



MÓDULO 1. MF0951_2
INTEGRAR COMPONENTES SOFTWARE EN PÁGINAS WEB

UNIDAD FORMATIVA 1.
UF1305 INTEGRAR COMPONENTES SOFTWARE EN PÁGINAS WEB.



4

4.1

Javascript. Módulo II

Arrays arreglos

_Arrays

_Array de 1- 2 dimensiones

_Qué es y para qué se usa

_Creación, acceso y añadir elementos

_Matrices Asociativas

_Métodos de Array

_Ordenar Array

_Iterar Arrays

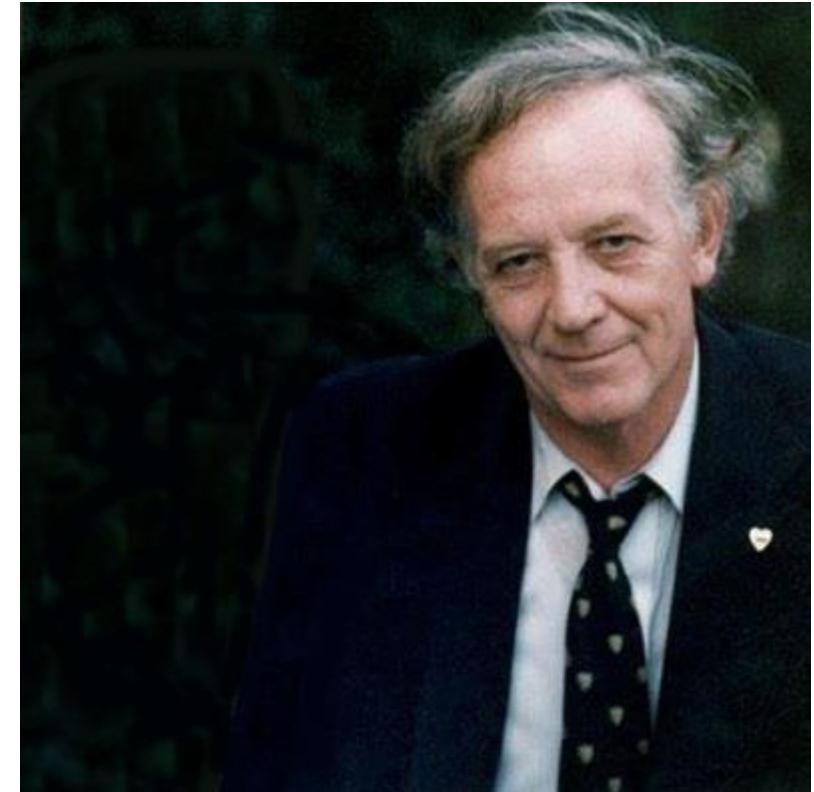
_Const Array

_Arrays multidimensionales



"¿Los índices de los arrays deberían comenzar en 0 o en 1?
Mi propuesta neutral de usar 0.5
fue rechazada, en mi opinión,
sin la debida consideración"

- Stan Kelly-Bootle



Stan Kelly-Bootle fue un escritor, cantante y compositor, y científico de la computación. Su canción más conocida es el "Liverpool Lullaby", que Judy Collins grabó en 1966 para su álbum, *In My Life*. Cilla Black grabó tres años más tarde como el B-side de su hit pop "Conversations".

Javascript. II

4.1.

Arrays/Arreglos

En JavaScript los arrays son objetos y una de las maneras de crearlos es con el constructor Array().

Arrays, Vectores, Matrices

Los arrays son una de las estructuras de datos fundamentales en programación.

También suelen llamarse vectores o matrices.

Se trata de un tipo de **dato complejo que agrupa varios elementos**.

Los arrays de **una dimensión** se pueden ver como **una lista** ordenada de varios elementos.

Los de **dos dimensiones**, como una matriz con **filas y columnas**.

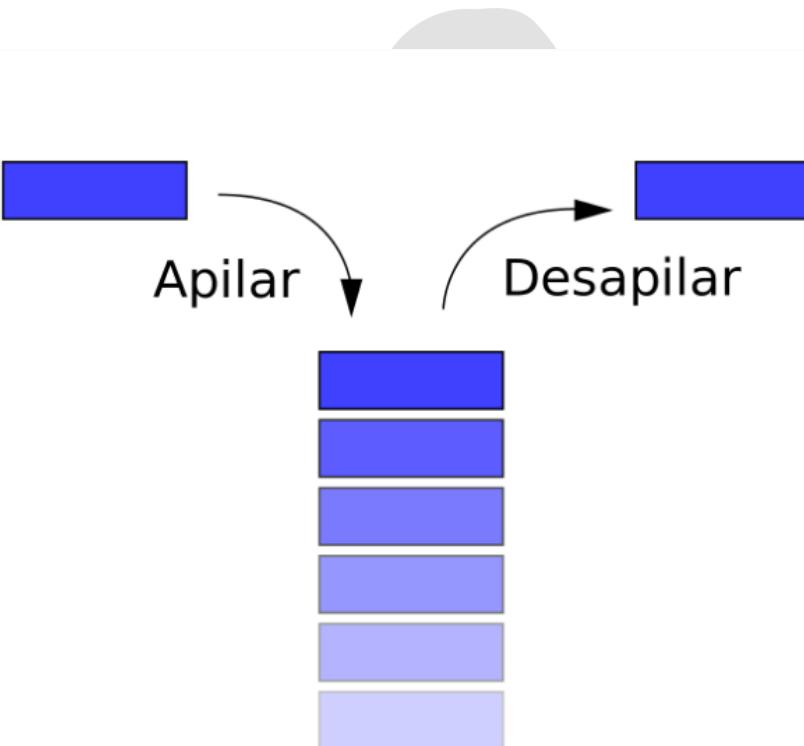
Se accede a cada uno de los elementos por medio de su posición en el array.

4.1.Array/arreglos_ Pilas o stacks

(Stack). Es una estructura de datos que consta de una serie de valores en el cual las inserciones y eliminaciones se hacen por un extremo llamado cima o tope. Esta estructura se conoce también como LIFO o stack (apilamiento).

Como su nombre lo indica, esta estructura va "apilando" los datos. A diferencia de las colas, los datos son extraídos en el orden inverso al que se introdujeron.

Son idénticas a las pilas ordinarias de la vida real: una pila de libros, una pila de ropa, etc.



4.1.Array/arreglos_ Pilas o stacks

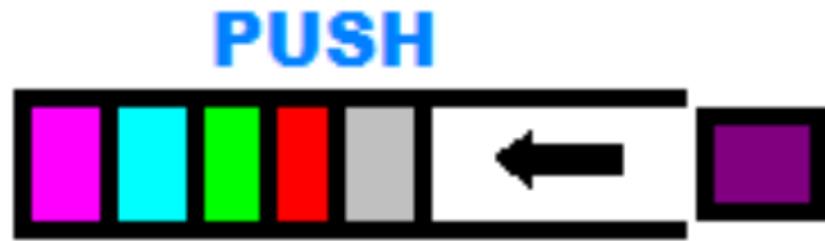
En teoría el tamaño de una pila no está limitado, pero en la práctica existe una limitación física determinada por la disponibilidad de memoria, por esta razón se utilizan vectores cuyo tamaño sea suficientemente grande para realizar las operaciones requeridas.

Para conocer el estado de la pila en un instante dado se usa una variable, que podría llamarse *tope* y que será utilizada como subíndice del vector. Si la pila no tiene elementos, se dice que es una pila vacía. Al agregar un elemento a la pila, la variable *tope* se incrementa en uno. Al quitar un elemento de la pila la variable *tope* se disminuye en uno.

4.1.Array/arreglos_operaciones básicas con pilas

Las operaciones básicas en una pila son:

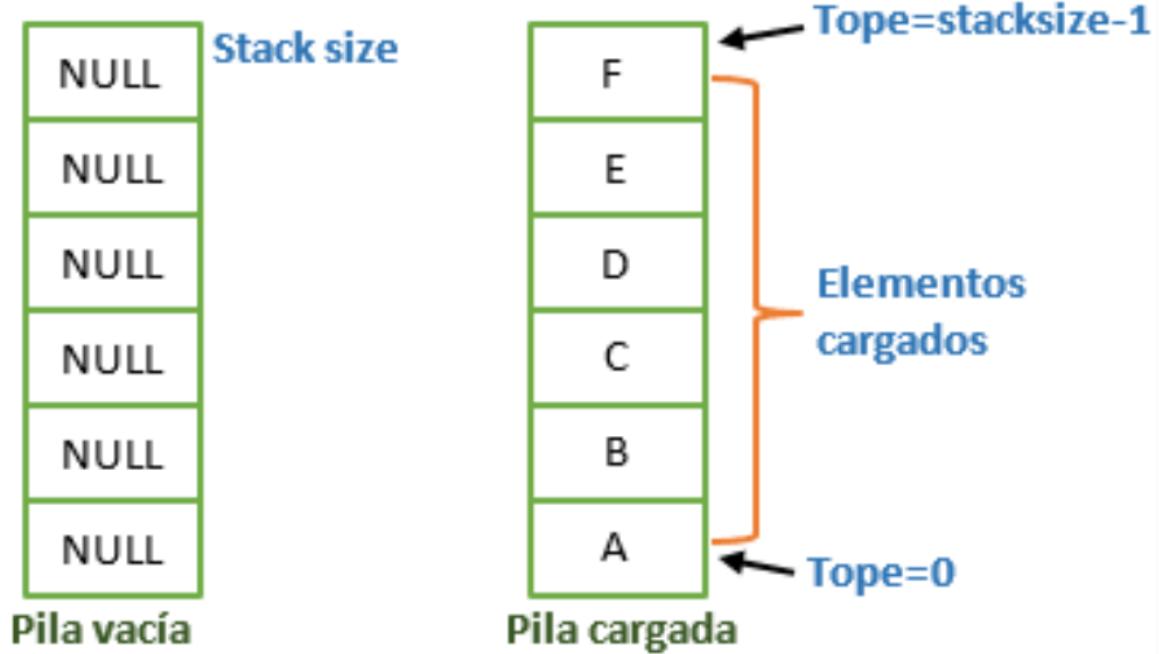
- **Crear o inicializar.**
- **pila vacía.**
- **pila llena.**
- **poner.**
- **sacar.**



Ejemplo con una pila

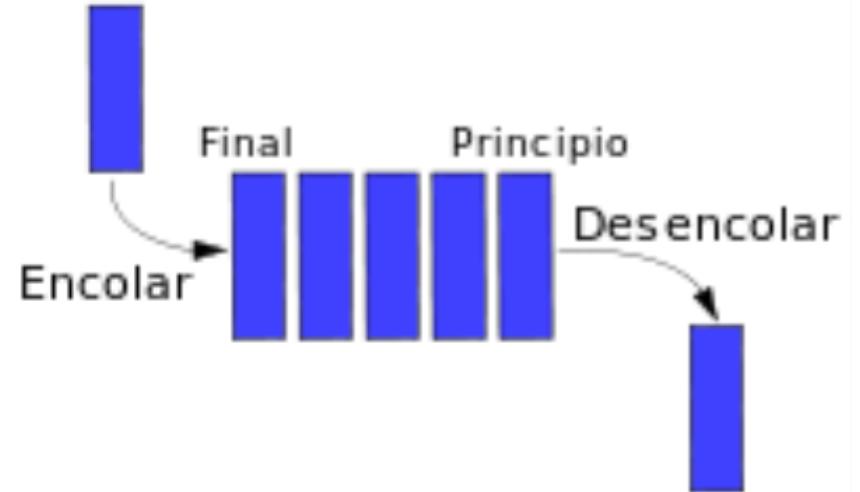
En la imagen se puede ver dos pilas (una vacía y otra cargada).

Por ejemplo, en la pila cargada, para obtener el elemento A, o sea $\text{Tope}[0]=A$, se debe ir sacando primero el elemento F ($\text{Tope}[5]=F$), luego ($\text{Tope}[4]=E$) y así hasta llegar al elemento buscado (A).



4.1.Array/arreglos_Colas o Queues

Una cola (queue en inglés) es una estructura de datos compuesta por una serie de elementos donde insertamos data al final de la serie, y retiramos data por el frente.



Es lo que llamamos una estructura FIFO (First In, First Out). Usamos colas para varias cosas, ordenar operaciones, la cola de impresión, y el clásico ejemplo de una cola en la ventanilla de un banco.

4.1.Array/arreglos_Colas o Queues

La particularidad de una estructura de datos de cola es el hecho de que solo podemos acceder al primer y al último elemento de la estructura.

Así mismo, los elementos solo se pueden eliminar por el principio y solo se pueden añadir por el final de la cola.



Ejemplos de colas en la vida real serían: personas comprando en un supermercado, esperando para entrar a ver un partido de béisbol, esperando en el cine para ver una película, una pequeña peluquería, etc. La idea esencial es que son todos líneas de espera.

15	20	9	18	19
----	----	---	-------	----	----

1.Ejemplo de Cola

15	20	9	18	19
----	----	---	-------	----	----

2.Vamos a Insertar el 13 en la Cola.

15	20	9	18	19	13
----	----	---	-------	----	----	----

3.Sacamos el frente de la Cola (15)

20	9	18	19	13
----	---	-------	----	----	----

4.1.Array/arreglos_Funciones/Métodos. Concatenar

FUNCIONES PARA AÑADIR ELEMENTOS O CONCATENAR ARRAYS

FUNCIÓN	UTILIDAD	EJEMPLOS aprenderaprogramar.com
concat(it1, it2, ... , itN)	Devuelve la concatenación de it1, it2, ..., itN con el array sobre el que se invoca. It1, it2, ..., itN pueden ser tipos primitivos u objetos.	var A3 = A1.concat(5, 9); var A4 = A1.concat(A2); var A5 = A1.concat([-4, 22, 11]);
push(x)	Añade x al final del array como nuevo (o nuevos) elemento, y devuelve la nueva longitud del array.	A1.push(55, 66); //Añade 55 y 66 al final
unshift(x)	Añade x al principio del array como nuevo (o nuevos) elementos.	A1.unshift([77, 88, 99]); //Ahora A1 tiene 3 elementos más, al principio
splice (ind, 0, it1, it2, ... , itN)	Modifica el array añadiendo los elementos it1, it2, ..., itN, que son insertados en la posición ind (desplazando a los existentes).	A1.splice(3, 0, 'xxx', 'yyy'); //Inserta xxx en posición 3, yyy en posición 4 y desplaza a los elementos existentes antes.
splice (ind, cuant, it1, it2, ... , itN)	Modifica el array eliminando cuantos elementos e insertando it1, it2, ..., itN, desde el índice ind.	ejemplo.splice(3, 2, 'es', 'un', 'en'); //Se borran dos elementos y se insertan tres, con lo que la longitud del array es 1 más de la anterior.

4.1.Array/arreglos_Funciones/Métodos. Añadir

FUNCIONES PARA EXTRAER ELEMENTOS O PARTES DE UN ARRAY CON ELIMINACIÓN

FUNCIÓN	UTILIDAD	EJEMPLOS aprenderaprogramar.com
pop()	Elimina el último elemento del array y lo devuelve.	var ultimoElemento = A1.pop(); //Ahora A1 tiene 1 elemento menos
shift()	Elimina el primer elemento del array y lo devuelve.	var primerElemento = A1.shift(); //Ahora A1 tiene 1 elemento menos
splice (ind, cuant)	Modifica el array borrando cuant elementos a partir del índice ind.	A1.splice(3, 2);
splice (ind, cuant, it1, it2, ..., itN)	Modifica el array eliminando cuant elementos e insertando it1, it2, ..., itN, desde el índice ind.	ejemplo.splice(3, 2, 'es', 'un', 'en'); //Se borran dos elementos y se insertan tres, con lo que la longitud del array es 1 más de la anterior.
delete A[ind]	Elimina el elemento con índice ind del array A. El contenido a[ind] pasa a ser undefined.	delete A[7]; //Ahora A[7] contiene undefined

4.1.Array/arreglos_Funciones/Métodos. Extraer

FUNCIONES PARA EXTRAER PARTES O ELEMENTOS DE UN ARRAY SIN ALTERARLO

FUNCIÓN	UTILIDAD	EJEMPLOS aprenderaprogramar.com
slice (firstIn, lastOut)	<p>Devuelve un array con los elementos extraídos entre los índices firstIn y lastOut-1.</p> <p>Es decir, el elemento en la posición firstIn se incluye y el elemento en la posición lastOut se excluye.</p>	<pre>var result = [1, 2, 3, 4, 5].slice(1,4); //result contiene [2, 3, 4]</pre>
slice (firstIn)	<p>Devuelve un array con los elementos extraídos entre el índice firstIn y el último elemento.</p> <p>Si se indica un valor negativo, se extrae un array con los firstIn últimos elementos.</p>	<pre>var result = [1, 2, 3, 4, 5].slice(-2) //result contiene [4, 5]</pre>
slice ()	<p>Si se usa esta función sin argumentos devuelve un array con todos los elementos.</p>	<pre>var result = [1, 2, 3, 4, 5].slice(); //result contiene [1, 2, 3, 4, 5]</pre>

4.1.Array/arreglos_Funciones/Métodos. Recuperar indices

FUNCIONES PARA RECUPERAR ÍNDICES DE POSICIONES

FUNCIÓN	UTILIDAD	EJEMPLOS aprenderaprogramar.com
indexOf(x)	Busca x dentro del array y devuelve la posición de la primera ocurrencia.	<code>var result = A1.indexOf(14);</code>
lastIndexOf()	Busca x dentro del array empezando por el final y devuelve la posición de primera ocurrencia.	<code>var result = A1.lastIndexOf(14);</code>

4.1.Array/arreglos_Funciones/Métodos. Transformar array en strings

FUNCIONES PARA TRANSFORMAR ARRAYS EN STRINGS O TIPOS PRIMITIVOS

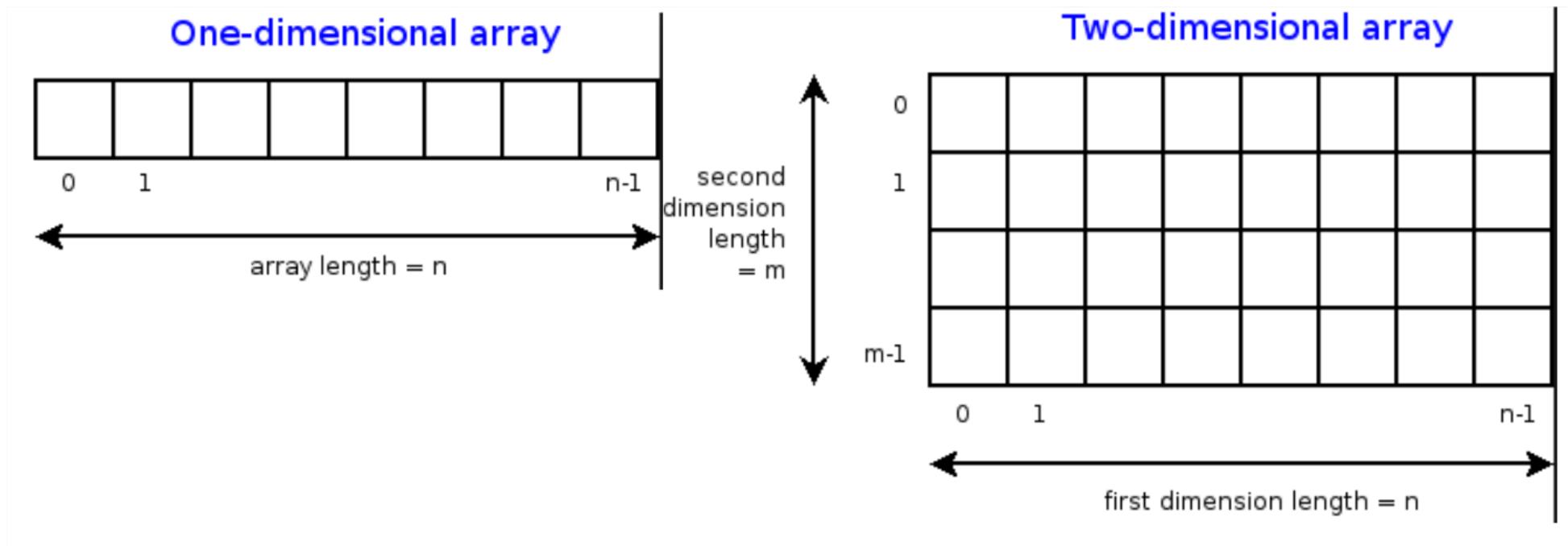
FUNCIÓN	UTILIDAD	EJEMPLOS aprenderaprogramar.com
join(separador)	Une los elementos de un array en una cadena de texto donde cada elemento está separado por 'separador'.	<code>var frase = ['quiero', 'aprender'].join(' '); //frase es tipo String y contiene 'quiero aprender'</code>
toString()	Une los elementos de un array en una cadena de texto donde cada elemento está separado por una coma.	<code>var frase = ['quiero', 'aprender'].toString(); //frase es tipo String y contiene 'quiero,aprender'</code>
valueOf()	Este método devuelve la representación como tipo primitivo de un objeto. En el caso de un array, hace lo mismo que el método <code>toString()</code> .	<code>alert (A1.valueOf()); //Mismo resultado que alert(A1);</code>

4.1.Array/arreglos_Funciones/Métodos. Ordenar

FUNCIONES QUE PERMITEN ORDENAR O REORDENAR ARRAYS

FUNCIÓN	UTILIDAD	EJEMPLOS aprenderaprogramar.com
<code>reverse()</code>	Invierte el orden de los elementos en el array (el final pasa a ser el principio).	<code>A1.reverse();</code> //Los elementos quedan ordenados al revés
<code>sort()</code>	Si no recibe parámetros, ordena los elementos del array por orden alfabético (que no coincide con el numérico), quedando el array modificado. Comentaremos esta función con más detenimiento.	<code>var result = [2, 11, 111, 7].sort();</code> //result vale [11, 111, 2, 7] porque el orden es alfabético, no numérico.

4.1.Array/arreglos_ Array de 1- 2 dimensiones



4.1.Array/arreglos_Qué es y para qué se usa

Por ejemplo, para almacenar gastos diarios para cada día del año, puede declarar una variable de matriz con 365 elementos en lugar de declarar 365 variables.

Cada elemento de la matriz contiene un valor. La siguiente instrucción declara la variable de la matriz con 365 elementos.

```
var anual= new Array(365)
```

De manera predeterminada, una matriz se indexa comenzando por cero, de modo que el límite superior de la matriz es 364 en lugar de 365.

4.1.Array/arreglos_Qué es y para qué se usa



Si tiene una lista de elementos (una lista de nombres de automóviles, por ejemplo), almacenar los automóviles en variables individuales podría tener este aspecto:

```
let car1 = "Saab";
let car2 = "Volvo";
let car3 = "BMW";
```

Crear una matriz

```
const cars = ["Saab", "Volvo", "BMW"];
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p id="demo"></p>

<script>
const cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
</script>

</body>
</html>
```

JavaScript Arrays

Saab,Volvo,BMW



4.1.Array/arreglos_ Creación arrays básicos

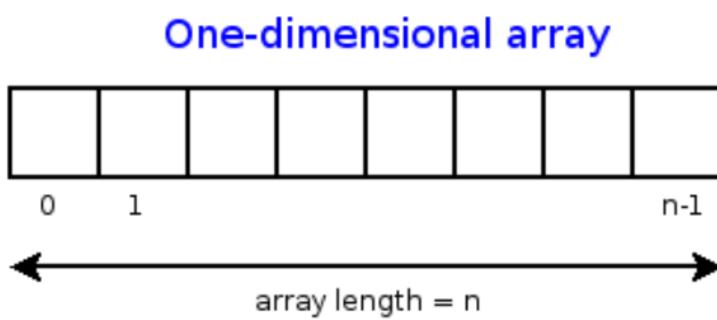
En JavaScript los arrays son objetos y una de las maneras de crearlos es con el constructor `Array()`.

```
var vacio = new Array();//vacío
```

```
var sistemas = new Array("Ubuntu", "Windows", "Android");
```

```
//con tres elementos
```

One-dimensional array



4.1.Array/arreglos_ Creación arrays básicos

Si no se usa **ningún argumento**, se crea un array vacío.

Si se le pasan varios elementos se crea una array con esos elementos.

Hay una excepción con la que hay que **tener cuidado**: si se llama al constructor con un solo número, crea un array con ese número de elementos (todos con valor undefined).

```
var l1= new Array(4,3); //array con dos elementos (un 4 y un 3)
```

```
var l1= new Array(4); //array con cuatro elementos (todos undefined)
```



```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p id="demo"></p>

<script>
const cars = [];
cars[0]= "Saab";
cars[1]= "Volvo";
cars[2]= "BMW";
document.getElementById("demo").innerHTML = cars;
</script>

</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p id="demo"></p>

<script>
const cars = new Array("Saab", "Volvo", "BMW");
document.getElementById("demo").innerHTML = cars;
</script>

</body>
</html>
```



Para acceder a un elemento, hay que usar su posición dentro del array. La primera posición es la 0.

```
var lenguajes = new Array("Java", "C++", "JavaScript");

alert(lenguajes[0]);//muestra Java

alert(lenguajes[2]);//muestra JavaScript

alert(lenguajes[4]);//no existe la posición, devuelve undefined
```



```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>JavaScript array elements are accessed using numeric indexes (starting from 0).</p>

<p id="demo"></p>

<script>
const cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars[0];
</script>

</body>
</html>
```

Cambio de un elemento de matriz

```
const cars = ["Saab", "Volvo", "BMW"];
cars[0] = "Opel";
```

JavaScript Arrays

JavaScript array elements are accessed using numeric indexes (starting from 0).

Saab



La propiedad de longitud

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let length = fruits.length;
```

Acceso al primer elemento del arreglo

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits[0];
```

Acceso al último elemento del arreglo

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits[fruits.length - 1];
```





```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p id="demo"></p>

<script>
const cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
</script>

</body>
</html>
```

JavaScript Arrays

Saab,Volvo,BMW



4.1.Array/arreglos_ Acceso a un array entero



```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>The best way to loop through an array is using a standard for loop:</p>

<p id="demo"></p>

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fLen = fruits.length;

let text = "<ul>";
for (let i = 0; i < fLen; i++) {
  text += "<li>" + fruits[i] + "</li>";
}
text += "</ul>";

document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

También puedes usar la **Array.forEach()** función:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>Array.forEach() calls a function for each array element.</p>

<p id="demo"></p>

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];

let text = "<ul>";
fruits.forEach(myFunction);
text += "</ul>";
document.getElementById("demo").innerHTML = text;

function myFunction(value) {
  text += "<li>" + value + "</li>";
}
</script>

</body>
</html>
```

4.1.Array/arreglos_ Añadir un elemento

Para añadir un elemento, basta con asignarlo a la posición que queramos, aunque no exista. Se crean también las posiciones intermedias necesarias, con valor undefined.

```
var lenguajes = new Array("Java", "C++", "JavaScript");
lenguajes[7]= "Lisp";//lenguajes pasa a tener 8 elementos
r= lenguajes[7];
alert(r);//muestra "Lisp"
r= lenguajes[5];
alert(r);//no muestra nada,r tiene valor undefined
```

4.8. Listas (arrays)_Acceso a los ...

4.1.Array/arreglos_ Añadir un elemento



Agregar elementos con índices altos puede crear "agujeros" indefinidos en una matriz:

```
const fruits = ["Banana", "Orange", "Apple"];
fruits[6] = "Lemon"; // Creates undefined "holes" in fruits
```

Si usa índices con nombre, JavaScript redefinirá la matriz en un objeto.

Después de eso, algunos métodos y propiedades de matriz producirán **resultados incorrectos**.

```
const person = [];
person["firstName"] = "John";
person["lastName"] = "Doe";
person["age"] = 46;
person.length; // Will return 0
person[0]; // Will return undefined
```

4.1.Array/arreglos_ Matrices asociativas



- Muchos lenguajes de programación admiten matrices con índices con nombre.
- Las matrices con índices con nombre se denominan matrices asociativas (o hashes).
- JavaScript no admite matrices con índices con nombre.
- En JavaScript, las matrices siempre usan índices numerados .

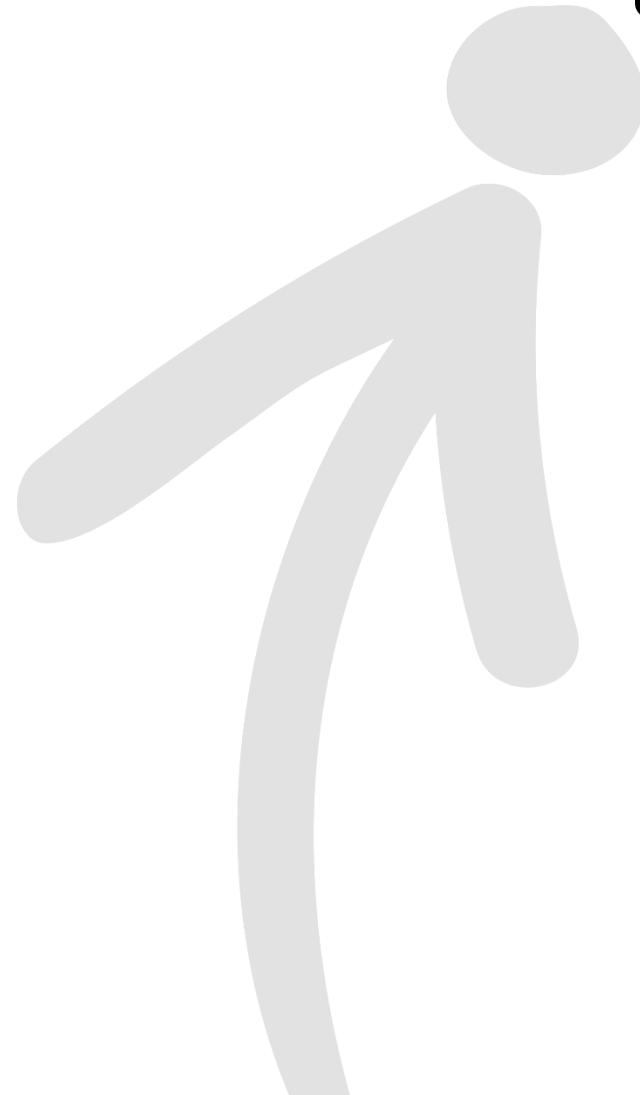
```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

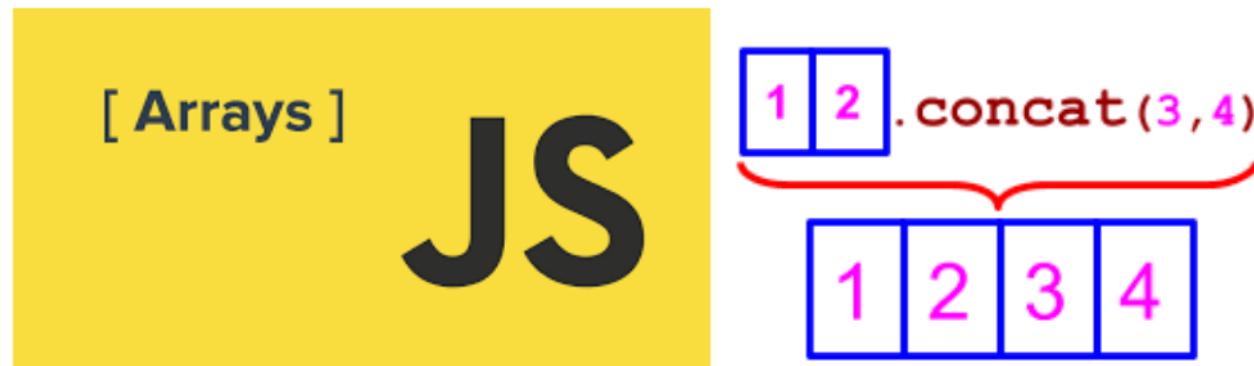
<p id="demo"></p>

<script>
const person = [];
person[0] = "John";
person[1] = "Doe";
person[2] = 46;
document.getElementById("demo").innerHTML =
person[0] + " " + person.length;
</script>

</body>
</html>
```



Como en el caso de las cadenas, JavaScript tiene un buen número de métodos predefinidos para trabajar con arrays.





El método de JavaScript **toString()** convierte una matriz en una cadena de valores de matriz (separados por comas).

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>
<h2>toString()</h2>
<p>The toString() method returns an array as a comma separated string:</p>

<p id="demo"></p>

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.toString();
</script>

</body>
</html>
```

JavaScript Array Methods

toString()

The **toString()** method returns an array as a comma separated string:

Banana,Orange,Apple,Mango



El **join()** método también une todos los elementos de la matriz en una cadena. Se comporta como `toString()`, pero además puedes especificar el separador:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>
<h2>join()</h2>
<p>The join() method joins array elements into a string.</p>
<p>In this example we have used " * " as a separator between the elements:</p>

<p id="demo"></p>

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.join(" * ");
</script>

</body>
</html>
```



JavaScript Array Methods

join()

The `join()` method joins array elements into a string.

In this example we have used " * " as a separator between the elements:

Banana * Orange * Apple * Mango



El **pop()** método elimina el último elemento de una matriz:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>
<h2>pop()</h2>
<p>The pop() method removes the last element from an array.</p>

<p id="demo1"></p>
<p id="demo2"></p>

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;
fruits.pop();
document.getElementById("demo2").innerHTML = fruits;
</script>

</body>
</html>
```

JavaScript Array Methods

pop()

The **pop()** method removes the last element from an array.

Banana,Orange,Apple,Mango

Banana,Orange,Apple

El **pop()** método devuelve el valor que "salió":

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits.pop();
```



El **push()** método agrega un nuevo elemento a una matriz (al final):

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>
<h2>push()</h2>
<p>The push() method appends a new element to an array:</p>

<p id="demo1"></p>
<p id="demo2"></p>

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;
fruits.push("Kiwi");
document.getElementById("demo2").innerHTML = fruits;
</script>

</body>
</html>
```

JavaScript Array Methods

push()

The **push()** method appends a new element to an array:

Banana,Orange,Apple,Mango

Banana,Orange,Apple,Mango,Kiwi

El **push()** método devuelve la nueva longitud de la matriz:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let length = fruits.push("Kiwi");
```



El **shift()** método elimina el primer elemento de la matriz y "cambia" todos los demás elementos a un índice más bajo.

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>
<h2>shift()</h2>
<p>The shift() method removes the first element of an array (and "shifts" the other elements to the left):</p>

<p id="demo1"></p>
<p id="demo2"></p>

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;
fruits.shift();
document.getElementById("demo2").innerHTML = fruits;
</script>

</body>
</html>
```

JavaScript Array Methods

shift()

The `shift()` method removes the first element of an array (and "shifts" the other elements to the left):

Banana,Orange,Apple,Mango

Orange,Apple,Mango

El **shift()** método devuelve el valor que fue "desplazado":

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits.shift();
```



El **unshift()** método agrega un nuevo elemento a una matriz (al principio) y "desplaza" los elementos más antiguos:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>
<h2>unshift()</h2>
<p>The unshift() method adds new elements to the beginning of an array:</p>

<p id="demo1"></p>
<p id="demo2"></p>

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;
fruits.unshift("Lemon");
document.getElementById("demo2").innerHTML = fruits;
</script>

</body>
</html>
```

JavaScript Array Methods

unshift()

The **unshift()** method adds new elements to the beginning of an array:

Banana,Orange,Apple,Mango

Lemon,Banana,Orange,Apple,Mango

El **unshift()** método devuelve la nueva longitud de la matriz.

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");
```



La **length** propiedad proporciona una manera fácil de agregar un nuevo elemento a una matriz:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>
<p>The length property provides an easy way to append new elements to an array without using the push() method:</p>

<p id="demo1"></p>
<p id="demo2"></p>

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;
fruits[fruits.length] = "Kiwi";
document.getElementById("demo2").innerHTML = fruits;
</script>

</body>
</html>
```



JavaScript Array Methods

The length property provides an easy way to append new elements to an array without using the push() method:

Banana,Orange,Apple,Mango

Banana,Orange,Apple,Mango,Kiwi

4.1.Array/arreglos_ Métodos del objeto Array

El **concat()** método crea una nueva matriz fusionando (concatenando) matrices existentes:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>
<h2>concat()</h2>
<p>The concat() method merges (concatenates) arrays:</p>

<p id="demo"></p>

<script>
const myGirls = ["Cecilie", "Lone"];
const myBoys = ["Emil", "Tobias", "Linus"];
const myChildren = myGirls.concat(myBoys);

document.getElementById("demo").innerHTML = myChildren;
</script>

</body>
</html>
```

JavaScript Array Methods

concat()

The concat() method merges (concatenates) arrays:

Cecilie,Lone,Emil,Tobias, Linus

El concat()método puede tomar cualquier número de argumentos de matriz:

```
const arr1 = ["Cecilie", "Lone"];
const arr2 = ["Emil", "Tobias", "Linus"];
const arr3 = ["Robin", "Morgan"];
const myChildren = arr1.concat(arr2, arr3);
```

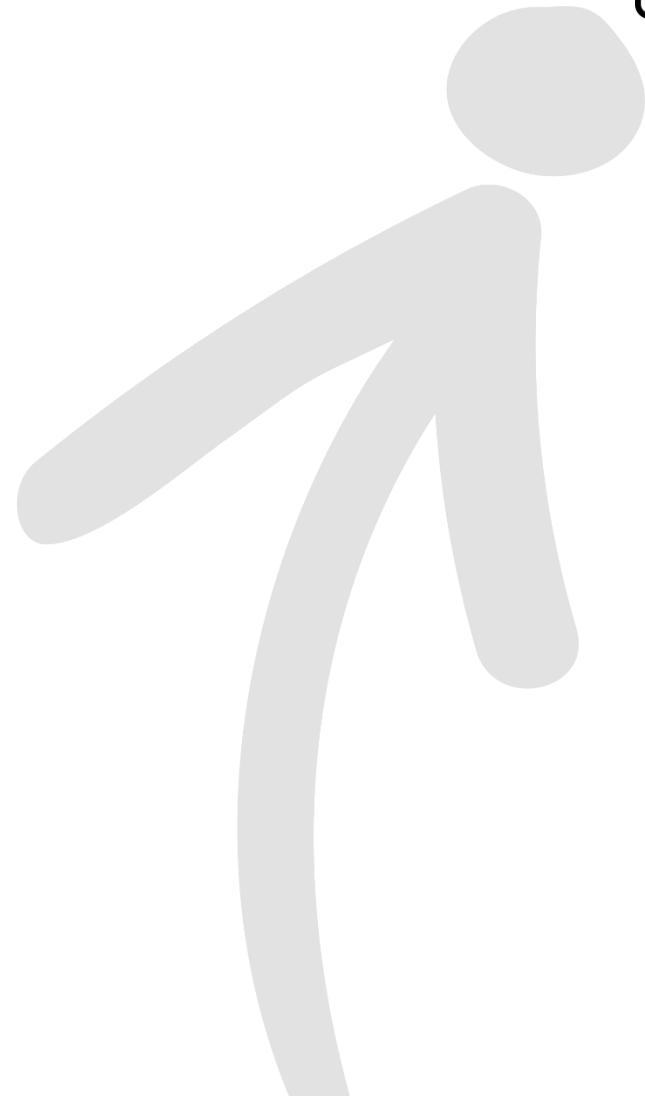
El concat()método también puede tomar cadenas como argumentos:

```
const arr1 = ["Emil", "Tobias", "Linus"];
const myChildren = arr1.concat("Peter");
```



El concat()método crea una nueva matriz fusionando (concatenando) matrices existentes:

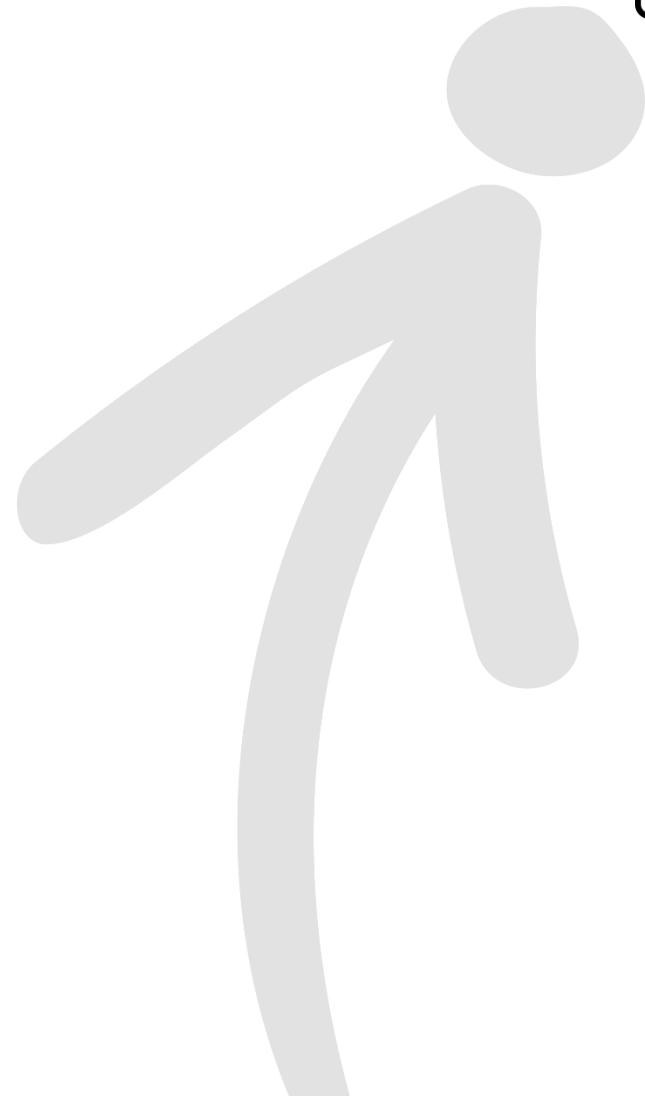
```
xxxxxxxxxxxxxxxxxxxx
```





El concat()método crea una nueva matriz fusionando (concatenando) matrices existentes:

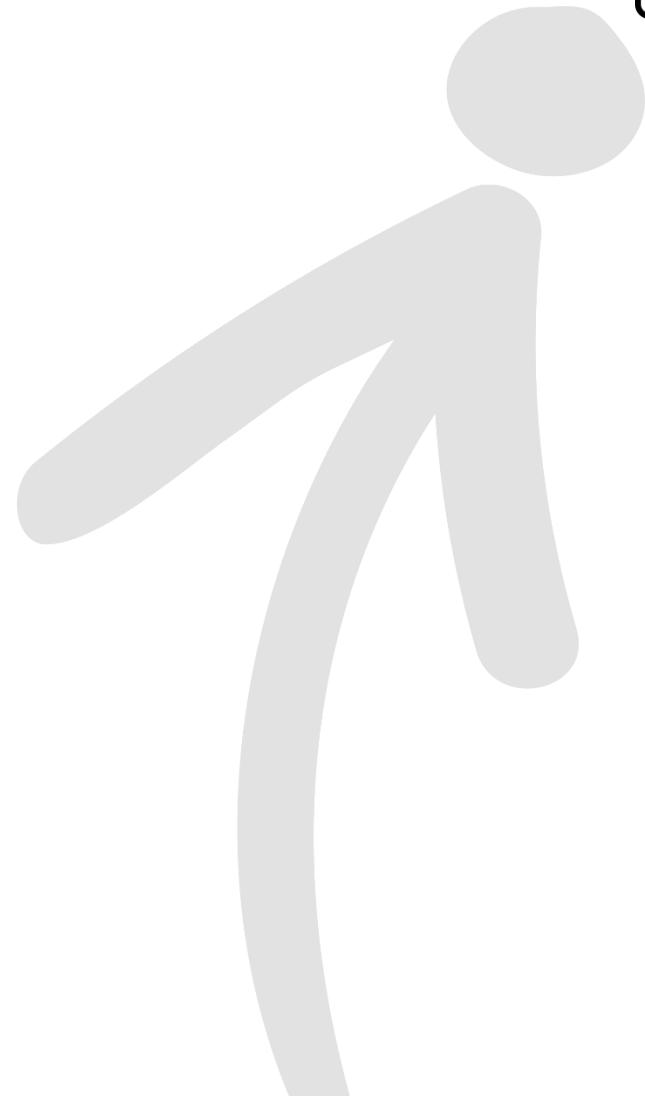
```
xxxxxxxxxxxxxxxxxxxx
```





El concat()método crea una nueva matriz fusionando (concatenando) matrices existentes:

```
xxxxxxxxxxxxxxxxxxxx
```





El **splice()** método se puede usar para agregar nuevos elementos a una matriz:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>
<h2>splice()</h2>
<p>The splice() method adds new elements to an array:</p>

<p id="demo1"></p>
<p id="demo2"></p>

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;

fruits.splice(2, 0, "Lemon", "Kiwi");
document.getElementById("demo2").innerHTML = fruits;
</script>

</body>
</html>
```

JavaScript Array Methods

splice()

The **splice()** method adds new elements to an array:

Banana,Orange,Apple,Mango

Banana,Orange,Lemon,Kiwi,Apple,Mango

El primer parámetro (2) define la **posición en** la que se deben **agregar** (empalmar) nuevos elementos.
El segundo parámetro (0) define **cuántos** elementos se deben **eliminar**.
El resto de parámetros ("Limón", "Kiwi") definen los nuevos **elementos a añadir**.
El **splice()** método devuelve una matriz con los elementos eliminados:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 2, "Lemon", "Kiwi");
```



Usando splice() para eliminar elementos

El **splice()** método se puede usar para agregar nuevos elementos a una matriz:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>
<h2>splice()</h2>
<p>The splice() methods can be used to remove array elements:</p>

<p id="demo1"></p>
<p id="demo2"></p>

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;
fruits.splice(0, 1);
document.getElementById("demo2").innerHTML = fruits;
</script>

</body>
</html>
```

JavaScript Array Methods

splice()

The **splice()** methods can be used to remove array elements:

Banana,Orange,Apple,Mango

Orange,Apple,Mango



El **slice()** método corta una parte de una matriz en una nueva matriz.

Este ejemplo corta una parte de una matriz a partir del elemento de matriz 1 ("Naranja"):

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>
<h2>slice()</h2>
<p>This example slices out a part of an array starting from array element 1 ("Orange"):</p>

<p id="demo"></p>

<script>
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const citrus = fruits.slice(1);
document.getElementById("demo").innerHTML = fruits + "<br><br>" + citrus;
</script>

</body>
</html>
```

JavaScript Array Methods

slice()

This example slices out a part of an array starting from array element 1 ("Orange"):

Banana,Orange,Lemon,Apple,Mango

Orange,Lemon,Apple,Mango

El **slice()** método puede tomar **dos argumentos** como **slice(1, 3)**. Luego, el método selecciona elementos desde el argumento inicial y hasta (pero sin incluir) el argumento final.

```
const fruits =
["Banana", "Orange", "Lemon", "Apple", "Mango"];
const citrus = fruits.slice(1, 3);
```

Si se omite el argumento final, como en los primeros ejemplos, el **slice()** método corta el resto de la matriz.

```
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const citrus = fruits.slice(2);
```





El **sort()**método ordena una matriz alfabéticamente:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Sort</h2>
<p>The sort() method sorts an array alphabetically:</p>

<p id="demo1"></p>
<p id="demo2"></p>

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;

fruits.sort();
document.getElementById("demo2").innerHTML = fruits;
</script>

</body>
</html>
```

JavaScript Array Sort

The `sort()` method sorts an array alphabetically:

Banana,Orange,Apple,Mango

Apple,Banana,Mango,Orange



El **reverse()** método invierte los elementos en una matriz.
Puede usarlo para ordenar una matriz en orden descendente:

JavaScript Array Sort Reverse

The **reverse()** method reverses the elements in an array.

By combining **sort()** and **reverse()** you can sort an array in descending order:

Banana,Orange,Apple,Mango

Orange,Mango,Banana,Apple

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Sort Reverse</h2>

<p>The reverse() method reverses the elements in an array.</p>
<p>By combining sort() and reverse() you can sort an array in descending order:</p>

<p id="demo1"></p>
<p id="demo2"></p>

<script>
// Create and display an array:
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;

// First sort the array
fruits.sort();

// Then reverse it:
fruits.reverse();

document.getElementById("demo2").innerHTML = fruits;
</script>

</body>
</html>
```

4.1.Array/arreglos_ Ordenar un array



Puede usar **Math.max.apply** para encontrar el número más alto en una matriz:

```
function myArrayMax(arr) {  
    return Math.max.apply(null, arr);  
}
```



Puede usar **Math.min.apply** para encontrar el número más bajo en una matriz:

```
function myArrayMin(arr) {  
    return }  
}
```

Ejemplo (Buscar Max)

```
function myArrayMax(arr) {  
    let len = arr.length;  
    let max = -Infinity;  
    while (len--) {  
        if (arr[len] > max) {  
            max = arr[len];  
        }  
    }  
    return max;  
}
```

Ejemplo (Buscar mínimo)

```
function myArrayMin(arr) {  
    let len = arr.length;  
    let min = Infinity;  
    while (len--) {  
        if (arr[len] < min) {  
            min = arr[len];  
        }  
    }  
    return min;  
}
```



De forma predeterminada, la sort() función ordena los valores como **cadenas** .

Esto funciona bien para cadenas ("Apple" viene antes de "Banana").

Sin embargo, si los números se ordenan como cadenas, "25" es mayor que "100", porque "2" es mayor que "1".

Debido a esto, el sort() método producirá un resultado incorrecto al ordenar números.

Puede solucionar esto proporcionando una función de comparación :

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Sort</h2>
<p>Sort the array in ascending order:</p>

<p id="demo1"></p>
<p id="demo2"></p>

<script>
const points = [40, 100, 1, 5, 25, 10];
document.getElementById("demo1").innerHTML = points;

points.sort(function(a, b){return a - b});//función anónima
document.getElementById("demo2").innerHTML = points;
</script>

</body>
</html>
```

JavaScript Array Sort

Sort the array in ascending order:

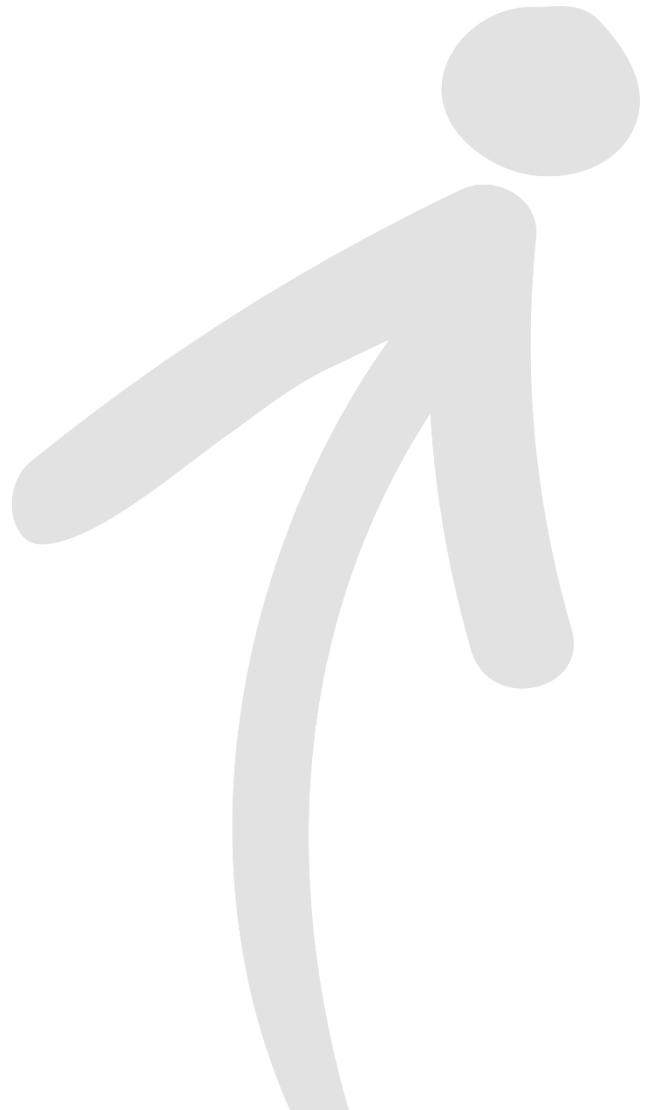
40,100,1,5,25,10

1,5,10,25,40,100

4.1.Array/arreglos_ Ordenar un array. La función de comparación

- El propósito de la función de comparación es definir un orden de clasificación alternativo.
- La función de comparación debe devolver un valor negativo, cero o positivo, según los argumentos:
- Cuando la sort()función compara dos valores, envía los valores a la función de comparación y ordena los valores según el valor devuelto (negativo, cero, positivo).
- Si el resultado es negativo ase ordena antes b.
- Si el resultado es positivo bse ordena antes a.
- Si el resultado es 0, no se realizan cambios con el orden de clasificación de los dos valores.





4.1.Array/arreglos_ Iteración de un array



El **forEach()** método llama a una función (una función de devolución de llamada) una vez para cada elemento de la matriz.

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array.forEach()</h2>
<p>Calls a function once for each array element.</p>

<p id="demo"></p>

<script>
const numbers = [45, 4, 9, 16, 25];

let txt = "";
numbers.forEach(myFunction);
document.getElementById("demo").innerHTML = txt;

function myFunction(value, index, array) {
  txt += value + "<br>";
}
</script>

</body>
</html>
```

JavaScript Array.forEach()

Calls a function once for each array element.

```
45
4
9
16
25
```

```
const numbers = [45, 4, 9, 16, 25];
let txt = "";
numbers.forEach(myFunction);

function myFunction(value, index, array) {
  txt += value + "<br>";
}
```



El **map()**método crea una nueva matriz realizando una función en cada elemento de la matriz.

El map()método no ejecuta la función para elementos de matriz sin valores.

El map()método no cambia la matriz original.

Este ejemplo multiplica cada valor de matriz por 2:

JavaScript Array.map()

Creates a new array by performing a function on each array element.

90,8,18,32,50

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array.map()</h2>
<p>Creates a new array by performing a function on each array
element.</p>

<p id="demo"></p>

<script>
const numbers1 = [45, 4, 9, 16, 25];
const numbers2 = numbers1.map(myFunction);

document.getElementById("demo").innerHTML = numbers2;

function myFunction(value, index, array) {
    return value * 2;
}
</script>

</body>
</html>
```



El **filter()** método crea una nueva matriz con elementos de matriz que pasa una prueba.

Este ejemplo crea una nueva matriz a partir de elementos con un valor superior a 18:

JavaScript Array.filter()

Creates a new array with all array elements that passes a test.

45,25

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array.filter()</h2>
<p>Creates a new array with all array elements that passes a test.</p>

<p id="demo"></p>

<script>
const numbers = [45, 4, 9, 16, 25];
const over18 = numbers.filter(myFunction);

document.getElementById("demo").innerHTML = over18;

function myFunction(value, index, array) {
  return value > 18;
}
</script>

</body>
</html>
```



El **reduce()**método ejecuta una función en cada elemento de la matriz para producir (reducirlo a) un solo valor.

El reduce()método funciona de izquierda a derecha en la matriz. Ver reduceRight()también

El reduce()método no reduce la matriz original.

Este ejemplo encuentra la suma de todos los números en una matriz:

JavaScript Array.reduce()

This example finds the sum of all numbers in an array:

The sum is 99

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array.reduce()</h2>
<p>This example finds the sum of all numbers in an array:</p>

<p id="demo"></p>

<script>
const numbers = [45, 4, 9, 16, 25];
let sum = numbers.reduce(myFunction);

document.getElementById("demo").innerHTML = "The sum is " + sum;

function myFunction(total, value, index, array) {
  return total + value;
}
</script>

</body>
</html>
```



El **reduceRight()** método ejecuta una función en cada elemento de la matriz para producir (reducirlo a) un solo valor.

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array.reduceRight()</h2>
<p>This example finds the sum of all numbers in an array:</p>

<p id="demo"></p>

<script>
const numbers = [45, 4, 9, 16, 25];
let sum = numbers.reduceRight(myFunction);

document.getElementById("demo").innerHTML = "The sum is " + sum;

function myFunction(total, value, index, array) {
  return total + value;
}
</script>

</body>
</html>
```

JavaScript Array.reduceRight()

This example finds the sum of all numbers in an array:

The sum is 99





El **every()** método verifica si todos los valores de la matriz pasan una prueba.
Este ejemplo verifica si todos los valores de la matriz son mayores que 18:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array.every()</h2>
<p>The every() method checks if all array values pass a test.</p>

<p id="demo"></p>

<script>
const numbers = [45, 4, 9, 16, 25];
let allOver18 = numbers.every(myFunction);

document.getElementById("demo").innerHTML = "All over 18 is " +
allOver18;

function myFunction(value, index, array) {
  return value > 18;
}
</script>

</body>
</html>
```

JavaScript Array.every()

The `every()` method checks if all array values pass a test.

All over 18 is false



El **some()** método verifica si algunos valores de matriz pasan una prueba.
Este ejemplo verifica si algunos valores de matriz son mayores que 18:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array.some()</h2>
<p>The some() method checks if some array values pass a test.</p>

<p id="demo"></p>

<script>
const numbers = [45, 4, 9, 16, 25];
let someOver18 = numbers.some(myFunction);

document.getElementById("demo").innerHTML = "Some over 18 is " +
someOver18;

function myFunction(value, index, array) {
  return value > 18;
}
</script>

</body>
</html>
```

JavaScript Array.some()

The **some()** method checks if some array values pass a test.
Some over 18 is true



El **indexOf()** método busca en una matriz el valor de un elemento y devuelve su posición.

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array.indexOf()</h2>

<p id="demo"></p>

<script>
const fruits = ["Apple", "Orange", "Apple", "Mango"];
let position = fruits.indexOf("Apple") + 1;

document.getElementById("demo").innerHTML = "Apple is found in
position " + position;
</script>

</body>
</html>
```

JavaScript Array.indexOf()

Apple is found in position 1





El **find()**método devuelve el valor del primer elemento de la matriz que pasa una función de prueba. Este ejemplo encuentra (devuelve el valor de) el primer elemento que es mayor que 18:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array.find()</h2>
<p id="demo"></p>

<script>
const numbers = [4, 9, 16, 25, 29];
let first = numbers.find(myFunction);

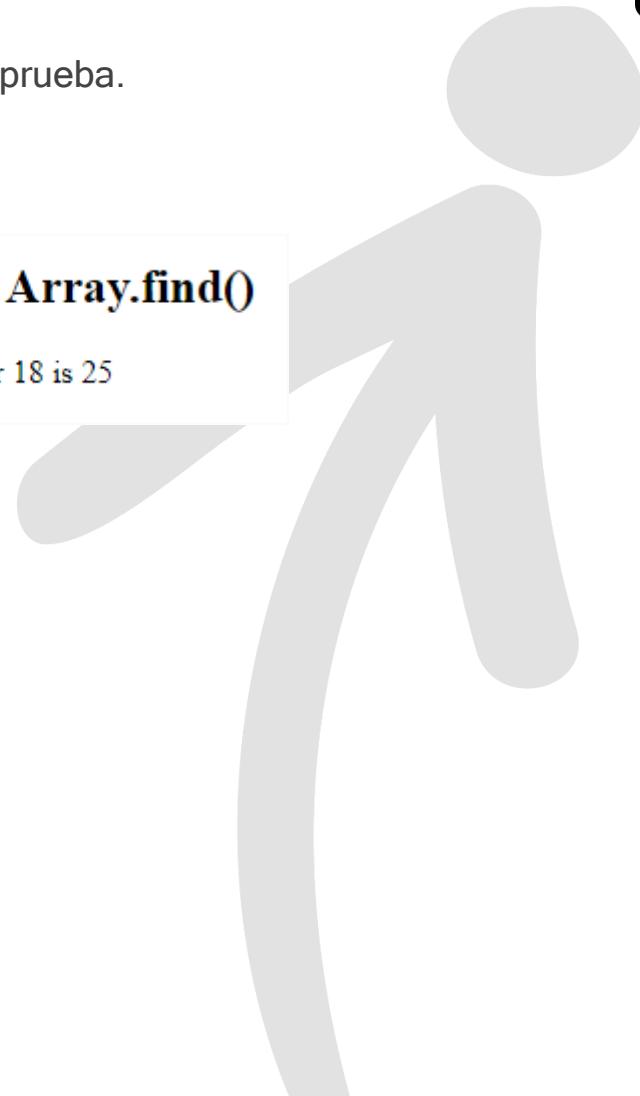
document.getElementById("demo").innerHTML = "First number over 18 is
" + first;

function myFunction(value, index, array) {
  return value > 18;
}
</script>

</body>
</html>
```

JavaScript Array.find()

First number over 18 is 25





El **Array.from()** método devuelve un objeto Array de cualquier objeto con una propiedad de longitud o cualquier objeto iterable.

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>The Array.from() method returns an Array object from any object with a length property or any iterable object.</p>

<p id="demo"></p>

<script>
const myArr = Array.from("ABCDEFG");
document.getElementById("demo").innerHTML = myArr;
</script>

<p>The Array.from() method is not supported in Internet Explorer.</p>

</body>
</html>
```

JavaScript Arrays

The `Array.from()` method returns an `Array` object from any object with a `length` property or any iterable object.

A,B,C,D,E,F,G

The `Array.from()` method is not supported in Internet Explorer.



El **entries()** método devuelve un objeto Array Iterator con pares clave/valor:

```
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Arrays</h1>
<h2>The entries() method</h2>

<p>entries() returns an Array Iterator object with key/value pairs:</p>

<p id="demo"></p>

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
const f = fruits.entries();

for (let x of f) {
  document.getElementById("demo").innerHTML += x + "<br>";
}
</script>

<p>The entries() method is not supported in Internet Explorer 11 (or earlier).</p>

</body>
</html>
```

JavaScript Arrays

The entries() method

entries() returns an Array Iterator object with key/value pairs:

0,Banana
1,Orange
2,Apple
3,Mango

The entries() method is not supported in Internet Explorer 11 (or earlier).



ECMAScript 2016 introducido **Array.includes()** a las matrices. Esto nos permite verificar si un elemento está presente en una matriz (incluido NaN, a diferencia de indexOf).

```
<!DOCTYPE html>
<html>
<body>

<h1>Array includes()</h1>

<p>Check if the fruit array contains "Mango":</p>

<p id="demo"></p>

<p><strong>Note:</strong> The includes method is not supported in Edge 13 (and earlier versions).</p>

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML =
fruits.includes("Mango");
</script>

</body>
</html>
```

Array includes()

Check if the fruit array contains "Mango":

true

Note: The includes method is not supported in Edge 13 (and earlier versions).

4.1.Array/arreglos_ Const array



en 2015, JavaScript introdujo una nueva palabra clave importante: **const**.
Se ha convertido en una práctica común declarar matrices usando const:

```
const cars = ["Saab", "Volvo", "BMW"];
```

No se puede reasignar

```
const cars = ["Saab", "Volvo", "BMW"];
cars = ["Toyota", "Volvo", "Audi"]; // ERROR
```

Las matrices no son constantes

La palabra clave **const** es un poco engañoso.
NO define una matriz constante. Define una referencia constante a una matriz.
Debido a esto, aún podemos cambiar los elementos de una matriz constante.

Los elementos se pueden reasignar

```
// You can create a constant array:
const cars = ["Saab", "Volvo", "BMW"];

// You can change an element:
cars[0] = "Toyota";

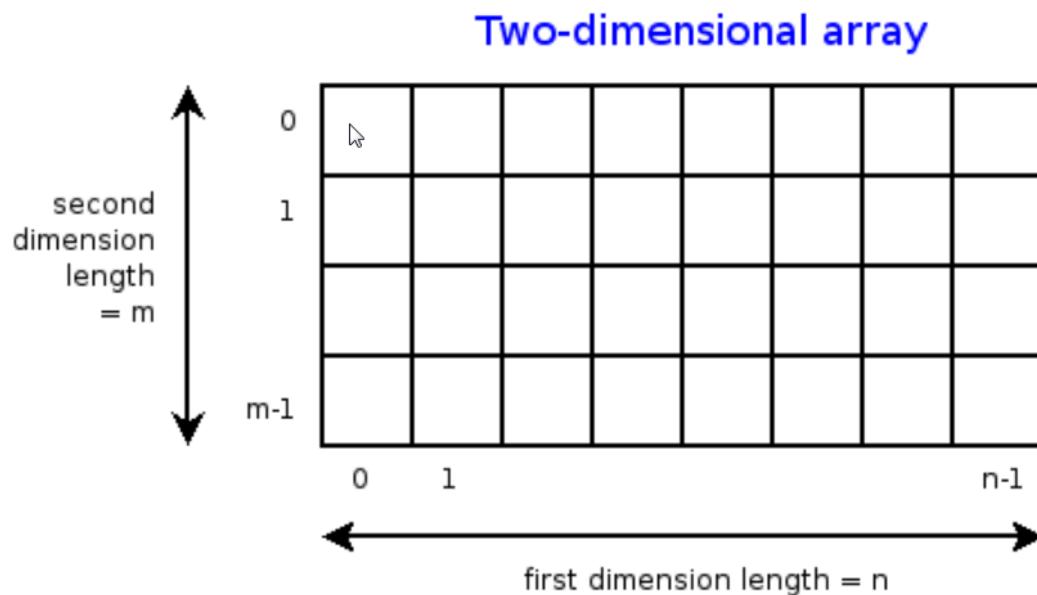
// You can add an element:
cars.push("Audi");
```



4.1.Array/arreglos_Arrays multidimensionales

Un array de dos dimensiones no es más que un array dentro de otro, es decir, un array de arrays.

Si declaramos un array de 10 posiciones y en cada posición guardamos un array de 10 posiciones obtenemos un array bidimensional de 10×10 , similar a una tabla de 10 filas y 10 columnas.



4.1.Array/arreglos_Arrays multidimensionales

Las matrices se pueden declarar usando literales, como en el siguiente fragmento de código:

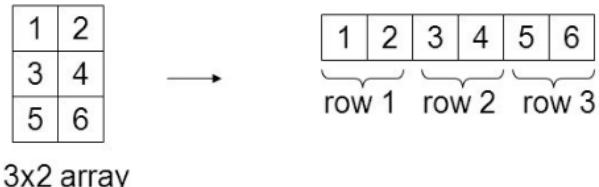
```
var matriz=[[1,2],[3,4],[5,6]];
alert(matriz[2][1]);//muestra el 6
alert(matriz[1][1]);//muestra el 4
alert(matriz[0][0]); // muestra el 1
```

Para acceder a un elemento en una matriz hay que usar dos índices.

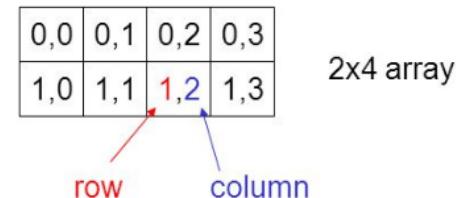
El primero indica la fila y el segundo la columna.

Multidimensional arrays

- ¿Cómo se posiciona en memoria?



- ¿Cómo se define su posición?



row = fila

column = columna

6

4.1.Array/arreglos_Arrays multidimensionales

También es posible crearlas usando el constructor Array().

El siguiente ejemplo crearía una matriz de tres filas y dos columnas.

Primero crea un array de tres posiciones y luego asigna un array con dos elementos a cada una de esas posiciones.

```
var filas = 3;
var columnas = 2;
var matriz= new Array(filas);
for(var i= 0; i< matriz.length;i++){
    matriz[i]= new Array(columnas);
```

Multidimensional arrays

- ¿Cómo se posiciona en memoria?

1 2	→	1 2 3 4 5 6
3 4		<u>row 1</u> <u>row 2</u> <u>row 3</u>
5 6		

3x2 array

- ¿Cómo se define su posición?

0,0 0,1 0,2 0,3	
1,0 1,1 1,2 1,3	

2x4 array

row

column

row = fila

column = columna

6

Arrays multidimensionales

Veamos un ejemplo gráfico para entenderlo mejor. Declaramos un array de 2×2 en el que introduciremos números

TABLA	COLUMNA 0	COLUMNA 1
FILA 0	posición 0,0 = 25	posición 0,1 = 12
FILA 1	posición 1,0 = 34	posición 1,1 = 6

//Declaramos el array bidimensional

```
var nuevoArray = new Array(2);  
nuevoArray[0] = new Array(2);  
nuevoArray[1] = new Array(2);
```

//Metemos un dato en cada posición

```
nuevoArray[0][0] = 25;  
nuevoArray[0][1] = 12;  
nuevoArray[1][0] = 34;  
nuevoArray[1][1] = 6;
```

4.1.Array/arreglos_Arrays multidimensionales

Aunque se pueden crear arrays bidimensionales de manera manual como en el ejemplo anterior, lo normal es hacerlo mediante bucles.

Veamos un ejemplo de un array bidimensional de 10 x 10

//Declaración del array de 10 posiciones

```
var nuevoArray = new Array(10);
```

//Bucle para meter en cada posición otros array de 10

```
for(var i=0; i<10; i++) {
    nuevoArray[i] = new Array(10);
}
```

4.1.Array/arreglos_Arrays multidimensionales

Si para recorrer un array de una posición utilizábamos un bucle for, haremos lo mismo pero usando dos bucles for, uno dentro de otro.

//Bucle que recorre el primer array

```
for(var i=0; i<10; i++) {
```

//Bucle que recorre el array que está en la posición i

```
    for(var j=0; j<10; j++) {
```

```
        document.write(nuevoArray[i][j]);
```

```
    }
```

4.1.Array/arreglos_Arrays multidimensionales

En el ejemplo anterior sabemos que se trata de un array bidimensional de 10×10 , pero puedo que no conozcamos el tamaño del array.

Veamos un ejemplo utilizando el método length

//Bucle que recorre el primer array

```
for(var i=0; i<nuevoArray.length; i++) {
```

//Bucle que recorre el array que está en la posición i

```
    for(var j=0; j<nuevoArray[i].length; j++) {  
        document.write(nuevoArray[i][j]);  
    }  
}
```

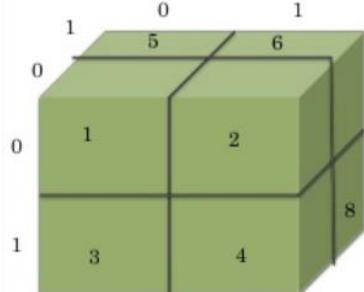
4.1.Array/arreglos_Arrays multidimensionales

Arrays multidimensionales

Puede darse el caso que necesitemos utilizar un array de 3 dimensiones o más.

En ese caso la estructura sigue siendo la misma

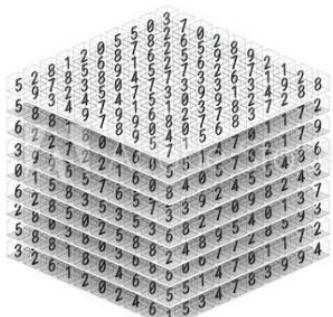
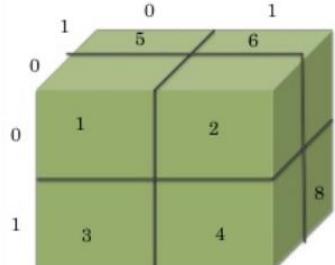
Arrays de más de dos dimensiones



```
///Declaramos el primer array
var nuevoArray = new Array(10);
//Metemos un array en cada posición
for(var i=0; i<nuevoArray.length; i++) {
    //nuevoArray[i] = new Array(10);
}
//Volvemos a recorrer los arrays para la
3ª dimensión
//Recorremos el primer array
for(var i=0; i<nuevoArray.length; i++) {
    //Recorremos el array de cada
    posición i
    for(var j=0; j<nuevoArray[i].length; j++)
    {
        //Creamos un array en cada
        posición
        nuevoArray[i][j] = new Array(10);
    }
}
```

Arrays multidimensionales

Para recorrer el array anterior de 3 dimensiones haríamos lo siguiente



2 dimensiones

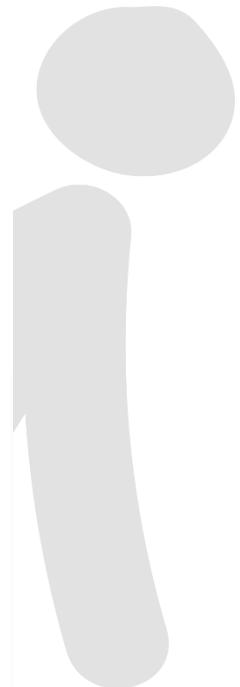
```
//Recorremos el primer array  
for(var i=0; i<nuevoArray.length; i++) {  
    //Recorremos el array de cada  
    //posición i  
    for(var j=0; j<nuevoArray[i].length; j++) {  
        //Recorremos el array de cada  
        //posición i j  
        for(var k=0; k<nuevoArray[i][j].length; k++) {  
  
            document.write(nuevoArray[i][j][k]);  
        }  
    }  
}
```

Ejemplo práctico 3 dimensiones

Vamos a crear una agenda de 1 año, en cada año metemos 12 meses, en cada mes 31 días y en cada día 24 horas. en cada hora podemos meter un evento.

Necesitamos un array de 3 dimensiones, es decir, un array para los 12 meses, en cada mes un array de 31 días, y en cada día un array de 24 horas.

```
//Declaramos el array
var agenda = new Array(12);
//Vamos a recorrerlo para meter en cada
//posición un array de 31
for(var i=0; i<agenda.length; i++) {
    agenda[i] = new Array(31);
    //Lo recorremos para meter en cada
    //posición un array de 24
    for(var j=0; j<agenda[i].length; j++) {
        agenda[i][j] = new Array(24);
    }
}
```



4.1.Array/arreglos_Arrays multidimensionales

Podemos crear eventos y guardarlos en su posición. Meses de 1 al 12, Días del 1 al 31 y horas de 0 a 23.

Hay que tener en cuenta que si creamos un array de 12 posiciones para los meses, lo que creamos es un array que va desde 0 hasta 11, y pasa lo mismo con el de los días, que irá desde 0 hasta 30.

Entonces a la hora de introducir los datos vamos a tener que restarle 1 a los meses y a los días

```
//Metiendo datos en la agenda  
Vamos a meter (Concierto, mes 4, día 23,  
hora 19)  
agenda[3][22][18] = "Concierto";  
//Tenemos que restarle 1 al mes y al día
```

```
//Queremos saber que evento hay el mes  
6, día 3 a las 22 horas  
var evento = agenda[5][2][22];  
//Igual que antes tenemos que restar 1 al  
mes y al día
```

**Barcelona**

Francesc Tàrrega 14
08027 Barcelona
93 351 78 00

Madrid

Campanar 12
28028 Madrid
91 502 13 40

Reus

Alcalde Joan Bertran 34-38
43202 Reus
977 31 24 36

info@grupcief.com

www.grupcief.com

