Utilización de los objetos predefinidos del lenguaje

Arrays

- Son un tipo de objeto y no tienen tamaño fijo sino que podemos añadirle elementos en cualquier momento.
- 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 (recomendado)

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

Arrays

- 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*

```
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
```

Arrays

- Vamos a ver los principales métodos y propiedades de los arrays
- **lenght**
- 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]
```

Arrays

- Añadir elementos
- Podemos añadir elementos al final de un array con 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'
```

Arrays

- **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...)
```

```
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]
```

Arrays

- 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)
```

```
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]
```

- 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'
```

Arrays

Este método es el contrario del m .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"]
```

Arrays

- .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]
```

Arrays

- .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
- **Desestructuración de arrays**

permiten extraer los elementos del array directamente a variables y viceversa. Ejemplo:

```
let arrayNotas = [5.2, 3.9, 6, 9.75, 7.5, 3]
let [primera, segunda, tercera] = arrayNotas   // primera = 5.2, segunda = 3.9, tercera = 6
let [primera, , , cuarta] = arrayNotas         // primera = 5.2, cuarta = 9.75
let [primera, ...resto] = arrayNotas           // primera = 5.2, resto = [3.9, 6, 9.75, 3]
```

- Programación Funcional
- 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 aprobado.

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

```
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]
```

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

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

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

every / some

La primera devuelve **true** si **TODOS** los elementos del array cumplen la condición y **false** en caso contrario. La segunda devuelve **true** si **ALGÚN** elemento del array cumple la condición

```
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
```

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 * 10%)
```

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, 0)    // total = 35.35
// podríamos haber omitido el valor inicial 0 para total
let sumaNotas = arrayNotas.reduce((total,nota)  => total + =  nota)    // total = 35.35
```

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

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

Array.from
Devuelve un array a partir de otro al que se puede aplicar una función de transformación (es similar a *map*). Ejemplo: queremos subir un 10% cada nota:

```
let arrayNotas = [5.2, 3.9, 6, 9.75, 7.5, 3]
let arrayNotasSubidas = Array.from(arrayNotas, nota  => nota + nota * 10%)
```

- Sin embargo al copiar objetos (y los arrays son un tipo de objeto) la nueva variable apunta a la misma posición de memoria que la antigua por lo que los datos de ambas son los mismos:

```
let a = [54, 23, 12]
let b = a      // a = [54, 23, 12] b = [54, 23, 12]
b[0] = 3       // a = [3, 23, 12] b = [3, 23, 12]
let fecha1 = new Date('2018-09-23')
let fecha2 = fecha1          // fecha1 = '2018-09-23'   fecha2 = '2018-09-23'
fecha2.setFullYear(1999)   // fecha1 = '1999-09-23'   fecha2 = '1999-09-23'
```

- Si queremos obtener una copia de un array que sea independiente del original podemos obtenerla con slice o con Array.from

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

- **Map**
- **El método Array.map() crea una nueva matriz a partir de los resultados de llamar a una función para cada elemento (solo elementos)**

```
const numbers = [4, 9, 16, 25];
const newArr = numbers.map(Math.sqrt)// 2,3,4,5
```

- **Set**

  Es como un *Map* pero que no almacena los valores sino sólo la clave. Podemos verlo como una colección que no permite duplicados. Tiene la propiedad **size** que devuelve su tamaño y los métodos **.add** (añade un elemento), **.delete** (lo elimina) o **.has** (indica si el elemento pasado se encuentra o no en la colección)

  Una forma sencilla de eliminar los duplicados de un array es crear con él un *Set*

```
let ganadores = ['Márquez', 'Rossi', 'Márquez', 'Lorenzo', 'Rossi', 'Márquez', 'Márquez']
let ganadoresNoDuplicados = new Set(ganadores)    // {'Márquez, 'Rossi', 'Lorenzo'}
// o si lo queremos en un array:
let ganadoresNoDuplicados = Array.from(new Set(ganadores))    // ['Márquez, 'Rossi', 'Lorenzo']
```

```
const mySet1 = new Set()

mySet1.add(1)           // Set [ 1 ]
mySet1.add(5)           // Set [ 1, 5 ]
mySet1.add(5)           // Set [ 1, 5 ]
mySet1.add('some text') // Set [ 1, 5, 'some text' ]
```

```
mySet1.has(1)              // true
mySet1.has(3)              // false, since 3 has not been added to the set
mySet1.has(5)              // true
mySet1.has(Math.sqrt(25))  // true
mySet1.has('Some Text'.toLowerCase()) // true
```

```
mySet1.size         // 5

mySet1.delete(5)    // removes 5 from the set
```

```
console.log(mySet1)
```