palabras que empiezan por “Z”. Elimina estas palabras del array y devuelve el nuevo sin estas palabras.

```
filterWords(["Bob", "Alex", "Zoello"]);
```

```
filterWords(["León", "Zebra", "Gacela"]);
```

```
filterWords(["Mercedes", "Bmw", "Audi", "Porche"]);
```

# 04

# Métodos y Funciones del Array

# ARRAY

Hemos visto **métodos** básicos de un Array para poder manipularlo y **modificarlo**.

- array.push()
- array.pop(),
- array.shift(),
- array.unshift()
- array.reverse()
- array.sort()

Otros que **no lo modifican**, sino que devuelven un nuevo array.

- array.concat()
- array.join()
- array.at()
- array.with()

Veamos algunos más.

# .forEach()

El método **forEach** en JavaScript permite recorrer un array y aplicar en cada uno de sus elementos una acción en particular a través de una función.

**NOTA**: No necesitamos pasarle el arrays, puesto que es un MÉTODO de la clase ARRAY

```
array.forEach(function(elementoActual, indice, array));
//elementoActual: es el valor del elemento actual del array.
//índice: es el índice del elemento actual del array (opcional).
//array: es el array que se está recorriendo (opcional).
```

```
const vocales = ["a", "e", "i", "o", "u"];

vocales.forEach(function (vocal, indice, arr) {
      console.log(`El valor del elemento en la posición ${indice} es ${vocal} del array que estamos recorriendo que es ${arr}`);
});
```

```
El valor del elemento en la posición 0 es a del array que estamos recorriendo que es a,e,i,o,u
El valor del elemento en la posición 1 es e del array que estamos recorriendo que es a,e,i,o,u
El valor del elemento en la posición 2 es i del array que estamos recorriendo que es a,e,i,o,u
El valor del elemento en la posición 3 es o del array que estamos recorriendo que es a,e,i,o,u
El valor del elemento en la posición 4 es u del array que estamos recorriendo que es a,e,i,o,u
```

# .forEach() / filter()

```
// Array de precios
const precios = [ 25,150,80,30,200];

// Array auxiliar para almacenar los precios filtrados
const preciosFiltrados = [];

// Filtrar precios con precio mayor a 100 utilizando forEach
precios.forEach((precio) => {
  if (precio > 100) {
    preciosFiltrados.push(precio);
  }
});

// Imprimir los precios filtrados por consola
console.log(preciosFiltrados);
/*
[150,200]
*/
```

```
const precios = [25, 150, 80, 30, 200];
// .filter() ya crea el array por ti y te lo devuelve
const preciosFiltrados = precios.filter(precio => precio > 100);
console.log(preciosFiltrados); // [150, 200]
```

```
//Funciones flecha
let numeros = [10, 20, 30, 40, 50];
numeros.forEach((numero) => {console.log(numero);});
```

# .every() //.some()

Como su nombre indica, .**every()** devuelve true si todos los elementos del array cumplen la condición, false en caso contrario.

**.some()**, devuelve true si alguno de los elementos cumple la condición y false si ninguno la cumple.

```
let arrayNotas = [5.2, 3.9, 6, 9.75, 7.5, 3]
let todosAprobados = arrayNotas.every(nota  => nota > =  5)    // false
let algunAprobado = arrayNotas.some(nota  => nota > =  5)      // true
```

# .find()

Retorna el primer elemento que cumpla, en caso contrario retorna *undefined*.

```
let arrayNotas = [5.2, 3.9, 6, 9.75, 7.5, 3]
let primerAprobado = arrayNotas.find(nota  => nota > =  5)     // primerAprobado = 5.2
```

```
//Veremos más adelante objetos, pero si imaginemos la clase alumno con
propiedades como:
//nombre, apellido, dni, dirección
//Sería ideal poder buscar un alumno con un DNI concreto:
let alumnoBuscada = alumnos.find(alumno  => alumno.dni === '21345678Z')
// devolverá el objeto completo
```

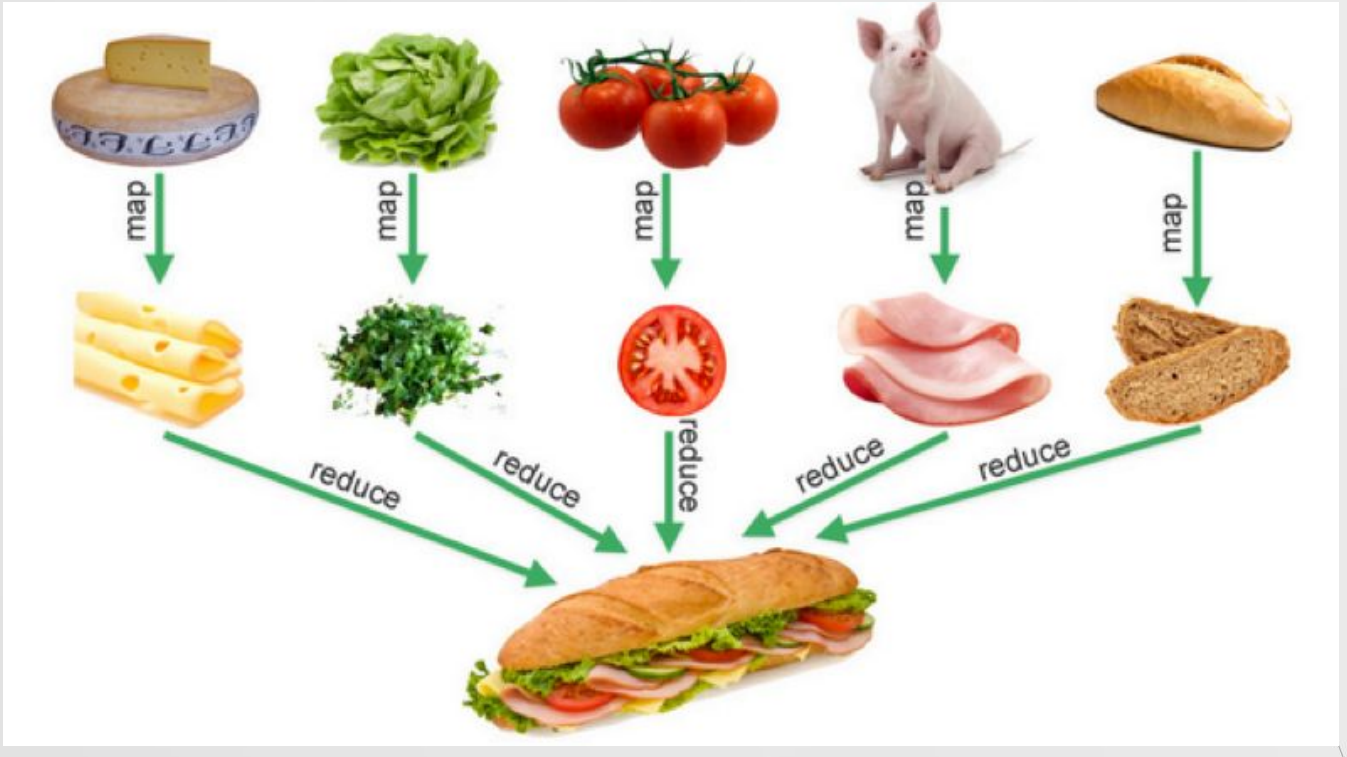
# .findIndex()

Retorna la posición del primer elemento que cumpla, en caso contrario retorna *-1*. Tiene más sentido en el trabajo por objetos.

# Ejercicio 1.2: Array

Dado un array con los días de la semana

- Obtén el primer día que empieza por ‘M’.
- Obtén la posición del día que empieza por ‘V’.
- Indica si algún día empieza por ‘S’.
- Indica si todos los días acaban por ‘s’.


map
map
map
map
map
reduce
reduce
reduce
reduce
reduce

# .map() //.reduce()

**.map()** es un método bastante empleado. Permite aplicar un mismo cambio a **todos** los elementos de una array creando un array nuevo con los cambios incorporados.

```
let arrayNotas = [5.2, 3.9, 6, 9.75, 7.5, 3]
let arrayNotasSubidas = arrayNotas.map(nota  => nota + nota * 0.10)
```

**.reduce()** permite obtener un valor calculado que se aplica a todos los elementos del array. Al ser un cálculo opcionalmente le podemos pasar el valor inicial, en caso contrario tomará el primer elemento.

```
let arrayNotas = [5.2, 3.9, 6, 9.75, 7.5, 3]
let sumaNotas = arrayNotas.reduce((total,nota)  => total+nota, 0)     // total = 35.35
// podríamos haber omitido el valor inicial 0 para total
let sumaNotas = arrayNotas.reduce((total,nota)  => total+nota)     // total = 35.35

//Pero también pudo querer obtener la nota más alta
let maxNota = arrayNotas.reduce((max,nota)  => nota > max ? nota : max)     // max = 9.75
```

# Ejercicio 1.3: Array

Dado un array con las siguientes notas `[3.4, 7.9, 8.0, 2.5, 5.6, 5.4, 9.0]`

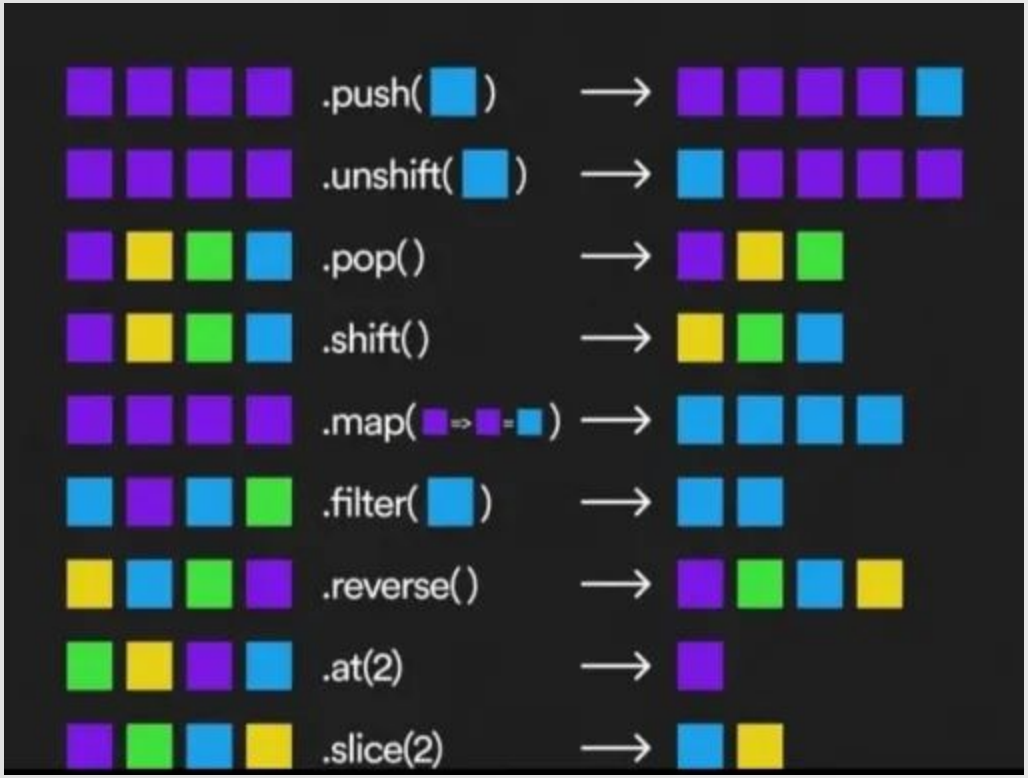
- Calcula la nota media
- Obtén la primera nota superior a 5
- Las notas son sobre 10, cambialas a sobre 20.

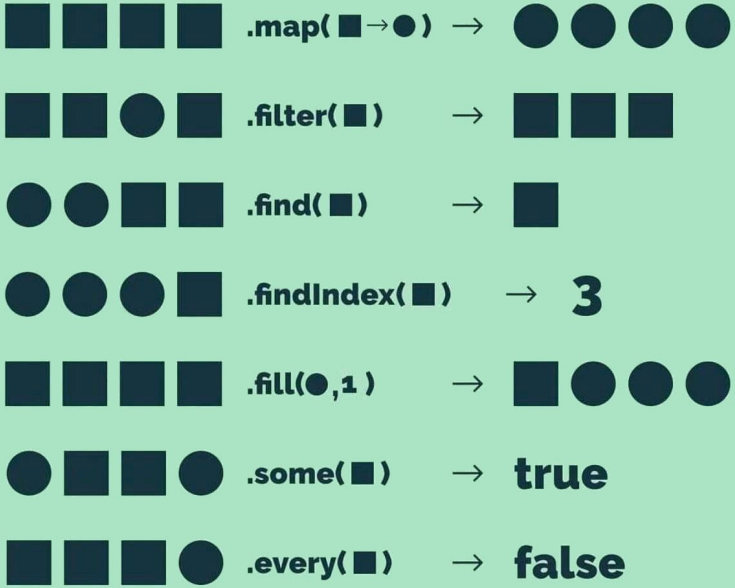
# FUNCIÓN Array.from()

**Array.from()** es un método *estático* de JavaScript que permite crear una nueva instancia de un Array a partir de un objeto iterable o uno similar a un array. Es una herramienta fundamental cuando trabajas con el DOM o estructuras de datos que parecen listas pero no tienen los métodos de los arrays.

```
newArray = Array.from(objetoIterativo, funciónMapa, thisArg)
```

```
let arrayNotas = [5.2, 3.9, 6, 9.75, 7.5, 3]
let arrayNotasSubidas = Array.from(arrayNotas, nota  => nota + nota * 0.10)
```


.push( )
.unshift( )
.pop()
.shift()
.map( )
.filter( )
.reverse()
.at(2)
.slice(2)


.map( → )
.filter( )
.find( )
.findIndex( )
3
.fill( ,1 )
.some( )
true
.every( )
false

# Ejercicio coches

Dado el siguiente array multidimensional realiza un programa :

1. Devuelve los vehículos de la marca que no sean ni 'Alfa Romeo' ni 'Kia' que tengan más de 20 años.
2. Ordena el array ordenado por precio descendentemente.
3. Devuelve el array con un 20% de rebaja al precio.

Restricciones:
Se deben utilizar 3 de los siguientes métodos map(), filter(), reduce() y sort()

```
let coches = [
    ["seat","Codoba",1997,10000],
    ["Kia","Sorento",1994,1000],
    ["seat","Todelo",2000,10000],
    ["Fiat","Bravo",2010,10200],
    ["Fiat","500",2010,10000],
    ["Mercedes","Calse B",2001,39000],
    ["seat","Ibiza",1993,10100],
    ["Alfa Romeo","Julieta",2002,10000],
    ["Alfa Romeo","159",2002,10400],
    ["Mercedes","Calse C",2001,1000],
    ["Alfa Romeo","147",1990,10500],
    ["Fiat","Punto",1990,999],
    ["Citroen","Saxo",1980,10300],
    ["Renault","Superc 5",1980,12000],
    ["BWM","M3",2020,1000],
    ["Kia","Picanto",1990,1000],
    ["Alfa Romeo","spider",1970,14500],
    ["Mercedes","Calse A",1994,60100],
    ["Mercedes","Calse D",2011,21221]
];
```

# Desestructuración de Arrays (Array Destructuring)

Es una sintaxis que permite "extraer" valores de arrays y asignarlos a variables individuales de forma rápida y limpia, siguiendo el orden de los índices.

Funciona como un "espejo": lo que pones a la *izquierda* del = debe tener la misma forma (la misma estructura de corchetes) que lo que hay a la *derecha*. JavaScript simplemente empareja las posiciones.

```
let coche = ["Seat", "Ibiza", 2022]
const marca = coche[0];
const modelo = coche[1];
const año = coche[2];

//Mejor con destructuracion: "Coge el primer elemento y ponlo en 'marca', el segundo en
'modelo'..."
const [marca, modelo, año, cv=150] = coche;
console.log(modelo);
```