02. COL.LECIONS, FUNCIONS GLOBALS I DATES EN JAVASCRIPT

IES ESTACIÓ CURS 2021-2022

Col·leccions, funcions globals i dates JavaScript

- 1. Col·leccions.
  - 1.1 Arrays.
  - 1.2 Recorrent arrays.
  - 1.3 Mètodes d'arrays.
  - 1.4 Extensions d'arrays.
  - 1.5 Rest i spread.
  - 1.6 Desestructuració d'arrays.
  - 1.7 Map.
  - 1.8 Set.
- 2. Objectes i funcions globals.
  - 2.1 Funcions globlals.
  - 2.2 El objecte Math.
- 3. Dates.

1. Col·leccions

- **1.1 Arrays.**
  - En JS els arrays són un tipus d'objectes.
  - Podem crear un array amb la instància d'un objecte de classe Array.
  - No tenen una grandària fixa, per tant podem inicialitzar-lo amb una grandària i després afegir-li més elements.
  - El constructor pot rebre 0 paràmetres (array buit), 1 número (la grandària del array), o en qualsevol altre cas, es crearà un array amb els elements rebuts.
  - Hem de tindre en compte que en JS un array pot contindre al mateix temps diferents tipus de dades: **number**, **string**, **boolean**, **object**, **etc**.

```
let  a = new Array(); // Crea un array vacío
a[0] = 13;
console.log(a.length); // Imprime 1
console.log(a[0]); // Imprime 13
console.log(a[1]); // Imprime undefined
```

1. Col·leccions

- **1.1 Arrays.**
  - Si accedim a una posició de l'array que no ha sigut definida retornarà el valor **undefined**.
  - La longitud d'un array depén de les posicions assignades.
  - Exemple: que ocorre quan assignes una posició major que la longitud i que no és consecutiva a l'últim valor assignat.

```
let  a = new Array(12); // Crea un array de tamaño 12
console.log(a.length); // Imprime 12
a[20] = "Hello";
console.log(a.length); // Ahora imprime 21 (0-20). Las posiciones 0-19 tendrán el valor undefined
```

  - Podem reduir la longitud de l'array modificant directament la propietat de la longitud de l'array (**length**).
  - Si reduïm la longitud d'un array, les posicions majors a la nova longitud seran considerades com **undefined (esborrades).**

```
let  a = new Array("a", "b", "c", "d", "e"); // Array con 5 valores
console.log(a[3]); // Imprime "d"
a.length = 2; // Posiciones 2-4 serán destruidas
console.log(a[3]); // Imprime undefined
```

1. Col·leccions

- **1.1 Arrays.**
  - Podem crear un array utilitzant **claudàtors**([ ])en lloc d'usar **new Array().** Els elements que posem dins, separats per coma seran els elements que inicialment tindrà l'array.

```
let  a = ["a", "b", "c", "d", "e"]; // Array de tamaño 5, con 5 valores inicialmente
console.log(typeof a); // Imprime object
console.log(a instanceof Array); // Imprime true. a es una instancia de array
a[a.length] = "f"; // Insertamos in nuevo elemento al final
console.log(a); // Imprime ["a", "b", "c", "d", "e", "f"]
```

1. Col·leccions

- **1.2 Recorrent arrays.**
  - Podem recórrer un array utilitzant bucles **while** i **for**, creant un comptador per **l'índex** que incrementarem en cada iteració.
  - Una altra versió és el bucle **for .... in**, amb aquest bucle podem iterar els índexs d'un array o les propietats d'un objecte. (Similar al **foreach** però recorrent els índexs en compte dels valors).

```
let  ar = new Array(4, 21, 33, 24, 8);

let i = 0;
while(i < ar.length) { // Imprime 4 21 33 24 8
   console.log(ar[i]);
   i++;
}

for(let i = 0; i < ar.length; i++) { // Imprime 4 21 33 24 8
   console.log(ar[i]);
}

for (let i in ar) { // Imprime 4 21 33 24 8
   console.log(ar[i]);
}
```

1. Col·leccions

- **1.2 Recorrent arrays.**
  - Iterant les propietats d'un objecte (s'estudiarà més avant).

```
let person = {
    nombre : "John",
    edad  : 45,
    telefono: "65-453565"
};

/**
 * Imprimirá:
 * nombre: John
 * edad: 45
 * telefono: 65-453565
 */
for (let field in person) {
    console.log(field + ": " + person[field]);
}
```

1. Col·leccions

- **1.2 Recorrent arrays.**
  - També podem iterar els elements d'un array o inclús els caràcters d'una cadena sense utilitzar l'índex, amb el bucle **for ... of** (similar al **foreach**).

```
let a = ["Item1", "Item2", "Item3", "Item4"];

for(let index in a) {
   console.log(a[index]);
}

for(let item of a) { // Hace lo mismo que el bucle anterior
   console.log(item);
}

let str = "abcdefg";

for(let letter of str) {

  if(letter.match(/^[aeiou]$/)) {
     console.log(letter + " es una vocal");
  } else {
     console.log(letter + " es una consonante");
  }
}
```

1. Col·leccions

- **1.3 Mètodes d'arrays.**
  - Inserir valors al principi d'un array **(unshift)** i al final **(push).**

```
let  a = [];
a.push("a"); // Inserta el valor al final del array
a.push("b", "c", "d"); // Inserta estos nuevos valores al final
console.log(a); // Imprime ["a", "b", "c", "d"]
a.unshift("A", "B", "C"); // Inserta nuevos valores al principio del array
console.log(a); // Imprime ["A", "B", "C", "a", "b", "c", "d"]
```

  - Eliminar del principi **(shift)** i del final **(pop)** de l'array.

```
console.log(a.pop()); // Imprime y elimina la última posición → "d"
console.log(a.shift()); // Imprime y elimina la primera posición → "A"
console.log(a); // Imprime ["B", "C", "a", "b", "c"]
```

  - Convertir una array a string utilitzant **join()** en compte de **toString()**.

```
let  a = [3, 21, 15, 61, 9];
console.log(a.join()); // Imprime "3,21,15,61,9""
console.log(a.join(" -#- ")); // Imprime "3 -#- 21 -#- 15 -#- 61 -#- 9"
```

1. Col·leccions

- **1.3 Mètodes d'arrays.**
  - Concatenar dos arrays usant **concat().**

```
let  a = ["a", "b", "c"];
let  b = ["d", "e", "f"];
let  c = a.concat(b);
console.log(c); // Imprime ["a", "b", "c", "d", "e", "f"]
console.log(a); // Imprime ["a", "b", "c"] . El array original no ha sido modificado
```

  - El mètode **slice** retorna un nou sub-array.

```
let a = ["a", "b", "c", "d", "e", "f"];
let b = a.slice(1, 3); // (posición de inicio → incluida, posición final → excluida)
console.log(b); // Imprime ["b", "c"]
console.log(a); // Imprime ["a", "b", "c", "d", "e", "f"]. El array original no es modificado
console.log(a.slice(3)); // Un parámetro. Devuelve desde la posición 3 al final → ["d", "e", "f"]
```

  - **Splice** elimina elements de l'array original i retorna els elements eliminats. També permet inserir nous valors.

```
let  a = ["a", "b", "c", "d", "e", "f"];
a.splice(1, 3); // Elimina 3 elementos desde la posición 1 ("b", "c", "d")
console.log(a); // Imprime ["a", "e", "f"]
a.splice(1,1, "g", "h"); // Elimina 1 elemento en la posición 1 ("e"), e inserta "g", "h" en esa posición
console.log(a); // Imprime ["a", "g", "h", "f"]
a.splice(3, 0, "i"); // En la posición 3, no elimina nada, e inserta "i"
console.log(a); // Imprime ["a", "g", "h", "i", "f"]
```

1. Col·leccions

- **1.3 Mètodes d'arrays.**
  - Podem invertir l'ordre de l'array utilitzant el mètode **reverse().**

```
let a = ["a", "b", "c", "d", "e", "f"];
a.reverse(); // Hace el reverse del array original
console.log(a); // Imprime ["f", "e", "d", "c", "b", "a"]
```

  - El mètode **sort** ordena els elements d'un array.

```
let a = ["Peter", "Anne", "Thomas", "Jen", "Rob", "Alison"];
a.sort(); // Ordena el array original
console.log(a); // Imprime ["Alison", "Anne", "Jen", "Peter", "Rob", "Thomas"]
```

  - ¿Què ocorre si intentem ordenar elements que no són string? Per defecte, l'ordenarà pel seu valor string (hem de tindre en compte que si son objectes, s'intentarà cridar al mètode **toString()** per ordenar-lo). Per a això, haurem de passar una funció (d'ordenació), que comparará 2 valors de l'array i retornarà un valor numèric indicant quin és menor (negatiu si el primer és menor, 0 si són iguals i positiu si el primer és major). https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_sort2

```
let a = [20, 6, 100, 51, 28, 9];
a.sort(); // Ordena el array original
console.log(a); // Imprime [100, 20, 28, 51, 6, 9]
a.sort((n1, n2) => n1 - n2);
console.log(a); // Imprime [6, 9, 20, 28, 51, 100]
```

1. Col·leccions

- **1.3 Mètodes d'arrays.**
  - Exemple: ordenarem persones segons la seua edat.

```
function Persona(nombre, edad) { // Constructor de la clase persona
   this.nombre = name;
   this.edad = edad;

   this.toString = function() { // Método toString()
      return this.nombre + " (" + this.edad + ")";
   }
}

let  personas = [];
personas[0] = new Persona("Thomas", 24);
personas[1] = new Persona("Mary", 15);
personas[2] = new Persona("John", 51);
personas[3] = new Persona("Philippa", 9);

personas.sort((p1, p2) => p1.age - p2.age));
console.log(personas.toString()); // Imprime: "Philippa (9),Mary (15),Thomas (24),John (51)"
```

1. Col·leccions

- **1.3 Mètodes d'arrays.**
  - Usant **indexOf()**, podem conéixer si el valor que li passem es troba en l'array o no. Si ho troba ens retorna la primera posició on està, i si no, ens retorna -1. Usant el mètode **lastIndexOf()** ens retorna la primera ocurrència trobada començant des del final.

```
let a = [3, 21, 15, 61, 9, 15];
console.log(a.indexOf(15)); // Imprime 2
console.log(a.indexOf(56)); // Imprime -1. No encontrado
console.log(a.lastIndexOf(15)); // Imprime 5
```

  - El mètode **every** retorna un boolean indicant si tots els elements de l'array compleixen certa condició. Aquesta funció rebrà qualsevol paràmetre, el testarà, i retornarà true or false en cas que complisca la condició o no.

```
let a = [3, 21, 15, 61, 9, 54];
console.log(a.every(num =>  num < 100));  // Comprueba si cada número es menor a 100. Imprime true
console.log(a.every(num => num % 2 == 0)); // Comprueba si cada número es par. Imprime false
```

  - El mètode **some** es similar a **every**, però retorna true en el moment que uno dels elements de l'array complisca la condició.

```
let a = [3, 21, 15, 61, 9, 54];
console.log(a.some(num => num % 2 == 0));  // Comprueba si algún elemento del array es par. Imprime true
```

1. Col·leccions

- **1.3 Mètodes d'arrays.**
  - Podem iterar pels elements d'un array usant el mètode **forEach**. De manera opcional, podem portar un seguiment de l'índex al qual s'està accedint a cada moment, i fins i tot rebre l'array com tercer paràmetre.

```
let a = [3, 21, 15, 61, 9, 54];
let sum = 0;
a.forEach(num => sum += num);
console.log(sum); // Imprime 163

a.forEach((num, indice, array) => { // índice y array son parámetros opcionales
   console.log("Índice " + indice + " en [" + array + "] es " + num);
}); // Imprime -> Índice 0 en [3,21,15,61,9,54] es 3, Índice 1 en [3,21,15,61,9,54] es 21, ...
```

  - El mètode **map** rep una funció que transforma cada element de l'array i el retorna. És a dir, retorna un nou array de la mateixa gràndaria que conté tots els elements transformats.

```
let  a = [4, 21, 33, 12, 9, 54];
console.log(a.map(num => num*2)); // Imprime [8, 42, 66, 24, 18, 108]
```

  - Per filtrar els elements d'un array i obtindré com a resultat un array que continga només els elements que complissen certa condició utilitzen el mètode **filter**.

```
let  a = [4, 21, 33, 12, 9, 54];
console.log(a.filter(num => num % 2 == 0));  // Imprime [4, 12, 54]
```

1. Col·leccions

- **1.3 Mètodes d'arrays.**
  - El mètode **reduce** usa una funció que acumula un valor, processant cada element (segon paràmetre) amb el valor acumulat (primer paràmetre). Com el segon paràmetre de **reduce**, cal passar un valor inicial. Si no passes un valor inicial, el primer element de l'array serà utilitzat com a tal (si l'array està buit retorna **undefined**).

```
let a = [4, 21, 33, 12, 9, 54];
console.log(a.reduce((total, num) => total + num, 0));  // Suma todos los elementos del array. Imprime 133
console.log(a.reduce((max, num) => num > max? num : max, 0));  // Número máximo del array. Imprime 54
```

  - Per fer el mateix que **reduce,** però al revés, utilitzaren **reduceRight**.

```
let a = [4, 21, 33, 12, 9, 154];
// Comienza con el último número y resta todos los otros números
console.log(a.reduceRight((total, num)  => total - num));
// Imprime 75 (Si no queremos enviarle un valor inicial, empezará con el valor de la última posición del array
```

1. Col·leccions

- **1.4 Extensions d'arrays.**
  - **Array.of(value):** si volem instanciar una array amb només un valor, i aquest és un número, amb **new Array()** no podem fer-ho, ja que el crea buit amb eixe número de posicions.

```
let array = new Array(10); // Array vacío (longitud 10)
let array = Array(10); // Mismo que arriba: array vacío ( longitud 10)
let array = Array.of(10); // Array con longitud 1 -> [10]
let array = [10]; // // Array con longitud 1 -> [10]
```

  - **Array.from(array, func):** funciona de forma similar al mètode **map**, crea un array a partir d'un altre. S'aplica una operació de transformació (funció lambda o anònima) per cada ítem.

```
let array = [4, 5, 12, 21, 33];
let array2 = Array.from(array, n => n * 2);
console.log(array2); // [8, 10, 24, 42, 66]
let array3 = array.map(n => n * 2); // Igual que Array.from
console.log(array3); // [8, 10, 24, 42, 66]
```

  - **Array.fill(value):** aquest mètode sobreescriu totes les posicions d'un array amb un nou valor. És una bona opció per inicialitzar un array creat amb N posicions.

```
let sums = new Array(6); // Array con 6 posiciones
sums.fill(0); // Todas las posiciones se inicializan a 0
console.log(sums); // [0, 0, 0, 0, 0, 0]

let numbers = [2, 4, 6, 9];
numbers.fill(10); // Inicializamos las posiciones  al valor 10
console.log(numbers); // [10, 10, 10, 10]
```

1. Col·leccions

- **1.4 Extensions d'arrays.**
  - **Array.fill(value, start, end):** aquest mètode fa el mateix que l'anterior però omplint l'array des de una posició inicial (inclosa) fins una final (exclosa). Si no s'especifica l'última posició, s'omplirà fins al final.

```
let numbers = [2, 4, 6, 9, 14, 16];
numbers.fill(10, 2, 5); // Las posiciones 2,3,4 se ponen a 10
console.log(numbers); // [2, 4, 10, 10, 10, 16]

let numbers2 = [2, 4, 6, 9, 14, 16];
numbers2.fill(10, -2); // Las dos últimas posiciones se ponen a 10
console.log(numbers2); // [2, 4, 6, 9, 10, 10]
```

  - **Array.find(condition):** troba i retorna el primer valor que complisca la condició establida. Amb **findIndex**, retornem la posició que ocupa eixe valor en l'array.

```
let numbers = [2, 4, 6, 9, 14, 16];
console.log(numbers.find(num => num >= 10)); // Imprime 14 (primer valor encontrado >= 10)
console.log(numbers.findIndex(num => num >= 10)); // Imprime 4 (numbers[4] -> 14)
```

  - **Array.copyWithin(target, startwith):** copia els valors de l'array començant des de la primera posició **startwith**, fins a la posició **target** en la resta de posicions de l'array (en ordre).

```
let numbers = [2, 4, 6, 9, 14, 16];
numbers.copyWithin(3, 0); // [0] -> [3], [1] -> [4], [2] -> [5]
console.log(numbers); // [2, 4, 6, 2, 4, 6]
```

1. Col·leccions

- **1.5 Rest i Spread.**
  - **Rest** és l'acció de transformar un grup de paràmetres en un array i **spread** és justament el contrari, transforma els elements d'un array (o d'un string) a variables.
  - Per utilitzar rest en els paràmetres d'una funció, es declara sempre com últim paràmetre (1 màxim) i li posen tres punts '...' davant. Aquest paràmetre es transformarà automàticament en un array contenint tots els paràmetres restants que li passen a la funció.
  - Si per exemple, el paràmetre **rest** està en la tercera posició, contindrà tots els paràmetres que es passen a excepció del primer i del segon (a partir del tercer).

```
function getMedia(...notas) {
   console.log(notas); // Imprime [5, 7, 8.5, 6.75, 9] (está en un array)
   let total = notas.reduce((total,notas) => total + notas, 0);
   return total / notas.length;
}
console.log(getMedia(5, 7, 8.5, 6.75, 9)); // Imprime 7.25

function imprimirUsuario(nombre, ...lenguajes) {
   console.log(nombre + " sabe " + lenguajes.length + " lenguajes: " + lenguajes.join(" - "));
}

// Imprime "Pedro sabe 3 lenguajes: Java - C# - Python"
printUserInfo("Pedro", "Java", "C#", "Python");
// Imprime "María sabe 5 lenguajes: JavaScript - Angular - PHP - HTML - CSS"
printUserInfo("María", "JavaScript", "Angular", "PHP", "HTML", "CSS");
```

1. Col·leccions

- **1.5 Rest i Spread.**
  - **Spread** és el contrari de **rest.**
  - Si tenim una variable que conté un array, i posen tres punts '...' davant, extraurà tots els seus valors.
  - Per exemple, podem utilitzar el mètode **Math.max**, aquest rep un nombre indeterminat de paràmetres i retorna el major de tots.

```
let nums = [12, 32, 6, 8, 23];
console.log(Math.max(nums)); // Imprime NaN (array no es válido), deben ser números
console.log(Math.max(...nums)); // Imprime 32 -> equivalente a Math.max(12, 32, 6 ,8 ,23)
```

  - Podem utilitzar també spread si necessitem clonar un array.

```
let a = [1, 2, 3, 4];
let b = a; // Referencia el mismo array que 'a' (las modificaciones afectan a ambos).
let c = [...a]; // Nuevo array (copia de a) -> contiene [1, 2, 3, 4]
```

  - També serveix per a concatenar arrays.

```
let a = [1, 2, 3, 4];
let b = [5, 6, 7, 8];
let c = [...a,...b, 9, 10]; //  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

1. Col·leccions

- **1.6 Desestructuració d'arrays.**
  - **Desestructurar** un array és l'acció d'extraure elements individuals d'un array (o propietats d'un objecte) directament en variables individuals. També podem desestructurar un string en caràcters.
  - Exemple: assignem els tres primers elements d'un array, en tres variables diferents, utilitzant una única assignació.

```
let array = [150, 400, 780, 1500, 200];
let [first, second, third] = array; // Asigna los tres primeros elementos del array
console.log(third); // Imprime 780
```

  - Si volem saltar algun valor, deixaren un buit dins dels claudàtors([ ]) i no serà assignat.

```
let array = [150, 400, 780, 1500, 200];
let [first, , third] = array; // Asigna el primer y tercer elemento
console.log(third); // Imprime 780
```

  - Podem assignar la resta de l'array a l'última variable que posen entre claudàtors utilitzant **rest**.

```
let array = [150, 400, 780, 1500, 200];
let [first, second, ...rest] = array; // rest -> array
console.log(rest); // Imprime [780, 1500 ,200]
```

1. Col·leccions

- **1.6 Desestructuració d'arrays.**
  - Si volem assignar més valors dels que conté l'array i no volem obtenir **undefined**, podem utilitzar valors per defecte.

```
let array = ["Peter", "John"];
let [first, second = "Mary", third = "Ann"] = array; // rest -> array
console.log(second); // Imprime "John"
console.log(third); // Imprime "Ann" -> valor por defecto
```

  - També podem desestructurar **arrays anidats**.

```
let sueldos = [["Pedro", "Maria"], [24000, 35400]];
let [[nombre1, nombre2], [sueldo1, sueldo2]] = sueldos;
console.log(nombre1 + " gana " + sueldo1 + "€"); // Imprime "Pedro gana 24000€"
```

  - Un array pot ser desestructurat enviat com a paràmetre a una funció en valors individuals.

```
function imprimirUsuario([id, nombre, email], otraInfo = "Nada") {
    console.log("ID: " + id);
    console.log("Nombre: " + nombre);
    console.log("Email: " + email );
    console.log("Otra info: " + otraInfo );
}

let infoUsu = [3, "Pedro", "peter@gmail.com"];
imprimirUsuario(infoUsu, "No es muy listo");
```

1. Col·leccions

- **1.7 Map.**
  - Un **Map** és una col·lecció que guarda parelles de [clau,valor], els valors són accedits usant la corresponent clau. En JS, un objecte pot ser considerat com un tipus de Mapa però amb algunes limitacions (Només amb strings i enteros com a claus).

```
let obj = {
   0: "Hello",
   1: "World",
   prop1: "This is",
   prop2: "an object"
}

console.log(obj[1]); // Imprime "World"
console.log(obj["prop1"]); // Imprime "This is"
console.log(obj.prop2); // Imprime "an object"
```

1. Col·leccions

- **1.7 Map.**
  - La nova col·lecció **Map** permet usar qualsevol objecte com a clau. Creem la col·lecció usant el constructor **new Map(),** i podem usar els **mètodes set, get i delete** per a emmagatzemar, obtenir o eliminar un valor basat en una clau.

```
let person1 = {name: "Peter", age: 21};
let person2 = {name: "Mary", age: 34};
let person3 = {name: "Louise", age: 17};

let hobbies = new Map(); // Almacenará una persona con un array de hobbies (string)
hobbies.set(person1, ["Tennis", "Computers", "Movies"]);
console.log(hobbies); // Map {Object {name: "Peter", age: 21} => ["Tennis", "Computers", "Movies"]}

hobbies.set(person2, ["Music", "Walking"]);
hobbies.set(person3, ["Boxing", "Eating", "Sleeping"]);
console.log(hobbies);
```

```
Map {Object {name: "Peter", age: 21} => ["Tennis", "Computers", "Movies"], Object
▼ {name: "Mary", age: 34} => ["Music", "Walking"], Object {name: "Louise", age: 17}
  => ["Boxing", "Eating", "Sleeping"]} i
    size: (...)
  ▶ __proto__: Map
  ▼ [[Entries]]: Array[3]
    ▼ 0: {Object => Array[3]}
      ▼ key: Object
          age: 21
          name: "Peter"
        ▶ __proto__: Object
      ▼ value: Array[3]
          0: "Tennis"
          1: "Computers"
          2: "Movies"
          length: 3
```

1. Col·leccions

- **1.7 Map.**
  - Quan usem un objecte com a clau, hem de saber que emmagatzemem una referència al objecte (Després veurem **WeakMap**). Per tant, s'ha d'usar la mateixa referència per a accedir a un valor que per a eliminar-lo en eixe mapa.

```
console.log(hobbies.has(person1)); // true (referencia al objeto original almacenado)
console.log(hobbies.has({name: "Peter", age: 21})); // false (mismas propiedades pero objeto diferente!)
```

  - La propietat **size** retorna la longitud del mapa i podem iterar a través per ell usant [Symbol. iterator] o el bucle **for..of**. Per a cada iteració, es retorna un array amb dues posicions 0 → **key** i 1 → **value**.

```
console.log(hobbies.size); // Imprime 3
hobbies.delete(person2); // Elimina person2 del Map
console.log(hobbies.size); // Imprime 2
console.log(hobbies.get(person3)[2]); // Imprime "Sleeping"

/** Imprime todo:
 * Peter: Tennis, Computers, Movies
 * Louise: Boxing, Eating, Sleeping */
for(let entry of hobbies) {
   console.log(entry[0].name + ": " + entry[1].join(", "));
}
for(let [person, hobArray] of hobbies) { // Mejor
   console.log(person.name + ": " + hobArray.join(", "));
}

hobbies.forEach((hobArray, person) => { // Mejor
   console.log(person.name + ": " + hobArray.join(", "));
});
```

1. Col·leccions

- **1.7 Map.**
  - Si tenim un array que conté altres arrays amb dues posicions (**key, value**), podem crear un mapa directament a partir d'aquest.

```
let prods = [
   ["Computer", 345],
   ["Television", 299],
   ["Table", 65]
];

let prodMap = new Map(prods);
console.log(prodMap); // Map {"Computer" => 345, "Television" => 299, "Table" => 65}
```

1. Col·leccions

- **1.8 Set.**
  - **Set** és com **Map**, però no emmagatzema els valors (només la clau). Pot ser vist com una col·lecció que no permet valors duplicats (en un array pot haver-hi valors duplicats). S'usa, **add, delete** i **has** → són mètodes que retornen un booleà per a emmagatzemar, esborrar i veure si existeix un valor.

```
let set = new Set();
   set.add("John");
   set.add("Mary");
   set.add("Peter");
   set.delete("Peter");
   console.log(set.size); // Imprime 2

    set.add("Mary"); // Mary ya existe
    console.log(set.size); // Imprime 2

    // Itera a través de los valores
    set.forEach(value => {
       console.log(value);
    })
```

  - Podem crear un **Set** des d'un array (la qual cosa elimina els valors duplicats).

```
let names = ["Jennifer", "Alex", "Tony", "Johny", "Alex", "Tony", "Alex"];
let nameSet = new Set(names);
console.log(nameSet); // Set {"Jennifer", "Alex", "Tony", "Johny"}
```

2. Objectes i funcions globals

- Javascript té algunes funcions i objectes globals, les quals poden ser accedides des de qualsevol lloc. Aquestes funcions i objectes són bastant útils per a treballar amb números, cadenes, etc.

- **2.1 Funcions globals.**
  - **parseInt (value)** → Transforma qualsevol valor en un enter. Retorna el valor sencer, o **NaN** si no pot ser convertit.
  - **parseFloat (value)** → Igual que **parseInt**, però retorna un decimal.
  - **IsNaN (value)** → Retorna true si el valor és **NaN**.
  - **isFinite (value)** → Retorna true si el valor és un número finit o false si és infinit.
  - **Number (value)** → Transforma un valor en un número (o **NaN**).
  - **String (value)** → Converteix un valor en un string (en objectes crida a **toString()**).

2. Objectes i funcions globals

- **2.1 Funcions globals.**
  - **encodeURI (string), decodeURI (string)** → Transforma una cadena en una URL codificada, codificant caràcters especials a excepció de: , / ? : @ & = + $ #. Usa **decodeURI** per a transformar-ho a **string** una altra vegada.
    - Decoded: http://domain.com?*val=1 2 3&*val2=r+i%6
    - Encoded: http://domain.com?val=1%202%203&val2=r+y%256
  - **encodeURIComponent (string), decodeURIComponent (string)** → Aquestes funcions també codifiquen i descodifiquen els caràcters especials que encodeURI no fa. S'han d'usar per a codificar elements d'una url com a valors de paràmetres (no la url sencera).
    - Decoded: http://domain.com?val=1 2 3&val2=r+y%6
    - Encoded: “http%3A%2F%2Fdomain.com%3Fval%3D1%202%203%26val2%3Dr%2By%256”

2. Objectes i funcions globals

- **2.2 L'objecte Math.**
  - L'objecte **Math** ens proporciona algunes constants o mètodes bastant útils.
  - **Constants** → E (Número de *Euler), PI, LN2 (algorisme natural en base 2), LN10, LOG2E (base-2 logaritme de E), LOG10E, SQRT1_2 (arrel quadrada de ½), SQRT2.
  - **round (x)** → Arredoneix x a l'enter més pròxim..
  - **floor(x)** → Arredoneix x cap avall (5.99 → 5. Lleva la part decimal).
  - **ceil(x)** → Arredoneix x cap amunt (5.01 → 6)
  - **min(*x1,*x2,...)** → Retorna el número més baix dels arguments que se li passen.
  - **max(*x1,*x2,...)** → Retorna el número més alt dels arguments que se li passen.
  - **pow(x, i)** → Retorna xy (x elevat a i).
  - **abs(x)** → Devuelve el valor absoluto de x.

2. Objectes i funcions globals

- **2.2 L'objecte Math.**
  - **random()** → Retorna un nombre decimal aleatori entre 0 i 1 (no inclosos).
  - **cos(x)** → Retorna el cosinus de x (en radians).
  - **sense(x)** → Retorna el sinus de x.
  - **tan(x)** → Retorna la tangent de x.
  - **sqrt(x)** → Retorna l'arrel quadrada de x

```
console.log("Raíz cuadrada de 9: " + Math.sqrt(9));
console.log("El valor de PI es: " + Math.PI);
console.log(Math.round(4.546342));
// Número aleatorio entre 1 y 10
console.log(Math.floor(Math.random() * 10) + 1);
```

3. Dates

- En Javascript tenim la classe **Date**, que encapsula informació sobre dates i mètodes per a operar, permetent-nos emmagatzemar la data i hora local (timezone).

```
let date = new Date(); // Crea objeto Date almacena la fecha actual
console.log(typeof date); // Imprime object
console.log(date instanceof Date); // Imprime true
console.log(date); // Imprime (en el momento de ejecución) Fri Jun 24 2016 12:27:32 GMT+0200 (CEST)
```

- Podem enviar-li al constructor el nombre de mil·lisegons des del 1/1/1970 a les 00.00:00 GMT (Anomenat **Epoch** o **UNIX time**). Si passem més d'un número, (només el primer i el segon són obligatoris),l'ordre hauria de ser: 1º → any, 2º → mes (0..11), 3º → dia, 4º → hora, 5º → minut, 6º → segon. Una altra opció és passar un string que continga la data en un format vàlid.

```
let date  = new Date(1363754739620); // Nueva fecha 20/03/2013 05:45:39 (milisegundos desde Epoch)
let date2 = new Date(2015, 5, 17, 12, 30, 50); // 17/06/2015 12:30:50 (Mes empieza en 0 -> Ene, ... 11 -> Dic)
let date3 = new Date("2015-03-25"); // Formato de fecha largo sin la hora YYYY-MM-DD (00:00:00)
let date4 = new Date("2015-03-25T12:00:00"); // Formato fecha largo con la fecha
let date5 = new Date("03/25/2015"); // Formato corto MM/DD/YYYY
let date6 = new Date("25 Mar 2015");  // Formato corto con el mes en texto (March también sería válido).
let date7 = new Date("Wed Mar 25 2015 09:56:24 GMT+0100 (CET)"); // Formato completo con el timezone
```

3. Dates

- Si, en lloc d'un objecte Dóna't, volem directament obtindre els mil·lisegons que han passat des del 1/1/1970 (**Epoch**), el que hem de fer és usar els mètodes **Date.parse**(string) i **Date.UTC**(any, mes, dia, hora, minut, segons). També podem usar **Date.now(),** per a la data i hora actual en mil·lisegons.

```
let nowMs = Date.now(); // Momento actual en ms
let dateMs = Date.parse("25 Mar 2015"); // 25 Marzo 2015 en ms
let dateMs2 = Date.UTC(2015, 2, 25); // 25 Marzo 2015 en ms
```

- La classe **Date** té **setters** i **getters** per a les propietats: **fullYear**, **month** (0-11), **date** (dia), **hours**, **minutes**, **seconds**, i **milliseconds**. Si passem un valor negatiu, per exemple, mes = -1, s'estableix l'últim mes (des.) de l'any anterior.

```
// Crea un objeto fecha de hace 2 horas
let twoHoursAgo = new Date(Date.now() - (1000*60*60*2)); // (Ahora - 2 horas) en ms
// Ahora hacemos lo mismo,pero usando el método setHours
let now = new Date();
now.setHours(now.getHours() - 2);
```

3. Dates

- Quan volem imprimir la data, tenim mètodes que ens la retornen en diferents formats:

```
let now = new Date();

console.log(now.toString());
console.log(now.toISOString()); // Imprime 2016-06-26T18:00:31.246Z
console.log(now.toUTCString()); // Imprime Sun, 26 Jun 2016 18:02:48 GMT
console.log(now.toDateString()); // Imprime Sun Jun 26 2016


console.log(now.toLocaleDateString()); // Imprime 26/6/2016
console.log(now.toTimeString()); // Imprime 20:00:31 GMT+0200 (CEST)
console.log(now.toLocaleTimeString()); // Imprime 20:00:31
```