# LENGUAJES DE MARCAS

# Fundamentos de JavaScript

## Anexo. Colecciones

Pablo Matías Garramone Ramírez

# Arrays

- En JavaScript, los arrays son un tipo de objetos. Podemos crear un array con la instancia de un objeto de clase **Array**.
- Estos no tienen un tamaño fijo, por tanto, podemos inicializarlo con un tamaño y luego añadirle mas elementos.
- El constructor puede recibir:
  - 0 parámetros (array vacío)
  - 1 numero (el tamaño del array)
  - Cualquier otro caso, se creará un array con los elementos recibidos.
- Debemos tener en cuenta que en JavaScript un array puede contener al mismo tiempo diferentes tipos de datos: number, string, boolean, object, etc.

```
let a = new Array(); // Crea un array vacío
a[0] = 13;
console.log(a.length); // Imprime 1
console.log(a[0]); // Imprime 13
console.log(a[1]); // Imprime undefined
```

- Fíjate que cuando accedes a una posición del array que no ha sido definida, devuelve **undefined**. La longitud de un array depende de las posiciones que han sido asignadas.

Pablo Matías Garramone Ramírez

# Array. Ampliar o reducir Array. Crear con []

- Cuando asignamos un valor del array más grande que la dimensión máxima definida en el array, JS automáticamente amplia el array para que quepa hasta la posición indicada y pone todo lo no asignada a undenfined.

```
let  a = new Array(12); // Crea un array de tamaño 12
console.log(a.length); // Imprime 12
a[20] = "Hello";
console.log(a.length); // Ahora imprime 21 (0-20). Las posiciones 0-19 tendrán el valor undefined
```

- Podemos reducir la longitud del array modificando directamente la propiedad de la longitud del array (**length)**. Si reducimos la longitud de un array, las posiciones mayores a la nueva longitud serán consideradas como undefined (borradas).

```
let  a = new Array("a", "b", "c", "d", "e"); // Array con 5 valores
console.log(a[3]); // Imprime "d"
a.length = 2; // Posiciones 2-4 serán destruidas
console.log(a[3]); // Imprime undefined
```

- Puedes crear un array usando corchetes en lugar de usar **new Array()**. Los elementos que pongamos dentro, separados por coma serán los elementos que inicialmente tendrá el array.

```
let  a = ["a", "b", "c", "d", "e"]; // Array de tamaño 5, con 5 valores inicialmente
console.log(typeof a); // Imprime object
console.log(a instanceof Array); // Imprime true. a es una instancia de array
a[a.length] = "f"; // Insertamos in nuevo elemento al final
console.log(a); // Imprime ["a", "b", "c", "d", "e", "f"]
```

Pablo Matías Garramone Ramírez

# Array. Recorrer un Array

- Podemos recorrer un array con los bucles tradicionales **FOR y WHILE**. Yendo posición a posición hasta el final del array (length).
- Otra opción es el **bucle for..in**. Con este bucle podemos iterar los índices de un array o las propiedades de un objeto (similar al bucle foreach de otros lenguajes, pero recorriendo los índices en lugar de los valores).
- Una tercera forma, desde ES2015, es que podemos iterar por los elementos de un array o incluso por los caracteres de una cadena sin utilizar el índice. Para ello se utiliza el bucle **for..of** (que se comporta como un bucle foreach de otros lenguajes). **bucles_iterar arrays.js**

```
let  ar = new Array(4, 21, 33, 24, 8);

let i = 0;
while(i < ar.length) { // Imprime 4 21 33 24 8
   console.log(ar[i]);
   i++;
}

for(let i = 0; i < ar.length; i++) { // Imprime 4 21 33 24 8
   console.log(ar[i]);
}

for (let i in ar) { // Imprime 4 21 33 24 8
   console.log(ar[i]);
}
```

```
let a = ["Item1", "Item2", "Item3", "Item4"];

for(let index in a) {
   console.log(a[index]);
}

for(let item of a) { // Hace lo mismo que el bucle anterior
   console.log(item);
}

let str = "abcdefg";

for(let letter of str) {
   if(letter.match(/^[aeiou]$/)) {
      console.log(letter + " es una vocal");
   } else {
      console.log(letter + " es una consonante");
   }
}
```

Pablo Matías Garramone Ramírez

# Array. Métodos

- **unshift y push**: Insertar valores en un array al principio (unshift) y al final (push)

```
let a = [];
a.push("a"); // Inserta el valor al final del array
a.push("b", "c", "d"); // Inserta estos nuevos valores al final
console.log(a); // Imprime ["a", "b", "c", "d"]
a.unshift("A", "B", "C"); // Inserta nuevos valores al principio del array
console.log(a); // Imprime ["A", "B", "C", "a", "b", "c", "d"]
```

- **shift y pop**: Eliminar valores en un array al principio (shift) y al final (pop). Devuelven el valor eliminado:

```
console.log(a.pop()); // Imprime y elimina la última posición → "d"
console.log(a.shift()); // Imprime y elimina la primera posición → "A"
console.log(a); // Imprime ["B", "C", "a", "b", "c"]
```

- **join y toString**: Convierte un array en string, con join podemos elegir el carácter separador

```
let a = [3, 21, 15, 61, 9];
console.log(a.join()); // Imprime "3,21,15,61,9""
console.log(a.join(" -#- ")); // Imprime "3 -#- 21 -#- 15 -#- 61 -#- 9"
```

- **concat**: Concatenar 2 o más arrays

```
let a = ["a", "b", "c"];
let b = ["d", "e", "f"];
let c = a.concat(b);
console.log(c); // Imprime ["a", "b", "c", "d", "e", "f"]
console.log(a); // Imprime ["a", "b", "c"] . El array original no ha sido modificado
```

Pablo Matías Garramone Ramírez

# Array. Métodos II

- **slice**: Nos devuelve un nuevo sub-array. Indicando posición de inicio y fin.

```
let a = ["a", "b", "c", "d", "e", "f"];
let b = a.slice(1, 3); // (posición de inicio → incluida, posición final → excluida)
console.log(b); // Imprime ["b", "c"]
console.log(a); // Imprime ["a", "b", "c", "d", "e", "f"]. El array original no es modificado
console.log(a.slice(3)); // Un parámetro. Devuelve desde la posición 3 al final → ["d", "e", "f"]
```

- **splice**: Elimina elementos del array original y devuelve los eliminados, también permite insertar nuevos valores:

```
let  a = ["a", "b", "c", "d", "e", "f"];
a.splice(1, 3); // Elimina 3 elementos desde la posición 1 ("b", "c", "d")
console.log(a); // Imprime ["a", "e", "f"]
a.splice(1,1, "g", "h"); // Elimina 1 elemento en la posición 1 ("e"), e inserta "g", "h" en esa posición
console.log(a); // Imprime ["a", "g", "h", "f"]
a.splice(3, 0, "i"); // En la posición 3, no elimina nada, e inserta "i"
console.log(a); // Imprime ["a", "g", "h", "i", "f"]
```

- **reverse**: Invierte el orden de un array

```
let  a = ["a", "b", "c", "d", "e", "f"];
a.reverse(); // Hace el reverse del array original
console.log(a); // Imprime ["f", "e", "d", "c", "b", "a"]
```

- **indexOf y lastIndexOf**: Usando **indexOf**, podemos conocer si el valor que le pasamos se encuentra en el array o no. Si lo encuentra nos devuelve la primera posicion donde esta, y si no, nos devuelve -1. Usando el metodo **lastIndexOf** nos devuelve la primera ocurrencia encontrada empezando desde el fnal.

```
let a = [3, 21, 15, 61, 9, 15];
console.log(a.indexOf(15)); // Imprime 2
console.log(a.indexOf(56)); // Imprime -1. No encontrado
console.log(a.lastIndexOf(15)); // Imprime 5
```

Pablo Matías Garramone Ramírez

# Array. Métodos III

- **sort**: Ordena todos los elementos del array sin son strings

```
let  a = ["Peter", "Anne", "Thomas", "Jen", "Rob", "Alison"];
a.sort(); // Ordena el array original
console.log(a); // Imprime ["Alison", "Anne", "Jen", "Peter", "Rob", "Thomas"]
```

- Si no son strings, deberemos pasar a sort una función de ordenación, que comparará los dos valores del array y devolverá el valor numérico indicando cuál es menor (negativo si es menor, 0 sin son iguales y postivo si es mayor).

```
let a = [20, 6, 100, 51, 28, 9];
a.sort(); // Ordena el array original
console.log(a); // Imprime [100, 20, 28, 51, 6, 9]
a.sort((n1, n2) => n1 - n2);
console.log(a); // Imprime [6, 9, 20, 28, 51, 100]
```

- **every**: Devolverá un booleano si todos los elementos del array cumplen o no una condición:

```
let a = [3, 21, 15, 61, 9, 54];
console.log(a.every(num =>  num < 100));  // Comprueba si cada número es menor a 100. Imprime true
console.log(a.every(num => num % 2 == 0)); // Comprueba si cada número es par. Imprime false
```

- **some**: Similar a every, pero devuelve cierto en el momento que uno de los elementos cumplen la condición

```
let a = [3, 21, 15, 61, 9, 54];
console.log(a.some(num => num % 2 == 0));  // Comprueba si algún elemento del array es par. Imprime true
```

- **forach**: Itera los elementos de un array. De forma opcional, podemos llevar un seguimiento del índice al que esta accediendo en cada momento, e incluso recibir el array como tercer parámetro

```
let a = [3, 21, 15, 61, 9, 54];
let sum = 0;
a.forEach(num => sum += num);
console.log(sum); // Imprime 163

a.forEach((num, indice, array) => { // índice y array son parámetros opcionales
   console.log("Índice " + indice + " en [" + array + "] es " + num);
}); // Imprime -> Índice 0 en [3,21,15,61,9,54] es 3, Índice 1 en [3,21,15,61,9,54] es 21, ...
```

Pablo Matías Garramone Ramírez

# Array. Métodos IV

- **map**: Modifica los elementos de un array. Recibe una función que transforma cada elemento y lo devuelve. Al final devuelve un nuevo array del mismo tamaño que el original pero transformado por la función dada.

```
let a = [4, 21, 33, 12, 9, 54];
console.log(a.map(num => num*2)); // Imprime [8, 42, 66, 24, 18, 108]
```

- **filter**: Para filtrar los elementos de un array, devuelve un nuevo array con todos los elementos del mismo que cumplen una condición dada.

```
let a = [4, 21, 33, 12, 9, 54];
console.log(a.filter(num => num % 2 == 0));  // Imprime [4, 12, 54]
```

- **reduce**: Utiliza una función que acumula un valor, procesando cada elemento (segundo parámetro) con el valor acumulado (primer parámetro). Como segundo parámetro de reduce deberías pasar el valor inicial del array, si no lo pones por defecto es el primero, si el array está vacío devuelve undefined.

```
let a = [4, 21, 33, 12, 9, 54];
console.log(a.reduce((total, num) => total + num, 0));  // Suma todos los elementos del array. Imprime 133
console.log(a.reduce((max, num) => num > max? num : max, 0));  // Número máximo del array. Imprime 54
```

- **reduceRight**: Hace lo mismo que reduce pero al revés

```
let a = [4, 21, 33, 12, 9, 154];
// Comienza con el último número y resta todos los otros números
console.log(a.reduceRight((total, num)  => total - num));
// Imprime 75 (Si no queremos enviarle un valor inicial, empezará con el valor de la última posición del array
```

Pablo Matías Garramone Ramírez

# Array. Métodos Objeto Array

- **Array.of(value)** → Si queremos instanciar un array con un solo valor, y este es un numero, con new Array() no podemos hacerlo, ya que crea vacío con ese numero de posiciones

```
let array = new Array(10); // Array vacío (longitud 10)
let array = Array(10); // Mismo que arriba: array vacío ( longitud 10)
let array = Array.of(10); // Array con longitud 1 -> [10]
let array = [10]; // // Array con longitud 1 -> [10]
```

- **Array.from(array, func)** → Funciona de forma similar al método **map**, crea un array desde otro array. Se aplica una operación de transformación (función lambda o anónima) para cada ite

```
let array = [4, 5, 12, 21, 33];
let array2 = Array.from(array, n => n * 2);
console.log(array2); // [8, 10, 24, 42, 66]
let array3 = array.map(n => n * 2); // Igual que Array.from
console.log(array3); // [8, 10, 24, 42, 66]
```

- **Array.fill(value)** → Este método sobrescribe todas las posiciones de un array con un nuevo val ay que se ha creado con N posiciones.

```
let sums = new Array(6); // Array con 6 posiciones
sums.fill(0); // Todas las posiciones se inicializan a 0
console.log(sums); // [0, 0, 0, 0, 0, 0]

let numbers = [2, 4, 6, 9];
numbers.fill(10); // Inicializamos las posiciones  al valor 10
console.log(numbers); // [10, 10, 10, 10]
```

- **Array.fill(value, start, end)** → Este método hace lo mismo que antes pero rellenando el array desde una posición inicial (incluida) hasta una final (excluida). Si no se especifica la ultima p

```
let numbers = [2, 4, 6, 9, 14, 16];
numbers.fill(10, 2, 5); // Las posiciones 2,3,4 se ponen a 10
console.log(numbers); // [2, 4, 10, 10, 10, 16]

let numbers2 = [2, 4, 6, 9, 14, 16];
numbers2.fill(10, -2); // Las dos últimas posiciones se ponen a 10
console.log(numbers2); // [2, 4, 6, 9, 10, 10]
```

Pablo Matías Garramone Ramírez

# Array. Métodos Objeto Array II

- **Array.fnd(condition)** → Encuentra y devuelve el primer valor que encuentre que cumple la condición que se establece. Con **fndIndex**, devolvemos la posicion que ocupa ese valor en el array.

```
let numbers = [2, 4, 6, 9, 14, 16];
console.log(numbers.find(num => num >= 10)); // Imprime 14 (primer valor encontrado >= 10)
console.log(numbers.findIndex(num => num >= 10)); // Imprime 4 (numbers[4] -> 14)
```

- **Array.copyWithin(target, startwith)** → Copia los valores del array empezando desde la posición **startWith**, hasta la posición **target** en el resto de posiciones del array (en orden).

```
let numbers = [2, 4, 6, 9, 14, 16];
numbers.copyWithin(3, 0); // [0] -> [3], [1] -> [4], [2] -> [5]
console.log(numbers); // [2, 4, 6, 2, 4, 6]
```

Pablo Matías Garramone Ramírez

# Array. Rest

- **Rest** es la acción de transformar un grupo de parámetros en un array, y **spread** es justo lo opuesto, extraer los elementos de un array (o de un string) a variables.
- Para usar **rest** en los parámetros de una función, se declara siempre como ultimo parámetro (**1 máximo**) y se le ponen tres puntos'...' delante del mismo.
- Este parámetro se transformara automáticamente en un array conteniendo todos los parámetros restantes que se le pasan a la función.
- Si por ejemplo, el parámetro ***rest*** esta en la tercera posición, contendrá todos los parámetros que se le pasen a excepción del primero y del segundo (a partir del tercero).

```
function getMedia(...notas) {
   console.log(notas); // Imprime [5, 7, 8.5, 6.75, 9] (está en un array)
   let total = notas.reduce((total,notas) => total + notas, 0);
   return total / notas.length;
}
console.log(getMedia(5, 7, 8.5, 6.75, 9)); // Imprime 7.25

function imprimirUsuario(nombre, ...lenguajes) {
   console.log(nombre + " sabe " + lenguajes.length + " lenguajes: " + lenguajes.join(" - "));
}

// Imprime "Pedro sabe 3 lenguajes: Java - C# - Python"
imprimirUsuario("Pedro", "Java", "C#", "Python");
// Imprime "María sabe 5 lenguajes: JavaScript - Angular - PHP - HTML - CSS"
imprimirUsuario("María", "JavaScript", "Angular", "PHP", "HTML", "CSS");
```

Pablo Matías Garramone Ramírez

# Array. Spread

- **Spread** es lo “opuesto” de **rest**. Si tenemos una variable que contiene un array, y ponemos los tres puntos ‘...’ delante de este, extraerá todos sus valores.
- Podemos usar la propiedad por ejemplo con el método **Math.max,** el cual recibe un numero indeterminado de parámetros y devuelve el mayor de todos.

```
let nums = [12, 32, 6, 8, 23];
console.log(Math.max(nums)); // Imprime NaN (array no es válido)
console.log(Math.max(...nums)); // Imprime 32 -> equivalente a Math.max(12, 32, 6 ,8 ,23)
```

- Podemos usar spread para clonar un array.

```
let a = [1, 2, 3, 4];
let b = a; // Referencia el mismo array que 'a' (las modificaciones afectan a ambos).
let c = [...a]; // Nuevo array -> contiene [1, 2, 3, 4]
```

Pablo Matías Garramone Ramírez

# Desestructuración de Arrays

- **Desestructuración** es la acción de extraer elementos individuales de un array (o propiedades de un objeto) directamente en variables individuales.
- Podemos también *desestructurar* un string en caracteres.
- Vamos a ver un ejemplo donde asignamos los tres primeros elementos de un array, en tres variables diferentes, usando una única asignación.

```
let array = [150, 400, 780, 1500, 200];
let [first, second, third] = array; // Asigna los tres primeros elementos del array
console.log(third); // Imprime 780
```

- Si nos queremos saltar alguno simplemente ponemos las comas sin valor alguno.

```
let array = [150, 400, 780, 1500, 200];
let [first, , third] = array; // Asigna el primer y tercer elemento
console.log(third); // Imprime 780
```

Pablo Matías Garramone Ramírez

# Desestructuración de Arrays II

- Podemos asignar el resto del array a la ultima variable que pongamos entre corchetes usando **rest** (como en el punto anterior del tema):

```
let array = [150, 400, 780, 1500, 200];
let [first, second, ...rest] = array; // rest -> array
console.log(rest); // Imprime [780, 1500 ,200]
```

- Si queremos asignar mas valores de los que contiene el array y no queremos obtener undefined, podemos usar valores por defecto:

```
let array = ["Peter", "John"];
let [first, second = "Mary", third = "Ann"] = array; // rest -> array
console.log(second); // Imprime "John"
console.log(third); // Imprime "Ann" -> valor por defecto
```

- También podemos desestructurar **arrays anidados**:

```
let sueldos = [["Pedro", "Maria"], [24000, 35400]];
let [[nombre1, nombre2], [sueldo1, sueldo2]] = sueldos;
console.log(nombre1 + " gana " + sueldo1 + "€"); // Imprime "Pedro gana 24000€"
```

- También se puede desestructurar un array enviado como parámetro a una función en valores individuales:

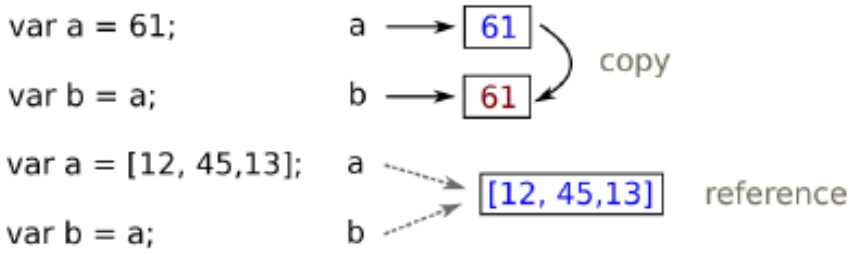
```
function imprimirUsuario([id, nombre, email], otraInfo = "Nada") {
   console.log("ID: " + id);
   console.log("Nombre: " + nombre);
   console.log("Email: " + email );
   console.log("Otra info: " + otraInfo );
}

let infoUsu = [3, "Pedro", "peter@gmail.com"];
imprimirUsuario(infoUsu, "No es muy listo");
```

Pablo Matías Garramone Ramírez

# Referencia vs Copia

- A diferencia de los tipos primitivos como **boolean**, **number** o **string**, los arrays son tratados como un objeto en JavaScript, lo cual significa que tienen un puntero o referencia a la dirección de memoria que contiene el array.
- Cuando se copia una variable o se envía esa variable como parámetro a una función, estamos copiando la referencia y no el array, por tanto ambas variables apuntaran a la misma zona de memoria.
- Si por ejemplo cambiamos la información que teníamos almacenada en una de esas variables, estaremos cambiando la información de la otra también ya que son el mismo objeto.




Pablo Matías Garramone Ramírez

# Referencia vs Copia. Ejemplos

```
var a = 61;
var b = a; // b = 61
b = 12;
console.log(a); // Imprime 61
console.log(b); // Imprime 12

var a = [12, 45, 13];
var b = a; // b ahora referencia al mismo array que a
b[0] = 0;
console.log(a); // Imprime [0, 45, 13]
```

- Otro ejemplo, pasando el array a una función:

```
function changeNumber(num) {
   var localNum = num; // Variable local. Copia el valor
   localNum = 100; // Cambia solo localNum. El valor fue copiado
}

var num = 17;
changeNumber(num);
console.log(num); // Imprime 17. No ha cambiado el valor original


function changeArray(array) {
   var localA = array; // Variable local. Hace referencia al original
   localA.splice(1,1,101, 102); // Elimina 1 item en la posición1 e inserta 101 y 102 ahí
}

var a = [12, 45, 13];
changeArray(a);
console.log(a); // Imprime [12, 101, 102, 13]. Ha sido cambiado dentro de la función
```

Pablo Matías Garramone Ramírez