Programació en el costat del client: llenguatges script en els navegadors

Vicent Moncho Mas
Jordi Ustrell Garrigós

PID_00220470

UOC
Universitat Oberta de Catalunya

Índex

1. Introducció..... 5
1.1. Internet 5
1.1.1. Els inicis 5
1.1.2. Components del Web 6
1.2. JavaScript 7
1.2.1. L'origen de JavaScript 7
1.2.2. JavaScript i la programació de scripts CGI 9

2. Característiques bàsiques, ubicació de codi i primers passos. 12
2.1. Característiques bàsiques 12
2.2. Ubicació del codi 12
2.2.1. JavaScript incrustat en el codi HTML 13
2.2.2. Inclusió de fitxers .js en les pàgines HTML 15
2.2.3. Quan el navegador no suporta JavaScript 16
2.3. Primers passos 16
2.3.1. El mètode *write*..... 17
2.3.2. Inclusió d'etiquetes 19
2.3.3. Detectar errors durant el desenvolupament 20
2.3.4. Depuració del codi 20

3. Variables, operadors i estructures..... 23
3.1. Paraules reservades 23
3.2. Dades 24
3.3. Literals 24
3.4. Comentaris 25
3.5. Variables 25
3.5.1. Declaració i assignació de variables 25
3.5.2. Àmbit de les variables 26
3.5.3. Conversions de tipus 27
3.6. Operadors 28
3.6.1. Operadors aritmètics 28
3.6.2. Operadors d'assignació 29
3.6.3. Operadors de comparació 30
3.6.4. Operadors lògics o booleans 31
3.6.5. Operadors de bits 31
3.6.6. L'operador + ... 32
3.6.7. L'operador condicional ?: 33
3.6.8. Signes de puntuació 33
3.6.9. Operadors per al treball amb objectes 34
3.6.10. Precedència d'operadors 34
3.7. Estructures de control condicionals 35
3.7.1. Estructura de control *if*..... 36

3.7.2. L'estructura de control *if ... else* 36
3.7.3. L'estructura de control *if ... else if* 37
3.7.4. L'estructura de control *switch* 39
3.8. Estructures de control iteratives 40
3.8.1. L'estructura de control *for* 41
3.8.2. L'estructura de control *while* 42
3.8.3. L'estructura de control *do ... while* 43
3.8.4. La sentència *break* 44
3.8.5. La sentència *continue* 45
3.9. La sentència *with* 45

4. Funcions i objectes bàsics 47
4.1. Definició d'una funció 47
4.1.1. Definició de la funció en JavaScript 47
4.1.2. Ubicació en el document HTML 48
4.1.3. Crida i execució de les funcions 49
4.1.4. Ús dels paràmetres de la funció 50
4.1.5. Propietats de les funcions 52
4.1.6. Retorn de la funció 53
4.1.7. Funcions recursives 55
4.1.8. Funcions locals 56
4.1.9. Funcions com a objectes 57
4.2. Funcions predefinides 57
4.3. Introducció a l'objecte Array 62
4.3.1. Crear un *array* 62
4.3.2. Accés als elements 63
4.3.3. Afegir i modificar elements en un *array* 64
4.3.4. Eliminar elements d'un *array* 64
4.3.5. La propietat *length* 64

1. Introducció

1.1. Internet

1.1.1. Els inicis

Al final de la dècada de 1980, Internet era el magatzem de dades més gran que mai no s'havia imaginat. Tanmateix, la informació era molt difícil de localitzar i, per tant, d'usar; faltava la tecnologia que permetés als usuaris accedir a aquesta informació en creixement constant. Cap a 1990, el CERN (Consell Europeu per a la Recerca Nuclear) va considerar que calia accedir a la informació generada pels seus membres, per a conèixer els avenços i servir-se'n. Va ser llavors quan Tim Berners-Lee va presentar el Projecte Web, que consistia a crear una "teranyina" d'informació.

Berners-Lee va observar que la majoria de les persones, per a ordenar gràficament la informació, utilitzaven rodones i fletxes, que es podien representar com a nodes i enllaços per a connectar la informació que ja hi havia. La seva proposta incloïa, a més, la creació de plataformes múltiples per a unificar la informació que contenia Internet, informació que hi havia en diversos formats no compatibles, programes diferents, protocols diferents, etc.

La idea de Berners-Lee d'organitzar la informació en una teranyina utilitzant hipertextos, i també l'ús de navegadors amb interfícies gràfiques, va accelerar el desenvolupament d'Internet.

World Wide Web[1] significa 'xarxa' o 'teranyina global'. El 1991, es van instal·lar les primeres connexions del WWW per a ús intern del CERN i, aquell mateix any, el sistema WWW va passar a Internet.

(1)Abreujat, *Web*; escrit també *WWW* o fins i tot *W3*.

Des de llavors, accedir al World Wide Web és bastant senzill: només fa falta un equip connectat a Internet. La navegació es basa a fer un clic als enllaços.

El Web utilitza un model client-servidor en què l'usuari executa una aplicació "client" des de l'ordinador. L'usuari comença amb el document de benvinguda del servidor per defecte, i pot consultar la informació que hi ha emmagatzemada en servidors remots.

En resum, el **Web** es pot descriure com un sistema hipermèdia global que, per mitjà de diversos protocols, permet a l'usuari elaborar i presentar hipertextos complexos, amb enllaços a diversos documents que són físicament en altres servidors. Aquests documents són textos, hipertextos, arxius –imatges, so i animacions– o resultats de cerques en bases de dades.

1.1.2. Components del Web

Els tres components bàsics del Web són els següents:

- El sistema d'adreçament URL (*Uniform Resource Locator*), que permet identificar documents o arxius.

- El protocol HTTP (*HyperText Transfer Protocol*), que utilitzen els servidors i els clients WWW per a navegar per mitjà d'enllaços entre documents.

- El llenguatge HTML (*HyperText Markup Language*), que es fa servir per a presentar la informació.

1) El sistema d'adreçament URL

El sistema URL és format per una cadena de caràcters que assigna una adreça única a cadascun dels recursos d'informació disponibles a Internet. Hi ha un únic URL per a cada lloc de cadascun dels documents del World Wide Web. Un URL té la sintaxi següent:

```
protocol://servidor[:port][/ruta][/cadena_de_consulta]
```

en la qual els protocols més habituals són HTTP i FTP, mentre que el servidor és especificat per una adreça d'Internet o simplement pel nom de màquina al qual es vol accedir. El port és una característica del protocol de transport utilitzat que permet que un servidor/protocol faci servir diferents ports o vies de comunicació.

La part ruta és opcional i, en cas que no s'especifiqui, s'utilitza la que s'ha definit per defecte. Per acabar, la cadena de consulta serveix per a passar informació al servidor des del client.

2) El protocol HTTP

L'HTTP (*HyperText Transfer Protocol*) és el protocol que defineix la sintaxi i la semàntica de les transaccions d'informació a Internet. Es basa en un flux de petició/resposta entre el client i el servidor. El client que fa la petició es coneix com a *agent de l'usuari*, mentre que, de la informació transmesa, se'n diu

recurs i s'identifica mitjançant un URL. Els recursos són els arxius, el resultat de l'execució d'un programa, una consulta a una base de dades, la traducció automàtica d'un document, etc.

Una de les limitacions de l'HTTP és que es tracta d'un protocol sense estat, és a dir, que no guarda cap informació de les connexions anteriors (no té memòria sobre la conversa que s'ha fet), la qual cosa constitueix una limitació en la programació d'aplicacions web, com es veurà més endavant.

3) El llenguatge HTML

HTML són les sigles d'*HyperText Markup Language*, de la qual cosa es dedueix que es tracta d'un llenguatge de marques o etiquetes l'objectiu del qual és definir el contingut o l'estructura d'un document, però no del format o l'aparença. No es tracta d'un llenguatge de programació, ja que en HTML no hi ha les sentències de control de flux característiques de qualsevol llenguatge de programació.

La primera descripció de l'HTML disponible públicament va ser un document anomenat *HTML Tags*, que va publicar per primera vegada a Internet Tim Berners-Lee el 1991. En aquest document es descrivien vint-i-dos elements, que comprenien el disseny inicial i relativament simple de l'HTML.

Berners-Lee considerava l'HTML com una ampliació de l'SGML, però no va ser reconeguda formalment com a tal fins que a mitjan 1993 l'IETF va publicar una primera proposició per a una especificació de l'HTML.

Web recomanat

L'especificació de l'estàndard del llenguatge es pot consultar en la web del W3C (www.w3.org/html/).

Tots els exemples que s'utilitzaran al llarg dels materials de l'assignatura es basen en l'estàndard HTML 4, però són vàlids (possiblement amb petites modificacions) amb XHTML i HTML 5.

1.2. JavaScript

1.2.1. L'origen de JavaScript

L'origen de JavaScript es remunta al començament de la dècada de 1990, quan es comencen a executar les primeres aplicacions web desenvolupades en HTML. En aquestes aplicacions, el component principal eren els formularis que s'encarregaven de transmetre la informació des de l'usuari fins al servidor web.

Com que la tecnologia de connexió a la xarxa Internet es basava en línies analògiques amb mòdems que tenien una velocitat màxima de 28,8 kbps, es va fer imprescindible optimitzar els processos que estructuraven la comunicació.

D'aquesta manera va sorgir la necessitat d'un llenguatge de programació que s'executés en el navegador de l'usuari. L'objectiu perseguit era que el llenguatge fes certes comprovacions en l'equip client mateix, de manera que s'aconseguia un estalvi en el temps del procés, ja que si les validacions es feien en el servidor s'havia d'afegir el temps necessari en la transmissió de la informació en els dos sentits.

Brendan Eich, un programador que treballava a Netscape, va pensar que podia solucionar aquest problema adaptant altres tecnologies existents (com l'ScriptEase) en el navegador Netscape Navigator 2.0, que s'havia de comercialitzar el 1995. Inicialment, Eich va anomenar el seu llenguatge *LiveScript*.

Arran d'una aliança firmada per Netscape amb Sun Microsystems per a desenvolupar el llenguatge de programació nou i just abans de la comercialització, Netscape va decidir canviar el nom de *LiveScript* pel de ***JavaScript***, que va ser presentat oficialment el 4 de desembre de 1995. La raó del canvi de nom va ser exclusivament per motius de màrqueting, ja que *Java* era la paraula de moda en el món informàtic i d'Internet de l'època.

La **primera versió de JavaScript** va ser un èxit complet i el Netscape Navigator 3.0 ja incorporava la versió següent del llenguatge, la versió 1.1. Alhora, Microsoft va comercialitzar la seva pròpia adaptació de JavaScript, que va anomenar *JScript* a partir del seu navegador Internet Explorer 3 l'agost de 1996.

Aquest camí paral·lel entre les dues grans companyies va provocar divergències en la implementació de l'HTML i en la interpretació de JavaScript, perquè aquest està directament relacionat amb l'estructura del DOM del navegador. Per a evitar aquesta situació, Netscape va decidir estandarditzar el llenguatge JavaScript i amb aquest objectiu va enviar el 1997 l'especificació JavaScript 1.1 a l'organisme ECMA[2].

[(2)]European Computer Manufacturers Association

L'ECMA va crear el **comitè TC39**, que pretenia "estandarditzar un llenguatge script multiplataforma i independent de qualsevol empresa". El primer estàndard que va crear el comitè TC39 es va anomenar *ECMA-262*, en el qual es va definir per primera vegada el llenguatge ECMAScript. D'altra banda, l'Organització Internacional per a l'Estandardització (ISO) va adoptar **l'estàndard ECMA-262** mitjançant la seva comissió IEC, i va donar lloc a l'estàndard ISO/IEC-16262.

Web recomanat

L'especificació d'ECMA-262 es pot trobar a Estàndard ECMA-262: http://www.ecma-international.org/publications/standards/Ecma-262.htm.

La taula següent (taula 1) mostra l'evolució del llenguatge JavaScript i com es va anar incorporant als navegadors principals:

Taula 1

Versió	Data de comercialització	Equivalència	Netscape	Mozilla Firefox	Internet Explorer	Opera	Safari	Google Chrome
1.0	Març de 1996		2.0		3.0			

Font: es.wikipedia.org/wiki/JavaScript.

Versió	Data de comercialització	Equivalència	Netscape	Mozilla Firefox	Internet Explorer	Opera	Safari	Google Chrome
1.1	Agost de 1996		3.0					
1.2	Juny de 1997		4.0-4.05					
1.3	Octubre de 1998	Especificació ECMA-262 1a. / especificació ECMA-262 2a.	4.06-4.7x		4.0			
1.4			Netscape Server					
1.5	Novembre de 2000	Especificació ECMA-262 3a.	6.0	1.0	5.5 (JScript 5.5) 6 (JScript 5.6) 7 (JScript 5.7) 8 (JScript 5.8)	6.0	3.0-5	1.0-10.0.666
1.6	Novembre de 2005	1.5 + arrays extres + arrays i strings genèriques + E4X		1.5				
1.7	Octubre de 2006	1.6 + generadors de Python + iteradors + let		2.0				
1.8	Juny de 2008	1.7 + generador d'expressions + expressions closure		3.0		11.50		
1.8.1.		1.8 + suport natiu JSON + actualitzacions menors		3.5				
1.8.2.	Juny de 2009	1.8.1 + actualitzacions menors		3.6				
1.8.5.	Juliol de 2010	1.8.2 + compatibilitat amb ECMAScript 5		4	9	11.60		

Font: es.wikipedia.org/wiki/JavaScript.

1.2.2. JavaScript i la programació de scripts CGI

Si bé l'HTML ha complert els requisits per als quals va ser dissenyat, han anat apareixent necessitats noves dins d'aquest camp. Es tractava de fer més flexible la possibilitat de dissenyar pàgines, menys tediosa la interactivitat amb l'usuari o facilitar l'accés remot a dades.

La idea de fons de la programació CGI (*Common Gateway Interface*), per posar un exemple, és que construeixi el document HTML corresponent a un enllaç d'hipertext en el mateix moment en què es fa un clic a l'enllaç. El mecanisme és el següent:

- El client indica un URL que es correspon amb un fitxer que s'executa en el servidor.

- La resposta a aquesta execució és la devolució del servidor d'un document HTML.

Els llenguatges dels servidors que utilitzen aquest mecanisme són coneguts com a ***scripts CGI***; dos dels scripts més coneguts són **Perl** i **PHP**.

La interactivitat que proporciona el llenguatge JavaScript és diferent de la que ofereixen els programes CGI, perquè el servidor ja no hi té cap paper una vegada que el document s'ha carregat al navegador del client. Es tracta de dues aproximacions diferents, encara que complementàries, a un problema semblant.

En el cas dels llenguatges de servidor és imprescindible que hi hagi el servidor que executa el programa, mentre que en el cas de JavaScript s'executa en el navegador mateix i això permet fer aplicacions web distribuïdes en diferents suports que es poden executar en qualsevol sistema sense haver de tenir un servidor web.

Aquestes dues alternatives tecnològiques fan que el dissenyador tingui la capacitat de repartir la feina entre servidor i client. Com a regla general, els scripts de client s'han de fer servir per al següent:

- Validar l'entrada d'usuari en un formulari.
- Sol·licitar a l'usuari la confirmació abans d'executar una acció i mostrar finestres que adverteixin d'errors o perills.
- Processar les dades rebudes del servidor (sumes, mitjanes, etc.).
- Introduir sentències condicionals en el codi HTML.
- Portar a terme, en general, qualsevol acció que no requereixi informació del servidor.

Per la seva banda, els scripts de servidor s'han de fer servir per al següent:

- Mantenir informació d'una sèrie d'accessos de clients.
- Mantenir dades compartides entre diversos clients o aplicacions.
- Accedir a bases de dades o fitxers del servidor.
- Trucar a biblioteques externes del servidor.
- Personalitzar miniaplicacions de Java dinàmicament; per exemple, visualitzar dades usant una miniaplicació de Java.

L'avantatge més bo de JavaScript davant del CGI és que proporciona independència del servidor, mentre que, paradoxalment, el desavantatge més gros és la dependència del client. D'altra banda, el fet de traslladar el codi al client fa que cada esdeveniment que s'ha de processar no requereixi comunicació per mitjà de la Xarxa tal com passa amb el CGI. Això permet un gran estalvi de temps, sobretot en el processament de formularis. Concretament, és possible

fer el filtratge de la validesa de les dades introduïdes en els camps sense que les avaluï un CGI. Es poden generar respostes i indicacions dinàmiques abans d'enviar el formulari fins que l'usuari l'empleni correctament.

No obstant això, hi ha alguns aspectes que s'han de centralitzar en un servidor i s'escapen de les possibilitats dels llenguatges de script. Per exemple, en el cas dels formularis, descarregaran el servidor de fer la feina del primer filtre, però al final hi haurà ineludiblement un programa CGI per a processar i emmagatzemar les dades definitives.

2. Característiques bàsiques, ubicació de codi i primers passos

2.1. Característiques bàsiques

Com s'ha comentat a l'apartat anterior, JavaScript és un llenguatge *script* que va ser desenvolupat per a incrementar les funcionalitats del llenguatge HTML.

Les seves **característiques** més importants són:

- **JavaScript és un llenguatge interpretat** (no requereix compilació): l'intèrpret és el propi navegador de l'usuari, que s'encarrega d'executar les sentències JavaScript.

- **JavaScript és un llenguatge basat en objectes**: el model d'objectes de JavaScript inclou els elements necessaris per a poder accedir a la informació continguda en una pàgina HTML i actuar sobre la interfície del navegador.

- **JavaScript és un llenguatge orientat a esdeveniments**: permet desenvolupar *scripts* que executin accions en resposta a un esdeveniment ocorregut en el navegador, per exemple, a un clic de ratolí sobre un gràfic.

Els llenguatges de *script* defineixen un estil de programació diferent dels llenguatges de propòsit general com C, C++ o Java. Són una alternativa que aporta agilitat i facilitat de programació, amb la qual cosa augmenta la productivitat.

Un llenguatge de *script* és un llenguatge de programació de fàcil d'utilitzar. Els llenguatges de propòsit general es dissenyen per a construir estructures i algorismes que treballen amb els elements de més baix nivell de l'ordinador. En contrast, els llenguatges de *script* assumeixen l'existència de potents components que poden utilitzar obviant la complexitat del sistema.

Una altra característica dels llenguatges de *script* és que, normalment, són interpretats. Això, a més de implicar l'estalvi del procés de compilació en cada modificació del codi, permet realitzar aplicacions més flexibles quant a la implantació en diferents plataformes.

2.2. Ubicació del codi

Hi ha dues maneres bàsiques d'incloure codi JavaScript en les pàgines HTML.

- JavaScript incrustat en el codi HTML.

- JavaScript en un arxiu .js referenciat des de la pàgina HTML.

Totes dues maneres requereixen les etiquetes HTML: `<script>` i `</script>`

Aquestes etiquetes alerten el navegador que ha d'interpretar les línies de codi contingudes entre elles com a codi JavaScript. D'aquesta manera, per a introduir codi JavaScript en una pàgina HTML utilitzem la sintaxi següent:

```
<script type="text/javascript">
   //Sentències JavaScript
</script>
```

Un aspecte important a tenir en compte és que **JavaScript és sensible a les majúscules i minúscules**. Això significa que qualsevol paraula del llenguatge JavaScript ha d'estar escrita correctament amb les lletres en minúscules o majúscules, ja que si no, el navegador produirà un error.

2.2.1. JavaScript incrustat en el codi HTML

El codi JavaScript pot situar-se en el cos del document o bé en la capçalera d'aquest.

JavaScript en el cos del document:

Un *script* situat en el cos del document s'executa quan es carrega el cos o contingut de la pàgina. Per tant, les etiquetes `<script>` i `</script>` estaran situades en algun lloc entre les etiquetes `<body>` i `</body>` del document.

```
<html>
<head>
<title>El meu document</title>
</head>
<body>
<script  type="text/javascript">
   //Sentències JavaScript
</script>
</body>
</html>
```

L'exemple següent inclou un script situat en el cos del document. El codi JavaScript escriu un text en la pàgina HTML:

```
<html>
<head>
<title>Exemple</title>
```

```
</head>
<body>
<script  type="text/javascript">
   document.write("Això és un exemple. L'script és en el cos del document.");
</script>
</body>
</html>
```

JavaScript en la capçalera del document

Un script situat en la capçalera del document s'interpreta o executa abans que es carregui el contingut de la pàgina. La capçalera és el lloc apropiat per als scripts que no modifiquen els atributs de la pàgina i que s'executen en resposta a una acció de l'usuari. En la capçalera no es pot situar un script que modifiqui parts del document i s'executi en carregar la pàgina, ja que aquest codi s'executa abans de la càrrega del document i desconeix *a priori* els objectes continguts.

```
<html>
<head>
<title>Exemple</title>
<script  type="text/javascript">
   //Sentències JavaScript
</script>
</head>
<body>
</body>
</html>
```

L'exemple següent inclou un script situat en la capçalera del document. El codi JavaScript escriu un text en una finestra d'alerta des d'una funció que es crida quan es carrega la pàgina.

```
<html>
<head>
<title>Exemple</title>
<script  type="text/javascript">
function salutacio(){
   alert("Hola");
}
</script>
</head>
<body onLoad="salutacio()">
</body>
</html>
```

Càrrega d'una funció

Per a garantir que una funció es carrega abans de cridar-la, ha de posar-se en la capçalera del document.

En una pàgina HTML es poden usar tantes etiquetes `<script></script>` com siguin necessàries. Per tant, és possible situar scripts en la capçalera i en el cos d'un mateix document.

```
<html>
<head>
<title>Exemple</title>
<script  type="text/javascript">
   //Sentències JavaScript
</script>
</head>
<body>
<script  type="text/javascript">
   //Sentències JavaScript
</script>
</body>
</html>
```

2.2.2. Inclusió de fitxers .js en les pàgines HTML

Quan la complexitat del web augmenta, és convenient que el codi escrit es pugui reutilitzar en diferents pàgines. Una alternativa a la d'inserir el mateix codi en diferents pàgines (cada modificació d'aquest codi implicaria actualitzar totes les pàgines que el contenen) és crear fitxers independents i incloure una referència al fitxer en cadascuna de les pàgines, és a dir, utilitzar biblioteques externes de scripts.

Aquestes biblioteques no inclouen les etiquetes `<script>` i `</script>` ni cap altre codi HTML, només contenen els scripts. Són fitxers de text i la seva extensió per a JavaScript és .js.

Etiquetes HTML

Les biblioteques JavaScript no inclouen cap etiqueta HTML.

Per a indicar al navegador que ha d'incloure un fitxer .js, en l'etiqueta **`<script>`** s'han d'incloure com a paràmetres el nom i la ubicació del fitxer .js.

```
<html>
<head>
<title>Exemple</title>
<script  type="text/javascript"  src="biblioteca.js"></script>
</head>
<body>
</body>
</html>
```

En el cas anterior, el fitxer biblioteca.js es troba situat en el mateix directori que la pàgina HTML, però si cal fer referència a rutes absolutes, el protocol per al fitxer és http://, igual que amb els fitxers HTML. Per exemple:.

```
<script  type="text/javascript" src="http://www.gem.com/biblioteca.js"></ script >
```

Si s'observa el codi font d'una pàgina HTML des del navegador, el codi del fitxer .js no apareix en la finestra. El que sí apareix és la ruta completa en què es troba el fitxer. Per tant, l'usuari podrà accedir a aquesta ruta, si està accessible, i visualitzar el codi font del fitxer.

2.2.3. Quan el navegador no suporta JavaScript

En alguns casos, el navegador pot no tenir JavaScript activat o pot usar tecnologia que no suporta JavaScript (per exemple, en alguns dispositius mòbils, etc.). En aquests casos, s'ha de proporcionar una alternativa.

La manera més simple d'oferir una alternativa al contingut generat per JavaScript és proporcionar un contingut alternatiu utilitzant l'etiqueta <noscript> per a mostrar continguts en navegadors que no suportin JavaScript. D'aquesta manera, si JavaScript està habilitat, l'etiqueta <noscript> és ignorada.

```
<script type="text/javascript">
   document.write("Hora actual ETS:" + currentTime)
</script>
<noscript>
   Enllaçar a una pàgina que mostra el temps per un script de servidor
   Consultar <a href="time.htm"> ara </ a>
</noscript>
```

2.3. Primers passos

Per a començar a generar les pàgines és necessari triar un editor que permeti escriure el codi HTML i JavaScript. L'editor que es triï no és determinant, sempre que permeti generar fàcilment fitxers de text estàndard i desar-lo amb les extensions htm o html.

El component següent que s'ha de considerar és el navegador; dins del procés de programació s'han d'anar fent proves sobre cada pas implementat. No és necessari provar el codi estant connectat a Internet, pot fer-se fora de línia.

Execució del codi JavaScript

El codi JavaScript es pot executar fora de línia, no és necessari estar connectat a Internet.

Per a executar-lo simplement hem de fer doble clic sobre el fitxer HTML que hem creat i s'obrirà automàticament en el nostre navegador predeterminat, o podrem obrir-lo des del propi navegador.

Per a treballar còmodament, és convenient tenir oberts de manera simultània l'editor de textos en el qual s'escriu el codi i el navegador per a visualitzar els resultats. Els passos que s'han de seguir són:

- Escriure el codi en el document de text mitjançant l'editor de textos.
- Desar l'última versió en el disc.
- Visualitzar el document en el navegador. Si el document ja s'ha visualitzat prèviament, serà suficient d'actualitzar la pàgina.

Tots els manuals de programació comencen amb un primer exemple que es basa a mostrar en pantalla el missatge "Hola Món". No serem menys, i per a això es presentaran alguns mètodes bàsics de JavaScript que permeten dur a terme aquestes accions.

2.3.1. El mètode *write*

En JavaScript, per a escriure una cadena de text en la pàgina HTML s'utilitza el mètode *write* de l'objecte *Document* de la manera següent:

```
document.write(cadena de text);
```

Aquesta línia de codi escriu la cadena de text en la pàgina HTML que conté l'script, a més s'ha de tenir en compte que el mètode *write* modifica el codi font de la pàgina HTML que el crida.

Nota

El mètode *write* requereix un paràmetre que correspon al text que escriurà en el document.

Quan es crida un mètode d'un objecte perquè dugui a terme una determinada tasca, aquest ha d'anar seguit d'uns parèntesis entre els quals s'inclou la informació necessària per a dur a terme aquesta tasca. Un mètode pot no requerir cap paràmetre o requerir-ne un o més, de manera que en JavaScript, en el primer cas, es posen els parèntesis sense res entre ells.

És possible incloure la cadena de text com un literal o mitjançant una variable que contingui el text. Per a introduir-la com un literal:

```
document.write("El meu primer JavaScript");
```

Per a fer-ho mitjançant una variable, en primer lloc, s'ha d'assignar el valor a la variable, que a continuació utilitzem com a paràmetre en la crida:

```
texto = "El meu primer JavaScript";
document.write(text);
```

Ús de cometes

Per a escriure un literal és imprescindible l'ús de les cometes, mentre que una variable no ha de portar-les.

El paràmetre del mètode *write* permet formes més complexes que les exposades en els exemples anteriors. Per exemple, l'escriptura de dues cadenes de text consecutives. Si bé és obvi que podem usar dues vegades el mètode *write* per a escriure dues cadenes de text, podem fer-ho usant-lo només una vegada, per exemple:

```
nombre = "Maria";
document.write("Hola " + nom);
```

En aquest cas, l'operador + no està sumant valors numèrics, sinó que està concatenant dues cadenes de text (això significa que uneix les dues cadenes). D'altra banda, també és possible afegir caràcters especials a les cadenes de text. Per exemple, l'escriptura d'una cadena de text més un retorn de carro:

```
nombre = "Maria";
alert("Hola\n" + nom);
```

Funció alert()

Aquesta funció és utilitzada en JavaScript per a crear una finestra modal en la qual es mostra un missatge a l'usuari final. Aquest missatge es passa a la funció mitjançant el seu paràmetre d'entrada.

Els caràcters especials són els que es detallen en la taula següent:

Taula 2

Caràcters especials	
Caràcter	**Significat**
\n	Nova línia
\t	Tabulador
\r	Retorn de carro
\f	Salt de "pàgina" (caràcter ASCII 12)
\b	Retrocés d'un espai

Com es pot veure, per a mostrar l'efecte dels caràcters especials s'ha utilitzat el mètode a*lert* en lloc de *document.write*. Vegem el codi següent:

```
<html>
<head>
<title>Exemple</title>
</head>
<body>
<script type="text/javascript">
   nom="Maria";
   document.write("Hola\n" + nom);
</script>
```

```
</body>
</html>
```

El codi no produeix el salt de línia que es podria esperar; el motiu és que el salt de línia es produeix en el codi HTML i aquests salts són ignorats pel navegador, que solament mostra un salt de línia si hi ha una etiqueta

Hi ha una variant del mètode anterior que permet escriure una cadena de text seguida d'un retorn de carro:

```
document.writeln(cadena de text)
```

Aquest mètode també utilitza el caràcter especial \n per a indicar el salt de línia, en conseqüència, tal com s'ha comentat abans, aquest salt serà ignorat pel navegador. Per a aconseguir que aquest salt de línia sigui interpretat, hem d'utilitzar-lo dins d'una etiqueta HTML que sigui capaç d'interpretar text preformatat i, en conseqüència, interpretar els caràcters especials; aquesta etiqueta és `<pre></pre>`. Vegem com quedaria l'exemple anterior utilitzant aquesta etiqueta:

```
<pre>
<script type="text/javascript" >
   nom = "Maria";
   document.writeln("Hola");
   document.write(nom);
</script>
</pre>
```

2.3.2. Inclusió d'etiquetes

Com s'ha comentat en l'apartat anterior, el mètode *write* reescriu tot el codi font de la pàgina, per la qual cosa, si es vol utilitzar el mètode i a més aplicar certa estructura o format HTML, s'han d'introduir etiquetes HTML en la cadena de text. D'aquesta manera, amb la inclusió d'aquestes etiquetes no solament es modifica el codi font de la pàgina, sinó també el resultat que es visualitza.

Per tant, l'exemple següent introdueix un salt de línia en introduir l'etiqueta HTML `<br>`:

```
<html>
<head>
<title>Exemple</title>
</head>
<body>
<script  type="text/JavaScript">
   nombre="Maria" ;
   document.write("Hola<br>" + nom) ;
```

```
</script>
</body>
</html>
```

D'aquesta manera, amb la inclusió d'etiquetes HTML és possible generar qualsevol codi HTML de manera dinàmica.

2.3.3. Detectar errors durant el desenvolupament

Un dels aspectes a tenir en compte quan es programa amb JavaScript és el fet que es tracta d'un llenguatge interpretat pel navegador. Així, els scripts no es compilen abans d'executar-se, sinó que es van executant a mesura que el navegador llegeix el codi. D'aquesta manera, quan el navegador es troba amb un error en el codi ho indica, i mostra la línia i una descripció del tipus d'error que s'ha comès.

Nota

En el blog de l'assignatura trobareu informació sobre eines per a detectar errors en els diferents navegadors.

2.3.4. Depuració del codi

De vegades, pot ser que el nostre codi no funcioni correctament encara que sintàcticament sigui correcte i, per tant, pot ser que no provoqui cap error en el navegador; per això és aconsellable depurar el codi a mesura que es desenvolupa per a assegurar-se que tot funciona correctament.

Nota

En el blog de l'assignatura trobareu més informació sobre com es pot accedir a la consola.

Antigament, per a la depuració del codi se solia utilitzar la funció *alert()*. El problema que té l'ús d'aquesta funció és que atura l'execució de l'script fins que l'usuari ha acceptat l'alerta. En el seu lloc és més recomanable utilitzar l'objecte *Console*, que ens permet escriure missatges en la consola del navegador sense interrompre el flux d'execució. En la majoria de navegadors compatibles, es pot accedir a la consola prement el botó F12, o podem utilitzar complements per a això, com ara Firebug, entre molts altres.

A continuació, mostrem els mètodes principals que ens ofereix *Console* per escriure missatges en la consola del navegador:

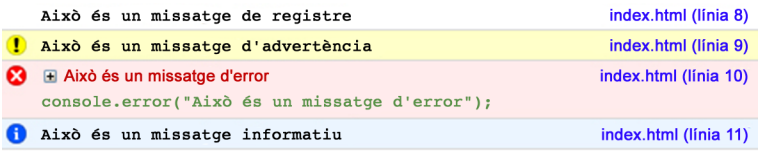
Taula 3

Mètodes de l'objecte *Console*	
log(missatge)	Escriviu un missatge mostrant el contingut que rep com a paràmetre.
warn(missatge)	Escriviu un missatge d'alerta mostrant el contingut que rep com a paràmetre.
error(missatge)	Escriviu un missatge d'error mostrant el contingut que rep com a paràmetre.
info(missatge)	Escriviu un missatge informatiu mostrant el contingut que rep com a paràmetre.

En l'exemple següent podem observar com s'utilitzen els mètodes anteriors:

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>Exemple de depuració de codi</title>
<script type="text/JavaScript" >
   console.log("Això és un missatge de registre");
   console.warn("Això és un missatge d'advertència");
   console.error("Això és un missatge d'error");
   console.info("Això és un missatge informatiu");
</script>
</head>
<body>
</body>
</html>
```

Dels mètodes exposats anteriorment, el més utilitzat és *log()*. Com a resultat de l'exemple anterior en la consola es mostrarà una cosa similar a això:



És possible passar tants paràmetres als mètodes de *Console* com sigui necessari, i automàticament s'escriuran en la mateixa línia de la consola:

```
console.log( param1, param2, param3, ... paramN);
```

Aquests mètodes també poden rebre com a paràmetre un o més objectes JavaScript, per exemple:

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>Escriure objectes a la consola</title>
<script type="text/JavaScript" >
   animal = new Object();
   animal.tipus   = "Gos";
   animal.nom = "Tom";
   console.log(animal);
</script>
</head>
<body>
```

```
</body>
</html>
```

3. Variables, operadors i estructures

En aquest apartat es presenta la sintaxi de JavaScript, que comença per les variables literals, continua amb els operadors que permeten calcular o manipular els valors i acaba amb la implementació de les estructures de control (iteració i decisió).

3.1. Paraules reservades

Es tracta de paraules que ja tenen significat en JavaScript . Per exemple, la paraula *if* s'utilitza per a construir estructures condicionals i, per tant, no ha d'utilitzar-se amb un altre propòsit, com el d'assignar aquest nom a una variable.

En la taula següent s'indiquen les paraules reservades de JavaScript, encara que s'ha de tenir en compte que algunes, encara que no s'usen en les versions actuals, estan reservades per a versions futures del llenguatge.

Taula 4

Paraules reservades de Javascript				
abstract	delete	function	Null	throw
boolean	do	goto	Package	throws
break	double	if	private	transient
byte	else	implements	protected	true
case	enum	import	public	try
catch	export	in	return	typeof
char	extends	instanceof	short	var
class	false	int	static	void
const	final	interface	super	while
continue	finally	long	switch	with
debugger	float	native	syncronized	true
default	for	new	this	

3.2. Dades

Els scripts han de manejar dades per a produir resultats. En tot llenguatge de programació hi ha uns conjunts de tipus de dades que defineixen el rang de valors que poden prendre i el tipus d'operacions que podem fer amb ells. Així, per exemple, una dada de tipus numèric prendrà valors numèrics i mai valors de text.

En JavaScript hi ha els tipus següents de dades:

Taula 5

Tipus de dades		
Tipus	**Descripció**	**Exemples**
Nombre	Qualsevol valor numèric.	5, 12.8, 2e6, ... (*)
Cadena	Qualsevol cadena de text. S'expressa entre cometes dobles o simples.	“Hola”, ‘Hola’
Booleà	Només dos possibles valors: *true* (vertader) i *false* (fals).	If (entrada == 10) pas = true else pas = false
NULL	Paraula clau per a indicar que no hi ha valor.	Nom.value = null
Objecte	Estructura complexa que conté propietats i mètodes.	
Funció	Conjunt de sentències que realitzen una tasca específica.	

(*) Les dades de tipus numèric poden expressar-se en diferents bases:
Decimal: si el nombre no comença per 0, està representat en base 10.
Octal: si el nombre s'escriu començant amb el dígit 0, està en base 8.
Hexadecimal: un nombre començat per 0x o 0X, per exemple 0x1A, estarà expressat en base 16.

3.3. Literals

Es consideren literals els valors que són assignats a les variables. Si es té en compte la taula anterior, tenim els següents:

- Literals numèrics: un literal numèric és qualsevol nombre enter o real expressat en base decimal, octal o hexadecimal, per exemple, 13, #8, 125.34, 0xFF.

- Literal cadenes de text: s'expressen entre cometes dobles o simples, per exemple, “Hola”, “1234”, ‘Pepe’.

- Literals booleans: només poden tenir dos valors, *true* i *false*.

3.4. Comentaris

Els comentaris s'usen en tots els llenguatges de programació per a afegir text en el codi que ajuda a entendre'l, i són línies que l'intèrpret o el compilador ignoren. JavaScript utilitza el mateix format de comentaris que els llenguatges C/C++.

La introducció de comentaris en el codi implementat és una mostra d'elegància i bon fer, ja que ajuda a la comprensió, tant per part de l'autor en revisions posteriors com per part d'altres programadors.

Per a comentar una línia es posen els caràcters // a l'inici de la línia:

```
// Línia de comentari
```

Per a comentar diverses línies s'usen els caràcters /* (inici de comentari) i */ (fi de comentari):

```
/* comentari de
vàries línies */
```

Nota

En el web de l'assignatura trobareu recomanacions sobre com podeu documentar el vostre codi.

3.5. Variables

Els valors no constants que són manejats pels scripts s'emmagatzemen en variables, entenent per *variable* un contenidor d'informació.

Cada variable de l'script ha de tenir un nom que la identifiqui. El nom pot estar format per lletres, nombres i el signe de subratllat _, però no pot contenir espais o qualsevol altre signe de puntuació ni començar per un nombre. Tampoc no es poden fer servir les paraules clau i, com el llenguatge en general, són sensibles a les majúscules i minúscules.

Són noms vàlids per a les variables, per exemple: *resultat, elmeuNum, _num, num2* i *num_1*. En canvi, *2sier, el meu resultat* o *lameva;casa* no són vàlids.

Els valors que pot emmagatzemar una variable poden ser de qualsevol tipus de dades permès en el llenguatge.

3.5.1. Declaració i assignació de variables

Per a declarar una variable i assignar-hi un valor s'utilitza la sintaxi següent:

```
var nom_variable;
```

```
nom_variable = valor;
```

La primera línia declara la variable, i la segona assigna un valor a la variable declarada. Com es veurà més endavant, la primera línia és opcional.

En l'exemple següent es declara una variable de nom *resultat*, i s'hi assigna el resultat de la suma de dos nombres.

```
var resultat;
resultat = 5 + 2;
```

En JavaScript no és necessari indicar de quin tipus és una variable en el moment de definir-la. El tipus de dada queda establert en el moment en què s'assigna un valor a la variable. Aquesta característica es coneix com a **llenguatge feblement tipificat**.

Nota

Com a alternativa, podríeu intentar mostrar els resultats d'aquests exemples a la consola del navegador.

En l'exemple següent, la dada que s'assigna a la variable és de tipus cadena de text:

```
nom = "Maria";
document.write("Hola" + nom);
```

A l'hora de declarar una variable, també és possible dur a terme l'assignació de manera simultània:

```
var resultat = 5 + 2;
```

Una altra possibilitat que ofereix el llenguatge és declarar més d'una variable en una mateixa línia:

```
var x,y,z;
```

3.5.2. Àmbit de les variables

L'àmbit de les variables defineix la ubicació del codi des del qual es pot accedir al contingut d'una variable. D'aquesta manera, una variable és global quan és accessible des de tot l'script o codi i, d'altra banda, una variable és local quan només és accessible en l'estructura on s'ha creat (per exemple, a l'interior d'una funció).

Encara que pogués semblar una pràctica més simple utilitzar totes les variables globals, aquesta tècnica té l'inconvenient d'utilitzar més recursos, ja que ha de mantenir accessibles els valors; mentre que si s'utilitzen variables locals, quan se surt de l'àmbit de definició de la variable s'alliberen els recursos que fan accessible la variable.

En JavaScript, per a definir una variable local dins d'una estructura, s'ha d'utilitzar la paraula reservada *var*; en cas que no s'utilitzi, s'estarà definint la variable amb abast global, i encara que estigui dins de l'estructura, serà accessible després de sortir d'aquesta.

En JavaScript, una variable és local si es declara utilitzant *var* i està situada dins de l'estructura que en defineix l'àmbit.

Si la variable és declarada fora de qualsevol estructura, és global independentment de l'ús o no de la paraula reservada *var*.

En l'exemple següent, les variables *x*, *w* i *y* són globals, mentre que la variable *z* és local:

```
var x = 1;   // x accessible dins i fora de la funció proves
y = x + 2;   // y accessible dins i fora de la funció proves
function proves(){

   var z = y + 1   // z només accessible dins de la funció proves
   w = x + z + 4   // w és accessible dins i fora de la funció proves
   document.write("El valor de w és " + w )
}
```

3.5.3. Conversions de tipus

Com s'ha vist, les variables adquireixen el tipus corresponent en el moment en què se'ls assigna el valor. D'aquesta manera, en l'expressió següent:

```
lameva_var = "1000";
```

La variable *lameva_meva* és de tipus cadena de text, mentre que en l'assignació següent la variable serà de tipus nombre.

```
lameva_var = 1000;
```

El tipus de la variable o de la dada determinarà quin tipus d'operacions es poden fer.

Una particularitat de JavaScript és la conversió a cadena d'un tipus numèric quan en una operació + un dels operands és una cadena. Per exemple, atesa l'assignació següent:

```
lameva_var1 = "13";
lameva_var2 = 5;
```

```
x = lameva_var1 + lameva_var2;
```

S'obté que la variable *x* adquireix el tipus cadena de caràcters, ja que *lameva_var1* és de tipus cadena de caràcters. El valor que pren *x* és "135", atès que l'operador + en aquest exemple concatena les dues variables.

A més de les normes comentades, JavaScript disposa de funcions especials per a fer conversions específiques. Són les funcions *eval, parseInt i parseFloat*, que es veuran més endavant.

3.6. Operadors

Els operadors permeten formar expressions. Un literal o una variable ja formen per si sols una expressió, però juntament amb els operadors es poden construir expressions més complexes.

Per exemple, en l'expressió $x = y + 2$, són variables *x* i *y*, mentre que 2 és un literal i els signes = i + són operadors.

Els operadors es classifiquen segons la funcionalitat en els grups següents:

- Operadors aritmètics
- Operadors d'assignació
- Operadors de comparació
- Operadors booleans
- Operadors de bits
- Operadors especials: l'operador +, l'operador ?:, els signes de puntuació (coma, parèntesis i claudàtors) i els operadors per al treball amb objectes

3.6.1. Operadors aritmètics

Els operadors aritmètics permeten fer operacions matemàtiques. En la taula següent es mostren els operadors disponibles en JavaScript :

Taula 6

Operadors aritmètics			
Operador	**Descripció**	**Exemple**	**Resultat**
+	Suma	3 + 4	7
-	Resta	3 – 4	–1
++	Increment	3++	4
--	Decrement	3--	2
*	Producte	3 * 4	12
/	Divisió	3 / 4	0.75

Operadors aritmètics			
Operador	**Descripció**	**Exemple**	**Resultat**
%	Mòdul (resta de la divisió)	3 % 4	3
-	Menys unari (negació)	– (3 + 4)	–7

Els operadors d'increment i decrement es poden usar de dues maneres diferents dins d'una expressió d'assignació.

Si es posiciona davant d'una expressió, provoca que primer tingui lloc l'increment i, a continuació, l'assignació. Per exemple:

```
x = 13;
y = ++x;
```

En aquest exemple, les variables adquireixen els valors $x = 14$ i $y = 14$. El motiu és que primer assignem a x el valor 13, a continuació s'incrementa el valor de x en una unitat i, finalment, assignem el valor de x a la variable y.

La posició dels operadors d'increment/decrement afecta l'assignació d'aquests.

No obstant això, si es posiciona darrere de l'expressió, observem que primer té lloc l'assignació i, a continuació, l'increment. Per exemple:

```
x = 13;
y = x++;
```

En aquest exemple, tenim l'assignació del valor 13 a la variable x, però a continuació, en la línia següent, primer té lloc l'assignació del valor de x a la variable y (aquesta prendrà el valor 13) i, després, s'augmenta el valor de la variable x en una unitat (pren el valor 14).

3.6.2. Operadors d'assignació

Els operadors d'assignació permeten assignar el resultat d'una expressió a una variable; a més permeten dur a terme una operació aritmètica en ell mateix moment en què té lloc l'assignació.

La taula següent mostra els operadors disponibles i un exemple senzill d'ús d'aquests.

Taula 7

Operadors d'assignació			
Operador	**Descripció**	**Exemple**	**Equival a...**
=	Assigna	*x = y + z*	
+=	Suma i assigna	*x += y*	*x = x + y*
-=	Resta i assigna	*x -= y*	*x = x - y*
*=	Multiplica i assigna	*x *= y*	*x = x * y*
/=	Divideix i assigna	*x /= y*	*x = x / y*
%=	Calcula el mòdul i assigna	*x %= y*	*x = x % y*

3.6.3. Operadors de comparació

Els operadors de comparació permeten comparar els valors entre dues expressions, que poden ser variables, literals o fins i tot més complexes. Els operadors de comparació són utilitzats en les estructures iteratives i de decisió que es presentaran més endavant.

La següent taula següent mostra els operadors de comparació, amb un exemple d'aquests:

Taula 8

Operadors de comparació			
Operador	**Descripció**	**Exemple**	**Resultat**
==	Igualtat	3 == “3”	Cert
!=	Desigualtat	3 != “3”	Fals
<	Més petit	3 < 3	Fals
>	Més gran	3 > 3	Fals
<=	Més petit o igual	3 <= 3	Cert
>=	Més gran o igual	3 >= 3	Cert
===	Igualtat estricta	3 === “3”	Fals
!==	Desigualtat estricta	3 !== “3”	Cert

La igualtat i la desigualtat estrictes serveixen per al mateix que la igualtat i la desigualtat no estrictes, però fan una comprovació estricta de tipus. Això vol dir que no només es comprova que el valor sigui el mateix, sinó que a més es comprova si són del mateix tipus. Es van incloure en l'estàndard ECMAScript.

En la igualtat i la desigualtat no estrictes es fa una conversió de tipus (si és necessària) abans de fer la comparació. Per això "3" == 3 donarà com a resultat *cert* a pesar que "3" és un caràcter i 3 un nombre.

3.6.4. Operadors lògics o booleans

Els operadors lògics permeten formar expressions que només poden tenir com a resultat un valor booleà (*true* o *false*). Per a la seva interpretació és necessari conèixer la lògica booleana.

El comportament d'un operador lògic sol definir-se mitjançant la seva corresponent taula de veritat, en aquesta es mostra el resultat que produeix l'aplicació d'un determinat operador a un o dos valors lògics. Les operacions lògiques més usuals són:

- L'operador NOT, que és un operador unari (aplicat a un únic operant). La seva lògica és molt simple: canvia el valor de *cert* a *fals* i viceversa.

- L'operador OR, que es pot aplicar a dos o més operands. Si tots els operands són falsos, assigna el valor *fals*. En cas contrari, és a dir, si almenys un és cert, assigna el valor *cert*.

- L'operador AND, que es pot aplicar a dos o més operands. Si tots són certs, assigna el valor *cert*. En cas contrari, és a dir, si almenys un és fals, assigna el valor *fals*.

En la taula següent es mostren els operadors booleans en JavaScript i un exemple de cadascun.

Taula 9

Operadors lògics			
Operador	**Descripció**	**Exemple**	**Resultat**
&&	i (AND)	*x = 1* *(x > 0) && (x <= 5)*	Cert
||	o (OR)	*x = 1* *(x > 5) || (x == 0)*	Fals
!	negació (NOT)	*x = 1* *!(x > 5)*	Cert

3.6.5. Operadors de bits

Els operadors de bits permeten modificar o comparar valors en bits, és a dir, operen amb conjunts d'uns i zeros, encara que el resultat que retornen està expressat en la manera estàndard de JavaScript per als valors numèrics.

Taula 10

Operadors de bits	
Operador	**Descripció**
&	(AND) Retorna un 1 si els dos valors són 1.
|	(OR) Retorna un 1 si almenys un dels dos valors és 1.
^	(XOR) Retorna un 1 si solament un dels dos valors és 1.
<<	Desplaçament a l'esquerra d'un determinat nombre de bits. Els espais de la dreta que queden "buits" s'emplenen amb 0.
>>	Desplaçament a la dreta d'un determinat nombre de bits i afegeix per l'esquerra els bits que s'han desbordat per la dreta
>>>	Desplaçament a la dreta d'un determinat nombre de bits. Els espais de l'esquerra que queden "buits" s'emplenen amb 0.
&=	*x*&=*i* equival a x =*x*&*i*
|=	*x*|=*i* equival a x =*x*|*i*
^=	*x*^=*i* equival a x =*x*^*i*
<<=	*x*<<=*i* equival a x =*x*<<*i*
>>=	*x*>>=*i* equival a x =*x*>>*i*
>>>=	*x*>>>=*i* equival a x =*x*>>>*i*

No es tracta d'operadors que tinguin ús comú en un llenguatge script; el seu ús és més comú en llenguatges de programació que accedeixen a nivells bàsics de l'ordinador com accessos directes a memòria o a disc.

3.6.6. L'operador +

El significat del símbol + és diferent en funció del tipus de les expressions entre les quals es troba. Quan les expressions són de tipus nombre, el símbol + actua com a operador aritmètic, per exemple:

```
x = 1;
y = x + 13
```

En l'exemple, la variable **y** obté el valor 14.

Però, si almenys un dels operands és del tipus cadena de text, l'operador + es transforma en l'operador que concatena les dues expressions. Per exemple:

```
nombre = "Maria";
frase = "Hola " + nom;
```

La variable frase adquireix el valor "Hola Maria".

L'operador + farà una **suma o concatenació** depenent del tipus dels operands.

3.6.7. L'operador condicional ?:

Aquest operador és una abreviatura de l'estructura de control simple *if...else*. Aquesta s'estudiarà amb més detall en l'apartat següent, però a continuació se'n presenta sintaxi:

```
condició ? valor1 : valor2;
```

Si la condició es compleix, l'estructura assigna el valor1; en cas contrari, valor2. Per exemple, en l'assignació següent:

```
x = 5 <= 8 ? 1 : 4
```

Si assigna a la variable *x* el valor 1. Això és perquè es compleix la condició (5 <= 8) i, per tant, l'expressió retorna el primer valor, l'1, que és assignat a la variable *x*.

3.6.8. Signes de puntuació

El signe de puntuació coma s'utilitza per a delimitar sèries d'elements. Per exemple, anteriorment s'havia definit un conjunt de variables en una única línia:

```
var x,y,z;
```

Els signes ( ) s'utilitzen en les crides a les funcions, de manera que els paràmetres s'indiquen entre els parèntesis.

És especialment important l'ús dels parèntesis en la creació d'expressions aritmètiques complexes, ja que s'utilitzen per a definir la prioritat. Un parell d'exemples:

```
x = suma(3,4); //s'assigna a x el resultat de la funció suma
x = (y+3)*3; //s'assigna a x el producte per 3 del valor de y sumat a 3
```

Els signes [ ] s'utilitzen per a referenciar els elements que pertanyen a un *array* o vector; aquests es presentaran més endavant.

3.6.9. Operadors per al treball amb objectes

Permeten crear, eliminar i referenciar els objectes del llenguatge, i els objectes definits pel programador.

Taula 11

Operadors per al treball amb objectes		
Operador	**Descripció**	**Exemple**
new	Crea un objecte.	avui = new Date()
delete	Destrueix un objecte o un element d'un array.	delete avui; delete vector[1];
this	Referencia l'objecte actual.	
in	Ítem en l'objecte.	
typeof	Instància de l'objecte.	
void	Retorn de cap valor.	

3.6.10. Precedència d'operadors

En una expressió complexa en la qual intervenen més d'un operador, és possible determinar l'ordre en què aquests han d'avaluar-se mitjançant l'ús de parèntesis. Per exemple:

```
document.write( 5 - ( 2 * 2 ) );
```

mostrarà el valor 1 en el document HTML; mentre que l'expressió:

```
document.write( ( 5 - 2 ) * 2 );
```

mostrarà com a resultat el valor 6.

En cas de no usar parèntesis, la prioritat s'estableix segons l'ordre següent (de més a menys prioritat):

Taula 12

Prioritat d'operadors	
Crida a una funció i membre de...	( ) [ ]
Negació, menys unari, increment i decrement	! - ++ --
Producte, divisió i mòdul	* / %
Suma i resta	+ #
Desplaçament de bits	>> << >>>
Comparació de relació	< <= > >=

Prioritat d'operadors	
Comparació d'igualtat i desigualtat	== !=
AND binari	&
XOR binari	^
OR binari	|
AND lògic	&&
OR lògic	||
Condicional	?:
Assignació	= += -= *= /= %= &= |= ^= <<= >>= >>>=
Coma	,

Quan en una expressió intervenen operadors que tenen la mateixa prioritat, s'avaluen d'esquerra a dreta.

Vegem a continuació uns exemples:

```
X = 5+4*3; // el producte té més prioritat, per tant X = 17
X = ++X*3; // es calcula l'increment i a continuació el producte i el resultat és X = 54
X = 2;
Y = 7;
Z = (Y++ - X) + Y; //El valor final de Z serà 13
```

En l'últim exemple veiem que primer es fa la resta entre *Y* i *X*, i com a resultat d'aquesta operació s'obté el valor 5; tot seguit s'incrementa el valor de Y; finalment se suma el valor calculat anteriorment 5 amb el nou valor de Y, i ens dóna com a resultat Z = 13.

3.7. Estructures de control condicionals

Una estructura de control condicional és una estructura que controla el flux. D'aquesta manera, s'aconsegueix que l'script dugui a terme una tasca o una altra depenent del resultat d'avaluar una certa condició.

En JavaScript, les estructures de control condicionals són les següents:

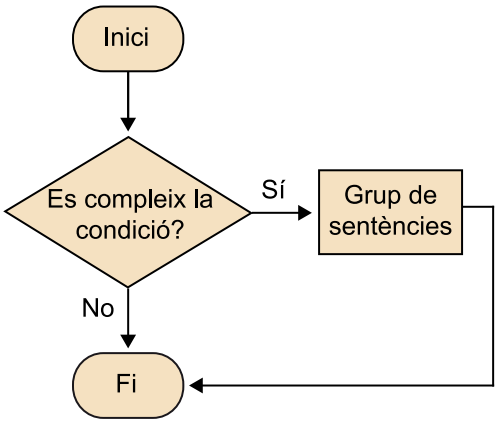
- *if... else*, que en la forma més simple pot ometre l'*else* i, en els casos més complexos, pot prendre la forma *if... else if...*
- *switch*

3.7.1. Estructura de control *if...*

El seu funcionament és molt bàsic: du a terme una acció específica si es compleix una condició. En general la sintaxi és:

```
if ( condició ){
   //grup de sentències
}
```

Figura 1. Diagrama de flux per a l'estructura de control *if...*



En cas que el grup de sentències quedi reduït a una sola sentència, les claus poden ometre's, però aquestes també tenen un efecte delimitador per al programador i ajuden a visualitzar les accions definides en l'estructura.

A continuació, es presenta un exemple en què es mostrarà un text addicional a la pàgina, en cas que l'usuari hagi introduït la clau d'entrada correcta, en aquest cas "csi".

```
var resp = prompt("Introdueix el codi per veure la salutació","");
if (resp=="csi") document.write("<h3>Benvingut al nostre lloc web</h3>");
```

El mètode prompt()

Aquest mètode mostra un quadre de diàleg modal format pels botons Acceptar i Cancel·lar, un text definit pel primer paràmetre enviat a la funció, i un camp de text amb valor predeterminat definit pel segon paràmetre. La funció retorna el valor inserit en el camp de formulari si l'usuari prem Acceptar o *null* si prem Cancel·lar o la creu de tancar.

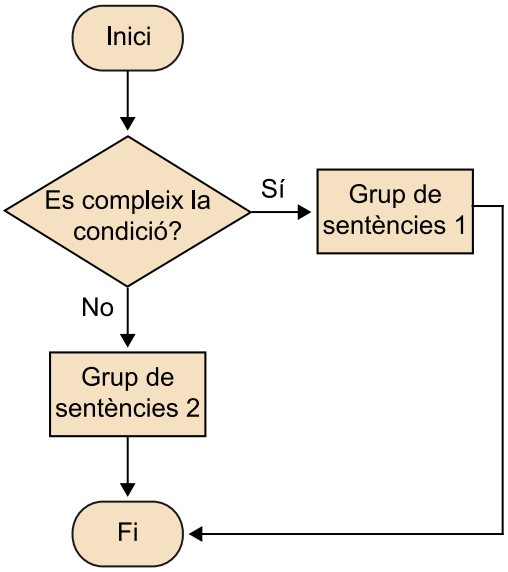
3.7.2. L'estructura de control *if ... else*

Aquesta estructura inclou la possibilitat d'executar un conjunt d'accions en cas que la condició no es compleixi. La sintaxi és la següent:

```
if ( condició ){
   //grup de sentències 1
```

```
} else {
   //grup de sentències 2
}
```

Figura 2. Diagrama de flux per a l'estructura de control *if ... else*



En l'exemple següent es mostra com es pot incloure una acció alternativa en cas que no es compleixi la condició avaluada en el condicional:

```
var resp = prompt("Introdueix el número del mes actual","");
if ( resp == "8" ){
   document.write("Som a l'agost");
} else {
   document.write("No som a l'agost");
}
```

Una manera alternativa d'escriure aquesta sentència és usar l'expressió plantejada en l'apartat anterior:

```
condició ? expressió1 : expressió2 ;
```

D'aquesta manera, l'exemple anterior quedaria de la manera següent:

```
resp == "8" ? document.write("Som a l'agost") : document.write("No som a l'agost");
```

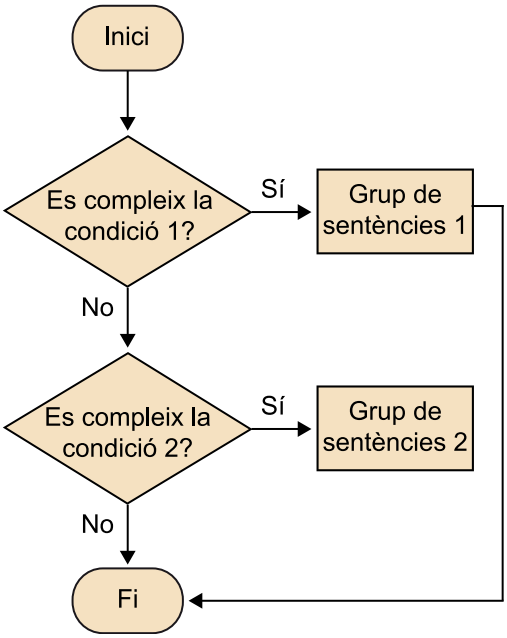
3.7.3. L'estructura de control *if ... else if ...*

Aquesta estructura permet l'ús de nous condicionals en cas que no es compleixi el primer. La sintaxi es presenta a continuació:

```
if ( condició1 ){
   //grup de sentències 1
```

```
} else if ( condició2 ) {
   //grup de sentències 2
}
```

Figura 3. Diagrama de flux per a l'estructura de control *if ... else if ...*



En l'exemple següent es mostra una estructura *if... else if*:

```
var dia = prompt("Introdueix el número de dia de la setmana:", 0);
if (dia == 1){
   document.write("Avui és dilluns");
} else if (dia == 2){
   document.write("Avui és dimarts");
} else if (dia == 3){
   document.write("Avui és dimecres");
} else if (dia == 4){
   document.write("Avui és dijous");
} else if (dia == 5){
   document.write("Avui és divendres");
} else if (dia == 6){
   document.write("Avui és dissabte");
} else if (dia == 7){
   document.write("Avui és diumenge");
} else {
   document.write("El dia ha de ser entre 1 i 7");
}
```

És possible afegir tantes estructures *else if* com sigui necessari, encara que a aquest tipus de necessitats s'adequa millor l'estructura *switch* que es presenta a continuació.

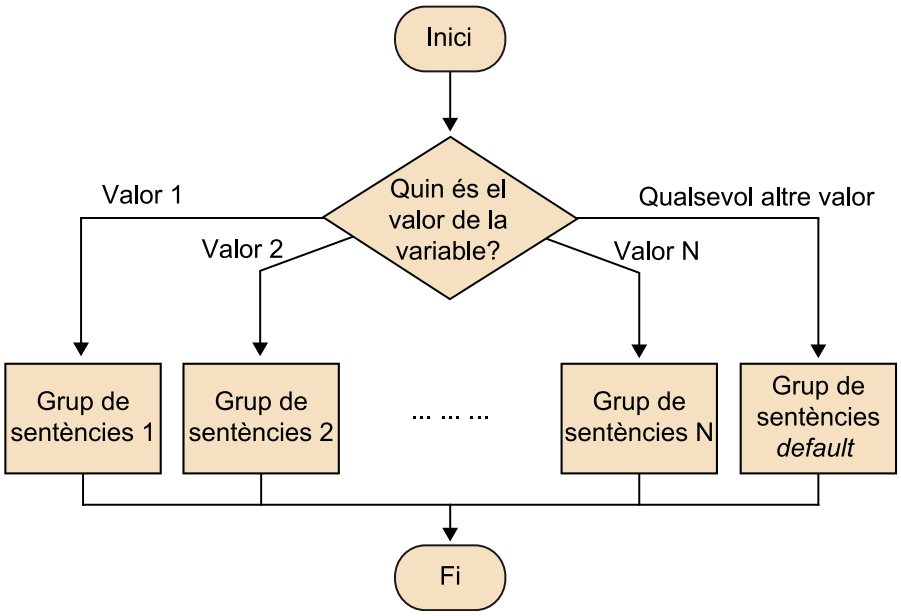
3.7.4. L'estructura de control *switch*

L'estructura *switch* es basa en l'avaluació d'una variable, de manera que en el cos de l'estructura es va comparant de manera estricta amb un conjunt de valors. Així doncs, en cas de coincidència, s'executaran el conjunt de sentències que van associades al valor coincident.

La sintaxi de l'estructura és la següent:

```
switch ( variable ){
   case valor1:
      //grup de sentències 1
      break;
   case valor2:
      //grup de sentències 2
      break;
   ......................
   case valorN:
      //grup de sentències N
      break;
   default:
      //grup de sentències si no es compleix cap dels casos anteriors
}
```

Figura 4. Diagrama de flux per a l'estructura de control *switch*



El paper de la paraula reservada *break* és sortir de l'estructura, de manera que no continuï amb les sentències que hi ha en les opcions següents. Per tant, es pot obligar a passar per les sentències corresponents dos valors només ometent la instrucció *break* en les sentències del primer.

L'exemple següent mostra una estructura alternativa múltiple:

```
var diac = prompt("Introdueix el número de dia de la setmana:", 0);
var dia = parseInt(diac);
switch(dia){
   case 1: document.write("Avui és dilluns");
      break;
   case 2: document.write("Avui és dimarts");
      break;
   case 3: document.write("Avui és dimecres");
      break;
   case 4: document.write("Avui és dijous");
      break;
   case 5: document.write("Avui és divendres");
      break;
   case 6: document.write("Avui és dissabte");
      break;
   case 7: document.write("Avui és diumenge");
      break;
   default:document.write("El dia ha de ser entre 1 i 7");
}
```

En aquest exemple ha estat necessari emprar la funció *parseInt()*, ja que el valor retornat per la funció *prompt()* és una cadena de caràcters i en el *switch* s'estan usant valors numèrics.

En l'exemple anterior de l'estructura *if... else if...*, no va ser necessari fer la conversió, ja que l'operador == no compara de manera estricta.

És important tenir en compte que dins del conjunt d'accions de cadascuna de les estructures anteriors es pot introduir una nova estructura condicional, la qual cosa provoca la implantació d'estructures condicionals.

3.8. Estructures de control iteratives

Una sentència iterativa, repetitiva o bucle és una estructura que permet repetir una tasca un nombre determinat de vegades. En JavaScript, les estructures de control iteratives són les següents:

- *for*
- *while*
- *do ... while*

En aquestes estructures, el nombre d'iteracions està determinat per la condició de finalització, encara que és possible interrompre'n l'execució mitjançant l'ús de les sentències *break* i *continue*.

3.8.1. L'estructura de control *for*

En aquesta estructura s'utilitza un comptador que determina el nombre de vegades en què ha de fer-se l'acció especificada. És comú utilitzar aquesta estructura quan el nombre d'iteracions és un valor conegut. En general, la sintaxi és la següent:

```
for ([expressió inicialització];[condició];[expressió increment o decrement] ){
   //grup d'accions
}
```

En aquesta estructura s'utilitza una variable que fa el paper de comptador. Aquest pot ser usat de manera que incrementi el seu valor, en una certa quantitat, en cada iteració fins a un valor determinat o de manera que el valor inicial vagi decreixent en cada iteració.

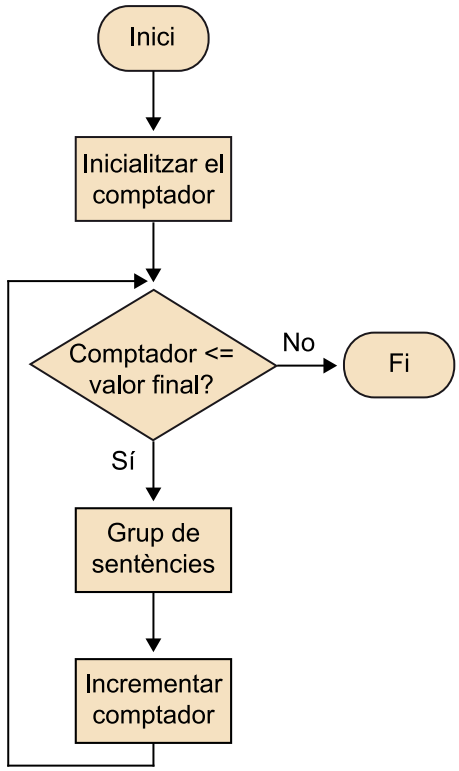
Amb un comptador ascendent, l'expressió adquireix la forma següent:

```
for (comptador = valor inicial; comptador <= valor final; comptador += increment ){
   //grup d'accions
}
```

Amb un comptador descendent l'expressió pren la forma següent:

```
for (comptador = valor inicial; comptador >= valor final; comptador -= decrement] ){
   //grup d'accions
}
```

Figura 5. Diagrama de flux per a l'estructura de control *for*



L'exemple següent calcula i mostra la taula de multiplicar del 2:

```
for ( var i = 0; i < 11; i++ ){
   result = i * 2;
   document.write( "2 * " + i + " = " + result + "<br>");
}
```

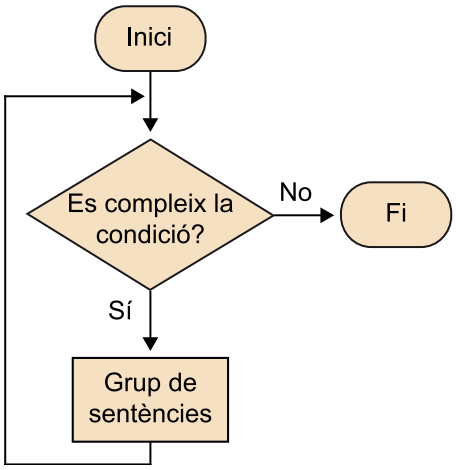
En la primera línia del *for* de l'exemple, l'expressió (var i= 0; i<=10; i++) fa el següent: en primer lloc, inicialitza la variable i el valor 0, executa el cos del bucle; en els passos següents del bucle, el valor de la variable, i augmenta en una unitat a causa de la tercera entrada *i*++; el bucle continuarà executant-se, fins que la variable *i* adquireixi el valor 11, que, com que supera el valor màxim 10, finalitza l'execució del bucle.

3.8.2. L'estructura de control *while*

L'estructura de control *while* es basa en l'execució d'un grup d'accions en cas que es compleixi una certa condició. Aquest grup d'accions pot no executar-se mai si la condició no es compleix, ja que aquesta es troba a l'inici de l'estructura. La sintaxi de l'estructura *while* és la següent:

```
while (condició){
   //grup d'accions
}
```

Figura 6. Diagrama de flux per a l'estructura de control *while*



L'exemple següent calcula la suma dels valors positius més petita o igual a un nombre introduït per l'usuari.

```
var suma = 0;
var num = parseInt( prompt("Introdueix un nombre més gran que 0", "") );
if (num > 0){
   while(num > 0){
      suma += num;
      num--;
   }
     document.write("La suma dels valors més petits o iguals que el valor introduït és: <b>" + suma
+ "</b>");
}
```

En l'exemple se sol·licita a l'usuari el valor numèric, aquest es converteix a numèric amb el mètode *parseInt*, ja que en sol·licitar-se amb el mètode *prompt*, el valor és emmagatzemat com a cadena de text. Comença el codi amb una estructura *if* que comprova si el valor introduït és més gran que zero i, en cas que sigui així, passa a executar l'estructura *while*.

El condicional de l'estructura *while* comprova que la variable sigui més gran que zero, i en cas que sigui així executa el cos de l'estructura. En el cos es fa la suma i es redueix la variable a una unitat. D'aquesta manera, s'arribarà al valor 0 i, en deixar de complir-se el condicional, no s'executarà cap pas més del bucle.

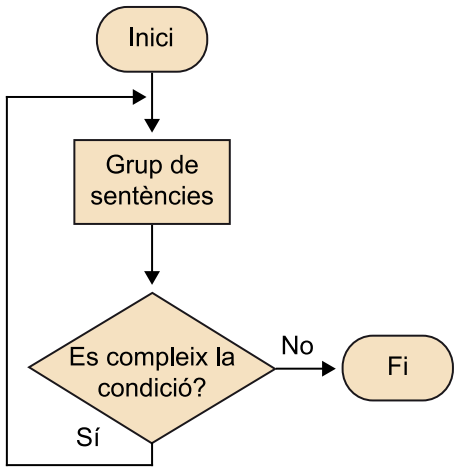
3.8.3. L'estructura de control *do ... while*

Aquesta estructura es diferencia de l'anterior en el fet que les accions s'executen almenys una vegada. Això és perquè la condició de finalització es troba al final del bloc.

La sintaxi de l'estructura és la següent:

```
do{
   //grup d'accions;
} while(condició)
```

Figura 7. Diagrama de flux per a l'estructura de control *do ... while*



L'exemple següent permet calcular el resultat d'expressions matemàtiques. L'usuari decideix al final de cada càlcul si vol continuar fent operacions o no.

```
var resp = "";
var exp = "";
do{
   exp = prompt("Introdueix una expressió matemàtica", "");
   alert("El resultat de l'expressió és: " + eval(exp) );
   resp = prompt("Vols fer un altre càlcul? (s/n)", "s");
}while( resp == "S" || resp == "s")
```

En aquest cas, se sol·licita a l'usuari que introdueixi una expressió que en la línia següent és avaluada amb el mètode *eval*. A continuació, se sol·licita a l'usuari el valor *s*, en cas que vulgui executar un nou pas del bucle. D'aquesta manera, en la línia següent es comprova el valor introduït per l'usuari i, en cas afirmatiu, executarà un altre pas del bucle.

3.8.4. La sentència *break*

La sentència *break* s'utilitza per a forçar la interrupció de l'execució d'un bucle. A continuació, es presenta un exemple en el qual s'usa la instrucció *break* per a aturar el bucle que calcula la taula de multiplicar del 2.

```
for ( var i = 0; i < 11; i++ ){
   result = i * 2;
   document.write( "2 * " + i + " = " + result + "<br>");
   resp = prompt("Vols continuar?","s");
```

```
    if ( resp == "n" || resp == "N" ) break;
}
```

En l'exemple anterior s'introdueix la possibilitat de sortir del bucle mitjançant la sentència *break*, encara que la variable de control continuï complint la condició del bucle.

La utilització del *break* està desaconsellada en una programació "elegant", ja que dificulta considerablement el manteniment dels programes.

3.8.5. La sentència *continue*

La sentència *continue* interromp l'execució de la iteració actual i transfereix el control a la iteració següent.

En l'exemple següent, s'usa la instrucció *continue* per a no imprimir el resultat del producte 2*5 en la taula de multiplicar del 2.

```
for ( var i = 0; i < 11; i++ ){
   if ( i == 5 ) continue;
   result = i * 2;
   document.write( "2 * " + i + " = " + result + "<br>");
}
```

Com en el cas del *break*, tampoc no és recomanable l'ús de la instrucció *continue*. Aquest mateix exemple, sense utilitzar la sentència, es pot implementar de la manera següent:

```
for ( var i = 0; i < 11; i++ ){
   if ( i != 5 ) {
      result = i * 2;
      document.write( "2 * " + i + " = " + result + "<br>");
   }
}
```

3.9. La sentència *with*

La sentència *with* permet usar un grup de sentències que fan referència a un objecte, de manera que no sigui necessari indicar, en cadascuna, l'objecte a què fan referència.

La sintaxi és la següent:

```
with ( objecte ){
   //grup de sentències
}
```

En l'exemple següent s'omet el nom de l'*array* en les sentències que hi fan referència dins de l'estructura *with*:

```
ciutats = new Array("Barcelona", "Ciutat de Mallorca", "Càceres", "Sevilla")
with (ciutats){
   sort();
   reverse();
}
document.write("Les dades finalment són: " + ciutats);
```

Array

És una estructura similar a una variable, però s'hi poden emmagatzemar diversos valors. D'aquesta manera, els *arrays* permeten desar valors i accedir-hi de manera independent, utilitzant un índex que marqui la posició del valor que es vol consultar. En tractar-se d'una estructura o objecte, disposa d'un conjunt de mètodes i propietats, que són cridats segons la sintaxi nom_objecte.metode. En aquest mòdul i al llarg del mòdul "Orientació a objectes en JavaScript " parlarem amb més detall de l'objecte *Array*.

Sense l'estructura *with*, s'hauria d'especificar el nom de l'*array* mitjançant la sintaxi següent:

```
ciutats.sort();
ciutats.reverse();
```

4. Funcions i objectes bàsics

4.1. Definició d'una funció

Les funcions són una de les parts fonamentals en JavaScript. Una funció es defineix com un conjunt de sentències que duen a terme una tasca específica, i que es pot cridar des de qualsevol part de l'aplicació.

Una funció es compon de diverses sentències que, en conjunt, duen a terme una tasca determinada.

4.1.1. Definició de la funció en JavaScript

Una funció es defineix mitjançant l'ús de la paraula *function*, seguida del següent:

- el nom de la funció.
- la llista de paràmetres de la funció, tancats entre parèntesis i separats per comes.
- les sentències JavaScript que defineixen la funció, tancades entre claus.

És a dir:

```
function nom_funcio( [paràmetre 1, paràmetre 2, ... ] ){
   bloc de sentències;
}
```

L'existència d'arguments, com el nombre d'aquests, és opcional i depèn de cada funció en concret.

El nom de la funció pot contenir el caràcter _, però ni espais ni accents, i igual que la resta de codi JavaScript, és sensible a les majúscules i minúscules.

Taula 13

Exemples de noms admesos per a les funcions		

`function VeureMissatge()`
`{`
`  document.write("Hola")`
`}`
	`function veure_missatge()`
	`{`
	`  document.write("Hola")`
	`}`
		`function _veureMissatge()`
		`{`
		`  document.write("Hola")`
		`}`

El bloc de sentències va entre les claus { i }, que especifiquen l'inici i el final de la funció.

4.1.2. Ubicació en el document HTML

Les funcions es defineixen en el document HTML entre les etiquetes <script> i </script>. La millor ubicació és en la capçalera del document, ja que aquestes són carregades abans del cos del document. En el cas de les funcions, encara que es referencïi a objectes de la pàgina, la funció es carrega però no s'executa fins que no és cridada per la qual cosa no es produirà cap error. Les funcions també poden definir-se en el cos del document o en un fitxer extern.

```
<html>
<head>
<title>Exemple funció</title>
<script type="text/javascript">
<!--
function nom_funcio(paràmetres){
   bloc de sentències
}
//-->
</script>
</head>
<body>
</body>
</html>
```

En cas d'haver-hi més d'una funció, aquestes també se situarien entre les etiquetes <script> i </script>.

```
<html>
<head>
<title>El meu document</title>
<script type="text/javascript">
<!--
   function nom_funcio_1(paràmetres){
      bloc de sentències
   }

   function nom_funcio_2(paràmetres){
      bloc de sentències
   }
//-->
</script>
</head>
<body>
</body>
```

```
</html>
```

En un document es poden incloure tantes funcions com sigui necessari.

4.1.3. Crida i execució de les funcions

Les funcions no s'executen per si soles, s'han de cridar, bé des d'una altra funció o bé des del cos del document.

En l'exemple següent es crida una funció des del cos del document, concretament es crida quan l'usuari prem un botó:

```
<html>
<head>
<title>Exemple crides</title>
<script type="text/javascript">
<!--
   function salutacio() {
      alert("Hola");
   }
//-->
</script>
</head>
<body bgcolor="#FFFFFF" text="#000000">
<p> Pitja el botó:
<form name="elmeuForm">
<input type="button" name="botó" value="Salutació" onClick="javaScript: salutacio()" >
</form>
</p>
</body>
</html>
```

Perquè una funció s'executi s'ha de cridar.

Els esdeveniments són el mecanisme que permet controlar les accions dels usuaris i definir un comportament associat a aquestes accions. Per exemple, quan un usuari prem un botó, edita un camp de text o abandona una pàgina, es produeix un esdeveniment.

Les accions que s'han d'executar es defineixen amb els programes controladors d'esdeveniments. Per exemple, el programa controlador d'esdeveniments *onclick* serveix per a descriure les accions que s'han d'executar quan es fa clic.

En l'exemple següent,

<input type=button value="Prémer" onclick="sentències_javascript...">

s'ha afegit un atribut nou en l'etiqueta que té el mateix nom que l'esdeveniment i que indica el codi que s'ha d'executar quan ocorri l'esdeveniment.

Cada element té la seva pròpia llista d'esdeveniments suportats, la gestió d'esdeveniments s'estudiarà en l'apartat cinquè del mòdul "Introducció al DOM".

4.1.4. Ús dels paràmetres de la funció

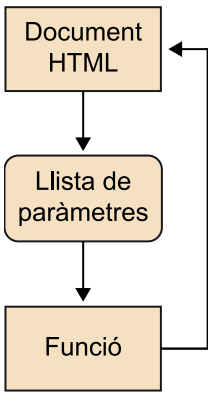
Els paràmetres són les dades d'entrada que la funció necessita per a dur a terme una tasca. Una funció pot rebre tants paràmetres com sigui necessari.

En l'*script* següent, a diferència del que es mostra en l'exemple anterior, la funció rep el text que mostrarà en el missatge:

```
function salutació(text) {
   alert(text);
}
```

Els paràmetres són les dades d'entrada a la funció.

Figura 8



L'exemple següent mostra com una mateixa funció es pot cridar diverses vegades:

```
<html>
<head>
<title>Exemple crides</title>
<script type="text/javascript">
<!--
```

```
function salutacio(text) {
   alert(text);
}
//-->
</script>
</head>
<body bgcolor="#FFFFFF" text="#000000">
<p>Pitja el boto:
<form name="elmeuForm">
<input type="button" name="boto1" value="Salutacio 1" onClick="javaScript:salutacio('Hola Joan')">
<br><br>
<input type="button" name="boto2" value="Salutacio 2" onClick="javaScript:salutacio('Hola Maria')">
</form>
</p>
</body>
</html>
```

El resultat que es visualitzarà en el navegador serà diferent per a cada crida, ja que depèn del valor del paràmetre.

En aquest exemple també es pot veure com en alguns casos és necessari alternar l'ús de les cometes dobles amb les cometes simples. Concretament en la línia:

```
onClick="javaScript:salutacio('Hola Joan')"
```

el text que es passa a la funció com a paràmetre, s'ha posat entre cometes simples.

Això es fa així per a no confondre el navegador, ja que les cometes dobles s'estan fent servir per a indicar l'inici i el final de la crida. Per exemple, si s'escrivís:

```
onClick="javaScript:salutacio("Hola Joan")"
```

el navegador interpretaria que la sentència que s'ha d'executar en *onClick* és *javaScript:salutacio*, la qual cosa provocaria un error.

Un altre aspecte que s'ha de destacar és l'àmbit en què la variable és definida. Per exemple, quan es passa un paràmetre a una funció, aquest s'usa dins de la funció com una variable. Fora de la funció no existeix i, per tant, no s'hi pot fer servir.

El paràmetre d'una funció, només existeix dins de la pròpia funció.

A més, quan es passa una variable com a paràmetre d'una funció, aquesta es passa per valor, és a dir, la manipulació a l'interior de la funció de la variable no modifica el valor de la variable a l'exterior de la funció.

Aquesta característica no sempre es compleix, ja que quan s'hi passen tipus compostos com *arrays* o objectes, aquests són passats per referència, amb la qual cosa la modificació dels seus valors a l'interior de la funció modifica els valors a l'exterior.

Per acabar, si es crida una funció que espera certs paràmetres però aquests no se li passen en la crida, la funció emplena els paràmetres amb el valor *null*.

4.1.5. Propietats de les funcions

Quan es crea una funció, aquesta adquireix un conjunt de propietats i mètodes que s'estudiaran en apartats posteriors, però n'hi ha una de relacionada amb els paràmetres d'entrada que és especialment útil, sobretot quan s'usen biblioteques de tercers, cosa molt habitual quan es programa amb JavaScript. Es tracta de la propietat *length*.

Amb aquesta propietat, de només lectura, es pot consultar el nombre de paràmetres que admet la funció, és a dir, el nombre de paràmetres d'entrada que s'han definit en crear la funció.

Per exemple:

```
function calcula(par1,par2,par3){
   //bloc de codi
}
alert("La funció calcula es defineix amb " + calcula.length + " paràmetres.");
```

Aquest *script* mostra la finestra emergent en què indica que la funció s'ha definit amb tres paràmetres.

D'altra banda, es poden consultar els paràmetres passats a una funció mitjançant l'*array arguments*[ ] associat a la funció. Aquest *array* conté cadascun dels paràmetres utilitzats en la crida de la funció, independentment del nombre de paràmetres definits en la creació de la funció.

Aquesta característica, encara que en principi pot trencar la lògica, és molt interessant quan es necessita crear una funció amb un nombre de paràmetres variable.

L'exemple següent utilitza la propietat anterior per a la creació d'una funció que fa la suma dels valors passats com a argument d'aquesta:

```
function sumadora()
```

```
{
   var total=0;
   for (var i=0; i< sumadora.arguments.length; i++)
   {
      total += sumadora.arguments[i];
   }
   resultat=total;
}
```

Com s'observa en l'exemple, la funció s'ha definit sense cap paràmetre d'entrada, de manera que el nombre d'aquests es coneix per la longitud de l'*array* de paràmetres. El resultat de la funció s'emmagatzema en la variable global *resultat*.

En el codi anterior, s'han encadenat dues propietats: en primer lloc, tal com s'ha comentat, la propietat arguments de l'objecte funció retorna un *array* que conté cadascun dels paràmetres que s'han passat a la funció; al seu torn, l'objecte *array* disposa del mètode *length*, que indica el nombre d'elements que té l'*array*.

Amb la concatenació de les dues propietats anteriors s'ha aconseguit obtenir el nombre de paràmetres que s'han passat a la funció i s'han utilitzat com a límit superior de l'estructura *for*.

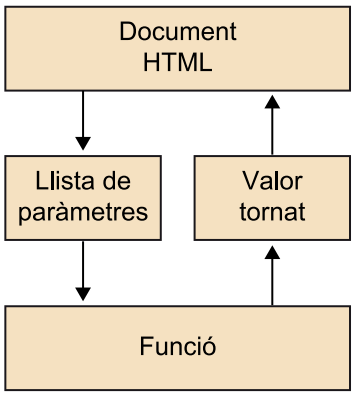
Un exemple de crida de la funció anterior podria ser:

```
alert(sumadora(2,3,6,5,4,1));
```

4.1.6. Retorn de la funció

Les funcions també poden retornar un valor, encara que, com ja s'ha vist en exemples anteriors, no és imprescindible. El valor que la funció ha de retornar s'especificarà en el cos de la funció amb la paraula reservada *return*.

Figura 9




```
function quadrat(num){
   resultat = num * num;
   return resultat;
}
```

En conseqüència, és possible assignar una funció a una variable.

```
nom_variable = nom_funcio(paràmetres)
```

A més, una funció pot contenir diverses sentències *return*, d'aquesta manera es força la finalització de la funció en executar el *return* i retorna el valor assignat. Quan una funció no disposa de la sentència *return*, retorna el valor *undefined*.

En l'exemple següent es pot observar com una funció retorna el resultat i com aquesta és cridada des d'una altra funció en el mateix *script*:

```
<html>
<head>
<title>Exemple funció</title>
<script type="text/javascript">
<!--
function quadrat(num){
   resultat = num * num;
   return resultat;
}
function calcul() {
   resul = quadrat(7);
   alert("el quadrat del número 7 és: " + resul);
}
//-->
</script>
</head>
<body bgcolor="#FFFFFF" text="#000000">
<p>Pitja el botó:</p>
<form name="elmeuForm">
<p> <input type="button" name="boto" value="Calcula" onClick="javaScript:calcul()">
</p>
</form>
</body>
</html>
```

També es podria haver escrit la funció de la manera següent:

```
function calcul() {
   alert("el quadrat del número 7 és: " + quadrat (7));
}
```

ja que, com que retorna un valor, la funció té valor per si mateixa. És a dir, que provoquem l'execució de la funció en el mateix moment en què volem consultar-ne el valor per mostrar-lo en el missatge.

4.1.7. Funcions recursives

El llenguatge JavaScript té la possibilitat de crear funcions recursives (una funció recursiva és la que en el seu propi cos es crida a si mateixa).

La recursivitat és una eina molt potent en algunes aplicacions, especialment les de càlcul, i es pot fer servir com una alternativa a la repetició o estructura repetitiva. L'escriptura d'una funció recursiva és similar a la seva homònima no recursiva; no obstant això, per a evitar que la recursivitat continuï indefinidament cal incloure una condició de finalització.

Moltes funcions matemàtiques es defineixen de manera recursiva. Un exemple d'això és el factorial d'un nombre enter *n*.

La funció factorial es defineix com:

$$n! = \begin{cases} 1 \text{ si } n=0 \\ n * (n - 1) * (n - 2) * \ldots * 2 * 1 \text{ si } n > 0 \end{cases}$$

Si s'observa la fórmula anterior, quan $n > 0$, és fàcil definir $n!$ En funció de (n # 1)!.

Per exemple, en el cas de 5! tenim:

5! = 5 * 4 * 3 * 2 * 1 = 120

4! = 4 * 3 * 2 * 1 = 24

3! = 3 * 2 * 1 = 6

2! = 2 * 1= 2

1! = 1 * 1= 1

0! = 1 = 1

Les expressions anteriors es poden transformar en:

5! = 5 * 4!

4! = 4 * 3!

3! = 3 * 2!

2! = 2 * 1!

1! = 1 * 0!

En termes generals seria:

$$n! = \begin{cases} 1 \text{ si } n=0 \\ n * (n - 1)! \quad \text{si } n > 0 \end{cases}$$

La implementació de manera recursiva en JavaScript de la funció factorial seria:

```
<html>
<head>
<title>Exemple funció recursiva</title>
<script type="text/javascript">
<!--
function Factorial(num){
   if (num==0) return 1;
   else return ( num * Factorial(num - 1) )
}
//-->
</script>
</head>
<body>
<script type="text/javascript">
   document.write( Factorial(5) )
</script>
</body>
</html>
```

4.1.8. Funcions locals

De vegades és útil limitar l'àmbit d'ús d'una funció a una altra funció, és a dir, que la primera funció solament pugui ser utilitzada a l'interior de la segona. En aquest cas es parla de *funcions locals*.

Per a crear una funció local solament és necessari declarar-la a l'interior del bloc d'instruccions de la funció que la conté:

```
function nom_funcio( [paràmetre 1, paràmetre 2, ... ] ){
   function nom_local([paràmetre 1, paràmetre 2, ...]){
      bloc de sentències
   }
bloc de sentències
```

```
}
```

4.1.9. Funcions com a objectes

Com s'estudiarà en els apartats següents, en JavaScript pràcticament tot element és un objecte i en el cas de les funcions no és una excepció. És possible definir funcions a partir de l'objecte *Function*. La sintaxi del constructor d'aquest objecte és:

```
var nomFuncio = new Function("paràmetre 1", "paràmetre 2",...,"bloc de sentències");
```

D'aquesta manera, en primer lloc es defineixen els paràmetres de la funció i, en últim lloc, s'especifica el bloc de sentències que defineix el comportament de la funció.

En l'exemple següent s'observa l'assignació de l'objecte a una variable salutació:

```
var salutacio = new Function("nom", "alert('Benvingut '+nom); ");
```

D'aquesta manera, es pot cridar la funció amb la sintaxi següent:

```
salutacio("Victor");
```

4.2. Funcions predefinides

Les funcions predefinides en JavaScript no són mètodes associats a un objecte, sinó que formen part del llenguatge per si mateixes. Aquestes funcions que permeten avaluar expressions, fer conversions de tipus i verificar el resultat d'una expressió són les que es descriuen a continuació.

1) Funció eval

Taula 14

Eval	
Descripció	Avalua i executa l'expressió o sentència continguda en la cadena de text que rep com a paràmetre.
Sintaxi	**eval(cadena de text)**
Exemple	missatge = 'Hola Joan'; eval("alert(' " + missatge + " ')");

2) Funció *parseInt*

Taula 15

parseInt	
Descripció	Converteix una cadena de text en un nombre enter segons la base indicada. Si s'omet la base, se suposa que està en base 10. Si la conversió produeix un error, retorna el valor NaN.
Sintaxi	**parseInt(cadena de text, [base])**
Exemple	Any = parseInt(“2001”); Any += 100;

3) Funció *parseFloat*

Taula 16

parseFloat	
Descripció	Converteix una cadena de text en un nombre real. Si la conversió produeix un error, retorna el valor NaN.
Sintaxi	**parseFloat(cadena de text)**
Exemple	Pi = parseFloat(“3.141516”); A = pi * r * r;

4) Funció *isNaN*

Taula 17

isNaN	
Descripció	Retorna *true* si el paràmetre és NaN, i *false* en cas contrari.
Sintaxi	**isNaN(valor)**

Exemple	if (isNaN(“2001”)) {
	alert(“No és una dada numèrica”);
	}

Verificar el retorn de la funció és útil per a validar dades introduïdes per l'usuari. El valor *NaN* és de gran utilitat en aquests casos. Per exemple, en un formulari en el qual se sol·liciti a l'usuari el seu any de naixement, un primer pas en la validació de la dada introduïda és la comprovació que s'han introduït només xifres.

```
function valida(any){
   if (isNaN(Number(any)))  alert("Dada incorrecta. Torna a introduir la dada sol·licitada")
}
```

El valor *NaN* retornat per una funció indica que s'ha produït un error.

5) Funció *isFinite*

Taula 18

isFinite	
Descripció	Retorna *true* si el paràmetre és un nombre finit, i *false* en cas contrari.
Sintaxi	**isFinite(nombre)**
Exemple	if (isFinite(2001)) alert(“ok”);

6) Funció *Number*

Taula 19

Number	
Descripció	Converteix a nombre una expressió. Si la conversió produeix un error, retorna el valor NaN.
Sintaxi	**Number(expressió)**
Exemple	Number(“2001”) retorna 2001;
	Number(“Hola”) retorna NaN;

7) Funció *String*

Taula 20

String	
Descripció	Converteix a cadena de text una expressió. Si la conversió produeix un error, retorna el valor NaN.
Sintaxi	**String(expressió)**
Exemple	String(123456);

L’exemple següent mostra diverses possibilitats en la utilització de les funcions descrites:

```
<html>
<head>
<title>Exemple funcions</title>
</head>
<body bgcolor="#FFFFFF">
<script type="text/javascript">
   vcadena = "Hola Joan";
   vnombre = 2001;
   alert(eval(vnombre + "25"));      // mostra   200125
   alert(eval(vnombre + 25));      // mostra   2026
   alert( parseInt(vcadena));      // mostra NaN
   alert( parseInt("2001"));      // mostra 2001
   alert( parseInt(3.141516));      //       mostra 3
   alert( parseInt(vnombre + 3.141516));      // mostra 2004
```

```
    alert( parseInt("24 hores"));      // mostra 24
    alert( parseFloat(vcadena));      // mostra NaN
    alert( parseFloat(vnombre));      // mostra 2001
    alert( parseFloat("3.141516"));      // mostra 3.141516
    alert(isNaN(eval(vnombre + "25")));      // mostra false
    alert(isNaN(parseInt(vcadena)));      // mostra true
    alert(isFinite(vnombre));      // mostra true
    alert(isFinite(vcadena));      // mostra false
    alert(isFinite(vnombre/0));      // mostra false
    alert(Number("1234"));      // mostra 1234
    alert(Number(vcadena));      // mostra NaN
    alert(Number("2 peixos"));      // mostra NaN
    alert(String(vnombre) + " anys");      // mostra el text 2001 anys
    alert( String(vnombre + 3.141516));      // mostra 2004.141516
</script>
</body>
</html>
```

8) Funcions *decodeURI*, *decodeURIComponent*, *encodeURI* i *encodeURI-Component*

Aquestes funcions implementades permeten fer la conversió de cadenes de text a cadenes de text URI (*uniform resource identifier*) vàlides, i viceversa. Aquestes cadenes, que per exemple poden ser adreces web, crides a CGI, etc. han d'experimentar una conversió que permeti passar cada caràcter d'un lloc a un altre per Internet.

Per exemple, el caràcter espai en blanc tindrà una conversió hexadecimal a %20.

S'utilitzen *encodeURI* i *decodeURI* per a URI completes, ja que alguns símbols com "://", que formen part del protocol, o el "?" en les cadenes de cerca o els delimitadors de directoris no es codifiquen correctament.

Per a URI simples que no continguin delimitadors s'usaran les funcions *encodeURIComponent* i *decodeURIComponent*.

Taula 21

encodeURI	
Descripció	Converteix una cadena de text que representa una cadena URI vàlida.
Sintaxi	**encodeURI(String)**

encodeURI	
Exemple	<html> <body onLoad="javascript:location.href=encodeURI('http://www.eldominio.es/pepeillo de los palotes.htm')"> </body> </html> El retorn seria el text: http://www.eldominio.es/pepeillo%20de%20els%20palotes.htm

Taula 22

decodeURI	
Descripció	Descodifica una cadena URI codificada.
Sintaxi	**decodeURI(String)**
Exemple	<html> <body onLoad="javascript:alert(decodeURI('http://www.eldominio.es/pepeillo%20de%20los%20palotes.htm'))"> </body> </html> El retorn seria: *http://www.eldominio.es/pepeillo del*spalotes.htm

Taula 23

encodeURIComponent	
Descripció	Converteix una cadena de text que representa un component d'una cadena URI.
Sintaxi	**encodeURIComponent(String)**
Exemple	<html> <body onLoad="javascript:alert(encodeURIComponent('pagina:2.htm'))"> </body> </html> El retorn seria el text: pagina%3A2.htm amb la funció encodeURI hauria retornat: pagina:2.htm ja que els ":" són un delimitador.

Taula 24

decodeURIComponent	
Descripció	Descodifica un component d'una cadena URI codificada.
Sintaxi	**decodeURIComponent(String)**
Exemple	<html> <body onLoad="javascript:alert(decodeURIComponent('pagina%3A2.htm'))"> </body> </html> El retorn seria: pagina:2.htm

9) Les funcions *escape* i *unescape*

En versions antigues dels navegadors, les funcions *escape* i *unescape* compli-en la mateixa funció que *encodeURI* i *decodeURI*, encara que aquestes no eren capaces de codificar/descodificar tots els caràcters de la recomanació RFC2396.

Taula 25

Escape	
Descripció	Converteix una cadena de text que representa una cadena URI vàlida.
Sintaxi	**escape(String, [,1])** el paràmetre opcional 1 determina si es codificaran els caràcters delimitadors.

Taula 26

Unescape	
Descripció	Descodifica una cadena de text que representa una cadena URI vàlida.
Sintaxi	**unescape(String)**

10) Funció *toString*

Taula 27

toString	
Descripció	Converteix un objecte en una cadena de text. El resultat depèn de l'objecte al qual s'aplica. Els casos són els següents: *String* : retorna el mateix *string*. *Number*: retorna l'*string* equivalent. *Boolean*: *true* o *false*. *Array*: els elements de l'*array* separats per comes. *Function*: el text que defineix la funció. Alguns dels objectes DOM que no tenen gens que retornar com *string*, si se'ls aplica la funció toString retornen alguna cosa com: [*object* tipus de l'objecte].
Sintaxi	**toString([base])**
Exemple	... var lletres= new Array("a", "b", "c") document.write(lletres.toString()) ... El resultat a la pàgina seria: a,b,c

4.3. Introducció a l'objecte Array

En aquest apartat farem una breu introducció a l'objecte Array, que és un dels objectes predefinits de JavaScript i que estudiarem amb més detall al llarg del mòdul 3.

Els *arrays* són com contenidors que ens permeten emmagatzemar múltiples variables de tot tipus i accedir-hi de manera individual utilitzant índexs numèrics començant pel zero.

4.3.1. Crear un *array*

Hi ha diferents maneres de crear un *array*:

1) Utilitzant només el constructor de l'*array*. Així s'obté com a resultat un *array* buit.

```
var vector1 = new Array();
```

2) Indicant com a paràmetre del constructor la longitud que tindrà l'*array*, així passa aquest valor com a paràmetre.

```
var vector2 = new Array(longitud);
```

3) Passant com a paràmetres del constructor els valors que contindrà l'*array*.

```
var vector3 = new Array(element0, element1, ..., elementN);
```

Per exemple, es pot definir la longitud de l'*array* i després omplir cadascuna de les seves posicions:

```
colors = new Array(16);
colors[0] = "Blau";
colors[1] = "Groc";
colors[2] = "Verd";
```

O bé, iniciar l'*array* en el mateix moment de la seva creació:

```
colors = new Array("Blau", "Vermell", "Verd");
```

4.3.2. Accés als elements

Per a accedir als elements d'un *array* s'utilitza el nom de l'*array* seguit de l'índex de l'element que s'ha de consultar tancat entre claudàtors. Sobre el valor de l'índex s'han de tenir en compte els punts següents:

- Els *arrays* s'indexen a partir del valor zero, de manera que el primer valor de l'*array* ha de tenir l'índex 0.

- Si es consulta un element de l'*array* que no ha estat assignat, l'*array* retornarà el valor *undefined*.

A continuació es mostra un exemple d'accés als elements d'un *array*:

```
var vector1 = [22, 26, 28];
var primer = vector1[0];
var segon = vector1[1];
var fora = vector1[3];
```

En l'exemple anterior, les dues primeres variables contindran els valors 22 i 26, mentre que la variable *fora* contindrà el valor *undefined*, ja que l'*array* no té cap valor emmagatzemat en la posició 3.

4.3.3. Afegir i modificar elements en un *array*

A diferència d'altres llenguatges de programació, en JavaScript no és necessari augmentar la memòria de manera explícita si s'ha d'augmentar la grandària de l'*array*. La grandària és gestionada directament pel llenguatge, la qual cosa simplifica la feina del programador. Vegem-ne un exemple:

```
vector1[3] = 32;
```

En el codi anterior s'afegeix un element a l'*array* vector1 i, com es pot veure, no s'ha fet de manera consecutiva, és a dir, és possible deixar espais buits en un *array* en cas que sigui necessari.

4.3.4. Eliminar elements d'un *array*

Els elements d'un *array* es poden eliminar utilitzant la sentència "delete":

```
delete vector1[3];
```

La sentència anterior elimina el quart element de l'*array* vector1, en el sentit que hi assigna el valor *undefined*, però no modifica la grandària de l'*array*.

4.3.5. La propietat *length*

Una propietat d'un objecte és una variable pròpia d'aquest objecte.

Vegeu també

En el mòdul "Introducció a la programació orientada a objectes" s'explica amb més detall què és una propietat d'un objecte.

Aquesta propietat emmagatzema l'índex de la posició disponible següent al final d'*array*, encara que hi hagi índexs al mig que no continguin cap valor, sempre fa referència al primer espai lliure després de l'últim element. Per exemple:

```
var vector3 = new Array();
vector3[50] = "Hola Món";
longitud = vector3.length;
```

Si bé es podria esperar que *length* retornés el valor 1, tal com s'ha explicat, *length* retornarà el valor 51, ja que aquest és el primer índex lliure després de l'últim.

Aquest comportament de la propietat *length* fa poc recomanable l'assignació de buits en els elements d'un *array*, ja que en aquests casos *length* no representa el nombre d'elements reals.

A més de la informació que proporciona la propietat *length*, es tracta d'un mecanisme realment interessant d'eliminació d'elements d'un *array*. Això és perquè qualsevol índex que contingui un valor més alt que el que està assignat a *length* s'estableix a *undefined*. D'aquesta manera, tots els valors d'un *array* es poden eliminar assignant a la propietat *length* el valor zero.