code
URJC

Tema 3
Estructuras de datos

Micael Gallego
micael.gallego@gmail.com
@micael_gallego

Estructuras de datos

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

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.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

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

code URJC

- **Sintaxis especial**

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

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

```
var 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

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

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

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/push

Arrays

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

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

var popped = myFish.pop();

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

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/pop

Arrays

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

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

var shifted = myFish.shift();

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

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/shift

Arrays

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

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

Borrar elementos

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

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

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/splice

Arrays

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

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

Sustituir elementos

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

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

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/splice

Arrays

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

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

Añadir elementos

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

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

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/splice

Ejercicio 1

- Crear un ejemplo básico para probar el funcionamiento de los Arrays
  - Declarar una lista de **String**.
  - Añadir y eliminar elementos de la lista
  - Definir una función **addElemToArray(…)** que reciba un array de **String** y un **String** como parámetro y añada el **String** al array

Conjuntos (Set)

code URJC

- 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

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set

Conjuntos (Set)

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

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

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

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set/add

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 === sin son "el mismo" objeto
  - Dos objetos u arrays con los mismos valores, no son ===

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set

Conjuntos (Set)

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

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

var 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
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set/size

Conjuntos (Set)

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

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

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

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set/delete

Conjuntos (Set)

- **Creación de Set**
  - Desde un array

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

  - Desde un string

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

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set

Conjuntos (Set)

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

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

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

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

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set

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**

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map

Mapas (Map)

- **set() and get():**
  - Permite asociar un valor a una clave y recuperar el valor posteriormente

```
var coches = new Map();

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

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

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

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map/get
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map/set

Mapas (Map)

- **size:**
  - Devuelve el número de pares clave / valor

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

myMap.size // 3
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map/size

Mapas (Map)

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

```
var myMap = new Map();

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

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

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

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map/delete

Mapas (Map)

- **Creación de Map**
  - Desde un array

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

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

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

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map

Mapas (Map)

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

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

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

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

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map

Mapas (Map)

- **Un Map es diferente a un objeto porque**
  - Además de string y symbol, las claves pueden ser objetos o funciones
  - Las claves se pueden recorrer en el orden de inserción
  - Es fácil determinar el número de claves (size)
  - Se puede recorrer más fácilmente
  - Puede tener mejor rendimiento cuando se añaden y borrar claves constantemente

Mapas (Map)

```
var configuracion = new Map();

configuracion.set('lenguaje', 'ingles');
configuracion.set('servidor', 'http://...');
Configuracion.set('correo', 'a.b@xyz');
...
var lenguaje = configuracion.get('lenguaje');
var servidor = configuracion.get('servidor');
```

```
var configuracion = {
  lenguaje: 'ingles',
  servidor: 'http://...',
  correo: 'a.b@xyz'
}
...
var { lenguaje, servidor } = configuracion;
```

Ejercicio 2

- Se tiene una colección de aeropuertos (objetos con información de un aeropuerto), y se desea poder obtener un aeropuerto dado su nombre
- Declarar la estructura de datos adecuada para asociar el nombre de cada aeropuerto con el objeto aeropuerto correspondiente
- Introducir varios aeropuertos asociados a sus nombres: “El Prat”, “Barajas”, “Castellón”
- Obtener el objeto aeropuerto dado su nombre: “Barajas”

Comparativa: Array, Set y Map

code URJC

	Array	Set	Map
Tamaño	`length`	`size`	`size`
Añadir	`push(elem)` `splice(...)`	`add(elem)`	`set(key,value)`

Eliminar	`shift()`
	`pop()`
	`splice(...)`
		`delete(elem)`	`delete(key)`

Estructuras de datos

code URJC

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

Recorrer una estructura de datos

code URJC

- 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**

```
var 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**

```
var 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(...)**

```
var ciudades = ['Ciudad Real', 'Madrid', 'Valencia'];

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

- Ideal cuando se usa con más operaciones funcionales

Recorrer un Array

code URJC

- Borrar elementos mientras se recorre el Array
  - Recorrer for en orden inverso

Ejemplo: Borrar números impares

```
var numbers = [17, 2, 14, 15, 20, 8, 7, 1, 9, 19, 3, 18, 5];

for (let i = numbers.length - 1; i >= 0; --i) {
  if (numbers[i] % 2 === 0) {
    numbers.splice(i, 1); // Remove value in position i
  }
}
```

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

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

```
var 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

code URJC

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

```
var 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

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

```
var 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

code URJC

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

```
var 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

code URJC

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

Ordenación y Búsqueda

- 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 o1 es mayor que o2. Negativo en caso contrario

Ordenar por longitud de los nombres

```
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

code URJC

- **Búsqueda en conjuntos (has(...))**

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

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

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

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

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set/has

Ordenación y Búsqueda

- **Búsqueda en mapas (get(…) y has(…))**

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

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

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

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

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set/has

Ordenación y Búsqueda

code URJC

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

```
var 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

code URJC

- 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	- Eficiente inserción/borrado al final **O(1)** - Ineficiente cualquier otra modificación **O(n)**	- Eficiente inserción/borrado. Pero más costosa que el Array **O(1)**	- 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 o módulo GestorViajes)
- Al gestor de viajes se le pueden pedir:
  - Devolver todos los viajes que tienen una determinada ciudad origen
  - Devolver todos los viajes que tienen una determinada ciudad destino
  - Devolver todos los viajes
  - Devolver todas 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

Estructuras de datos

code URJC

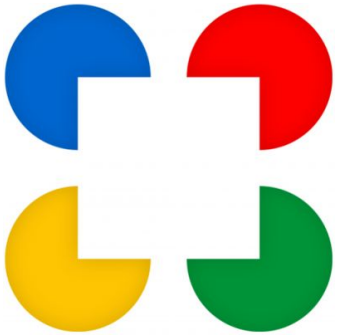
- Arrays, conjuntos y mapas
- Recorrer una estructura de datos
- Ordenación y Búsqueda
- **Librerías de estructuras de datos**

Librerías de estructuras de datos

code URJC

- Las estructuras de datos de JavaScript ES6+ son las **esenciales**
- En ciertas ocasiones son necesarias otras estructuras **más avanzadas** con comportamiento **específico**
- Existen diversas **librerías en NPM** que implementan estas estructuras de datos y utilidades

Librerías de estructuras de datos



Closure Library

- Librería de utilidades de Google
- Uno de sus módulos proporciona estructuras de datos avanzadas

https://developers.google.com/closure/library

Librerías de estructuras de datos

code URJC

- **Google Closure Library**

```
$ npm install google-closure-library
```

- AvlTree
- CircularBuffer
- Heap
- InversionMap
- LinkedMap
- Map
- Node
- Pool
- PriorityPool
- PriorityQueue
- QuadTree
- Queue
- Set
- SimplePool
- StringSet
- TreeNode
- Trie

https://google.github.io/closure-library/api/goog.structs.html

Librerías de estructuras de datos

code URJC

- **Google Closure Library**

```
require("google-closure-library");

goog.require("goog.structs.PriorityQueue");

var queue = new goog.structs.PriorityQueue();

queue.enqueue(1,'value1_p1');
queue.enqueue(3,'value2_p3');
queue.enqueue(2,'value3_p2');
queue.enqueue(1,'value4_p1');

while(queue.getCount() > 0){
   console.log(queue.dequeue());
}

// value1_p1 value4_p1 value3_p2 value2_p3
```

Librerías de estructuras de datos

code URJC

- **Buckets**
  - A JavaScript Data Structure Library

- Linked List
- Dictionary
- Multi Dictionary
- Binary Search Tree
- Stack
- Queue
- Set
- Bag
- Binary Heap
- Priority Queue

https://github.com/mauriciosantos/Buckets-JS

Ejercicio 4

- Reimplementar el Ejercicio 3 con un **MultiDictionary** de la librería Buckets