TEMA 5. ARRAYS

1. INTRODUCCIÓN.

En los lenguajes de programación existen estructuras de datos especiales, que nos sirven para guardar información más compleja que una variable simple.

Existen diferentes de estructuras de datos que se pueden utilizar para almacenar datos, pero una estructura de las más utilizadas en todos los lenguajes son los arrays. Un array es una estructura de datos que nos permite trabajar **con un conjunto de datos** al mismo tiempo que identificamos por la posición que ocupan dentro del array.

Los arrays nos permiten guardar un gran número de datos y acceder a ellos de manera independiente. Cada elemento es referenciado por la posición que ocupa dentro del array. Dichas posiciones se llaman **índices** y siempre son correlativos.

Se puede considerar que todos los arrays son de una dimensión: la dimensión principal, pero los elementos de dicha fila pueden a su vez contener otros arrays o matrices, lo que nos permitiría hablar de **arrays multidimensionales.**

2. ARRAYS.

JavaScript nos permite almacenar diferentes informaciones en cada posición de un array, lo normal es que todos los datos que almacenemos en un array sean del mismo tipo, ya que esto nos permite tratar toda la información de la misma forma y nos va a facilitar el trabajo.

A medida que vayas diseñando tu aplicación, tendrás que identificar las pistas que te permitan utilizar arrays para almacenar datos. Por ejemplo, imagínate que tienes que almacenar un montón de coordenadas geográficas de una ruta, esto sí que sería un buen candidato para emplear una estructura de datos de tipo array, ya que podríamos asignar un valor a cada posición del array, realizar cálculos, ordenar los puntos, etc. Siempre que veas similitudes en los datos o frecuencia de uso de datos, será una buena opción para usar un array.

Por ejemplo, si tenemos las siguientes variables:

```
let coche1="Seat";
let coche2="BMW";
let coche3="Audi";
let coche4="Toyota";
```

Este ejemplo sería un buen candidato para convertirlo en un array, ya que te permitiría introducir más marcas de coche, sin que tengas que crear nuevas variables para ello. Lo haríamos del siguiente modo:

```
let misCoches=[];
misCoches[0]="Seat";
misCoches[1]="BMW";
misCoches[2]="Audi";
misCoches[3]="Toyota";
```

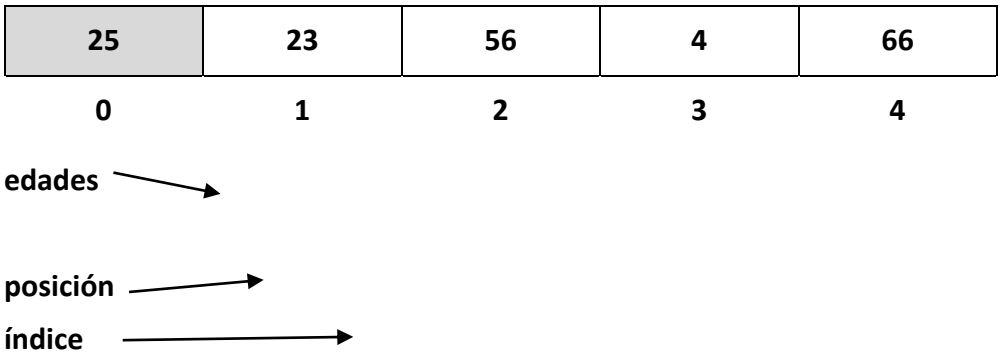
También podemos añadir datos en la declaración del array, es decir, podemos inicializarlos. Para el array que hemos creado anteriormente el resultado siguiente sería el mismo.

```
let misCoches = ["Seat", “BMW", "Audi", "Toyota"];
```

Si tenemos el siguiente array

```
let edades = [25, 23, 56, 4, 66]
```

1ª posición	2ª posición	3ª posición	4ª posición	5ª posición



Para acceder al contenido lo haremos con el identificador del array y la posición:

identificador[posición];

edades[0] es la primera posición y contiene 25.

edades[1] es la segunda posición y contiene 23.

edades[2] es la tercera posición y contiene 56.

edades[3] es la cuarta posición y contiene 4.

edades[4] es la quinta posición y contiene 66.

Los arrays en JavaScript se comportan como listas, es decir, podemos añadir y eliminar nuevas posiciones al array en tiempo de ejecución. Esto quiere decir que no es una estructura de datos estática, como tradicionalmente se han tratado los arrays.

2.1. CREACIÓN DE UN ARRAY.

Existen dos formas de crear un Array como un objeto y con los corchetes, de las dos formas conseguimos los mismo y podemos realizar las mismas operaciones.

2.1.1. CREAR UN ARRAY COMO UN OBJETO

Podemos crear un array, pero sin ninguna posición, es decir, tenemos un array vacío:

```
let nombres =  new Array();
```

Podemos crear un array con un tamaño, pero cada una de las posiciones estará vacía:

```
let nombres = new Array(10);
```

Y podemos crear un array con un tamaño y con un valor en cada posición, el número de elemento que será el tamaño y los elementos en contenido:

```
let nombres = new Array("Cristina", "María", "Miguel");
```

En este caso estamos creando un array de tres posiciones, donde cada una almacena una cadena. Los datos deben ir separados por comas.

2.1.2. CREAR UN ARRAY CON CORCHETES

Veamos la segunda forma de crear un array, que es a través de los corchetes, este método es el más utilizado actualmente.

Podemos crear un array vacío:

```
var nombres = [];
```

Y podemos crear un array con elementos:

```
let nombres = ["Cristina", "María", "Miguel"];
```

En este caso estamos creando un array de tres posiciones, donde cada una almacena una cadena. Los datos deben ir separados por comas.

Lo arrays en JavaScript son objetos que disponen de unas propiedades y métodos propios, una de las propiedades nos indica su longitud. Esta propiedad es **length** (longitud, que será 0 para un array vacío).

A diferencia de otros lenguajes de programación, en los que hacer el dimensionamiento previo del array tiene ventajas, en JavaScript no nos dará ningún tipo de ventaja especial, ya que podremos asignar un valor a cualquier posición del array en cualquier momento, esté o no definida previamente. La propiedad **length** se ajustará automáticamente al nuevo tamaño del array. Por ejemplo podríamos hacer una asignación tal como personas[53] y eso que nuestro array es de 40 posiciones. En este caso no ocurrirá ningún problema, ya que la propiedad **length** del array se ajustará automáticamente para poder almacenar la posición 53, con lo que la nueva longitud del array será 54, ya que la primera posición del array es la [0] y la última es la [53], tendremos por lo tanto 54 elementos en el array.

```
nombres[53] = "Gloria";
longitud = nombre.length; // asignará a la variable longitud el valor 54
```

2.2. ESCRIBIR Y LEER EN UN ARRAY

Introducir o leer datos en un array, es tan simple como crear una serie de sentencias de asignación o lectura por cada elemento del array.

```
let sistemaSolar = [];
sistemaSolar[0] = "Mercurio";
sistemaSolar[1] = "Venus";
sistemaSolar[2] = "Tierra";
sistemaSolar[3] = "Marte";
sistemaSolar[4] = "Júpiter";
 sistemaSolar[5] = "Saturno";
sistemaSolar[6] = "Urano";
sistemaSolar[7] = "Neptuno";
```

Si queremos leer un valor de nuestro array

```
unPlaneta = sistemaSolar[2]; // almacenará en un Planeta la cadena "Tierra".
```

Una vez que hemos leído un valor de un array, ese valor se comporta como si fuera un variable del tipo que hemos leído, lo que cambia es la sintaxis para acceder al dato, pero una vez que hemos accedido, el dato se comporta en función de su tipo.

Si queremos escribir en el array:

```
SistemaSolar[2] = “Tierra ” + SistemaSolar[2];
```

Si mostramos el contenido de todo el array tendríamos:

Mercurio", "Venus", "Tierra Tierra", "Marte", "Júpiter", "Saturno", "Urano", "Neptuno"

2.3. RECORRIDO DE UN ARRAY.

Existen múltiples formas de recorrer un array para mostrar sus datos. Veamos algunos ejemplos con el array del sistema Solar:

Cuando tenemos que recorrer todas las posiciones de un array la estructura de control adecuada es un proceso repetitivo por contador, un for, for..of o bien un for..in.

Empleando un bucle for:

```
for (let i=0;i<sistemaSolar.length;i++)
```

```
console.log(sistemaSolar[i]);
```

Empleando un bucle for of:

```
for (let planeta of sistemaSolar)
console.log(planeta);
```

Empleando un bucle for in:

```
for (let  indice in sistemaSolar)
console.log(sistemaSolar[indice]);
```

Los tres casos anteriores son análogos y vamos a obtener el mismo resultado con los tres.

También podemos utilizar el método forEach() que pertenece a los arrays:

```
sistemaSolar.forEach(planeta=> {
   console.log(planeta);
})
```

Con un forEach() podemos acceder tanto al elemento, como hemos hecho en el ejemplo anterior, como al índice de la posición:

```
sistemaSolar.forEach((planeta, indice)=> {
   console.log((indice + 1) + "._"planeta);
})
```

Podemos realizar más operaciones sobre un array, como una búsqueda, ordenación, … aunque estas operaciones las podemos realizar de nosotros vamos a utilizar métodos de los arrays, ya que son más rápidos y eficientes.

3. PROPIEDADES Y MÉTODOS.

Propiedades del objeto array:

Propiedad	Descripción
length	Devuelve el número de elementos en un array.

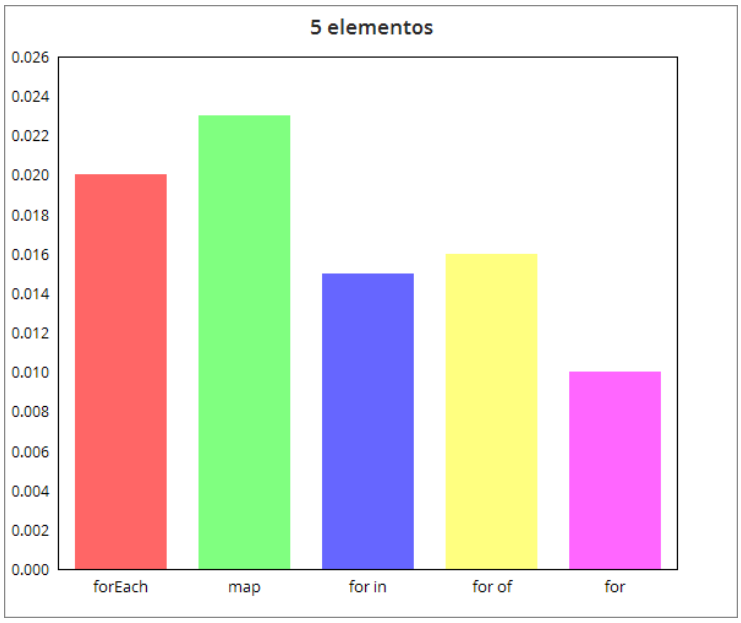
Métodos de array:

Métodos	Descripción
isArray(variable)	Nos permite saber si una variable es un array, devuelve un valor booleano.
concat()	Une dos o más arrays, y devuelve una copia de los arrays unidos.
join("separador")	Une todos los elementos de un array en una cadena de texto, separados por el separador indicado.
includes(valor)	Nos permite saber si una cadena contiene un valor, un substring de dicha cadena. Nos devuelve un valor booleano.
pop()	Elimina el último elemento de un array y devuelve ese elemento.
push(element1, element2,...)	Añade nuevos elementos al final de un array, y devuelve la nueva longitud.
shift()	Elimina el primer elemento de un array, y devuelve ese elemento.
unshift(element1, element2,...)	Añade nuevos elementos al comienzo de un array, y devuelve la nueva longitud.

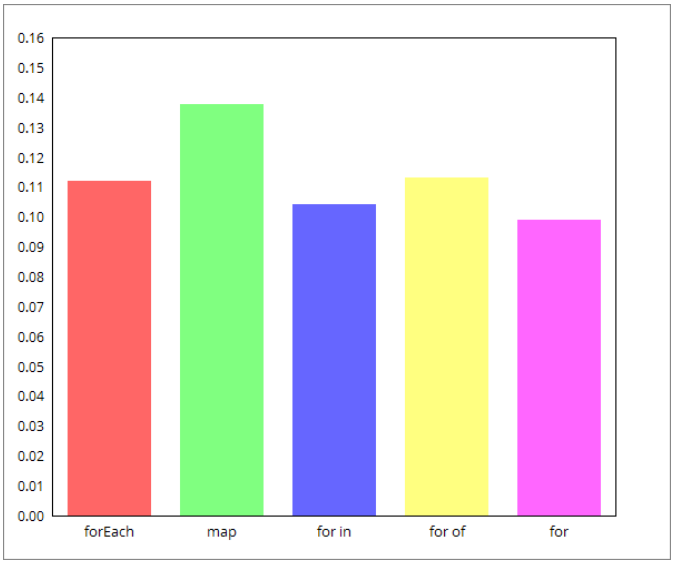
slice(a,b)	Extrae elementos de un array desde el índice a hasta el índice b. Si no existe b lo hace desde a hasta el final, si no existe ni a ni b hace una copia del original.
sort([callback])	Ordena los elementos de un array alfabéticamente (valor Unicode), si le pasamos un callback los ordena en función del algoritmo que le pasemos.
splice()	Cambia el contenido de un array eliminando elementos existentes y/o agregando nuevos elementos. a - Índice de inicio b - Número de elementos (opcional) ítems - Elementos a añadir en el caso de que se añadan. (opcional)
toString()	Convierte un array a una cadena y devuelve el resultado.
reverse()	Invierte el orden de los elementos en un array.
indexOf()	Devuelve el primer índice del elemento que coincida con el valor especificado, o -1 si ninguno es encontrado.
lastIndexOf()	Devuelve el último índice del elemento que coincida con el valor especificado, o -1 si ninguno es encontrado.
from(iterable)	Convierte en array el elemento iterable
some(callback)	Comprueba si al menos un elemento del array cumple la condición
every(callback)	Comprueba si todos los elementos del array cumplen la condición
map(callback)	Transforma todos los elementos del array y devuelve un nuevo array
filter(callback)	Filtra todos los elementos del array que cumplan la condición y devuelve un nuevo array
reduce(callback)	Reduce todos los elementos del array a un único valor

4. RENDIMIENTOS

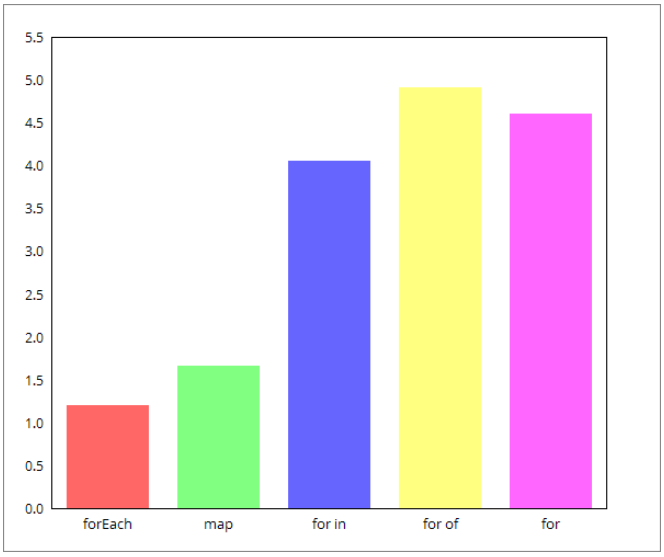
El rendimiento de los procesos repetitivos varía en función de las veces que se repita el proceso:



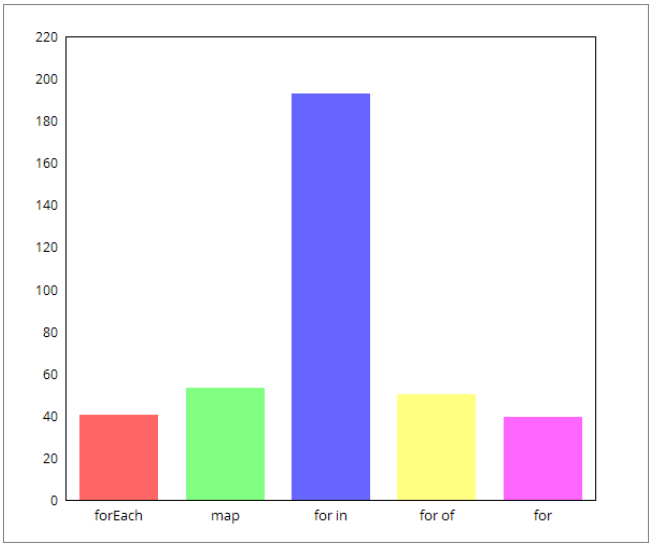
5 elementos | milisegundo



100 elementos | milisegundo



10.000 elementos | milisegundo



1.000.000 elementos | milisegundo

5. ARRAYS PARALELOS.

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

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

Por ejemplo, consideremos los siguientes arrays:

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

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

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

```
let profesores = ["Cristina","Catalina","Vieites","Benjamin"];
let asignaturas=["Seguridad","Bases de Datos","Sistemas Informáticos","Redes"];
let alumnos=[24,17,28,26];
for (let indice in profesores)
document.write("<br/>"+profesores[indice]+" del módulo de "+asignaturas[indice]+", tiene
"+alumnos[indice]+" alumnos en clase.");
}
```

Éste será el resultado que obtendremos de la aplicación anterior:

Cristina del módulo de Seguridad, tiene 24 alumnos en clase.
Catalina del módulo de Bases de Datos, tiene 17 alumnos en clase.
Miguel del módulo de Sistemas Informáticos, tiene 28 alumnos en clase.
Benjamín del módulo de Redes, tiene 26 alumnos en clase.

Para que los arrays paralelos sean homogéneos, éstos deberán tener la misma longitud, ya que de esta forma se mantendrá la consistencia de la estructura lógica creada.

6. ARRAYS MULTIDIMENSIONALES.

Una alternativa a los arrays paralelos es la simulación de un array multidimensional. Si bien es cierto que en JavaScript los arrays son unidimensionales, podemos crear arrays que en sus posiciones contengan otros arrays u otros objetos. Podemos crear de esta forma *arrays bidimensionales,* tridimensionales, etc.

Por ejemplo, podemos realizar el ejemplo anterior creando un **array bidimensional** de la siguiente forma:

```
let datos = new Array();
datos[0] = new Array("Cristina","Seguridad",24);
datos[1] = new Array("Catalina", "Bases de Datos",17);
datos[2] = new Array("Miguel", "Sistemas Informáticos",28);
```

```
datos[3] = new Array("Benjamin","Redes",26);
```

También podemos hacerlo con la siguiente sintaxis

```
let datos = [];
datos[0] = ["Cristina","Seguridad",24];
datos[1] = ["Catalina", "Bases de Datos",17];
datos[2] = ["Miguel", "Sistemas Informáticos",28];
datos[3] = ["Benjamin","Redes",26];
```

o bien usando una definición más breve con los corchetes:

```
let parejas = [["amarillo", "Un color"],["atlántico", "Un océano"], ["ordenador", "Una gran herramienta "], ["laurel", "Un árbol"]];
```

Para acceder a un dato en particular, de un array de arrays, se requiere que hagamos una doble referencia. La primera referencia, será a una posición del array principal, y la segunda referencia, a una posición del array almacenado dentro de esa casilla del array principal. Esto se hará escribiendo el nombre del array, y entre corchetes cada una de las referencias:

```
nombrearray[indice1][indice2]
```

Por ejemplo:

```
document.write("<br/>Quien imparte Bases de Datos? "+datos[1][0]); // Catalina
document.write("<br/>Asignatura de Miguel: "+datos[2][1]); // Sistemas Informáticos
document.write("<br/>Alumnos de Benjamín: "+datos[3][2]); // 26
```

Para hacer un recorrido, podemos utilizar dos for anidados que nos permitan recorrer todas, las posiciones:

```
for (let i = 0; i < datos.length; i++) {
   for (let j = 0; j < datos[i].length; j++) {
      console.log(datos[i][j])
   }
}
```

Obtendríamos como resultado:

```
Cristina
Seguridad
24
Catalina
Bases de Datos
17
Miguel
Sistemas Informáticos
28
Benjamin
Redes
26
```

También podemos utilizar un solo for accediendo de forma individual a los elementos de esa posición, en este caso hay que tener cuidado con no desbordar.

```
for (let i=0; i<datos.length; i++){
```

```
console.log(datos[i][0]+" del módulo de "+datos[i][1]+", tiene "+datos[i][2]+" alumnos en clase.");
}
```

Obtendríamos como resultado:

Cristina del módulo de Seguridad, tiene 24 alumnos en clase.
Catalina del módulo de Bases de Datos, tiene 17 alumnos en clase.
Miguel del módulo de Sistemas Informáticos, tiene 28 alumnos en clase.
Benjamín del módulo de Redes, tiene 26 alumnos en clase.

Si queremos realizar la misma operación que en el caso anterior con un bucle **for..of**:

```
for (let dato of datos)
console.log(dato[0]+" del módulo de "+ dato[1]+", tiene "+dato[2]+" alumnos en clase.");
}
```

Obtendríamos el mismo resultado que en el caso anterior:

Cristina del módulo de Seguridad, tiene 24 alumnos en clase.
Catalina del módulo de Bases de Datos, tiene 17 alumnos en clase.
Miguel del módulo de Sistemas Informáticos, tiene 28 alumnos en clase.
Benjamín del módulo de Redes, tiene 26 alumnos en clase.