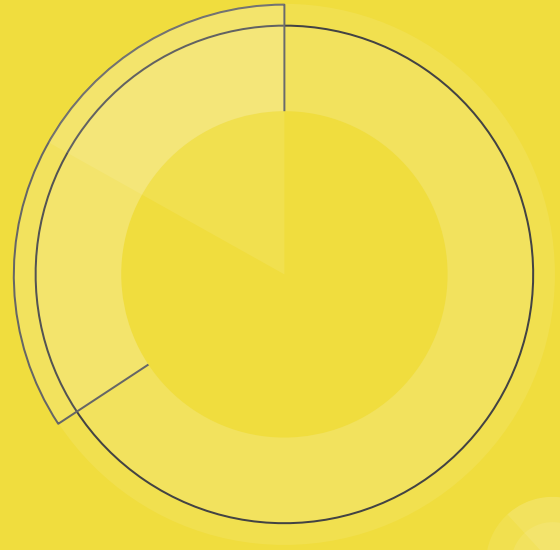


Javascript



Recursos

<https://uniwebsidad.com/libros/javascript>

<https://github.com/statickidz/TemarioDAW/tree/master/DWEC>

<https://github.com/sergarb1/ApuntesDWEC>

<https://developer.mozilla.org/es/docs/Web/JavaScript/Guide>

<https://www.w3schools.com/js/>

<https://github.com/getify/You-Dont-Know-JS/blob/2nd-ed/README.md>

<https://javascript.info/js>

<https://exploringjs.com/es6/index.html>

<https://exploringjs.com/impatient-js/toc.html>

<https://eloquentjavascript.net/>

<https://losapuntesdemajo.vercel.app/>





Integrar Javascript en HTML



```
<script type="text/javascript">  
// codi  
</script>
```

```
<script type="text/javascript" src="scripts.js"></script>
```

```
<noscript>  
<p> El teu navegador no suporta javascript. </p>  
</noscript>
```



Ocultar el codi Javascript



- No es pot ocultar.
- Sols podem ofuscar i comprimir.

```
var _0x47a0=['log','Hello\x20World!']; (function
(_0x558f55,_0x47a08a){var _0x257f99= function (_0x256ed6)
{while(--_0x256ed6) {_0x558f55['push'] (_0x558f55['shift']());
}};_0x257f99(++_0x47a08a);}(_0x47a0,0x1cb));var _0x257f
=function(_0x558f55,_0x47a08a){_0x558f55=_0x558f55-0x0;var
_0x257f99=_0x47a0[_0x558f55];return _0x257f99;};function
hi(){console[_0x257f('0x1')](0x257f('0x0'));}hi();
```



ECMA



- Mocha -> Livescript -> Javascript
- A partir de 1997 el W3C va definir les especificacions.
- La sintaxi és semblant a C i Java, però no té res a veure.
- Especificacions importants:
 - ECMAScript 5th Suportat per tots els navegadors actuals.
 - **ECMAScript 2015 o ES6** Molt suportat actualment, incorpora classes, mòduls, iteradors, funcions fletxa, let, const, promeses...
 - ECMAScript [2022](#), última especificació.



Comentaris



```
// Comentari d'una línia
```

```
/* Comentari  
de  
varies línies */
```



Declaració de Variables

JS

`a = 1;`

No recomanable.
Són globals es declaren on es declaren.

`var a = 1;`

Única manera abans de ES6.

`let a = 1;`

Soluciona problemes de scope de **var**.
No es pot declarar dues vegades.

`const a = 1;`

No es pot modificar el valor.

`var a;
a = 1;`

Declaració amb valor 'undefined'.
Assignació del valor.

`window.a = 1;`

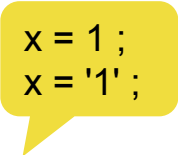
Equivalent a variable global, però
el codi queda més clar.



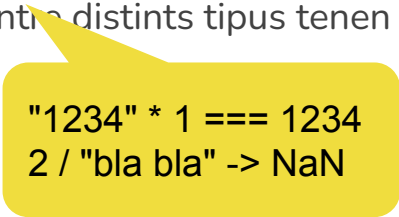
Tipus de variables

JS

- Javascript és un llenguatge **no tipat**.
- **No tipat == No declarar el tipus de dades.**
- No obstant:
 - Una vegada declarades, les variables tenen un tipus.
 - Una variable pot tindre distints tipus al llarg de l'execució
(**Dynamic-Typed**)
 - No hi ha, com en altres, un únic tipus de dades.
 - Algunes operacions entre distints tipus tenen el resultat en un d'ells.
(**Weakly-Typed**)



```
x = 1 ;  
x = '1' ;
```



```
"1234" * 1 === 1234  
2 / "bla bla" -> NaN
```




Tipus suportats



Tipus	Exemple	Descripció
Cadena	"Hola Mon"	Caràcters dins de cometes
Número	9.34	Números en . per a decimals
Boolean	true	true o false
Null	null	Sense valor
Function		Una funció és referenciable com una variable
Object		Objectes com arrays o altres



typeof()

```
var array_mix = [  
  "abcdef", 2 , 2.1 , 2.9e3 , 2e-3 ,  
  0234 , 0x23AF , true , [1,2,3] , {'a': 1, 'b': 2}  
];  
for (let i=0;i<array_mix.length;i++) {  
  console.log(typeof(array_mix[i]));  
}
```

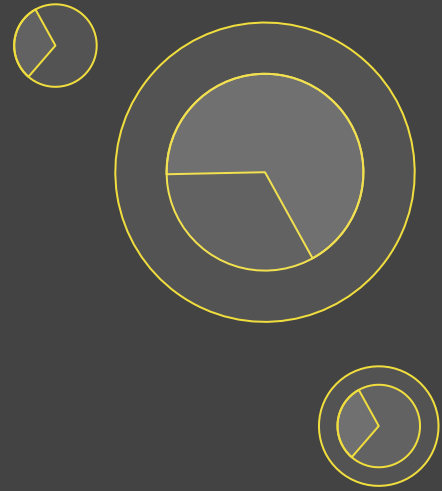


Conversions



- Cadenes a números -> `parseInt()` `parseFloat()`
- Números a cadenes -> `""+3600 = "3600"`
- Longitud de cadenes i arrays -> `(""+3600).length = 4`

Funcions





Funcions



Declaració de funció
(En temps de compilació)

```
function suma_y_mostra(numero1, numero2) {  
  let resultat = numero1 + numero2;  
  alert("El resultat és " + resultat);  
  return resultat;  
}
```

- Javascript no dona error si crides a una funció amb més arguments.
- L'orde dels arguments és important.
- Les funcions poden utilitzar i modificar variables globals. (side-effects)
- Pot o no tindre return.
- Amb () invoques a la funció, sense ella a l'objecte que representa.
- Una funció és un objecte.



Funcions com a variables

JS

```
function toCelsius(fahrenheit) { return (5/9) * (fahrenheit-32); }  
var x = toCelsius(77);  
console.log(`La temperatura és: ${x} C`);  
// Dirèctament en la variable  
x = function toCelsius(fahrenheit) { return (5/9) * (fahrenheit-32); }  
console.log(`La temperatura és: ${x(77)} C`);  
// Sense nom de funció  
x = function (fahrenheit) { return (5/9) * (fahrenheit-32); }  
console.log(`La temperatura és: ${x(77)} C`);
```

Expressió de funció
(En temps d'execució)
(No hoising)

- Amb nom pot ser cridada per si mateixa, però des de fora és la variable.



Àmbit de les funcions

JS

- Les funcions han d'estar en l'àmbit en el que són cridades.
- Les funcions poden ser declarades després de quan es defineixen (amb la sintaxi de la declaració de funció).
- Les funcions no poden ser declarades després si es defineixen amb una expressió de funció.

```
console.log(square(5)); // 25
/* ... */
function square(n) { return n*n }
```

```
console.log(square); // undefined
console.log(square(5)); // TypeError
let square = function (n) {
  return n * n;
}
```



Àmbit (Scope)

JS

Global

```
var a = 1;
function global() {
  console.log(a);
}
global();
console.log(a);
```

Local o de funció

```
function local() {
  var a = 2;
  console.log(a);
}
local();
console.log(a);
```

De bloc

```
for (let i = 0; i < 10; i++) {
  console.log(i);
}
console.log(i); // error
```




Àmbit de les variables en funcions

JS

- Una variable en una funció no pot ser accedida per un altre lloc.
- Una funció pot accedir a les variables globals o a les d'una funció pare.
- Les funcions poden ser anidades, aquesta és la manera de fer variables privades:
 - La funció interna sols és accessible en sentències de la externa.
 - La funció interna forma un tancament (closure), aquesta pot utilitzar les variables de la externa i la externa no pot accedir a les de la interna.
 - La funció externa pot retornar la funció interna.

```
function addSquares(a,b) {  
  function square(x) {  
    return x * x;  
  }  
  return square(a) + square(b);  
}  
  
a = addSquares(2,3); // retorna 13  
b = addSquares(3,4); // retorna 25  
c = addSquares(4,5); // retorna 41
```

```
function outside(x) {  
  function inside(y) {  
    return x + y;  
  }  
  return inside;  
}  
  
let fn_inside = outside(3);  
let result = fn_inside(5); // retorna 8  
let result1 = outside(3)(5); // retorna 8
```



Hoisting



- Permet utilitzar variables o funcions abans de ser declarades.
- Els llenguatges compilats tenen hoising i els interpretats no, no obstant JS és un híbrid que sí té un “precompilat” i per tant té hoising.
- En JS es permet en var i les funcions. En let i const no permet utilitzar abans.
- En qualsevol cas, és millor respectar l'ordre declarant variables globals, funcions globals... abans del codi que s'executarà.



Funcions anònimes



- Quan no necessites que la funció es cride en un altre lloc.
- Per a passar una funció com a argument d'una altra funció.
- Per a guardar una funció en una variable (no Hoising)

```
var nums = [0,1,2];  
var doubledNums = nums.map( function(element){ return element * 2; } ); // [0,2,4]  
var foo = function(){ /*...*/ };
```



Constructor Function



```
var suma = new Function('a','b',"return a + b ");  
console.log(suma(10,20));
```

- No recomanable
- És menys eficient perquè es crea en execució i no en compilació.
- Pot donar problemes de seguretat com eval()



Funcions fletxa



- Una simplificació de les funcions anònimes.
- No es necessita escriure **function**, **ni return ni {}**
- No es comporten com a objectes ni tenen **this**.
- No són Hoisted.
- Es recomana utilitzar **const**, ja que sempre són una constant.
- Si tenen més d'una instrucció necessites els **{}** i el **return**.
- No es poden fer mètodes (al no tindre **this**, no poden accedir a l'objecte) .

```
// ES5
var x = function(x, y) {
  return x * y;
}

// ES6
const x = (x, y) => x * y;
```

```
persona = { nom: 'Pepe', cognom: 'Garcia',
  consulta: function () { return `${this} ${this.nom} ${this.cognom}` },
  consultar: () => `${this} ${this.nom} ${this.cognom}`
}

console.log(persona.consulta(), persona.consultar());
```



Funcions auto-invocades

JS

```
(function () {  
  var aName = "Barry";  
})();  
aName // "Uncaught ReferenceError: aName  
is not defined"  
var result = (function () {  
  var name = "Barry";  
  return name;  
})();  
result; // "Barry"
```

- Si posem () en la definició d'una funció, aquesta es crea i s'executa en el moment, sense que ningú la cride.
- Les variables no són accessibles de fora de la funció.
 - Codi que s'executa immediatament, però aïllat de la resta.
- Accepta arguments amb els () del final.
- Amb una expressió de funció, guarda el que retorne, no la funció.
- Recomanables per al "main" de l'aplicació.



Arguments per defecte

JS

```
var x = function(x=2, y=2) {  
  return x * y;  
}  
  
var multi = function(x,y){  
  if (x === undefined) {x=2;}  
  if (y === undefined) {y=2;}  
  console.log(arguments.length); // és un array  
  return x*y;  
}
```



Call, Apply, Bind



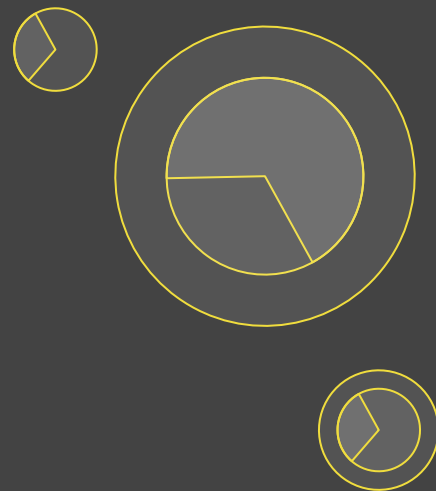
- Call: Per cridar a una funció indicant quin és el seu context d'execució.
- Apply: Igual que Call, però es poden enviar els arguments com un array.
- Bind: Permet crear una funció amb el context indicat.

```
function Car(type, fuelType) {  
  this.type = type;  
  this.fuelType = fuelType;  
}  
  
function setBrand(brand) {  
  Car.call(this, "convertible", "petrol");  
  this.brand = brand;  
  console.log(`Car details = `, this);  
}  
  
const newBrand = new setBrand('Brand1');
```

```
this.x = 9;  
var module = {  
  x: 81,  
  getX: function() { return this.x;  
  }  
};  
  
module.getX(); // 81  
var getX = module.getX;  
getX(); // 9  
  
var boundGetX = getX.bind(module);  
boundGetX(); // 81
```


Elements del llenguatge

https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Expressions_and_Operators





Operadors de comparació

JS

Operator	Description	Example	Result
<code>==</code>	Equal to	<code>1 == 1</code>	true
<code>===</code>	Equal in value and type	<code>1 === '1'</code>	false
<code>!=</code>	Not equal to	<code>1 != 2</code>	true
<code>!==</code>	Not equal in value and type	<code>1 !== '1'</code>	true
<code>></code>	Greater than	<code>1 > 2</code>	false
<code><</code>	Less than	<code>1 < 2</code>	true
<code>>=</code>	Greater than or equal to	<code>1 >= 1</code>	true
<code><=</code>	Less than or equal to	<code>2 <= 1</code>	false



Estructuras de control

JS

```
if ( a === 1 ) { ... } else { ... }  
var h = a < b ? 5 : 10 ;  
for (let i = 0 ; i < 10 ; i++) {...}  
while ( i <= 10 ) {...}  
do {...} while (i <= 10)
```



Comunicació amb l'usuari



- alert, confirm, Prompt:
https://www.w3schools.com/js/js_popup.asp
- [console.log\(\)](#), .error(), .warn(), debug(), info()
 - console.log("%s is %d years old.", "Bob", 42)
 - console.log("%cThis is green text on a yellow background.", "color:green; background-color:yellow");
 - console.dir()
- <https://developer.mozilla.org/es/docs/Web/API/Console>
- CSS en la consola: <https://javascript.plainenglish.io/a-pretty-console-log-786f46d0bc3c> <https://javascript.plainenglish.io/adding-css-to-console-log-dde2e167ee7a>



Template Literals

```
console.log("We don't make mistakes. We just have happy accidents." - Bob Ross'); // escapant la ''
console.log(`"We don't make mistakes. We just have happy accidents.\" - Bob Ross"); // escapant les ""
console.log(`"We don't make mistakes. We just have happy accidents." - Bob Ross`); // escapant amb `
console.log('Homer J. Simpson\n' + '742 Evergreen Terrace\n' + 'Springfield'); // Multiples línies
console.log(`Homer J. Simpson
742 Evergreen Terrace
Springfield`); // En ` es pot fer literal

let a = 2; console.log('La variable a val: '+a); // concatenant string i número
console.log('La variable a val:',a); // log accepta varis arguments
console.log(`La variable a val: ${a}`); //La millor manera, amb ${}
console.log(`${host}/login/oauth/authorize?client_id=${clientId}&scope=${scope}`); // millor per a moltes
variables

let edat = 19; console.log(`L'alumne és: ${edat < 18 ? 'menor' : 'major' }`) // es pot clavar una expressió
```



Arrays

- Es fan en [], separat per comes.
- Es pot accedir a un element com en C o Java: `a[0] = 1;`
- No cal definir la longitud al declarar-lo.
- Poden tindre qualsevol tipus de dades, inclús altres arrays i objectes o funcions.
- Un array és un objecte i es pot construir en new: `var cars = new Array("Saab", "Volvo", "BMW");`
- Un array és un objecte i té mètodes interessants:
 - `a.length`
 - `a.sort()`
 - `a.push()`
- https://www.w3schools.com/js/js_arrays.asp



Recorrer Arrays



```
for (let i =0; i< a.length; i++){ console.log(a[i]);}
```

```
for (let i of a){console.log(i);}
```

```
a.forEach(i => console.log(i))
```



Buscar en Arrays

JS

```
const alligator = ["thick scales", 80, "4 foot tail", "rounded snout"];

alligator.includes("thick scales"); // returns true
alligator.find(el => el.length < 12); // returns '4 foot tail'
alligator.find((el, idx) => typeof el === "string" && idx === 2); // returns '4 foot
tail'
alligator.indexOf("rounded snout"); // returns 3
alligator.filter(el => el === 80); //returns [80, 80]
```




Altres operacions en Arrays



- [splice\(\)](#) Per eliminar elements o agregar nous a partir d'una posició.
(Modifica l'array)
- [slice\(\)](#) Per extreure una porció de l'array (No el modifica, retorna un nou array)
- [flat\(\)](#) Convertir un array multidimensional en un array de menys dimensions.
- [flatMap\(\)](#) Aplicar una funció a cada element i llevar una dimensió a l'array.
(no el modifica)
- [join\(\)](#) Transforma un array en una cadena.
- [split\(\)](#) Transforma una cadena en un array.

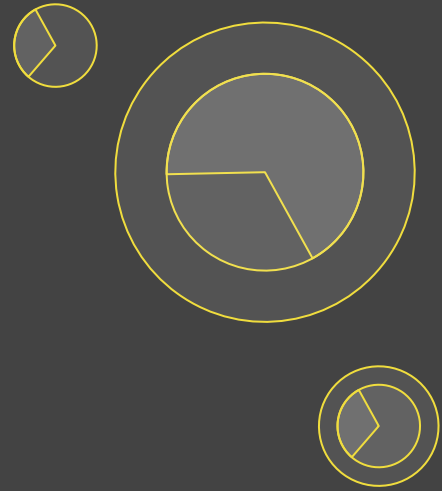


Use strict

A yellow square containing the letters 'JS' in a bold, black, sans-serif font, representing the JavaScript logo.

- use strict, class, let... són funcionalitats de ES6 per fer JS més paregut als llenguatges tradicionals.
- No es poden utilitzar variables no declarades.
- Obliga a utilitzar var, let, const.
- Pot estar en una funció o de forma global.
- No permet utilitzar this. en funcions fora d'objectes. (sense strict és window)
- No és necessari si utilitzem mòduls.

Objectes i classes





Objectes

JS

- Javascript sempre ha suportat objectes (classes no)
- Un objecte és com un array indexat per nom (array associatiu en PHP o diccionari en Python)
- Es pot accedir als atributs amb el . i els []

```
var persona = {  
  nombre: ['Bob', 'Smith'], edad: 32, genero: 'masculino', intereses:  
  ['música', 'esquí'],  
  bio: function () {  
    alert(this.nombre[0] + ' ' + this.nombre[1] + ' tiene ' + this.edad + '  
años. Le gusta ' + this.intereses[0] + ' y ' + this.intereses[1] + '.');  
  }  
};
```



for .. in

JS

```
let user = { name: "John", age: 30 };
alert( "age" in user ); // true, user.age exists
alert( "blabla" in user ); // false, user.blabla doesn't exist

let user = {
  name: "John",
  age: 30,
  isAdmin: true
};

for (let key in user) {
  // keys
  alert( key ); // name, age, isAdmin
  // values for the keys
  alert( user[key] ); // John, 30, true
}
```



Objecte predefinit: String



- Funcions importants: `toLowerCase()`, `concat()`, `charAt()`, `indexOf(text,[index])`, `lastIndexOf(text,[index])`, `replace(text1,text2)`.
- `split(caracter, [troços])`: Separa la cadena per un caracter separador, retorna un array. Trossos, si es defineix, indica la quantitat màxima de trossos.
- `substring(inici, [fin])`: Trau la subcadena donat un principi i un possible fi.



Objecte predefinit: Date

```
// Crea una data amb la data i hora del sistema  
var d=new Date();  
// Crea una data amb una cadena  
d=new Date("October 13, 2014 11:13:00");  
// Crea una data amb any, mes, dia, hora, minuts, segons, milisegons  
d=new Date(99,5,24,11,33,30,0);  
// Crea una data amb any mes i dia  
d=new Date(99,5,24);
```



Objecte predefinit: Date

A yellow square containing the letters 'JS' in a bold, black, sans-serif font, representing JavaScript.

- Funcions importants:
 - `setMonth(mes)`, `getMonth()`; `setDate(dia)`, `getDate()`, `setHours(hora, minut, segon)`, `getHours()`, etc.
 - `getDay()`: Torna del 0 al 6 el dia de la setmana.
 - `toString()`: De data a cadena
 - `toGMTString()`: de data a cadena en format GMT
 - `toUTCString()`: de data a cadena en format UTC



Objecte predefinit: Array



- Mètodes interessants:
 - `Join([separador])`: Crea una cadena amb els elements de l'array amb un separador.
 - `push(element,element2...)`: Clava elements al final.
 - `pop()`: Trau l'últim element i el lleva de l'array.
 - `reverse()`: inverteix l'array.
 - `sort()`: Ordena alfabèticament
 - `slice(inici,[final])`: Trau els elements entre un inici i un final.



Objecte predefinit: Math



- Conté funcions que ens ajuden en operacions matemàtiques.
- Constants importants: E, PI, LN2 (logaritme neperià de 2), LN10, LOG2E (logaritme en base 2 de E), LOG10E.
- Funcions de redondeig: floor(), ceil(), round()
- Funcions matemàtiques: abs(), max(x,y), min(x,y), pow(x,y), random(), sqrt()



Altres objectes predefinits



- Natiu:
 - String
 - Number
 - Boolean
 - Data
 - RegExp
 - Array
 - Funcion
 - Object
- Alt Nivell (depenen del navegador):
 - Window
 - Screen
 - Navigator
 - Location
 - History
 - Document



Classes



- Abans de ES6 les classes es feien en **funcions** i encara s'utilitza molt aquesta tècnica.
- ES6 va incorporar el terme **class** i es pareix més a altres llenguatges.
- Javascript **no funciona per classes**, sinó per prototips.
- Javascript sempre ha tingut objectes, però no classes de la forma tradicional.
- Les classes en ES6 no tenen atributs i funcions privats. Això es fa en **scopes**.
- Les classes són més una tècnica de programació que una funcionalitat del llenguatge.
- Tot són objectes i les classes són objectes funció.



Objectes amb funcions (Mètodes interns)



```
function Apple (type) {  
  this.type = type;  
  this.color = "red";  
  this.getInfo = function() {  
    return this.color + ' ' + this.type + ' apple';  
  };  
}
```

- El que hem fet és declarar la funció “constructor” dels objectes.
- És com una “plantilla” per a crear nous objectes.
- És invocada en new per a ser constructor. this. representa a l'objecte creat.
- L'objecte creat és una instància de la funció.



Objectes amb funcions (Mètodes en prototype)

JS

```
function Apple (type) {  
  this.type = type;  
  this.color = "red";  
}  
Apple.prototype.getInfo = function () {  
  return this.color + ' ' + this.type + ' apple';  
};
```

- Més eficient al no recrear la funció cada vegada que fem un objecte.



Objectes literals

JS

```
var apple = {  
  type: "macintosh",  
  color: "red",  
  getInfo: function () {  
    return this.color + ' ' + this.type + ' apple';  
  }  
}
```

- De vegades s'anomena singleton, una única instància d'un objecte
- Es fan noves instàncies amb Object.create()
- No recomanable per a fer més d'una instància.
- Els objectes creats així són instàncies directes d'Object.



Singleton amb una funció

JS

```
var apple = new function() {  
  this.type = "macintosh";  
  this.color = "red";  
  this.getInfo = function () {  
    return this.color + ' ' + this.type + ' apple';  
  };  
}
```

- new function() fa dos coses, defineix una funció anònima i la invoca amb new. D'aquesta manera tens un singleton amb una funció.



Què és Prototype

JS

- Tots els objectes tenen un prototype.
- El prototype és un objecte que a la vegada té un prototype.
- Varis objectes units per prototypes s'anomenen prototype chain.
- Amb prototype, un objecte pot delegar en altres objectes fills.
- Els objectes tenen un prototype comú Object

```
var homework = {  
  topic: "JS"  
};  
homework.toString(); // [object Object]
```

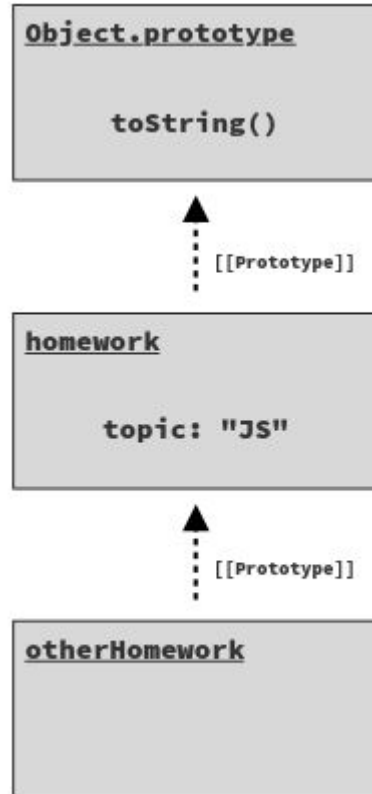
La funció toString() està
en Object.prototype



Object Linkage

```
var homework = {  
  topic: "JS"  
};  
  
var otherHomework = Object.create(homework);  
otherHomework.topic;    // "JS"
```

Prototype Chain





Prototype en Objectes i Funcions

JS

- Les funcions tenen una propietat anomenada `.prototype` que és un objecte amb el constructor (que és ella mateixa) i un prototip que és `Object`
- Els objectes literals o creats amb `new` no tenen `.prototype`.
- Per saber el prototip d'un objecte: `Object.getPrototypeOf(objecte)` (`__proto__` es considera no estàndard)
- Guardar coses en el `.prototype` de les funcions les fa accessibles als objectes creats amb `new` en eixa funció.



Copiar objectes



- Es pot fer en ES6 amb spreading:

```
const copyOfObject = {...originalObject};
```

```
const copyOfArray = [...originalArray];
```

- Es pot fer en Object.assign()

```
Object.assign({}, originalObject)
```

- En cas de necessitat un 'deep copy' s'ha de fer a mà



Object Destructuring

JS

```
// Desestructuració d'arrays  
const foo = ['uno', 'dos', 'tres'];  
const [rojo, amarillo, verde] = foo;  
console.log(rojo); // "uno"  
console.log(amarillo); // "dos"  
console.log(verde); // "tres"
```

```
// Desestructuració d'objectes  
const o = {p: 42, q: true, a: {r: 20, s: 'abc'}};  
const {p, q} = o;  
console.log(p,q); // 42 true  
const {p: foo, q: bar} = o; // Nous noms  
console.log(foo,bar); // 42 true  
var {a} = o;  
var {a: {r: R}} = o; // Objectes anidats i canvi de nom  
console.log(a,R);
```



Object Literal enhancement

A yellow square logo with the letters 'JS' in black, representing JavaScript.

```
const a = 'foo';  
const b = 42;  
const c = {};  
const object1 = { a, b, c }; // No cal fer a: a, b: b ...  
console.log(object1); // Object { a: "foo", b: 42, c: {} }
```



Classes amb class (ES6)



- Totes les classes són funcions, que són objectes.
- Javascript és un llenguatge basat en prototips o classless. Els objectes no es creen instanciant classes, sino clonant altres objectes.
- Cada objecte té una propietat interna anomenada Prototype que pot ser utilitzada per estendre les propietats i mètodes de l'objecte.
- La paraula reservada **class** en ES6 és una comoditat sintactica.
- Molts programadors no la recomanen perquè amaga el que realment està passant.

```
const x = function() {}  
const y = class {}  
const constructorFromFunction = new x();  
const constructorFromClass = new y();
```



Exemple de creació de classes

A yellow square containing the letters 'JS' in a bold, black, sans-serif font, representing JavaScript.

```
function Hero(name, level) {  
    this.name = name;  
    this.level = level;  
}
```

```
class Hero {  
    constructor(name, level) {  
        this.name = name;  
        this.level = level;  
    }  
}
```




Exemple de creació de mètodes

```
function Hero(name, level) {  
  this.name = name;  
  this.level = level;  
}  
  
Hero.prototype.greet = function()  
{  
  return `${this.name} says  
hello.`;  
}
```

```
class Hero {  
  constructor(name, level) {  
    this.name = name;  
    this.level = level;  
  }  
  greet() {  
    return `${this.name} says hello.`;  
  }  
}
```



Herència

```
// Creant un nou constructor a partir  
del pare  
function Mage(name, level, spell) {  
    // Enllaçar constructors amb call()  
    Hero.call(this, name, level);  
    this.spell = spell;  
}
```

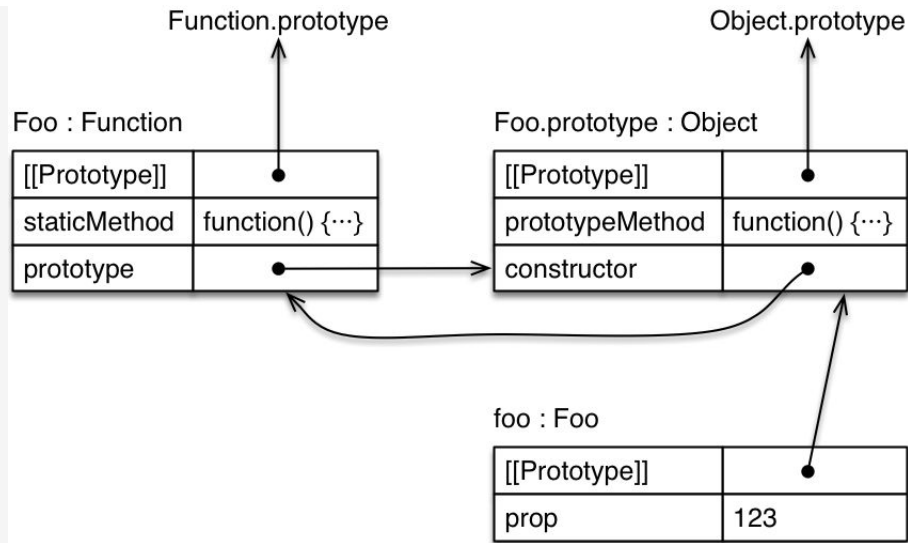
```
class Mage extends Hero {  
    constructor(name, level, spell) {  
        // Enllaçar constructors amb  
super  
        super(name, level);  
        this.spell = spell;  
    }  
}
```



Atributs estàtics



```
class Foo {  
  constructor(prop) {  
    this.prop = prop;  
  }  
  static staticMethod() {  
    return 'classy';  
  }  
  prototypeMethod() {  
    return 'prototypical';  
  }  
}  
const foo = new Foo(123);
```





Classes i atributs privats

- Per defecte, en ES6, tot és públic.
- En [ES2019 han incorporat #](#) per a fer les variables internes privades, però no és compatible en tots els navegadors a dia de hui.
- Si volem atributs privats deguem fer ús de les funcions internes i els scopes.

```
class SmallRectangle {  
  constructor() { let width = 20; let height = 10;  
    this.getDimension = () => { return {width: width, height: height}};  
    this.increaseSize = () => { width++; height++; }; }  
}  
  
const rectangle = new SmallRectangle();  
console.log(rectangle.getDimension());  
console.log(rectangle.height);    // => undefined  
console.log(rectangle.width);     // => undefined
```



Closure

JS

- Una manera de fer variables privades i imitar el comportament de les classes.
- https://www.w3schools.com/js/js_function_closures.asp
- <https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Funciones>
- <https://github.com/getify/You-Dont-Know-JS/blob/2nd-ed/get-started/ch3.md#closure>

```
var add = (function () { // Funció autoinvocada
  var counter = 0; // Closure
  return function () {counter += 1; return counter} // add serà aquesta funció
})();

console.log(add()); // 1
console.log(add()); // 2
```



Closure



- Fa ús de les funcions anidades.
- La funció interna hereta les variables de l'externa.
- Sols podem accedir a la funció interna des de l'externa.
- La funció externa no pot accedir als atributs de la funció interna.
- Els atributs declarats en this. es queden en el context d'execució de la funció interna.

```
function triangle(a,b) {  
  this.a = a;  
  this.b = b; // Amb this es queda al prototype i és accessible  
  console.log(this);  
  var hipo = Math.sqrt(this.a*this.a + this.b*this.b); // no és accessible  
  this.hipotenusa = function () {  
    return `La Hipotenusa es: ${hipo}`; // La funció sí que té accés  
  }  
}  
  
var t = new triangle(10,20); // prova a llevar el new  
console.log(t.hipotenusa(),t.hipo); // 22.36 undefined
```



Setters i Getters

- Si tenim atributs privats o que han de ser obtinguts amb un control previ podem fer Setters i Getters.
- Es poden utilitzar com si foren propietats de classe.

JS

```
class Producte {  
    constructor(nom,preu) {  
        this.nom = nom; this.preu = preu;  
    }  
    set setPreu(preu) {  
        if(!isNaN(preu)) this.preu = preu;  
        else this.preu = 0  
    }  
    get getPreu() {  
        return parseFloat(this.preu);  
    }  
}  
  
let p1 = new Producte(PC,1000);  
p1.setPreu = 900;
```



This

- En una funció representa al context d'execució. El context depèn de com és cridada.
- Si s'executa fora d'un objecte, this és l'objecte Window.
- Executada en mode strict, this sempre necessita un objecte com a context.

```
function classroom(teacher) {  
  // "use strict";  // prova el mode estricte  
  this.plant = 3;  // sense new, this és window  
  console.log(this);  
  return function study() {  
    console.log(  
      `${ teacher } says to study ${ this.topic  
    } in plant ${this.plant}`  
    );  
  };  
}  
  
let assignment = classroom("Kyle");  // Prova a  
// ficar el new  
console.log(assignment);  
assignment();  
  
let clase = { topic: 'mates',  
  plant: '5',  // prova a comentar aquesta línia  
  assignment: assignment  
}  
  
clase.assignment();
```




This segons com invoquem

- Invocació simple: (En el codi global)
This és l'objecte global window o undefined en mode strict.
- Invocació com a mètode: (Dins d'un objecte) This és l'objecte que conté el mètode.
- Invocació indirecta: (.call() o .apply())
This és el primer argument de la invocació.
- En el constructor: És l'objecte que s'està creant.

```
// Simple Invocation
function simpleInvocation() {console.log(this);}
simpleInvocatoin();

// Method Invocation
const methodInvocation = { method(){ console.log(this);}};
methodInvocation.method();

// Indirect Invocation
const context = { value1: 'A', value2: 'B' };
function indirectInvocation() { console.log(this);}
indirectInvocation.call(context);
indirectInvocation.apply(context);

// Constructor Invocation
function constructorInvocation() { console.log(this);}
new constructorInvocation();
```



This i That

JS

- En ocasions, les funcions anidades tenen que accedir al this de la funció superior i no poden.
- Alguns programadors utilitzen self (no recomanat), thiz o that per a guardar el this.
- El més recomanable és utilitzar noms de variables més semàntics.

```
(function () {  
  "use strict";  
  document.addEventListener("DOMContentLoaded", function () {  
    var colours = ["red", "green", "blue"];  
    document.getElementById("header")  
      .addEventListener("click", function () {  
        // this és una referència al clicat  
        var that = this;  
        colours.forEach(function (element, index) {  
          console.log(this, that, index, element);  
          // this és undefined  
          // that és el que s'ha clicat  
        });  
      });  
  });  
})();
```



This i funcions fletxa

JS

```
function UiComponent() {  
    var _this = this;  
    var button =  
document.getElementById('myButton');  
    button.addEventListener('click', function ()  
{  
        console.log('CLICK');  
        _this.handleClick();  
    });  
}  
UiComponent.prototype.handleClick = function ()  
{  
    ...  
};
```

```
function UiComponent() {  
    var button =  
document.getElementById('myButton');  
    button.addEventListener('click', () => {  
        console.log('CLICK');  
        this.handleClick(); // (A)  
    });  
}
```



This i funcions fletxa

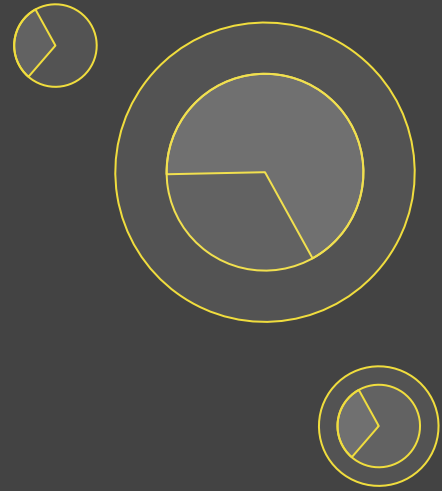
- This és l'objecte del context extern de la funció.
- En mode estricte, this pot donar problemes.
- Són molt útils per a callbacks dins de mètodes, ja que encara fan referència a l'objecte del mètode. (diapositiva anterior)

<https://blog.bitsrc.io/arrow-functions-vs-regular-functions-in-javascript-458ccd863bc1>

```
var variable = "Global Level Variable";
let myObject = {
  variable: "Object Level Variable",
  arrowFunction: () => {
    console.log(this.variable);
  },
  regularFunction() {
    console.log(this.variable);
  }
};

myObject.arrowFunction();
myObject.regularFunction();
```

DOM

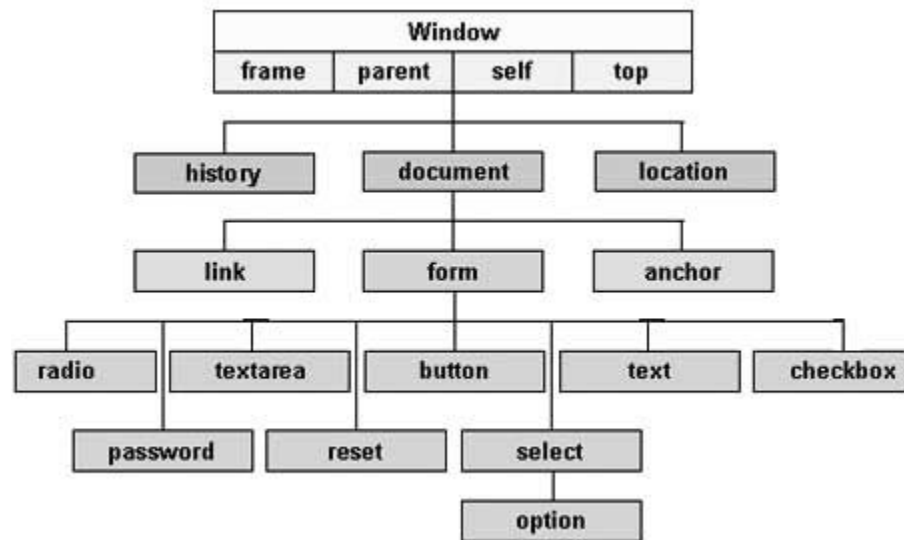




DOM



- Estructura del document HTML en forma d'objectes.
- Window: La finestra on va tot.
- Document: La pàgina web.





DOM : Window

- És un objecte predefinit.
 - Els mètodes alert(), prompt()... són de Window, però no cal nombrar a l'objecte.
 - Entre altres mètodes, destaquem de moment:
 - setTimeout(funció, temps): Espera el temps que diguem per executar una funció.
 - setInterval(funció, temps): Cada cert temps, executa la funció.
 - clearTimeout(identificador): Anula el timeOut.



DOM: Trobar els nodes

A yellow square containing the letters 'JS' in a bold, black, sans-serif font, representing JavaScript.

- Normalment es cerca sobre el document, encara que es pot cercar sobre un node i els seus fills.
- `document.getElementById()`
- `getElementsByTagName()`
- `getElementsByName()`
- `querySelector()`, `querySelectorAll()`
- `.children` `.firstChild` `.lastChild` `.parentElement`



DOM: Modificar els nodes

JS

- .innerHTML, .innerText, outerHTML
- .insertAdjacentHTML()
- .append('contingut', <etiqueta>), .append(node)
- .prepend
- .after() .before()
- .cloneNode(deep)
- .remove()
- .className
- .style.color = "blue";
- Més antics:
 - removeChild()
 - appendChild()

No recomanable per a nous nodes ja que força a remapejar el DOM

```
elementsQuery.parentNode.removeChild(elementsQuery);  
// Per a esborrar cal cridar al node pare
```



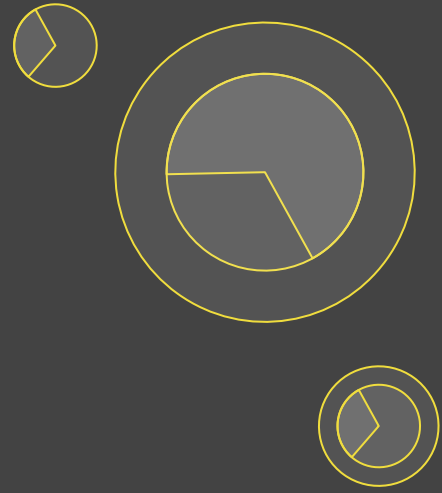
Esperar a que carregue el DOM

JS

```
(function () {  
  "use strict";  
  document.addEventListener("DOMContentLoaded", function () {  
    for (let i = 0; i < 100; i++) {  
      let contenidor = document.getElementById("content");  
      let numero = document.createElement("p");  
      numero.innerHTML = i;  
      contenidor.appendChild(numero);  
    }  
  });  
})();
```

També podem posar el script al final del body.

Formularis





Formularis



- En una aplicació web, els formularis es validen tant en el client com en el servidor.
- El contingut dels inputs es veu o modifica en l'atribut **value**:
 - Radio Button i Checkbox: Han de tindre un **name** comú. Atribut **value** i atribut **checked**
 - Select: Atribut **options** i **selectedIndex**



Esdeveniments en formularis

JS

```
<form onsubmit="return validar();">
```

- Si la funció validar retorna true, s'envia el formulari.
- Dins de validar podem fer les comprovacions necessàries.

```
<form onsubmit="this.disabled=true">
```

- Evita que es torne a enviar el formulari

```
let elemento=document.getElementById("formulario");  
elemento.submit();
```

- Enviar formulari per Javascript
- onsubmit="event.preventDefault();
-

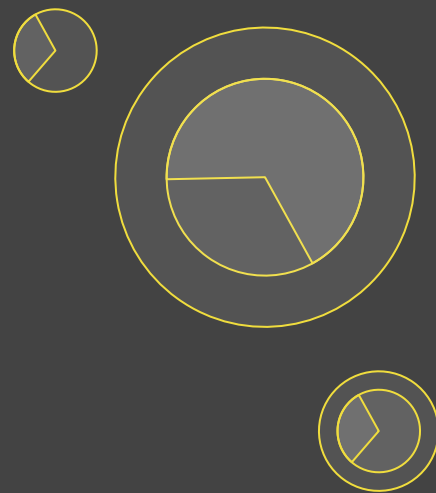


Expressions regulars

JS

```
function validate()
{
  var phoneNumber = document.getElementById('phone-number').value;
  var phoneRGEX = /^[({}{0,1}[0-9]{3}[)]{0,1}[-\s\.] {0,1}[0-9]{3}[-\s\.] {0,1}[0-9]{4}$/;
  var phoneResult = phoneRGEX.test(phoneNumber);
  alert("phone:"+phoneResult );
}
```

Esdeveniments





Events & Handlers



- Esdeveniments (Events):
 - Mecanisme a accionar quan l'usuari interacciona amb la web.
 - Es pot capturar l'esdeveniment per a que es realitze una tasca.
 - L'encarregat de gestionar els esdeveniments és el DOM.
 - Un **Objecte** Event amb propietats.
- Manejador (Handler):
 - Acció que es va a manejar, per exemple a l'esdeveniment 'click' el seu manejador és 'onClick'
 - Una **funció**.



Events en línia



- Es diu el que s'ha de fer en el codi HTML (No recomanable)

```
<p onmouseover="this.style.background='#FF0000';"  
onmouseout="this.style.background='#FFFFFF';">HOLA</p  
>
```



Registre d'esdeveniments tradicional



- La pàgina ha d'estar completament carregada. Es pot fer que espere amb `window.onload`.
- No recomanable, sobretot per no poder assignar més d'un handler a un event.

```
window.onload = function () {  
    document.getElementById('hola').onmouseover = function () {  
this.style.background = '#FF0000';};  
    document.getElementById('hola').onmouseout = function () {  
this.style.background = '#FFFFFF';};  
}
```



Registre d'esdeveniments avançat W3C

JS

- `addEventListener`("event sense on",funció,false)
- Es poden afegir més funcions al mateix esdeveniment.
- No funciona en IE o Edge (Buscar per `attachEvent`)
- `removeEventListener`

```
(function () {  
    "use strict";  
    document.addEventListener("DOMContentLoaded", function () {  
        document.getElementById('hola').addEventListener('mouseover',function () { this.style.background =  
'#FF0000';} , false);  
        document.getElementById('hola').addEventListener('mouseout',function () { this.style.background =  
'#FFFFFF';} , false);  
    });  
})();
```



Treure informació del Event

```
(function () {  
    "use strict";  
    document.addEventListener("DOMContentLoaded", function () {  
        document.getElementById('hola').addEventListener('mouseover', manejador, false);  
        document.getElementById('hola').addEventListener('mouseout', manejador, false);  
    });  
})();  
  
function manejador(e) {  
    console.log(e.type, e.target);  
    if (e.type == 'mouseover') {this.style.background = '#FF0000'; } // Quí és this?  
    if (e.type == 'mouseout') {this.style.background = '#FFFFFF'; }  
    if (e.target.id == 'hola' ) {console.log('Hola');}  
}
```



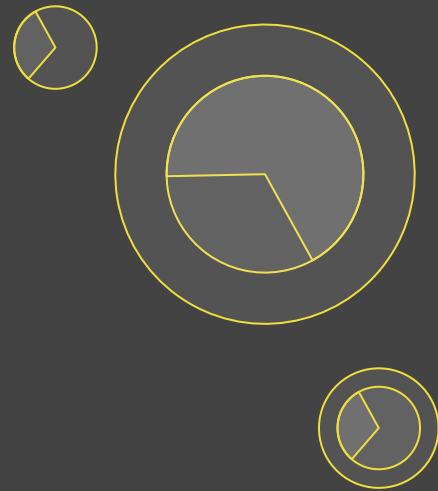
Propagació i captura dels esdeveniments



- Un esdeveniment es propaga de l'element en el que passa als seus pares.
- Tant l'element com els pares el poden capturar durant la propagació i fer coses distintes.
- Si volem manejar el mateix esdeveniment en molts elements es pot ficar al pare i dins de la funció preguntar a **event.target** quin element és el primer.
- Es pot bloquejar la propagació amb **event.stopPropagation()**

```
document.querySelector("table").addEventListener("click", (event) =>
{
    if (event.target.nodeName == 'TD')
        event.target.style.background = "rgb(230, 226, 40)";
}
);
```

Iteradors





Iterables

JS

- Els més utilitzats són: Array, Set, Map i String.
- Els objectes normals no són iterables.
- Un iterable permet ser consumit amb for ... of o Spread Operators (...)
- Per a que un objecte siga iterable ha de tindre la propietat **Symbol.iterator** que retorna un iterador.
- Un iterador té un mètode per recorre els seus valors finits o infinits:
- next() Aquest retorna un objecte amb les propietats done (boolean) i value (el valor)

```
var iterator = someString[Symbol.iterator]();
```

```
iterator.next();
```

```
iterator.next();
```

```
iterator.next();
```

Funció generadora

```
var myIterable = {};
```

```
myIterable[Symbol.iterator] = function* () {
```

```
  yield 1;
```

```
  yield 2;
```

```
  yield 3;
```

```
};
```

```
[...myIterable]; // [1, 2, 3]
```



Iteradors (ES6)

JS

- L'estructura de control for ... of ... permet recórrer un iterable.
- Els iterables més evidents són els arrays. Altres són els Strings, Maps o Sets
- L'operador ... (de propagació) permet expandir un iterador.
- Encara que no coneixem l'estructura interna, un Iterable sempre proporcionarà les funcions **hasNext()** i **next()**

```
let iterable = [10, 20, 30];  
for (let value of iterable) { value += 1; console.log(value); }
```

```
function f(x, y, z) { }  
var args = [0, 1, 2];  
f(...args);
```

```
var parts = ['shoulder', 'knees'];  
var lyrics = ['head', ...parts, 'and', 'toes'];
```




Recorrer iterables



querySelector retorna un NodeList (objecte) que conté el seu forEach

- `forEach(function(item,index,array){})`
 - Una funció per a cada element del iterable.
 - No retorna res.
- `map(function(item,index,array){})`
 - Retorna un array amb el resultat de la funció.
 - El callback sempre tindrà return.
- `filter(function(item,index,array){})`
 - Retorna un array amb els elements que passen el filtre.
 - El callback sempre retornarà un boolean.
- `reduce(function(Anterior,item,index,array){})`
 - Retorna un únic valor calculat amb els elements de l'array.

```
let a = [1,2,3,4,5];

a.forEach(element => { console.log(element); });
b = a.map(function(item) {return item**2;});
console.log(b);
console.log(a.filter(function(item) {return item%2 == 0; }));
var total = a.reduce(function (previous, current) {
  console.log(previous,current);
  return previous + current;
}, 0);
console.log(total);
```



Set()



- Objecte Iterable que representa a una col·lecció de valors.
- No es poden repetir valors (no funciona en objectes).
- Té funcions com `.add()`, `.delete()`, `clear()` o `.has()`
- Per saber la longitud és en `.size`
- Es pot recorre com un array amb `for .. of` o `forEach`.
- Es pot transformar en array amb l'operador spread: `[...mySet]`
- No pot ser accedit de forma aleatòria com un array.
- Útil com a abstracció i per garantir que no es repeteix cap element.



Map()



- És una col·lecció de clau-valor (com un objecte)
- Les claus poden ser més que números o lletres, poden ser de qualsevol tipus
- Els Maps tenen .size. Mentre que no es pot saber directament la mida d'un objecte.
- Es pot iterar directament en un Map.
- Els Maps no tenen un prototip com l'objecte.
- Té funcions com clear(), delete(key), entries(), get(), has(), keys(), set(key,value), values()
- Es pot utilitzar quan vols un objecte o paraules clau com a claus. També quan vols iterar freqüentment o saber la longitud. (Molt útil en dades tretes d'un API).

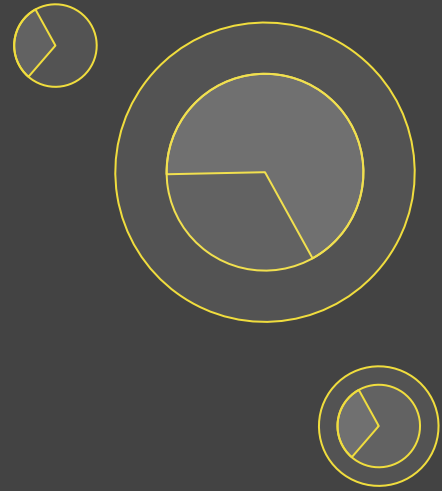


Comparativa



	Maps	Objectes	Sets	Arrays
Recorrer	Si	No	Si	Si
Repetits	Valors si	Els valors si	No	Si
Clau-valor	Objectes com a clau	Si	El valor és la clau	La clau és l'índex
Esborrar	.delete()	delete	.delete()	No directament
Filterar, ordenar, map	No	No	No	Si
Accés aleatori	Si	Si	No	Si

Mòduls





Mòduls



- Col·lecció de dades i funcions útils per al programa principal.
- Ha de garantir:
 - Encapsulació
 - POLE (Principle Of Less Exposure)
- Tindrà una API pública i pot tindre detalls privats.
- Pot ser stateful, és a dir, guardar informació.
- Codi en varis fitxers .js sense mòduls:
 - El espai de noms globals es plena de variables i funcions.
 - Encara que estiga en varis fitxers, és igual que si estigués tot seguit i l'ordre importa.



Coses que no són mòduls (Espais de noms)

JS

```
// namespace, not module
var Utils = {
  cancelEvt(evt) {
    evt.preventDefault(); evt.stopPropagation(); evt.stopImmediatePropagation();
  },
  wait(ms) {
    return new Promise(function c(res) { setTimeout(res, ms); });
  },
  isValidEmail(email) {
    return /^[^@]+@[^@.]+\.[^@.]+/.test(email);
  }
};
```



Coses que no són mòduls (Estructures de dades)

JS

```
// data structure, not module
var Student = {
  records: [{ id: 14, name: "Kyle", grade: 86 }, { id: 73, name: "Suzy", grade: 87 }, { id: 112, name: "Frank", grade: 75 }, { id: 6, name: "Sarah", grade: 91 }],
  getName(studentID) {
    var student = this.records.find(student => student.id == studentID);
    return student.name;
  }
};
Student.getName(73);
// Suzy
```




Mòduls abans de ES6



- CommonJS
- Asynchronous Module Definition (AMD)
- Universal Module Definition (UMD)
- Ningú d'ells és Javascript natiu i necessiten biblioteques.
- També es fan fer ferramentes com Babel o Webpack que tradueixen després en codi sense mòduls.



Mòduls manuals (Amb estat i control d'accés)

JS

```
function defineStudent() {  
  var records = [{ id: 14, name: "Kyle", grade: 86 }, { id: 73, name: "Suzy", grade:  
87 }, { id: 112, name: "Frank", grade: 75 }, { id: 6, name: "Sarah", grade: 91 }];  
  var publicAPI = {getName};  
  return publicAPI;  
  function getName(studentID) {  
    var student = records.find(student => student.id == studentID);  
    return student.name;  
  }  
}  
  
var fullTime = defineStudent();  
fullTime.getName(73);           // Suzy
```



Mòduls ES6



- Utilitzen import i export
- No afegixen res a l'àmbit global
- Sempre estan en mode estricte
- Necessiten un servidor, ja que demana els fitxers en http.



Móduls moderns ESM

A yellow square containing the letters "JS" in a bold, black, sans-serif font.

```
export { getName };  
  
var records = [{ id: 14, name: "Kyle", grade: 86 }, { id: 73, name: "Suzy", grade:  
87 }, { id: 112, name: "Frank", grade: 75 }, { id: 6, name: "Sarah", grade: 91 }];  
  
function getName(studentID) {  
  var student = records.find(  
    student => student.id == studentID  
  );  
  return student.name;  
}
```

```
export function getName(studentID) {}  
export default function getName(studentID) {
```



Móduls moderns ESM

```
import { getName } from "/path/to/students.js";  
// getName(73);  
import { getName as getStudentName } from "/path/to/students.js";  
import getName from "/path/to/students.js"; // Si getname és default  
import { default as getName, /* .. others .. */ } from "/path/to/students.js";  
import * as Student from "/path/to/students.js";  
// Student.getName(73);
```



<script> amb mòduls



```
<script type="module" src="functions.js"></script>  
<script type="module" src="script.js"></script>
```