

Fundamentos de la Web

Bloque II: Tecnologías de cliente web

Tema 7.2 – Estructuras de Datos en JavaScript

Estructuras de datos

- **Arrays, conjuntos y mapas**
- Recorrer una estructura de datos
- Ordenación y Búsqueda

Arrays, conjuntos y mapas

- Las **estructuras de datos** permiten almacenar **colecciones** de elementos en memoria
- Existen **varios tipos** de estructuras en función de su comportamiento
- En **JavaScript** desde **ES6** existen las siguientes estructuras de datos en la API estándar:

Array

Set

Map

Arrays, conjuntos y mapas

- **Array**
 - Puede contener elementos duplicados y se puede acceder por posición
- **Set**
 - No puede tener dos o más objetos iguales
 - Se puede preguntar por la existencia de un elemento de forma rápida
- **Map**
 - Asocia valores a claves
 - El acceso del valor asociado a la clave es muy rápido

Arrays

- Colección que **mantiene el orden de inserción** y que puede contener elementos **duplicados**
- Se accede a los elementos indicando su **posición**
- **Crece de forma dinámica.** No es necesario especificar su tamaño.

Arrays

- Es la estructura de datos **más usada**
- Es la estructura de datos **más eficiente para la inserción** de elementos (al final)
- No obstante, **no** es muy **eficiente** para **búsquedas** (porque son secuenciales)

Arrays

- Sintaxis especial

```
let msgArray = [];
msgArray[0] = 'Hello';
msgArray[1] = 'Bye';

console.log(msgArray[0]); // 'Hello'
console.log(msgArray[2]); // undefined
console.log(msgArray.length); // 2
```

```
let msgArray = new Array();
msgArray[0] = 'Hello';
msgArray[99] = 'world';

console.log(msgArray[2]); // undefined
console.log(msgArray.length); // 100
```

Arrays

- **push()**
 - Añade uno o más elementos al final del array y devuelve la nueva longitud del array

```
let sports = ['soccer', 'baseball'];
let total = sports.push('football', 'swim');

console.log(sports); // ['soccer', 'baseball', 'football', 'swim']
console.log(total); // 4
```


Arrays

- **pop()**
 - Elimina un elemento del **final** del array y lo devuelve

```
let myFish = ['angel', 'clown', 'mandarin', 'sturgeon'];

let popped = myFish.pop();

console.log(myFish); // ['angel', 'clown', 'mandarin' ]
console.log(popped); // 'sturgeon'
```

Arrays

- **shift()**
 - Elimina un elemento del **inicio** del array y lo devuelve

```
let myFish = ['angel', 'clown', 'mandarin', 'sturgeon'];

let shifted = myFish.shift();

console.log(myFish); // ['clown', 'mandarin', 'sturgeon']
console.log(shifted); // 'angel'
```

Arrays

- **splice()**
 - Añade y/o elimina elementos del array

```
var deletedItems = array.splice(start[, deleteCount[, item1[, item2[, ...]]]])
```

Borrar elementos

```
let myFish = ['angel', 'clown', 'drum', 'mandarin', 'sturgeon'];
let removed = myFish.splice(3, 1);

// removed is ["mandarin"]
// myFish is ["angel", "clown", "drum", "sturgeon"]
```

Arrays

- **splice()**
 - Añade y/o elimina elementos del array

```
var deletedItems = array.splice(start[, deleteCount[, item1[, item2[, ...]]]])
```

Sustituir elementos

```
let myFish = ['angel', 'clown', 'drum', 'sturgeon'];
let removed = myFish.splice(2, 1, 'trumpet');

// myFish is ["angel", "clown", "trumpet", "sturgeon"]
// removed is ["drum"]
```

Arrays

- **splice()**
 - Añade y/o elimina elementos del array

```
var deletedItems = array.splice(start[, deleteCount[, item1[, item2[, ...]]]])
```

Añadir elementos

```
let myFish = ['angel', 'clown', 'mandarin', 'sturgeon'];
let removed = myFish.splice(2, 0, 'drum');

// myFish is ["angel", "clown", "drum", "mandarin", "sturgeon"]
// removed is [], no elements removed
```

Conjuntos (Set)

- No admite elementos **duplicados**
- Si se añade un elemento al conjunto y ya había otro **igual**, no se produce ningún cambio en el conjunto
- Es muy **eficiente** buscando entre sus elementos
- Pero eso hace que la **inserción** sea un poco más **costosa** que en los Arrays

Conjuntos (Set)

- **add()**
 - Añade un elemento al final de un Set

```
let mySet = new Set();
mySet.add(1);
mySet.add(5).add('some text'); // chainable

console.log(mySet); // Set [1, 5, "some text"]
```

Conjuntos (Set)

- Para saber si un elemento es igual a otro del conjunto se usa el operador ===
 - Dos strings son === si tienen los mismos caracteres
 - Dos objetos o arrays son === si son “el mismo” objeto
 - Dos objetos u arrays con los mismos valores, no son ===

Conjuntos (Set)

- **size**
 - Devuelve los elementos del Set

```
let mySet = new Set();
mySet.add(1);
mySet.add(5);
mySet.add(5); //duplicated
mySet.add('some text');
mySet.add('some text'); //duplicated

let o = {a: 1, b: 2};
mySet.add(o);
mySet.add(o); //duplicated
mySet.add({a: 1, b: 2}); // same values but new object is inserted

mySet.size; // 5
```

Conjuntos (Set)

- **delete()**
 - Borra el elemento (si existe)

```
let mySet = new Set();
mySet.add('foo');

mySet.delete('bar'); // Returns false. No "bar" element found.
mySet.delete('foo'); // Returns true. Successfully removed.
```

Conjuntos (Set)

- Creación de Set

- Desde un array

```
let myArray = ['value1', 'value1', 'value2'];
let mySet = new Set(myArray);
console.log(mySet); // Set ['value1', 'value2']
```

- Desde un string

```
let text = 'India';
let mySet = new Set(text); // Set ['I', 'n', 'd', 'i', 'a']
mySet.size; // 5
```

Conjuntos (Set)

- Creación de array desde Set
 - Con Array.from(set) o spread operator

```
let mySet = new Set();
mySet.add(1);
mySet.add(3);
mySet.add(5);
mySet.add(7);

let array1 = Array.from(mySet); // [1,3,5,7]

let array2 = [...mySet]; // [1,3,5,7]
```

Ejercicio 1

- Implementar una función **cuentaDistintas** que reciba un array con nombres de provincias (que podrían estar repetidas) e indique cuántas provincias sin repetir hay en ese array
- Ejecuta la función con un array de ejemplo para verificar que funciona como se espera

Mapas (Map)

- Define una estructura de datos que asocia (**mapea**) claves con valores
- **No** permite claves repetidas (===)
- Varias claves distintas pueden estar asociadas al mismo valor (**valores repetidos**)
- La búsqueda de un valor asociado a una clave es muy **eficiente**

Mapas (Map)

- **set() and get():**

- Permite asociar un valor a una clave y recuperar el valor posteriormente

```
let coches = new Map();

let toledo = new Coche('Seat', 'Toledo', 110);
let punto = new Coche('Fiat', 'Punto', 90);

coches.set('M-1233-YYY', toledo);
coches.set('M-1234-ZZZ', punto);

let c = coches.get('M-1234-ZZZ'); // Coche ['Fiat', 'Punto', 90]
```

Mapas (Map)

Ejemplo1

- **set() and get():**

```
let myMap = new Map();
myMap.set('a', { name: 'alpha' });
myMap.set('b', { name: 'beta' });
myMap.set('g', { name: 'gamma' });
```

```
console.log(myMap);
```

```
let myMap2 = new Map();
myMap2.set('a', new Date(1995,11,17));
myMap2.set('b', new Date(1920,11,17));
myMap2.set('g', new Date(2020,10,20));
```

```
console.log(myMap2);
```


Mapas (Map)

- **size:**

- Devuelve el número de pares clave / valor

```
let myMap = new Map();
myMap.set('a', 'alpha');
myMap.set('b', 'beta');
myMap.set('g', 'gamma');

myMap.size // 3
```

Mapas (Map)

- **delete():**
 - Borra una clave y su valor asociado

```
let myMap = new Map();

myMap.set('bar', 'foo');

myMap.delete('bar'); // Returns true. Successfully removed.

console.log(myMap); // Map []
```

Mapas (Map)

- Creación de Map
 - Desde un array

```
let kvArray = [['key1', 'value1'], ['key2', 'value2']];

//Create a map from 2D key-value Array
let myMap = new Map(kvArray);

myMap.get('key1'); // returns "value1"
```

Mapas (Map)

- Creación de array desde Map
 - Con Array.from(map) o spread operator

```
let myMap = new Map();
myApp.set('k1', 'val1');
myApp.set('k2', 'val2');

let array1 = Array.from(myMap); // [['k1','val1'], ['k2','val2']]

let array2 = [...myMap]; // [['k1','val1'], ['k2','val2']]
```

Comparativa: Array, Set y Map

	Array	Set	Map
Tamaño	length	size	size
Añadir	push(elem) splice(...)	add(elem)	set(key,value)
Eliminar	shift() pop() splice(...)	delete(elem)	delete(key)

Ejercicio 2

- Se pide crear una clase **GestorAeropuertos** que permita almacenar aeropuertos (nombre y ciudad)
- Se desea poder acceder a la información completa de un aeropuerto por su nombre de la forma más eficiente posible
- Se debe declarar como atributo de GestorAeropuertos la estructura de datos más eficiente
- Introducir varios aeropuertos y verificar el funcionamiento correcto cuando se accede a uno de ellos usando su nombre

Estructuras de datos

- Arrays, conjuntos y mapas
- **Recorrer una estructura de datos**
- Ordenación y Búsqueda

Recorrer una estructura de datos

- Acceder a cada elemento de una estructura de datos depende de su tipo:
 - Array
 - Acceso por posición con **bucle for**
 - Acceso **secuencial**
 - Conjunto (Set)
 - Acceso **secuencial**
 - Mapa (Map)
 - Acceso secuencial a la **colección de valores**
 - Acceso secuencial al **conjunto de claves**
 - Acceso secuencial al **conjunto de entradas**

Recorrer un Array

- Acceso por posición con **bucle for**

```
let ciudades = ['Ciudad Real', 'Madrid', 'Valencia'];  
  
for (let i=0; i < ciudades.length; i++) {  
  let ciudad = ciudades[i];  
  console.log(ciudad);  
}
```

Recorrer un Array

- Acceso secuencial con **for of**

```
let ciudades = ['Ciudad Real', 'Madrid', 'Valencia'];  
  
for (let ciudad of ciudades) {  
    console.log(ciudad);  
}
```

- En general es la forma preferida. Más conciso

Recorrer un Array

- Acceso secuencial con **forEach(...)**

```
let ciudades = ['Ciudad Real', 'Madrid', 'Valencia'];  
  
ciudades.forEach(ciudad => {  
    console.log(ciudad);  
});
```

- Ideal cuando se usa con más operaciones funcionales

Recorrer un Conjunto

- Acceso secuencia con **for of** o **forEach(...)**

```
let ciudades = new Set();
ciudades.add('Ciudad Real');
ciudades.add('Madrid');
ciudades.add('Valencia');

for (let ciudad of ciudades) {
  console.log(ciudad);
}

ciudades.forEach(ciudad => {
  console.log(ciudad);
});
```

Se recorren en el orden de inserción

Recorrer un Mapa

Ejemplo2

- Formas de recorrer un mapa
 - Acceso secuencial a la **colección de valores**

```
let myMap = new Map();
myMap.set('a', 'alpha');
myMap.set('b', 'beta');
myMap.set('g', 'gamma');

for(let value of myMap.values()){
    console.log(value);
}

//alpha
//beta
//gamma
```

Recorrer un Mapa

Ejemplo3

- Formas de recorrer un mapa
 - Acceso secuencial a la **colección de claves**

```
let myMap = new Map();
myMap.set('a', 'alpha');
myMap.set('b', 'beta');
myMap.set('g', 'gamma');

for(let key of myMap.keys()){
    console.log(key, myMap.get(key));
}

//a alpha
//b beta
//g gamma
```

Recorrer un Mapa

Ejemplo4

- Formas de recorrer un mapa
 - Acceso secuencial a la **colección de entradas**
 - Con *destructuring* queda más conciso

```
let myMap = new Map();  
myMap.set('a', 'alpha');  
myMap.set('b', 'beta');  
myMap.set('g', 'gamma');  
  
for(let [key, value] of myMap){  
    console.log(key,value);  
}  
  
//a alpha  
//b beta  
//g gamma
```

Recorrer un Mapa

- Formas de recorrer un mapa
 - `forEach(...)` por cada entrada

```
let myMap = new Map();
myMap.set('a', 'alpha');
myMap.set('b', 'beta');
myMap.set('g', 'gamma');

myMap.forEach((key, value) => {
  console.log(key,value);
});

//a alpha
//b beta
//g gamma
```


Estructuras de datos

- Arrays, conjuntos y mapas
- Recorrer una estructura de datos
- **Ordenación y Búsqueda**

Ordenación y Búsqueda

Ejemplo5

- Sólo se pueden ordenar los **Arrays**
- Método **sort(...)** de Array

```
let nombres = ['Pepe', 'Juanin', 'Antonio'];  
  
nombres.sort();  
  
console.log(nombres); // ['Antonio', 'Juanin', 'Pepe']
```

Ordenación y Búsqueda

- Se puede especificar el **orden de comparación**
 - Se usa una función que devuelve un valor positivo si o_1 es mayor que o_2 . Negativo en caso contrario

Ordenar por longitud de los nombres

Ejemplo6

```
let nombres = ['Pepe', 'Juanin', 'Antonio'];
nombres.sort((s1,s2) => s1.length - s2.length);
console.log(nombres); // ['Pepe', 'Juanin', 'Antonio']
```

Ordenación y Búsqueda

- ¿Qué es una búsqueda?
 - **Array:** Saber la posición de un elemento (si está en el array)
 - **Conjunto:** Saber si está el elemento
 - **Map:** Saber el valor asociado a la clave (si está)

Ordenación y Búsqueda

- Búsqueda en conjuntos (has(...))

```
let mySet = new Set();
mySet.add('foo');

mySet.has('foo'); // returns true
mySet.has('bar'); // returns false

let set1 = new Set();
let obj1 = {'key1': 1};
set1.add(obj1);

set1.has(obj1); // returns true
set1.has({'key1': 1}); // returns false because they are
                        // different object references
```

Ordenación y Búsqueda

- Búsqueda en mapas (get(...) y has(...))

```
let myMap = new Map();
myMap.set('bar', 'foo');

myMap.has('bar'); // returns true
myMap.has('baz'); // returns false
```

```
let myMap = new Map();
myMap.set('bar', 'foo');

myMap.get('bar'); // Returns "foo".
myMap.get('baz'); // Returns undefined
```

Ordenación y Búsqueda

- **Búsquedas en Arrays**

- `array.indexOf(searchElement[, fromIndex])`
- Se busca la posición del primer elemento === desde la posición indicada

```
let array = [2, 9, 9];
array.indexOf(2); // 0
array.indexOf(7); // -1
array.indexOf(9, 2); // 2
array.indexOf(2, -1); // -1
array.indexOf(2, -3); // 0
```

Ordenación y Búsqueda

- Hay que **elegir muy bien** la estructura de datos que se utiliza en un programa
 - Arrays
 - Eficiente la inserción al final $O(1)$
 - Eficiente el acceso por posición $O(1)$
 - Ineficiente la búsqueda $O(n)$
 - Conjuntos
 - Eficiente la inserción $O(1)$ (aunque menos que la lista)
 - No se puede hacer acceso por posición
 - Eficiente la búsqueda $O(1)$
 - Mapas
 - Igual que los conjuntos

Ordenación y Búsqueda

	Array	Set	Map
Acceso por posición	Eficiente $O(1)$	No se puede	No se puede
Modificación	<ul style="list-style-type: none"> - Eficiente inserción/borrado al final $O(1)$ - Ineficiente cualquier otra modificación $O(n)$ 	<ul style="list-style-type: none"> - Eficiente inserción/borrado. Pero más costosa que el Array $O(1)$ 	<ul style="list-style-type: none"> - Eficiente inserción/borrado. Pero más costosa que el Array $O(1)$
Búsqueda	Ineficiente $O(n)$	Eficiente $O(1)$	Eficiente $O(1)$

Ejercicio 3

- Implementar una aplicación que permita gestionar en memoria un conjunto de viajes de una aerolínea
- Cada viaje se representa con la ciudad origen, destino y la duración del viaje
- Se dan de alta los viajes en un gestor (clase **GestorViajes**)
- Al gestor de viajes se le pueden pedir:
 - Devolver un array con los viajes que tienen una determinada ciudad origen
 - Devolver un array con los viajes que tienen una determinada ciudad destino
 - Devolver un array con los viajes
 - Devolver un array con las ciudades en las que hay viajes
- Hay que conseguir el **menor tiempo de ejecución** de las consultas, aunque sean necesarias varias estructuras de datos