JavaScript 02 – Arrays (I)

Introducción

Los arrays, también llamados arreglos, vectores, matrices, listas…, son un tipo de objeto que no tiene tamaño fijo, podemos añadirle elementos en cualquier momento.

Para referenciar (hacer referencia a) los elementos se hace con un índice numérico. A diferencia de los objetos que se referencian con un nombre

Podemos crearlos como instancias del objeto Array:

```
let a = new Array()         // a = []
let b = new Array(2,4,6)    // b = [2, 4, 6]
```

Pero lo recomendado es crearlos usando notación JSON:

```
let a = []
let b = [2,4,6]
```

Sus elementos pueden ser de cualquier tipo, incluso podemos tener elementos de tipos distintos en un mismo array.

Si no está definido un elemento su valor será *undefined*. Ej.:

```
let a = ['Lunes', 'Martes', 2, 4, 6]
console.log(a[0])  // imprime 'Lunes'
console.log(a[4])  // imprime 6
a[7] = 'Juan'        // ahora a = ['Lunes', 'Martes', 2, 4, 6, , , 'Juan']
console.log(a[7])  // imprime 'Juan'
console.log(a[6])  // imprime undefined
```

Operaciones con Arrays

Vamos a ver los principales métodos y propiedades de los arrays.

length

Esta propiedad devuelve la longitud de un array:

```
let a = ['Lunes', 'Martes', 2, 4, 6]
console.log(a.length)  // imprime 5
```

Podemos reducir el tamaño de un array cambiando esta propiedad:

```
a.length = 3  // ahora a = ['Lunes', 'Martes', 2]
```

Añadir elementos

Podemos añadir elementos al final de un array con el método `push` o al principio con `unshift`:

```
let a = ['Lunes', 'Martes', 2, 4, 6]
a.push('Juan')   // ahora a = ['Lunes', 'Martes', 2, 4, 6, 'Juan']
a.unshift(7)     // ahora a = [7, 'Lunes', 'Martes', 2, 4, 6, 'Juan']
```

Eliminar elementos

Podemos borrar el elemento del final de un array con `pop` o el del principio con `shift`. Ambos métodos devuelven el elemento que hemos borrado:

```
let a = ['Lunes', 'Martes', 2, 4, 6]
let ultimo = a.pop()         // ahora a = ['Lunes', 'Martes', 2, 4] y ultimo = 6
let primero = a.shift()      // ahora a = ['Martes', 2, 4] y primero = 'Lunes'
```

splice

Permite eliminar elementos de cualquier posición del array y/o insertar otros en su lugar. Devuelve un array con los elementos eliminados. Sintaxis:

```
Array.splice(posicion, num. de elementos a eliminar, 1º elemento a insertar, 2º elemento a insertar, 3º...)
```

Ejemplo:

```
let a = ['Lunes', 'Martes', 2, 4, 6]
let borrado = a.splice(1, 3)       // ahora a = ['Lunes', 6] y borrado = ['Martes', 2, 4]
a = ['Lunes', 'Martes', 2, 4, 6]
borrado = a.splice(1, 0, 45, 56)   // ahora a = ['Lunes', 45, 56, 'Martes', 2, 4, 6] y borrado = []
a = ['Lunes', 'Martes', 2, 4, 6]
borrado = a.splice(1, 3, 45, 56)   // ahora a = ['Lunes', 45, 56, 6] y borrado = ['Martes', 2, 4]
```

EJERCICIO: Guarda en un array la lista de la compra con Peras, Manzanas, Kiwis, Plátanos y Mandarinas. Haz lo siguiente con splice:

- Elimina las manzanas (debe quedar Peras, Kiwis, Plátanos y Mandarinas)

- Añade detrás de los Plátanos, Naranjas y Sandía. (Debe quedar: Peras, Kiwis, Plátanos, Naranjas, Sandía y Mandarinas)

- Quita los Kiwis y pon en su lugar Cerezas y Nísperos. (Debe quedar: Peras, Cerezas, Nísperos, Plátanos, Naranjas, Sandía y Mandarinas)

slice

Devuelve un subarray con los elementos indicados, pero sin modificar el array original. Sería como hacer un `substr` pero de un array en vez de una cadena.

Sintaxis: `Array.slice(posicion, num. de elementos a devolver)`

Ejemplo:

```
let a = ['Lunes', 'Martes', 2, 4, 6]
let subArray = a.slice(1, 3) // ahora a=['Lunes', 'Martes', 2, 4, 6] y subArray=['Martes', 2, 4]
```

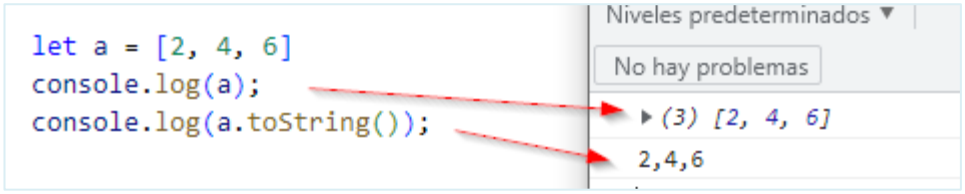
Es muy útil para hacer una copia de un array:

```
let a = [2, 4, 6]
let copiaDeA = a.slice() // ahora ambos arrays contienen lo mismo pero son diferentes arrays
```

Arrays y Strings

Cada objeto, y los arrays son un tipo de objeto, tienen definido el método **`.toString()`** que lo convierte en una cadena. Este método es llamado automáticamente cuando, por ejemplo, queremos mostrar un array por la consola.

En el caso de los arrays esta función devuelve una cadena con los elementos del array separados por coma.



Además, podemos convertir los elementos de un array a una cadena con **.join()** especificando el carácter separador de los elementos. Ej.:

```
let a = ['Lunes', 'Martes', 2, 4, 6]
let cadena = a.join('-')       // cadena = 'Lunes-Martes-2-4-6'
```

Este método es el contrario del **.split()** que convierte una cadena en un array. Ej.:

```
let notas = '5-3.9-6-9.75-7.5-3'
let arrayNotas = notas.split('-')        // arrayNotas = [5, 3.9, 6, 9.75, 7.5, 3]
let cadena = 'Que tal estás'
let arrayPalabras = cadena.split(' ')    // arrayPalabras = ['Que`, 'tal', 'estás']
let arrayLetras = cadena.split('')       // arrayLetras = ['Q','u','e`,' ','t',a',l','
','e',s',t',á',s']
```

sort

Ordena **alfabéticamente** los elementos del array

```
let a = ['hola','adios','Bien','Mal',2,5,13,45]
let b = a.sort()       // b = [13, 2, 45, 5, "Bien", "Mal", "adios", "hola"]
```

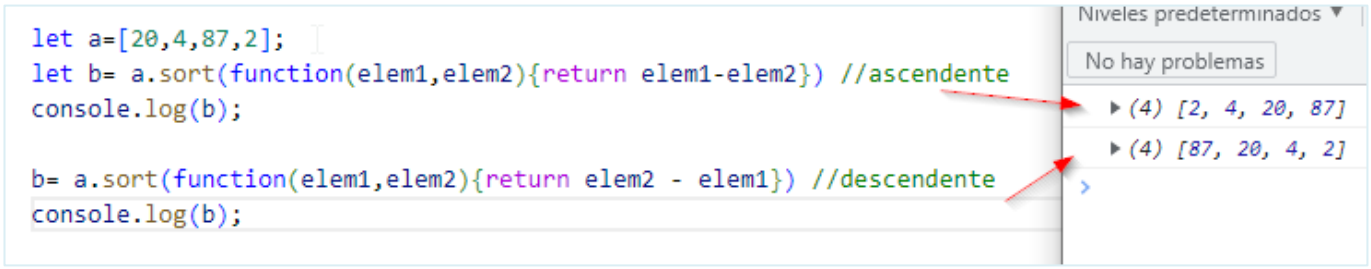
Si no se especifica otra cosa, el orden que se sigue es el de los códigos ascii, por lo que los dígitos numéricos van antes que las letras, y las mayúsculas antes que las minúsculas.

También podemos pasarle una función que le indique cómo ordenar. Esta función debe devolver un valor negativo si el primer elemento es menor, positivo si es mayor, o 0 si son iguales.

Ejemplo: ordenar un array de cadenas sin tener en cuenta si son mayúsculas o minúsculas:

```
let a = ['hola','adios','Bien','Mal']
let b = a.sort(function(elem1, elem2) {
  if (elem1.toLocaleLowerCase() < elem2.toLocaleLowerCase()) return -1;
  if (elem1.toLocaleLowerCase() > elem2.toLocaleLowerCase()) return 1;
  if (elem1.toLocaleLowerCase() = elem2.toLocaleLowerCase()) return 0;
});
         // b = ["adios", "Bien", "hola", "Mal"]
```

También se utiliza para ordenar números, tanto de forma ascendente como descendente:



Es frecuente utilizar esta función es para ordenar arrays de objetos. Por ejemplo, si tenemos un objeto *persona* con los campos *nombre* y *edad*, para ordenar un array de objetos persona por su edad haremos:

```
let personasOrdenado = personas.sort(function(persona1, persona2) {
    return persona1.edad-persona2.edad
})
```

Usando *arrow functions* quedaría más sencillo:

```
let personasOrdenado = personas.sort((persona1, persona2)  => persona1.edad-persona2.edad)
```

Si lo que queremos es ordenar por un campo de texto podemos usar la función *toLocaleCompare*:

```
let personasOrdenado = personas.sort((persona1, persona2)  => persona1.nombre.localeCompare(persona2.nombre))
```

EJERCICIO: Haz una función que ordene las notas de un array pasado como parámetro. Si le pasamos [4,8,3,10,5] debe devolver [3,4,5,8,10]. Pruébalo en la consola

Otros métodos comunes

Otros métodos que se usan a menudo con arrays son:

- `.concat()`: concatena arrays

```
let a = [2, 4, 6]
let b = ['a', 'b', 'c']
let c = a.concat(b)      // c = [2, 4, 6, 'a', 'b', 'c']
```

- `.reverse()`: invierte el orden de los elementos del array

```
let a = [2, 4, 6]
let b = a.reverse()      // b = [6, 4, 2]
```

- `.indexOf()`: devuelve la primera posición del elemento pasado como parámetro o -1 si no se encuentra en el array
- `.lastIndexOf()`: devuelve la última posición del elemento pasado como parámetro o -1 si no se encuentra en el array

Functional Programming

Se trata de un paradigma de programación (una forma de programar) donde se intenta que el código se centre más en qué debe hacer una función que en cómo debe hacerlo.

El ejemplo más claro es que intenta evitar los bucles *for* y *while* sobre arrays o listas de elementos.

Normalmente, cuando hacemos un bucle es para recorrer la lista y realizar alguna acción con cada uno de sus elementos. Lo que hace *functional programing* es que a la función que debe hacer eso, además de pasarle como parámetro la lista sobre la que debe actuar, se le pasa como segundo parámetro la función que debe aplicarse a cada elemento de la lista.

Desde la versión 5.1 javascript incorpora métodos de *functional programing* en el lenguaje, especialmente para trabajar con arrays:

filter

Devuelve un nuevo array con los elementos que cumplen determinada condición del array al que se aplica. Su parámetro es una función, habitualmente anónima, que va interactuando con los elementos del array. Esta función recibe como primer parámetro el elemento actual del array (sobre el que debe actuar). Opcionalmente puede tener como segundo parámetro su índice y como tercer parámetro el array completo. La función debe devolver **true** para los elementos que se incluirán en el array a devolver como resultado y **false** para el resto.

Ejemplo: dado un array con notas devolver un array con las notas de los aprobados. Esto usando programación *imperativa* (la que se centra en *cómo se deben hacer las cosas*) sería algo como:

```
let arrayNotas = [5.2, 3.9, 6, 9.75, 7.5, 3]
let aprobados = []
for (let i = 0; i < arrayNotas.length; i++) {
  let nota = arrayNotas[i]
  if (nota >=  5) {
    aprobados.push(nota)
  }
}       // aprobados = [5.2, 6, 9.75, 7.5]
```

Usando *functional programming* (la que se centra en *qué resultado queremos obtener*) sería:

```
let arrayNotas = [5.2, 3.9, 6, 9.75, 7.5, 3]
let aprobados = arrayNotas.filter(function(nota) {
  if (nota >=  5) {
    return true
  } else {
    return false
  }
})       // aprobados = [5.2, 6, 9.75, 7.5]
```

Podemos refactorizar esta función para que sea más compacta:

```
let arrayNotas = [5.2, 3.9, 6, 9.75, 7.5, 3]
let aprobados = arrayNotas.filter(function(nota) {
  return nota >= 5  // nota > =  5 se evalúa a 'true' si es cierto o 'false' si no lo es
})
// aprobados = [5.2, 6, 9.75, 7.5]
```

Y usando funciones tipo flecha la sintaxis queda mucho más simple:

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

Las 7 líneas del código usando programación *imperativa* quedan reducidas a sólo una.

EJERCICIO: Dado un array con los días de la semana obtén todos los días que empiezan por 'M'

find

Como *filter* pero NO devuelve un **array** sino el primer **elemento** que cumpla la condición (o *undefined* si no la cumple ninguno). Ejemplo:

```
let arrayNotas = [4, 5.2, 3.9, 6, 9.75, 7.5, 3]
let primerAprobado = arrayNotas.find(nota  => nota >=  5)    // primerAprobado = 5.2
```

Este método tiene más sentido con objetos. Por ejemplo, si queremos encontrar un coche de color rojo dentro de un array llamado coches cuyos elementos son objetos con un campo 'color' haremos:

```
let coches = [
    {
      "color": "morado",
      "tipo": "berlina",
      "capacidad": 7
    },
    {
      "color": "rojo",
      "tipo": "camioneta",
      "capacidad": 5
    },
    {
        "color": "rojo",
        "tipo": "furgón",
        "capacidad": 7
    }
  ]

let cocheBuscado = coches.find(coche  => coche.color === 'rojo')    // devolverá el
objeto completo del primer elemento que cumpla la condición.
```

EJERCICIO: Dado un array con los días de la semana obtén el primer día que empieza por 'M'

findIndex

Como *find* pero en vez de devolver el elemento devuelve su posición (o -1 si nadie cumple la condición).

```
let cocheBuscado = coches.findIndex(coche  => coche.color === 'rojo')    // devolverá 1
```

En el ejemplo de los coches el valor devuelto sería 1, ya que el segundo elemento cumple la condición.

Al igual que el anterior tiene más sentido con arrays de objetos.

EJERCICIO: Dado un array con los días de la semana, obtén la posición en el array del primer día que empieza por 'M'.

Solución:

every / some

- **every** devuelve **true** si **TODOS** los elementos del array cumplen la condición y **false** en caso contrario.
- **some** devuelve **true** si **ALGÚN** elemento del array cumple la condición. Ejemplo:

```
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
```

EJERCICIO: Dado un array con los días de la semana indica si algún día empieza por 'S'. Dado un array con los días de la semana indica si todos los días acaban por 's'

map

map permite modificar cada elemento de un array y devuelve un nuevo array con los elementos del original modificados.

Ejemplo: queremos subir un 10% cada nota:

```
let arrayNotas = [5.2, 3.9, 6, 9.75, 7.5, 3];
let arrayNotasSubidas = arrayNotas.map(nota  => nota + nota * 1.1);
```

EJERCICIO: Dado un array con los días de la semana devuelve otro array con los días en mayúsculas.

Solución:

```
let diasSemanaMayus= diasSemana.map(dia => dia.toUpperCase());
console.log(diasSemanaMayus);
```

reduce

reduce devuelve un valor calculado a partir de los elementos del array. En este caso la función recibe como primer parámetro el valor calculado hasta ahora y el método tiene como 1º parámetro la función y como 2º parámetro al valor calculado inicial (si no se indica será el primer elemento del array).

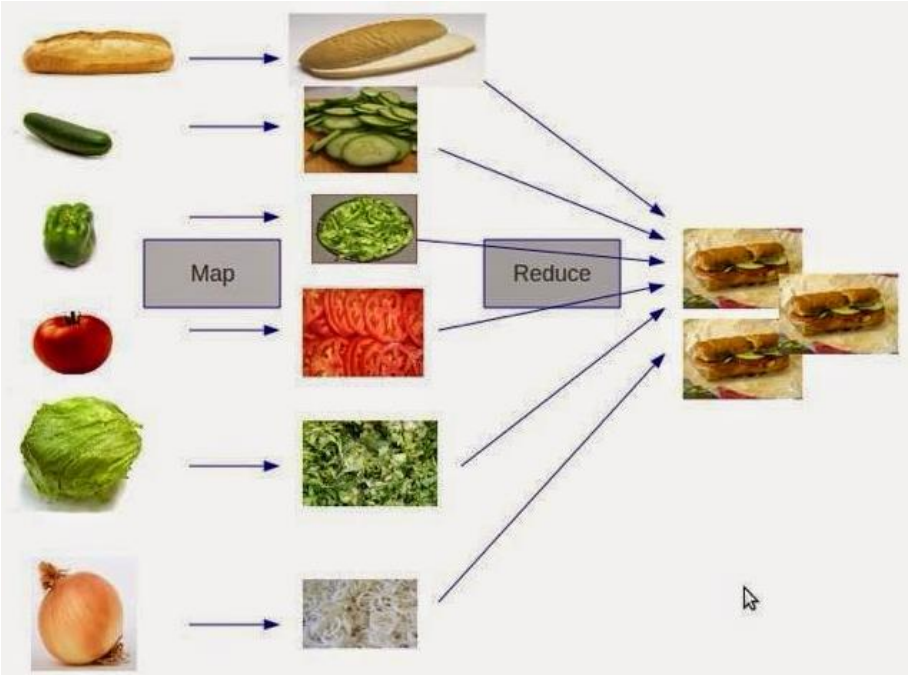
Ejemplo: queremos obtener la suma de las notas:

```
let arrayNotas = [5.2, 3.9, 6, 9.75, 7.5, 3];
let sumaNotas = arrayNotas.reduce((total,nota)  => total +=  nota, 30)  // total = 65.35
sumaNotas = arrayNotas.reduce((total,nota)  => total +=  nota)    // total = 35.35
```

Ejemplo: queremos obtener la nota más alta:

```
let arrayNotas = [5.2, 3.9, 6, 9.75, 7.5, 3]
let maxNota = arrayNotas.reduce((max,nota)  => nota > max ? nota : max)    // max = 9.75
```

En el siguiente ejemplo gráfico tenemos un "array" de verduras al que le aplicamos una función *map* para que las corte y al resultado le aplicamos un *reduce* para que obtenga un valor (el sandwich) con todas ellas:



EJERCICIO: Dado el array de notas anterior usar un método para que devuelva la nota media.

forEach

Es el método más general de los que hemos visto. No devuelve nada, sino que permite realizar algo con cada elemento del array.

```
let arrayNotas = [5.2, 3.9, 6, 9.75, 7.5, 3]
arrayNotas.forEach((nota, indice)  => {
  console.log('El elemento de la posición ' + indice + ' es: ' + nota)
})
```

includes

Devuelve **true** si el array incluye el elemento pasado como parámetro. Ejemplo:

```
let arrayNotas = [5.2, 3.9, 6, 9.75, 7.5, 3]
arrayNotas.includes(7.5)     // true
```

EJERCICIO: Dado un array con los días de la semana indica si algún día es el 'Martes'