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

Vicent Moncho Mas

PID_00191132

UOC
Universitat Oberta de Catalunya

Índex

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 Protocol*), 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, situació de codi i primers passos

2.1. Característiques bàsiques

Tal com s'ha dit en l'apartat anterior, JavaScript és un **llenguatge script** que es va desenvolupar per a incrementar les funcionalitats del llenguatge HTML.

Les seves **característiques** més importants són les següents:

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

- **JavaScript és un llenguatge basat en objectes:** el model d'objectes del JavaScript inclou els elements necessaris perquè accedeixi a la informació que conté una pàgina HTML i actuï sobre la interfície del navegador.

- **JavaScript és un llenguatge orientat a esdeveniments:** permet desenvolupar scripts que executin accions en resposta a un esdeveniment que ha ocorregut en el navegador; per exemple, en resposta a un clic del ratolí a 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 velocitat i facilitat de programació, amb la qual cosa augmenta la productivitat.

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

Una altra característica dels llenguatges de script és que, normalment, són interpretats. Això, a més d'implicar l'estalvi del procés de compilació en cada modificació del codi, permet fer aplicacions més flexibles a l'hora d'implantarlos en diferents plataformes.

2.2. Situació 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 de fer-ho requereixen les etiquetes HTML: **<script>** i **</script>**.

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

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

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

El codi JavaScript es pot situar en el cos del document o bé en la capçalera del document.

JavaScript en el cos del document

Un script situat en el cos del document s'executa quan es carrega el cos o el contingut de la pàgina. Per tant, les etiquetes `<script>` i `</script>` s'han de situar 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 a 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 a la capçalera del document s'interpreta o s'executa abans que es carregui el contingut de la pàgina. La capçalera és el lloc adequat per als scripts que no modifiquen els atributs de la pàgina i que s'executen en resposta a una acció de l'usuari. A la capçalera no es pot situar un script que modifiqui parts del document i s'executi quan es carrega la pàgina, ja que aquest codi s'executa abans de carregar el document i no sap, *a priori*, els objectes que conté.

```
<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 és cridada 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 ser cridada, s'ha de posar a la capçalera del document.

En una pàgina HTML es poden fer servir tantes etiquetes `<script></script>` com faci falta. Per tant, es poden situar scripts a la capçalera i al 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>
```

Quan el navegador no suporta JavaScript

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

La manera més simple d'oferir una alternativa al contingut que ha generat 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>]
```

2.2.1. Inclusió de fitxers .js en les pàgines HTML

Quan la complexitat del web augmenta, cal que el codi escrit es pugui reutilitzar en diferents pàgines. Una alternativa a inserir el mateix codi en diferents pàgines (cada modificació d'aquest codi implicaria actualitzar totes les pàgi-

nes 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 l'extensió que tenen per a JavaScript és .js.

Etiquetes HTML

Les biblioteques JavaScript no inclouen cap etiqueta HTML.

Per a indicar al navegador que hi ha d'incloure un fitxer .js, en l'etiqueta **`<script>`** s'han d'incloure com a paràmetres el nom i el lloc 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 és 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 ens fixem en el codi font d'una pàgina HTML des del navegador, el codi del fitxer .js no apareix a la finestra. El que sí que hi apareix és la ruta completa en què hi ha el fitxer. Per tant, l'usuari pot accedir a aquesta ruta, si és accessible, i visualitzar el codi font del fitxer.

2.3. Primers passos

Per a començar a generar les pàgines, s'ha de triar un editor que permeti escriure el codi HTML i JavaScript. No és determinant l'editor que es tria, sempre que permeti generar fàcilment fitxers de text estàndard i guardar-los 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 cadascun dels passos implementats. No cal provar el codi estant connectat a Internet, sinó que es pot fer fora de línia.

Execució del codi JavaScript

El codi JavaScript es pot executar fora de línia; no fa falta estar connectat a Internet.

Per a treballar còmodament, cal tenir oberts d'una manera simultània l'editor de text en què s'escriu el codi i el navegador per a visualitzar els resultats. Els passos que s'han de seguir són aquests:

- Escriure el codi en el document de text mitjançant l'editor de text.
- Guardar la darrera versió al disc.
- Visualitzar el document en el navegador. Si el document ja s'ha visualitzat, n'hi ha prou 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 podem ser menys que els altres, i per això avançarem alguns mètodes bàsics de JavaScript que permeten fer aquestes accions.

2.3.1. El mètode write

En JavaScript, per a escriure una cadena de text a 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 a 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.

Quan es fa una crida a un mètode d'un objecte perquè dugui a terme una determinada tasca, aquest mètode ha d'anar seguit d'uns parèntesis entre els quals s'inclou la informació necessària per a fer aquesta tasca. Pot ser que un mètode no requereixi cap paràmetre o que en requereixi un o més d'un, de manera que en JavaScript, en el primer cas, es posen els parèntesis sense res entre l'un i l'altre.

Nota

El mètode write requereix un paràmetre que correspon al text que ha d'escriure en el document.

Es pot 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, s'ha d'assignar, en primer lloc, el valor a la variable, que tot seguit utilitzem com a paràmetre en la crida:

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

Ús de les cometes dobles

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

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, ho podem fer usant-lo només una vegada. Per exemple:

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

En aquest cas, l'operador + no fa la suma de valors numèrics, sinó que concatena dues cadenes de text (això vol dir que fa la unió de les dues cadenes). D'altra banda, també es poden afegir caràcters especials a les cadenes de text. Per exemple, l'escriptura d'una cadena de text més una tecla de retorn:

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

Funció alert()

Aquesta funció s'utilitza en JavaScript per a crear una finestra modal en què es mostra un missatge a l'usuari final. Aquest missatge es passa a la funció mitjançant el paràmetre d'entrada que té.

Els caràcters especials són els que es detallen en la taula 2.

Taula 2

Caràcter	Significat
\n	Línia nova
\t	Tabulador
\r	Retorn
\f	Salt de pàgina (caràcter ASCII 12)
\b	Retrocés d'un espai

Com es veu, per a mostrar l'efecte dels caràcters especials, s'ha fet servir el mètode alert 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 esperàvem; el motiu és que en el navegador mostra el que el codi HTML hi indica i no hi hem afegit l'etiqueta de salt de línia `<br>` i, per tant, encara que el text en el codi HTML el tinguem en diferents línies, el navegador no troba l'etiqueta de salt de línia i el mostra en la mateixa línia.

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

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

Així, l'exemple anterior també es pot escriure de la manera següent:

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

2.3.2. Inclusió d'etiquetes

Tal com s'ha dit en l'apartat anterior, el mètode write reescriu tot el codi font de la pàgina, per la qual cosa, si volem utilitzar el mètode i, a més, aplicar certa estructura o format HTML, hem d'introduir etiquetes HTML a 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">
        nom="Maria";
        document.write("Hola<br>" + nom);
</script>
</body>
</html>
```

Així, amb la inclusió d'etiquetes HTML, es pot generar qualsevol codi HTML d'una manera dinàmica.

2.3.3. Detectar errors durant el desenvolupament

Un dels aspectes que cal 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-los, sinó que s'executen a mesura que el navegador llegeix el codi. D'aquesta manera, quan el navegador es troba un error en el codi, ho indica, i mostra la línia i una descripció del tipus d'error que s'ha comès.

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 s'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 s'ha de fer servir per a cap més propòsit, com el d'assignar aquest nom a una variable.

En la taula 3 s'indiquen les paraules reservades de JavaScript, encara que s'ha de tenir en compte que n'hi ha algunes que, encara que no s'usen en les versions actuals, es reserven per a versions futures del llenguatge.

Taula 3. 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 tenir i el tipus d'operacions que hi podem fer. Així, per exemple, una dada de classe numèrica tindrà valors numèrics però mai valors de text.

En JavaScript hi ha els tipus de dades següents:

Taula 4

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 té dos valors possibles: *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 fan una tasca específica.	

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

3.3. Literals

S'entén per *literals* els valors que són assignats a les variables; si es té en compte la taula anterior, en tenim els següents:

- Literals numèrics: qualsevol nombre enter o real expressat en base decimal, octal o hexadecimal; per exemple: 13; –8; 125.34; 0xFF.
- Literals cadenes de text: s'expressen entre cometes dobles o simples; per exemple: "Hola", "1234", 'Pep'.
- Literals booleans: només tenen dos valors possibles: *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 comprendre'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 comprendre'l, tant l'autor en revisions posteriors com altres programadors.

Per a comentar una línia, es posen els caràcters // al començament de la línia:

```
// Línia de comentari
```

Per a comentar diverses línies, s'usen els caràcters /* (començament de comentari) i */ (final de comentari):

```
/* comentari de
```

```
diverses línies */
```

3.5. Variables

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

Cada variable de l'script ha de tenir un nom que la identifiqui. El nom pot ser format per lletres, nombres i el signe de subratllat _, però no pot tenir espais ni cap altre signe de puntuació ni començar per un nombre. Tampoc no es poden fer servir les paraules clau i, igual que el llenguatge en general, són sensibles a les majúscules i minúscules. Alguns noms vàlids per a les variables són, per exemple, *resultat*, *elmeuNum*, *_num*, *num2* i *num_1*, mentre que *2sier*, *El meu resultat* o *lameva;casa* no ho són.

Els valors que pot emmagatzemar una variable poden ser de qualsevol tipus de dades permeses 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 defineix 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 amb el nom de *resultat*, i s'hi assigna el resultat de la suma de dos nombres.

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

En JavaScript no cal 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 dèbilment tipificat**.

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 es pot fer assignar d'una 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 el lloc del codi des d'on 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 en què s'ha creat (per exemple, a dins d'una funció).

Encara que pot semblar una pràctica més simple fer servir 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, es defineix la variable amb abast global i, encara que sigui dins de l'estructura, és accessible després de sortir-ne.

> En JavaScript, una variable és local si es declara utilitzant *var* i és dins de l'estructura que defineix l'àmbit d'aquesta variable.

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

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

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

la variable *lameva_var* és de tipus cadena de text, mentre que en l'assignació següent la variable és de tipus nombre:

```
lameva_var = 1000;
```

El tipus de la variable o de la dada determina quins 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, si tenim l'assignació següent:

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

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

A més d'aquestes normes, JavaScript té 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 tots sols una expressió, però juntament amb els operadors es poden construir expressions més complexes.

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

Els operadors es classifiquen segons la funcionalitat que tenen 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 (comes, 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 5 es mostren els operadors disponibles en JavaScript:

Taula 5

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
%	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, fa que de primer es porti a cap l'increment i després 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 de primer assignem a x el valor 13, després 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 operadors.

Tanmateix, si es posiciona darrere de l'expressió, veiem que de primer fa l'assignació i després l'increment. Per exemple:

```
x = 13;
```

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

En aquest exemple, tenim l'assignació del valor 13 a la variable *x*, però tot seguit, en la línia següent, es fa, en primer lloc, l'assignació del valor de *x* a la variable *y* (pren el valor 13) i, en segon lloc, 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; es disposa d'un conjunt ampli d'operadors que permeten fer una operació aritmètica en el mateix moment en què es duu a terme l'assignació.

La taula 6 mostra els operadors disponibles i també un exemple d'ús d'aquests operadors.

Taula 6

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 més complexes. Els operadors de comparació són utilitzats en les estructures iteratives i de decisió, que es presentaran més endavant.

La taula 7 mostra els operadors de comparació i un exemple d'aquests operadors:

Taula 7

Operador	Descripció	Exemple	Resultat
==	Igualtat	3 == "3"	Cert
!=	Desigualtat	3 != "3"	Fals
<	Més petit que	3 < 3	Fals
>	Més gran que	3 > 3	Fals
<=	Més petit que o igual que	3 <= 3	Cert

Operador	Descripció	Exemple	Resultat
>=	Més gran que o igual que	3 >= 3	Cert
===	Igualtat estricta	3 === "3"	Fals
!==	Desigualtat estricta	3 !== "3"	Cert

La igualtat i desigualtat estrictes serveixen per al mateix que la igualtat i la desigualtat no estrictes, però fan una comprovació estricta de tipus. Van ser inclosos en l'estàndard ECMAScript.

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 interpretar-los, s'ha de conèixer la lògica booleana.

El comportament d'un operador lògic se sol definir mitjançant la taula de veritat corresponent, en què es mostra el resultat que produeix aplicar un determinat operador a un valor lògic o a dos. Les operacions lògiques més usuals són les següents:

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

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

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

En la taula 8 es mostren els operadors booleans en JavaScript i també un exemple de cadascun.

Taula 8

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 l'àmbit de bits, és a dir, operen amb conjunts d'uns i zeros, encara que el resultat que retornen és expressat en la manera estàndard de JavaScript per als valors numèrics.

Taula 9

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 només 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*& = *y* equival a *x* = *x*&*y*
|=	*x*| = *y* equival a *x* = *x*|*y*
^=	*x*^ = *y* equival a *x*=x^*y*
<<=	*x*<< = *y* equival a *x* = *x*<<*y*
>>=	*x*>> = *y* equival a *x* = *x*>>*y*
>>>=	*x*>>> = *y* equival a *x* = *x*>>>*y*

No es tracta d'operadors que tinguin un ús comú en un llenguatge script; l'ús que tenen 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 segons el 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.

Si hi ha, però, un dels operands que és del tipus cadena de text, l'operador + es transforma en l'operador que fa la concatenació de les dues expressions. Per exemple:

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

La variable *frase* adquireix el valor "Hola, Maria".

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

3.6.7. L'operador :?

Aquest operador és una abreviació de l'estructura de control simple *if-then-else*. Aquesta estructura s'estudiarà amb més detall en l'apartat següent, però la sintaxi es presenta tot seguit:

```
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
```

s'assigna a la variable *x* el valor 1. Això és així perquè es compleix la condició (5 < = 8) i, per tant, l'expressió torna 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 sola línia:

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

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

L'ús dels parèntesis en la creació d'expressions aritmètiques complexes té una importància especial, 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 una *array* o a un vector; aquests signes es presentaran més endavant.

3.6.9. Operadors per a treballar amb objectes

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

Taula 10

Operador	Descripció	Exemple
new	Crea un objecte.	avui = new Date()
delete	Destrueix un objecte.	delete avui
this	Referencia l'objecte actual.	
in	Ítem a l'objecte.	
typeof	Instància de l'objecte.	
void	No retorna cap valor.	

Els operadors anteriors s'estudiaran amb detall en el mòdul de programació orientada a objectes.

3.6.10. Precedència d'operadors

En una expressió complexa en què intervé més d'un operador es pot determinar l'ordre en què s'han d'avaluar aquests operadors amb l'ús de parèntesis. Per exemple:

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

mostra el valor 1 en el document HTML, mentre que l'expressió següent:

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

mostra 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 11. 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	< <= > >=
Comparació d'igualtat i desigualtat	== !=
AND binari	&
XOR binari	^
OR binari	|
AND lògic	&&
OR lògic	||
Condicional	:?
Assignació	= += -= *= /= %= &= |= ^= <<= >>= >>>=
Coma	,

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

Vegem-ne tot seguit un parell d'exemples:

```
x = 5+4*3; // el producte té més prioritat. x = 17
x = ++x*3;// es calcula l'increment i a continuació el producte i el resultat és x = 54
```

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 faci 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, en la forma més simple de la qual pot ometre else, i en els casos més complexos, pot prendre la forma if... else if...
- switch

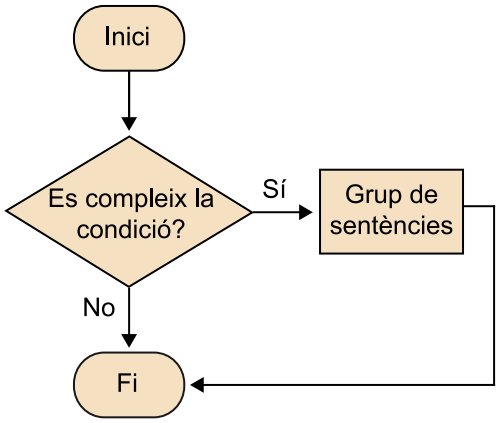
3.7.1. Estructura de control if...

El funcionament és molt bàsic: es duu a terme una acció específica en cas que es compleixi una condició. En general, la sintaxi és la següent:

```
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 es poden ometre, però aquestes claus també tenen un efecte delimitador per al programador i ajuden a visualitzar les accions definides en l'estructura.

Tot seguit es presenta un exemple en què es mostra un text addicional a la pàgina, si l'usuari ha 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 una caixa de diàleg modal formada pels botons Acceptar i Cancel·lar, un text definit pel primer paràmetre enviat a la funció i un camp de text amb un valor predeterminat definit pel segon paràmetre.

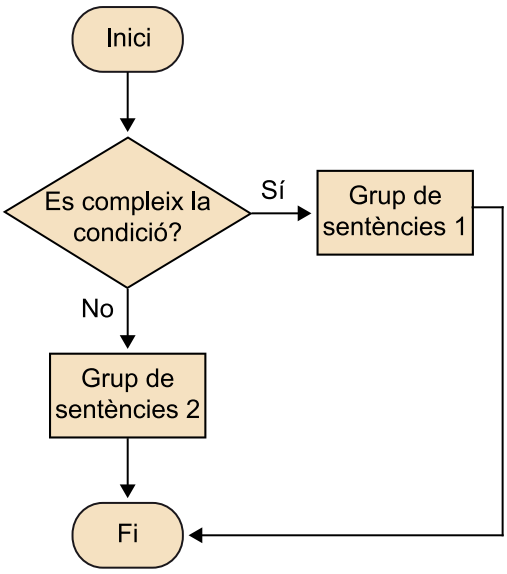
La funció torna el valor inserit en el camp de formulari si l'usuari prem Acceptar o null si prem Cancel·lar o l'aspa de tancar.

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 s'ha de fer per a 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 queda així:

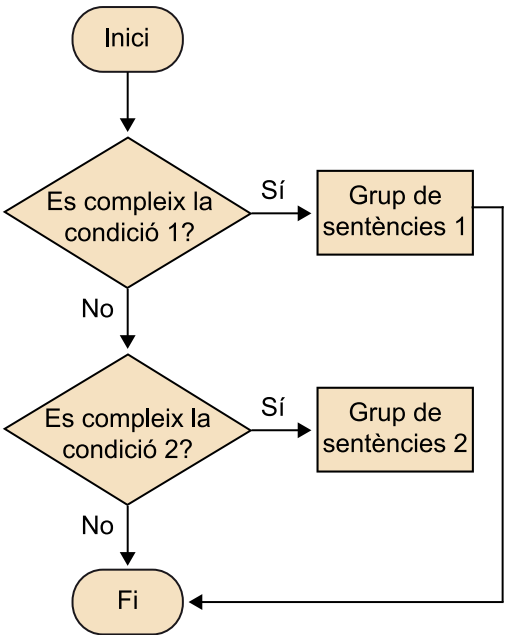
```
resp == "8" ? document.write ("Som a l'agost"): document.write("No som a l'agost");
```

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

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

```
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");
}
```

S'hi poden afegir tantes estructures else if com faci falta, encara que per a aquests casos és més adequada l'estructura switch, que es presenta tot seguit.

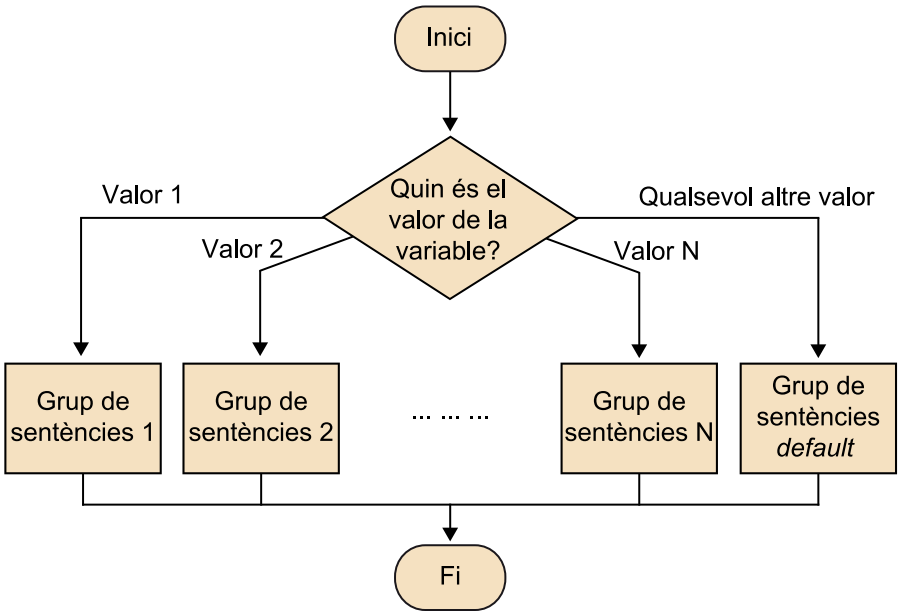
3.7.4. L'estructura de control switch

L'estructura switch es basa a avaluar una variable, de manera que en el cos de l'estructura es compara amb un conjunt de valors. Així, doncs, en cas de coincidència, s'executa el conjunt de sentències que està associat 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 papel de la palabra reservada *break* es salir de la estructura, de manera que no sigue con las sentencias que hay en las siguientes opciones. Por tanto, se puede obligar a pasar por las sentencias correspondientes a dos valores, con tan sólo omitir la instrucción *break* en las sentencias del primero.

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 fet falta emprar la funció parseInt(), ja que el valor que ha tornat la funció prompt() és una cadena de caràcters i en el switch es fan servir valors numèrics.

En l'exemple anterior de l'estructura, if... else if..., no ha calgut fer la conversió, ja que l'operador (==) no compara d'una manera estricta.

És important tenir en compte que dins del conjunt d'accions de cadascuna de les estructures anteriors es pot introduir una estructura condicional nova, 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 és determinat per la condició d'acabament, encara que se'n pot interrompre l'execució amb 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 que s'ha de portar a cap 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. Aquesta variable es pot usar de manera que incrementi el valor que té, en una certa quantitat, en cada iteració fins a un valor determinat o de manera que el valor inicial decreixi en cada iteració.

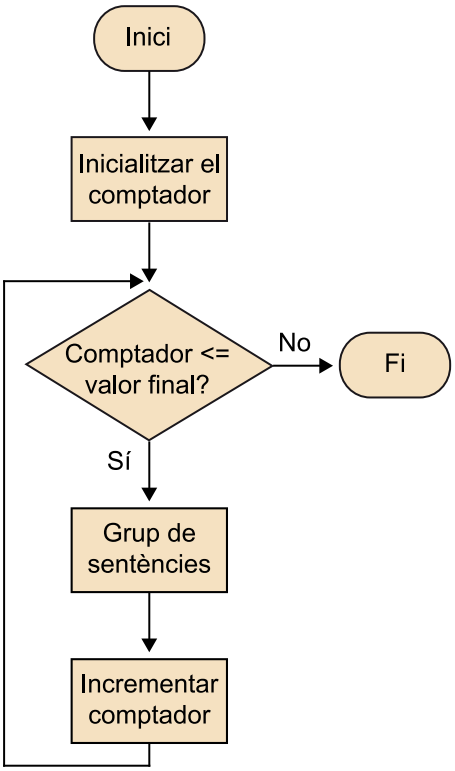
Amb un comptador ascendent l'expressió té la forma següent:

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

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

```
for (comptador = valor inicial; comptador > = valor final; comptador -= increment ){
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>");
}
```

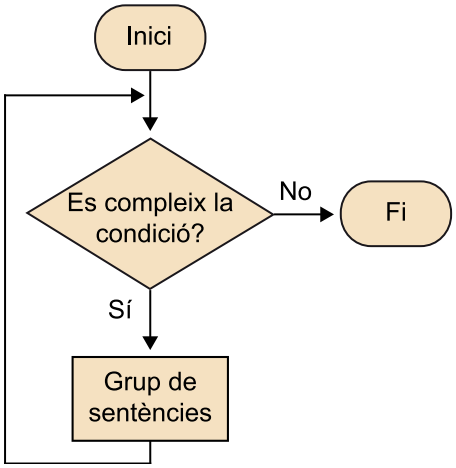
En la primera línia del for de l'exemple, l'expressió *(var i = 0; i < 11; i++)* fa el següent: de primer, inicialitza la variable *i* al valor 0, executa el cos del bucle; després, en els passos següents del bucle, el valor de la variable *i* augmenta en una unitat a causa de la tercera entrada, i++.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

3.8.2. L'estructura de control while

L'estructura de control while es basa a dur a terme un grup d'accions en cas que es compleixi una certa condició. Pot ser que aquest grup d'accions no s'executi mai si la condició no es compleix, ja que aquesta condició és al començament 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 petits o iguals que 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 es demana a l'usuari el valor numèric; aquest valor que es converteix a numèric amb el mètode parseInt, ja que, com que es demana amb el mètode prompt, el valor és emmagatzemat com a cadena de text. Comença el codi amb una estructura if que comprova que el valor que s'ha introduït és més gran que zero i, en cas que sigui així, executa l'estructura while.

El condicional de l'estructura while comprova que la variable és 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'arriba al valor 0 i, com que es deixa de complir el condicional, no s'executa cap més pas del bucle.

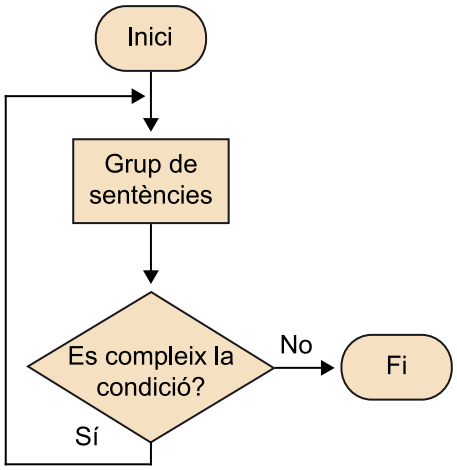
3.8.3. L'estructura de control do... while

Aquesta estructura es diferencia de l'anterior en el fet que les accions s'executen almenys una vegada. Això és així perquè la condició d'acabament és 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. Tot seguit, se sol·licita a l'usuari el valor "s", en cas que vulgui executar un pas nou del bucle. D'aquesta manera, en la línia següent es comprova el valor que ha introduït l'usuari i, en cas afirmatiu, executa 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. Tot seguit es presenta un exemple en què es fa servir la instrucció break per a detenir 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 amb la sentència break, encara que la variable de control continuï complint la condició del bucle.

No s'aconsella utilitzar el break 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 fer servir 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>");
          }
}
```

3.8.6. La sentència with

La sentència with permet usar un grup de sentències que fan referència a un objecte, de manera que no faci falta indicar, en cadascuna de les sentències, 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);
```

Un array

És una estructura semblant a una variable, però s'hi poden emmagatzemar diversos valors. D'aquesta manera els arrays permeten guardar diversos valors i accedir-hi d'una manera independent, utilitzant un índex que marca la posició del valor que es vol consultar. Com que es tracta d'una estructura o un objecte, disposa d'un conjunt de mètodes i propietats, que són cridats segons la sintaxi nom_objecte.metode.

Sense l'estructura with, s'ha d'especificar el nom de l'array amb 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 pot ser cridada des de qualsevol part de l'aplicació.

Una funció es compon de vàries sentències que, en conjunt, fan una tasca determinada.

4.1.1. Definició de la funció en JavaScript

Una *funció* es defineix amb l'ús de la paraula *function* seguida dels elements següents:

- El nom de la funció.
- La llista de paràmetres de la funció, inclosos entre parèntesis i separats per comes.
- Les sentències JavaScript que defineixen la funció, incloses entre claus.

És a dir:

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

L'existència d'arguments, i també el nombre d'aquests arguments, és opcional i depèn de cada funció concreta.

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 12

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 el començament 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>`, i la ubicació més bona és la capçalera del document, ja que la capçalera es carrega abans que el cos del document. En el cas de les funcions, encara que es referenciï a objectes de la pàgina, la funció es carrega, però no s'executa fins que no és cridada, de manera que no es produeix cap error. Les funcions també es poden definir en el cos del document o en un fitxer extern. En un document extern es poden incloure tantes funcions com siguin necessàries.

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

Si hi ha més d'una funció, se situen també 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>
```

4.1.3. Crida i execució de les funcions

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

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

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

Perquè una funció i s'executi ha de ser cridada.

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 surt d'una pàgina, es produeix un esdeveniment.

Controladors d'esdeveniments

La definició de les accions que s'han d'executar es fa 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 un clic.

En l'exemple següent:

```
<html>
&lt;input type=button value="Pitjar" onclick="sentencies_javascript
 ..."&gt;</html>
```

s'ha afegit un atribut nou a l'etiqueta que té el mateix nom que l'esdeveniment i que indica el codi que s'ha d'executar quan passa aquest esdeveniment. Cada element té la seva pròpia llista d'esdeveniments suportats.

La gestió d'esdeveniments s'estudia en l'apartat 4 del mòdul "Introducció al DOM".

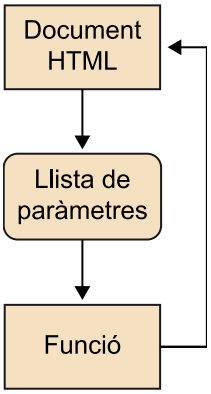
4.1.4. Ús dels paràmetres de la funció

Els paràmetres són les dades d'entrada, que la funció necessita per a fer una tasca. Una funció pot rebre tants paràmetres com faci falta.

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

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

Figura 8



L'exemple següent mostra que una mateixa funció pot ser cridada 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="miform">
<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 visualitza en el navegador és diferent per a cada crida, perquè depèn del valor del paràmetre.

En aquest exemple es pot veure també que en alguns casos fa falta alternar l'ús de les cometes dobles amb el de les cometes simples. Concretament, en la línia següent:

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

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

El motiu d'això és no confondre el navegador, ja que les cometes dobles es fan servir per a indicar el començament i l'acabament de la crida. Per exemple, si s'escriu el següent:

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

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

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

A més, quan es passa una variable com a paràmetre d'una funció, aquesta es passa per valor, és a dir, la manipulació a dins de la funció de la variable no modifica el valor de la variable a fora de la funció. Per exemple, el següent script:

```
num=2;
function funciona(num){
   num=3;
   alert(num);
}
funciona(num);
alert(num);
```

Assigna a la variable el valor 2, però crida a la funció funciona, que canvia el valor de la variable a 3 i ho mostra amb un alert. A continuació hi ha un nou alert que mostra el valor de la variable, però com que està fora de la funció aquest emmagatzema el valor 2.

Ara bé, aquesta característica no es compleix sempre, ja que quan s'hi passen tipus compostos com arrays o objectes, aquests tipus són passats per referència, de manera que la modificació dels valors a dins de la funció modifica els valors a fora. Per exemple, si modifiquem el codi anterior, on la variable ara és un array (aquesta estructura es presentarà en un mòdul posterior):

```
var num=new Array();
num[0]=2;
function funciona(num){
   num[0]=3;
   alert(num[0]);
}
funciona(num);
alert(num[0]);
```

Els dos alerts tornen el valor 3, ja que en aquest cas l'assignació realitzada a l'interior de la funció sí que modifica la variable original, ja que aquestes estructures no passen el valor que contenen, passen la referència d'on és aquesta.

Per acabar, si es fa una crida a una funció que espera certs paràmetres, però aquests paràmetres no es passen en la crida, la funció emplena els paràmetres amb el valor "null" o "undefined".

Per exemple, en el codi següent, com que no tenim valor a la variable let, el segon alert tornarà "undefined":

```
num=2;
function funciona(num,let){
   alert(num);
   alert(let);
}
funciona(num);
```

4.1.5. Propietats de les funcions

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

Mitjançant aquesta propietat, només de 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 quan s'ha creat 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 i 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[ ] associada a la funció. Aquesta array conté cadascun dels paràmetres que s'han utilitzat en la crida de la funció, independentment del nombre de paràmetres que s'hagin definit en la creació de la funció.

Aquesta característica, encara que d'entrada pot trencar la lògica, és molt interessant i imprescindible quan cal crear una funció amb un nombre de paràmetres variable.

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

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

Tal com es veu en l'exemple, la funció s'ha definit sense cap paràmetre d'entrada, de manera que el nombre d'aquests paràmetres se sap 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:

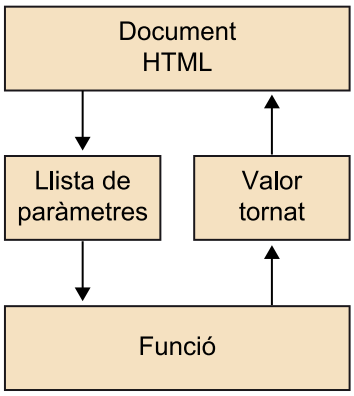
- En primer lloc, tal com s'ha dit, la propietat arguments de l'objecte Function torna un array que conté cadascun dels paràmetres que s'han passat a la funció.
- En segon lloc, l'objecte Array disposa del mètode length, que indica el nombre d'elements que té l'array.

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.

4.1.6. Retorn de la funció

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

Figura 9




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

En conseqüència, es pot 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 l'acabament de la funció en executar el return i torna el valor assignat. Quan una funció no disposa de la sentència return, torna el valor undefined.

En l'exemple següent es pot veure com torna el resultat una funció i com 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="miform">
<p> <input type="button" name="boto" value="Calcula" onClick="javaScript:calcul()">
</p>
</form>
</body>
</html>
```

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

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

ja que, com que torna 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 el valor que té per mostrar-ho en el missatge.

4.1.7. Funcions recursives

Funcions recursives

Una funció recursiva és la que en el seu propi cos es crida a si mateixa.

El llenguatge JavaScript té la possibilitat de crear funcions recursives. 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 semblant a l'homònima no recursiva; tanmateix, per a evitar que la recursivitat continuï indefinidament, s'hi ha d'incloure una condició d'acabament.

Hi ha moltes funcions matemàtiques que es defineixen d'una manera recursiva. N'és un exemple el factorial d'un nombre enter *n*. La funció factorial es defineix com a:

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

Si ens fixem en la fórmula anterior, quan n > 0, és fàcil definir n! en funció de (n -1)!

Per exemple, en el cas de 5! tenim el següent:

$$5! = 5 * 4 * 3 * 2 * 1 = 120$$
$$4! = 4 * 3 * 2 * 1 = 24$$

$$3! = 3 * 2 * 1 = 6$$
$$2! = 2 * 1 = 2$$
$$1! = 1 * 1 = 1$$
$$0! = 1 = 1$$

Les expressions anteriors es poden transformar en les següents:

$$5! = 5 * 4!$$
$$4! = 4 * 3!$$
$$3! = 3 * 2!$$
$$2! = 2 * 1!$$
$$1! = 1 * 0!$$

En termes generals, és així:

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

La implementació d'una manera recursiva en JavaScript de la funció factorial és de la manera següent:

```
<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

A vegades, és útil limitar l'àmbit d'ús d'una funció a una altra funció, és a dir, que la primera funció només es pugui utilitzar dins de la segona. En aquest cas, es parla de *funcions locals*.

Per a crear una funció local, només fa falta declarar-la a dins 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 les etapes següents, en JavaScript gairebé tot element és un objecte i, en el cas de les funcions, no és una excepció. Es poden definir funcions a partir de l'objecte Function; la sintaxi del constructor d'aquest objecte és la següent:

```
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 segon lloc, s'especifica el bloc de sentències que defineix el comportament de la funció.

En l'exemple següent es veu l'assignació de l'objecte a la variable *salutacio*:

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

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

```
salutacio("Víctor");
```

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 tot seguit:

1) Funció eval

Taula 13

Descripció	Avalua i executa l'expressió o sentència que conté la cadena de text que rep com a paràmetre.
Sintaxi	**eval(cadena de text)**
Exemple	eval("x=10;y=30;document.write(x+y)"); escriu el valor 40 a la pàgina web.

2) Funció parseInt

Taula 14

Descripció	Converteix una cadena de text en un nombre enter segons la base indicada. Si s'omet la base, se suposa que és 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 15

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 16

Descripció	Retorna true si el paràmetre no és un nombre i false quan és un nombre.
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 que ha introduït l'usuari. El valor NaN és d'una gran utilitat en aquests casos. Per exemple, en un formulari en què se sol·licita a l'usuari l'any de naixement, un primer pas en la validació de la dada introduïda és comprovar que s'han introduït només xifres.

Quan una funció torna el valor NaN, indica que s'ha produït un error.

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

5) Funció isFinite

Taula 17

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 18

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

7) Funció String

Taula 19

Descripció	Converteix en 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 encodeURIComponent

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 o crides a CGI, 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 té 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'usen les funcions encodeURIComponent i decodeURIComponent.

Taula 20. EncodeURI

Descripció	Converteix una cadena de text que representa una cadena URI vàlida.
Sintaxi	**encodeURI(String)**
Exemple	<html> <body onLoad="javascript:location.href=encodeURI('http://www.eldominio.es/pepeillo de los palotes.htm')" > </body> </html> El retorn és el text següent: http://www.eldominio.es/pepeillo%20de%20los%20palotes.htm

Taula 21. 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 és aquest: http://www.eldominio.es/pepeillo de los palotes.htm

Taula 22. 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 és el text següent: pagina%3A2.htm Amb la funció encodeURI hauria tornat: pagina:2.htm, ja que els ":" són un delimitador.

Taula 23. 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 és aquest: pagina:2.htm

9) Les funcions escape i unescape

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

Taula 24. 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 25. Unescape

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

10) Funció toString

Taula 26

Descripció	Converteix un objecte en una cadena de text. El resultat depèn de l'objecte a què s'aplica. Els casos són els següents: String: retorna la mateixa 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 han de retornar res, com string, si s'hi aplica la funció toString retornen una 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 és aquest: a,b,c

Activitats

Exercicis a desenvolupar, compartir i discutir al fòrum de l'aula

1. Creeu un script que calculi l'import d'una venda en un comerç: l'usuari ha d'introduir el nom del producte, el preu per unitat i el nombre d'unitats. L'script mostrarà en una finestra modal el nom del producte, el nombre d'unitats i el preu total.

2. Suposant que s'assignen els valors 7 i 2 a les variables x i y respectivament, calculeu i expliqueu el valor de la variable y en els casos següents:

a) $y = -2 + -x$;

b) $y \mathrel{+}= 2$;

c) $y = (y == x)$;

d) $y = y{+}{+} -x$;

3. Creeu un script que sol·liciti dos nombres i mostri el més gran en una finestra modal i que en cas que aquests nombres siguin iguals ho indiqui.

4. Creeu un script que sol·liciti un nombre i indiqui si aquest nombre és parell o senar.

5. Creeu un script que sol·liciti una nota d'una prova i indiqui "Apte" si aquesta nota és més gran que 5 i "No apte" en cas contrari.

6. Creeu un script que sol·liciti dues dates de naixement i que retorni la diferència d'edat en anys, mesos i dies.

7. Creeu un script per jugar a endevinar un nombre entre 1 i 10 (generat a l'atzar) fins que s'encerti o s'hagin dit els 10 possibles.

8. Creeu un script que sol·liciti un nombre i que en mostri la taula de multiplicar.

9. Creeu un script que calculi la mitjana d'una llista de nombres positius que finalitza amb un nombre negatiu.

10. Creeu un script que llegeixi deu nombres, els guardi en un vector i els mostri en l'ordre invers de la seva entrada.

11. Creeu una funció que calculi la mitjana de diversos nombres introduïts, on aquests poden ser un nombre variable de valors.

12. Creeu una funció que intercanvia el valor de dos variables, es a dir, si $x = 4$ i $y = 2$, després d'aplicar la funció $x = 2$ i $y = 4$.

13. Creeu una funció que calculi si un any és de traspàs o no.

14. Creeu una funció que calculi si un nombre és múltiple d'un altre.

15. Creeu una funció que, utilitzant l'anterior, calculi el nombre de triennis que té un treballador a partir del nombre d'anys treballats.