Métodos de los Arrays

Repaso...

- En clases anteriores hemos trabajado con algunos métodos de la clase Array:

- of()
- concat()
- push()
- toString()
- join()
- forEach()
- map()

- Veamos algunos ejemplos en la consola...

pop() y shift()

- pop() : Este método elimina el último elemento de un array, y retorna dicho elemento eliminado.

```
> arr = ['X', 1, Math.PI, new Date().getFullYear()]
◂ ▶ (4) ["X", 1, 3.141592653589793, 2021]
> eliminado = arr.pop()
◂ 2021
> arr
◂ ▶ (3) ["X", 1, 3.141592653589793]
>
```

- shift() : Igual a pop(), pero este elimina el primer elemento en vez de el último.

reverse()

- Invierte el orden de los elementos de un array

```
> numeros = Array.of(1, 2, 3)
▶ (3) [1, 2, 3]
> numeros.reverse()
▶ (3) [3, 2, 1]
> numeros
▶ (3) [3, 2, 1]
>
```

splice()

- Cambia el contenido de un array, eliminando elementos existentes, o agregando nuevos elementos.
- unArray.splice(start, deleteCount, item1[, …, itemN]);

- *Start*: indica desde qué índice se realiza la acción
- *DeleteCount*: indica el número de elementos a eliminar del array antiguo. Si se omite, o si su valor es mayor que (arr.length - start), entonces todos los elementos desde start hasta el final del array serán eliminados.

  Si deleteCount es igual a 0 o negativo, no se eliminará ningún elemento. En este caso, se debe especificar al menos un nuevo elemento (ver más abajo).
- *Items*: elemento a insertar (pueden ser varios)
- Veamos ejemplos en la documentación:
- https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/spl

isArray()

- Recibe un parámetro, y retorna true si es un array, o false en caso contrario.

```
> Array.isArray(null)
<· false
> Array.isArray([])
<· true
> Array.isArray({ name: "Pepe" })
<· false
> Array.isArray([{ name: "Pepe" }])
<· true
> Array.isArray(new Date())
<· false
> |
```

at()

- Retorna el valor de un array, recibiendo el índice como parámetro. En caso que sea un numero negativo, comenzará el conteo desde el último índice

```
> arr = ['X', 1, Math.PI, new Date().getFullYear()]
◂ ▶ (4) ["X", 1, 3.141592653589793, 2021]
> arr.at(0)
◂ "X"
> arr.at(2)
◂ 3.141592653589793
> arr.at(-1)
◂ 2021
> arr.at(-2)
◂ 3.141592653589793
>
```

slice()

- Devuelve una copia de una parte de un array dentro de un nuevo array, en un rango de índices (de inicio a fin, y el final no incluido). El array original no se modificará

- UnArray.slice(inicio, final)
- Ambos parámetros son opcionales, y puedo indicar un solo parámetro (tomado como inicio)

Ejemplos de slice()

```
> a = [1, 2, 3, 4];
◂ ▶ (4) [1, 2, 3, 4]
> a.slice()
◂ ▶ (4) [1, 2, 3, 4]
> a.slice(2)
◂ ▶ (2) [3, 4]
> a.slice(0, 1)
◂ ▶ [1]
> a.slice(0, 2)
◂ ▶ (2) [1, 2]
>
```

filter()

- Este método crea un nuevo array con todos los elementos que cumplan la condición implementada por la función dada.

- UnArray.filter(function())

- Recordemos que a las funciones que se pasan como parámetro a una función, se les llama callback

Ejemplo de filter:

- Obtengamos todos los nombres de un array que empiezan con la letra A.

```
> nombres = ["Ana", "Pedro", "Juan", "axel", "Lorena", "Mateo"];
◂ ▶ (6) ["Ana", "Pedro", "Juan", "axel", "Lorena", "Mateo"]
>

> nombres.filter(nombre => nombre[0].toLowerCase() == "a")
◂ ▶ (2) ["Ana", "axel"]
>
```

some()

- Similar al método filter
- Verifica si al menos un elemento del array cumple con la condición implementada por la función callback

```
> edades = [8, 16, 10, 19, 17]
◂ ▶ (5) [8, 16, 10, 19, 17]
> edades.some(edad => edad > 17)
◂ true
> otrasEdades = [10, 13, 17]
◂ ▶ (3) [10, 13, 17]
> otrasEdades.some(edad => edad > 17)
◂ false
>
```