JESUÏTES educació

DAWM06UF2

ESTRUCTURES DEFINIDES PEL PROGRAMADOR

OBJECTES

UF2.4.2. ESTRUCTURES DEFINIDES PEL PROGRAMADOR. ARRAY I COL·LECCIONS

CFGS Desenvolupament d'Aplicacions Web

M06. Desenvolupament web en entorn client

Fundació Jesuïtes Educació - Escola del Clot

Sergi Grau sergi.grau@fje.edu

OBJECTIUS

- Aprendre el funcionament de l'objecte Array de JS
- Treballar amb estructures dinàmiques de dades
- Aplicar les millores que aporten les darreres versions d'ES en el treball de matrius en JS

INTRODUCCIÓ

- Un array és una col·lecció ordenada de valors, cada valor es denomina element, i te una posició coneguda com a índex. estan basat en zero índex i permeten indexar 32 bits.

- En JS un array pot contenir qualsevol tipus de dada.

- En JS son estructures de dades dinàmiques, i no necessàriament han de tenir continuïtat.

- La propietat length ens retorna el nombre d'elements, en el cas de ser contigua.

- Array es una especialització d'object, però es mes complex que simplement una propietat amb un enter.

- el seu accés es mes ràpid que a les propietats d'un objecte.

CREACIÓ D'ARRAYS

JESUÏTES educació
DAWM06UF2 sergi.grau@fje.edu

JS

```
//creació d'un array com a literal
let empty = [];
let primes = [2, 3, 5, 7, 11];
let misc = [ 1.1, true, "a", ];
//els elements poden ser expressions
let table = [base, base+1, base+2, base+3];
//poden contenir objectes
let b = [[1,{x:1, y:2}], [2, {x:3, y:4}]];
//si ometem un element, l'array contindrà a la seva posició un undefined
let count = [1,,3];
let undefs = [,,]; //només es permet ometre 1 element, per tant aquí tenim només 2 elements


//utilitzant el constructor Array()
let a = new Array();
let a = new Array(10); //nombre d'elements
let a = new Array(5, 4, 3, 2, 1, "testing, testing");
```

ACCEDINT A LES POSICIONS D'UN ARRAY

JS

```
//els arrays són un tipus particular d'objecte i JavaScript converteix l'enter de l'índex a
string
let a = ["world"];
let value = a[0];
a[1] = 3.14;
i = 2;
a[i] = 3;
a[i + 1] = "hello";
a[a[i]] = a[0];

//tots els índex són propietats d'objectes, però no al reves, cal que estiguin entre 0 i
2^32-1
// si utilitzem nombres fora del rang o cadenes, s'afegeixen propietats i no índex a
l'objecte array

a[-1.23] = true; //crea prop "-1.23"
A["1000"] = 0; // el 1001 element d'un array
A[1.000] // igual que a[1]

//si consultem una posició incorrecta d'un array s'obté undefined, i no fóra de límits
perque són propietats
```

ARRAYS NO CONTIGUES

JESUÏTES educació
DAWM06UF2 sergi.grau@fje.edu

JS

```
//els arrays en JavaScript poden ser no contigues
//normalment length retorna el nombre d'elements, però si no és una matriu contigua
retorna un nombre major al nombre d'elements
a = new Array(5); //length=5
A = []; //length=0
A[1000] = 0; //length=1001

//aquestes matrius són poc eficients


let a1 = [,,,];[undefined, undefined, undefined]
let a2 = new Array(3);
0 in a1 => true: a1 té cap element amb index 0
0 in a2 => false: a2 no té cap element amb index 0

Si volem fer que length sigui només de lectura, fem
a = [1,2,3];
Object.defineProperty(a, "length", {writable: false});
a.length = 0; //a no es canvia
```

PROPIETAT LENGHT

JS

```
//podem modificar la propietat length per ajustar la mida d'un array
a = [1,2,3,4,5];
a.length = 3; //[1,2,3].
a.length = 0;// []
a.length = 5; //sense elements com Array(5)
```

AFEGINT I ESBORRANT ELEMENTS EN UN ARRAY

JS

```
//la manera més simple és assignar valors a nous índex
a = []
a[0] = "zero";
a[1] = "un";

//podem utilitzar el mètode push() per afegir elements al final del array
A = []; //matriu buida
a.push("zero")
a.push("un", "dos")

//per afegir elements al principi d'un array utilitzem el mètode unshift()
//desplaçant els elements a índexs més elevats

//per esborrar elements utilitzem l'operador delete
a = [1,2,3];
delete a[1];
1 in a => false
a.length => 3, delete no afecta a la propietat length, per a fer-ho utilitzem shift()

//podem eliminar del final amb el mètode pop(), i retorna el valor de l'últim element,
funcionant la matriu com una estructura de dades pila
```

ITERANT ARRAYS

JESUÏTES educació
DAWM06UF2 sergi.grau@fje.edu

JS

```
//amb el bucle for
let claus = Object.keys(o); // array de les propietats
let valors = []
// desem els valors de les propietats en un array
for(let i = 0; i < claus.length; i++) {
   let clau= claus[i];
   valors[i] = o[clau];
}

//bucle optimitzat, només obté un cop la longitud
for(let i = 0, len = claus.length; i < len; i++) {
...
}
// si l'array conté valors no vàlids els hem d'excloure
for(let i = 0; i < a.length; i++) {
 if (!a[i]) continue; // salta null, undefined, i elements no existents
}
```

ITERANT ARRAYS

JESUÏTES educació
DAWM06UF2 sergi.grau@fje.edu

JS

```
//per saltar només undefined i elements no existents
for(let i = 0; i < a.length; i++) {
        if (a[i] === undefined) continue;
    }
    //saltar només elements no existents
    for(let i = 0; i < a.length; i++) {
     if (!(i in a)) continue ;
    }
    //el bucle for/in és molt útil amb matrius no contingues
    for(let index in matriu) {
    let valor = matriu[index];
    ...
    }
```

MATRIUS MULTIDIMENSIONALS

JESUÏTES educació
DAWM06UF2 sergi.grau@fje.edu

JS

```
//JavaScript no suporta matrius multidimensionals, però podem fer matrius de matrius


let taula = new Array(10); // 10 files a la taula
for(let i = 0; i < taula.length; i++)
 taula[i] = new Array(10); //cada fila 10 columnes
//inicialitza la matriu
for(let fila = 0; fila < taula.length; fila++) {
   for(col = 0; col < taula[fila].length; col++) {
     taula[fila][col] = fila*col; }
}

let producte = taula[5][7];
```

MÈTODES D'ARRAY

JS

```
//El mètode join() converteix tots els elements d'un array en una
concatenació de cadenes, podem especificar la cadena separadors

let a = [1, 2, 3];
a.join(); // => "1,2,3"
a.join(" "); // => "1 2 3"
a.join(""); // => "123"
let b = new Array(10);
b.join('-'); => '---------'

//El mètode reverse() inverteix l'ordre dels elements d'una matriu sense
crear una nova matriu

let a = [1,2,3];
a.reverse().join() // => "3,2,1"
```

MÈTODES D'ARRAY

JS

```
//El mètode sort() ordena els elements d'un array. Quan es crida sense
arguments realitza l'ordenador lexicogràfica. Si l'array conté elements
indefinits els ordena al final. Si colem utilitzar un altre criteri
d'ordenació hem d'especificar-ho com arguments de sort(). Aquesta funció
decideix quan ha de apareixer: si el primer argument va abans que el segon,
la comparació ha de retornar <0, si són iguals 0 i si va després >0

let a = new Array("banana", "poma", "taronja");
a.sort();
let s = a.join(", "); // s == "banana, poma, taronja"

let a = [33, 4, 1111, 222];
a.sort(); // ordre lexicogràfic: 1111, 222, 33, 4 a.sort(function(a,b) { //
ordre numèric: 4, 33, 222, 1111
  return a-b; // retorna <0, 0, o >0, depenent de l'ordre});
a.sort(function(a,b) {return b-a}); // ordre numèric invers
```

JS

```
//El mètode concat() crea i retorna una nova matriu amb la concatenació de
dues matrius

let a = [1,2,3];
a.concat(4, 5); //retorna [1,2,3,4,5]
a.concat([4,5]); //retorna [1,2,3,4,5]
a.concat([4,5],[6,7]); //retorna [1,2,3,4,5,6,7]
a.concat(4, [5,[6,7]]); //retorna [1,2,3,4,5,5,[6,7]]

//El mètode slice() retorna un subarray. Es poden especificar valors negatius

let a = [1,2,3,4,5];
a.slice(0,3); // Retorna [1,2,3]
a.slice(3); // Retorna [4,5]
a.slice(1,-1); // Retorna [2,3,4]
a.slice(-3,-2); // Retorna [3]
```

MÈTODES D'ARRAY

JS

```
//El mètode splice() permet inserir i esborrar elements d'una matriu, a
diferència de slice, modifica directament la matriu. Per tant els índex de la
matriu es veuran modificats. El primer argument especifica la posició on comença
la inserció o l'esborrament., si s'omet el segon argument, s'esborren tots els
elements, els següents arguments indiquen els elements a afegir

let a = [1,2,3,4,5,6,7,8];
a.splice(4); // Retorna [5,6,7,8];a és [1,2,3,4]
a.splice(1,2); // Retorna [2,3]; a és [1,4]
a.splice(1,1); // Retorna [4]; a és [1]

//El mètode slice() retorna un subarray. Es poden especificar valors negatius

let a = [1,2,3,4,5];
a.splice(2,0,'a','b'); // Retorna []; a és [1,2,'a','b',3,4,5] a.splice(2,2,
[1,2],3); // Retorna ['a','b']; a és [1,2,[1,2],3,3,4,5]
```

JS

```
//Els mètodes push() i pop() permeten treballar amb arrays com si es tractés d'una estructura de pila. Push() per afegir (apilar) i retorna la nova longitud de la pila i pop() per treue (desapilar)

let pila = []; // pila: []
pila.push(1,2); // pila: [1,2] retorna 2
pila.pop();// pila: [1] retorna 2
pila.push(3); // pila: [1,3] retorna 2
pila.pop(); // pila: [1] retorna 3
pila.push([4,5]); // pila: [1,[4,5]] retorna 2
pila.pop();// pila: [1]  retorna [4,5]
pila.pop();// pila: [] retorna 1
```

JS

```
//Els mètodes unshift() i shift() funcionen com push() i pop(), excepte que
inserixen i esborren elements al principi de l'array. Unshift() afegeix al
principi, shift() treu del principi

let a = []; // a:[]
a.unshift(1);  // a:[1] retorna: 1
a.unshift(22);// a:[22,1] retorna: 2
a.shift();// a:[1] retorna: 22
a.unshift(3,[4,5]);// a:[3,[4,5],1] retorna: 3
a.shift(); // a:[[4,5],1] retorna: 3
a.shift(); // a:[1]  retorna: [4,5]
a.shift();// a:[] retorna: 1
```

MÈTODES ES5, FOREACH

JESUÏTES educació
DAWM06UF2 sergi.grau@fje.edu

JS

```
//El bucle forEach() permet iterar una matriu invocant una funció per cada element. No permet el break i si volem finalitzar abans de temps hem de generar una excepció

let data = [1,2,3,4,5];
let suma = 0;
data.forEach(function(valor) { suma += valor; });
suma; => 15

data.forEach(function(v, i, a) { a[i] = v + 1; });
data // => [2,3,4,5,6]
```

MÈTODES ES5, MAP I FILTER

JESUÏTES educació
DAWM06UF2 sergi.grau@fje.edu

JS

```
//El mètode map() passa cada element d'una matriu a una funció, i retorna una matriu amb la transformació, a diferència de forEach, retorna una nova matriu

a = [1, 2, 3];
b = a.map(function(x) { return x*x; }); // b és [1, 4, 9]

//El mètode filter() retorna una matriu amb un subconjunt d'elements d'una matriu

a = [5, 4, 3, 2, 1];
petits = a.filter(function(x) { return x < 3 }); // [2, 1]
contingutsIndexparell = a.filter(function(x,i) { return i%2==0 }); // [5, 3, 1]
```

MÈTODES ES5, EVERY

JESUÏTES educació
DAWM06UF2 sergi.grau@fje.edu

JS

```
//El mètode every() és un predicat que retorna cert si i només si es cert per
a tots els elements. Some(), retorna cert si algun compleix la condició

a = [1,2,3,4,5];
a.every(function(x) { return x < 10; }) // => true:  a.every(function(x)
{ return x % 2 === 0; }) // => false:
a.some(function(x) { return x%2===0; }) // => true  a.some(isNaN) // => false
```

JS

```
//El mètode indexOf() i lastIndexOf() cerca en una matriu elements amb un
determinat i valor i retorna la seva posició o -1.

a = [0,1,2,1,0];
a.indexOf(1) // => 1: a[1] és 1
a.lastIndexOf(1) // => 3: a[3] és 1
a.indexOf(3) // => -1


//El mètode isArray() ens permet saber si un objecte és un array

Array.isArray([]) // => true
Array.isArray({}) // => false

//A EC5 les cadenes són objectes arrays

let s = test;
s.charAt(0) // => "t"
s[1] // => "e"
```

JS

```
//from, crea un nou array a partir d'un objecte iterable
console.log(Array.from('foo'));
// Array ["f", "o", "o"]

console.log(Array.from([1, 2, 3], x => x + x));
//  Array [2, 4, 6]
```

JS

```
const array1 = ['a', 'b', 'c', 'd', 'e'];

// copia del index 0 al index 3
console.log(array1.copyWithin(0, 3, 4));
// Array ["d", "b", "c", "d", "e"]

// copia del index 1 all elements des del index 3 fins els final
console.log(array1.copyWithin(1, 3));
//  Array ["d", "d", "e", "d", "e"]
```

MÈTODES ES5, FILL

JS

```
//fill omple els elements d'un array amb element estàtics

const array1 = [1, 2, 3, 4];

// omple amb 0 des de 2  fins 4
console.log(array1.fill(0, 2, 4));
//  [1, 2, 0, 0]

//omple amb 5 des de posició 1
console.log(array1.fill(5, 1));
//  [1, 5, 5, 5]

console.log(array1.fill(6));
// [6, 6, 6, 6]
```

MÈTODES ES5, FILL

JS

```
//fill omple els elements d'un array amb element estàtics

const array1 = [1, 2, 3, 4];

// omple amb 0 des de 2  fins 4
console.log(array1.fill(0, 2, 4));
//  [1, 2, 0, 0]

//omple amb 5 des de posició 1
console.log(array1.fill(5, 1));
//  [1, 5, 5, 5]

console.log(array1.fill(6));
// [6, 6, 6, 6]
```

MÈTODES ES5, FIND

JESUÏTES educació
DAWM06UF2 sergi.grau@fje.edu

JS

```
//troba un element en una matriu
const array1 = [5, 12, 8, 130, 44];

const found = array1.find(element => element > 10);

console.log(found);
// 12


//idem però retorna el index
const array1 = [5, 12, 8, 130, 44];

const isLargeNumber = (element) => element > 13;

console.log(array1.findIndex(isLargeNumber));
// 3
```

MÈTODES ES5, REDUCE

JESUÏTES educació
DAWM06UF2 sergi.grau@fje.edu

JS

```
//retorna un sol valor aplicant la funció d'acumulació
const array1 = [1, 2, 3, 4];
const reducer = (accumulator, currentValue) => accumulator + currentValue;

// 1 + 2 + 3 + 4
console.log(array1.reduce(reducer));
//10

// 5 + 1 + 2 + 3 + 4
console.log(array1.reduce(reducer, 5));
//15
```

COL·LECCIONS (MAP), ES6/ES7

- Els arrays són estructures de seqüencies, és a dir les dades tenen un determinat ordre a l'estructura.

- L'objecte **Map** és un simple mapa de parelles clau/valor. Qualsevol valor (tant objectes com valors primitius) poden fer-se servir tant com a clau com a valor.

COL·LECCIONS (MAP), ES6

- Els Objects son similars als Maps en el sentit que tots dos permeten assignar valors a claus, obtenir aquests valors, esborrar claus i detectar si una clau té un valor assignat o no. Degut a això, històricament s'han fet anar Objects com a Maps; tot i això existeixen diferències importants entre Objects i Maps que fan que l'ús de Map sigui millor.

- Un Object té un prototipus, això implica que hi haurà algunes claus definides inicialment. Aquest problem es pot adreçar utilitzant map = Object.create(null).

- Les claus d'un Object són Strings, mentre que les claus d'un Map poden ser de qualsevol tipus.

- És fàcil obtindre la mida d'un Map mentre que el tamany d'un Object ha de ser calculat manualment.

- Utilitzeu mapes en comptes d’objectes quan les claus no se sàpiguin en temps d'execució, o bé quan totes les claus o els valors siguin del mateix tipus.

- Utilitzeu objectes quan hi hagi una lògica que operi els elements individualment.

EXEMPLE MAP

JS

```
let contactes = new Map()
contactes.set('Jessie', {telf: "213-555-1234", adreça: "123 N 1st Ave"})
contactes.has('Jessie') // true
contactes.get('Hilary') // undefined
contactes.set('Hilary', {telf: "617-555-4321", adreça: "321 S 2nd St"})
contactes.get('Jessie') // {phone: "213-555-1234", adreça: "123 N 1st Ave"}
contactes.delete('Raymond') // false
contactes.delete('Jessie') // true
console.log(contactes.size) // 1


//per a clonar un mapa i fer una unió
let clon = new Map(original)

let merged = new Map([...primer, …segon])
let merged_amb_array = new Map([...primer, ...segon, [1, 'eins']])
```

EXEMPLE MAP

JS

```
var mapa = new Map();

var objecte = {},
    clauFuncio = function () {},
    clau = "cadena";

// preparar els valors
mapa.set(clau, "valor associat a amb 'un string'");
mapa.set(objecte, "valor associat amb objecte");
mapa.set(clauFuncio, "valor associat amb with clauFuncio");

mapa.size; // 3

// obtenir els valors
mapa.get(clau);    // "valor associat amb 'un string'"
mapa.get(objecte);       // "valor associat amb objecte"
mapa.get(clauFuncio);      // "valor associat amb clauFuncio"

mapa.get("cadena");   // "valor associat amb 'un string'"
                      // com que keyString === 'un string'
mapa.get({});            // undefined, perquè objecte !== {}
mapa.get(function() {}) // undefined, perquè clauFuncio !== function () {}
```

	Map	Object
Accidental Keys	A `Map` does not contain any keys by default. It only contains what is explicitly put into it.	An `Object` has a prototype, so it contains default keys that could collide with your own keys if you're not careful. **Note:** As of ES5, this can be bypassed by using `Object.create(null)`, but this is seldom done.
Key Types	A `Map`'s keys can be any value (including functions, objects, or any primitive).	The keys of an `Object` must be either a `String` or a `Symbol`.
Key Order	The keys in `Map` are ordered in a simple, straightforward way: A `Map` object iterates entries, keys, and values in the order of entry insertion.	Although the keys of an ordinary `Object` are ordered now, they didn't used to be, and the order is complex. As a result, it's best not to rely on property order. The order was first defined for own properties only in ECMAScript 2015; ECMAScript 2020 defines order for inherited properties as well. See the OrdinaryOwnPropertyKeys and EnumerateObjectProperties abstract specification operations. But note that no single mechanism iterates **all** of an object's properties; the various mechanisms each include different subsets of properties. (`for-in` includes only enumerable string-keyed properties; `Object.keys` includes only own, enumerable, string-keyed properties; `Object.getOwnPropertyNames` includes own, string-keyed properties even if non-enumerable; `Object.getOwnPropertySymbols` does the same for just `Symbol`-keyed properties, etc.)
Size	The number of items in a `Map` is easily retrieved from its `size` property.	The number of items in an `Object` must be determined manually.
Iteration	A `Map` is an iterable, so it can be directly iterated.	Iterating over an `Object` requires obtaining its keys in some fashion and iterating over them.
Performance	Performs better in scenarios involving frequent additions and removals of key-value pairs.	Not optimized for frequent additions and removals of key-value pairs.

ITERANT MAP

JS

```
let mapa = new Map()
mapa.set(0, 'zero')
mapa.set(1, 'un')

for (let [key, valor] of mapa) {
  console.log(key + ' = ' + valor)
}
// 0 = zero
// 1 = un

for (let clau of mapa.keys()) {
  console.log(clau)
}
// 0
// 1

for (let valor of mapa.values()) {
  console.log(valor)
}
// zero
// un

for (let [clau, valor] of mapa.entries()) {
  console.log(clau + ' = ' + valor)
}
// 0 = zero
// 1 = un
```

JS

```
mapa.forEach(function(valor, clau) {
    console.log(clau + ' = ' + valor)
  })
  // 0 = zero
  // 1 = un
```

JS

```
let array = [['key1', 'value1'], ['key2', 'value2']]

//transformen un array en un mapa
let mapa = new Map(array)

mapa.get('key1') // retorna "value1"

//  Array.from() per a passar mapa a array
console.log(Array.from(mapa))

// una manera senzilla de fer-ho amb spread syntax
console.log([...mapa])

// o keys()o values() iteratdrs, i convertir-ho a array
console.log(Array.from(mapa.keys())) // ["key1", "key2"]
```

COL·LECCIONS (SET), ES6/ES7

- L'objecte **Set** permet emmagatzemar valors únics de qualsevol tipus, ja siguin valors primitius o bé referències a objectes.

- Els objectes Set són coleccions de valors, els seus elements poden ser iterats en ordre d'inserció. Un valor només pot aparèixer un cop dins el Set; és únic dins la col·lecció del Set.

- Degut a que cada valor dins el Set ha de ser únic, la igualtat dels valors serà comprovada i aquesta no es basa en el mateix algoritme que l'emprat per l'operador ===

EXEMPLE SET

JS

```
let conjunt = new Set()
conjunt.add(1)            // Set [ 1 ]
conjunt.add(5)            // Set [ 1, 5 ]
conjunt.add(5)            // Set [ 1, 5 ]
conjunt.add('some text') // Set [ 1, 5, 'some text' ]
let o = {a: 1, b: 2}
conjunt.add(o)

conjunt.add({a: 1, b: 2})   // són objectes diferents

conjunt.has(1)              // true
conjunt.has(3)              // false
conjunt.has(5)              // true
conjunt.has(Math.sqrt(25))  // true
conjunt.has('Some Text'.toLowerCase()) // true
conjunt.has(o)        // true

conjunt.size          // 5

conjunt.delete(5)     // esborra 5 del conjunt
conjunt.has(5)        // false

conjunt.size          // 4
console.log(conjunt)
```

JS

```
let conjunt = new Set()

// itera els elemets del conjunt
// 1, "cadena", {"a": 1, "b": 2}, {"a": 1, "b": 2}
for (let item of conjunt) console.log(item)

//  1, "cadena", {"a": 1, "b": 2}, {"a": 1, "b": 2}
for (let item of conjunt.keys()) console.log(item)

//  1, "cadena", {"a": 1, "b": 2}, {"a": 1, "b": 2}
for (let item of conjunt.values()) console.log(item)

// 1, "cadena", {"a": 1, "b": 2}, {"a": 1, "b": 2}
for (let [key, value] of conjunt.entries()) console.log(key)
```

SET I ARRAY

JS

```
//converteix Set a Array.from
let matriu = Array.from(mySet) // [1, "cadena", {"a": 1, "b": 2}, {"a": 1, "b": 2}]

conjunt.add(document.body)
conjunt.has(document.querySelector('body')) // true

//  Set a Array
conjunt2  = new Set([1, 2, 3, 4])
conjunt2.size                     // 4
[...conjunt2]                     // [1, 2, 3, 4]

// intersecció
let interseccio = new Set([...conjunt].filter(x => conjunt2.has(x)))

// diferenncia
let diferencia = new Set([...conjunt].filter(x => !conjunt2.has(x)))

// iteració amb  forEach()
conjunt.forEach(function(value) {
  console.log(value)
})

// eliminant duplicats en un array
const numbers = [2,3,4,4,2,3,3,4,4,5,5,6,6,7,5,32,3,4,5]
console.log([...new Set(numbers)])
// [2, 3, 4, 5, 6, 7, 32]
```

OPERACIONS DE CONJUNTS

JS

```
function isSuperset(set, subset) {
  for (let elem of subset) {
      if (!set.has(elem)) {
          return false
      }
  }
  return true
}

function union(setA, setB) {
  let _union = new Set(setA)
  for (let elem of setB) {
      _union.add(elem)
  }
  return _union
}

function intersection(setA, setB) {
  let _intersection = new Set()
  for (let elem of setB) {
      if (setA.has(elem)) {
          _intersection.add(elem)
      }
  }
  return _intersection
}

function symmetricDifference(setA, setB) {
  let _difference = new Set(setA)
```

OPERACIONS DE CONJUNTS

JS

```
function symmetricDifference(setA, setB) {
  let _difference = new Set(setA)
  for (let elem of setB) {
      if (_difference.has(elem)) {
          _difference.delete(elem)
      } else {
          _difference.add(elem)
      }
  }
  return _difference
}

function difference(setA, setB) {
  let _difference = new Set(setA)
  for (let elem of setB) {
      _difference.delete(elem)
  }
  return _difference
}

let setA = new Set([1, 2, 3, 4])
let setB = new Set([2, 3])
let setC = new Set([3, 4, 5, 6])

isSuperset(setA, setB)          // => true
union(setA, setC)               // => Set [1, 2, 3, 4, 5, 6]
intersection(setA, setC)        // => Set [3, 4]
symmetricDifference(setA, setC) // => Set [1, 2, 5, 6]
difference(setA, setC)          // => Set [1, 2]
```

WEAKMAP

- L'objecte **`WeakMap`** és una col·lecció de parelles clau/valor on les claus són dèbilment referenciades. Les claus han de ser objectes i els valors poden ser valors aribitraris.

- Les claus de WeakMaps són només de tipus Object. Primitive data types com a claus no són permesos (e.g. a Symbol no pot ser una clau WeakMap).

- La clau d'un WeakMap es sostè dèbilment. El que significa que, si no hi ha altres referències fortes a la clau, llavors la entrada sencera serà eliminada del WeakMap pel recol·lector de brossa (garbage collector).

WEAKMAP

- L'objecte **`WeakMap`** és una col·lecció de parelles clau/valor on les claus són dèbilment referenciades. Les claus han de ser objectes i els valors poden ser valors aribitraris.

- Les claus de WeakMaps són només de tipus Object. Primitive data types com a claus no són permesos (e.g. a Symbol no pot ser una clau WeakMap).

- La clau d'un WeakMap es sostè dèbilment. El que significa que, si no hi ha altres referències fortes a la clau, llavors la entrada sencera serà eliminada del WeakMap pel recol·lector de brossa (garbage collector).

- El programador expert en JavaScript s'adonarà que aquesta API es podria implementar en JS amb dos arrays (una per a claus, i una per valors) compartides pels quatre mètodes de l'API. Tal implementació tindria dos inconvenients principals. El primer és que la cerca té un cost de O(n) (on n és el nombre de claus al mapa). El segon és que té problemes de fuita de memòria (memory leak). Amb els mapes mantinguts manualment, l'array de claus mantindria referències a le objectes clau, evitant que aquests fóssin eliminats de memòria pel recol·lector de brossa.

- Als WeakMaps natius, les referències als objectes clau són "dèbils", que vol dir que el recol·lector de brossa pot eliminar l'objecte de memòria si aquest només és referenciat per referències dèbils.

+ Degut a que les referències són dèbils les claus del WeakMap no són enumerables (és a dir, no hi ha cap mètode que us retornarà un llistat de claus). Si aquest mètode existís, aquest dependria de l'estat del recol·lector de brossa, introduïnt un comportament no determinista. Si voleu tenir un llistat amb les claus, l'haureu de mantenir pel vostre compte.

WEAKMAP, ES6

JESUÏTES educació
DAWM06UF2 sergi.grau@fje.edu

JS

```
const wm1 = new WeakMap(),
      wm2 = new WeakMap(),
      wm3 = new WeakMap();
const o1 = {},
      o2 = function() {},
      o3 = window;

wm1.set(o1, 37);
wm1.set(o2, 'azerty');
wm2.set(o1, o2);
wm2.set(o3, undefined);
wm2.set(wm1, wm2); // claus i valors poden ser objectes. sempre WeakMaps!

wm1.get(o2); // "azerty"
wm2.get(o2); // undefined, no hi ha clau 02 a wm2
wm2.get(o3); // undefined

wm1.has(o2); // true
wm2.has(o2); // false
wm2.has(o3); // true (encara que és 'undefined')

wm3.set(o1, 37);
wm3.get(o1); // 37

wm1.has(o1); // true
wm1.delete(o1);
wm1.has(o1); // false
```

WEAKSET

- Els objectes **WeakSet** són col·leccions d'objectes. Un objecte al WeakSet només pot passar un cop, és únic en la col·lecció de WeakSet.

- Al contrari que Sets, WeakSets són únicament col·leccions d'objectes i no de valors arbitraris de qualsevol tipus.

- WeakSet és dèbil (weak): Les referències a la col·lecció es mantenen dèbilment. SI no hi ha cap altra referència a un objecte emmagatzemat en WeakSet, poden ser recollits com a brossa. Això també vol dir que no hi ha cap llista d'objectes actuals emmagatzemats a la col·lecció. WeakSets no són enumerables.

WEAKSET, ES6

JESUÏTES educació
DAWM06UF2 sergi.grau@fje.edu

JS

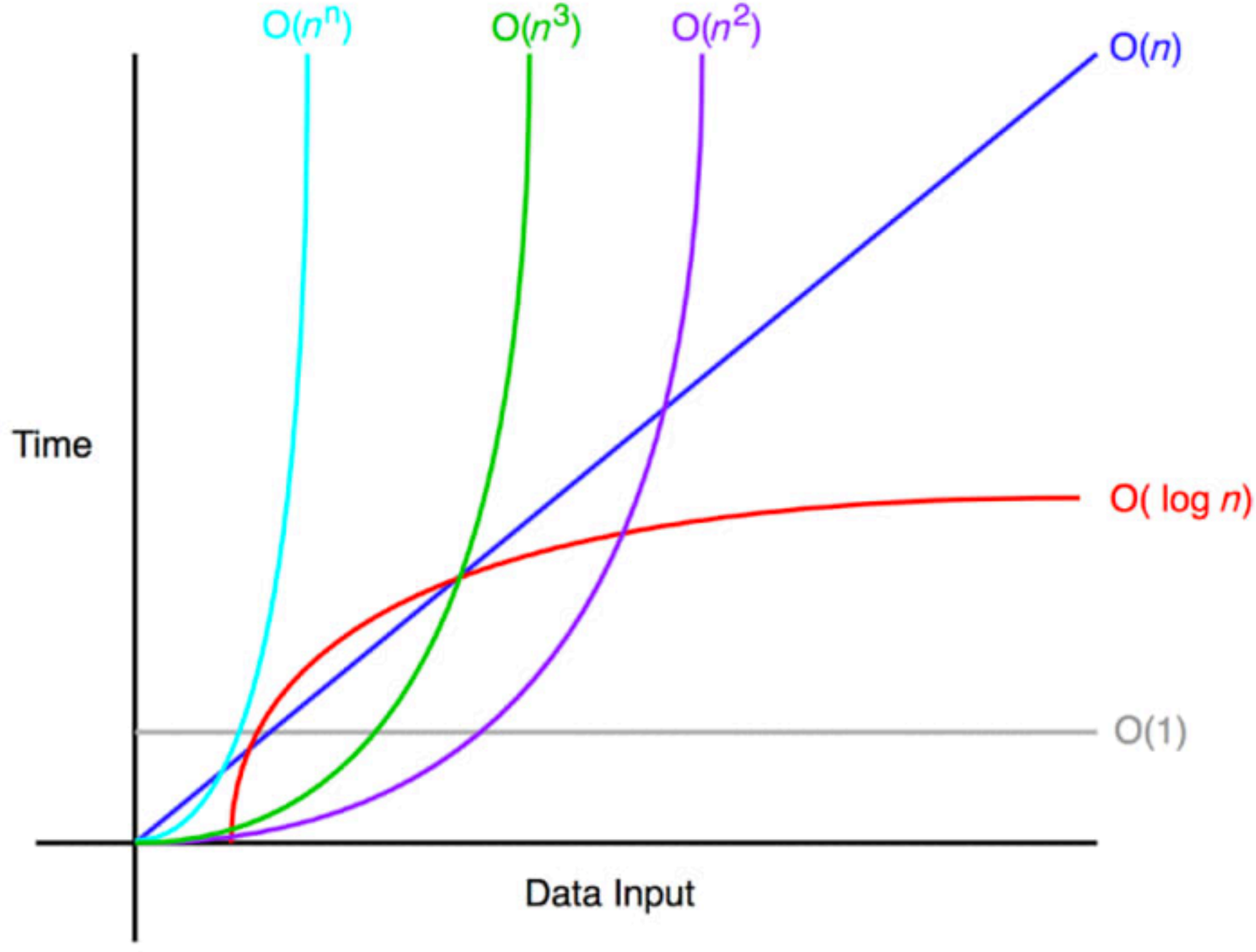
```
const ws = new WeakSet();
const foo = {};
const bar = {};

ws.add(foo);
ws.add(bar);

ws.has(foo);    // true
ws.has(bar);    // true

ws.delete(foo); // esborra foo del conjunt
ws.has(foo);    // false
ws.has(bar);    // true, bar és mantingut
```

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)


O(n^n)
O(n^3)
O(n^2)
O(n)
Time
O(log n)
O(1)
Data Input

BIBLIOGRAFIA

+ https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array
+ https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map
+ https://codeburst.io/array-vs-set-vs-map-vs-object-real-time-use-cases-in-javascript-es6-47ee3295329b
+ https://tc39.es/ecma262/#sec-intro
+ https://www.bigocheatsheet.com/