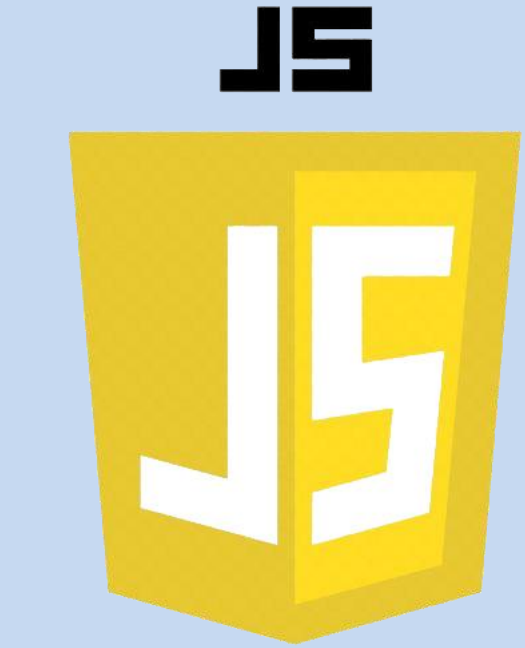


JavaScript

i

jQuery

Omar del Río García

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

CONTINGUT

Generalitat de Catalunya
MINISTERIO DE EDUCACIÓN, FORMACIÓN PROFESIONAL Y DEPORTES
MINISTERIO DE TRABAJO Y ECONOMÍA SOCIAL
SERVICIO PÚBLICO DE EMPLEO ESTATAL SEPE SERVEI PÚBLIC D'OCUPACIÓ ESTATAL
Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

INTRODUCCIÓ

TIPOLOGIES DE LLENGUATGES DE PROGRAMACIÓ

Un **llenguatge de programació** és un idioma artificial dissenyat per a expressar processos que poden ser reproduïts per màquines. Es fan servir per a crear programes que controlen el comportament lògic d'una màquina i per a expressar algoritmes amb precisió.

En diem llenguatge perquè està format per un conjunt de símbols, regles sintàctiques i semàntiques que en defineixen l'estructura i el significat dels elements i expressions.

Els **llenguatges interpretats** són aquells que requereixen un programa auxiliar o intèrpret que tradueix el llenguatge a binari per tal que la màquina el pugui processar i executar. Exemples: PHP, Phyton, JavaScript, etc.

Els **llenguatges compilats** necessiten un programa annex anomenat compilador que fa la transformació a un llenguatge intel·ligible per a la màquina i genera un arxiu que es pot executar sense la necessitat de cap altre programa intermediari; és el que s'anomena arxiu executable. Exemples: C, C++, Java, etc.

Els **llenguatges transpilats** són aquells que, abans d'executar-se, es transformen en un altre llenguatge de nivell similar, habitualment per motius de compatibilitat o per aprofitar característiques avançades no disponibles en totes les plataformes. En aquest procés, un transpiler converteix el codi font original en un altre codi font equivalent però més àmpliament suportat. Exemples: Haxe, Sass/SCSS, TypeScript, etc.

CONCEPTUALITZACIÓ DE LES BASES DEL PENSAMENT COMPUTACIONAL

El pensament computacional es basa a pensar de la mateixa manera que ho faria un científic informàtic quan ens enfrontem a un problema. En altres paraules, és un procés que permet formular problemes de manera que les seves solucions poden ser representades com a seqüències d'instruccions i algoritmes.

Aquest tipus de pensament el podem definir com un procés de reconeixement d'aspectes relacionats amb la informàtica en la qual s'apliquen eines i tècniques per a comprendre, raonar i solucionar problemes tant naturals com artificials. Aquestes característiques són l'**abstracció**, el **pensament algorítmic**, la **descomposició** i el **reconeixement de patrons**.

És probable que durant el procés de resolució d'aquests problemes existeixi informació irrellevant. L'**abstracció** és la característica de prescindir de la informació irrellevant perquè en la taula estigui només la informació necessària per al compliment de l'objectiu.

El **pensament algorítmic** és una altra de les característiques del pensament computacional: és necessari per comunicar i interpretar una sèrie d'instruccions ordenades que ens portin a un resultat concret i predictible.

És a dir, el pensament algorítmic ens permet automatitzar solucions. Un bon exemple d'aquest pensament és la cuina: les receptes són algoritmes en sí mateix. Com es prepara un sandvitx de mantega de cacauet i melmelada? Pensar i escriure els passos necessaris, sense oblidar-ne cap

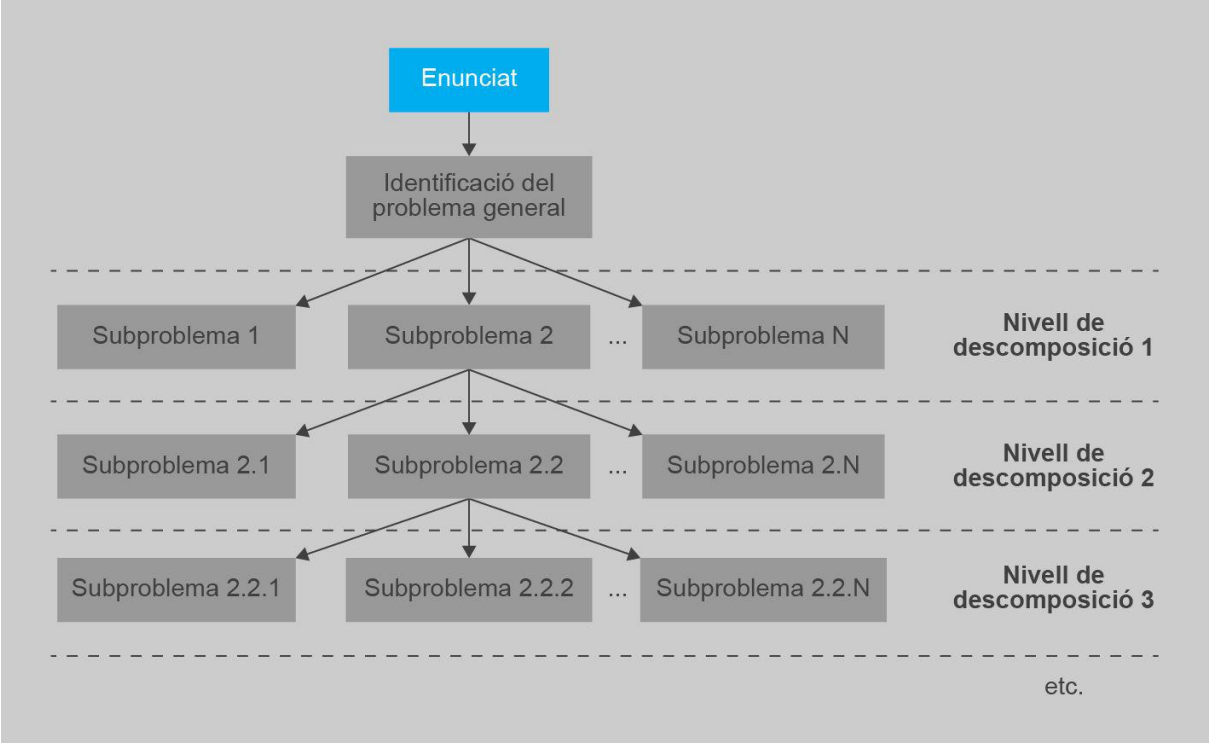
Generalitat de Catalunya
MINISTERIO DE EDUCACIÓN, FORMACIÓN PROFESIONAL Y DEPORTES
MINISTERIO DE TRABAJO Y ECONOMÍA SOCIAL
SERVICIO PÚBLICO DE EMPLEO ESTATAL SEPE SERVEI PÚBLIC D'OCUPACIÓ ESTATAL
Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

detall, estructura el pensament d'una forma computacional.

Una altra característica del pensament computacional és la **descomposició**: en enfrontar-se a un problema, aquest ha de desarticular-se per a convertir-lo en una pràctica més senzilla.

Si el problema original és massa complex per solucionar-lo de cop, cal descompondre'l en diferents problemes menors cada vegada més específics i concrets fins que siguin solucionables.

Aquesta forma de plantejar subproblemes cada vegada més concrets a partir d'un problema general es diu disseny descendent.



Una vegada desarticulat el problema principal, cadascun d'aquests s'ha de resoldre sobre la base d'una metodologia similar que s'hagi utilitzat amb altres problemes ja resolts.

Aquesta característica és el **reconeixement de patrons**: saber generalitzar un procés de resolució amb la finalitat que aquest serveixi per a poder resoldre altres problemes similars.

Quan pensem en els problemes, podem reconèixer similituds entre ells i que es poden resoldre de manera similar. A això es denomina coincidència de patrons, i és una cosa que fem naturalment tot el temps en la nostra vida diària.

Pensament computacional i programació no són sinònims, però comparteixen processos similars: tots dos són un mitjà que serveix per a descompondre i resoldre problemes. Mentre que el pensament computacional és aplicable a moltes disciplines, la programació limita aquests processos exclusivament en l'àmbit de la informàtica.

Generalitat de Catalunya
MINISTERIO DE EDUCACIÓN, FORMACIÓN PROFESIONAL Y DEPORTES
MINISTERIO DE TRABAJO Y ECONOMÍA SOCIAL
SERVICIO PÚBLICO DE EMPLEO ESTATAL SEPE SERVEI PÚBLIC D'OCUPACIÓ ESTATAL
Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

JAVASCRIPT

JavaScript és un llenguatge de programació utilitzat per a crear petits programes encarregats de realitzar accions dins de l'àmbit d'una pàgina web. Amb JavaScript podem crear efectes especials en les pàgines i definir interactivitats amb l'usuari. El navegador de el client és l'encarregat d'interpretar les instruccions JavaScript i executar-les per realitzar aquests efectes i interactivitats, de manera que el major recurs, i potser l'únic, amb què compta aquest llenguatge és el propi navegador.

JavaScript és un llenguatge interpretat que s'introdueix en una pàgina web HTML. Un llenguatge interpretat vol dir que a les instruccions les analitza i processa el navegador en el moment que han de ser executades.

Entre les accions típiques que es poden realitzar en JavaScript tenim dos vessants. D'una banda els efectes especials sobre pàgines web, per crear continguts dinàmics i elements de la pàgina que tinguin moviment, canvien de color o qualsevol altre dinamisme. De l'altra, JavaScript ens permet executar instruccions com a resposta a les accions de l'usuari, de manera que podem crear pàgines interactives amb programes com calculadores, agendes, o taules de càlcul.

JavaScript és un llenguatge amb moltes possibilitats, permet la programació de petits scripts, però també de programes més grans, orientats a objectes, amb funcions, estructures de dades complexes, etc. Tota aquesta potència de JavaScript es posa a disposició del programador, que es converteix en el veritable propietari i controlador de cada cosa que passa a la pàgina. Tot el que veurem a continuació ens servirà de base per endinsar-nos més endavant en el desenvolupament de pàgines enriquides de la banda de el client.

JavaScript, a l'igual que ActionScript en Flash o Visual Basic Script, és una de les múltiples maneres que han sorgit per estendre les capacitats del llenguatge HTML (llenguatge per al disseny de pàgines d'Internet). A l'ésser la més senzilla, és de moment la més estesa. JavaScript no és un llenguatge de programació pròpiament dit com C, C ++, Delphi, etc. És un llenguatge script o orientat a document, com poden ser els llenguatges de macros que tenen molts processadors de text i fulls de càlcul. No es pot desenvolupar un programa amb JavaScript que s'executi fora d'un navegador, tot i que en aquest moment comença a expandir-se a altres àrees com la programació en el servidor amb **NODE.JS**.

UNA MICA D'HISTÒRIA SOBRE JAVASCRIPT

Segons va creixent la web i els seus diferents usos es van anar complicant les pàgines i les accions que es volien realitzar a través d'elles. Al poc temps va quedar reflectit que HTML no era suficient per realitzar totes les accions que es poden arribar a necessitar en una pàgina web. En altres paraules, HTML s'havia quedat curt ja que només servia per presentar el text en una pàgina, definir el seu estil i poc més.

Al complicar els llocs web, una de les primeres necessitats va ser que les pàgines responguessin a algunes accions de l'usuari, per desenvolupar petites funcionalitats més enllà dels propis enllaços. El primer ajudant per cobrir les necessitats que estaven sorgint va ser Java, que és un llenguatge de propòsit general, però que havia creat una manera d'incrustar programes en pàgines web. A través de la tecnologia del Applets, es podia crear petits programes que s'executaven en el navegador dins de les pròpies pàgines web, però que tenien possibilitats similars als programes de propòsit general. La programació d'Applets va ser un gran avanç i Netscape, aleshores el navegador més popular, havia trencat la primera barrera de l'HTML a la fer possible la programació dins de les pàgines web. No hi ha dubte que l'aparició dels Applets va suposar un gran avanç en la

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

història de l'web, però no ha estat una tecnologia definitiva i moltes altres han seguit implementant el camí que va començar amb ells.

Netscape, després de fer els seus navegadors compatibles amb els applets, va començar a desenvolupar un llenguatge de programació a què va anomenar LiveScript que var permetre crear petits programes en les pàgines i que fos molt més senzill d'utilitzar que Java. De manera que el primer JavaScript es diu LiveScript, però no va durar molt aquest nom, ja que abans de llançar la primera versió del producte es va forjar una aliança amb Sun Microsystems, creador de Java, per desenvolupar en conjunt aquest nou llenguatge.

L'aliança va fer que el JavaScript es dissenyés com un germà petit de Java, només útil dins de les pàgines web i molt més fàcil d'utilitzar, de manera que qualsevol persona, sense coneixements de programació, pogués endinsar-se en el llenguatge i utilitzar-ho al seu aire. A més, per programar JavaScript només cal un kit de desenvolupament, ni compilar els scripts, ni realitzar-los en fitxers externs a el codi HTML, com passava amb els applets.

Netscape 2.0 va ser el primer navegador que entenia JavaScript i la seva iniciativa va ser seguida per altres clients web com Internet Explorer a partir de la versió 3.0. No obstant això, la companyia Microsoft va nomenar a aquest llenguatge com JScript i tenia lleugeres diferències respecte a JavaScript, algunes de les quals perduren fins al dia d'avui.

Diferències entre diferents navegadors

Com hem dit el JavaScript de Netscape i el de Microsoft Internet Explorer tenia lleugeres diferències, però és que també el mateix llenguatge va evolucionar a mesura que els navegadors presentaven les seves diferents versions a mesura que les pàgines web es feien més dinàmiques i més exigents les necessitats de funcionalitats.

Les diferències de funcionament de JavaScript ha marcat la història del llenguatge i la manera en què els desenvolupadors es relacionen amb ell, a causa que estaven obligats a crear codi que funcionés correctament en diferents plataformes i diferents versions d'aquestes. Avui dia, continuen havent-hi moltes diferències i per solucionar-ho han sorgit molts productes com els Frameworks JavaScript, que ajuden a realitzar funcionalitats avançades de DHTML sense haver de preocupar en fer versions diferents dels scripts, per a cada un dels navegadors possibles de mercat.

Diferències entre Java i JavaScript

Realment JavaScript es va cridar així perquè Netscape, que estava aliat als creadors de Java a l'època, va voler aprofitar el coneixement i la percepció que les persones tenien del popular llenguatge. Amb tot, es va crear un producte que tenia certes similituds, com la sintaxi del llenguatge o el nom. Es va fer entendre que era un germà petit i orientat específicament per fer coses en les pàgines web, però també es va fer caure a moltes persones en l'error de pensar que són el mateix. Volem que quedi clar que el JavaScript no té res a veure amb Java, excepte en els seus orígens, com s'ha pogut llegir fa unes línies. Actualment, són productes totalment diferents i no guarden entre si més relació que la sintaxi idèntica i poc més. Algunes diferències entre aquests dos llenguatges són les següents: el Compilador. Per programar en Java necessitem un Kit de desenvolupament i un compilador. No obstant això, JavaScript no és un llenguatge que necessiti que els seus programes es compilen, sinó que aquests s'interpreten per part del navegador quan aquest llegeix la pàgina.

- **Orientat a objectes**: Java és un llenguatge de programació orientat a objectes. (Més tard veurem que vol dir orientat a objectes, per a qui no ho sàpiga encara). JavaScript s'ha actualitzat perquè també

sigui orientat a objectes, però podem programar sense necessitat de crear classes, tal com es realitza en els llenguatges de programació estructurada com C o Pascal.

- **Propòsit**: Java és molt més potent que JavaScript, això és degut al fet que Java és un llenguatge de propòsit general, amb el qual es poden fer aplicacions d'allò més variat, però, amb JavaScript només podem escriure programes perquè s'executin en pàgines web.
- **Estructures fortes**: Java és un llenguatge de programació fortament tipat, això vol dir que al declarar una variable haurem d'indicar el seu tipus i no podrà canviar d'un tipus a un altre automàticament. Per la seva banda JavaScript no té aquesta característica, i podem ficar en una variable la informació que desitgem, sense importar el tipus d'aquesta. A més, podrem canviar el tipus d'informació d'una variable quan vulguem.
- **Altres característiques**: Java és molt més complex, tot i que també més potent, robust i segur. Té més funcionalitats que JavaScript i les diferències que els separen són prou importants com per distingir-los fàcilment.

Què necessites per treballar amb JavaScript

Per programar en JavaScript necessitem bàsicament el mateix que per desenvolupar pàgines web amb HTML: un **entorn integrat de desenvolupament** o **IDE** (acrònim en anglès de *integrated development environment*) o editor de text i un navegador compatible amb JavaScript.

Diferents versions de JavaScript, els navegadors que les accepten i els seus avenços.

El llenguatge ha anat avançant durant els seus anys de vida i incrementant les seves capacitats. Al principi podia fer moltes coses a la pàgina web, però tenia poques instruccions per crear efectes especials.

Amb el temps també l'HTML ha avançat i s'han creat noves característiques com les capes, que permeten tractar i maquetar els documents de manera diferent. JavaScript ha avançat també i per gestionar totes aquestes noves característiques s'han creat noves instruccions i recursos.

Realment qualsevol navegador mitjanament modern tindrà ara totes les funcionalitats de JavaScript que necessitarem. No obstant això, pot anar bé conèixer les primeres versions de JavaScript que comentem, a manera de curiositat.

- **JavaScript1**: va néixer amb el Netscape 2.0 i suportava gran quantitat d'instruccions i funcions, gairebé totes les que existeixen ara ja es van introduir en el primer estàndard.
- **JavaScript1.1**: és la versió de JavaScript que es va dissenyar amb l'arribada dels navegadors 3.0. Implementava poc més que la seva anterior versió, com ara el tractament d'imatges dinàmicament i la creació d'arrays.
- **JavaScript1.2**: la versió dels navegadors 4.0. Aquesta té com a desavantatge que és una mica diferent en plataformes Microsoft i Netscape, ja que tots dos navegadors van créixer de manera diferent i estaven en plena lluita pel mercat.
- **JavaScript1.3**: versió que implementen els navegadors 5.0. En aquesta versió s'han llimat algunes diferències entre els dos navegadors.
- **JavaScript 1.5**: Versió que implementa Netscape 6.
- Per la seva banda, Microsoft també ha evolucionat fins a presentar la seva versió 5.5 de JScript (així anomenem el JavaScript utilitzat pels navegadors de Microsoft).

- **ECMAScript**: El 1997 els autors van proposar Javascript com a estàndard de l'European Computer Manufacturers Association ECMA, que tot i el seu nom no és europeu, sinó internacional, amb la seu a Ginebra.
- Per evitar aquestes incompatibilitats, el World Wide Web Consortium (W3C) va dissenyar l'estàndard Document Object Model (DOM, o Model d'Objectes del Document en català), que incorpora el Konqueror, les versions 6 d'Internet Explorer i Netscape Navigator, Opera versió 7, i Mozilla des de la seva primera versió.

SINTAXIS BÀSICA

JavaScript és un llenguatge de programació i, tal i com hem comentat abans, està format per un conjunt de símbols, regles sintàctiques i semàntiques que en defineixen l'estructura i el significat dels elements i expressions.

Per fer-nos una idea, quan redactem un contingut, la frase o oració és el conjunt de paraules amb sentit complet, i normalment les acabem amb un signe de puntuació. A aquesta estructura li diem sentència en programació.

A la seva vegada, les oracions es componen d'unitats gramaticals o sintagmes: subjecte, predicat, complement directe, complement indirecte,... Dins d'una sentència, a aquestes unitats li direm expressions.

En les frases, podem tenir sustantius per fer referència a algun concepte. En la programació, tenim les variables que emmagatzemen informació.

En les oracions tenim verbs per expressar accions. En la programació, tenim comandaments per donar les ordres.

En les frases tenim conjuncions per poder unir mots i sintagmes. En la programació tenim els operadors.

ORACIONS.....SENTENCIES

SINTAGMES..... EXPRESSIONS

SUBSTANTIUS .....VARIABLES

CONJUNCIONS ..... OPERADORS

VERBS..... COMANDOS

Cada vegada que escrivim una instrucció cal acabar amb el caràcter punt i coma. És importantíssim tenir en compte que JavaScript és sensible a majúscules i minúscules.

Encara que ens deixem el punt i coma ";" al final d'una sentència, JavaScript si detecta un salt de línia al codi la finalitza implícitament, encara que és una bona pràctica finalitzar-les explícitament per millorar el rendiment del navegador.

COMENTARIS

Una part fonamental de la programació és afegir comentaris que ens ajudin a compendre cada part del codi sense que el navegador els interpreti. Per afegir comentaris tenim l'opció d'afegir un comentari d'una sola línia **//** i quan canviem de línia continuem amb l'execució normal, o l'opció d'obrir un comentari **/*** i no tancar-lo ***/** fins més endavant:

Generalitat de Catalunya
MINISTERIO DE EDUCACIÓN, FORMACIÓN PROFESIONAL Y DEPORTES
MINISTERIO DE TRABAJO Y ECONOMÍA SOCIAL
SERVICIO PÚBLICO DE EMPLEO ESTATAL SEPE SERVEI PÚBLIC D'OCUPACIÓ ESTATAL
Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

```
// comentari d'una línia

/*
      Comentari
      en vàries
      línies de codi
*/
```

FORMES D'EXECUTAR SCRIPTS DE JAVASCRIPT

Hi ha dues maneres bàsiques d'executar scripts JavaScript en una pàgina: al carregar la pàgina o com a resposta a accions de l'usuari.

EXECUCIÓ DIRECTA

És el mètode d'executar scripts més bàsic. En aquest cas s'inclouen les instruccions dins de l'etiqueta **<script>**. Quan el navegador llegeix la pàgina i troba un script interpreta les línies de codi i les va executant una després d'una altra. Anomenem a aquesta manera execució directa ja que quan es llegeix la pàgina s'executen directament els scripts.

```
<!DOCTYPE html>
<html lang="ca">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Exemple JS</title>
    <style>
        body {font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;}
    </style>
</head>
<body>
    <h1>Exemple JS</h1>
    <script>
        document.write("Hola món!");
    </script>
</body>
</html>
```

En un mateix document .html podem tenir tantes etiquetes `<script>` com facin falta en qualsevol part del document, encara que el més habitual és que estiguin al `<head>` del document.

RESPOSTA A UN ESDEVENIMENT

És l'altra manera d'executar scripts, però abans de veure-la hem de definir els esdeveniments: són les accions que realitza l'usuari. Els programes com JavaScript estan preparats per atrapar determinades accions realitzades, en aquest cas sobre la pàgina, i realitzar accions com a resposta. D'aquesta manera es poden realitzar programes interactius, ja que controlem els moviments de l'usuari i responem a ells. Hi ha molts tipus d'esdeveniments diferents, per exemple la pulsació d'un botó, el moviment del ratolí o la selecció de text de la pàgina.

Les accions que volem fer com a resposta a un esdeveniment s'han d'indicar dins el mateix codi HTML, però en aquest cas s'indiquen en atributs HTML que es col·loquen dins de l'etiqueta que volem que respongui a les accions de l'usuari.

```
<button type="button" onclick="alert("Hola món!");">Saluda</button>
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

INCLOURE FITXERS EXTERNS DE JAVASCRIPT

Una altra manera d'incloure scripts en pàgines web, implementada a partir de JavaScript 1.1, és incloure arxius externs on es poden col·locar moltes funcions que s'utilitzin a la pàgina. Els fitxers solen tenir extensió **.js** i s'inclouen d'aquesta manera:

```
<script src="arxiu_extern.js" defer>
// estic incloent el fitxer " arxiu_extern.js "
</script>
```

Aquesta és la forma més habitual i adequada per mantenir més organitzat el codi en els nostres projectes web.

En cas de vincular amb un arxiu .js, a part de l'atribut src podem afegir altres atributs per definir quan s'executarà aquest codi:

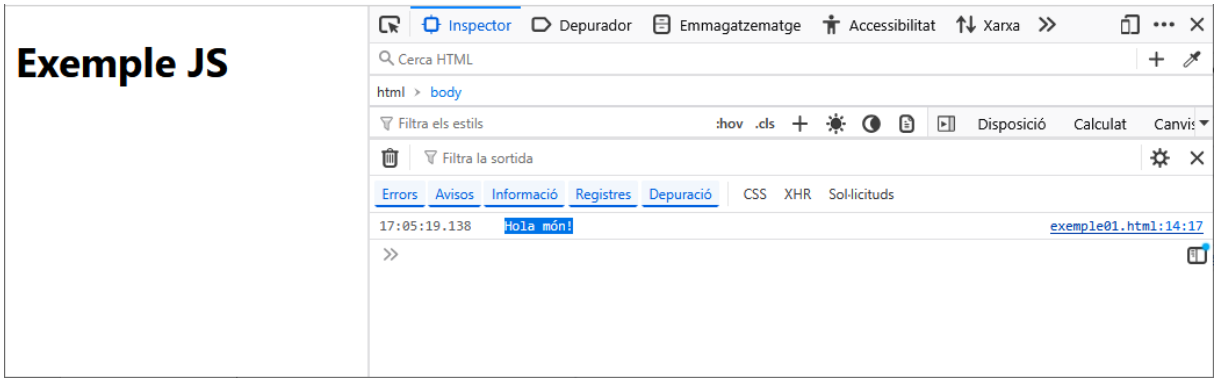
- **src**: especifica l'URL d'un fitxer d'script extern.
- **async**: especifica que l'script es descarrega en paral·lel a la pàgina, i s'executa tan aviat com està disponible (abans que acabi la carrega de la pàgina).
- **defer**: especifica que l'script es descarrega en paral·lel a la pàgina, i s'executa després que la pàgina hagi acabat de carregar-se.

És important aclarir que si utilitzem una etiqueta `<script>` per enllaçar a un altre arxiu .js, no podem utilitzar aquesta mateixa etiqueta per afegir codi, perquè serà sobreescrit pel document enllaçat per l'atribut src.

DEPURACIÓ DEL CODI

Una forma habitual i ràpida per depurar o mostrar resultats del codi és emprar l'objecte **console** amb els diferents mètodes, el més emprat el **.log()**. Amb l'expressió **console.log(**"*missatge*"**)** podem llençar missatges en la consola de depuració del propi navegador:

```
<script>
        console.log("Hola món!");
</script>
```

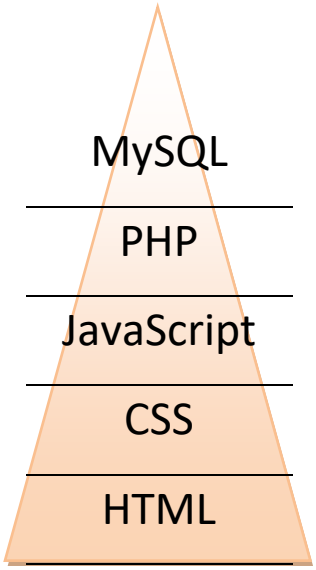


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

L'objectiu d'aquest manual no és mostrar totes les opcions possibles perquè és un llenguatge viu que canvia contínuament, per tant, recomano consultar blocs, fòrums i altres web de referència com per exemple http://www.w3schools.com/ o https://developer.mozilla.org/

De forma ràpida, podem llistar els llenguatges web com una piràmide, on cadascun d'ells complementa l'anterior:

- En la base tenim el **HTML**, que ens permet estructurar els continguts per la seva representació.
- El **CSS** complementa al HTML per donar-li estil i disseny, una millor aparença a l'estructura anterior.
- El **JavaScript** dóna dinamisme i efectes especials al HTML i al CSS. La suma dels tres llenguatges es denomina DHTML, i la seva principal característica és que només cal un navegador per veure els resultats.
- El **PHP** és un llenguatge de servidor, per tant, permet augmentar la seguretat i utilitzar recursos de forma independent al navegador de l'usuari.
- El **MySQL** és una tecnologia de base de dades, que li permet al PHP emmagatzemar dades i continguts per recuperar-los posteriorment; és la base per desenvolupar eines web com els gestors de continguts.



Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

VARIABLES

Un dels fonaments de la programació és conèixer l'estat del sistema en tot moment i ho aconseguim acumulant informació en les variables. Podem imaginar-nos les variables com calaixos dins la memòria RAM del dispositiu, uns calaixos que creem segons els necessitem però que s'eliminaran automàticament quan canviem de document. Podem crear tants calaixos -variables- com necessitem, però per poder organitzar-nos i poder trobar aquesta informació cal etiquetar aquests calaixos, és a dir, cal nombrar les variables.

Les variables són contenidors d'informació per poder recuperar-la o actualitzar-la en qualsevol moment a partir del seu nom.

En el moment de crear una variable cal donar-li un nom. Aquest nom és completament arbitrari, però cal seguir unes normes bàsiques per assegurar-nos el bon funcionament:

- No podem utilitzar paraules ja reservades pel propi llenguatge JavaScript: les nombrades en aquest manual.
- No podem emprar espais ni signes de puntuació: els noms s'han d'escriure tots seguits, sense accents ni altres caràcters especials, emprant l'alfabet anglès; només podem utilitzar el guió baix "_" seguint la pràctica *snake_case* o el símbol de "$".
- Poden contenir números, però mai com primer caràcter: el primer caràcter sempre serà una lletra.
- Com a bona pràctica, es recomana que tinguin noms autodefinitoris: això implica que els noms siguin compostos per diferents paraules.
- Com a bona pràctica, es recomana emprar la pràctica del *lowerCamelCase*: escriure frases o paraules compostes eliminant els espais i posant en majúscula la primera lletra de cada paraula, excepte la primera paraula que es manté en minúscula.

Tipus d'informació	Nom de variable incorrecte	Nom de variable correcte
Nom del client	N-C	nomClient
1er nombre	1 nombre	nombre1
Número 2	Número2	num_2

DECLARACIÓ I INSTANCIACIÓ

Al procés de crear una variable assignant-li un nom únic i irrepetible se li diu **declaració**. És important tenir en compte que JavaScript permet sobreescriure les variables si al declarar una variable li assignem un nom ja existent, provocant errors posteriors en el tractament en la informació.

La **instanciació** és el procés d'assignar un valor a la variable, i es fa amb l'operador "=". Si intentem fer una instanciació sense declarar prèviament la variable, JavaScript farà una declaració implícita i ens deixarà continuar sense errors, però és una mala pràctica perquè es genera codi molt confús.

Per declarar una variable utilitzem un dels comandos reservats pel propi llenguatge:

Tipus de comando	Exemple de declaració i instanciació	Quan l'utilitzarem
var	var nomClient = "Maria";	És el mètode clàssic i l'utilitzarem quan volem una variable d'àmbit global.
let	let comptador = 1;	És el mètode més modern i l'utilitzarem quan volem una variable d'àmbit local.

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

const	const nomClient = "Jordi";	És el mètode més modern per definir constants: variables que una vegada instanciades ja no es tornaran a canviar.

Quan utilitzem **var** o **let** podem fer la declaració en una sentencia i la instanciació en una altre sentencia, però quan emprem **const** cal fer la declaració i la instanciació en la mateixa sentència.

Podem declarar múltiples variables simultàniament amb un únic comando **var** o **let** si separem els diferents noms amb comes ",".

```
<script>
    // Declaració de variables
    var nomClient, cognomClient;

    // Instanciació de variables
    nomClient = "Martí";
    cogomClient = "Garcia";

    // Declaració i instanciació de constant
    const idioma = "ca";
</script>
```

ÀMBIT DE LES VARIABLES –SCOPE-

En la definició de **var** i **let** hem comentat que la principal diferència és l'àmbit del seu ús –scope-: se l'anomena àmbit de les variables al lloc on aquestes estan disponibles. En general, quan declarem una variable fem que estigui disponible al lloc on s'ha declarat. Això passa en tots els llenguatges de programació i, com JavaScript es defineix dins d'una pàgina web, les variables que declarem a la pàgina estaran accessibles dins d'ella.

A JavaScript no podrem accedir a variables que hagin estat definides en una altra pàgina. Per tant, la pròpia pàgina on es defineix és l'àmbit més habitual d'una variable i l'anomenarem a aquest tipus de variables globals a la pàgina. Veurem també es poden fer variables amb àmbits diferents del global, és a dir, variables que declararem i tindran validesa en llocs més acotats.

VARIABLES GLOBALS

Com hem dit, les variables globals són les que estan declarades en l'àmbit més ampli possible, que en JavaScript és una pàgina web. Per declarar una variable global a la pàgina simplement ho farem amb la paraula var.

```
<script>
    var variableGlobal;
</script>
```

Les variables globals són accessibles des de qualsevol lloc de la pàgina, és a dir, des de l'script on s'han declarat i tots els altres scripts de la pàgina, inclosos els manejadors d'esdeveniments, com l'onclick, que ja vam veure que es podia incloure dins de determinades etiquetes HTML.

VARIABLES LOCALS

També podrem declarar variables en llocs més acotats, com per exemple una funció: a aquestes variables les anomenarem locals. Quan es declarin variables locals només hi podrem accedir dins del lloc on s'ha declarat, és a dir, si l'havíem declarat en una funció només hi podrem accedir quan estem en aquesta funció.

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

Les variables poden ser locals a una funció, però també poden ser locals a altres àmbits, com per exemple un bucle. En general, són àmbits locals qualsevol lloc acotat per claus. Aquests elements es tracten més endavant en aquest manual.

TIPUS DE VARIABLES

JavaScript no és un llenguatge fortament tipat, i això vol dir que les variables no tenen un tipus definit en la seva declaració, sinó que es defineixen segons el tipus de valor que li assignem en la instanciació. Així, segons el valor que instanciem a una variable, tindrem els següents tipus:

Tipus de variable	Exemple	Explicació
string	`var nomClient = "Maria";`	Cadena de text. Els valors van entre cometes dobles " " o senzilles ' '.
number	`var registre = 1.5;`	Números. Els valors només poden ser numèrics; el separador decimal és el punt ".".
boolean	`var estat = true;`	Els valors booleans són els valors reservats **true** o **false**.
undefined	`var edat;`	És el valor predeterminat de les variables declarades però ni instanciades.
null	`var cognomClient = null;`	És un tipus de valor que és "sense valor"; es pot emprar per deixar una variable buida o buidar de valor una variable ja instanciada prèviament.
function	`var laMevaFuncio = function(){};`	Una de les formes de declarar funcions és com una variable. Les funcions s'expliquen més endavant.
object	`var persona = {}`	Els objectes són la forma d'emmagatzemar informació de forma més estructurada i complexa. Els objectes s'expliquen més endavant.

Definir correctament el tipus de les variables ens permet emprar els operadors de la forma que més ens interessi, per tant cal controlar el tipus de les variables.

Cal insistir que el tipus d'una variable s'estableix en la instanciació, per tant, si una variable es torna a instanciar amb un tipus de valor diferent, la variable canvia també de tipus. Per sapiguer en cada moment el tipus d'una variable podem emprar l'operador **typeof**:

```
<script>
        var laMevaVariable;
        console.log(laMevaVariable, typeof laMevaVariable); // undefined undefined

        laMevaVariable = "Jordi";
        console.log(laMevaVariable, typeof laMevaVariable); // Jordi string

        laMevaVariable = 2026;
        console.log(laMevaVariable, typeof laMevaVariable); // 2026 number

        laMevaVariable = true;
        console.log(laMevaVariable, typeof laMevaVariable); // true boolean

        laMevaVariable = null;
        console.log(laMevaVariable, typeof laMevaVariable); // null object

        laMevaVariable = {};
        console.log(laMevaVariable, typeof laMevaVariable); // Object {} object
    </script>
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

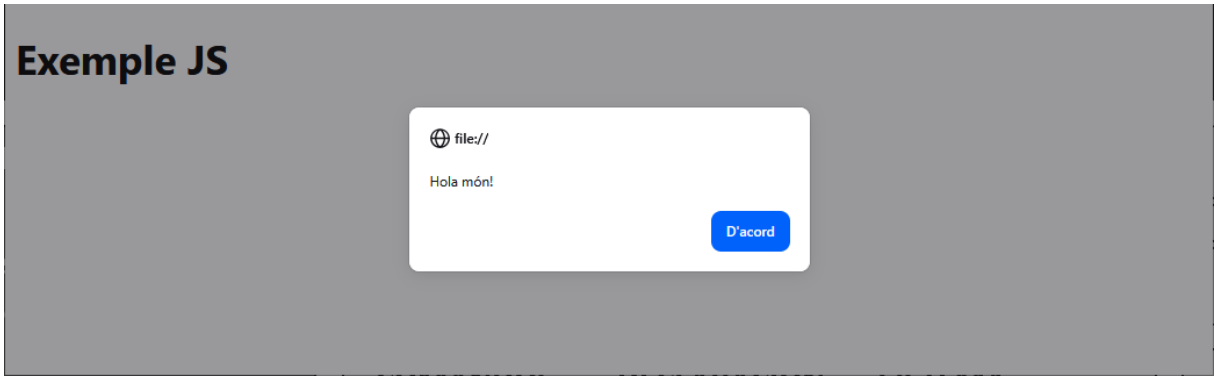
COMANDOS DE SORTIDA I ENTRADA DE VALORS

Una vegada tenim valors emmagatzemats en les variables, només cal nombrar el seu nom per fer referència al seu contingut. Però si volem veure el seu valor en pantalla, cal utilitzar algun dels comandos del llenguatge. Ja hem vist `console.log()`, però tenim altres per poder mostrar directament en pantalla:

ALERT

Amb el comando **`alert()`** podem mostrar missatges curts en una petita finestra del propi navegador; només cal posar el missatge tipus text o el nom de la variable entre els parèntesis:

```
<script>
    const missatge = "Hola món!";
    alert(missatge);
</script>
```

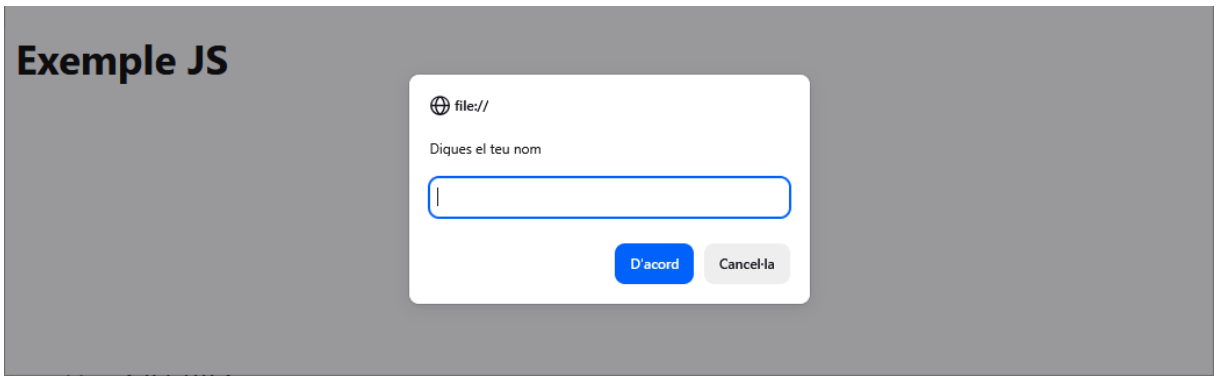


PROMPT

Amb el comando **`prompt()`** podem demanar informació en una petita finestra del propi navegador perquè el visitant ompli amb la seva informació; només cal posar la nostra pregunta tipus text i utilitzar aquest prompt per instanciar una variable:

```
<script>
    const nomVisitant = prompt("Digues el teu nom");
    alert(nomVisitant);
</script>
```

El valor que retorna el prompt és sempre un tipus string.



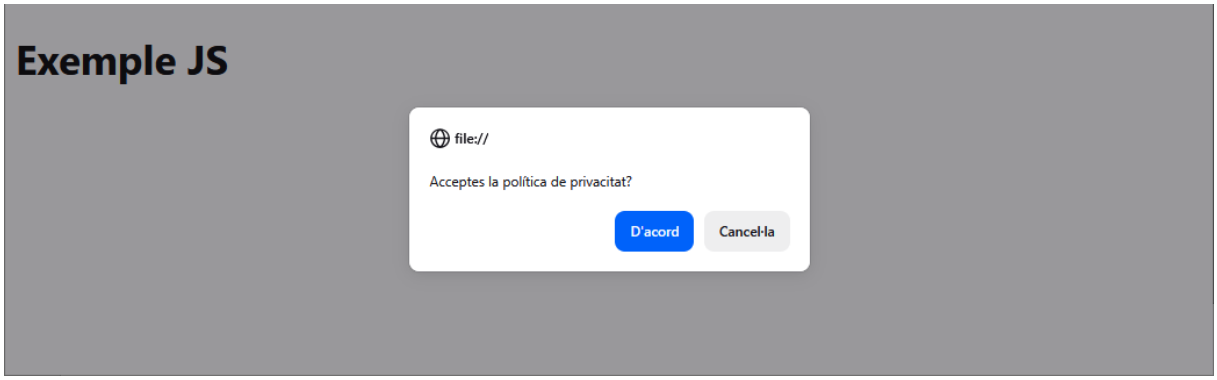
Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

CONFIRM

Amb el comando **`confirm()`** podem mostar informació en una petita finestra del propi navegador perquè el visitant l'accepti (si pitja el botó *D'acord*) o la rebutgi (si pitja el botó *Cancel·la*); només cal posar la nostra pregunta tipus text i utilitzar aquest confirm per instanciar una variable:

```
<script>
    const acceptacio = confirm("Acceptes la política de privacitat?");
    alert(acceptacio);
</script>
```

El valor que retorna el confirm és sempre un tipus Boolean (true o false).



CONVERSIÓ DEL TIPUS DE VALORS

Si volem convertir un valor tipus a un altre, podem emprar funcions generals del JavaScript que necessiten que li possem el valor original entre parèntesis i ens retorna el mateix valor però com un tipus diferent:

Funció	Exemple	Explicació

Boolean()	`var x = "1"; // string`
	`x = Boolean(x);`
	`// Boolean true`
		Retorna un booleà convertit des de qualsevol altre tipus de valor.

Number()	`var x = "4.5"; // string`
	`x = Number(x);`
	`// number 4.5`
		Retorna un número convertit des d'una cadena de text.

parseFloat()	`var x = "4.5"; // string`
	`x = parseFloat(x);`
	`// number 4.5`
		Retorna un número de punt flotant convertit des d'una cadena de text.

parseInt()	`var x = "4.5"; // string`
	`x = parseInt(x);`
	`// number 4`
		Retorna un número enter convertit des d'una cadena de text.

String()	`var x = 6; // number`
	`x = String(x):`
	`// string '6'`
		Converteix el valor d'un objecte en una cadena de text.

```
<script>
    var edat = prompt("Digues la teva edat"); // string
    edat = Number(edat); //number
</script>
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

OPERADORS

Els operadors ens permeten aplicar canvis, càlculs o processos sobre els valors de les variables segons el seu tipus, per tant és molt important controlar el tipus de les variables per assegurar-nos que l'operador s'apliqui correctament. Així, categoritzem els operadors segons el tipus de variable a qui s'aplica.

El primer operador que ja hem vist és el d'assignació per poder fer les instanciacions:

Tipus d'operador	Exemple	Explicació
=	`nomClient = "Maria";`	Permet assignar un valor a una variable en una instanciació.

TEXT

Els operador de text es poden utilitzar en les cadenes de text i variables tipus string:

Tipus d'operador	Exemple	Explicació
"	`nomClient = "Maria";`	Permet definir un valor tipus text (string).
'	`cognomClient = 'Garcia';`	Permet definir un valor tipus text (string).
+	`alert(nomClient + ' ' + cognomClient)`	Permet concatenar dos valors en una mateixa cadena de text.
+=	`nomClient += ' ';` `nomClient += cognomClient;`	Assignació amb concatenació: permet concatenar al mateix valor que ja hi és en una variable
`	`alert(`Benvingut/da ${nomClient} ${cognomClient}`);`	L'accent obert permet fer *templates*: estructures tipus text on podem emprar lliurement les altres cometes i on afegim valors JavaScript amb **${}**
\"	`alert("Benvinguts a \"La Meva Web\"!");`	Cometes literals: amb la barra d'escapament podem afegir " sense que s'interpretin com un operador.
\'	`alert(' Benvinguts a L\'Hospitalet');`	Cometa literal: amb la barra d'escapament podem afegir ' sense que s'interpreti com un operador.
\n	`alert("Benvinguts a:\n \"La Meva Web\"!");`	Salt de línia: amb la barra d'escapament podem afegir un salt de línia en una finestra alert, prompt o confirm.
\t	`alert("Benvinguts a:\n\t \"La Meva Web\"!");`	Tabulació: amb la barra d'escapament podem afegir una tabulació en una finestra alert, prompt o confirm.

NÚMEROS

Els operadors numèrics permeten fer càlculs sobre valors tipus number:

Tipus d'operador	Exemple	Explicació
+	`var num = 7 + 3; // 10`	Suma.
-	`var num = 7 - 3; // 4`	Resta.
*	`var num = 7 * 3; // 21`	Producte.
/	`var num = 7 / 3; // 2,33`	Fracció.
%	`var num = 7 % 3; // 1`	Mòdul: retorna el residu d'una divisió.

Generalitat de Catalunya
MINISTERIO DE EDUCACIÓN, FORMACIÓN PROFESIONAL Y DEPORTES
MINISTERIO DE TRABAJO Y ECONOMÍA SOCIAL
SERVICIO PÚBLICO DE EMPLEO ESTATAL SEPE SERVEI PÚBLIC D'OCUPACIÓ ESTATAL
Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

+=	`var num = 7;` `num += 3; // 10`	Assignació amb suma: suma el número al valor de la variable.
-=	`var num = 7;` `num -= 3; // 4`	Assignació amb resta: resta el número al valor de la variable.
***=**	`var num = 7;` `num *= 3; // 21`	Assignació amb multiplicació: multiplica el número al valor de la variable.
/=	`var num = 7;` `num /= 3; // 2.33`	Assignació amb fracció: fa la fracció del número al valor de la variable.
++	`var num = 7;` `num ++; // 8`	Increment unitari: suma 1 al valor de la variable.
--	`var num = 7;` `num --; // 6`	Decrement unitari: resta 1 al valor de la variable.

```
<script>
    var edat = prompt("Digues la teva edat"); // string
    edat = parseInt(edat); //number
    edat ++;
    alert(`El pròxim aniversari faràs ${edat} anys!`);
</script>
```

LÒGICS I DE COMPARACIÓ

Aquests operadors de comparació retornen sempre un valor boolean de *true* o *false*:

Tipus d'operador	Exemple	Explicació
==	`var comp = 7 == 3; // false`	Igualtat: compara dos valors.
===	`var comp = 7 === '7'; //false`	Identitat: compara dos valors i el seu tipus.
!=	`var comp = 7 != 3; // true`	Diferència.
>	`var comp = 7 > 3; // true`	Major que: compara dos valors.
>=	`var comp = 7 >= 3; // true`	Major o igual que: compara dos valors.
<	`var comp = 7 < 3; // false`	Menor que: compara dos valors.
<=	`var comp = 7 <= 3; // false`	Menor o igual que: compara dos valors.
&&	`var comp = 7 > 3 && 3 > 7;` `// false`	Operador I lògic: concatena dues comparacions.
||	`var comp = 7 > 3 || 3 > 7;` `// true`	Operador O lògic: concatena dues comparacions.
!	`var comp = !true; // false`	Operador de negació: nega el valor.

L'operació lògica **AND** obté el seu resultat combinant dos valors booleans. L'operador s'indica mitjançant el símbol && i el seu resultat solament és *true* si els dos operands són *true*:

variable1	variable2	variable1 && variable2
true	true	**true**
true	false	**false**
false	true	**false**
false	false	**false**

L'operació lògica **OR** també combina dos valors booleans. L'operador s'indica mitjançant el símbol || i el seu resultat és *true* si algun dels dos operands és *true*:

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

variable1	variable2	variable1 || variable2
true	true	**true**
true	false	**true**
false	true	**true**
false	false	**false**

En el següent exemple es planteja dues possibilitats: que el visitant accepti o no la política de privacitat i que l'edat que ha informat sigui superior o igual a 18. Si es compleixen totes dues possibilitats (totes dues tenen un valor *true*) tindrem *true* en la variable *controlAcces*; si qualsevol de les variables *acceptacio* o *controlEdat* tenen un valor *false*, tindrem un *false* en la variable *controlAcces*:

```
<script>
    var acceptacio = false, edat = 0, controlEdat = false, controlAcces = false;
    acceptacio = confirm("Acceptes la política de privacitat?");
    edat = parseInt(prompt("Digues la teva edat"));
    controlEdat = edat >= 18
    controlAcces = acceptacio && controlEdat;
    alert(`Pots accedir? ${controlAcces}`);
</script>
```

PRIORITAT DEL OPERADORS

Segons l'ordre de prioritat, els operadors s'executen abans o després:

1. OPERADORS DE CÀLCUL DE 1er ORDRE:
   - + suma
   - - resta
   - * multiplicació
   - / divisió (fracció)
   - % residu d'una divisió
2. OPERADORS CONDICIONALS DE 2on ORDRE:
   - ==, ===
   - >=, <=
   - >, <
   - !=
3. OPERADORS LÒGICS DE 3er ORDRE:
   - &&
   - ||
   - !
4. OPERADORS DE CÀLCUL DE 4rt ORDRE:
   - +=; -=; *=; /=; %= operador matemàtic combinat
5. OPERADORS DE CÀLCUL DE 5è ORDRE:
   - ++ increment unitari
   - -- decrement unitari

Generalitat de Catalunya
MINISTERIO DE EDUCACIÓN, FORMACIÓN PROFESIONAL Y DEPORTES
MINISTERIO DE TRABAJO Y ECONOMÍA SOCIAL
SERVICIO PÚBLICO DE EMPLEO ESTATAL SEPE SERVEI PÚBLIC D'OCUPACIÓ ESTATAL
Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

ESTRUCTURES DE CONTROL

Els programes que es poden realitzar utilitzant solament variables i operadors són una simple successió lineal d'instruccions bàsiques: el navegador llegeix cada sentencia, l'executa una única vegada i salta a la següent fins que s'acaba el document.

No obstant això, no es poden realitzar programes que mostrin un missatge si el valor d'una variable és igual a un valor determinat i no mostrin el missatge en la resta de casos. Tampoc es pot repetir de manera eficient una mateixa instrucció, com per exemple sumar un determinat valor a tots els elements d'un array.

Per a realitzar aquest tipus de programes són necessàries les **estructures de control de flux**, que són instruccions del tipus "*si es compleix aquesta condició, fes-ho; si no es compleix, fes això un altre*". També existeixen instruccions del tipus "*repeteix això mentre es compleixi aquesta condició*".

Les estructures de control de flux permeten controlar el flux d'execució del codi delimitant quina part del codi s'executa o quantes vegades ho fa.

Si s'utilitzen estructures de control de flux, els programes deixen de ser una successió lineal d'instruccions per a convertir-se en programes intel·ligents que poden prendre decisions en funció del valor de les variables.

Les estructures de control de flux es caracteritzen per tancar el codi a avaluar entre claus d'obertura **{** i tancament **}**. Aquestes claus delimiten un àmbit –scope- i per tant podem declarar variables amb **let** i que aquestes variables només siguin vàlides dins de les claus.

Una altra característica és la capacitat de niar una estructures dins d'altres, de forma que podem crear algoritmes complexos amb múltiples respostes.

CONDICIONS

Quan volem programar diferents respostes o opcions en el nostre codi, ho fem segons una condició: si es compleix, donem una resposta, si no, donem un altre resposta. És a dir: com a programadors hem de deixar en el codi totes les opcions que necessitem plantejar, però en l'execució d'aquest codi només es mostrarà l'opció s'escaigui en aquell moment.

Per plantejar les condicions tenim diferents estructures: el **IF** ens permet plantejar una possibilitat i el **ELSE** complementa la contraria; per altra banda, el **SWITCH** ens permet plantejar diferents possibilitats.

IF

L'estructura més utilitzada en JavaScript i en la majoria de llenguatges de programació és l'estructura **if**. S'empra per a prendre decisions en funció d'una condició. La seva definició formal és:

```
<script>
    if ( condició ) {
        ...
    }
</script>
```

Si l'expressió continguda entre parèntesis o **condició** retorna un valor ***true*** (per exemple, una comparació) s'executen totes les instruccions –sentències- que es troben dins de {...}. Si la condició no es compleix (és a dir, si el seu valor és ***false***) no s'executa cap instrucció –sentències- continguda en {...} i el programa continua executant la resta d'instruccions –sentències- del script.

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

```
<script>
    var acceptacio = false, edat = 0, controlEdat = false, controlAcces = false;
    acceptacio = confirm("Acceptes la política de privacitat?");
    edat = parseInt(prompt("Digues la teva edat"));
    controlEdat = edat >= 18
    controlAcces = acceptacio && controlEdat;
    if ( controlAcces === true ) {
        alert(`Benvingut/da!`);
    }
    if ( controlAcces === false ) {
        alert(`No pot accedir`);
    }
</script>
```

L'expressió dins dels parèntesis ha de retornar un un valor booleà de *true*, i en l'exemple anterior la variable *controlAccess* ja és d'aquest tipus, per tant la primer comparació és redundant, i la segona comparació es pot simplificar si emprem l'operador de negació:

```
<script>
    var acceptacio = false, edat = 0, controlEdat = false, controlAcces = false;
    acceptacio = confirm("Acceptes la política de privacitat?");
    edat = parseInt(prompt("Digues la teva edat"));
    controlEdat = edat >= 18
    controlAcces = acceptacio && controlEdat;
    if ( controlAcces ) {
        alert(`Benvingut/da!`);
    }
    if ( !controlAcces ) {
        alert(`No pot accedir`);
    }
</script>
```

En aquest exemple es planteja les dues possibilitats de la variable *controlAcces*: que sigui *true* o *false*, però segons les respostes a les preguntes d'*acceptacio* i *edat* només tindrà un únic valor, per tant només una de les condicions es cumplirà i un dels alert s'executarà.

IF ... ELSE

Normalment, les decisions que s'han de realitzar no són del tipus "si es compleix la condició, fes-ho; si no es compleix, no facis res", sinó solen ser del tipus "si es compleix aquesta condició, fes-ho; si no es compleix, fes això un altre".

Per a aquest segon tipus de decisions, existeix una variant de l'estructura **if** anomenada **if...else**. La seva definició formal és la següent:

```
<script>
    if ( condició ) {
        ...
    } else {
        ...
    }
</script>
```

Si l'expressió continguda entre parèntesis o **condició** retorna un valor ***true*** (per exemple, una comparació) s'executen totes les instruccions –sentències- que es troben dins del primer {...}. Si la condició no es compleix (és a dir, si el seu valor és ***false***) s'executen totes les instruccions –sentències- que es troben dins del segon {...} precedit d'**else**.

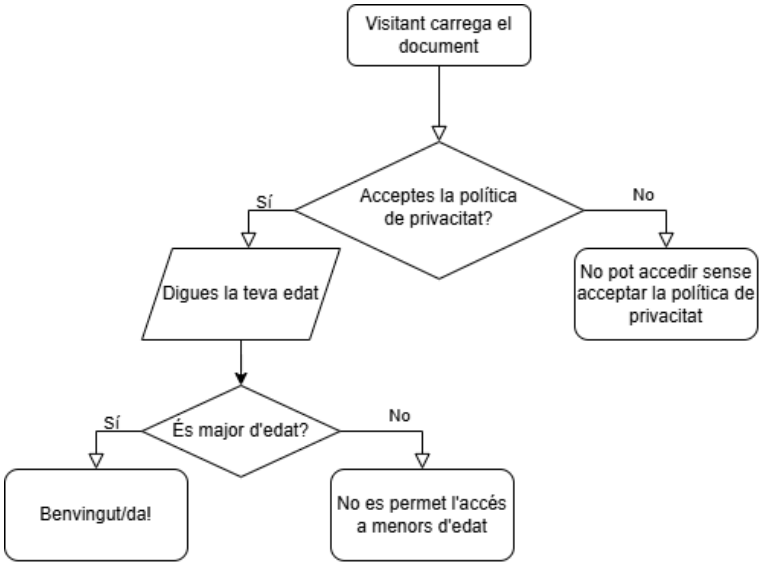
Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

El comando **else** no pot anar sol: sempre acompanyarà a un **if** per definir la situació contrària o ***false*** de la condició.

```
<script>
    var acceptacio = false, edat = 0, controlEdat = false, controlAcces = false;
    acceptacio = confirm("Acceptes la política de privacitat?");
    edat = parseInt(prompt("Digues la teva edat"));
    controlEdat = edat >= 18
    controlAcces = acceptacio && controlEdat;
    if ( controlAcces ) {
        alert(`Benvingut/da!`);
    } else {
        alert(`No pot accedir`);
    }
</script>
```

En aquest exemple, hem substituït el segon **if** i la seva condició per un **else**: en totes dues condicions s'avaluava la mateixa variable, per tant, podem simplificar l'estructura de control de flux condicional a una sola avaluació amb dues possibilitats.

Com es menciona al principi del capítol, les estructures de control es poden niar si volem donar diferents respostes. Per exemple, segons aquest diagrama volem donar diferents respostes i a continuació es planteja el codi amb **if** niats:




```
<script>
    var acceptacio = false, edat = 0;
    acceptacio = confirm("Acceptes la política de privacitat?");
    if ( !acceptacio ) {
        alert("No pot accedir sense acceptar la política de privacitat");
    } else {
        edat = parseInt(prompt("Digues la teva edat"));
        if ( edat < 18 ) {
            alert("No es permet l'accés a menors d'edat");
        } else {
            alert("Benvingut/da!");
        }
    }
</script>
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

ELSE ... IF

Quan necessitem verificar diverses opcions el més fàcil és utilitzar **else if**, així evitem haver de niar en excés. És important recordar que la primera condició vàlida serà la que usi el navegador, i que ignorarà totes les altres.

La seva definició formal és la següent:

```
<script>
    if ( condició ) {
        ...
    } else if ( condició ) {
        ...
    } else if ( condició ) {
        ...
    } else {
        ...
    }
</script>
```

Si la primera condició és *false*, no s'executa i salta a la segona; si aquesta també és *false*, no s'executa i salta a la tercera; i així fins que troba una condició *true* que s'executi o l'últim **else**.

Amb l'exemple anterior, quedaria així el codi amb **else … if**:

```
<script>
    var acceptacio = false, edat = 0;
    acceptacio = confirm("Acceptes la política de privacitat?");
    edat = parseInt(prompt("Digues la teva edat"));
    if ( !acceptacio ) {
        alert("No pot accedir sense acceptar la política de privacitat");
    } else if ( edat < 18 ) {
        alert("No es permet l'accés a menors d'edat");
    } else {
        alert("Benvingut/da!");
    }
</script>
```

TERNARI

Per simplificar l'estructura d'una condició tenim l'operador condicional ternani: no és un operador com a tal sinó una estructura on la seva definició formal és la següent:

```
condicio ? expressioTrue : expressioFalse
```

És molt pràctica en el cas de voler instanciar una variable amb dues possibilitats:

```
<script>
    var edat = parseInt(prompt("Digues la teva edat"));
    var missatge = ( edat >= 18 ) ? "Benvingut/da!" : "No es permet l'accés a menors d'edat";
    alert(missatge);
</script>
```

SWITCH

Fins ara les estructures **if** permeten avaluar una condició que pot ser *true* o *false*; però si la condició pot tenir múltiples valors, l'estructura **switch** permet avaluar diferents possibilitats anomedades casos. Cada cas és un punt d'entrada, però cal definir el final de cada cas amb un **break**. I si es dona la situació de cap cas que coincideixi, podem establir un resultat per defecte amb el **default**:

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

```
<script>
    switch ( expressió ) {
        case valor1:
            sentencies;
            break;
        case valor2:
            sentencies;
            break;
        case valor3:
        case valor4:
            sentencies;
            break;
        default:
            sentencies;
            break;
    }
</script>
```

Com que els casos són només punts d'inici i no de final, podem posar diferents casos seguits perquè tinquin el mateix resultat. L'expressió pot ser una variable tipus text o numèrica, els valors són les diferents possibilitats i a continuació utilitzem els dos punts "**:**" per indicar totes les sentencies que s'han d'executar fins al **break**. Sense el **break**, s'executarien totes les sentencies fins el tancament del **switch**.

```
<script>
    const nomUsuari = prompt("Digues el teu nom:");
    var missatge = "";
    switch ( nomUsuari ) {
        case "Jordi":
            missatge = "Qui no s'arrisca, no pisca.";
            break;
        case "Maria":
            missatge = "Com més serem, més riurem.";
            break;
        case "Pep":
        case "Ona":
            missatge = "De mica en mica s'omple la pica.";
            break;
        default:
            missatge = "Hi ha més dies que llonganisses.";
            break;
    }
    alert( missatge );
</script>
```

isNaN()

Un cas especial es la funció gobal `isNaN()`: és una funció que retorna un booleà segons el valor no és un número –*true*- o sí és un número –*false*-:

```
<script>
    var edat = Number(prompt("Digues la teva edat")), missatge = "";
    if ( isNaN(edat) ) {
        missatge = "Cal introduir una edat vàlida!";
    } else if ( edat >= 18 ) {
        missatge = "Benvingut/da!";
    } else {
        missatge = "No es permet l'accés a menors d'edat.";
    }
    alert(missatge);
</script>
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

BUCLES

El concepte de bucle fa referència a la repetició de sentències tantes vegades com faci falta sense la necessitat de duplicar línies de codi, per tant, trenquem la linealitat de l'execució del navegador.

WHILE

Permet repetir les sentencies niades en les claus mentre la condició sigui *true*. Això implica que necessitem una instanciació inicial, una comparació que mantingui el bucle i una actualització per evitar entrar en un bucle infinit:

```
<script>
    var control = valor; // inicialització
    while ( control == valor ) { // condició
        ...
        control = nouValor; // actualització
    }
</script>
```

Un exemple és demanar un cert tipus de dada, i no deixar continuar a l'usuari fins que la introdueixi correctament:

```
<script>
    var edat = parseInt(prompt("Digues la teva edat")), missatge = "";
    while ( isNaN(edat) ) {
        alert ("Cal introduir una edat vàlida!");
        edat = parseInt(prompt("Digues la teva edat"));
    }
    missatge=(edat>=18)? "Benvingut/da!" : "No es permet l'accés a menors d'edat" ;
    alert(missatge);
</script>
```

DO ... WHILE

Si el que necessitem és fer una acció com a mínim una vegada, i després avaluar si cal repetir-la, podem emprar l'estructura **do ... while** perquè justament fa això: primer executa el contingut de les claus i després avalua si cal continuar:

```
<script>
    do {
        ...
    } while ( condició )
</script>
```

En l'exemple anterior, demanem dues vegades l'edat; per tant, podem simplificar el codi demanat-lo sempre una vegada i després avaluar si cal tornar a demanar-ho:

```
<script>
    var edat = null, missatge = "";
    do {
        missatge=(edat===null)?"Digues la teva edat":"Cal introduir una edat vàlida!";
        edat = parseInt( prompt(missatge) );
    } while ( isNaN(edat) )
    missatge=(edat>=18)? "Benvingut/da!" : "No es permet l'accés a menors d'edat";
    alert(missatge);
</script>
```

Generalitat de Catalunya
MINISTERIO DE EDUCACIÓN, FORMACIÓN PROFESIONAL Y DEPORTES
MINISTERIO DE TRABAJO Y ECONOMÍA SOCIAL
SERVICIO PÚBLICO DE EMPLEO ESTATAL SEPE SERVEI PÚBLIC D'OCUPACIÓ ESTATAL
Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

FOR

Aquest bucle presenta una estructura optimitzada per a controlar l'execució de la iteració de manera numèrica, és a dir, per especificar exactament quantes vegades volem que es faci el bucle.

El bucle **for** es divideix en tres parts separades per un punt i coma:

1. **Expressió inicial**: serà tot allò que s'executarà en iniciar-se el bucle. Normalment la declaració d'una variable numèrica instanciada amb el valor inicial.
2. **Condició**: serà avaluada abans de cada iteració. Aquest és l'únic paràmetre obligatori i és una condició de comparació amb la variable inicial que manté el bucle mentre retorna *true*.
3. **Expressió d'actualització**: s'executarà al final de cada iteració. Cal augmentar o disminuir el valor de la variable inicial perquè la condició arribi un moment que torni *false* i evitar entrar en un bucle infinit.

```
<script>
    for ( inicialització; condició; actualització ) {
          ...
    }
    for ( let i = 1; i <= 10; i ++ ) {
        console.log(i);
    }
    for ( let i = 10; i >= 1; i -- ) {
        console.log(i);
    }
</script>
```

Normalment, la variable que controla els bucles **for** es diu **i**, ja que recorda a la paraula índex i el seu nom tan curt estalvia molt temps i espai. Si cal fer bucles niats podem utilitzar els noms de variables **j** o **k**.

La variable inicial es declara dins de l'estructura de control, per tant, té un àmbit –scope- local, per tant, és l'exemple d'ús del **let**.

Aquests mateixos exemples es poden fer amb **while**, però no queda el codi tan ordenat com amb el **for**:

```
<script>
    var i = 1; // inicialització
    while ( i <= 10 ) { // condició
        console.log(i);
        i ++; // actualització
    }
</script>
```

FOR ... IN I FOR ... OF

Un cas especial de bucles són aquestes estructures **for** pensades per recórrer matrius –arrays-, per tant els veurem més endavant.

SENTÈNCIES BREAK I CONTINUE

L'estructura de control **for** és molt senzilla d'utilitzar, però té l'inconvenient que el nombre de repeticions que es realitzen només es poden controlar mitjançant les variables definides en la zona d'actualització del bucle. Les sentències **break** i **continue** permeten manipular el comportament normal dels bucles for per a detenir el bucle o per a saltar-se algunes repeticions. Concretament, la sentència **break** permet acabar de manera abrupta un bucle i la sentència **continue** permet saltar-se algunes repeticions del bucle.

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

FUNCIONS

En programació és molt freqüent que un determinat procediment de càlcul definit per un grup de sentències hagi de repetir-se diverses vegades, ja sigui en un mateix programa o en altres programes, la qual cosa implica que s'hagi d'escriure tants grups d'aquelles sentències com vegades aparegui aquest procés.

L'eina més potent amb què es compta per a facilitar, reduir i dividir el treball en programació, és escriure aquells grups de sentències una sola i única vegada sota la forma d'una **funció**.

Un programa és un desenvolupament complex de realitzar i per tant és important que estigui ben **estructurat** i també que sigui intel·ligible per a les persones. Si un grup de sentències realitza una tasca ben definida, llavors pot estar justificat l'aïllar aquestes sentències formant una funció, encara que resulti que només se la nomeni o faci servir una vegada.

Fins ara hem vist com resoldre un problema plantejant un únic algorisme. Amb funcions podem segmentar un programa en diverses parts. Davant d'un problema, plantegem un algoritme, aquest pot constar de petits algoritmes.

Una funció és un conjunt de sentències encapsulades que pot ser utilitzat des de diferents parts d'un programa tantes vegades com faci falta.

Les funcions de JavaScript són l'ànima d'aquest llenguatge, per això es consideren ciutadans de primera classe, una entitat que suporta totes les operacions generalment disponibles per a altres entitats: aquestes operacions normalment inclouen ser passats com a argument, retornats d'una funció i assignats a una variable.

A partir d'ara veurem l'aplicació dels principis de la programació funcional en altres apartats d'aquest manual.

DECLARACIÓ I INVOCACIÓ

Podem **declarar** les funcions de dues formes: instanciant una variable amb una funció anònima o utilitzant el comando **function** i donant un nom únic seguint les mateixes directrius en donar un nom a una variable. La primera diferència entre una variable i una funció és que en una variable emmagatzema dades, i en una funció emmagatzema sentències; la segona, que en declarar una funció usem els parèntesis "()": en el següent apartat explicarem el seu ús.

```
<script>
    // declaració en una variable amb una funció anónima:
    const laMevaFuncio = function () {
        ...
    }
    // declaració amb nom:
    function laMevaFuncio () {
        ...
    }
</script>
```

Una vegada declarada la funció, tot el codi dins de les seves claus quedarà a l'espera i el navegador no l'executarà fin que la funció sigui **invocada** a través del seu nom:

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

```
<script>
    // declaració amb nom:
    function laMevaFuncio () {
        ...
    }
    // invocació a través del nom:
    laMevaFuncio ();
</script>
```

Dins de les funcions podem declarar variables, però com que el contingut es tanca entre claus "{}", les funcions generen el seu propi àmbit –scope- i podem utilitzar variables globals (declarades fora de les funcions) o variables locals (declarades dins de les funcions amb **let**).

PARÀMETRES I ARGUMENTS

Quan volem fer funcions amb un nivell d'abstracció realment alt, hem de recórrer a l'aïllament. De tal forma que la nostra funció no depengui d'unes certes variables o dades externes a ella.

Quan declarem una funció, podem incloure uns certs **paràmetres** entre els parèntesis que actuaran com a referències. Funcionaran internament igual que variables locals, de tal forma que a l'hora d'executar la funció podrem passar-li uns certs **arguments** –valors- i així tenir funcions amb un major nivell d'abstracció.

En declarar una funció podem afegir tants paràmetres com necessitem separats per comes, però les bones pràctiques de programació ho limiten a tres; si necessites més, cal que desglossis la funció en diverses més simples. Cada paràmetre tindrà un nom propi, com si fos una variable, però al ser local es poden repetir els mateixos noms entre diferents funcions. I en la invocació, passem els arguments o valors de cada paràmetre en el mateix ordre i també separat per comes ",":

```
<script>
    // declaració amb paràmetres
    function laMevaFuncio ( param1, param2, param3) {
        ...
    }
    // invocació amb arguments:
    laMevaFuncio ( arg1, arg2, arg3 );
</script>
```

En el següent exemple, declarem una funció amb un paràmetre que s'utilitza dins de la funció com una variable local, i la invoquem diverses vegades: en cada invocació passem un valor diferent com argument:

```
<script>
    function saluda( nom ) {
        alert("Hola " + nom);
    }
    saluda ( "Ot" );
    saluda ( "Marta" );
</script>
```

FUNCIONS QUE RETORNEN UN VALOR

Un altre dels punts forts a l'hora de plantejar estructures de codi modulars i reutilitzables, és tenir en compte el retorn. El retorn ens permet retornar un valor en acabar d'executar-se la funció. Aquest valor pot ser qualsevol tipus de dada dels molts que tenim en JavaScript. Perquè les funcions siguin modulars i reutilitzables cal que no

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

totes finalitzin un procés, per exemple mostrar un *alert*, sinó que facin un procés i retornin un valor, i aquell qui hagi invocat la funció reculli aquest valor i continuï processant-lo.

Per fer que una funció retorni un valor, només cal utilitzar el comando **return** al final del codi d'aquesta:

```
<script>
    function missatge ( text ) {
        return "Hola " + text;
    }
    function saluda ( nom ) {
        alert ( missatge (nom) )
    }
    saluda ( "Ot" );
    saluda ( "Marta" );
</script>
```

NIAMENT

Dins de les claus de les funcions podem niar altres estructures ja vistes com a condicions i bucles, però també altres funcions que només es podran invocar dins de la funció principal. Això, per una banda, pot complicar el desenvolupament del codi, però per l'altra ens dona moltes més possibilitats de modularització i reutilització.

```
<script>
    var nom = null;
    function alerta ( tipus ) {
        let text = "";
        function controlEdat () {
            let edat = null, missatge = "";
            do {
                missatge=(edat===null)?"Digues la teva edat":"Cal introduir una edat vàlida!";
                edat = parseInt( prompt(missatge) );
            } while ( isNaN(edat) )
            missatge=(edat>=18)?"Benvingut/da!":"No es permet l'accés a menors d'edat";
            return missatge;
        }
        function preguntaNom () {
            if ( nom === null) {
                nom = prompt("Quin és el teu nom?");
            }
            return "Hola " + nom;
        }
        switch ( tipus ) {
            case "edat":
                text = controlEdat();
                break;
            default:
                text = preguntaNom ();
                break;
        }
        return text;
    }
    alert ( alerta ( "edat") );
    alert ( alerta ( "salutació") );
</script>
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

FAT ARROW

Les funcions **fat arrow** s'utilitzen per ometre la paraula **function** i simplificar l'estructura original de les funcions, però aquesta estructura és limitada i no es pot usar en totes les situacions. Aquesta simplificació també permet que el **return** sigui implícit si no s'usen les claus "{}"; fins i tot podem estalviar-nos els parèntesis "()" si tenim un únic paràmetre. Després, la invocació es fa com sempre a través del nom.

Desglossament de la funció fletxa:

Segons la definició d'una funció tradicional tindriem la següent declaració:

```
<script>
    function duplicar (a){
        return a * 2;
    }
</script>
```

Però amb les funcions fletxa ho podem simplificar:

1. Elimina la paraula **function** i col·loca la fletxa entre el paràmetre i la clau d'obertura:

```
<script>
    const duplicar = (a) => {
        return a * 2;
    }
</script>
```

2. Treu les claus del cos i la paraula **return**: el retorn està implicit:

```
<script>
    const duplicar = (a) => a * 2;
</script>
```

3. Suprimeix els parèntesis dels paràmetres si només hi ha un únic paràmetre:

```
<script>
    const duplicar = a => a * 2;
</script>
```

Una de les raons per les quals es van introduir les funcions fletxa va ser per eliminar complexitats de l'àmbit **this** i fer que l'execució de funcions sigui molt més intuïtiva. En les funcions tradicionals, de manera predeterminada, **this** està en l' àmbit del *window* (del document), però a les funcions fletxa no predeterminen **this** a l'àmbit o abast del document: l'executen en l'àmbit o abast en què es creen. Aquest concepte de **this** es desenvolupa més endavant en aquest manual.

FUNCIONS GENERALS

Les funcions generals són funcions ja definides en el JavaScript; en aquest manual ja s'han comentat algunes com les funcions de conversió de tipus de variable o per avaluar si un valor és o no un número, però existeixen algunes més. La més remarcable es la funció **eval()** que permet avaluar una expressió de text aportada com argument com si fos una sentència:

```
<script>
    const text = "2 + 3";
    alert( eval(text) );
</script>
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

ESDEVENIMENTS

En la introducció d'aquest manual s'explica que els esdeveniments són otra forma d'incloure codi JavaScript en el codi HTML com atributs de les etiquetes: el valor de l'atribut és el codi JavaScript:

```
<button type="button" esdeveniment="sentències;">Botó</button>
```

Els esdeveniments són les diferents formes que tenim d'interactuar amb els diferents elements –etiquetes- del codi HTML del document. Una mateixa etiqueta pot tenir associats diferents esdeveniments.

Però dins d'aquestes cometes estem molt limitats per afegir totes les estructures que hem vist fins ara perquè no podem afegir el codi en diferents línies i el codi s'ofusca molt, i tampoc podem emprar cometes dobles. Per tant, els esdeveniments són una forma ideal de combinar amb les funcions: podem declarar-les als scripts i invocar-les dins dels esdeveniments:

```
<script>
    const doblar = a => a * 2;
</script>
<p>
    <button type="button" onclick="alert( doblar(3) );">Doblar 3</button>
    <button type="button" onclick="alert( doblar(7) );">Doblar 7</button>
</p>
```

Nom de l'esdeveniment com atribut	Definició
onblur	Quan un element de formulari perd el focus
onchange	Quan el valor d'un camp de formulari és modificat
onclick	Quan es fa clic amb el botó del ratolí
oncontextmenu	Quan es fa clic amb el botó alternatiu del ratolí
ondblclick	Quan es fa doble clic en un objecte
onfocus	Quan un element de formulari adquireix el focus
oninput	Quan s'està modificant un camp de formulari
onkeydown	Quan es pressiona una tecla
onkeypress	Quan es pressiona una tecla
onkeyup	Quan es deixa de pressionar una tecla
onload	Quan una pàgina o imatge acaba de carregar-se
onmousedown	Quan es pitja el botó del ratolí
onmousemove	Quan es mou el ratolí
onmouseout	Quan el cursor del ratolí surt de l'element
onmouseover	Quan el cursor del ratolí es posa damunt
onmouseup	Quan es deixa anar el botó del ratolí
onreset	Quan es pitja el botó de reset d'un formulari
onresize	Quan es modifica la grandària d'una finestra
onselect	Quan se selecciona text d'un camp de formulari
onsubmit	Quan es pitja el botó submit d'un formulari
onwheel	Quan la roda del ratolí puja o baixa sobre un element

En el següent exemple es veu l'esdeveniment **onsubmit** i **oninput** aplicat a un formulari:

```
<form action="" onsubmit="return confirm('Vols enviar el formulari?');">
    <input type="text" name="formulari" value="Enviat!" oninput="alert('Canvi!');">
    <button type="submit">Enviar</button>
</form>
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

OBJECTES INTEGRATS DEL LLENGUATGE

Són els objectes nadius del llenguatge, definits per l'especificació ECMAScript. Existeixen sempre, independentment d'on s'executi JavaScript. També se'n diuen tipus nadius (*native objects*) o constructors nadius (*native constructors*). Aquests objectes formen part del nucli del llenguatge: no els crees tu, ja hi són quan arrenca el motor de JavaScript.

Com que ja estan creats, funcionen com plantilles que podem instanciar en variables amb la funció constructora **new**. Aquests objectes ens proporcionen formes més complexes per manipular la informació gràcies al fet que tenen **propietats** i **mètodes**.

Podem imaginar els objectes del JavaScript com els objectes físics reals: si agafem un retolador, aquest té propietats –informació- com la longitud, volum o color, però també té mètodes –accions- com destapar, pintar o tapar.

Com que tenim els objectes ja definits, les seves propietats i mètodes també estan ja definides, és a dir, tenen noms ja reservats. Sintàcticament, utilitzarem la funció constructora **new** per crear un objecte nadiu:

```
<script>
    const text = new String("Hola món!");
</script>
```

A partir d'ara, la variable text serà un nou objecte (en aquest cas *string*) i podem aplicar propietats i mètodes amb la sintaxi del punt ".". Els mètodes, com que són accions, són funcions ja predefinides i s'escriuen amb parèntesis "()" perquè poden tenir arguments:

```
<script>
    const text = new String("Hola món!");
    console.log( text.length ); // 9
    console.log( text.toUpperCase() ); // "HOLA MÓN!"
</script>
```

TEXT

L'objecte *String* és el que ens permet manipular els texts, però a nivell d'aplicació de mètodes i propietats, una variable tipus text també els rep:

```
<script>
    const salutacio1 = new String("Hola món!");
    const salutacio2 = "Hola món!";
</script>
```

Propietat	Descripció
length	Retorna la longitud d'una cadena.

Mètode (arguments)	Descripció
at()	Retorna un caràcter indexat d'una cadena.
charAt()	Retorna el caràcter a un índex (posició) especificat.
charCodeAt()	Retorna l'Unicode del caràcter en un índex especificat.
codePointAt()	Retorna el valor Unicode en un índex (posició) d'una cadena.

Generalitat de Catalunya
MINISTERIO DE EDUCACIÓN, FORMACIÓN PROFESIONAL Y DEPORTES
MINISTERIO DE TRABAJO Y ECONOMÍA SOCIAL
SERVICIO PÚBLICO DE EMPLEO ESTATAL SEPE SERVEI PÚBLIC D'OCUPACIÓ ESTATAL
Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

concat()	Retorna dues o més cadenes unides.
endsWith()	Retorna si una cadena acaba amb un valor especificat.
fromCharCode()	Retorna els valors Unicode com a caràcters.
includes()	Retorna si una cadena conté un valor especificat.
indexOf()	Retorna l'índex (posició) de la primera aparició d'un valor en una cadena.
isWellFormed()	Retorns certs si una cadena està ben formada.
lastIndexOf()	Retorna l'índex (posició) de l'última aparició d'un valor en una cadena.
localeCompare()	Compara dues cadenes en la localització actual.
match()	Cerca una cadena per un valor, o una expressió regular, i retorna les coincidències.
matchAll()	Cerca una cadena per un valor, o una expressió regular, i retorna les coincidències.
padEnd()	Posa una cadena al final.
padStart()	Apadeix una cadena des del principi.
prototype	Et permet afegir propietats i mètodes a un objecte.
repeat()	Retorna una nova cadena amb diverses còpies d'una cadena.
replace()	Cerca un patró en una cadena i retorna una cadena on es substitueix la primera coincidència.
replaceAll()	Cerca un patró en una cadena i retorna una nova cadena on es substitueixen totes les coincidències.
search()	Cerca en una cadena un valor, o expressió regular, i retorna l'índex (posició) de la coincidència.
slice()	Extreu una part d'una cadena i retorna una nova cadena.
split()	Divideix una cadena en una matriu de subcadenes.
startsWith()	Comprova si una cadena comença amb caràcters especificats.
substr()	Depreciat. Utilitza substring() o slice() en comptes d'això.
substring()	Extreu caràcters d'una cadena, entre dos índexs (posicions) especificats.
toLocaleLowerCase()	Retorna una cadena convertida a lletres minúscules, utilitzant la localització de l'amfitrió.
toLocaleUpperCase()	Retorna una cadena convertida a majúscules, utilitzant la localització de l'amfitrió.
toLowerCase()	Retorna una cadena convertida a lletres minúscules.
toString()	Retorna una cadena o un objecte cadena com a cadena.
toUpperCase()	Retorna una cadena convertida a lletres majúscules.
toWellFormed()	Retorna una cadena on "substituts solitaris" es substitueixen pel caràcter de substitució Unicode.
trim()	Retorna una cadena amb espais en blanc eliminats.
trimEnd()	Retorna una cadena amb espais en blanc eliminats des del final.
trimStart()	Retorna una cadena amb espais en blanc eliminats des de l'inici.
valueOf()	Retorna el valor primitiu d'una cadena o d'un objecte cadena.

NÚMERO

L'objecte integrat **Math** ens aporta infinitat de recursos matemàtics avançats com la constant de Euler, gestió de logaritmes, sins, cosinus, tangents... Cada lector ha d'indagar i valorar el que realment vol usar, ja que molts

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

d'aquests mètodes i propietats van més enllà dels nostres objectius, i no aporten directament valor al context d'aprendre a programar en JavaScript. Però sí que hi ha alguns mètodes que poden ser útils:

Mètode (arguments)	Descripció
ceil(x)	Retorna x, arrodonit cap amunt a l'enter més proper.
floor(x)	Retorna x, arrodonit cap avall a l'enter més proper.
max(x1,x2,..)	Retorna el nombre amb el valor més alt.
min(x1,x2,..)	Retorna el nombre amb el valor més baix.
random()	Retorna un nombre aleatori entre 0 i 1.
round(x)	Arrodoneix x a l'enter més proper.

```
<script>
    var x = Math.random() * 10;
    x = Math.ceil(x);
    console.log(x);
</script>
```

L'objecte integrat **Number** ens dona accés a mètodes similars a les funcions generals:

Mètode (arguments)	Descripció
isFinite()	Comprova si un valor és un nombre finit.
isInteger()	Comprova si un valor és un enter.
isNaN()	Comprova si un valor és NaN.
parseFloat()	Analitza una cadena i retorna un nombre.
parseInt()	Analitza una cadena i retorna un nombre sencer.
toFixed(x)	Formata un nombre amb x nombres de dígits després del punt decimal.
toLocaleString()	Converteix un nombre en una cadena, segons la configuració local.
toPrecision(x)	Formata un nombre a x longitude.
toString()	Converteix un nombre en una cadena.
valueOf()	Retorna el valor primitiu d'un nombre.

```
<script>
    let x = Math.random() * 10;
    console.log( x.toFixed(0) );
</script>
```

ARRAY

Els arrays són estructures que ens permeten emmagatzemar moltes dades, sense haver de preocupar-nos per l'ordre o l'organització interna: s'organitza automàticament. Una altra forma més senzilla d'entendre-ho, és imaginar que un array és senzillament com una llista de diferents valors tipus text, números, booleans i fins i tot altres arrays niats -anomenats arrays multidimensionals-.

Podem instanciar una variable amb la clàusula **new** o de forma abreviada amb claudàtors "**[]**":

```
<script>
    const noms1 = new Array('Maria','Josep');
    const noms2 = ['Maria','Josep'];
</script>
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

Si volem recuperar algun valor de l'array, només cal utilitzar el nom de l'objecte i entre els claudàtors afegir un número de posició, tenint en compte que els valors d'un array comencen a ordenar-se des del 0:

```
<script>
                //  0        1
    const noms = ['Maria','Josep'];
    console.log( noms[0] ); // Maria
</script>
```

Propietat	Descripció
length	Retorna la quantitat d'elements de l'array

Amb aquesta propietat podem fer un bucle per recórrer tots els elements d'un array:

```
<script>
    const noms = ['Maria','Josep'];
    const longitut = noms.length;
    for (let i = 0; i < longitut; i ++) {
        console.log( noms[i] );
    }
</script>
```

Però en el captítol del bucles es mencionen les estructures **for ... in** i **for ... of** que són estructures més optimitzades per reccorer els valors d'un array:

```
<script>
    const noms = ['Maria','Josep'];
    for (let i in noms) { // la variable "i" emmagatzema les posicions
         console.log( noms[i] );
    }
    for (let nom of noms) { // la variable "nom" emmagatzema el valor
        console.log( nom );
    }
</script>
```

Mètode (arguments)	Descripció
at()	Retorna un element indexat d'un array.
concat()	Uneix arrays i retorna un array amb els arrays units.
copyWithin()	Copia elements de l'array dins de l'array, cap a i des de posicions especificades.
entries()	Retorna un parell clau/valor.
every()	Comprova si cada element d'un array supera una prova.
fill()	Omple els elements d'un array amb un valor estàtic.
filter()	Crea un nou array amb cada element d'un array que supera una prova.
find()	Retorna el valor del primer element d'un array que supera una prova.
findIndex()	Retorna l'índex del primer element d'un array que supera una prova.
findLast()	Retorna el valor de l'últim element d'un array que ha passat una prova.
findLastIndex()	Retorna l'índex de l'últim element d'un array que ha passat una prova.
flat()	Concatena elements de subarray.
flatMap()	Mapeja tots els elements de l'array i crea un nou array pla.
forEach()	Crida una funció per a cada element de l'array.
from()	Crea un array a partir d'un objecte.

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

includes()	Comprova si un array conté l'element especificat.
indexOf()	Cerca un element a l'array i retorna la seva posició.
isArray()	Comprova si un objecte és un array.
join()	Uneix tots els elements d'un array en una cadena.
keys()	Retorna un Objecte d'Iteració de l'Array, que conté les claus de l'array original.
lastIndexOf()	Cerqueu un element a l'array, començant pel final, i retornant la seva posició.
map()	Crea un nou array amb el resultat de cridar una funció per a cada element de l'array.
of()	Crea un array a partir de diversos arguments.
pop()	Elimina l'últim element d'un array i retorna aquest element.
push()	Afegeix nous elements al final d'una matriu i retorna la nova longitude.
reduce()	Redueix els valors d'un array a un sol valor (d'esquerra a dreta).
reduceRight()	Redueix els valors d'un array a un sol valor (de dreta a esquerra).
reverse()	Inverteix l'ordre dels elements d'un array.
shift()	Elimina el primer element d'un array i retorna aquest element.
slice()	Selecciona una part d'un array i retorna el nou array.
some()	Comprova si algun dels elements d'un array supera una prova.
sort()	Ordena els elements d'un array.
splice()	Afegeix o elimina elements de l'array.
toReversed()	Inverteix l'ordre dels elements de l'array (a un nou array).
toSorted()	Ordena els elements d'un array (a un nou array).
toSpliced()	Afegeix o elimina elements de l'array (a un nou array).
toString()	Converteix un array en una cadena i retorna el resultat.
unshift()	Afegeix nous elements a l'inici d'una matriu i retorna la nova longitude.
valueOf()	Retorna el valor primitiu d'un array.
with()	Retorna un nou array amb elements actualitzats.

```
<script>
    const noms = ['Maria','Josep'];
    noms.push('Carles');
    noms.forEach( (nom) => {console.log(nom);} );
</script>
```

DATA

L'objecte integrat **Date** és el que ens permet recuperar informació de la data del sistema i manipular-la. Com ja hem vist, instanciem una variable amb la clàusula **new** i com argument li proporcionem la data que ens interessi:

```
<script>
    var ara = new Date(); // data actual
    var dia2 = new Date(3600*24*1000); // data en mil·lisegons des del 01/01/1970
    var anyNou = new Date("January 1, 2026 00:00:00");//data en text,no recomanable
    var diaAnyNou = new Date("2026,1,1"); // data en números: AAAA, MM, DD
    var iAnyNou = new Date("2026,1,1,0,0,0");//data en números: AAAA,MM,DD,HH,MM,SS
</script>
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

En el cas de les dates, podem dividir gairebé tots els mètodes en tres categories principals:

- **Getters**: Que ens retornen informació concreta.
- **Setters**: Que ens permeten ajustar informació concreta.
- **Uns altres**: Que ens facilitaran enormement el treball per convertir la informació.

Mètode (arguments)	Descripció
getDate()	Retorna el dia del mes (de l'1 al 31)
getDay()	Retorna el dia de la setmana (de 0 a 6)
getFullYear()	Retorna l'any
getHours()	Retorna l'hora (de 0 a 23)
getMilliseconds()	Retorna els mil·lisegons (de 0 a 999)
getMinutes()	Retorna els minuts (de 0 a 59)
getMonth()	Retorna el mes (de 0 a 11)
getSeconds()	Retorna els segons (de 0 a 59)
getTime()	Retorna el nombre de mil·lisegons des de mitjanit de l'1 de gener de 1970 i una data especificada
getTimezoneOffset()	Retorna la diferència horària entre l'hora UTC i l'hora local, en minuts
getUTCDate()	Retorna el dia del mes, segons l'hora universal (de l'1 al 31)
getUTCDay()	Retorna el dia de la setmana, segons l'hora universal (de 0 a 6)
getUTCFullYear()	Retorna l'any, segons el temps universal
getUTCHours()	Retorna l'hora, segons l'hora universal (de 0 a 23)
getUTCMilliseconds()	Retorna els mil·lisegons, segons el temps universal (de 0 a 999)
getUTCMinutes()	Retorna els minuts, segons el temps universal (de 0 a 59)
getUTCMonth()	Retorna el mes, segons el temps universal (de 0 a 11)
getUTCSeconds()	Retorna els segons, segons el temps universal (de 0 a 59)
getYear()	Depreciat. Utilitza el mètode getFullYear() en canvi
now()	Retorna el nombre de mil·lisegons des de mitjanit de l'1 de gener de 1970
parse()	Analitza una cadena de dates i retorna el nombre de mil·lisegons des de l'1 de gener de 1970
setDate()	Estableix el dia del mes d'un objecte de data
setFullYear()	Estableix l'any d'un objecte de data
setHours()	Estableix l'hora d'un objecte de data
setMilliseconds()	Estableix els mil·lisegons d'un objecte de data
setMinutes()	Estableix les actes d'un objecte de data
setMonth()	Estableix el mes d'un objecte de data
setSeconds()	Estableix els segons d'un objecte de data
setTime()	Fixa una data en un nombre especificat de mil·lisegons després o abans de l'1 de gener de 1970
setUTCDate()	Fixa el dia del mes d'un objecte de data, segons el temps universal
setUTCFullYear()	Estableix l'any d'un objecte de data, segons el temps universal
setUTCHours()	Fixa l'hora d'un objecte de data, segons el temps universal
setUTCMilliseconds()	Estableix els mil·lisegons d'un objecte de data, segons el temps universal

setUTCMinutes()	Estableix els minuts d'un objecte de data, segons l'hora universal
setUTCMonth()	Fixa el mes d'un objecte de data, segons el temps universal
setUTCSeconds()	Fixa els segons d'un objecte de data, segons el temps universal
setYear()	Depreciat. Utilitza el mètode setFullYear() en canvi
toDateString()	Converteix la part de data d'un objecte Date en una cadena llegible
toGMTString()	Depreciat. Utilitza el mètode toUTCString() en canvi
toISOString()	Retorna la data com a cadena, utilitzant l'estàndard ISO
toJSON()	Retorna la data com a cadena, formatada com a data JSON
toLocaleDateString()	Retorna la part de data d'un objecte Date com a cadena, utilitzant convencions locals
toLocaleTimeString()	Retorna la part de temps d'un objecte Date com a cadena, utilitzant convencions locals
toLocaleString()	Converteix un objecte Date en una cadena, utilitzant convencions de localització
toString()	Converteix un objecte Date en una cadena
toTimeString()	Converteix la part temporal d'un objecte Date en una cadena
toUTCString()	Converteix un objecte Date en una cadena, segons el temps universal
UTC()	Retorna el nombre de mil·lisegons en una data des de mitjanit de l'1 de gener de 1970, segons l'hora UTC
valueOf()	Retorna el valor primitiu d'un objecte Date

En següent exemple s'apliquen el mètode **toLocaleDateString()** per quedar-nos només amb la data amb un format estàndard, o els mètodes **getDay()**, **getDate()**, **getMonth()** i **getFullYear()** per recuperar cadascuna de les informacions de la data actual per donar-li el format que vulguem. Com que aquests mètodes tornen un número, utilitzem matrius amb els noms personalitzats per convertir els nombres a text. Finalment, utilitzem els mètodes **getHours()**, **getMinutes()** i **getSeconds()** per recuperar la informació de l'hora, amb condicions ternàries per afegir un 0 inicial si el valor és inferior a 10:

```
<script>
    const d = new Date();

    console.log( d.toLocaleDateString() );
    // 10/1/2026

    const diesSetmana = ['Diumenge','Dilluns', 'Dimarts', 'Dimecres', 'Dijous',
'Divendres', 'Dissabte'];
    const mesosAny = ['gener', 'febrer', 'març', 'abril', 'maig', 'juny', 'juliol',
'agost', 'setembre', 'octubre', 'novembre', 'desembre'];

    console.log( diesSetmana[d.getDay()] + " " + d.getDate() + " de " +
mesosAny[d.getMonth()] + " del " + d.getFullYear() );
    // Dissabte 10 de gener del 2026

    console.log(`${(d.getHours()<10)? "0"+d.getHours() : d.getHours()} :
${(d.getMinutes()<10)? "0"+d.getMinutes() : d.getMinutes()} :
${(d.getSeconds()<10)? "0"+d.getSeconds() : d.getSeconds()}`);
    // 09 : 08 : 38
</script>
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

OBJECTES HOST

Són objectes que no formen part del llenguatge, sinó que els proporciona l'entorn on s'executa JavaScript; en el nostre cas, el navegador i el mateix document HTML.

BOM

El **Browser Object Model** és la forma que interpreta el JavaScript el navegador com un objecte: d'aquesta manera li pot aplicar propietats i mètodes. L'objecte **Window** té una sèrie de propietats com **Console**, **History**, **Location**, **Navigator** o **Screen**, i a la seva vegada aquestes propietats tenen els seus propis mètodes:

Propietats i mètodes WINDOW	Descripció
window.addEventListener()	Adjunta un gestor d'esdeveniments a una finestra
window.alert() o alert()	Mostra una caixa d'alerta amb un missatge i un botó d'acceptació
window.confirm() o confirm()	Mostra un quadre de diàleg amb un missatge, un botó d'OK i un botó de Cancel
window.prompt() o prompt()	Mostra un quadre de diàleg que demana a l'usuari l'entrada de text
window.open()	Obre una nova finestra del navegador o una nova pestanya, depenent de la configuració del navegador i dels valors dels paràmetres
setInterval()	Crida una funció a intervals específics (en mil·lisegons).
clearInterval()	Esborra un temporitzador establert amb el mètode setInterval()
setTimeout()	Crida una funció després d'un nombre de mil·lisegons
clearTimeout()	Esborra un temporitzador establert amb el mètode setTimeout()
window.innerHeight	Retorna l'alçada de l'àrea de contingut d'una finestra
window.innerWidth	Retorna l'amplada de l'àrea de contingut d'una finestra
window.outerHeight	Retorna l'alçada exterior de la finestra del navegador, incloent-hi tots els elements de la interfície (com les barres d'eines o les barres de desplaçament).
window.outerWidth	Retorna l'amplada exterior de la finestra del navegador, incloent-hi tots els elements de la interfície (com les barres d'eines o les barres de desplaçament)
window.scrollBy()	Desplaça el document pel nombre especificat de píxels
window.scrollTo()	Desplaça el document fins a les coordenades especificades
window.scrollX **window.scrollY** **window.pageXOffset** **window.pageYOffset**	Retorna els píxels que un document ha desplaçat des de la cantonada superior esquerra de la finestra
window.print()	Obre el quadre de diàleg d'impressió, que permet a l'usuari seleccionar opcions d'impressió preferides
window.localStorage o localStorage	L'objecte localStorage emmagatzema dades sense data de caducitat. Les dades no s'eliminen i estan disponibles per a futures sessions.
localStorage.setItem()	Et permet desar parells clau/valor al navegador
localStorage.getItem()	Et permet recuperarel valor d'una clau
window.console o console	Proporciona accés a la consola de depuració del navegador
console.log()	Envia un missatge a la consola
console.table()	Mostra les dades tabulars com una taula
window.location o location	Conté informació sobre l'URL actual
location.hash	Estableix o retorna la part d'ancoratge (#) d'una URL
location.host	Estableix o retorna el nom d'amfitrió i el número de port d'una URL
location.hostname	Estableix o retorna el nom d'host d'una URL

Generalitat de Catalunya
MINISTERIO DE EDUCACIÓN, FORMACIÓN PROFESIONAL Y DEPORTES
MINISTERIO DE TRABAJO Y ECONOMÍA SOCIAL
SERVICIO PÚBLICO DE EMPLEO ESTATAL SEPE SERVEI PÚBLIC D'OCUPACIÓ ESTATAL
Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

location.href	Estableix o retorna tota la URL
location.origin	Retorna el protocol, el nom d'amfitrió i el número de port d'una URL
location.pathname	Estableix o retorna el nom del camí d'una URL
location.port	Estableix o retorna el número de port d'una URL
location.protocol	Estableix o retorna el protocol d'una URL
location.search	Estableix o retorna la part de la cadena de consulta d'una URL
window.history o history	Conté les URL visitades per l'usuari (a la finestra del navegador).
history.length	Retorna el nombre d'URLs (pàgines) a la llista d'historial
history.back()	Carrega l'URL anterior (pàgina) a la llista d'historial
history.forward()	Carrega la següent URL (pàgina) a la llista d'historial
history.go()	Carrega una URL específica (pàgina) de la llista d'historial
window.navigator o navigator	Conté informació sobre el navegador
navigator.language	Retorna el llenguatge del navegador
navigator.userAgent	Retorna la capçalera d'usuari-agent enviada pel navegador al servidor.
window.screen o screen	L'objecte pantalla conté informació sobre la pantalla del visitant
screen.availHeight	Retorna l'alçada de la pantalla (excloent la barra de tasques)
screen.availWidth	Retorna l'amplada de la pantalla (excepte la barra de tasques)
screen.height	Retorna l'alçada total de la pantalla
screen.width	Retorna l'amplada total de la pantalla

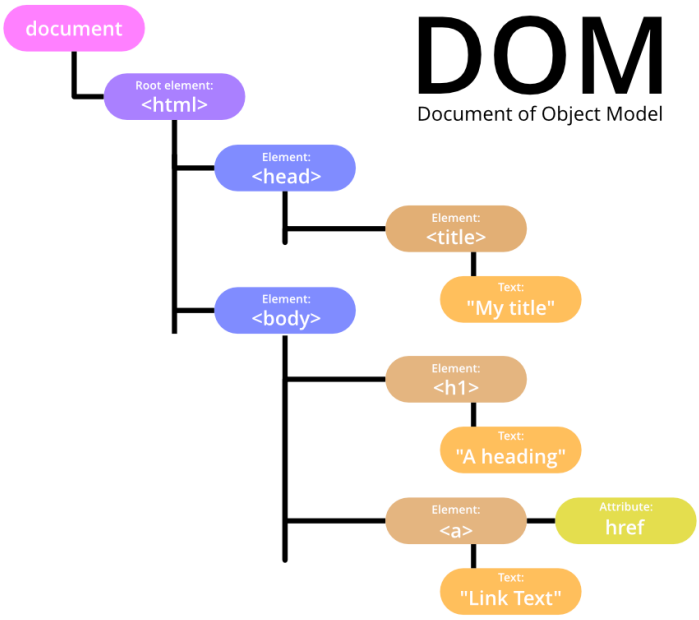
```
<script>
    var elBanner;
    window.addEventListener("load",function(){
        elBanner = setInterval(function(){
            alert("Encara hi ets aquí?");
        },5000);
    });
    function tancarBanner () {
        clearInterval(elBanner);
    }
</script>
<p>
    <button type="button" onclick="tancarBanner ();">Aturar banner</button>
</p>
```

DOM

El **Document Object Model** és la forma que interpreta el JavaScript el document com un objecte: d'aquesta manera li pot aplicar propietats i mètodes. El DOM dona una representació de les etiquetes HTML del document com un grup de **nodes** i objectes estructurats que tenen propietats i mètodes. Essencialment, connecta les pàgines web a scripts o llenguatges de programació.

En el següent esquema, cada rectangle representa un node DOM i les fletxes indiquen les relacions entre nodes. Dins de cada node, s'ha inclòs el seu tipus i el seu contingut.

Generalitat de Catalunya
MINISTERIO DE EDUCACIÓN, FORMACIÓN PROFESIONAL Y DEPORTES
MINISTERIO DE TRABAJO Y ECONOMÍA SOCIAL
SERVICIO PÚBLICO DE EMPLEO ESTATAL SEPE SERVEI PÚBLIC D'OCUPACIÓ ESTATAL
Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)



Document: node arrel del qual deriven tots els altres nodes de l'arbre.

Element: representa cadascuna de les etiquetes HTML. Es tracta de l'únic node que pot contenir atributs i l'únic del qual poden derivar altres nodes.

Attribute: es defineix un node d'aquest tipus per a representar cadascun dels atributs de les etiquetes HTML, és a dir, un per cada parell atribut="valor".

Text: node que conté el text tancat per una etiqueta HTML.

Les funcions que proporciona DOM per a accedir a un node a través dels seus nodes pare consisteixen a accedir al node arrel de la pàgina i després als seus nodes fills i als nodes fills d'aquests fills i així successivament fins a l'últim node de la branca acabada pel node buscat. No obstant això, quan es vol accedir a un node específic, és molt més ràpid accedir directament a aquest node i no arribar fins ell descendint a través de tots els seus nodes pare.

Per aquest motiu, no es presentaran les funcions necessàries per a l'accés jeràrquic de nodes i es mostren solament les propietats i mètodes de l'objecte **document** que permeten accedir de manera directa als nodes:

Propietat	Descripció
cookie	Retorna tots els parells de galetes nom/valor del document
forms	Retorna una col·lecció de tots <form> els elements del document
images	Retorna una col·lecció de tots <img> els elements del document
links	Retorna una col·lecció de tots <a> <area> els i elements del document que tenen un atribut href
title	Estableix o retorna el títol del document
URL	Retorna l'URL completa del document HTML

Mètode (arguments)	Descripció
addEventListener()	Adjunta un gestor d'esdeveniments al document
getElementById()	Retorna l'element que té l'atribut ID amb el valor especificat
getElementsByClassName()	Retorna un HTMLCollection que conté tots els elements amb el nom de classe especificat
getElementsByName()	Retorna una NodeList activa que conté tots els elements amb el nom especificat
getElementsByTagName()	Retorna un HTMLCollection que conté tots els elements amb el nom d'etiqueta especificat
hasFocus()	Retorna un valor booleà que indica si el document té focus

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

querySelector()	Retorna el primer element que coincideix amb un selector CSS especificat del document
querySelectorAll()	Retorna una NodeList estàtica que conté tots els elements que coincideixen amb un(s) selector(s) CSS especificat(s) del document
removeEventListener()	Elimina un gestor d'esdeveniments del document (que s'ha adjuntat amb el mètode addEventListener())
write()	Escriu expressions HTML o codi JavaScript en un document
writeln()	Igual que write(), però afegeix un caràcter de nova línia després de cada instrucció

```
<script>
    window.addEventListener("load",function(){
        const elBoto = document.getElementById("btnBoto");
        elBoto.addEventListener("click",function(){
            alert("Has pitjat el botó");
        });
    });
</script>
<p>
    <button type="button" id="btnBoto">Botó</button>
</p>
```

Una vegada tenim sel·leccionat un o un conjunt de nodes, podem aplicar noves propietats i mètodes:

Propietat	Descripció
length	Obtén el nombre de nodes sel·leccionats
classList	Retorna el(s) nom(s) de classe d'un element
clientHeight	Retorna l'alçada d'un element, incloent-hi el farciment
clientLeft	Retorna l'amplada de la vora esquerra d'un element
clientTop	Retorna l'amplada de la vora superior d'un element
clientWidth	Retorna l'amplada d'un element, incloent-hi el farciment
innerHTML	Estableix o retorna el contingut d'un element
innerText	Estableix o retorna el contingut textual d'un node i els seus descendents
scrollHeight	Retorna tota l'alçada d'un element, incloent-hi el farciment
scrollLeft	Estableix o retorna el nombre de píxels en què el contingut d'un element està desplaçat horitzontalment
scrollTop	Estableix o retorna el nombre de píxels en què el contingut d'un element es desplaça verticalment
scrollWidth	Retorna tota l'amplada d'un element, incloent-hi el farcit
style	Estableix o retorna el valor de l'atribut d'estil d'un element
textContent	Estableix o retorna el contingut textual d'un node i els seus descendents
(nomAtribut)	Qualsevol atribut d'una etiqueta HTML esdevé una propietat

Mètode (arguments)	Descripció
addEventListener()	Adjunta un gestor d'esdeveniments a un element
checkValidity()	Comprova si l'element té restriccions i si les compleix. Si l'element no compleix les seves restriccions, retorna fals.
hasAttribute()	Retorna cert si un element té un atribut donat
hasAttributes()	Retorna cert si un element té qualsevol atribut

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

hasChildNodes()	Retorna cert si un element té nodes fills
scrollIntoView()	Desplaça l'element a l'àrea visible de la finestra del navegador
setAttribute()	Estableix o canvia el valor d'un atribut
setAttributeNode()	Estableix o canvia un node d'atribut
reset()	Restableix un formulari al seu estat inicial
submit()	Envia les dades d'un formulari
preventDefault()	Cancel·la l'esdeveniment si és cancel·lable, és a dir, que l'acció per defecte que pertany a l'esdeveniment no es produirà
stopPropagation()	Impedeix que es cridi la propagació del mateix esdeveniment

Un cas especial és la propietat **classList** que fa referència a les classes CSS que s'apliquen a una etiqueta: com que això implica manipular l'estètica del node, tenim disponibles més mètodes i propietats especials només per manipular els noms de les classes:

Propietats i mètodes de classList	Descripció
add()	Afegeix un o més noms a la llista
contains()	Retorna veritable si la llista conté una classe
forEach()	Executa una funció de crida per a cada nom de la llista
length	Retorna el nombre de noms a la llista
remove()	Elimina un o més noms de la llista
replace()	Substitueix un nom a la llista
toggle()	Canvia entre fitxes a la llista
value	Retorna la llista de noms com una cadena
values()	Retorna un iterador amb els valors de la llista

En el següent exemple afegim un gestor d'esdeveniment a l'objecte window per esperar al fet que es carregui el document i que tinguem disponibles els botons. A continuació se seleccionen tots pel seu nom de classe CSS, i a cadascun afegim un gestor d'esdeveniment per detectar quan es pitgen per mostrar una alerta personalitzada amb el contingut del botó:

```
<script>
    window.addEventListener("load",function(){
        const elBoto = document.querySelectorAll(".elBoto");
        elBoto.forEach(function(b){
            b.addEventListener("click",function(){
                alert("Has pitjat el botó " + b.textContent);
            });
        });
    });
</script>
<p>
    <button type="button" class="elBoto">Botó 1</button>
    <button type="button" class="elBoto">Botó 2</button>
    <button type="button" class="elBoto">Botó 3</button>
</p>
```

Amb el mètode addEventListener() ja no cal afegir els esdeveniments com atributs de les etiquetes HTML, per tant el codi queda molt més net.

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

Aquest mètode necessita com primer argument l'esdeveniment a escoltar: el nom de l'esdeveniment és el mateix que el del atribut HTML però sense l'"**on**" inicial:

Nom de l'esdeveniment com argument	Definició
blur	Quan un element de formulari perd el focus
change	Quan el valor d'un camp de formulari és modificat
click	Quan es fa clic amb el botó del ratolí
contextmenu	Quan es fa clic amb el botó alternatiu del ratolí
dblclick	Quan es fa doble clic en un objecte
focus	Quan un element de formulari adquireix el focus
input	Quan s'està modificant un camp de formulari
keydown	Quan es pressiona una tecla
keypress	Quan es pressiona una tecla
keyup	Quan es deixa de pressionar una tecla
load	Quan una pàgina o imatge acaba de carregar-se
mousedown	Quan es pitja el botó del ratolí
mousemove	Quan es mou el ratolí
mouseout	Quan el cursor del ratolí surt de l'element
mouseover	Quan el cursor del ratolí es posa damunt
mouseup	Quan es deixa anar el botó del ratolí
reset	Quan es pitja el botó de reset d'un formulari
resize	Quan es modifica la grandària d'una finestra
select	Quan se selecciona text d'un camp de formulari
submit	Quan es pitja el botó submit d'un formulari
wheel	Quan la roda del ratolí puja o baixa sobre un element

En el següent exemple s'utilitzen diferents mètodes per controlar l'enviament d'un formulari validant-ho amb JavaScript i no pel navegador:

```
<script>
    window.addEventListener("load",function(){
        const forms = document.querySelectorAll('.per-validar');
        forms.forEach(form => {
            form.addEventListener("submit", event => {
                event.preventDefault();
                event.stopPropagation();
                if ( form.checkValidity() ) {
                    form.submit();
                } else {
                    alert("Verifica els camps del formulari");
                }
            });
        });
    });
</script>
<form class="per-validar" novalidate>
    <p>
        <label for="idNom">Nom</label>
        <input type="text" id="idNom" required name="elNom">
    </p>
    <p>
        <button type="submit">Enviar</button>
    </p>
</form>
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

A partir d'ara, amb la manipulació del DOM i accés als diferents nodes del document, podem abandonar els mètodes de l'objecte **window** com `alert()`, `confirm()` i `prompt()` per emprar elements propis del llenguatge HTML com paràgrafs, divisors, botons i entrades de formulari per interactuar amb els visitants i millorar l'experiència d'usuari.

En el següent exemple, s'empra un formulari perquè el visitant introdueixi les dades, es manipula la propietat "value" i "checked" per recollir la informació i es mostra la resposta de forma asíncrona en un contenidor:

```
<script>
    window.addEventListener("load",function(){
        const formAcces = document.getElementById('formAcces');
        const missatgeAcces = document.getElementById('missatgeAcces');

        formAcces.addEventListener("submit", event => {
            event.preventDefault();
            event.stopPropagation();
            missatgeAcces.textContent = "";

            let edat = document.getElementById('edat').value;
            let edatNum = parseInt(edat);
            let politicaLegal = document.getElementById('politicaLegal').checked;
            console.log (edat, edatNum, politicaLegal);

            if (politicaLegal && !isNaN(edatNum) && edatNum >= 18 ) {
                formAcces.submit();
            } else if ( isNaN(edatNum) ) {
                missatgeAcces.textContent = "Cal introduir l'edat.";
            } else if ( edatNum < 18 ) {
                missatgeAcces.textContent = "No es permet l'accés a menors d'edat.";
            } else if ( !politicaLegal ) {
                missatgeAcces.textContent = "Cal acceptar la política de privacitat.";
            } else {
                missatgeAcces.textContent = "Ompli el camps del formulari.";
            }
        });

    });
</script>
<form id="formAcces">
    <p>
        <label for="edat">Introdueix la teva edat:</label>
        <input type="text" id="edat" name="edat">
    </p>
    <p>
        <input type="checkbox" id="politicaLegal" name="politicaLegal">
        <label for="politicaLegal">Accepto la política de privacitat</label>
    </p>
    <p>
        <button type="submit">Validar</button>
    </p>
</form>
<p id="missatgeAcces"></p>
```

Generalitat de Catalunya
MINISTERIO DE EDUCACIÓN, FORMACIÓN PROFESIONAL Y DEPORTES
MINISTERIO DE TRABAJO Y ECONOMÍA SOCIAL
SERVICIO PÚBLICO DE EMPLEO ESTATAL SEPE SERVEI PÚBLIC D'OCUPACIÓ ESTATAL

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

OBJECTES LITERALS

Un objecte literal és un conjunt de claus i valors. Cada clau és un nom, i cada valor pot ser el que vulguis: un número, un text, un altre objecte, una funció… És com dir: "*Tinc una cosa que té aquestes característiques i pot fer aquestes accions*".

Un objecte en JavaScript és un contenidor flexible que agrupa dades i comportaments sota un mateix nom.

L'OBJECTE és una entitat de la vidal real que es trasllada al paradigma informàtic: tenen ATRIBUTS com característiques que el poden definir i tenen MÈTODES que són accions que poden realitzar els objectes sobre els seus propis atributs o sobre els d'altres objectes. Per exemple: l'objecte cotxe té propietats: "color", "marca", "model", "motor"; també te funcions: arrancar() o frenar().

Una forma ràpida de crear un objecte és instanciar una variable amb les claus "{}" i afegir dins les claus de dades que necessitem amb els valors corresponents emprant l'operador dos punts ":" i separant les claus amb coma ",". Els mètodes es defineixen com una clau més però amb una funció anònima com a valor. Una vegada declarat i instanciat l'objecte, podem fer-li referència pel nom i utilitzar les propietats i els mètodes amb l'operador del punt ".":

```
<script>
    const cotxe = {
        color:'negre',
        marca:'js',
        model:'object',
        motor:'híbrid',
        arrancar: function () {console.log("arrancar");},
        frenar: function () {console.log("frenar");}
    }
    cotxe.marca = "JavaScript";
    console.log(cotxe.marca);
    cotxe.arrancar();
    cotxe.frenar();
</script>
```

Els valors que podem emmagatzemar poden ser de qualsevol dels tipus vists amb les variables.

Les CLASSES són plantilles que defineixen quins atributs i mètodes han de tenir tots els objectes creats a partir d'aquesta classe; no asigna valors, només els tipus d'aquests. Les classes també poden tenir herències: d'aquesta forma les classes principals són Superclasses, i les subordinades són inferiors i es diuen Subclasses.

Utilitzem **class** per declarar un nou objecte (els objectes comencen sempre amb majúscula i en singular), i **extends** si aquesta és una subclasse d'una classe previament declarada.

La instanciació és l'acció de crear un objecte a partir d'una classe dins d'una variable i ho fem amb **new**. En el següent exemple simularem una fàbrica de brioixeria i pastissos:

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

```
<script>
    class Brioixeria {
        // Atributs amb els valors per defecte per definir el tipus:
        nom         = '';
        sabor       = '';
        pes         = 0;
        color       = '';
        racions     = 0;
    }
    class Pastis extends Brioixeria {
        // Atributs amb els valors per defecte per definir el tipus:
        // Com que aquest objecte depen d'una superclasse, els atributs originals no cal cridar-los
        espelmes    = 0;
    }
    const article1 = new Brioixeria();
    article1.nom         = 'Croissant';
    article1.sabor       = 'mantega';
    article1.pes         = 0.15;
    article1.color       = 'beix';
    article1.racions     = 1;

    document.writeln(`<dl>
        <dt>Nom:</dt><dd>${article1.nom}</dd>
        <dt>Sabor:</dt><dd>${article1.sabor}</dd>
        <dt>Pes:</dt><dd>${article1.pes} Kg</dd>
        <dt>Color:</dt><dd>${article1.color}</dd>
        <dt>Racions</dt><dd>${article1.racions}</dd>
    </dl>`);

    const article2 = new Pastis();
    article2.nom         = 'Pastís de poma';
    article2.sabor       = 'poma i caramel';
    article2.pes         = 1.5;
    article2.color       = 'marró';
    article2.racions     = 6;
    article2.espelmes    = 12;

    document.writeln(`<dl>
        <dt>Nom:</dt><dd>${article2.nom}</dd>
        <dt>Sabor:</dt><dd>${article2.sabor}</dd>
        <dt>Pes:</dt><dd>${article2.pes} Kg</dd>
        <dt>Color:</dt><dd>${article2.color}</dd>
        <dt>Racions</dt><dd>${article2.racions}</dd>
        <dt>Espelmes</dt><dd>${article2.espelmes}</dd>
    </dl>`);
</script>
```

En aquest exemple tenim una Superclasse Brioixeria que té una sèrie d'atributs assignats. I d'aquesta es crea una Subclasse Pastis que hereta els seus atributs, però es poden afegir nous atributs per personalitzar el nou objecte.

Si l'objecte té moltes propietats assignades, instanciar cadascuna d'elles individualment amb l'operador del punt és molt pessat. L'opció per agilitzar l'assignació inicial de valors és crear un constructor o funció constructora: dins de la classe, utilitzem **constructor** com nom de mètode genèric que permet afegir valors als atributs en el mateix moment de la instanciació. Utilitzem **this.** en les propietats per centrar l'àmbit -scope - dins de l'objecte i així cridem només als seus atributs, però si l'objecte és una Subclasse i hereta les propietas de la seva Superclasse, utilitzem **super.** si fem referència a elements heretats:

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

```
<script>
    class Brioixeria {
        nom         = '';
        sabor       = '';
        pes         = 0;
        color       = '';
        racions     = 0;

        constructor (nom, sabor, pes, color, racions) {
            this.nom = nom;
            this.sabor = sabor;
            this.pes = pes;
            this.color = color;
            this.racions = racions;
        }
    }
    class Pastis extends Brioixeria {
        espelmes    = 0;

        constructor (nom, sabor, pes, color, racions, espelmes) {
            super (nom, sabor, pes, color, racions);
            this.espelmes = espelmes;
        }
    }
    const article1 = new Brioixeria('Croissant', 'mantega', 0.15, 'beix', 1);

    document.writeln(`<dl>
        <dt>Nom:</dt><dd>${article1.nom}</dd>
        <dt>Sabor:</dt><dd>${article1.sabor}</dd>
        <dt>Pes:</dt><dd>${article1.pes} Kg</dd>
        <dt>Color:</dt><dd>${article1.color}</dd>
        <dt>Racions</dt><dd>${article1.racions}</dd>
    </dl>`);

    const article2 = new Pastis('Pastís de poma', 'poma i caramel', 1.5, 'marró',
6, 12);

    document.writeln(`<dl>
        <dt>Nom:</dt><dd>${article2.nom}</dd>
        <dt>Sabor:</dt><dd>${article2.sabor}</dd>
        <dt>Pes:</dt><dd>${article2.pes} Kg</dd>
        <dt>Color:</dt><dd>${article2.color}</dd>
        <dt>Racions</dt><dd>${article2.racions}</dd>
        <dt>Espelmes</dt><dd>${article2.espelmes}</dd>
    </dl>`);
```

A més del mètode per defecte de constructor, podem afegir mètodes o accions personalitzades als nostres objectes declarant funcions dins del propi objecte i modificant-lo en les Subclasses:

```
<script>
    class Brioixeria {
        nom         = '';
        sabor       = '';
        pes         = 0;
        color       = '';
        racions     = 0;
        constructor (nom, sabor, pes, color, racions) {
            this.nom = nom;
            this.sabor = sabor;
            this.pes = pes;
            this.color = color;
            this.racions = racions;
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

```
        }
        aText () {
            return `Sóc un ${this.nom} de sabor de ${this.sabor} amb un pes de
${this.pes} Kg, de color ${this.color} per a ${this.racions} racions`;
        }
    }

    class Pastis extends Brioixeria {
        espelmes    = 0;
        constructor (nom, sabor, pes, color, racions, espelmes) {
            super (nom, sabor, pes, color, racions);
            this.espelmes = espelmes;
        }
        aText () {
            return `Sóc un ${this.nom} de sabor de ${this.sabor} amb un pes de
${this.pes} Kg, de color ${this.color} per a ${this.racions} racions amb
${this.espelmes} espelmes`;
        }
    }

    const article1 = new Brioixeria('Croissant', 'mantega', 0.15, 'beix', 1);
    document.writeln( "<p>" + article1.aText() + "</p>");
    const article2 = new Pastis('Pastís de poma', 'poma i caramel', 1.5, 'marró', 6, 12);
    document.writeln( "<p>" + article2.aText() + "</p>");
</script>
```

Tot i que les propietats es poden recuperar i tornar a instanciar en qualsevol moment perquè JavaScript és un llenguatge que dona molta flexibilitat, en altres llenguatges orientats a objectes, les propietats es defineixen com a privades i no es poden utilitzar tan lliurement des de fora de la pròpia definició de l'objecte, d'aquí que es defineixin mètodes **getters** i **setters** per poder manipular-les:

GETTERS

Els Getters o micromètodes són mètodes per poder llegir els valors dels atributs, necessaris en altres llenguatges orientats a objectes perquè són privats i no es poden consultar des de fora de l'objecte. Com la resta de mètodes, són heretables i només cal definir-los una vegada en la Superclasse amb el nom que vulguem. En el nostre exemple:

```
    getNom () {
        return this.nom;
    }
    getSabor () {
        return this.sabor;
    }
    getPes () {
        return this.pes;
    }
    getColor() {
        return this.color;
    }
    getRacions () {
        return this.racions;
    }
```

I en la Subclasse:

```
    getEspelmes () {
        return this.espelmes;
    }
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

SETTERS

Els Setters o micromètodes són mètodes per poder modificar els valors dels atributs, necessaris en altres llenguatges orientats a objectes perquè són privats i no es poden modificar des de fora de l'objecte. Com la resta de mètodes, són heretables i només cal definir-los una vegada en la Superclasse amb el nom que vulguem. En el nostre exemple:

```
    setNom (v) {
        this.nom = v;
    }
    setSabor (v) {
        this.sabor = v;
    }
    setPes (v) {
        this.pes = v;
    }
    setColor(v) {
        this.color = v;
    }
    setRacions (v) {
        this.racions = v;
    }
```

I en la Subclasse:

```
    setEspelmes (v) {
        this.espelmes = v;
    }
```

La incorporació d'aquests micromètodes faria modificar els altres mètodes per incorporar-los. En el nostre exemple, modificaríem el mètode per mostra la informació en la Subclasse per llegir els atributs. Com que són atributs de la Superclasse, utilitzo **super** i els getters per llegir la informació; només s'usa el **this** pels atributs propis:

```
    aText () {
        return `Sóc un pastís amb el nom ${super.getNom()} de sabor de
${super.getSabor()} amb un pes de ${super.getPes()} Kg, de color
${super.getColor()} per a ${super.getRacions()} racions amb ${this.espelmes}
espelmes`;
    }
```

Ara, tot junt, l'exemple queda:

```
<script>
    class Brioixeria {
        nom         = '';
        sabor       = '';
        pes         = 0;
        color       = '';
        racions     = 0;

        constructor (nom, sabor, pes, color, racions) {
            this.nom = nom;
            this.sabor = sabor;
            this.pes = pes;
            this.color = color;
            this.racions = racions;
        }
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

```
        getNom () {
            return this.nom;
        }
        getSabor () {
            return this.sabor;
        }
        getPes () {
            return this.pes;
        }
        getColor() {
            return this.color;
        }
        getRacions () {
            return this.racions;
        }

        setNom (v) {
            this.nom = v;
        }
        setSabor (v) {
            this.sabor = v;
        }
        setPes (v) {
            this.pes = v;
        }
        setColor(v) {
            this.color = v;
        }
        setRacions (v) {
            this.racions = v;
        }

        aText () {
            return `Sóc un ${this.nom} de sabor de ${this.sabor} amb un pes de
              ${this.pes} Kg, de color ${this.color} per a ${this.racions} racions`;
        }
    }

    class Pastis extends Brioixeria {
        espelmes    = 0;

        constructor (nom, sabor, pes, color, racions, espelmes) {
            super (nom, sabor, pes, color, racions);
            this.espelmes = espelmes;
        }

        getEspelmes () {
            return this.espelmes;
        }

        setEspelmes (v) {
            this.espelmes = v;
        }

        aText () {
            return `Sóc un pastís amb el nom ${super.getNom()} de sabor de
                    ${super.getSabor()} amb un pes de ${super.getPes()} Kg, de color
                         ${super.getColor()} per a ${super.getRacions()} racions amb
                                                    ${this.espelmes} espelmes`;
        }
    }
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

```
    const article1 = new Brioixeria(
        'Croissant', 'mantega', 0.15, 'beix', 1
    );

    document.writeln( "<p>" + article1.aText() + "</p>");

    const article2 = new Pastis(
        'Pastís de poma', 'poma i caramel', 1.5, 'marró', 6, 12
    );

    document.writeln( "<p>" + article2.aText() + "</p>");
</script>
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

JQUERY

jQuery és una llibreria de JavaScript creada per simplificar tasques habituals en el desenvolupament web. Va néixer amb l'objectiu de fer el codi més curt, més llegible i més compatible entre navegadors en una època en què cada navegador interpretava JavaScript de manera diferent. La seva filosofia es resumeix en l'eslògan "*Write less, do more*" –Escriu menys, fes més-.

Amb jQuery pots seleccionar elements del DOM amb una sintaxi molt compacta, manipular contingut HTML, manipular classes i estils CSS, gestionar esdeveniments, crear animacions i fer peticions AJAX sense haver de preocupar-te pels detalls tècnics de cada navegador. La seva funció principal, **$()**, actua com una porta d'entrada ràpida per accedir i modificar elements de la pàgina.

Tot i que avui dia JavaScript modern (ES6+) i l'API del DOM han millorat molt i han reduït la necessitat de jQuery, encara es troba en molts projectes existents i és útil per mantenir o modernitzar aplicacions antigues. També continua sent una eina senzilla per a qui comença i vol manipular el DOM de manera intuïtiva.

En el capítol dedicat al DOM, hem vist alguns mètodes i propietats (no tots) de JavaScript per manipular-ho, i hem constatat que hi ha una mescla de propietats, mètodes i mètodes de propietats. jQuery simplifica la sintaxi definint tot com mètodes i variant només els arguments de cadascun.

A més, gràcies a l'ampli catàleg de ***plugins*** o extensions desenvolupades amb aquesta llibreria, és molt fàcil implementar noves funcionalitats o connectar-les entre elles sense que estudiar la seva sintaxi o com fer-les compatibles entre elles.

jQuery no substitueix al JavaScript: el complementa per facilitar la manipulació de DOM amb una sintaxi compacta basada en mètodes que resumeixen llargues expressions.

INSTAL·LACIÓ

En ser una llibreria com Bootstrap o FontAwesome, no cal instal·lar cap programari especial: només cal enllaçar amb l'arxiu base que ens interessi, disponibles a https://jquery.com/download/

- Arxiu minificat amb totes les funcionalitats: https://code.jquery.com/jquery-3.7.1.min.js
- Arxiu *sourcemap* -fitxer que serveixen per relacionar el codi minificat amb el codi original llegible i que es pugui llegir amb claritat el codi en les eines de depuració del navegador- complementari al minificat: https://code.jquery.com/jquery-3.7.1.min.map
- Arxiu sense minificar –no recomanat en producció pel seu pes- pel desenvolupament: https://code.jquery.com/jquery-3.7.1.js
- Arxiu minificat simplificat –slim- sense efectes visuals ni ajax: https://code.jquery.com/jquery-3.7.1.slim.min.js
- Arxiu sourcemap de la versió simplificada: https://code.jquery.com/jquery-3.7.1.slim.min.map
- Arxiu sense minificar simplificat: https://code.jquery.com/jquery-3.7.1.slim.js

Aquest arxiu normalment el descarregarem i el desarem en la carpeta del projecte, amb la resta de recursos. També tenim la possibilitat d'utilitzar un **CDN** (*content delivery network*) o xarxa de lliurament de continguts on hi han copies disponibles dels arxius.

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

Una vegada descarregat o amb l'enllaç CDN que ens interessi, el vincularem amb una etiqueta `<script>` al `<head>` del nostre document, tot i que normalment es possa al final del `<body>`, amb la resta de scripts, per no enlentir la càrrega del document:

```
<script src="jquery-3.7.1.min.js"></script>
```

És important que enllacem a la llibreria jQuery <u>abans</u> de la resta de scripts que utilitzin la seva sintaxi per evitar errors de lectura.

INICIALITZACIÓ

Si jQuery està pensat per manipular el DOM, abans de començar a utilitzar-lo hem d'assegurar-nos que tot el document estigui completament carregat, per tant, usarem una expressió similar al `window.addEventListener("load",function(){});` per esperar a executar el codi a què els nodes estiguin disponibles:

```
<script src="jquery-3.7.1.min.js"></script>
<script>
    $(document).ready(function(){
        ...
    });
</script>
```

Tot i que hi ha una versió abreviada més còmoda i ràpida:

```
<script src="jquery-3.7.1.min.js"></script>
<script>
    $(function(){
        ...
    });
</script>
```

A partir d'ara, totes les nostres sentències aniran dins de les claus d'aquesta funció anònima que s'invocarà una vegada el document estigui carregat.

SELECTORS

A partir d'ara, cada vegada que implementem una nova sentencia, l'estructura serà clara: **selector.mètode()**;

Per fer el selector necessitem l'expressió pròpia del jQuery **`$("selector")`**, on el selector pot ser qualsevol dels possibles selectors que ens dona el CSS. Aquesta expressió és similar al:
`document.querySelectorAll("selector");`

A part de simplificar, jQuery també farà que qualsevol mètode que apliquem al selector s'apliqui a tots els elements, és a dir, el mateix jQuery farà el bucle `forEach()` per nosaltres:

```
<script src="jquery-3.7.1.min.js"></script>
<script>
    $(function(){
        $('.ocult').hide();
    });
</script>
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

MÈTODES

El llistat de mètodes que mostra la documentació de jQuery a https://api.jquery.com/ és molt ampli, per això recomano una altra webapp que he desenvolupat com a guia visual, on els mètodes apareixen organitzats per temàtica: https://onaweb.cat/jquery/

En pitjar cadascun dels mètodes, s'obre una finestra amb la documentació original en anglès, on no només s'expliquen els arguments necessaris, sinó també hi ha exemples d'ús.

Cal mencionar que el JavaScript té moltes propietats que poden ser de lectura o escriptura si estan a la dreta o a l'esquerra de l'operador d'assignació. En jQuery també estan definits com a mètodes, però aquests seran de lectura –*getters*- si només tenen un argument, o d'escriptura –*setters*- si tenen dos arguments.

Si algun dels arguments és de tipus funció, com els *handler* dels mètodes d'esdeveniments, serà una funció anònima on podrem afegir el codi a executar en el cas que el mètode s'executi. Dins d'aquestes funcions anònimes utilitzarem la clàusula **this** per fer referència a l'objecte amb el qual s'està interactuant en aquell moment, així que l'ús de la funció anònima o funció de fletxa podria donar errors en la interpretació del **this**.

Una altra opció que té el JavaScript i també el jQuery és l'opció de concatenar diferents mètodes amb el punt: **`$("selector").mètode1().mètode2().mètode3()...;`**

Alguns d'aquest mètodes tenen la clàusula **`jQuery`** o **`$`** en lloc d'un selector perquè no cal aplicar a un node en concret: es poden emprar com una funció.

En el següent exemple hi han tres botons amb la mateixa classe CSS però diferents texts dins, i un contenidor identificat buit. A continuació hi ha l'enllaç a la lliberia jQuery i la seva inicialització, on hi ha un selector pels botons per la seva classe i un mètode d'esdeveniment amb dos arguments: l'esdeveniment "click" i la funció anónima amb dos sentències:

```
<p>
    <button type="button" class="btnBoto">Botó 1</button>
    <button type="button" class="btnBoto">Botó 2</button>
    <button type="button" class="btnBoto">Botó 3</button>
</p>
<div id="sortida"></div>

<script src="jquery-3.7.1.min.js"></script>
<script>
    $(function(){
        $('.btnBoto').on('click',function(){
            let textBoto = $(this).text();
            $('#sortida').text(textBoto)
        });
    });
</script>
```

En la primera sentència es declara la variable local "textBoto" i s'instància amb el selector **`$(this)`** que equival al botó exacte sobre el qual s'ha pitjat amb el mètode **`text()`** que equival a la propietat `innerText` o `textContent`; com que no té cap argument, aquest mètode funciona com lectura –*getter*-.

En la següent sentència tornem a començar amb un selector, en aquest cas el contenidor amb l'identificador, i tornem a aplicar el mètode **`text()`** que, en aquesta ocasió, sí que té un argument, per tant, funciona com escriptura –*setter*- i mostra el contingut de la variable "textBoto".

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

SELECTORS

En aquesta categoria tenim un resum de les diferents formes que tenim per seleccionar nodes del DOM, la majoria heretats del llenguatge CSS.

ATRIBUTS / CSS

En aquesta categoria tenim els mètodes per manipular (llegir o escriure) els atributs de les etiquetes HTML i les propietats CSS, així com les classes CSS que s'apliquen. També s'inclouen mètodes per manipular les dimensions, la posició en la finestra i els atributs `data-` de les etiquetes HTML.

Aquest atribut "data" mereix una subcategoria a part perquè és una forma molt habitual d'emmagatzemar informació pròpia per aquesta etiqueta (com una variable local) per després recuperar-la amb JavaScript i, en el nostre cas, amb jQuery.

Seguint l'exemple inicial del capítol, s'han afegit atributs `data-info` als botons per poder personalitzar el text a mostrar. Per tant, cada vegada que pitgem un botó hem que recuperar la informació d'aquell mateix botó, i ho fem amb el mètode **`data('info')`** on "info" és l'argument per especificar quin de tots els atributs `data-` que pot tenir una etiqueta volem seleccionar. Com que no hi ha un segon argument de valor, significa que és de lectura –*getter*-:

```
<p>
    <button type="button" class="btnBoto" data-info="Informació 1">Botó 1</button>
    <button type="button" class="btnBoto" data-info="Informació 2">Botó 2</button>
    <button type="button" class="btnBoto" data-info="Informació 3">Botó 3</button>
</p>
<div id="sortida"></div>
<script src=" jquery-3.7.1.min.js "></script>
<script>
    $(function(){
        $('.btnBoto').on('click',function(){
            let textBoto = $(this).data('info');
            $('#sortida').text(textBoto)
        });
    });
</script>
```

MANIPULACIÓ

Són mètodes per manipular el contingut o el que l'envolta, duplicar o fins i tot eliminar el node seleccionat. En l'exemple anterior es veu l'ús del mètode **`text()`** per manipular el text contingut en un node.

TRAVESSANT

Aquest conjunt de mètodes ens permet moure'ns per la estructura de nodes del document. Sempre necessitem començar des d'un node origen, el seleccionat, i a partir d'aquests ens podem moure a nodes inferiors –els fills-, superiors –els pares- o els que estan al mateix nivell –els germans-.

ESDEVENIMENTS

Aquest conjunt de mètodes ens permeten detectar les interaccions de l'usuari amb el ratolí, el teclat o relacionats amb els formularis.

En el següent exemple es modifica un exemple anterior per emprar la sintaxi jQuery: en aquest cas el selector del formulari (o formularis) ara és més senzill, no cal emprar cap bucle però si volem seguir utilitzant mètodes originals del JavaScript -`.checkValidity()` i `.submit()`- i que el jQuery no es confongui, afegim el

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

mètode **`.get(0)`** per indicar que en l'array de possibilitats del jQuery –per jQuery tot conjunt d'elements és un array, encara que sigui d'un únic element-, ens quedem amb el primer node –el formulari que estem manipulant-:

```
<form class="per-validar" novalidate>
    <p>
        <label for="idNom">Nom</label>
        <input type="text" id="idNom" required name="elNom">
    </p>
    <p>
        <button type="submit">Enviar</button>
    </p>
</form>
<script src=" jquery-3.7.1.min.js "></script>
<script>
    $(function(){
        $('.per-validar').on("submit", function(event) {
            event.preventDefault();
            event.stopPropagation();
            if ( $(this).get( 0 ).checkValidity() ) {
                $(this).get( 0 ).submit();
            } else {
                alert("Verifica els camps del formulari");
            }
        });
    });
</script>
```

EFECTES

Aquí tenim tot de mètodes per efectes visuals; no són efectes gaire complexos, però per fer efectes senzills per millorar l'experiència d'usuari poden ser un bon punt de partida.

Seguint l'exemple anterior, no emprarem un `alert()` sinó un missatge en pantalla ja definit al codi HTML:

```
<form class="per-validar" novalidate>
    <p>
        <label for="idNom">Nom</label>
        <input type="text" id="idNom" required name="elNom">
    </p>
    <p>
        <button type="submit">Enviar</button>
    </p>
</form>
<div class="avis">Verifica els camps del formulari</div>
<script src=" jquery-3.7.1.min.js "></script>
<script>
    $(function(){
        $('.avis').hide();
        $('.per-validar').on("submit", function(event) {
            event.preventDefault();
            event.stopPropagation();
            if ( $(this).get( 0 ).checkValidity() ) {
                $(this).get( 0 ).submit();
            } else {
                $(this).next('.avis').slideDown();
            }
        });
    });
</script>
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

En aquest exemple, en carregar el document ocultem el node amb el missatge, i quan es valida i dona un error de validació, des del formulari ens movem al següent node (el del missatge) per desplegar-lo.

AJAX

AJAX és una tècnica de JavaScript que permet comunicar-se amb el servidor sense recarregar tota la pàgina. El nom prové d'***Asynchronous JavaScript and XML***, tot i que avui dia no cal utilitzar XML: també funciona amb JSON, text o qualsevol altre format.

L'asincronia és la capacitat d'executar tasques sense bloquejar l'aplicació, permetent que altres operacions continuïn mentre una acció lenta es resol en segon pla.

La idea central és senzilla: la pàgina envia una petició al servidor "en segon pla", rep la resposta i actualitza només la part necessària del document. Això fa que les aplicacions web siguin més ràpides, fluides i interactives, perquè l'usuari no veu cap tall ni recàrrega completa.

Amb AJAX pots, per exemple, carregar dades noves, enviar formularis, actualitzar llistes o validar informació sense sortir de la pàgina. Inicialment es feia servir l'objecte `XMLHttpRequest`, però avui és molt habitual utilitzar l'API `fetch`, que és més moderna i clara.

En resum, AJAX és el mecanisme que permet que moltes webs funcionin de manera dinàmica i reactiva, fent que la comunicació amb el servidor sigui transparent per a l'usuari.

jQuery inclou la seva implementació amb el mètode **`ajax()`** i la resta de mètodes auxiliars per facilitar la gestió de dades i respostes.

```
<div id="galeria"></div>
<script src=" jquery-3.7.1.min.js "></script>
<script>
    $(function(){
        $.getJSON( "https://picsum.photos/v2/list", function( data ) {
            let items = [];
            $.each( data, function( key, val ) {
                items.push( `<div><img src="${val.download_url}" alt="foto
                                        ${val.id}" title="${val.author}"></div>` );
            });
            $("#galeria").append( items.join('') );
        });
    });
</script>
```

En l'anterior exemple es veu en ús el mètode **`$.getJSON()`** per connectar amb un servidor que dona un document json amb una llista de 30 imatges; una vegada establida la connexió, llegit l'arxiu i descodificat en format array, el podem recórrer per emmagatzemar en l'array local "items" els nous nodes d'HTML amb imatges. Els valors dels atributs de les imatges s'estableixen a partir de la informació del **JSON** (***JavaScript Object Notation***). Una vegada recorregut, es mostra el contingut dins del contenidor amb l'identificador "galeria".

NUCLI

Aquí tenim un "calaix desastre" on tenim la resta de mètodes que no pertanyen a la resta de categories. Molts mètodes són ajudes i auxiliars, com el mètode **`$.each()`** –que hem vist en l'exemple anterior- per fer un bucle als elements d'un array o el mètode **`.get()`** –que ja hem vist en exemples anteriors- per seleccionar el node natiu de JavaScript d'un array de nodes del jQuery.

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

PLUGINS

Un *plugin* de JavaScript és un petit mòdul de codi pensat per afegir funcionalitats concretes a una pàgina o aplicació sense haver de reescriure-les des de zero. Actua com una extensió reutilitzable: encapsula una característica (per exemple, un carrusel, un selector de dates o un sistema de pestanyes) i permet integrar-la fàcilment en diferents projectes mitjançant una API senzilla.

Però depèn de l'autor com estan desenvolupats i la sintaxi emprada, i això implica que cal llegir molt bé la documentació i estudiar el seu funcionament per entendre com personalitzar-lo i fer-lo funcionar amb la resta del codi del projecte.

En l'ecosistema de jQuery, per exemple, els *plugins* han estat una manera molt popular de compartir solucions i ampliar les capacitats de la llibreria i, a més, en estar escrits tots sota les mateixes premisses, són fàcils d'entendre i d'implementar.

Cada vegada que vulguem afegir un nou *plugin*, caldrà seguir les següents **regles d'or**:

1. **Trobar la pàgina web de documentació del *plugin***: a vegades l'autor del *plugin* ha creat un lloc web específic per documentar el *plugin*, però normalment els trobarem en la plataforma de github.com.
2. **Descarregar els arxius del *plugin***: en la mateixa web de l'autor trobarem l'enllaç de descàrrega o, si ja estem en GitHub, podem descarregar-nos el lot sencer.
3. **Localitzar els arxius necessaris per copiar-los al nostre projecte**: llegint la documentació veurem quins .css i .js necessitarem per fer funcionar el *plugin*. Aquests arxius seran els necessaris que copiem en la carpeta del projecte. Normalment, els trobem a la carpeta **`/dist/`**.
4. **Enllaçar els arxius en el codi HTML**: és important que l'enllaç als arxius .js es facin després de l'enllaç a la llibreria de jQuery, però abans del nostre document .js on afegim el nostre codi. És important aclarir que no treballarem en els arxius del *plugin* de la mateixa forma que no treballem en l'arxiu del jQuery.
5. **Crear l'estructura HTML**: els plugins tenen una aplicació pràctica sobre un contingut del nostre document i hem de crear-lo per a veure els seus resultats.
6. **Inicialitzar el *plugin***: els *plugins* escrits per jQuery estan definits com a nous mètodes, per tant, cal llegir la documentació per veure quin nom tenen i veure quins arguments o opcions de configuració tenen.

OWLCAROUSEL

Per mostrar com podem emprar un *plugin*, explicarem el *plugin* d'OwlCarousel per jQuery, una llibreria per poder fer pasis de diapositives o carrusels dinàmics.

1. **Pàgina web de documentació**: https://owlcarousel2.github.io/OwlCarousel2/
2. **Descarregar els arxius del *plugin*:** https://github.com/OwlCarousel2/OwlCarousel2/archive/2.3.4.zip
3. **Localitzar els arxius necessaris:** al descomprimir trobem una carpeta `/dist/` amb els arxius `owl.carousel.js` i `owl.carousel.min.js`. Com que el segon és la versió minificada del primer, més lleugera, és la que enlaçarem al nostre projecte. A més, segons la documentació, de la carpeta `/dist/assets/` necessitarem l'arxiu `owl.carousel.min.css` pels estils bàsics i `owl.theme.default.min.css` o `owl.theme.green.min.css` per afegir els controladors predeterminats.
4. **Enllaçar els arxius:** en el nostre exemple, quedaria una cosa semblant a:

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

```
<!DOCTYPE html>
<html lang="ca">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Carrusel amb OwlCarousel</title>
    <link rel="stylesheet" href="OwlCarousel2-2.3.4/dist/assets/owl.carousel.min.css">
    <link rel="stylesheet" href="OwlCarousel2-2.3.4/dist/assets/owl.theme.default.min.css">
    <style>
        body { font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;}
    </style>
</head>
<body>
    <h1>Carrusel amb OwlCarousel</h1>

    <script src="jquery-3.7.1.min.js"></script>
    <script src="OwlCarousel2-2.3.4/dist/owl.carousel.min.js"></script>
    <script src="funcions.js"></script>
</body>
</html>
```

5. **Crear l'estructura HTML**: creem un contenidor amb una llista d'imatges per a mostrar-les en carrusel:

```
<!DOCTYPE html>
<html lang="ca">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Carrusel amb OwlCarousel</title>
    <link rel="stylesheet" href="OwlCarousel2-2.3.4/dist/assets/owl.carousel.min.css">
    <link rel="stylesheet" href="OwlCarousel2-2.3.4/dist/assets/owl.theme.default.min.css">
    <style>
        body { font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;}
    </style>
</head>
<body>
    <h1>Carrusel amb OwlCarousel</h1>
    <div id="galeria">
        <img src="https://picsum.photos/id/0/960/540" alt="foto 0" title="foto 0">
        <img src="https://picsum.photos/id/1/960/540" alt="foto 1" title="foto 1">
        <img src="https://picsum.photos/id/2/960/540" alt="foto 2" title="foto 2">
        <img src="https://picsum.photos/id/3/960/540" alt="foto 3" title="foto 3">
        <img src="https://picsum.photos/id/4/960/540" alt="foto 4" title="foto 4">
    </div>
    <script src="jquery-3.7.1.min.js"></script>
    <script src="OwlCarousel2-2.3.4/dist/owl.carousel.min.js"></script>
    <script src="funcions.js"></script>
</body>
</html>
```

6. **Inicialitzar el *plugin***: en aquesta llibreria ens cal fer dues coses; primer, afegir las classes css al contenidor de galeria:

```
    <div id="galeria" class="owl-carousel owl-theme">
        ...
    </div>
```

Segon, fer el selector del contenidor i aplicar el mètode del *plugin* en el nostre arxiu funcions.js:

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

```
$(function(){
    $('#galeria').owlCarousel({
        loop:true,
        margin:10,
        nav:true,
        responsive:{
            0:{
                items:1
            },
            600:{
                items:3
            },
            1000:{
                items:5
            }
        }
    });
});
```

A partir d'ara, depèn del autor de quins arguments podem afegir al mètode per configurar el seu funcionament. Totes aquestes opcions estaran disponibles en la documentació de la web.

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)