# 1. Selecció d'arquitectura i eines de programació

## 1.1 Que és un llenguatge script?

El llenguatge base d'una pàgina web és HTML, es tracta d'un llenguatge de marques que només indica la forma en que es mostrarà la pàgina web en el navegador del client i que no permet cap interacció entre el client i la pàgina.

Un llenguatge script és codi de programació que s'integra amb el codi HTML de la pròpia pàgina i li proporciona interacció, convertint-la en dinàmica.

## 1.2 Diferents tipus de scripts

En el desenvolupament Web generalment es parla de scripts del costat del servidor i scripts del costat del client.

**Els scripts del costat del servidor** s'executen en el servidor web. En el moment en que es sol·licita la pàgina, el servidor web executa el script i genera el resultat en forma d'una pàgina HTML enviant-la al navegador del client. Entre els llenguatges més populars d'aquests tipus tenim:

- PHP
- ASP/ASP.NET
- JSP
- Perl
- Python
- Ruby

**Els scripts del costat del client** s'executen dins del navegador web del client, és a dir, corren en la pròpia màquina del client, sense cap interacció amb el servidor. El més empleat és JavaScript, que està activat en més del 95% dels navegadors web. Un altre llenguatge script del costat del client és VBScript (Visual Basic Script) es tracta d'un llenguatge script "propietari" de Microsoft, que només esta implementat en els navegadors Internet Explorer, per tant s'hauria d'evitar utilitzar-lo, tret que estiguem programant per un entorn tancat o una intranet en la que estem totalment segurs que tots els clients fan servir Internet Explorer.

Degut a la seva amplia acceptació, nosaltres en aquest Mòdul treballarem amb JavaScript.

## 1.3 És JavaScript un llenguatge de programació?

Si, JavaScript és un llenguatge de programació, donat que permet al programador codificar algorismes, indicar una sèrie d'accions que es duran a terme i que són típiques de tots els llenguatges de programació (treballa amb dades i objectes, bucles, decisions, etc.), una de les coses que el diferencia de la resta de llenguatges tradicionals i fa que sigui un llenguatge script és que el seu àmbit d'actuació està limitat a la pàgina web en la que s'integra.

**JavaScript no és Java,** tot i que tots dos comparteixen pràcticament la mateixa sintaxi i tenen noms similars, cal esmentar que es tracta de dos llenguatges diferents, Java és un llenguatge complert , compilat i que es pot utilitzar per crear qualsevol tipus d'aplicacions. Per la seva banda JavaScript és un llenguatge interpretat i el seu àmbit d'aplicació és una pàgina web.

## 1.4 Models d'execució de codi

En el mon de la programació d'aplicacions informàtiques ens troben amb dos tipus de models d'execució de codi:

Programes Compilats
Programes Interpretats

**Els programes compilats** s'escriuen en un llenguatge d'alt nivell, més proper al llenguatge humà i posteriorment es compilen. Es a dir, les instruccions es tradueixen a codi màquina, que és el codi que entén l'ordinador i el que el Sistema Operatiu pot executar de manera autònoma. El resultat d'aquesta traducció genera el que coneixem com a programa o codi executable. Una vegada tenim el programa el podrem executar tantes vegades com vulguem de manera ràpida, doncs ja està traduït a codi màquina. Si una vegada en funcionament, ens adonem que hi ha errors, o volem afegir-hi altres característiques, caldrà que modifiquem el codi font (el codi en llenguatge d'alt nivell) i tornem a generar l'executable, tornant a compilar.

**Els programes interpretats**, també estan escrits en un llenguatge d'alt nivell, però a diferència del compilats, aquests no generen un programa executable, sinó que corren dins d'un entorn d'execució propi que va executant les instruccions una a una a mida que les va interpretant o traduint, es a dir, el codi font es converteix en codi executable a mida que es va traduint. Si necessitem fer qualsevol canvi, ho podem fer directament sense preocupar-nos per res més ja que quan es torni a posar en marxa el programa es tornarà a traduir/executar tot el codi de nou, incloent-hi els nostres canvis.

Com a contrapartida, tenim que un programa interpretat és més lent que un programa compilat, ja que cada vegada que el posem en marxa s'ha d'anar traduint instrucció a instrucció, a diferència del compilat on totes les instruccions ja estan traduïdes. A més el programa interpretat sempre ha de funcionar dins del seu entorn d'execució (el programa que s'encarrega d'executar-lo), no es pot executar directament pel S.O. Amb la potència d'execució dels ordinadors actuals, en la pràctica, aquesta diferència de velocitat d'execució és ínfima.

## 1.5 Mecanismes d'execució de codi en un navegador Web

JavaScript és un llenguatge interpretat, i tal com hem vist a l'apartat anterior tots els llenguatges interpretats requereixen un entorn d'execució, que s'encarregui de traduir i executar les instruccions, tenim que **l'entorn d'execució del llenguatge JavaScript és el navegador web**.

L'intèrpret del llenguatge està integrat dins del propi navegador, això vol dir que el codi s'executa quan el navegador carrega la pàgina en la memòria de l'ordinador. Quan el document es carrega, l'intèrpret s'encarrega de convertir tot el codi JavaScript que aparegui dins la pàgina en el seu corresponent codi binari adequat pel S.O. de la màquina en la que s'està treballant.

Per poder dur a terme tot aquest procés, el navegador esta constituït per una sèrie d'elements i components que conformen *l'arquitectura del navegador*. Tot i que cada fabricant fa servir la seva pròpia arquitectura en els seus navegadors, la majoria tenen uns components bàsics que són comuns. Com a referència podem trobar els que apareixen a la següent figura 1.1:

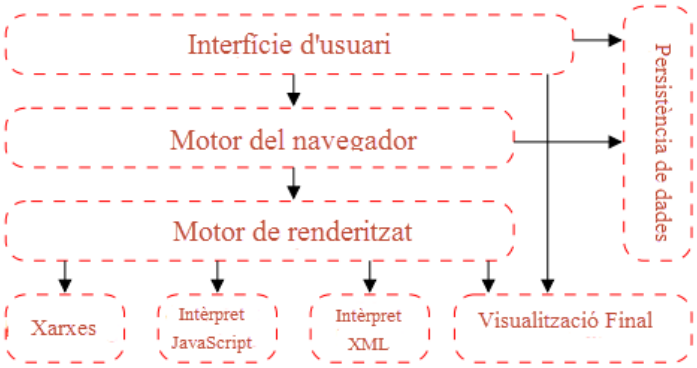



***Figura 1.1*** *Arquitectura de navegador*

**Interfície d'usuari**, El subsistema d'interfície d'usuari és la capa entre l'usuari i el motor del Navegador. Ofereix característiques com ara barres d'eines, visualització del progrés de càrrega de la pàgina, la gestió intel·ligent de descàrregues, les preferències de l'usuari, i la impressió. Es pot integrar amb l'entorn d'escriptori per a oferir gestió de sessió del navegador o la comunicació amb altres aplicacions d'escriptori

**Motor del navegador.** És un component integrable que proporciona una interfície d'alt nivell per al motor de renderitzat. Carrega una determinada URI i dóna suport a les accions primitives de navegació, com ara anar endavant, enrere o recarregar la pàgina. Ens proporciona enllaços per veure els diversos aspectes de la sessió de navegació, com ara el progrés actual de càrrega de pàgines i alertes de JavaScript. També permet la consulta i manipulació de la configuració del motor de renderitzat.

**Motor de renderitzat.** Produeix una representació visual per una URI. És capaç de mostrar documents HTML i XML, CSS, així com el contingut incrustat com ara les imatges. Calcula la mida exacta disseny de pàgina i usa algorismes per ajustar gradualment la posició dels elements dins de la pàgina. Aquest subsistema inclou també el analitzador de codi HTML.

**El subsistema de Xarxa.** Implementa els protocols de transferència de fitxers, com ara HTTP i FTP. Tradueix entre diferents conjunts de caràcters, i identifica de quins tipus de fitxer es tracta, si és text, àudio, vídeo, etc. (segons el tipus MIME –Multipurpose Internet Mail Extensions). També pot implementar una memòria cau per emmagatzemar i recuperat recursos accedits recentment.

**Intèrpret JavaScript.** Avalua el codi JavaScript (també conegut com ECMAScript), que pugui estar embegut en pàgines web. Certa funcionalitat de JavaScript, com ara l'obertura de finestres pop-up, pot ser inhabilitada pel motor del navegador o el motor de procés per raons de seguretat.

**Intèrpret XML.** Per accedir de manera més ràpida als elements HTML/XHTML del document, els navegadors incorporen un mòdul que permet carregar en memòria la representació del model d'objectes de la pàgina(DOM). Aquest és un dels subsistemes més reutilitzables en l'arquitectura accelera l'accés als diferents elements de la pàgina.

**Visualització final.** Proporciona funcions primitives de dibuix i posicionament a la finestra, un conjunt de "widgets" de la interfície d'usuari, i un conjunt de fonts per defecte. Pot estar molt lligat amb les llibreries de visualització del sistema operatiu.

**Persistència de dades.** Funciona com magatzem de les dades que necessitaran els diferents subsistemes del navegador, com ara les dades associades amb la sessió de navegació en el disc, la configuració de la barra d'eines, les galetes, els certificats de seguretat i la memòria cau.

El codi JavaScript pot aparèixer en qualsevol part dins el codi HTML, pot haver-hi més d'un bloc de codi dins la mateixa pàgina. L'ordre d'execució es correspon amb la seva posició dins de la pàgina, es a dir, s'executa a mida que es va trobant. Hi ha blocs de codi especials, funcions, subrutines i

manejadors d'esdeveniments, que només s'executen si es fa una crida expressa a ells, o quan es produeix un esdeveniment en concret. (Ho veurem al llarg del curs)

## 1.6 Capacitats i limitacions d'execució

Hem vist que l'àmbit d'execució dels programes JavaScript és el navegador del client, això ens dóna als programadors una gran potència, però també ens limita el camp d'actuació de manera significativa.

JavaScript és un llenguatge que millora l'experiència de navegació en el client, però tot el que fa referència al servidor queda fora del seu àmbit d'actuació.

Que l'àmbit d'execució de JavaScript sigui el navegador del client, no vol dir que el programador de JavaScript tingui "barra lliure" per fer el que vulgui dins del navegador o de l'ordinador de l'usuari. Hi ha accions que per motius de seguretat estan totalment prohibides i d'altres altament restringides.

Limitacions de JavaScript:

La més important: **No pot accedir al sistema d'arxius del client**. De manera que, no pot escriure, llegir ni alterar cap informació en el ordinador de l'usuari. Això és molt important, si no fos així qualsevol programador malintencionat podria provocar veritables maldecaps, en forma de virus, esborrant arxius crítics del S.O. omplint el disc amb fitxers brossa, etc. (Una petita excepció a aquesta limitació, que veurem al llarg del curs, són les galetes o cookies).
Com a conseqüència del punt anterior, JavaScript tampoc pot recórrer el sistema d'arxius per esbrinar quins programes tenim instal·lats, ni manipular arxius o documents privats que resideixin a l'ordinador del client. Només pot saber el tipus de navegador en que s'està executant i el sistema operatiu que es fa servir.
No té accés a la llista de contrasenyes emmagatzemades al navegador.
Tampoc no te accés a dades personals de l'usuari.
No pot modificar el contingut d'un lloc extern
No pot accedir a pàgines d'un domini diferent al seu.
No pot instal·lar automàticament programes executables.

Per contra és ideal per facilitar la interacció amb l'usuari en tasques que no requereixin comunicació amb el servidor i que solament amb HTML no seria possible realitzar. Així pot:

Mostrar alertes
Escriure missatges a la barra d'estat del navegador
Obrir noves finestres del navegador carregant en elles altres documents.
Validar la informació que han escrit en els formularis i assegurar-se que és correcta abans d'enviar-la al servidor, d'aquesta manera guanyem temps a l'estalviar-nos un viatge d'anada i tornada al servidor si la informació és errònia.
Pot realitzar càlculs.
Controlar els esdeveniments produïts pel ratolí i el teclat.
Mostrar missatges quan el cursor passi per sobre d'un objecte a la pantalla.
Treballar amb l'hora i la data del sistema.
Identificar el navegador, el sistema operatiu i els "plug-ins" que tenim instal·lats (com ara flash).
Accedir a les propietats CSS de qualsevol element de la pàgina i canviar-les de manera interactiva en resposta a alguna acció de l'usuari. Per exemple fent que es mostrin o desapareguin elements DIV, imatges; canviar el color de paràgrafs, etc.

## 1.7 Integració del codi amb les etiquetes HTML

Generalment, el codi JavaScript s'escriu directament dins del document HTML, delimitat per les etiquetes <SCRIPT> i </SCRIPT> .

Per escriure codi JavaScript, només necessitem un editor de text senzill, que generi "text pla" sense caràcters de format, com ara el bloc de notes de Windows. Una vegada que tenim el script integrat dins una pàgina HTML, només caldrà carregar la pàgina en el nostre navegador i comprovar que el codi funciona correctament.

Una altra manera d'escriure codi és desar-ho de manera separada dins un fitxer extern a la pàgina, amb l'extensió **.js**. Posteriorment podem cridar-lo o fer una referència a ell des d'una o més pàgines en les que vulguem fer-lo servir. Aquest mètode té l'avantatge que el codi es pot reutilitzar per vàries pàgines; només cal que es faci una referència o s'enllaci amb la pàgina..

```
<HTML>
<HEAD>
   <TITLE>Exemple</TITLE>
   <SCRIPT type="text/javascript" src="funcions.js"></SCRIPT>
</HEAD>
<BODY>
      ...
</BODY>
</HTML>
```

Amb el modificador **src,** li estem indicant que el codi font JavaScript està en un fitxer extern **funcions.js.** A tots els efectes això equival a haver escrit el codi directament dins la pàgina.

El fitxer **funcions.js** és un fitxer de text on hem escrit les rutines de JavaScript. No ha de contenir les etiquetes ***<SCRIPT>*** ni ***</SCRIPT>,*** ja que si ens fixem a la nostra pàgina ja en surten.

Aquesta segona forma de treballar, també té avantatges, si hem de modificar qualsevol rutina. Com que tot el codi està centralitzat en un únic fitxer, només haurem de canviar-lo en un únic lloc amb la qual cosa quedarà modificat per totes les pàgines que facin referència a aquest codi sense haver de canviar-les una a una.

Dins un document web podem tenir més d'un bloc de codi distribuït en la pàgina, a banda de codi en fitxers externs. Així doncs, en quin ordre s'executen els diferents blocs de codi?

El primer que s'executarà seran els blocs ***<SCRIPT>*** que estiguin inclosos dins de la secció <HEAD> de la pàgina, tant si inclouen el codi directament, com si fan referència a un fitxer extern de JavaScript. Aquesta secció és el lloc ideal per col·locar les funcions que es cridaran posteriorment des del codi.

A continuació s'executen les ordres incloses dins de la secció <BODY> a mida que es va generant la pàgina en el navegador. Si tenim més d'un bloc de codi dins d'aquesta secció, s'executa en l'ordre en que va apareixent a la pàgina.

Finalment, entren en lloc les funcions i/o els manejadors d'esdeveniments, aquestes es produeixen a petició del codi inclòs en els altres blocs, o com a resposta a qualsevol esdeveniment que s'hagi produït (pitjar una tecla, clicar el ratolí, desplaçar el ratolí, etc.).

Passos bàsics per crear un programa JavaScript.

> Creem una pàgina HTML, fent servir un editor senzill que generi fitxers de text en ASCII pur, com ara el bloc de notes que ja hem dit. També podem fer servir una pàgina web ja creada i continuar amb el pas següent.
> Inserim dins de la pàgina web les etiquetes <SCRIPT> i </SCRIPT> en el lloc on hagi d'anar el codi JavaScript.
> Escrivim el codi JavaScript dins d'aquests etiquetes.
> Desem el programa, tenint especial cura que l'extensió del mateix sigui **.html** o **.htm**
> Per executar el programa només caldrà obrir el nostre arxiu amb el navegador.

## 1.8 Eines de programació sobre clients web. Tecnologies i llenguatges Associats.

Ja hem vist que JavaScript es pot codificar en qualsevol editor que generi text pur, com ara amb el bloc de notes. Però, fer servir un editor una mica més avançat, que reconegui la sintaxi, ens pot fer la programació més fàcil. Un editor especialitzat, ens detectarà les diferents instruccions, sentències i funcions, assignant-li un color diferent, així com un color específic pels comentaris, les variables, les cadenes de text... També ens ajudarà a identificar on comencen i on acaben els blocs de codi. Informarà quan ens falti o sobri un parèntesi, una clau d'obertura o de tancament, etc.

Gairebé tots els editors enfocats a programació inclouen suport pel llenguatge JavaScript. Una altra opció que pot ser molt útil és la integració de l'editor amb el navegador. Recordem que JavaScript no necessita d'un compilador, sinó que és el propi navegador qui s'encarrega d'anar interpretant les ordres a mida que es va generant la pàgina.

Alguns editors que ens poden ser útils són Notepad++, Komodo, Textpad, Editplus...

Pel que fa als complements dels navegadors, són mòduls afegits orientats al desenvolupament que podem incorporar als navegadors, com ara Mozilla Firefox i que dóna la possibilitat d'incloure moltes extensions segons les necessitats de l'usuari. Nosaltres ens fixarem en les que ajuden al desenvolupament web.

A l'adreça https://addons.mozilla.org/es/firefox/ trobarem una gran quantitat d'extensions.

Potser un dels complements més interessant per treballar amb JavaScript des de Firefox sigui **Firebug** que el podrem descarregar des de l'adreça anterior o també des de http://getfirebug.com/, una vegada ens situem en aquesta pàgina fent servir el navegador Firefox, només caldrà clicar en l'enllaç que diu "Install Firebug", observeu la figura 1.2.

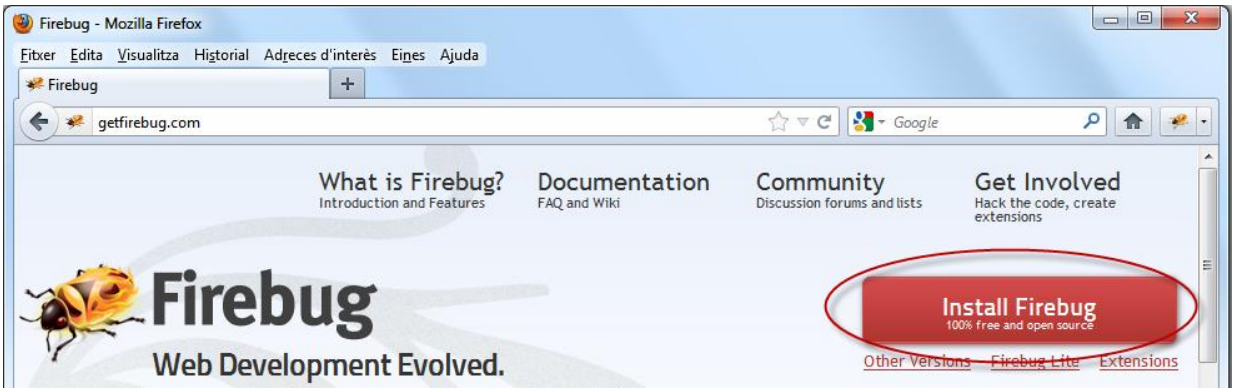



***Figura 1.2.*** *Pàgina web de Firebug*

Firebug executa i permet treballar amb el codi JavaScript que aparegui dins la pàgina que tinguem carregada en un moment determinat en Firefox. Com que JavaScript requereix una pàgina HTML per funcionar, podem utilitzar una pàgina buida, en la que incloure el codi per anar fent proves:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html lang="es">
<head>
    <title>Pàgina de proves</title>
</head>
<body>

</body>
</html>
```

Una vegada instal·lat el complement en Firefox, haurem de carregar la pàgina de proves i pitjar la tecla F12 per posar en marxa Firebug. Veurem que la pantalla es divideix en dos, la part inferior és

la part de Firebug. És probable que la primera vegada informi que no està activat el panell de script, l'haurem d'activar tal com s'indica en el mateix programa (figura 1.3):

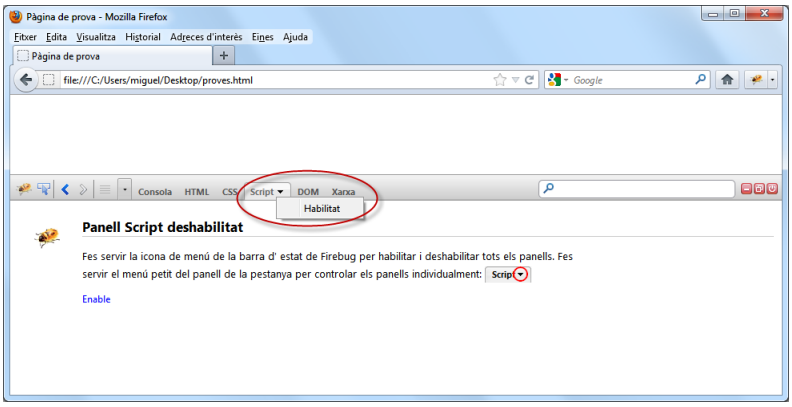



***Figura 1.3.*** *Activar JavaScript en Firebug*

En la modalitat Consola, Firebug inclou una línia de comandes que ens permet executar instruccions JavaScript de manera directa; apareix al fons de la pàgina i està precedida per **>>>**. Podem fer una prova escrivint en ella ***alert("Hola Manola")*** i tot seguit clicar Enter.

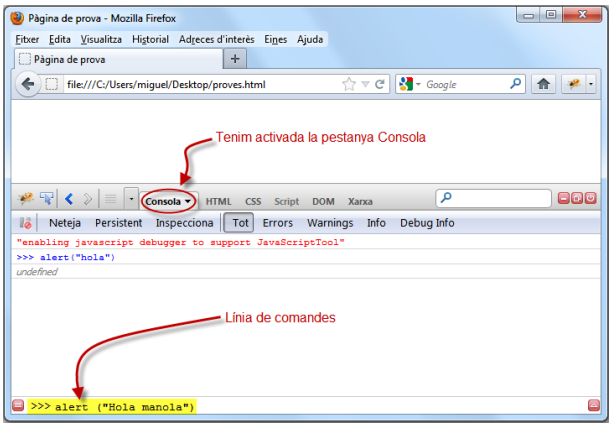



***Figura 1.4.*** *Línia de comandes*

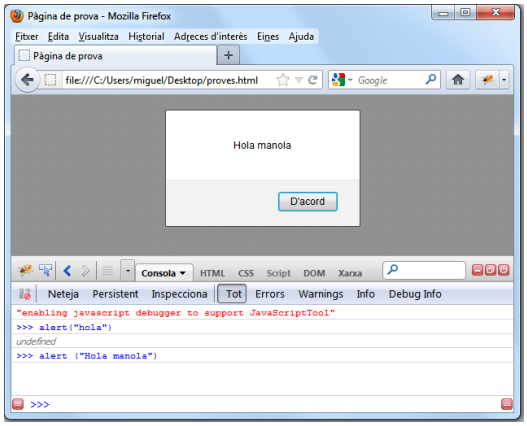



***Figura 1.5.*** *Resultat de l'execució*

Com que la majoria de programes en JavaScript estan formats per més d'una línia de codi, la línia de comandes de Firebug no ens serà de gran utilitat, així doncs, haurem e fer servir l'editor de codi en comptes de la línia de comandes, per activar-lo haurem de clicar la petita fletxa que apunta cap amunt a la part inferior dreta del programa (Figura 1.6).

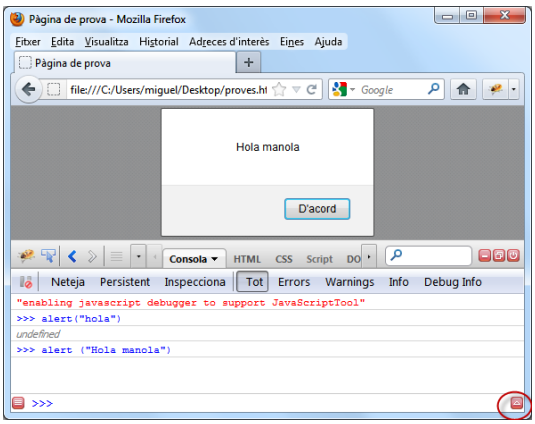



***Figura 1.6.*** *Activar editor comandes*

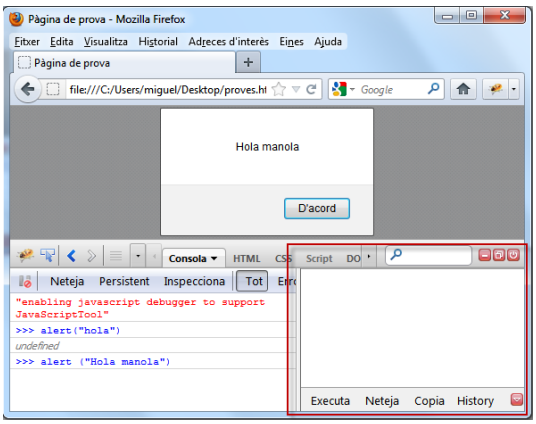



***Figura 1.7.*** *L'editor de comandes*

Apareix un requadre, que és on haurem d'escriure el nostre codi, fixeu-vos que inclou quatre ordres: Executa, Neteja, Còpia i History (figura 1.7). Quan acaben d'escriure el codi i vulguem provar com funciona clicarem a sobre de ***Executa*** (També podem fer servir la combinació de tecles ***Ctrl+Enter*** )

Si ara escrivim el mateix codi d'abans i cliquem executar, veurem que el resultat que es produeix és el mateix, la diferència és que amb l'editor poden escriure rutines de més d'una línia de codi.

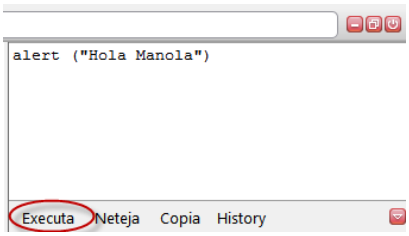



***Figura 1.8.*** *Escriure codi*

Quan programem en JavaScript directament, sense fer servir cap entorn de programació específic, una de les dificultats amb la que ens troben és que, si cometem qualsevol error en el codi, els navegadors no informen, o ho fan d'una manera tan escarida que fins i tot pot passar desapercebuda. Un altre avantatge de fer servir Firebug és que davant un error ens informa adequadament del mateix i del lloc on s'ha produït. Mireu la figura 1.9, observeu que tenim activada la pestanya **errors**.

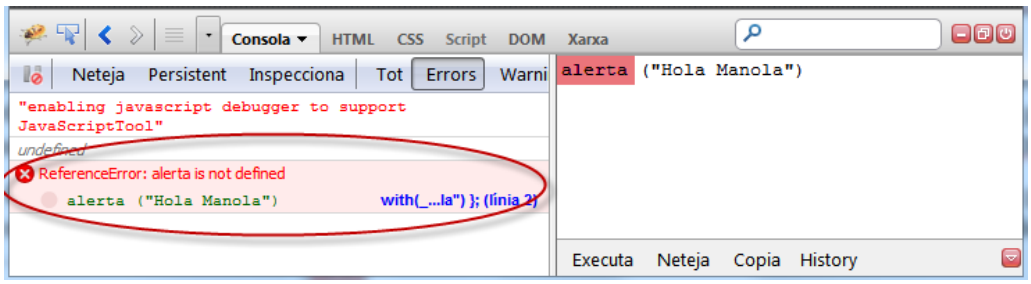



***Figura 1.9.*** *Errors en Firebug*

**Web Developer**, es tracta d'un altre afegit per Firefox que també facilita la feina als programadors de JavaScript. El podrem descarregar i instal·lar des de l'adreça de descàrregues d'extensions per Firefox, dins de l'apartat ***"Desarrollo Web"***. Observeu la figura 1.10.

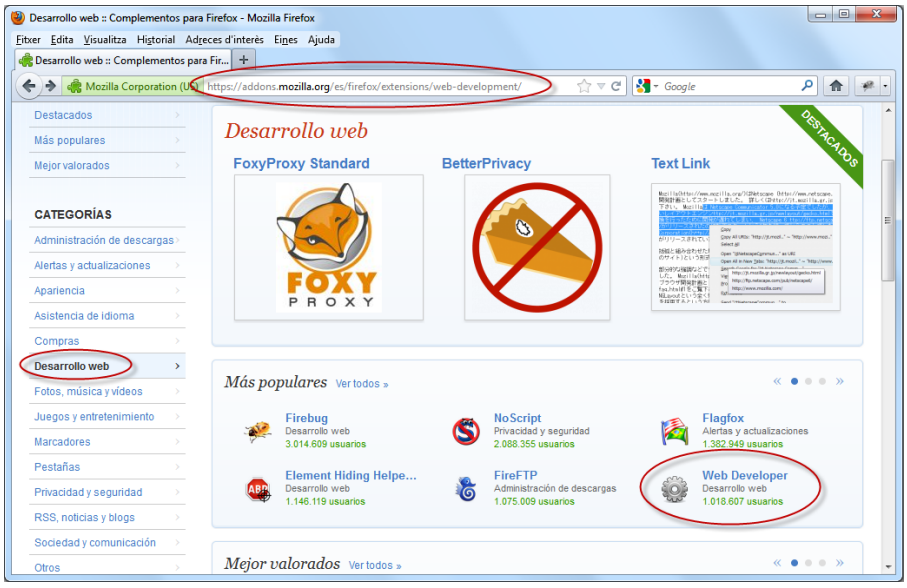



***Figura 1.10.*** *Panel de descàrregues de complements de desenvolupament web per Firefox.*

Una vegada instal·lada es presenta en forma d'una nova barra d'eines al navegador (figura 1.11), que ens proporciona una sèrie d'opcions en uns menús desplegables agrupats segons la seva funcionalitat.

Us recomanem que carregueu qualsevol pàgina, pot ser una pàgina vostra o navegueu a qualsevol lloc d'Internet, i que aneu provant totes les opcions que ofereix aquesta extensió, trobareu que n'hi ha de molt interessants.


***Figura 1.11***. *Nova barra d'eines de Web Developer en Firefox.*

Una altra opció bastant interessant que tenim a la nostra disposició i que treballa de manera semblant a Firebug, és fer servir el navegador Chrome de Google, i fer aparèixer la consola JavaScript tal com mostra la figura següent (figura 1.12).

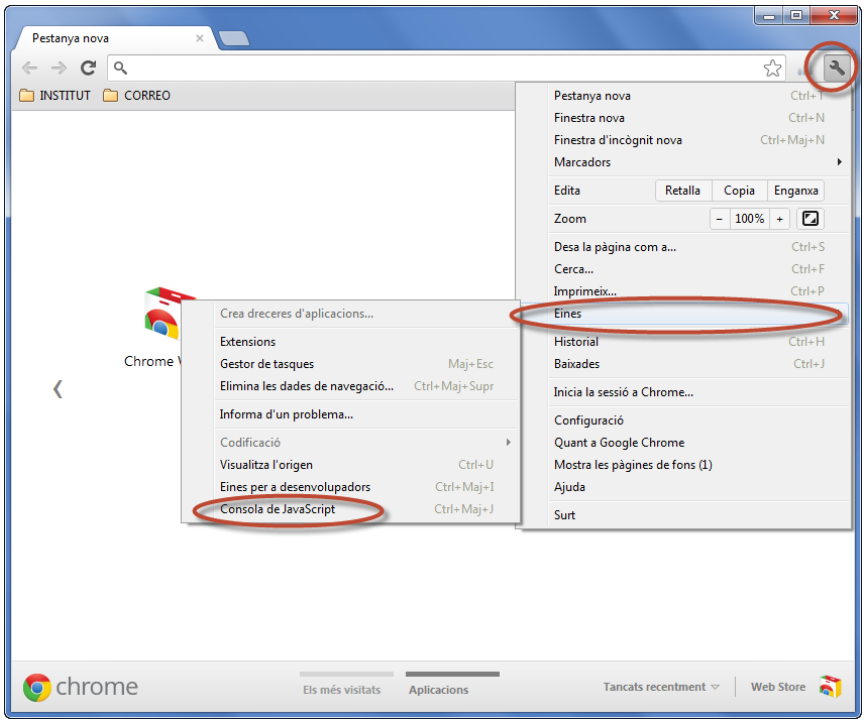



***Figura 1.12.*** *Con fer que aparegui la consola de JavaScript en Chrome.*

Veureu que la consola apareix a la part inferior de Chrome.

# 2. APLICACIÓ I VERIFICACIÓ DE LA SINTAXI DEL LLENGUATGE

## 2.1 Selecció del llenguatge de programació de clients web

Anteriorment ja hem vist alguns dels llenguatges de programació que tenim a la nostra disposició per programar en l'entorn client. També hem pogut comprovar que d'ells el més utilitzat en tots els nivells és JavaScript, és per això que al llarg de tot aquest curs farem servir **JavaScript**.

### 2.1.1 Sintaxis general de JavaScript

Quan parlem de la sintaxi d'un llenguatge de programació ens referim a aspectes com la forma en que es poden definir comentaris, blocs de codi, funcions, com es defineixen les variables, la forma en que se separen unes instruccions d'altres, etc. En aquest aspecte, JavaScript té una sintaxis molt semblant a la de Java o a la de C++.

Algunes de les característiques de sintaxi de JavaScript són:

**Majúscules i minúscules**: El llenguatge fa distinció entre majúscules i minúscules, això pot representar un problema per programadors que estan acostumats a altres llenguatges que no fan aquesta distinció, com per exemple programadors de Visual Basic .Net, o fins i tot el propi HTML que no fa distinció entre elles. Per tant hem de prestar molta atenció a aquesta regla que pot donar origen a molts errors en els nostres programes. Per exemple:

```
Alert ("Hola Manola")
alert ("Hola Manola")
```

a la primera instrucció hi ha un error, ja que la sintaxi correcta de la instrucció ***alert*** indica que s'ha d'escriure tota en minúscula i nosaltres la hem escrit començant per majúscula.

**El punt i coma, espais, tabulacions i salts de línia.** Tot i que no és obligatori, JavaScript separa les instruccions amb el signe punt i coma (*;*) igual que ho fan C++ o Java. Malgrat això, nosaltres podem escriure una instrucció per línia i ometre el punt i coma si volem. Per exemple, podem escriure:

```
total = 23
suma = 18
```

fixeu-vos que no n'hi ha punt i coma. Si volguéssim escriure les instruccions en una sola línia si és obligatori separar-les amb punt i coma

```
total = 23; suma = 18;
```

D'altra banda, també podem escriure una instrucció per línia si volem i fer que acabi en punt i coma:

```
total = 23;
suma = 18;
```

JavaScript ignora els espais i les tabulacions que apareguin dins del codi, així tenim que és el mateix escriure ***total = 23,*** que ***total=23*** pel que respecta als tabuladors, com també són ignorats, ens podem servir de gran utilitat per indentar el codi en programes extensos amb la finalitat de fer-los més clars i millorar la seva llegibilitat.

## 2.2 Variables

Es tracta d'un concepte de programació que és de sobres conegut per tots els programadors sigui qui sigui el llenguatge en que programin. Una variable és una zona de la memòria identificada per un nom on es guarda un valor o una dada que podrem anar canviant al llarg del transcurs del programa.

### 2.2.1 Declaració de variables

Per poder fer servir una variable, abans hem de declarar-la, o sigui hem informar a l'intèrpret del navegador de les nostres intencions perquè ens reservi un espai en la memòria per

emmagatzemar el valor que guardarà la variable. En JavaScript podem definir una variable de dues formes: de manera implícita o de manera explicita.

**Declaració explícita**. La forma de fer-ho és fent servir la paraula reservada **var** seguida del nom que volem donar a la variable:

```
var total;
```

També podem declarar més d'una variable en una mateixa línia, separant-les per comes:

```
var total, nom;
```

A la instrucció anterior hem reservat una posició en la memòria i la hem identificat amb el nom ***total***, i una altra com a ***nom,*** però que encara no tenen assignat cap valor. Per utilitzar una variable dins del nostre codi, abans haurem d'assignar-li un valor inicial, això ho fem utilitzant l'operador d'assignació (**=**), observeu l'exemple:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
     "http://www.w3.org/TR/html4/strict.dtd">
<HTML>
<HEAD>
<TITLE>Exemple</TITLE>
</HEAD>
<BODY>
<SCRIPT type="text/javascript">
<!--
            var total, nom;
            total=1200;
            nom="Joan";
            alert (nom);
            alert (total);
       //-->
</SCRIPT>
</BODY>
</HTML>
```

Fixeu-vos en la part en negreta del codi anterior. En primer lloc hem definit dues variables (***total*** i ***nom***), tot seguit li hem assignat un valor i posteriorment hem fet servir la instrucció ***alert*** per mostrar en una finestra d'avís el contingut de la variable nom i una altra per total. No ens hem de preocupar si el valor de la variable en un cas està entre cometes i en l'altre no, veurem de seguida la seva explicació. Observeu també que hem escrit ***alert (nom)*** sense cometes, en comptes de ***alert("nom")***. D'aquesta manera el navegador sap que ens referim a la variable ***nom*** i més concretament que volem recuperar el seu contingut. Si l'haguéssim escrit entre cometes, entendria que volem que mostri la paraula nom.

*Proveu l'exercici anterior i comproveu també el que s'ha explicat sobre alert, executant el codi amb alert(nom) i amb alert("nom").*

Una forma d'estalviar temps i que generalment fem servir tots els programadors consisteix en assignar valor a una variable en el mateix moment en que es declara, així l'exemple anterior podria haver quedat:

```
<SCRIPT type="text/javascript">
<!--
            var total=1200, nom="Joan";
            alert (nom);
            alert (total);
       //-->
</SCRIPT>
```

Observeu a la línia en negreta com hem aprofitat la mateixa instrucció per declarar les variables i a la vegada assignar-li valor. Si proveu aquest programa veureu que el seu funcionament no canvia en res respecte a la versió anterior.

**Declaració implícita**. A diferència de la majoria de llenguatges de programació, JavaScript ens permet utilitzar una variable sense haver-la declarat prèviament. Quan l'intèrpret es troba amb l'assignació d'un valor a una variable, em primer lloc mira si aquesta variable ja està definida i conté un valor previ, si és així el que fa es canviar al valor que conté pel nou que li estem assignant; si la variable no està definida entén que volem crear una variable nova a la que assignar-li el valor. Mireu l'exemple:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
     "http://www.w3.org/TR/html4/strict.dtd">
<HTML>
<HEAD>
<TITLE>Exemple</TITLE>
</HEAD>
<BODY>
<SCRIPT type="text/javascript">
<!--
            var total=1200;
            alert (total);
            nom="Manola";
            alert (nom);
            total=0;
            alert (total);
       //-->
</SCRIPT>
</BODY>
</HTML>
```

Repassem-lo: primer amb la instrucció ***var total=1200;*** hem declarat la variable ***total*** de manera **explícita** i li he assignat el valor ***1200***. A la instrucció ***nom="Manola"*** l'ordinador entén en primera instància, que volem assignar-li la paraula ***"Manola"*** a una variable ***nom,*** creada prèviament, però com que no la troba, la crea en aquest precís moment de manera **implícita** i li assigna el valor. Finalment la instrucció ***total=0;*** sembla ser un cas com ara l'anterior, la diferència està en que ara la variable ***total*** ja existeix i conté el valor ***1200***, per tant en comptes de crear-la de manera implícita, com que ja existeix, el que fa és canviar-li el seu valor, que passarà de 1200 a zero.

Pot semblar que treballar els nostres programes seguint la tècnica de la declaració implícita sigui la manera més còmoda pel programador, ja que no cal preocupar-se en definir les variables, doncs a mida que les necessitem ens les inventem, li assignem valor i es creen automàticament. Bé això és un plantejament erroni i de fet és una vulnerabilitat o una debilitat del llenguatge JavaScript. La majoria de llenguatges d'alt nivell ens obliguem a declarar explícitament les variables, de manera que si al nostre codi fem servir una variable que no ha estat declarada prèviament es generi un error, d'aquesta manera ens evitem maldecaps i errors lògics de programació que costen molt de detectar. Observeu el següent exemple:

```
<SCRIPT type="text/javascript">
<!--
            var preu = 1200;
            //... més linies de codi
            preus = 1300;
       //-->
</SCRIPT>
```

Hem declarat una variable ***preu*** i li hem assignat el valor ***1200***, i unes línies més avall, volíem canviar-li el valor perquè passi a ser ***1300***, però tal com ha passat a l'exemple, ens equivoquem al escriure la variable i l'anomenem ***preus.*** JavaScript, cerca una variable ***preus*** per assignar-li el valor ***1300***, però com que no la troba, entén que volem crear una de nova amb aquest nom i assignar-li el valor 1300. Així, ara tenim una variable ***preu*** amb el valor 1200 i una altra ***preus*** amb el valor 1300. Aparentment no s'ha produït cap error (de fet per JavaScript no n'hi ha cap error) però hi ha un error lògic de programació que ens pot donar molts maldecaps i errors de càlcul, quan posteriorment, utilitzem la variable ***preu*** pensant que el seu valor està actualitzat a ***1300***, però realment no ha estat modificat mai i continua emmagatzemant el valor ***1200***.

Si fessin aquest codi en Java o C++, el compilador ens hauria mostrar un missatge d'error dient que la variable ***preus*** no està definida, la qual cosa ens permet adonar-nos de l'error.

### 2.2.2 El nom de les variables

Ja hem vist que una variable és una posició de memòria a la que fem referència per un nom, aquest nom ens ho podem inventar nosaltres, els programadors, però s'ha de seguir unes certes regles:

El nom ha de començar per una lletra.
Després, només pot contenir combinacions de lletres, números i el guió baix.
No pot contenir espais en blanc, lletres accentuades ni caràcters específics de l'alfabet local. Es a dir només podem fer servir lletres que siguin les estàndards de l'alfabet anglès. (No feu servir ç, ñ, lletres accentuades ni cap caràcter especial)
No podem fer servir com a nom d'una variable cap paraula reservada de JavaScript
JavaScript diferència entre majúscules i minúscules, per tant hem de prestar atenció, ja que per exemple: les variables ***Total*** i ***total*** són dues variables distintes. Cal evitar utilitzar variables que només es diferencien entre si per l'ús diferenciat de les majúscules o minúscules.
Per raons de claredat, és recomanable que el nom de les variables es correspongui d'alguna manera amb el valor que emmagatzemen, evitant noms de variables curts que no aporten res a la claredat del codi, per la mateixa raó, tampoc no cal que el nom d'una variable sigui massa llarg. Potser una excepció a aquesta regla són les variables utilitzades com a índex per bucles o matrius, on s'acostuma a fer servir una sola lletra.
Tot i que cada programador pot fer servir les regles que més li agradin a l'hora de nomenar variables, una costum bastant estesa entre els programadors de JavaScript és escriure totes les variables amb minúscula, i si el nom de la variable està format més d'una paraules, escriure la primera lletra de les següents paraules en majúscula (evidentment sense cap espai), exemple:

```
var totalIva;
var preuAbansImpostos;
```

### 2.2.3 Àmbit d'una variable. Variables locals i globals.

En JavaScript les variables generalment no existeixen durant tota la vida de la pàgina, sinó que es creen i destrueixen sobre la marxa. El principi seguit és que una variable és local al bloc de codi on s'ha declarat, només existeix dins del context on s'ha declarat.

Per exemple, si la variable s'ha declarat dins d'una funció, només serà visible dins d'aquesta. Des de fora la funció la variable no existirà, de fet quan la funció acaba la variable es destrueix i s'allibera l'espai que ocupava.

Es considera que un bloc de codi és qualsevol bloc delimitat per clau { i }, així tenim que aquest comportament també afecta les variables declarades al cos del bucles i altres estructures (if, else, switch...).

A vegades, ens interessarà que una variable sigui global, es a dir, que no es destrueixi, que mantingui el seu valor al llarg de tota la vida de la pàgina i que sigui accessible a tots els procediments. Per aconseguir-ho, haurem de declarar-la fora de qualsevol funció o bloc de codi, d'aquesta manera d'intèrpret la identifica com a una variable global.

## 2.3 Tipus de dades

JavaScript és un llenguatge no tipificat, a diferència de la majoria de llenguatges, en JavaScript quan definim una variable no hem d'indicar el tipus de dades que emmagatzemarà. És més podem fer que al llarg del programa una variable emmagatzemi un valor numèric, per exemple, i una

mica més endavant canviar-ho per un valor de tipus cadena o qualsevol altre. Les variables s'adapten al valor que li assignem en cada moment.

No obstant això, els tipus bàsics amb els que treballa JavaScript són quatre:

**Indefinit**. Es tracta del valor que conté una variable que hem declarat, però a la que encara no li hem assignat valor. No l'hem de confondre amb la utilització d'una variable no declarada. La utilització en una expressió d'una variable no declarada provoca un error.
**Boolean**. Es tracta d'un tipus que només pot contenir dos possibles valor: ***true*** o ***false*** (vertader o fals)
**Numèrics**. Engloba totes les variables que contenen qualsevol valor numèric, tan enters com decimals, positiu o negatius. Un valor numèric pot intervenir en operacions aritmètiques.
**Cadena o string**. Es tracta de totes les variables que contenen cadenes o tires caràcters, es delimiten amb cometes ( " ), poden contenir qualsevol caràcter representable per l'ordinador i ho fem servir per emmagatzemar frases, paraules, lletres i text en general. A vegades els programadors novells confonen el tipus cadena quan guarda números amb el tipus numèric, per exemple:

```
var total = "100";
var total2 = 100;
```

El primer cas, tot i que sembla que estem desant el número 100, no és així realment, sinó que estem desant la cadena 100, es a dir la grafia de l'1 seguida de la del zero i de la del zero; l'ordinador el tracta com a text. En el segon cas, si que estem fent servir un valor numèric que podrem fer servir en operacions matemàtiques.

Quan fem servir una variable de tipus string dins una operació aritmètica, JavaScript intentarà convertir-la a numèrica i fer l'operació. Si es pot, no es produeix cap problema i realitza l'operació, si no pot, el resultat de l'operació és una valor especial ***NaN*** (Not a number).

A l'exemple anterior, JavaScript no tindria problema per convertir, la variable string ***"100"*** en numèrica, però que hagués passat si en comptes de ***var total="100",*** haguéssim tingut ***var total="cent"*** ? En aquest cas a l'intèrpret li és totalment impossible fer la conversió i es produiria un error.

Tot i que JavaScript intenta resoldre aquests problemes amb conversions automàtiques, no hauríem de confiar-nos i fer servir les variables de tipus adequat a cada operació o fer les conversions oportunes amb instruccions del llenguatge específicament dissenyades per convertir variables d'un tipus a un altre (les veurem més endavant).

Hi ha una instrucció que ens permet saber el tipus de dada que es guarda en cada moment en una variable, es tracta de **typeof()**.

 *Proveu el següent exemple*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
     "http://www.w3.org/TR/html4/strict.dtd">
<HTML>
<HEAD>
<TITLE>Exemple</TITLE>
</HEAD>
<BODY>
<SCRIPT type="text/javascript">
<!--
   var v1;
   alert (typeof(v1));
   var v2="Hola";
   alert (typeof(v2));
   var v3=55;
   alert (typeof(v3));
   var v4=true;
   alert (typeof(v4));
```

```
        //-->
</SCRIPT>
</BODY>
</HTML>
```

Observeu com sucessivament es van obrin finestres d'alerta mostrant el tipus de dada que hi ha a cada variable: undefined, string, numèric i boolean.

Suposem que hem estat fent servir una variable al nostre programa i volem deixar-la totalment buida, es a dir que no contingui res, ni un zero, ni un espai en blanc ni una cadena buida, absolutament res. Ho aconseguirem assignant-li a la variable el valor ***null*** (nul).

```
<SCRIPT type="text/javascript">
<!--
   var v1=5;
   alert ("v1 = " + v1 + " és del tipus " + typeof(v1));
   v1=null;
   alert ("v1 = " + v1 + " és del tipus " + typeof(v1));

//-->
</SCRIPT>
```

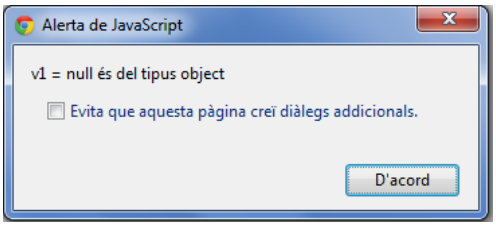



***Figura 2.1.*** *Treballant amb null*

Fixeu-vos en la sortida (figura 2.1), la variable v2 continua existint, però conté un valor ***null*** i el navegador ni tan sols sap de quin tipus de variable es tracta, per això li assigna el tipus genèric ***object***.

Hi ha altres tipus de dades com ara les destinades a manipular dates i hores, o els objectes, que veurem més endavant.

## 2.4 Assignacions

Ja hem vist en l'apartat de declaració d'una variable, que per assignar-li valor bàsicament s'ha d'escriure el nom de la variable, un signe igual i el valor que li volem assignar:

```
total=100;
```

El valor que assignem pot ser una valor directe (com a l'exemple anterior) o el resultat de qualsevol expressió que retorni un valor. JavaScript quan es troba una assignació, primer calcula el que hi ha a la dreta de l'igual i després assigna el resultat a la variable de l'esquerra.

```
preu = 100;
iva = 18;
total = preu +preu * iva / 100
```

Observeu que a la variable ***total*** se li assigna **el resultat de l'operació** que hi ha a la dreta de l'igual que és 118. Una variable sempre emmagatzema un valor.

Segons el tipus de valor que vulguem assignar, l'haurem de delimitar d'una manera o una altra, així tenim que:

**Valors numèrics:** no es delimiten amb res, simplement s'escriu el valor directament. Exemples:

```
var total = 2500; negatiu = -4;
var pi = 3.1415; dada = 12E5;
```

Fixeu-vos que, el valor s'escriu tal qual: no es posen separadors de milers, els números negatius s'escriuen amb el signe – al davant, als positius no cal ficar-li el signe +; els números decimals també s'escriuen sense cap delimitador, però cal tenir en compte que la coma decimal s'expressa com a un punt decimal.

També podem expressar les dades numèriques en notació exponencial, com es pot veure a l'exemple anterior ***dada=12E5***, estem assignant el valor 1200000 es a dir: 12 multiplicat per 10 elevat a 5. *En aquest cas, i sense que serveixi de precedent, JavaScript ens permet que la lletra E pugui estar en majúscula o minúscula,* ***dada=12e5*** també és correcte.

Les dades numèriques, gairebé sempre les escriurem dins del nostre programa en base decimal o base 10, que és com treballem habitualment, però JavaScript també ens permet fer servir altres bases de numeració com ara la base hexadecimal o l'octal.

Per d'indicar que el valor està en base hexadecimal farem que comenci per 0x o 0X (zero seguit de la lletra X en majúscula o en minúscula), recordeu que la base hexadecimal està formada per 16 dígits diferents que es corresponen amb tots els dígits numèrics del 0 al 9 més les lletres de la A a la F (en majúscules o en minúscules). Exemple:

```
var valorHex = 0xFF;
alert (valorHex);
```

si executem aquestes línies veurem que ens apareix una caixa d'alerta amb el valor 255 que és el valor decimal que correspon a FF en hexadecimal. Independentment de com hem assignat un valor numèric a una variable, JavaScript sempre el mostrarà en format decimal.

Per la seva banda els valors octals van precedits sempre del número 0 (zero), d'aquesta manera indiquem que el valor està expressat en octal. Exemple:

```
var valorOctal = 077;
alert (valorOctal);
```

en aquest cas, quan l'executem veurem que mostra el valor 63 que és el valor decimal que correspon amb l'octal 77. Recordeu que els números octals estan expressats en base 8, i estan formats només pels dígits numèrics compresos entre 0 i 7.

Aneu amb compte amb la dita que diu "*vals menys que un zero a l'esquerra*", en JavaScript un zero a l'esquerra indica que es tracta d'un número octal, per tant us podríeu endur més d'una sorpresa.

**Cadenes o strings**, es poden delimitar amb cometes dobles o cometes simples, per exemple: **nom="Joan"**; o bé **nom='Joan'**; Totes dues tenen el mateix comportament. Aquesta possibilitat de poder fer servir les cometes dobles o simples ens pot ser de molta utilitat. Per exemple, imagineu que volem assignar a la variable **bandoler** la cadena: ***Joan Sala i Ferrer, alies "Serrallonga" famós bandoler català***, potser el primer intent seria fer:

```
var bandoler="Joan Sala i Ferrer, alies "Serrallonga" famós bandoler català";
```

Però això ens genera un error, l'ordinador entén que una cadena comença amb les primeres cometes ***"Joan…*** i que acaba quan apareguin les següents***…alies ",*** però com la frase continua, es produeix un desajust de cometes que provoca l'error. Ho podrem resoldre de manera ràpida substituint les cometes delimitadores de la cadena. En comptes de fer servir les cometes dobles, utilitzem les cometes simples:

```
var bandoler='Joan Sala i Ferrer, alies "Serrallonga" famós bandoler català';
```

ara les cometes dobles de dins la frase no interfereixen amb les cometes delimitadores de la cadena que són cometes simples.

De manera semblant podem actuar en una situació inversa. Per exemple, si volguéssim assignar a la variable **institut** la cadena **Pons d'Icart** fent servir com a delimitador les cometes simples tindríem:

```
institut= 'Pons d'Icart';
```

aquest cas, ho solucionem delimitant la cadena amb les cometes dobles:

```
institut= "Pons d'Icart";
```

Ara ens trobem amb la pregunta que us hauríeu d'estar fent tots vosaltres. Que fem si la cadena que hem de delimitar inclou alhora cometes dobles i cometes simples?

La resposta està en les **seqüències d'escapament**. Es tracta d'una forma de dir-li a JavaScript que alguns caràcters els ha de tractar d'una manera especial.

Una seqüència d'escapament està formada per una contrabarra \ seguida d'un caràcter o una lletra que indica el que s'ha de fer. Ho comprovarem amb un exemple en el que apareixen alhora cometes dobles i cometes simples:

Imagineu que volem assignar la cadena ***La novel·la "L'ocell de foc" d'Emili Teixidor*** a la variable ***obra,*** observeu que la frase conté cometes dobles i cometes simples. Farem:

```
var obra = "La novel·la \"L'ocell de foc\" d'Emili Teixidor";
alert (obra);
```

us he marcat en color groc la seqüència d'escapament ( \" ), aquesta en concret, li diu a JavaScript que inclogui dins la cadena unes cometes sense considerar-les com un caràcter especial de delimitació, o sigui que les consideri com qualsevol altre caràcter. Si executeu el programa veureu que es mostra una finestra d'alerta amb el text tal com nosaltres volíem.

Una altra manera de fer el mateix podria haver estat:

```
var obra = 'La novel·la "L\'ocell de foc" d\'Emili Teixidor';
alert (obra);
```

Delimitem tota la cadena fent servir les cometes simples, i per evitar errors marquem amb seqüències d'escapament les cometes simples que apareixen dins del text.

Hi ha més seqüències d'escapament, però la majoria no tenen gran utilitat pels nostres propòsits observeu la següent taula.

| **Seqüències d'escapament** | |
|---|---|
| **Seqüència** | Descripció |
| **** | Insereix una contrabarra en el text |
| **\"** | Insereix unes cometes dobles |
| **\'** | Insereix una cometa simple |
| **\n** | Insereix un salt de línia |
| **\f** | Insereix un salt de pàgina (en documents per impressores) |
| **\t** | Insereix un salt de tabulació |
| **\b** | Retrocedeix un caràcter |

A banda de les que ja hem vist, possiblement les més interessants són la contrabarra i el salt de línia, la contrabarra perquè ens permet inserir una contrabarra en una cadena de text, si aparegués sola l'ordinador entendria que és el començament d'una seqüència d'escapament.

Amb la del salt de línia podem fer coses com ara:

```
<SCRIPT type="text/javascript">
<!--
      var obra = 'La novel·la "L\'ocell de foc"\nd\'Emili Teixidor';
      alert (obra);
 //-->
</SCRIPT>
```

Observeu la seqüència d'escapament marcada en groc i en la figura 2.2 la sortida que produeix al navegador Chrome.

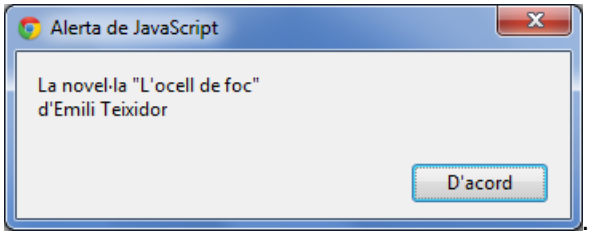



***Figura 2.2.*** *Sortida de \n en Chrome.*

**Booleans**, només poden contenir un dels dos possibles valors true o false. La seva declaració i assignació és igual que la de qualsevol altra variable, però tenint en compte que només li podem assignar una d'aquestes paraules clau, exemple:

```
var valorBool=true;
```

Hi ha d'altres tipus com ara les hores i dates o les matrius, però es tracta de tipus de dades especials (objectes) i per tant es treballa amb elles també d'una manera específica que veurem més endavant.

## 2.5 Operadors

Les variables per si soles són de poca utilitat. Fins ara, només hem vist com crear variables de diferents tipus i com mostrar el seu valor mitjançant la funció ***alert()***. Però, per fer programes realment útils, són necessàries un altre tipus d'eines.

Els operadors ens permeten manipular el valor de les variables de tipus string, realitzar operacions matemàtiques amb les numèriques o comparar diferents variables. D'aquesta manera, els operadors permeten als programes realitzar càlculs i prendre decisions lògiques en funció de comparacions i altres tipus de condicions. En JavaScript disposem dels següents tipus d'operadors bàsics:

Operador de cadenes.
Operadors aritmètics o numèrics.
Operadors de comparació.
Operadors d'assignació.
Operadors lògics.

### 2.5.1 Operador de cadenes

Bàsicament, per operar amb cadenes de caràcters o strings només disposem d'un operador, es tracta de la **concatenació,** representada pel signe més (+). La seva finalitat és unir dues o més cadenes en una sola, observeu l'exemple:

```
<SCRIPT type="text/javascript">
<!--
   nom = "Joan";
   cognoms = "Abad García"
   nomSencer = nom + cognoms;
   alert (nomSencer);
 //-->
</SCRIPT>
```

Que produeix la següent sortida (figura 2.3):

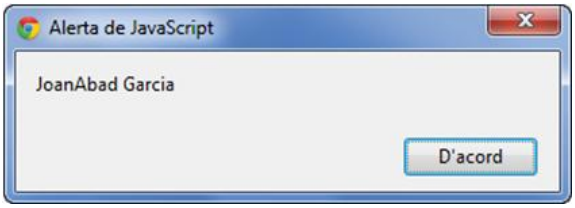



***Figura 2.4.*** *Concatenació.*

Observeu que a la variable ***nomSencer*** li hem assignat el resultat de la concatenació de **nom + cognoms**, és a dir hem agafat el contingut de la variable ***nom*** i el de la variable ***cognoms***, els hem ajuntat (concatenat) i el valor resultant l'hem assignat a ***nomSencer***. Posteriorment l'hem mostrat amb alert.

Fixeu-vos que a la sortida es mostra ***JoanAbad García***, sense espai entre Joan i Abad. Una concatenació ajunta el valor de les dues variables de cadena tinguin el valor que tinguin a dins seu, en el cas que ens ocupa, ni la variable nom acaba en espai ni la variable cognoms comença amb espai, per tant no hi ha espai. Si volguéssim un espai, hauríem d'afegir-lo abans de tancar les cometes en la variable nom (***nom="Joan "***) o bé al començament de la variable cognoms (***cognoms = " Abad García"***). Ara el programa funcionarà tal com volíem, però si us fixeu hem modificat el contingut d'una de les variables. Per tant una solució millor pot ser:

```
<SCRIPT type="text/javascript">
<!--
   nom = "Joan";
   cognoms = "Abad Garcia"
   nomSencer = nom + " " + cognoms;
   alert (nomSencer);
 //-->
</SCRIPT>
```

Hem concatenat el nom, seguit d'un espai (" ") i dels cognoms, es a dir, hem concatenat tres valors. La part marcada en groc és una cadena que conté un espai, es tracta d'un caràcter espai delimitat per cometes. Ara el resultat és el que volíem, i no hem alterat el contingut de cap de les variables. (Modifiqueu l'exemple i proveu-lo).

En qualsevol altre llenguatge de programació, si intentem concatenar un valor string amb un valor numèric el més probable és que es produeixi un error, donat que la concatenació sempre s'ha de fer entre valors de tipus string. Però JavaScript és bastant permissiu pel que fa al tractament dels tipus de dades, i en comptes de donar-nos error, farà una conversió implícita del valor numèric a string, realitzant la concatenació correctament. Exemple:

```
<SCRIPT type="text/javascript">
<!--
   v1= "Total factura: ";
   total = 1200;
   missatge = v1 + total;
   alert (missatge);
 //-->
</SCRIPT>
```

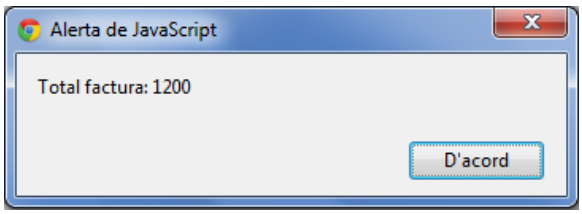



***Figura 2.4.*** *Resultat concatenació*

Ha agafat una còpia del valor de la variable numèrica, l'ha convertit en string i ha fet la concatenació, al final de l'operació la variable ***total*** continua sent numèrica i manté el seu valor de 1200, ja que ha treballat amb una còpia.

Com veurem al punt següent, el signe + també s'utilitza per sumar números. JavaScript sap que si els operands són numèrics ha de realitzar una suma, però si un d'ells o tots dos són string, ha de fer una concatenació. Exemple:

```
<SCRIPT type="text/javascript">
<!--
   v1= "123";
   v2 = 456;
   v3 = 789;
   alert (v1 + v2);
   alert (v2 + v3);
 //-->
</SCRIPT>
```

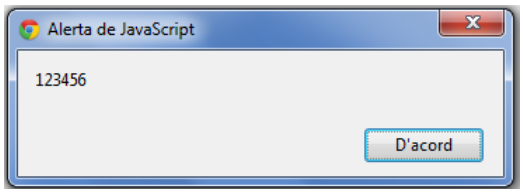



***Figura 2.5.*** *Primer alert – Concatenació*

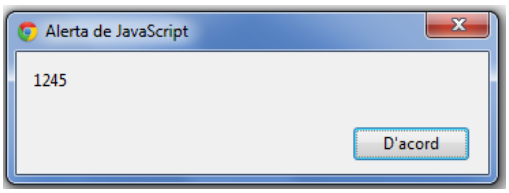



***Figura 2.6.*** *Segon alert – Suma*

En el primer alert està concatenant la cadena "123" amb el valor numèric 456, per tant els ajunta i mostra 123456. En el segon cas, tots dos valors són numèrics i entén que els volem sumar, per tant fa la suma 456 + 789 = 1245.

### 2.5.2. Canviar el tipus d'una variable de manera explícita

Ja hem vist que quan apareixen en una expressió variables de diferent tipus, JavaScript intenta convertir-les de manera implícita per poder dur a terme l'operació. A banda d'aquest comportament, també disposem de funcions creades expressament per realitzar conversions explícites de dades, d'aquesta manera evitem dependre del que decideixi JavaScript en cada moment, i així som nosaltres els que decidim expressament la conversió o el tipus de dades amb que volem treballar.

La funció **parseFloat();** ens permet (o al menys ho intentarà) convertir una cadena alfanumèrica en un valor numèric fraccionari (amb decimals):

```
<SCRIPT type="text/javascript">
<!--
   var valor="123.50";
   valor=parseFloat(valor);
   alert ("El contingut de valor es " + valor + " és del tipus "+ typeof(valor));
 //-->
</SCRIPT>
```

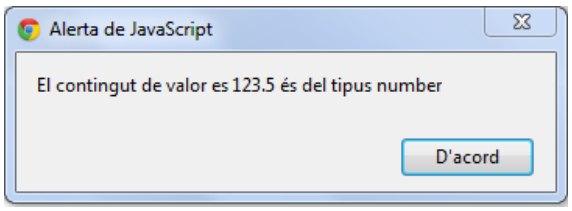



***Figura 2.7.*** *Sortida de parseFloat()*

En primer lloc hem creat una variable alfanumèrica a la que li hem assignat ***123.50,*** tots són dígits numèrics a més d'un punt que després de la conversió serà el punt decimal. A continuació, hem fet servir la funció ***parseFloat()*** que converteix una cadena en un valor numèric de tipus fraccionari o amb decimals. Fixeu-vos que entre parèntesis de la funció, com a paràmetre li passem el nom de la variable que volem convertir.

A l'exemple el resultat l'hem assignat a la mateixa variable ***valor***, amb la qual cosa li hem canviant el tipus, però podríem haver fet servir una variable diferent, per exemple ***nova=parseFloat(valor);*** amb la qual cosa, ara la variable ***nova*** seria de tipus numèric, però ***valor*** continuaria sent alfanumèrica.

Si la variable de cadena que volem convertir a numèrica conté caràcters no numèrics, convertirà fins on pugui, per exemple la cadena ***valor="123x50";*** donarà com a resultat ***123***. A partir de la x no

sap com seguir la conversió. ***Atenció***: si la cadena comença per un caràcter no numèric, no podrà convertir-la i donarà com a resultat ***NaN***.

La funció ***parseFloat(),*** s'acostuma fer servir per obtenir valors del tipus ***float*** (fraccionaris), tot i que com a resultat també pot donar un enter.

*Proveu els supòsits del paràgraf anterior.*

La funció **parseInt()**, és semblant a l'anterior però es fa servir per intentar convertir una cadena alfanumèrica en un valor enter. La seva sintaxi és ***parseInt (cadena, base de numeració).*** La base de numeració és opcional, si no s'especifica s'entén 10.

Tant aquesta funció com l'anterior la podem fer servir per assegurar-nos que una variable conté una dada numèrica. Aquesta tècnica la veurem més endavant.

Observeu ara un exemple de la utilització de parseInt():

```
<SCRIPT type="text/javascript">
<!--
   var res, valor="100";
   alert ("El contingut de valor es " + parseInt(valor) );
   alert (valor + " en binario equival a " + parseInt(valor,2));
   alert (valor + " en Hex equival a " + parseInt(valor,16)); //-->
</SCRIPT>
```

La funció **toString().** La farem servir per fer la inversa del que hem fet fins ara, és a dir per convertir valors numèrics a cadena. De fet, tot i que he dit que és una funció, no es tracta d'una funció, sinó que és un ***mètode*** d'un objecte, en aquest cas de les variables numèriques.

Com encara no hem treballat els objectes, per ara, el podem considerar con si fos una funció, ja veurem més endavant que són funcions i que són mètodes, la única cosa que hem de tenir en compte de moment és que la seva sintaxi és diferent al que hem vist fins el moment.

```
<SCRIPT type="text/javascript">
<!--
   var res, valor=128;
   res = valor.toString();
   alert ("El contingut de valor es " + valor + " és del tipu " + typeof(valor) );
   alert ("El contingut de res es " + res + " és del tipu " + typeof(res) );
//-->
</SCRIPT>
```

Observeu la sintaxi especial, ***nomvariable.toString().***

El mètode toSring() converteix el contingut de la variable a cadena, utilitzant la base de numeració decimal, però té un altra versió en la que podem especificar entre parèntesis la base en la volem obtenir el resultat. Mireu l'exemple:

```
<SCRIPT type="text/javascript">
<!--
   var res, valor=12;
   res = valor.toString(16);
   alert ("El contingut de valor es " + valor + "\n\n"+
          "En binari és " + valor.toString(2) + "\n" +
          "En Hex és " + valor.toString(16));
//-->
</SCRIPT>
```

*Proveu el resultat .*

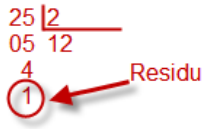
### 2.5.3 Operadors aritmètics

Son aquells que ens permeten realitzar operacions matemàtiques fonamentals entre valors de tipus numèric. Els podeu veure a la taula següent:

| Operador | Nom | Descripció |
|---|---|---|
| + | Suma | Efectua la suma entre els operands |
| - | Resta | Efectua la resta entre els operands |
| * | Multiplicació | Efectua la multiplicació entre els operands |
| / | Divisió | Efectua la divisió entre els operands |
| % | Mòdul | Retorna el residu produït al fer la divisió entre els operands |
| ++ | Increment | Incrementa l'operand en una unitat |
| -- | Decrement | Disminueix l'operand en una unitat |

Aquest operadors no tenen cap dificultat, ja que es tracta de les operacions bàsiques que podem realitzar amb números en matemàtiques. Potser els que necessiten una mica d'explicació siguin els tres últims.

**Mòdul**, (%) és el residu que obtenim quan realitzem una divisió entre dos valor numèrics, per exemple 25/2: Tenim que 25 % 2 → 1




A continuació tenim un exemple:

```
<SCRIPT type="text/javascript">
<!--
   v1= 25;
   v2=2;
   alert ("v1+v2=" + (v1 + v2));//Suma
   alert ("v1-v2=" + (v1 - v2));//Resta
   alert ("v1*v2=" + v1 * v2);//Multiplicació
   alert ("v1/v2=" + v1 / v2);//Divisió
   alert ("v1%v2=" + v1 % v2);//Mòdul
 //-->
</SCRIPT>
```

Observeu que tant la suma com la resta les hem hagut d'incloure entre parèntesis, això ho hem fet per obligar al navegador a que faci les operacions aritmètiques abans de fer la concatenació. Es tracta d'un problema de prioritat d'operadors que veurem en un punt següent. De moment només ens cal saber que la multiplicació, la divisió i el mòdul tenen una prioritat més elevada que la suma, la resta i la concatenació. Això vol dir que si per exemple, en una expressió apareix una suma i una multiplicació primer es farà la multiplicació.

*Proveu el mateix exemple, però sense que la suma ni la resta aparegui entre parèntesis:*

```
alert ("v1+v2=" + v1 + v2);//Suma
alert ("v1-v2=" + v1 - v2);//Resta
```

*executeu-lo i comproveu la sortida, que ha passat? Perquè creieu que ha passat?*

*Feu el bloc 1 d' exercicis. Punt 2, del full de pràctiques (Pàg. 1).*

**Increment i decrement**, es tracta d'incrementar o disminuir el valor d'una variable numèrica en una unitat, té dues variants que veurem tot seguit: ***increment previ*** i ***increment posterior***, segons es realitzi la operació abans o després de la operació en la que està intervenint. Ho veurem millor amb un exemple, primer provarem ***increment previ***:

```
<SCRIPT type="text/javascript">
<!--
     v1= 10;
     alert (++v1);
     alert ("El valor de v1 després d'alert és: " + v1);
 //-->
</SCRIPT>
```

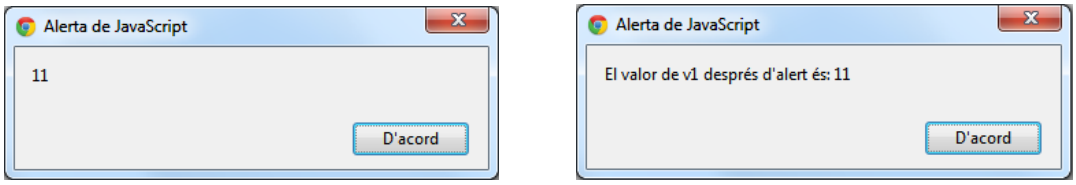



***Figura 2.8.*** *Sortida d'increment previ.*

Tornem a provar el mateix exemple, però ara ho farem amb un increment posterior:

```
<SCRIPT type="text/javascript">
<!--
   v1= 10;
   alert (v1++);
   alert ("El valor de v1 després d'alert és: " + v1)
 //-->
</SCRIPT>
```

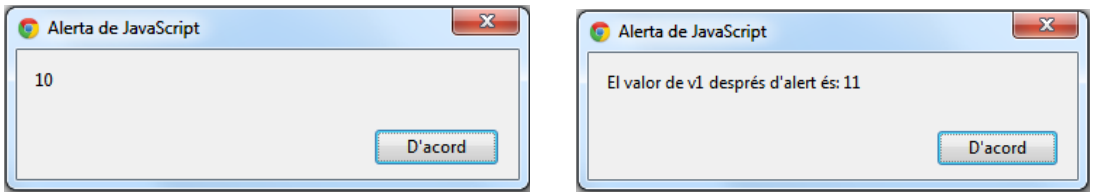



***Figura 2.9.*** *Sortida d'increment posterior*

Fixeu-vos que en tots dos casos la variable ***v1*** ha acabat emmagatzemant el valor 11. La forma d'indicar si volem ***increment previ*** o ***increment posterior*** es correspon a la posició on situem l'operador ***++*** (al davant o al darrera de la variable). També podeu veure el funcionament exacte d'aquests operadors: el primer cas ***++v1***, incrementa el valor abans de fer-lo servir en la instrucció alert, el segon cas, primer efectua la instrucció alert amb el valor actual de ***v1*** (10) i posteriorment l'incrementa.

L'operador de decrement **--** funciona de manera idèntica que l'operador d'increment **++,** però en comptes de sumant, restant.

Recordeu que JavaScript també pot treballar perfectament amb números fraccionaris, per tant, tot i que els exemples estaven realitzats amb nombres enters, no hi ha cap problema en fer servir nombres fraccionaris.

### 2.5.4 Operadors de comparació

Aquest operadors, tal com indica el seu nom, els farem servir per comparar entre si el valor de les variables, el resultat d'aquesta comparació sempre serà un valor boolean, (true o false - vertader o fals), les comparacions amb aquests operadors només es poden realitzar amb números o cadenes.

Aquests tipus d'operadors són de crucial importància en el llenguatge, ja que combinats amb d'altres instruccions, ens permeten prendre decisions segons s'acompleixi o no alguna condició.

A la taula següent podeu veure la relació d'operadors de comparació

| Operador | Nom | Descripció |
|---|---|---|
| == | Igual | Retorna true si tots dos operant-los són iguals |
| != | Diferent | Retorna true si tots dos operadors són diferents |

| | | |
|---|---|---|
| **>** | Major que | Retorna true si l'operand de l'esquerra és més gran que el de la dreta |
| **>=** | Major o igual que | Retorna true si l'operand de l'esquerra és més gran o igual que el de la dreta |
| **<** | Menor que | Retorna true si l'operand de l'esquerra és més petit que el de la dreta |
| **<=** | Menor o igual que | Retorna true si l'operand de l'esquerra és més petit o igual que el de la dreta |
| **===** | Estrictament igual | Retorna true si els dos operands són del mateix tipus i guarden el mateix valor. |
| **!==** | Estrictament diferent | És las inversa de l'anterior retorna true on l'anterior retornaria false. |

Observeu l'exemple:

```
<SCRIPT type="text/javascript">
<!--
   v1= 25;
   v2=2;
   alert ("v1=25 i v2=2\n\n" +
      "v1 == v2-" + (v1 == v2) + "\n" +
      "v1 != v2 -" + (v1 != v2) + "\n" +
      "v1 > v2 -" + (v1 > v2) + "\n" +
      "v1 < v2 -" + (v1 == v2));
 //-->
</SCRIPT>
```

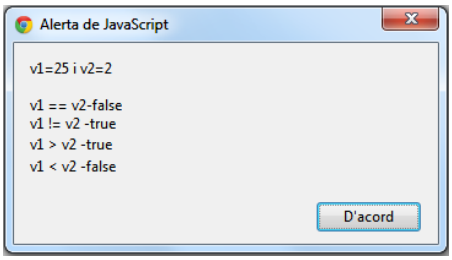



***Figura 2.10.*** *Sortida operadors de comparació*

Ara veurem un altre exemple en el que podem comprovar l'operador **d'igualtat estricta**

```
<SCRIPT type="text/javascript">
<!--
   v1 = "2";
   v2 = 2;
   alert ("v1=\"2\" i v2=2\n\n" +
      "v1 == v2-" + (v1 == v2) + "\n" +
      "v1 === v2 -" + (v1 === v2));
 //-->
</SCRIPT>
```

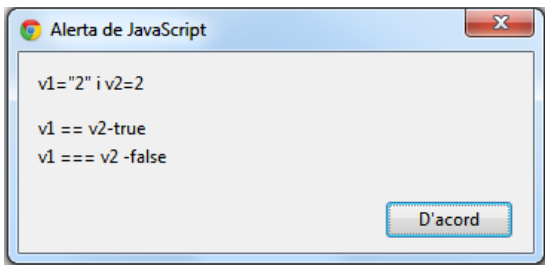



***Figura 2.11.*** *Sortida operador de comparació*

A la variable ***v1*** li hem assignat el valor de tipus cadena ***"2"***, mentre que a ***v2*** li hem assignat el valor numèric ***2***, quan els comparem, tenim que l'operador d'igualtat (**==**) retorna true, mentre que el d'igualtat estricta (**===**)retorna false.

**Comparació de cadenes.** Fins ara totes les comparacions les hem fet amb nombres, que són fàcils d'entendre, però, que passa quan comparem valors de tipus cadena? Quins resultats obtenim?

Amb els operadors d'igualtat o de desigualtat no hi ha cap problema, tots saben si "Hola" és igual a "Hola" o és diferent. Però que passa si volem saber si una cadena és més gran o més petita que una altra?

Un ordinador és un sistema que treballa a nivell bàsic sempre amb nombres, i més concretament amb valors binaris (1 i 0). Per tant, qualsevol informació que vulguem representar, en el seu nivell més baix ha d'estar expressada en forma numèrica. Les cadenes no són una excepció i es codifiquen internament seguint el codi ASCII ( ***American Standard Code for Information Interchange — Codi Estàndard Americà per Intercanvi d'Informació***) que fa correspondre a cada caràcter un número comprés entre 0 i 255. (cada caràcter ocupa 8 bits).

Ara ja estem en condicions de saber com resolt JavaScript aquest tipus de comparacions. Quan es troba una comparació de cadenes les compara caràcter a caràcter: el primer caràcter de la primera cadena amb el primer de la segona, si són iguals continua comparant el segon caràcter de la primera cadena amb el segon de la segona i així va fent successivament, fins que troba una comparació que trenqui la igualtat. En aquest moment, el caràcter que té el codi ASCII més gran fa que la cadena que l'inclou sigui més gran que l'altra.

Hem de tenir en compte que les lletres tenen codis consecutius (A=65, B=66, C=67... a=97, b=98, c=99...) i que les lletres majúscules tenen un codi ASCII més petit que les minúscules, per tant, no cal que ens sapiguem de memòria el codi ASCII, sinó que, si les ordenem alfabèticament, la que està pel davant de l'altra és la més petita (sense oblidar la diferència entre majúscules i minúscules).

Si en una comparació de cadenes tots els caràcters coincideixen, però una cadena acaba abans que l'altra, la que acaba abans és la més petita. Per exemple ***"Anton"*** és més petit que ***"Antoni"***

```
<SCRIPT type="text/javascript">
<!--
   var n1 = "Antonia", n2 = "Antonio", n3 = "ANTONIO";
   alert (n1 +" > "+ n2 +" = " +  (n1>n2));
   alert (n2 +" > "+ n3 +" = " +  (n2>n3));
 //-->
</SCRIPT>
```

Resultats:

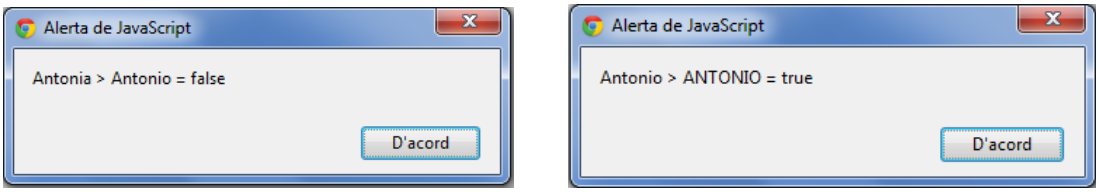



***Figura 2.12.*** *Comparació de cadenes*

En el primer alert tota la cadena és igual fins que arribem a la última lletra en que comparem la lletra a de "Antonia" amb la o de "Antonio", resultant que la a és més petita que la o i per tant la comparació és falsa. En el segon alert, la primera lletra de les dues cadenes és igual (una A majúscula), però al comparar la segona lletra es troba amb una **n** minúscula per una banda i una **N** majúscula per l'altra, com que les minúscules tenen un codi ASCII superior a les majúscules, el resultat és que "Antonio" és més gran que "ANTONIO".

**Atenció**: Tot i que us he parlat del codi ASCII, realment JavaScript codifica les cadenes de text seguint l'estàndard **UNICODE**.

ASCII utilitza 8 bits, per emmagatzemar cada caràcter, amb la qual cosa només pot codificar 256 caràcters diferents, aquesta limitació dóna problemes en un mon globalitzat en el que els ordinadors han de ser capaços de representar caràcters de qualsevol idioma del mon (llatins, hebreus, àrabs, japonesos...). Unicode resol el problema utilitzant 16 bits per representar cada caràcter, d'aquesta manera la capacitat augmenta de 255 a 65.536 caràcters diferents. Per evitar problemes de compatibilitat, els primers 256 caràcters d'Unicode es corresponen amb els de ASCII

### 2.5.5 Operadors d'assignació

Com ja hem vist a llarg de multitud d'exemples l'operador bàsic d'assignació és el signe igual (=), el procés d'assignació sempre és el mateix, s'assigna el valor que hi ha a la part dreta del signe igual a la variable que apareix a l'esquerra. Hem de tenir en compte que, a banda d'assignar un valor directament, també podem assignar el resultat d'una expressió. En aquests casos JavaScript primer calcula el resultat de l'expressió i després l'assigna a la variable. Això ens pot proporcionar expressions que en matemàtiques no tindrien sentit, com ara ***total = total + 1;*** però que és totalment correcta en programació. Si repassem el que acaben de veure, aquesta expressió primer calcula la part de la dreta de l'igual, és a dir: al valor actual de la variable ***total*** se li suma un, una vegada tenim el resultat el tornem ha assignar a la variable ***total,*** per tant estem incrementant el valor de ***total*** en un. Això es coneix en programació com un comptador.

A banda d'aquest operador d'assignació, JavaScript ens proporciona d'altres que són una forma abreujada de fer una operació matemàtica i una assignació alhora, aquests són: +=, -=, *=, /= i %=. Així tenim que ***total += 1;*** es una forma abreujada de fer ***total = total + 1;*** , (suma 1 a total). La resta d'operadors d'aquest tipus segueixen el mateix esquema. Exemple:

```
<SCRIPT type="text/javascript">
<!--
   var n1 = 25;
   alert (n1 -= 5);// resta 5 a n1;
 //-->
</SCRIPT>
```

### 2.5.6 Operadors lògics

Són operadors que treballen amb expressions booleanes senzilles per avaluar expressions de decisió més complexes. Un operador lògic pren dos operands booleans (true o false) i retorna com a resultat un altre valor boolean. Els operands acostumen a ser expressions lògiques que retornen un valor boolean.

Així tenim l'operador **&&** (AND), que retorna true si els dos operands són true. L'operador **||** (OR) que retorna true en el cas que al menys un dels operands sigui true i finalment tenim l'operand **!** (NOT) que inverteix el resultat d'una operació lògica.

Exemple: Imagineu que per entrar a un espectacle s'ha de ser major d'edat o tenir permís patern. Tenim una variable ***edat*** que emmagatzema la edat d'una persona, i la variable booleana ***permis*** que conté un valor boolean que indica si té permís patern. Volem saber si pot entrar a l'espectacle:

```
<SCRIPT type="text/javascript">
<!--
   var edat = 17; permis = true;
   alert ((edat>=18) || (permis=true));//
 //-->
</SCRIPT>
```

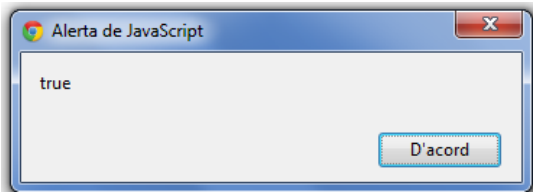



***Figura 2.13.*** *Resultat de l'operador OR ( || )*

Observeu que el resultat global és ***vertader***, això indica que la condició s'acompleix. Si analitzem el codi, el primer operand ***(edat>=18)*** és fals, perquè ***edat=17,*** mentre que el segon ***(permis=true)*** és vertader. Com el operador que estem fent servir **||** (OR) requereix que almenys un sigui vertader i un ja ho és, el resultat global és vertader.

*Ara imagineu que per entrar és obligatori ser major d'edat i a més el permís patern (una condició una mica irreal, però bé és un exemple). Modifiqueu el programa per representar aquesta situació i observeu el resultat, no canvieu el valor de les variables.*

Aquests exemples estan una mica forçats per demostrar el funcionament d'aquest operadors i pot ser no reflecteixen la seva utilitat real, de seguida veurem les instruccions de pressa de decisions i control de bucles on aquests operadors ens seran de molta utilitat.

### 2.5.7 Precedència dels operadors

A mida que anem programant, ens poden trobar amb expressions en les que intervenen més d'un operador, per tant hem de saber en quin ordre s'executaran. En JavaScript els operadors estan classificats jeràrquicament de manera que uns tenen major prioritat que altres. Així tenim que els operadors de major precedència són el increment previ (++) i el decrement previ (--), a continuació la multiplicació (*), la divisió (/) i el mòdul (%) i finalment la suma (+) i la resta (-).

Si es troben en una mateixa expressió operadors d'igual prioritat es resol per l'ordre en que apareixen, d'esquerra a dreta. Realment, els últims en executar-se seran els de increment posterior (++) i els de decrement posterior (--), si us recordeu s'executen després de calcular el resultat de tota l'expressió i abans de passar a la següent instrucció del codi.

Si volem forçar a que una operació d'una precedència inferior es realitzi abans que una superior, la inclourem entre parèntesis.

Jo sempre recomano que, si no esteu segurs de l'ordre en que s'executarà una expressió, feu servir parèntesis per assegurar-vos que ho fa en l'ordre que realment voleu. El programa no donarà error, ni anirà més lent, si fiqueu parèntesis de més tot i que no fossin necessaris.

## 2.6 Sentències

Una vegada sabem treballar amb les variables, que són les dades dels nostres programes i crear expressions bàsiques amb elles i els operadors, hem d'aprendre a crear codi, amb el que realment indiquem al nostre programa que ha de fer.

Fins ara hem vist com manejar les dades del programa, però només amb variables no podem crear un programa. Necessitem escriure codi font que ens permeti aconseguir que el programa faci més coses. Normalment els programes tenen un flux d'execució: s'executen línia a línia, interpretant i comprovant el resultat de l'execució. Però amb això no tenim prou, de vegades hem de controlar el que fa el programa, executant un codi o un altre depenent de les circumstàncies o de si s'acompleixen o no certes condicions, altres vegades hem repetir el mateix codi diverses vegades segons unes condicions concretes.

### 2.6.1 Sentències condicionals. Decisions

Són instruccions que ens permeten trencar el flux normal d'execució del nostre programa. Fins ara, les instruccions les executaven una darrera l'altra, de manera seqüencial. Ara podem avaluar condicions i executar un bloc d'instruccions si el resultat és vertader o un altre diferent si és fals.

La sentència condicional per excel·lència en JavaScript és **if,** que veurem tot seguit en les seves diferents variants:

La forma més simple en que podem utilitzar- la té la següent sintaxi:

```
if (condició){
    instrucció o bloc d'instruccions que
    s'executaran si la condició és vertadera
   }
```

Estudiem-la detingudament, observeu que està formada per la paraula clau ***if*** seguida de la condició que volem avaluar tancada entre parèntesis. A continuació tenim la instrucció o el grup d'instruccions que s'han d'executar si s'acompleix la condició, tancades entre claus { i }.

En aquesta forma de la instrucció ***if***, si s'acompleix la condició, s'executaran la/les instrucció/ons, del bloc de codi. Si no s'acompleix se saltarà les instruccions del bloc de codi i continuarà el programa amb les instrucció següents al tancament de les claus.

Si com a resultat del compliment de la condició, només s'ha d'executar una sola línia, JavaScript no obliga a que estigui inclosa entre claus { i }. Tot i que jo aconsello incloure-la sempre per claredat del codi que generem. Sintaxi:

```
if (condició)
    una única instrucció sense claus;
```

El bloc de codi resultant d'una instrucció ***if*** pot ser tan ampli com ens sigui necessari, pot contenir des de una única instrucció a multitud d'elles, fins i tot incloure altres instruccions condicionals, bucles, crides a funcions, etc. que per la seva banda també poden incloure d'altres blocs de codi.

A continuació podem veure l'ordinograma de com funcionaria aquesta forma de la instrucció if (figura 2.14)

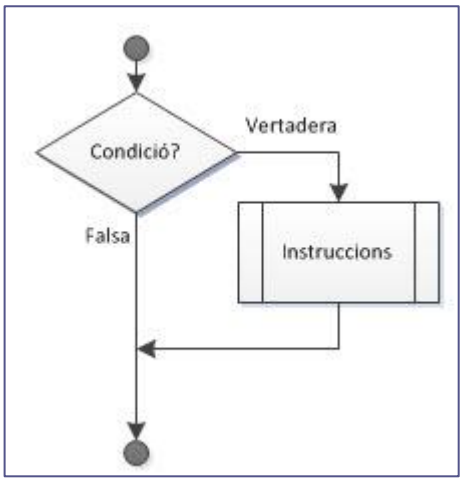



***Figura 2.14.*** *Ordinograma de la instrucció if*

Exemple:

```
<SCRIPT type="text/javascript">
<!--
   var edat;
   edat=prompt("Entra la teva edat","");
   edat=parseInt(edat);
   if (edat<18){
       alert("Ho sento, no pots entrar, ets menor d'edat");
      }
//-->
</SCRIPT>
```

Abans de comentar el funcionament del programa, deixeu-me que us expliqui una nova instrucció que he fet aparèixer per primera vegada en aquets codi, per fer-lo més interactiu. Es tracta de la instrucció ***prompt,*** que ens permet fer una pregunta a l'usuari i emmagatzemar la seva resposta en una variable. Aquesta instrucció, fa que aparegui una finestra tipus alert que ens fa una pregunta i espera una resposta. Observeu la figura 2.15. El valor que nosaltres teclegem s'assignarà a la variable. Al l'exemple assignem "12" a la variable edat (figura 2.15).

Prompt sempre retorna el valor que teclegem en forma de string, per tant, si cal, l'haurem de convertir al tipus adequat, com he fet a l'exemple amb la instrucció ***edat=parseInt(edat);***

També podeu veure que la sintaxi de prompt inclou dos paràmetres dins dels parèntesis, separats per una coma. El primer és un string on escriurem la pregunta ***("Entra la teva edat"),*** el segon és un valor que ja sortirà escrit com a resposta a la caixa de text, diguem-ne que és con un suggeriment que li fent a l'usuari, (li escrivim la resposta); per descomptat que ell, si vol, la pot canviar per la que vulgui. A l'exemple, com que jo no volia suggerir-li cap resposta he escrit com a segon paràmetre una cadena buida -dues cometes seguides sense res al mig- (***""***).

Si l'usuari clica el botó cancel·lar la instrucció prompt retorna un valor ***null***. En algunes versions antigues d'Internet Explorer es retornava una cadena buida "".

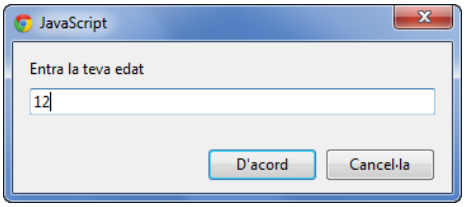



***Figura 2.15.*** *La instrucció prompt*

Una vegada que hem vist la instrucció ***prompt*** continuarem amb l'explicació de la resta del codi de l'exemple anterior i la instrucció ***if***.

Simplement li hem dit que si la edat és menor de 18 escrigui a la pantalla un alert informant que no pot entrar perquè és menor d'edat. Observeu la figura 2.16.

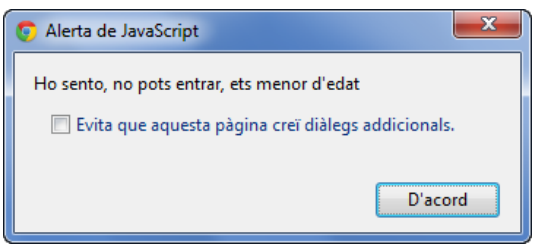



***Figura 2.16.*** *S'acompleix if*

Fixeu-vos que la **condició** que hem fet servir és una expressió relacional, de fet podríem haver fet servir qualsevol valor o expressió que retorni o contingui un valor boolean.

Observeu també que si s'acompleix la condició només tinc una instrucció alert, per tant, tot i que a l'exemple la he inclòs entre claus, podria no haver-ho fet. Observeu exemple modificat:

```
<SCRIPT type="text/javascript">
<!--
   var edat;
   edat=prompt("Entra la teva edat","22");
   edat=parseInt(edat);
   if (edat<18)
       alert("Ho sento, no pots entrar, ets menor d'edat");
//-->
</SCRIPT>
```

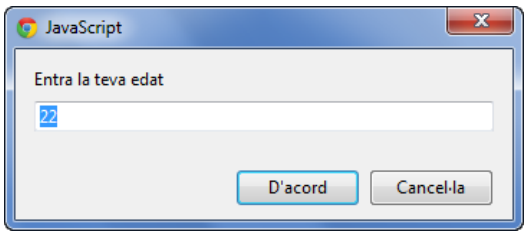



***Figura 2.17.*** *Prompt amb un valor suggerit*

Ara no està delimitat per claus { i }, si el proveu veureu que funciona exactament igual que abans. Proveu també a donar una resposta més gran de 17 veureu que la condició ***edat<18*** no s'acompleix i per tant la instrucció ***alert*** no s'executa. A l'exemple s'acaba el programa perquè no hi ha més línies de codi, però si les haguessin, el programa continuaria amb la de després de l'alert.

Fixeu-vos també que he aprofitat la modificació de l'exemple per fer que aparegui per defecte el valor 22 com a suggeriment de resposta (figura 2.17), al codi l'he marcat en color groc.

### If amb alternativa

En la forma anterior de la instrucció ***if***, indiquem el que s'ha de fer si s'acompleix la condició però no teníem cap alternativa pel cas que no s'acompleixi. A vegades aquest comportament ja ens està bé, però també hauríem de tenir la possibilitat de poder donar una alternativa, en el sentit de dir: "si s'acompleix la condició fes això, i si no s'acompleix fes això altre". Seria quelcom semblant a la representació de l'ordinograma de la figura 2.18.

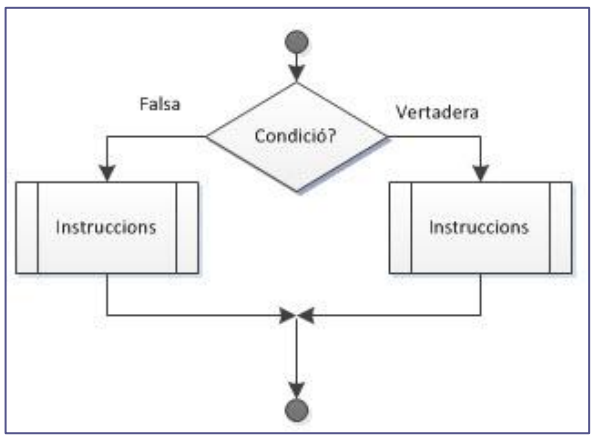



***Figura 2.18.*** *If amb alternativa.*

Doncs bé, JavaScript ens proporciona aquesta possibilitat tal com veiem al següent exemple:

```
<SCRIPT type="text/javascript">
<!--
   var edat;
   edat=prompt("Entra la teva edat","");
   edat=parseInt(edat);
   if (edat<18){
     alert("Ho sento, no pots entrar, ets menor d'edat"); }
   else{
     alert("Benvingut al nostre espectacle");}
//-->
</SCRIPT>
```

Fixeu-vos que el bloc alternatiu, el que s'executarà si no s'acompleix la condició, apareix després de la nova clàusula ***else.***

Executeu el programa dues vegades, en una doneu una edat inferior a 18 i en l'altra una major de 18, veureu que obteniu una resposta diferent en cadascun dels casos, és a dir si sou majors d'edat us dona la benvinguda i en cas contrari us informa que no podeu entrar.

Si s'acompleix la condició executa el bloc d'instruccions que hi ha després del ***if*** i se salta el de després de ***else.*** Si no s'acompleix la condició se salta el bloc d'instruccions del ***if*** i executa el del ***else***.

Com que tant després del ***if*** com després del ***else***, només hem escriure una única instrucció, aquí també podríem haver prescindit de les claus:

```
<SCRIPT type="text/javascript">
<!--
   var edat;
   edat=prompt("Entra la teva edat","");
   edat=parseInt(edat);
   if (edat<18)
      alert("Ho sento, no pots entrar, ets menor d'edat");
   else
      alert("Benvingut al nostre espectacle");
//-->
</SCRIPT>
```

Aneu amb compte amb el lloc on posicioneu el punt i coma dins una instrucció ***if***, observeu l'exemple, si la instrucció ***if (edat<18)*** o la instrucció ***else*** l'haguéssim acabat en un punt i coma el programa no hauria funcionat, produint un error, com podeu veure a la figura 2.19.

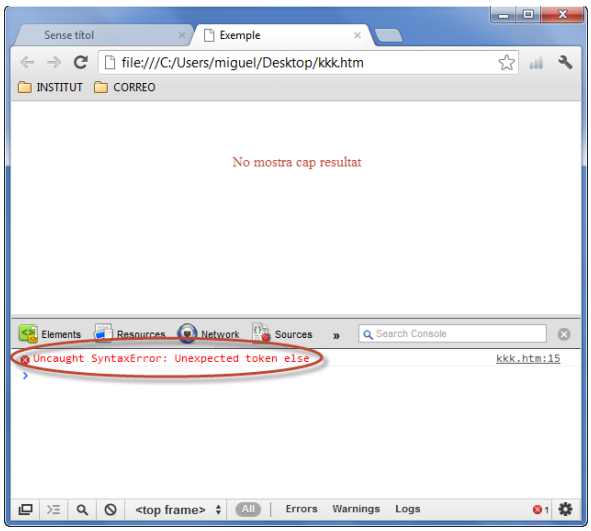



***Figura 2.19.*** *Error incorrecta utilització del punt i coma*

### Condicional múltiple. La sentència if-else-if

De vegades, per donar resposta a certs algorismes, el nostre programa requerirà formar línies de codi més complexes de manera que podem encadenar sentències ***if*** seguint el següent raonament: "si s'acompleix aquesta condició fes això, si no s'acompleix, mira si s'acompleix aquesta altra condició i fes això altre, si tampoc s'acompleix, mira aquesta altra condició...". Fixeu-vos, en l'organigrama que potser us aclarirà una mica més bé el que us vull dir (figura 2.20):

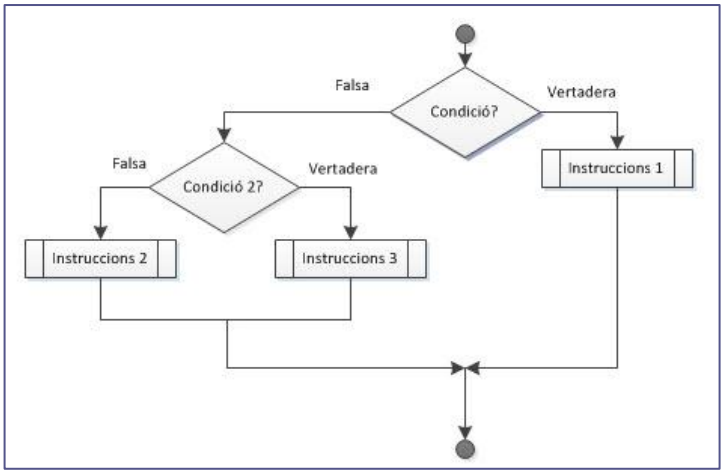



***Figura 2.20.*** *Ordinograma de la sentència if-else-if*

La seva sintaxi podria ser quelcom semblant a:

```
if (condició 1){
   Bloc d'instruccions que s'executaran si s'acompleix la condició 1.
   Quan acaba l'execució d'aquest bloc finalitza tota l'estructura if.
} else if (condició 2){
   Bloc d'instruccions que s'executaran si s'acompleix la condició 2.
   Si s'executa aquest bloc, en acabar dóna per finalitzada tota l'estructura if.
} else if (condició 3){
   Bloc d'instruccions que s'executaran si s'acompleix la condició 3.
   Si s'executa aquest bloc, en acabar dóna per finalitzada tota l'estructura if.
...
...
...
} else if (condició n){
   Bloc d'instruccions que s'executaran si s'acompleix la condició .
   Si s'executa aquest bloc, en acabar dóna per finalitzada tota l'estructura if.
} else {
   Bloc que s'executarà només si cap de les condicions que s'han avaluat en les
   línies anteriors ha resultat vertadera
}
```

Exemple amb codi real:

```
<SCRIPT type="text/javascript">
<!--
   var edat;
   edat=prompt("Entra la teva edat","");
   edat=parseInt(edat);
   if (edat<12){
       alert("És un infant");
   } else if (edat<18){
             alert("És adolescent");
   } else if (edat<30){
       alert ("És Jove");
   } else if (edat<50){
       alert ("És adult");
   } else if (edat<65){
       alert ("És madur");
   } else if (edat<80){
       alert ("És Jubilat");
   } else {
       alert ("És ancià");
   }
//-->
</SCRIPT>
```

*Proveu l'exemple i el seu funcionament entrant successivament diferents edats i observant el resultat. Perquè es torni a carregar la pàgina i us torni a fer la pregunta, cliqueu el botó d'actualitzar pàgina del vostre navegador o bé pitgeu la tecla F5.*

### La sentència switch

En molts casos, és necessari comparar el contingut d'una variable amb diferents valors possibles. Tot i que aquest problema es pot solucionar amb successius ***if - else if,*** JavaScript ens proporciona per aquests casos l'estructura condicional ***switch***, que es correspondria amb l'ordinograma de funcionament de la figura 2.21. I que respon a la següent sintaxi general:

```
Switch (expressió){
   case valor1:
      Instruccions que s'executen si espressió és igual a valor1;
      break;
   case valor2:
      Instruccions que s'executen si espressió és igual a valor2;
      break;
   ...
   ...
   ...
   case valor n:
      Instruccions que s'executen si espressió és igual a valor n;
      break;
    default:
   Instruccions que s'executen si el valor no es correspon amb cap dels anteriors;
}
```

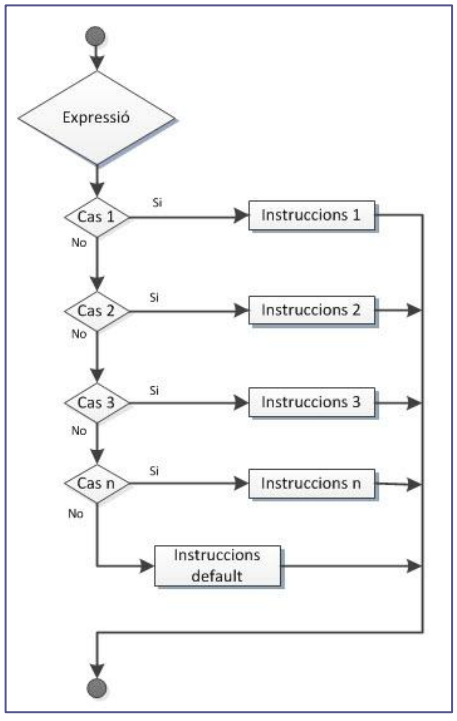



***Figura 2.21.*** *Ordinograma de la sentència Switch*

Tot seguit ho veurem en funcionament amb un exemple.

```
<SCRIPT type="text/javascript">
<!--
   var resultat, num=prompt("Entra un número entre 0 i 9","");
   num=parseInt(num);
   switch(num){
            case 0:
```

```
            resultat="Zero";
                 break;
              case 1:
            resultat="Un";
                 break;
              case 2:
            resultat="Dos";
                 break;
              case 3:
            resultat="Tres";
                 break;
              case 4:
            resultat="Quatre";
                 break;
              case 5:
            resultat="Cinc";
                 break;
              case 6:
            resultat="Sis";
                 break;
              case 7:
            resultat="Set";
                 break;
              case 8:
            resultat="Vuit";
                 break;
              case 9:
            resultat="Nou";
                 break;
         default:
            resultat="Error"
    }
    alert(resultat)
//-->
</SCRIPT>
```

*Executeu l'exemple i comproveu que funciona correctament, doneu-li també un valor erroni i comproveu que s'executa l'opció default (mostra error).*

Si analitzem el codi, tenim que primer de tot apareix entre parèntesis el nom de la variable que volem avaluar. A continuació va la resta de codi tancat entre claus.

Cada cas que volem avaluar apareix amb la paraula clau ***case*** seguida del valor que volem comparar i dos punts: (Exemple ***case 0:*** que equival a preguntar-se: la variable del switch val zero?). A continuació venen totes les instruccions que s'han d'executar si s'acompleix aquest cas. Aquestes instruccions no estan delimitades per claus, sinó que el navegador sap que el bloc s'acaba quan es troba una altra clàusula ***case, break***, ***default*** o les claus de tancament ***}.***

Quan un case s'acompleix, tota l'estructura a partir d'aquest punt es considera avaluada a true i això provoca que s'executin totes les instruccions associades al case que s'ha validat a true a més de totes les de les altres case que té ha per sota (tot i que no avaluïn a true la seva comparació amb la variable del switch).

Aquest comportament generalment no és el que nosaltres volem, sinó que la majoria de vegades necessitem que en el moment que s'acompleixi un case, s'executin les seves instruccions associades i es doni per acabada tota l'estructura switch. Això ho aconseguirem finalitzant el bloc d'instruccions amb la clàusula break.

Fixeu-vos que a l'exemple tots els casos els he acabat amb la clàusula ***break***. Quan el navegador es troba amb un break dóna per finalitzada tota l'expressió switch.

La clàusula ***default*** és opcional i engloba les instruccions que volem que s'executin si cap dels ***case*** s'ha validat a true. Fixeu-vos que no cal que acabi amb ***break***, de fet, com que sempre apareix en últim lloc, quan s'acaba el default, també s'acaba l'estructura switch.

**L'operador condicional ternari**

Moltes vegades ens troben en situacions de programació molt freqüents, en les què solament volem assignar un valor o un altre a una variable depenent si s'acompleix o no una condició, aquesta situació la podríem resoldre fàcilment amb una simple instrucció ***if-else.***

```
<SCRIPT type="text/javascript">
<!--
   var resultat, iva;
   iva=prompt("Quant has pagat d'IVA?","");
   if (iva<=4)
      resultat="IVA reduït";
   else
      resultat="IVA no reduït";
   alert (resultat);
//-->
</SCRIPT>
```

JavaScript simplifica aquesta operació posant a la nostra disposició l'operador ternari, amb el que podem reduir la sentència if-else a una sola línia:

```
<SCRIPT type="text/javascript">
<!--
   var resultat, iva;
   iva=prompt("Quant has pagat d'IVA?","");
   resultat = (iva<=4)?"IVA reduït":"IVA no reduït";
   alert (resultat);
//-->
</SCRIPT>
```

L'operador ternari està format per tres parts: la condició que acaba amb un interrogant, el valor que es retornarà en cas que la condició sigui certa i , separat per dos punts : el valor que es retornarà si la condició és falsa.

```
(condició)?ValorCert:ValorFals;
```

Com sempre passa amb les expressions on apareixen comparacions, la condició pot ser qualsevol valor o expressió que retorni un valor boolean.

*Feu el bloc 2 d' exercicis. Punt 2, del full de pràctiques (pàg. 1).*

## 2.7 Bucles

Els bucles són estructures de control que ens permeten fer que un bloc de codi es repeteixi un número determinat o indeterminat de cops.

A l'hora de fer el programa, podem saber el número exacte de cops que s'han de repetir les instruccions que conformen el cos del bucle, o sigui saben el número de voltes que ha de donar.

En altres casos no saben el número exacte de cops que s'ha d'executar el bucle, sinó que això be determinat pel resultat d'una condició que s'avalua al començament o al final del bucle per decidir si es dona una volta més o no.

### 2.7.1 El bucle for

És el tipus de bucle que farem servir quan saben per endavant el número exacte de voltes que volem que doni.

Imagineu que volem mostrar la taula de multiplicar (de l'1 al 10) d'un número que ens entrin des del teclat. Suposem que ens han entrat el número 2, sabem que haurem de fer un bucle que doni

10 voltes, i en cadascuna d'elles mostrar: 2x1=2, 2x2=4, 2x3=6 ... 2x10=20. Com podeu veure donem en total 10 voltes. El codi de l'exemple és el següent:

```
<SCRIPT type="text/javascript">
<!--
   var contador, num=prompt("Quina taula vols repassar","");
   for(comptador=1;comptador<10;comptador++){
      document.write(comptador+"x"+num+"="+comptador*num+"<BR />");
   }
//-->
</SCRIPT>
```

A la figura 2.22 podeu veure l'esquema de funcionament del bucle for.

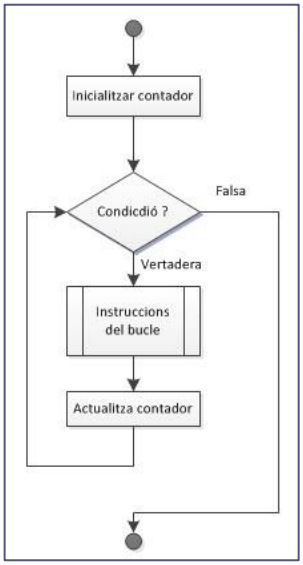



***Figura 2.22*** *Bucle for*

Ara veurem l'explicació de l'exemple de la taula de multiplicar. Primer de tot veurem que fa la línia ***for(comptador=1;comptador<=10;comptador++)***, com podeu veure està formada per tres parts separades per punt i coma. Primer de tot li assignem a la variable comptador el valor inicial 1 ***comptador=1;***. La següent part indica la condició que s'ha de complir perquè es repeteixi el bucle ***comptador<=10,*** finalment tenim el increment que se li ha d'aplicar a comptador al final de cada volta del bucle ***comptador++.***

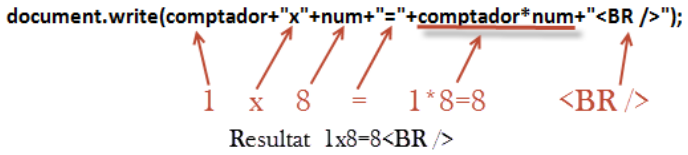
El cos del bucle pot semblar molt complicat, però només es tracta d'una expressió on apareixen vàries concatenacions.

***document.write(comptador+"x"+num+"="+comptador*num+"<BR />");***

Primer que res apareix ***document.write()***, tot i que el tractarem més endavant, podem avançar que ***document*** és un objecte que representa el document o la pàgina que estem mostrant i que inclou un mètode ***write*** que ens permet escriure directament a la pàgina. En altres paraules, si per exemple escrivim ***document.write("Hola"),*** li estem dient al navegador que escriu Hola a la pàgina que està mostrant.

Per seguir l'exemple, suposeu que volem repassar la taula del 8, per tant a la variable ***num*** tindrem un 8.

El cos del nostre bucle està format per una línia que es repetirà 10 vagades. La primera vegada comptador val 1 i num valdria 8, per tant el resultat de les concatenacions donarà la següent sortida:




Quan el bucle acaba la primera volta, la variable comptador s'incrementa en 1 (***comptador++***) i passa a valdre 2, com que la condició encara és vertadera (***2<=10***) el cos del bucle es torna a executar, però la variable del comptador ara val 2 amb la qual cosa la nova sortida ara és: ***2x8=16<BR />***. I així s'anirà executant successivament, amb els valors de comptador igual a 3, 4, 5... fins que arribi a 11, en aquest moment la condició de la instrucció ***for*** serà falsa (***11<=10***) i es donarà per acabada tota l'estructura. Fixeu-vos que quan la variable comptador val 11 el bucle ja no s'executa, per tant l'últim valor que hem tractat ha estat el 10.

Observeu també la utilització que hem fet de l'etiqueta de HTML ***<BR />***. La instrucció ***document.write*** primer escriu al document de sortida el que li hem indicat entre els parèntesis, posteriorment el navegador a l'hora de maquetar o "renderitzar" la pàgina ja s'encarregarà d'interpretar adequadament totes les possible marques HTML que puguin haver.

Com anirem veient al llarg del curs, el bucle for és un dels que més s'utilitzen en programació per la seva facilitat a l'hora de recórrer col·leccions o matrius.

Igual que hem fet bucles com l'anterior, en que el comptador avança cap al davant, de l'1 fins al 10, també podem fer bucles que vagin cap enrere. Només hem de treballar amb els paràmetres de la instrucció for.

```
for (comptador=10;comptador>=1;comptador--){
     Cos del bucle;
}
```

En aquest cas hem fet un bucle que va cap enrere, comencem amb el valor 10 i li anem restant 1 a cada volta del bucle, mentre s'acompleixi que el comptador és més gran o igual que 1.

També podem fer que el comptador en comptes de fer increments o decrements d'una unitat ho faci del valor que nosaltres vulguem. El següent exemple, mostra tots els números parells compresos entre 0 i 20.

```
<SCRIPT type="text/javascript">
<!--
   for(c=0;c<=20;c+=2){
        document.write(c + " ");//Concatenem un espai per separar els números
       }
 //-->
</SCRIPT>
```

### 2.7.2 Bucles for niats ("anidados")

Tal com passava amb els blocs d'instruccions if, que podien contenir a dins seu altres instruccions if. El cos del bucle d'una instrucció for per la seva banda també pot contenir un altre bloc for, donant lloc al que es coneix en programació com a *bucles niats*.

Imagineu que ara volem mostrar en una pàgina totes les taules de multiplicar de l'1 fins al 10. Ja sabem com fer servir un bucle for per generar la taula d'un número. Doncs bé, si fem que aquest codi sigui el cos d'un altre bucle for més extern que doni 10 voltes (d'1 a 10), ja tindrem resolt el problema i tindrem un programa que mostri totes les taules que volíem.

```
<SCRIPT type="text/javascript">
<!--
   for(c=1;c<=10;c++){
            for(c2=1;c2<=10;c2++){
              document.write(c+"x"+c2+"="+c*c2+"<BR />");
            }
            document.write("<BR />");
```

```
        }
 //-->
</SCRIPT>
```

Tal com hem dit, el cos del bucle més extern és un altre bucle. Fixeu-vos també que hem fet servir dues variables diferents com a comptador de cada bucle (***c*** i ***c2***). Per cada volta que dona el bucle extern, l'interior s'executa completament i dóna 10 voltes.

Observeu la utilització de les variables ***c*** i ***c2*** en la instrucció ***document.write()***.

La instrucció ***document.write("<BR />")***; introdueix un salt de línia addicional cada vegada que canviem de la taula d'un número a la del següent.

*Executeu l'exemple i comproveu el seu funcionament.*

### 2.7.3 Bucles que s'executen un número indefinit de vegades. Bucle while.

No sempre sabrem per anticipat el número de voltes que ha de donar un bucle. Per exemple: imagineu que volem fer un programa que vagi demanant números a l'usuari fins que escrigui un negatiu i en aquest moment que mostri la suma de tots els números que ha entrat (sense tenir en compte el negatiu). Com podeu imaginar, ara no podem fer servir un bucle ***for***, perquè no saben quants números voldran entrar. Per aquests casos tenim els bucles del tipus ***while***:

```
<SCRIPT type="text/javascript">
<!--
    var suma=0, num= parseInt(prompt("Entra un número \nEscriu un   »»
                                    negatiu per finalitzar",""));
    while(num>=0){
        suma+=num;
        num=parseInt(prompt("Entra un número \nEscriu un negatiu per »»
                                     finalitzar",""));
    }
    alert ("Els números que has entrat sumen "+suma)
 //-->
</SCRIPT>
```

**Atenció:** Al codi anterior apareixen dues instruccions molt llargues que en aquest manual no caben en una sola línia, per tant les he hagut de trencar en dos, he fet servir els caràcters **»»** per indicar-ho, quan us trobeu línies de codi semblant a aquestes recordeu les haureu d'escriure en una sola línia.

La instrucció ***while*** funciona de la següent manera, el cos del bucle, que apareix delimitat per claus, s'executarà tantes vagades com calgui, mentre la condició sigui vertadera. Fixeu-vos en el seu ordinograma (figura 2.23).

En aquests tipus de bucles hem de prestar atenció i assegurar-nos que, dins de les instruccions que conformen el cos del bucle, aparegui alguna que faci que en algun moment de l'execució la condició del bucle deixi de ser vertadera. Si no es dóna aquest cas el bucle no acabarà mai i haurem entrat en un bucle infinit. A tots els efectes per l'usuari el programa **"s'haurà penjat".**

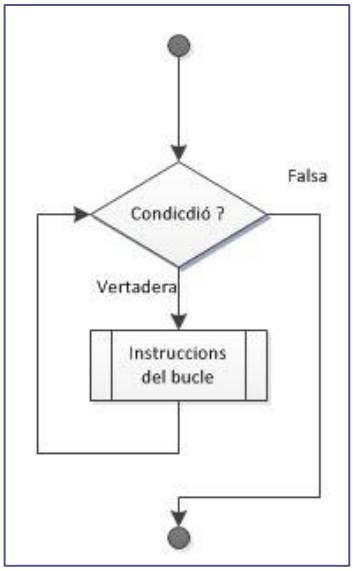



***Figura 2.23.*** *Bucle while*

### 2.7.4 Bucles do ... while

En l'exemple anterior, podria donar-se el cas que el bucle while no s'executés mai. Si d'entrada la condició s'avalua a ***false***, el bucle no s'executarà ni tan sols la primera vegada. En la nostra vida de programadors, ens trobarem en situacions en les que ens interessarà que, com a mínim, el bucle s'executi la primera vegada i després, depenent de la condició, continuarà o no. Per aquestes situacions el llenguatge ens proporciona una variant del bucle ***while***:

```
<SCRIPT type="text/javascript">
<!--
    var suma=0, num=0;
    do{
        suma+=num;
        num=parseInt(prompt("Entra un número \nEscriu un »»
                             negatiu per finalitzar",""));
    }while(num>=0)
    alert ("Els números que has entrat sumen "+suma)
 //-->
</SCRIPT>
```

Aquesta forma del bucle comença amb una clàusula ***do*** seguida del cos del bucle, com sempre delimitat per claus ***{*** i ***},*** i finalitza amb la paraula clau ***while*** seguida de la condició entre parèntesis. Com podeu veure, la condició la hem traslladat al final del bucle, es a dir, primer sempre s'executa el cos del bucle, sense cap condició, després avaluem la condició per determinar si es torna a repetir o no. Per tant, sabem que com a mínim el cos del bucle s'executarà una vegada.

Si compareu el codi amb el de l'exemple del punt anterior, veureu que en la forma anterior demanàvem dues vegades la introducció d'un numero, una abans de començar el bucle, perquè la variable ***num*** tingués un valor al fer la primera validació i posteriorment repetien l'operació dins del cos del bucle per anar llegint els valors successius.

Amb aquesta nova versió hem evitat duplicar la instrucció fent-la dins del cos del bucle i fent la validació de la condició al final del mateix.

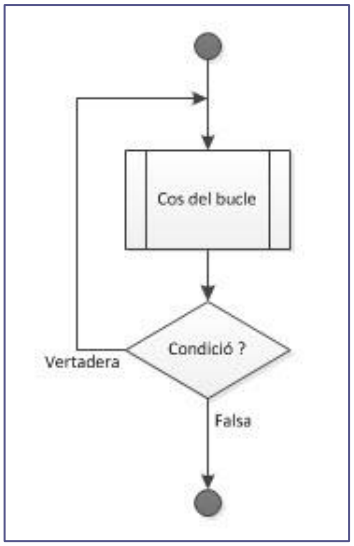



***Figura 2.24.*** *Bucle do .. while*

### 2.7.5 Alterar els cicles d'un bucle

Quan s'executa un bucle, el més normal és que les instruccions que formen el cos del mateix s'executin completament en cada iteració. Malgrat això, en certes ocasions ens interessarà que el bucle finalitzi prematurament si s'acompleix o no alguna condició. En altres ocasions ens interessarà interrompre el cicle actual del bucle i forçar a que s'iniciï un de nou des del començament. Per aconseguir aquest comportament tenim a la nostra disposició les instruccions ***break*** i ***continue*** respectivament.

En primer lloc veurem un exemple de ***break***. Es tracta de construir un programa que demani 1000 números a l'usuari i que en acabar mostri la seva suma. Com que probablement l'usuari no tindrà paciència per entrar els mil números, li proporcionarem una forma prematura de finalitzar, que consistirà en escriure el número 0.

```
<SCRIPT type="text/javascript">
<!--
 var suma=0, num=0;
 for(c=1;c<=1000;c++){
     num=prompt("Entra el número "+c+" de 1000\nEntra un 0 per finalitzar","");
     num=parseInt(num);
     if(num==0) break;
     suma+=num;
     }
     alert ("Els números que has entrat sumen "+suma)
 //-->
</SCRIPT>
```

Si explorem el codi, veurem que ens demanarà 1000 números, ja que estem dins un bucle ***for*** que donarà mil voltes i cadascuna d'elles ens demana un número. Però, fixeu-vos que dins del bucle hi ha una instrucció ***if*** que en cada volta s'encarrega de comprovar si hem entrat un zero, en cas afirmatiu executa la instrucció ***break***. Aquests instrucció dóna per acabat el bucle, sense tant sols finalitzar les instruccions que encara manquen per executar-se de dins el cos del bucle.

Veurem ara un exemple de la clàusula ***continue***. Volem fer un programa que demani 10 números i en acabar que mostri la suma només dels positius.

```
<SCRIPT type="text/javascript">
<!--
   var suma=0, num=0;
   for(c=1;c<=10;c++){
     num=prompt("Entra el número "+c,"");
```

```
      num=parseInt(num);
      if(num<0) continue;
    suma+=num;
    }
     alert ("Els números positius que has entrat sumen "+suma)
  //-->
</SCRIPT>
```

Fixeu-vos que ara, estem en un bucle que donarà 10 voltes i per tant demanarà 10 números. La diferència és que ara, en cada volta mirem si el número és negatiu (***num<0***), en cas afirmatiu ometem la resta del bucle i forcem a que comenci la següent iteració, amb la instrucció ***continue.*** D'aquesta manera els números negatius no se sumem mai al resultat ***suma***.

*Feu el bloc 3 d' exercicis. Punt 2, del full de pràctiques (pàg. 2).*

## 2.8 Comentaris al codi.

En JavaScript, com en tots els llenguatges de programació, es poden incloure comentaris per documentar el codi. Els comentaris són ignorats pel navegador, la seva única utilitat és facilitar la lectura del codi al programador. Donat que en l'actualitat la majoria de projectes es realitzen en equip, és una bona pràctica comentar el codi que estem generant, tant per nosaltres mateixos, per si l'hem de tornar a modificar més endavant, com per altres programadors que hagin de treballar en ell en el futur.

Tenim dues maneres d'incloure comentaris en el codi. La primera és mitjançant la doble barra (***//***), en aquest cas, el navegador entendrà que tot el que apareix des de la doble barra fins al final de la línia és un comentari i no el tindrà en compte.

La segona forma és incloure el comentari entre barra asterisc (***/****) al inici i asterisc barra (****/***) al final. Aquesta forma és ideal per tenir blocs de més d'una línia de comentaris. Observeu l'exemple:

```
<SCRIPT type="text/javascript">
<!--
     /* Això és un bloc de comentaris que serà ignorat per
        JavaScript. Com podeu veure es tracta d'un bloc
        format per vàries l'inies */

     alert ("Hola manola"); //Aquest és un altre comentari

           //-->
</SCRIPT>
```

Recordeu que el codi de JavaScript va incrustat en la pròpia pàgina HTML i això te una sèrie de implicacions; per exemple, l'usuari pot veure el vostre codi, inclosos els comentaris, només cal que accedeixi a l'opció *veure el codi font de la pàgina* (tots els navegadors ho permeten), per tant sigueu prudents en els comentaris que feu servir a les vostres pàgines.

Aquesta característica fa que els programadors de JavaScript siguem els programadors més generosos del mon. A diferència de la resta de programadors que distribueixen els seus programes compilats en forma d'executables que són inintel·ligibles pel ser humà, el nostre treball queda a la vista de qualsevol persona que es descarregui la nostra pàgina web.

Una problema amb que ens podem trobar, és que la persona que mira la nostra pàgina treballi amb un navegador molt antic que no suporti JavaScript, o que el tingui desactivat. En aquests casos, com el seu navegador no entén l'etiqueta <SCRIPT> la ignorarà però, mostrarà tot el codi JavaScript com si fos text normal.

Per evitar aquestes situacions farem servir una tècnica que consisteix en delimitar tot el nostre codi JavaScript entre comentaris de HTML. Observeu a l'exemple anterior que la primera línia que apareix després de l'etiqueta <SCRIPT...> és un inici de comentari de HTML (***<!--***) i la última abans de </SCRIPT> (***//-->***) inclou el final de comentari de HTML... D'aquesta manera, els navegadors

que no entenen JavaScript quan es trobin amb les etiquetes <SCRIPT> i </SCRIPT> les ignoraran perquè no les entén com a etiquetes valides de HTML, a continuació trobaran el codi JavaScript, però està delimitat entre ***<!--*** i ***//-->*** que són els delimitadors de comentaris de HTML amb la qual cosa també serà ignorat.

Per contra, els navegadors moderns estan ensinistrats per ignorar la línia d'inici de comentari de HTML ( ***<!-- )*** que aparegui després de <SCRIPT>. Pel que fa a la línia de final de comentari, ***//-->***, no hi ha cap problema, ja que per JavaScript es tracta d'un comentari, fixeu-vos que comença amb dues barres (//) i per HTML es tracta de la marca de final de comentari ja que acaba en ***-->***.

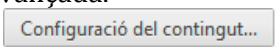
Relacionat amb aquest tema, JavaScript també ens proporciona unes etiquetes que ens ajuden en aquests casos, són ***<noscript>*** i ***</noscript>***. Si un navegador modern, que sap interpretar JavaScript, es troba aquestes etiquetes dins d'una pàgina, les ignorarà, així com tot el contingut que aparegui entre elles. Per contra si s'ho troba un navegador antic que no entén JavaScript, tampoc entendrà aquestes etiquetes, amb la qual cosa ignorarà les etiquetes, però no el contingut que aparegui entre elles. Així podem aprofitar aquesta característica per mostrar un missatge a l'usuari, o fins i tot, si volem, podríem fer una versió alternativa de la pàgina per navegadors sense JavaScript. Exemple:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
              "http://www.w3.org/TR/html4/strict.dtd">
<HTML>
       <HEAD>
              <TITLE>Exemple</TITLE>
       </HEAD>
<BODY>
  <SCRIPT type="text/javascript">
<!--
                  v1= "123";
                  v2 = 456;
                  alert (v1 + v2);
 //-->
  </SCRIPT>

<NOSCRIPT>
     <h2 style="color:red"> El teu navegador no suporta JavaScript,
               si us plau, mira de actualitzar-lo amb una versió més
               actual (és gratuït)</h2>
</NOSCRIPT>

  </BODY>
</HTML>
```

Si voleu comprovar que efectivament funciona, aneu a les opcions de configuració del navegador i desactiveu JavaScript, tot seguit proveu la pàgina. Si esteu fent servir Chrome:

Feu clic a la icona d'opcions  de la barra d'eines del navegador.
Seleccioneu **Configuració**.
Feu clic a Mostra la configuració avançada.
A l'apartat Privadesa clica el botó 

A la finestra que apareix selecciona l'opció per desactivar JavaScript

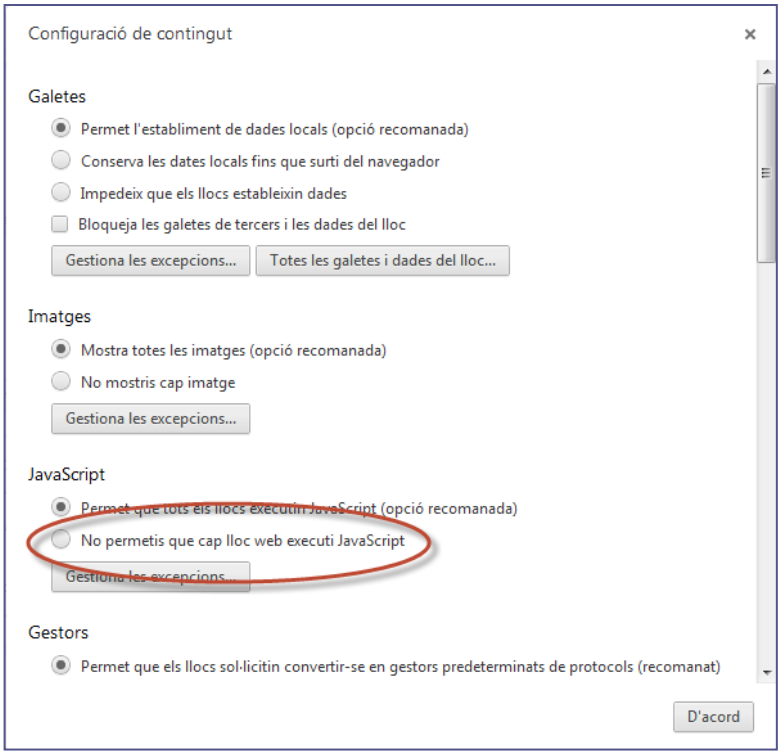



***Figura 2.25.*** *Desactivar JavaScript en Chrome*

La resta de navegadors també inclouen l'opció per desactivar JavaScript. No t'oblidis de tornar a activar aquesta opció una vegada fetes les proves, sinó no podràs continuar amb els exemples del curs.