Apunts UD4 (JavaScript I)

lloc: CIFP Francesc de Borja Moll
Curs: Llenguatges de marques i sistemes de gestió d'informació
Llibre: Apunts UD4 (JavaScript I)
Imprès per: Alejo Morell Bethencourt
Data: dilluns, 21 desembre 2020, 15:46

Taula de continguts

1. JavaScript

Introducció

En l'inici de l'era d'Internet, les pàgines web eren bàsicament estàtiques, mostraven un contingut fix sense possibilitat d'interacció amb l'usuari. Però quan va aparèixer "La Web 2.0", passant per La Web 3.0, i actualment La Web 4.0, si una cosa defineix el món web, és l'increment d'interacció i usabilitat, millorant l'experiència de l'usuari en la navegació. És aquí on el llenguatge JavaScript té un rol important. **La programació en el cantó del client codificada dins les pàgines web és la responsable de fer pàgines web atractives tal com les coneixem avui dia**.

Un poc d'història

És important dir que el llenguatge ha evolucionat per dos costats:

- Per el propi llenguatge: Ha anat incorporant operadors, estructures de control, regles de sintaxi, etc.
- Per els navegadors: Han anant incorporant noves instruccions (per manipular nous elements).

Els autors de JavaScript, van proposar el 1997 a l'*European Computer Manufacturers Association* (ECMA) que el seu llenguatge es definís com a estàndard. Així és com va néixer l'ECMAScript, que és l'estàndard de definició del llenguatge Javascript. Al llarg del temps, s'han anant publicant distintes versions d'ECMAScript. En els inicis d'Internet hi havia bastants problemes de compatibilitat entre navegadors (*Internet Explorer, Netscape Navigator, Opera*) a diferents nivells, i això va fer prendre consciència de la importància en l'adopció d'estàndards.

- A partir del 2012, tots el navegadors actuals suporten completament ECMAScript 5.1. Des de Juny 17, 2015, ECMA International, va publicar la versió número 6, que oficialment s'anomena ECMAScript 2015, i que inicialment se va dir ECMAScript 6 o ES6.

Per tal d'avançar en la compatibilitat entre navegadors web, no només és important que implementin el motor de JavaScript segons els estàndards ECMAScript, sinó que els diferents elements que hi ha en una web (botons, caixes de text, enllaços...) es comportin de la mateixa manera i responguin als mateixos *events* (esdeveniments). És per això que el W3C va definir el *Document Object Model* (DOM), o Model d'Objectes del Document, que defineix un estàndard d'objectes per representar documents HTML i/o XML.

Què és JavaScript

JavaScript és un llenguatge de guions (*script*, en anglès). És conegut sobretot pel seu ús en pàgines web, però també es pot utilitzar per realitzar tasques de programació i administració que no tenen res a veure amb la web.

El nom de JavaScript no deriva del llenguatge de programació Java, tot i que tots dos comparteixen una sintaxi similar. Semànticament, JavaScript és més pròxim als llenguatges *Self i ActionScript* (basat també en l'ECMAScript). El nom JavaScript és una marca registrada per *Oracle Corporation*.

Llenguatges interpretats o de guions

Els llenguatges de programació es divideixen, quant a la manera d'executar-se, entre els interpretats i els compilats. Alguns **llenguatges interpretats s'anomenen també llenguatges de guions** (*scripts* en anglès).

- Els llenguatges interpretats s'executen mitjançant un intèrpret, que processa les ordres que inclou el programa una a una.
- Els llenguatges compilats necessiten del compilador com a pas previ a l'execució.

JavaScript és un llenguatge interpretat.

Motors de JavaScript

Un motor de JavaScript és un programa que executa codi JavaScript, és a dir, el que s'encarrega de interpretar el codi de JavaScript en el navegador.
Els més coneguts els següents:

- SpiderMonkey: Motor emprat principalment en els navegadors web Firefox.
- V8: Motor emprat en els navegadors basats Chromium, actualment és un dels motors més utilitzats i el millor. El motiu és que la seva compilació se fa en temps d'execució i no abans, això fa que millori significativament els temps d'execució (compilació just a temps ò JIT).
- JavaScriptCore: Motor emprat principalment en els navegadors web Safari, després va evolucionar a Nitro (compila codi JS en codi màquina de manera nativa).
- Chakra: Motor emprat en el navegador web Edge, encara que actualment està sent reconstruït com a navegador basat en Chromium, per lo que en futures versions utilitzarà V8

> **Execució de JavaScript en línia d'ordres**: Javascript és un llenguatge *single-thread*, és a dir, té només un fil d'execució, per la qual cosa les tasques es van executant seqüencialment.

Avantatges de JavaScript en les pàgines web

- Permet una gran quantitat d'efectes dins les pàgines. Entre d'altres: finestres emergents *(Pop-up),* desplaçaments *(scrolls)*, transicions d'imatges, etc.
- Afegeix interactivitat amb l'usuari.
- Proporciona integració amb altres connectors (*plugins en anglès)*: no proporciona només accés als objectes HTML, també proporciona accés a objectes específics del navegador com Adobe Acrobat, Media Player, etc.
- Permet validació dels formularis en el costat del client. Una validació inicial dels formularis és possible per eliminar simples errors, com per exemple: com assegurar-se de què el format de data, DNI, correu electrònic o telèfon són correctes. Com a resultat, l'usuari té una resposta més ràpida que si el control es fes en el costat del servidor.
- Permet accedir a informació del sistema.

Desavantatges de JavaScript en les pàgines web

- La seguretat és el principal problema a JavaScript. Els fragments de codi JavaScript que s'afegeixen a les pàgines web es descarreguen en els navegadors i s'executen en el costat del client, permetent així la possibilitat que un codi maliciós es pugui executar en la màquina client i així explotar alguna vulnerabilitat de seguretat coneguda en les aplicacions, navegadors o fins i tot del sistema operatiu. Per prevenir això, existeixen estàndards de seguretat que restringeixen

l'execució de codi, en els navegadors. Aquestes restriccions són per exemple, la de deshabilitar l'accés a l'escriptura o lectura de fitxers.

- Tendeix a introduir una quantitat enorme de fragments de codi en els nostres llocs web. Això es resol fàcilment emmagatzemant el codi JavaScript en fitxers externs amb extensió JS. Així la pàgina web queda molt més neta i llegible. Actualment quasi sempre es separa totalment la part del contingut HTML, la part de funcionalitat JavaScript, i la part de disseny i maquetació (CSS), així tendrem tres perfils d'usuaris diferents (el que genera continguts HTML, el programador web, i el dissenyador), que podran treballar en el mateix projecte però en arxius diferents.

1.1. Estructura

IMPORTANT:

- És important saber que JavaScript distingeix entre majúscules i minúscules, és a dir, és case-sensitive.
- Utilitza el conjunt de caràcters Unicode.
- No té en compte els espais en blanc ni les noves línies.

Comentaris

Quan el comentari ocupa una sola línia s'escriurà a partir de dues barres (//), i quan el comentari s'escriu en més d'una línia anirà entre / * * /.

```
<script>
   // Aquest és un comentari d'una única línia
   alert ("Exemple de comentaris en línia"); //Aquí hi pot haver-hi un altre comentari en una lí
nia
</script>
```

```
<script>
    alert ("Exemple de comentari multi-línia");
    /* Una línia de comentari
       Una altra línia de comentari */
</script>
```

Hi ha una costum bastant estesa que consisteix en afegir un * (asterisc) a cada línia del bloc comentat per millorar la lectura identificació del bloc comentat:

```
<script>
    /*
    * Això és una línia de comentari
    * Això és una segona línia de comentari
    */
</script>
```

INCÍS: alert () és una declaració simple, (una funció), que mostra un quadre de diàleg que conté un missatge. El missatge es col·loca entre cometes dins del parèntesi de la funció

Estructura del codi

On col·locam el codi JavaScript

El codi JavaScript es defineix a través de l'element <script>. Aquesta té un atribut de tipus, que s'utilitza per indicar el tipus de llenguatge que utilitzarem (en aquest cas serà JavaScript). Amb HTML 4 i XHTML 1.x, el tipus d'atribut és obligatori, en canvi, amb HTML5, no ho és.

- **JavaScript dins la pàgina**

  Para situar el codi JavaScript directament a la pàgina web, es col·loca dins de l'element <script>, al <body>.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Dins de la pàgina</title>
    </head>
    <body>
       <script>
           alert ('Codi dins de la pàgina'); //La instrucció alert() mostra un missa
tge per pantalla.
       </script>
   </body>
</html>
```

- **JavaScript extern**

  Es convenient en molts de casos, escriure el codi JavaScript en un arxiu extern amb l'extensió "js". Aquest arxiu es crida des de la pàgina web mitjançant:
  - Amb la instrucció <script src="./arxiujavascript.js"></script>, que pot anar al <head> o al <body>.

```
Codi JavaScript (Arxiu "exemple.js"):
alert ('Un exemple de JavaScript extern');
```

```
Codi HTML:
<! DOCTYPE html>
<html>
   <head>
       <title>JavaScript extern</title>
   </head>
   <body>
       <script src="exemple.js"></script>
   </body>
</html>
```

```
Nota: Suposam en aquest exemple que l'arxiu "exemple.js" es troba dins del mateix di
rectori que l'arxiu del codi HTML.
```

Codi HTML:

```
<!DOCTYPE html>
<html lang="es">
   <head>
      <meta charset="utf-8" />
      <title>Incloent JS en HTML5</title>
      <script src="arxiujavascript.js"></script>
   </head>
   <body>
      <h1> Pàgina exemple </h1>
   </body>
</html>
```

arxiujavascript.js:

```
document.write ('Un arxiu JavaScript extern!');  //És un mètode del DOM que permet e
scriure un text a la pàgina.
```

Recordem: Per referenciar els arxius externs, hem d'emprar rutes relatives:

- Si l'arxiu està al mateix directori o carpeta des d'on el cridam, escriurem directament el nom del arxiu.
- Si l'arxiu està dins d'una carpeta o directori superior hem d'indicar el nom de la(s) carpeta(s): carpeta/nom_arxiu.
- Si l'arxiu es troba a una carpeta(s) inferior posarem: ../

- **Posició de l'element <script>**

  Una pàgina web es llegida per un navegador de forma lineal, és a dir, primer el <head>, després els elements del <body> un darrere l'altre. Si es crida un arxiu de JavaScript des del principi de la pàgina, el navegador, el carregarà primer, i si l'arxiu és molt gran, la càrrega farà que s'alenteixi la resta de la pàgina, i el contingut tardarà a aparèixer.
  Per evitar això, serà convenient (però no obligatori), col·locar els elements <script> just abans de tancar el <body>.

  Codi HTML:

  ```
  <!DOCTYPE html>
  <html>
      <head>
          <title>Col·locam scripts</title>
      </head>
      <body>
          <!-- Contingut de la pàgina Web -->
          <p> Text del paràgraf </p>
          <script>
               <!--Contingut del JavaScript-->
          </script>
          <script src="exemple.js"></script>  <!-- Cridam l'arxiu de JavaScript --
  >
      </body>
  </html>
  ```

1.2. Variables i valors

Identificadors

Els identificadors són noms i han de ser únics. S'utilitzen identificadors per anomenar variables (paraules clau, funcions i etiquetes). Les normes per a noms legals són el mateix en la majoria de llenguatges de programació:

- El primer caràcter ha de ser una lletra, un guió baix (_) o un signe de dòlar ($).
- Els caràcters posteriors poden ser lletres, dígits, guions baixos o signes de dòlar.
- No es permeten números com a primer caràcter.

Variables

La creació d'una variable a JavaScript s'anomena "declarar" una variable. Declarar una variable vol dir, que es reserva espai d'emmagatzemament de reserva en memòria.Les variables han de ser identificadors únics i han de complir les següents normes:

- Només poden contenir lletres, dígits, signe de subratllat (_) i el signe de dòlar ($).
- Han de començar amb una lletra, o bé amb el signe de subratllat (_) o el signe de dòlar ($).
- Recordar que es distingeix entre majúscules i minúscules (la variable x és diferent de la variable X).
- No es poden utilitzar les paraules reservades de JavaScript (while, for, next…).

Abans d'ES6, la única forma de crear variables en JavaScript era amb la paraula "var". Però a partir d'aquest estàndard hi ha tres tipus de declaracions en JavaScript:

```
var: Declara variables en àmbit global.
let: Declara variables locals en àmbit local.
const: Declara constants de només lectura.
```

- **Variables d'àmbit global i d'àmbit local**

  Una variable d'àmbit global es pot emprar en qualsevol lloc de l'script. Una variable d'àmbit local només té validesa dins de l'àmbit d'una funció.
  Les normes que s'apliquen per definir el tipus de variable (global o local), es resumeixen en:
  - Qualsevol variable declarada o no declarada a l'arrel d'un script és sempre d'àmbit global.
  - Qualsevol variable declarada dins d'una funció és d'àmbit local.
  - Una variable no declarada dins d'una funció adquireix àmbit global.

  En cas de dubte, si una variable no ha estat declarada dins d'un àmbit determinat, la variable adquireix comportament global.

  Les variables de JavaScript poden contenir números i també valors de text:

  - Els valors de text s'anomenen cadenes de text.
  - Les cadenes s'escriuen dins de cometes dobles o simples.
  - Els números s'escriuen sense cometes.

```
var pi = 3.14;
var persona = "Juan Más";
```

- **Variable amb "var"**

JavaScript és lo que es coneix com a un llenguatge dinàmic, això vol dir que les variables es creen mentre el programa s'executa i el tipus de dada de la variable és emmagatzemada en el valor i no en la variable com a tal, per lo que se poden crear variables que guarden diferents tipus de dades sense tenir que declara-los (això no passa per exemple en altres llenguatges com Java).

En aquests exemples empram sempre la paraula "var", seguida d'un nom i finalment assignam un valor:

```
var gran = true; // booleà
var edat = 21; // Sencer
var preu = 169.68; // Real
var nom = 'Joan'; // String
```

Es poden declarar moltes variables amb una sola instrucció, simplement separant les variables amb una coma.
Per assignar valors a una variable seria per exemple: x=5; y=6, i per calcular amb variables, seria per exemple: z=x+y;

Codi HTML:

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8"/>
    <title>Emprant variables </title>
  </head>
  <body>
    <h1>Programa amb variables </h1>
    <script>
      //declaram tres variables
      var x = 3; //Assignam valor a x
      var y = 5; // Assignam valor a y
      var z = x*y; // Calculam a partir d'x i y
      document.writeln ("El resultat és: ", z); //És un mètode de l'objecte document (Ho veurem al DOM), que escriu un text a la pàgina.
    </script>
  </body>
</html>
```

```
-------------------
```

Resultat:

Programa amb variables

```
El resultat és: 15
```

Nota: En aquest programa s'han declarat les variables x, y, z. A les dues primeres els hem donat un valor, i la tercera l'hem declarat com a producte de les dues primeres. Finalment es mostra el valor per pantalla. Encara que no és obligatori declarar les variables, sí que és convenient fer-ho.

En aquest exemple veim que podem crear la variable sense assignar-li un valor inicial, i s'assignarà el valor després, basat en alguna condició o depenent del resultat d'alguna funció.

```
var productes;  // se inicialitza a buida
```

També és possible assignar el valor a una variable i després re-escriure-ho:

```
var edat = 21;
edat = 22; // el valor ha estat re-assignat
```

- **Variables amb "let"**

Les diferències entre "var" i "let" són mínimes. Les dues poden guardar qualsevol tipus de dada sense declara quin tipus de dada guardaran, els mateixos codi anteriors, s'utilitzen també amb "let".

```
if (true) {
    let missat = 'Hola';
    alert (misssat); // 'Hola'
}
alert (misssat); // Donarà error, ja que la variable "missat" només afecta al àm
bit de la funció
```

- **Variables amb "const"**

Aquest tipus de variable si requereixen que tenguin un valor al ser declarades, i a més aquests valors no poden ser re-assignats.

- **Diferències entre "var" i "let"**

```
function ambvar()
{
  var x = 20;
  if (true)
  {
    var x = 100;  // mateixa variable!
    alert (x);  // Mostra 100
  }
  alert (x);  // Mostra 100
}
```

```
function amblet()
{
    let x = 20;
    if (true)
   {
        let x = 100;  // variable diferent
        alert (x);  // Mostra 100
   }
   alert (x);  // Mostra 20
}
```

Ampliació: https://cybmeta.com/var-let-y-const-en-javascript

Valors

La sintaxi de JavaScript defineix dos tipus de valors: valors fixos i valors variables:

- **Els valors fixos** s'anomenen literals.

  Els “literals” s'utilitzen per a representar valors en Javascript. Com el seu nom indica són literalment proporcionats pel programador en el codi. Un dels seus significats és “Que reprodueix exactament el que s'ha dit o s'ha escrit “, en el codi s'utilitza per a expressar els valors que es desitja que tinguin les variables, objectes, etc.

- **Els valors variables** són les variables.

Expressions

- Una expressió és una combinació de valors, variables i operadors, que calcula amb un valor.
- El càlcul s'anomena avaluació.
- Els valors poden ser de diversos tipus, com números i cadenes.

1.3. Tipus de dades i operadors

Tipus de dades

Com podem veure a la taula, tenim dos grans tipus de dades: el tipus primitius i els tipus objecte:

		NOM	DESCRIPCIÓ
Tipus de Dades en JavaScript	**Tipus primitius**	String	Cadenes de text
		Number	Valors numèrics
		Boolean	True o False
		Null	Tipus especial, conté null
		Undefined	Tipus especial, conté undefined
	Tipus Objecte	Tipus predefinits JavaScript	Date (dates) RegExp (expresions regulars) Error (dades d'error)
		Tipus definits per l'usuari	Funcions simples Classes
		Array	Matrius. No té mètodes
		Objectes especials	Objecte global
			Objecte prototip
			Altres

- **Tipus primitius:**

  Aquest tipus de dades són immutables, és a dir, no poden ser canviats.

  - **Strings**: És una cadena de caràcters. Se defineixen entre cometes simples o dobles.

    ```
    'Això és un String'
    "Això és un altre String"
    ```

    Si es volen emprar apòstrofs (cometes simples) a la mateixa cadena, hem d'escriure:

    ```
    var text = 'Visc a la finca C \' an Prunera ';
    ```

    Caràcters especials:

    - \' Cometa simple
    - \" Cometa doble
    - \b Retrocés
    - \f Salt de pàgina
    - \n Salt de línia
    - \t Tabulació
    - \\ Barra inclinada

  - **Booleans**: Representa una entitat lògica i pot tenir dos valors: true i false.

    ```
    var casat = true;
    ```

  - **Números**: JavaScript només té un tipus de dades numèric. No hi ha per exemple (com passa en altres llenguatges de programació nombre sencer, coma flotant. etc.).

- **Tipus especial null**: S'utilitza per indicar que alguna cosa existeix però no té valor, el seu valor serà null (quan una variable té el valor null, direm que és del tipus null).
- **Tipus especial undefined**: S'utilitza per indicar que alguna cosa no existeix. Una variable a la qual no se li ha assignat cap valor, direm que té un valor undefined. La diferència entre null i undefined és que mentre undefined ens informa que l'objecte o variable no està definit, null és un objecte buit.

Per aprofundir amb els tipus null, undefined, pot consultar per exemple: https://www.todojs.com/casos-especiales-undefined-null-y-nan/

- **Tipus objecte**:

  Tipus d'informació més complet que una variable senzilla. Els objectes serien les finestres, els formularis i els elements de control dels formularis (botons, caselles de verificació, per exemple).

  Un objecte és una col·lecció de dades relacionades, que generalment té variables i funcions, que dins l'àmbit de l'objecte s'anomenen propietats i mètodes. Es podria dir que un objecte és contenidor de propietats.

  - El nom d'una propietat pot ser una cadena de caràcters, inclús una buida.
  - El valor de la propietat pot ser qualsevol valor que podem utilitzar en JavaScript, excepte undefined.
  - Les propietats dels objectes s'escriuen en forma de parells de valor (nom_objecte : valor), separats per comes.
  - Es pot accedir a les propietats de l'objecte amb una simple anotació de punts.

  Es poden crear nous objectes i utilitzar objectes que ja creats.

  Els objectes JavaScript s'escriuen entre claus ({}, braces en anglès).

  Per crear un objecte a partir d'una classe s'utilitza la instrucció new.

  Els objectes són dinàmics, això vol dir que les seves propietats no tenen per què ser definides en el moment en què es crea l'objecte, podem afegir noves propietats a l'objecte en temps d'execució, (indicant el nom la propietat i assignant-li un valor o funció).

  Tots els objectes pre-definits per JavaScript tenen els seus mètodes. Els mètodes serveixen per fer accions com per exemple, una finestra emergent, modificar les propietats d'un objecte, afegir elements a una llista, etc.

  Crear un mètode :

  ```
  nom_del_mètode : function() { ...
  ```

  ```
  La forma bàsica per crear un mètode:

  var = cotxe{
  portes: “2”,
  rodes: “4”,
  color: “blau”
  };
  ```

```
La forma complexa:

var cotxe = new Object();
cotxe.portes = “2”;
cotxe.rodes = “4”;
cotxe.color = “blau”;
```

Accedir a un mètode:

```
nom_del_mètode.function()
```

```
Exemple:

cotxe.portes;
```

- **Matrius (Arrays):** Les matrius de JavaScript s'escriuen entre claudàtors ([], (en castellà corchetes, en anglès *brackets*). Els elements d'un array es separen per comes. Per crear un array s'utilitza new. Hi ha dues maneres de crear un *array:*
  - Crear un *array* sense dades i sense dimensió:

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

  - Crear un *array* amb dimensió: var meuarray = new Array(6);

```
<script>
   var diaSetmana = new Array(7); // Cream un array amb 7 posicions
   diaSetmana[0] = "Dilluns";
   diaSetmana[1] = "Dimarts";
   diaSetmana[2] = "Dimecres";
   diaSetmana[3] = "Dijous";
   diaSetmana[4] = "Divendres";
   diaSetmana[5] = "Dissabte";
   diaSetmana[6] = "Diumenge";
</script>
```

- **Date:** Per a operar amb dates (sumar, restar, etc.) hem d'obtenir el valor numèric intern de la data. Existeixen dos mètodes per a això: getTime() i valueOf(). Tots dos retornen el mateix valor en mil·lisegons, però el segon mètode reflecteix amb major claredat el que estem obtenint i és una bona opció.

```
var nomdata = new Date();
```

- **Les expressions regulars:**
  - Són patrons utilitzats per trobar una determinada combinació de caràcters dins d'una cadena de text.
  - Són utilitzades per cercar, reemplaçar i extreure informació de les cadenes de caràcters. També són objectes i alguns dels mètodes que funcionen amb expressions regulars en JavaScript són: regexp.exec, regexp.test, string.match, etc.
  - Les expressions regulars tenen un rendiment sensiblement superior a les operacions equivalents sobre *strings*.
  - Una expressió regular pot crear-se de qualsevol de les següents maneres:

- Utilitzant una representació literal de l'expressió regular, consistent en un patró entre barres: var re = /ab+c/;
- Cridant a la funció constructora de l'objecte RegExp: var re = new RegExp('ab+c');

```
Exemple:

var patro = /info/;
var patro = new RegExp(“info”);

--------------
Aquest patró és molt senzill, a un comparació amb una cadena, retornaria "tr
ue" en el cas de que la cadena amb la que se compara sigui "info".
```

Clica per: Ampliar expressions regulars de JavaScript

Operadors

- (+ , -, *, /, %): Per calcular valors.
- Assignació (=): Per assignar valors a variables.
- En el cas de cadenes, per exemple la suma és una concatenació: var x = "Juan" + " " + "Más"; La variable x valdrà: Juan Más.

Operadors de comparació de JavaScript	
Operador	Descripció
==	Igual a
===	Igual valor i Igual tipus
!=	No igual
!==	No igual valor o no igual tipus
>	Més gran
<	Més petit
>=	Més gran o igual
<=	Més petit o igual
?	Operador ternari (per sentències if)

Operadors aritmètics de JavaScript	
Operador	Descripció
+	Suma
-	Resta
*	Multiplicació
**	Exponent
/	Divisió
%	Mòdul (reste de la divisió)
++	Increment
--	Decrement

Operadors de tipus	
Operador	Descripció
typeof	Retorna el valor de la variable
instanceof	Retorna true si si un objecte és una instància d'un tipus d'objecte

Operadors d'assignació de JavaScript		
Operador	Exemple	Notació convencional
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

Ajustar fila de tabla

Operadors de bits de JavaScript					
Operador	Descripció	Exemple	El mateix que	Resultat	Decimal
&	AND	x = y	0101 & 0001	0001	1
|	OR	x += y	0101 | 0001	0101	5
~	NOT	x -= y	~0101	1010	10
^	XOR	x *= y	0101^0001	0100	4
<<	Desplaça a l'esquerra un número especificat de bits.	x /= y	0101<<1	1010	10
>>	Desplaça a la dreta un número especificat de bits.	x %= y	0101>>1	0010	2
>>>	Desplaça a la dreta, descartant i substituint per ceros.	x **= y	5>>>1	0010	2

Operadors lògics	
Operador	Descripció
&&	And
||	Or
!	Not

- **L'operador typeof**: Podem utilitzar l'operador de tipus per trobar el tipus d'una variable. L'operador typeof retorna el tipus d'una variable o d'una expressió:

```
var numero = 2;
alert (typeof numero ) / / Mostra "number"
```

```
var text = "eltext"; alert (typeof eltext) / / Mostra: "string"
```

```
var varbooleana = false; alert (typeof varbooleana) / / Mostra: "boolean"
```

També permet provar l'existència d'una variable:

```
alert (typeof variable) // Mostrarà "undefined"
```

Si la instrucció typeof retorna "undefined", és que la variable és inexistent o està declarada però no té cap valor.

És molt aconsellable demanar el tipus d'una variable, en alguns casos:

```
let variable;
if (typeof variable == 'undefined') {
        alert ('La variable és undefined, no té valor definit);
}
```

1.4. Estructures de control

A Javascript tenim tres tipus de sentències o instruccions:

- Seqüencials
- Selectives
- Iteratives

Sentències Seqüencials

Podem descriure tres sentències seqüencials en Javascript:

- **Assignació:**

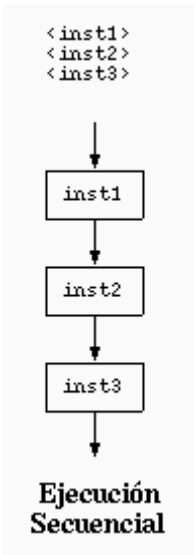
  L'assignació es realitza amb el símbol =, és a dir: "variable = expressió". Aquesta sentència, per si mateixa, no executa cap acció "visible" per a l'usuari de la nostra pàgina web, però és fonamental perquè els nostres programes puguin anar fent a poc a poc el seu treball, realitzant càlculs que es van emmagatzemant en les variables.

- **Escriptura:**

  Per simplificar una mica, la sentència d'escriptura podem considerar que és el "write". En realitat, hem de dir que és "document.write", ja que és un mètode d'un objecte que es diu document (que veurem en el proper capítol). Com hem dit, aquest és un dels mecanismes que podem utilitzar per escriure coses des de JavaScript perquè es puguin veure a través del navegador (normalment, l'altre mecanisme que s'ha imposat com estàndard és el d'utilitzar capes i la seva propietat innerHTML, (com també veurem en estudiar l'objecte document).

- **Lectura:**

  La sentència de lectura permet a l'usuari donar-li dades a el programa, o si es prefereix, permet el programa preguntar dades a l'usuari. En Javascript podem emprar l'objecte window, que té un mètode anomenat prompt que serveix per això.



Sentències Condicionals

- ## La sentència condicional: if (condició) else:

  Permet avaluar una condició i executar un grup d'instruccions si la condició resulta ser vertadera, i un altre grup d'instruccions, si la condició resulta ser falsa:

```
if ( condició ) {
    sentències_true;
}
else {
    sentències_false;
}
```

```
Exemple:

var edat = prompt ( "Introdueix la teva edat", "" ); //prompt permet escriure pe
r pantalla i llegir un valor, ho fa mitjançant una finestra emergent.
if ( edat < 18 ) {
    document.write ( "Ets menor d'edat.<br>" ); //document.write ens permet escr
iure un missatge a la pàgina
    document.write ( "No pots accedir als continguts." );
} else {
    document.write ( "Ets major d'edat.<br>" );
    document.write ( "Clica per veure els continguts." );
}
```

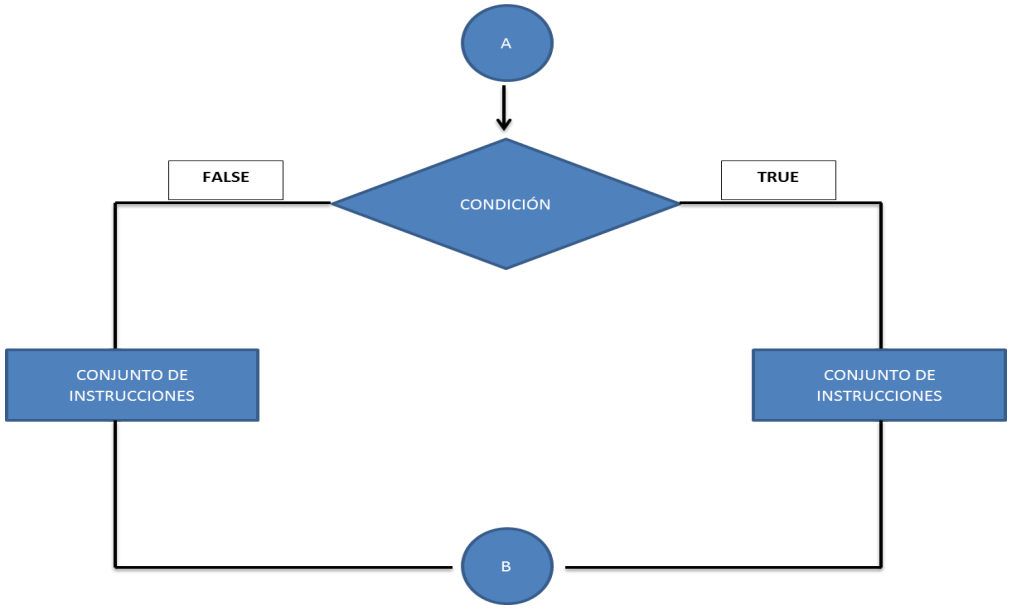
- ## La sentència condicional switch (variable):

  Permet comprovar per una mateixa variable distints valors:

```
switch (cond) {
  case valor : sentències; break;
  case valor : ---
  case valor : sentències; break;
  default: sentències; break;
}
```

Exemple:

```
var institut = prompt( "Indiqui a quin institut li agradaria cursar els seus est
udis: ");
switch( institut ) {
   case "Francesc de Borja Moll" : document.write ("En aquest institut es poden
 cursar tots el cicles de informàtica de forma presencial" ); break;
   case "IEDIB" : document.write ("En aquest institut només es pot cursar el gra
u superior ASIX a distància" ); break;
   case "Politècnic" : document.write ("No oferta cicles d'informàtica" ); brea
k;
   default:  document.write ( "L'institut que indica no existeix o no es reconei
x en aquesta aplicació" ); break;
}
```



Font:intprojammd.blogspot.com

Sentències Iteratives

- ## Sentència iterativa while (condició):

  Permet executar vàries vegades un tros de codi. Es coneix com a bucle.

```
while ( condició ) {
    sentències;
}
// ****************************
do {
    sentències;
} while ( condició );
// ****************************
```

- ## Sentència iterativa for ( ):

  Es repeteix el cicle fins que una condició especificada s'avalua com a falsa. (És similar al for de Java i C).

```
for ( inicialització; condició; increment ) {
    sentències;
}
```

- ## Sentència iterativa do while ():

  Aquest bucle és exactament igual que l'anterior però amb la diferència que la condició es comprova al final. Aquest tipus de bucles són molt utilitzats quan vam crear un menú d'opcions; mentre l'usuari no triï l'opció de sortir de el programa seguirem treballant amb ell. Per això necessitem comprovar l'opció que l'usuari a triat a la fi de el programa, ja que a el principi encara no sabem l'opció que ha escollit.

Exemple: Un programa que escriu la taula de multiplicar del 15, des de 1 fins 50

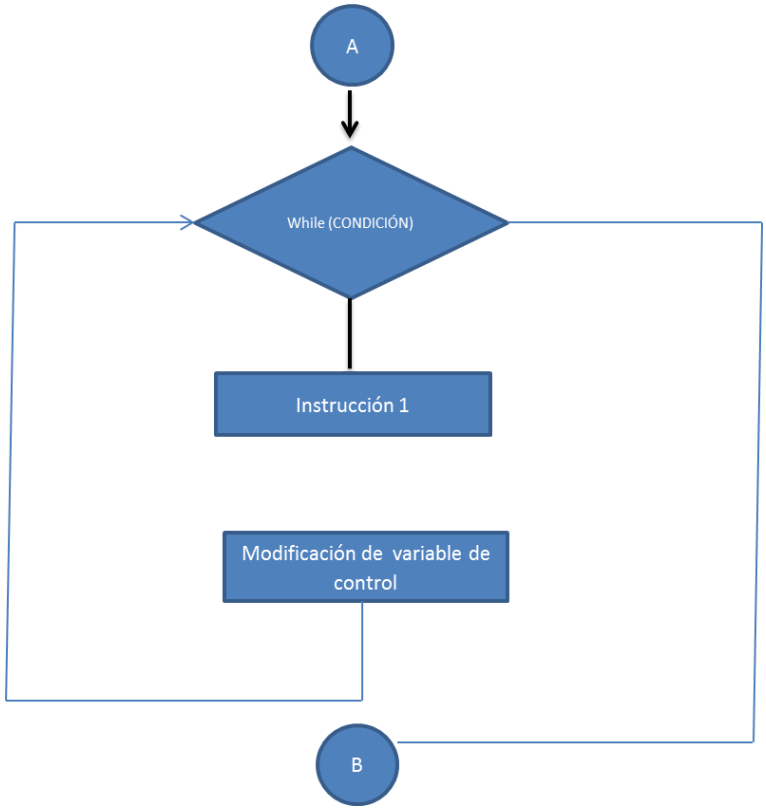
```
document.write ( "<h2>Taula de multiplicar del 15</h2>" );
var cont=1;
while( cont <= 50 ) {
    document.write ( "15 * " + cont + " = " );
    document.write ( 15*cont );
    document.write ( "<br>" );
    cont = cont+1;
}
```

Exemple: El mateix programa d'abans però amb la sentència for

```
for ( cont=1; cont<=50; cont=cont+1 ) {
  document.write ( "15 * " + cont + " = " );
  document.write ( 15*cont );
  document.write ( "<br>" );
}
```

Exemple: El mateix programa però amb do

```
<script>
    var resultat = 1;
    var numero = 5;
    do {
       resultat = resultat * numero;
       numero--;
    }
    while (numero <= 50);
       alert (resultat);
 </script>
```



Font:intprojammd.blogspot.com

Estructura de control d'errors: Try/Catch

Try … Catch correspon a tipus d'estructura per controlar i comprovar el flux d'un programa davant comportaments inesperats. La diferència entre aquesta estructura i l'anterior és que mentre es llança completament l'execució, s'aconsegueix una acció determinada de protecció, davant els possibles errors. Aquest tipus d'excepcions s'estructura mitjançant un bloc de codi que avalua una condició prèvia i proposa en conseqüència una execució predefinida i una altra alternativa davant d'anormalitats.

- ## Excepcions:

  Abans de continuar amb aquesta estructura de control, hem de saber que són les excepcions.

  Les excepcions, són imprevists que tenen lloc durant l'execució d'un programa; anormalitats que impedeixen o alteren el comportament o flux normal d'un programari.

  La seva funció, és separar el codi per al maneig d'errors de la lògica d'aplicació del programa.

  En general, els errors en Javascript són críptics i poc informatius, especialment a Internet Explorer. Això fa interessant el comptar amb operadors que ens permetin llançar els nostres propis missatges advertint l'usuari que alguna cosa ha anat malament.

  - **Excepcions amb Throw:** La forma més senzilla per llançar errors és utilitzant retrocés. Aquesta instrucció permet enviar al navegador un esdeveniment semblant a què es produeix quan ocorre algun imprevist o ens trobem davant d'un tipus inesperat de dades. El llenguatge permet enviar tota classe d'elements, incloent text, nombres, valors booleans o fins i tot objectes. No obstant això, l'opció més usual és enviar l'objecte natiu Error:

    ```
    throw new Error ( "Alguna cosa no ha anat be.");
    ```

Exemple amb control d'excepcions:

```
function suma(arguments) {
  var resultat = 0,
  l = arguments.length; //Funció que calcula la quantitat d'arguments
  if ( l > 10 ) throw alert ( 'Massa arguments!' );
  for ( var x = 0; x < l; x++ ) {
      resultat += arguments[x];
  }
  return resultat;
}
alert (suma( 3, 4, 34, 5, 7, 8, 1, 32 ) );  // Resultat: 94
alert (suma( 3, 4, 34, 5, 7, 8, 1, 32, 3, 5, 8 ) );  // Error: Massa arguments!
// La resta del codi serà ignorat quan es llança l'excepció
```

Es llancen excepcions emprant la instrucció "Throw" i després manejar-les emprant les declaracions Try...Catch.

La declaració "Try...Catch" consta d'un bloc "Try", que conté una o més declaracions, un bloc "Catch", que conté declaracions que especifiquen que fer si es llança una excepció en el bloc "Try", i un bloc "finally", que s'executa després de que s'executin els blocs "Try" i "Catch".

Exemple amb Try/Catch

```
function checkPassword ( elmeuString ) {
  var missatge = {};
  try {
      if (elmeuString.length < 6) throw 'CURT';
      if (elmeuString.length > 10) throw 'LLARG';
      missatge.estat = 'Password validat';
  }
  catch (err) {
    if (err == 'CURT') missatge.estat = 'Password massa curt';
    if (err == 'LLARG') missatge.estat = 'Password massa llarg';
  }
  finally {
    alert ( 'Password Avaluat: ' + missatge.estat );
  }
}
```

Cridam la funció:

```
checkPassword ( '1234' ); // Password Avaluat: Password massa curt
checkPassword ( '12345678901' ); // Password Avaluat: Password massa llarg
checkPassword ( '12345678' ); // Password Avaluat: Password validat
```

Instruccions break i continue

- **break:**

  Server per aturar l'execució d'un bucle i sortir d'ell.

```
for (i=0;i<10;i++) {
        document.write (i);
        escriu = prompt ("Vols continuar?");
        if (escriu == "no");
           break;
}
```

- **continue:**

  Serveix per aturar la iteració actual i tornar al principi del bucle per realitzar una altra iteració si correspon.

```
var i=0
while (i<7) {
        incrementar = prompt ("El valor de "i" és: " + i + ", vols continuar incrementar en un?", "si")
        if (incrementar == "no")
           continue;
        i++;
}
------------
Cada vegada que s'executa el bucle, es demana si es desitja incrementar el valor de variable "i". Si l'usuari diu que no, s'executa la sentència "continue", amb lo que es torna al principi del bucle sense arribar a incrementar la variable "i", ja que s'ignoren les sentències que es trobassin per davall el "continue".
```

1.5. Funcions

IMPORTANT:

- Una funció JavaScript és un bloc de codi dissenyat per realitzar una tasca determinada.
- Una funció JavaScript s'executa quan "alguna cosa" la invoca (la crida).
- Les funcions permeten reutilitzar el codi.
- La definició d'una funció (també anomenada declaració de funció o sentència de funció) consisteix en emprar la paraula clau (reservada: "function".
- Les sentencies JavaScript que defineixen la funció van tancades per clau, { }.

Sintaxi de la funció:

```
function nom (paràmetre1, paràmetre2, paràmetre3, ...) {
  // codi que s'ha d'executar
}
```

- Els paràmetres de funció es mostren dins dels parèntesis () a la definició de la funció.
- Els arguments de funció són els valors que rep la funció quan s'invoca.
- Dins de la funció, els arguments (els paràmetres) es comporten com a variables locals.
- Una funció és el mateix que un procediment o una subrutina, en altres llenguatges de programació.

Exemple:

```
<!DOCTYPE html>
<html>
    <body>
          <h2>les funcions a JavaScript</h2>
          <p>Funció que calcula un producte i retorna el resultat</p>
          <script>
              function producte(p1, p2) {
                    return p1 * p2;
              }
              document.write(producte(3,4))
          </script>
   </body>
</html>
```

Cridar les funcions

El codi dins de la funció s'executarà quan:

- Quan es produeix un esdeveniment (quan un usuari fa clic a un botó).
- Quan s'invoca (es crida) des del codi JavaScript.
- Automàticament (auto-invocat).

Retorn de les funcions

- Quan JavaScript rep una declaració de retorn, la funció deixarà d'executar-se.
- Si s'invoca la funció des d'una declaració, JavaScript "tornarà" per executar el codi després de la sentència d'invocació.
- Les funcions sovint calculen un valor de retorn. El valor de devolució es torna a "retornar" a la instrucció que l'ha cridada.

```
Exemple: Calcular el producte de dos números i retornar el resultat

var x = funcioexemple (4, 3); // El valor de retorn de la funció s'assignarà a x
function funcioexemple (a, b) {
  return a * b; // La funció retorna el producte de a i b
}
document.write(x);
---------------------
Resultat:
x = 12
```

Funcions predefinides de JavaScript

Javascript conté una gran quantitat de funcions en les seves llibreries.

Moltes de les llibreries s'implementen a través d'objectes, per exemple l'objecte Math i l'objecte String. Per exemple, l'objecte Math (la classe), s'utilitza per realitzar càlculs matemàtics. Per utilitzar-los s'opera a través de la classe, en lloc del objecte:

- Per treballar amb la classe Math no s'utilitza la instrucció new, sinó que s'utilitza el nom de la classe per accedir al les seves propietats i mètodes.

Hi ha algunes funcions de JavaScript que no estan associades a cap objecte. A continuació es mostren algunes de les funcions predefinides de JavaScript, però n'hi ha més:

Funcions per a strings:

- **length()**: Calcula el nombre de caràcters d'una variable de tipus text.

```
Exemple:

var cadena = 'Hola';
alert ('La variable té: ' + cadena.length + ' lletres');
-------------------
Resultat:
La variable té: 4 lletres
```

- **concat()**: Realitza la concatenació de dues o més variables de tipus text. Hem de recordar que l'operador + té la mateixa funció.

Exemple:

```
var cadena1 = "Hola";
var cadena2 = cadena1.concat(" com estàs?");
alert (cadena2);
-------------------
Resultat:
Hola com estàs?
```

- **toUpperCase()**: Transforma tots els caràcters d'una variable tipus text a majúscules, independentment de com estiguessin a l'inici.
- **toLowerCase()**: És la funció contrària a l'anterior. Transforma els caràcters d'una variable tipus text a minúscules, independentment de com estiguessin a l'inici.
- **charAt(posició)**: Retorna el caràcter contingut dins de la variable de text que indiqui la posició.

Exemple:

```
var cadena = "Hola";
var lletra = cadena.charAt(0); // lletra = H
lletra = cadena.charAt(2);     // lletra = l
```

- **indexOf(caràcter)**: Retorna un número que correspon a la posició en què es troba el caràcter indicat dins dels parèntesis. Si el caràcter s'inclou diverses vegades, indicarà només la seva primera posició començant des de l'esquerra. Si el caràcter indicat no es troba en cap posició, la funció torna el valor -1.

Exemple:

```
var cadena = "Hola";
var posicio = cadena.indexOf('a'); // posicio = 3
posicio = cadena.indexOf('b');     // posicio = -1
```

- **lastIndexOf(caràcter)**: Cerca la darrera posició on es troba el caràcter. Si la cadena no conté el caràcter, la funció torna el valor -1.

Exemple:

```
var cadena = "Hola, som estudiant de FP Informàtica, a FP de grau superior";
var n = cadena.lastIndexOf("FP");
-------------
Resultat:
n=39
```

- **substring(inici, final)**: Extreu els caràcters d'una cadena, entre els dos indexes especificats.

```
Exemple:

cadena = "Bona nit"
var cadena = cadena.substring(1, 6);
--------------
Resultat:
ona ni
```

- **split(separador)**: Retorna un array que contindrà cadenes de text que seran subcadenes de la variable original, separades pel caràcter separador que hagem indicat.

```
Exemple:

var cadena = "Hola com estàs!";
var paraules = cadena.split(""); // paraules = ["Hola", "com,", "estàs!"];
```

- **replace(string1, string2):** Cerca i reemplaça un string per un altre (treballa amb expressions regulars). Exemple
- **search(text):** Cerca un text dins d'un string, si el troba retorna l'índex, si no retorna -1 (treballa amb expressions regulars).
- **trim():** Elimina els espais inicials i finals d'un string.

- **Funcions per a arrays:**
  - **join(separador)**: És la funció contrària a split (). Retorna una variable de text que és el resultat d'unir tots els elements d'un array. pop (): Retorna l'últim element d'un array. L'array original modifica la longitud en un element.

```
Exemple:

var array1 = ["Bon", "Dia"];
var cadena = array1.join(""); // cadena = "BonDia"
cadena = array1.join(" ");   // cadena = "Bon Dia"
```

  - **push()**: Afegeix un element al final de l'array. L'array original es modifica i augmenta la seva longitud en 1 element. (També és possible afegir més d'un element a la vegada).

```
Exemple:

var array1 = [1, 2, 3, 4];
array1.push(5); // array1 = [1, 2, 3, 4, 5]
```

  - **shift()**: Retorna el primer element de l'array. La longitud de l'array disminueix amb un element.

```
Exemple:

var colors = [“grog”, “verd”, “vermell”];
colors.shift();

------
Retorna: grog
```

- **unshift(valors_a_afegir)**: Afegeix un o més elements al començament d'un array. La seva longitud augmentarà en un element.
- **reverse()**: Retorna l'array amb tots els seus elements col·locats en ordre invers a la seva posició original.
- **pop()**: Elimina l'últim element de l'array i el torna. L'array original es modifica i la seva longitud disminueix en 1 element.
- **slice(index1, index2):** Extreu una secció de l'array.

```
Exemple:

fruites =  ["meló", "mandarina", "poma", "cirera"]
frutes.slice(0,2);

-----
Retorna: ["meló", "poma"]
```

- **splice(index, numero):** Elimina elements de l'array.
- **sort():** Ordena alfabèticament els elements.
- També tenim les funcions **conca**t i l**ength.**

```
Exemple amb length:

var vocals = ["a", "e", "i", "o", "u"];
var numeroVocals = vocals.length; // numeroVocals = 5
```

```
Exemple amb concat:

var array1 = [1, 2, 3];
array2 = array1.concat(4, 5, 6);   // array2 = [1, 2, 3, 4, 5, 6]
array3 = array1.concat([4, 5, 6]); // array3 = [1, 2, 3, 4, 5, 6]
```

- **Funcions numèriques:**
  - **IsNaN()**: Retorna si l'expressió és o no un número. Permet protegir a l'aplicació de possibles valors numèrics no definits.

```
Exemple:

var numero1 = 0;
var numero2 = 0;
if (isNaN(numero1/numero2)) {
  alert ("La divisió donarà error");
}
else {
  alert ("La divisió es igual a => " + numero1/numero2);
}
```

- **toFixed(dígits)**: Retorna el número amb tants decimals com indiquem en el paràmetre dígits.
- **ParseInt i ParseFloat:** Són dues funcions globals que ens permeten convertir cadenes de text en valors numèrics.

```
var nounum = parseFloat(cadena);
var nounum = ParseInt(cadena);
```

- **Funcions de dates:**
  - **getDay(), getDate(), getMonth(), getFullYear()**: Retornen el dia de la setmana, del mes, i l'any, en hora local respectivament.
  - **getHours(), getMinutes(), getSeconds(), getMilliseconds()**: Retornen les hores, minuts, segons i milisegons.
  - **setDay(dia), setDate(dia), setMonth(mes), setFullYear(any)**: Estableixen el dia de la setmana, del mes, i del any.
  - **setHours(hora), setMinutes(minut), setSeconds(segon), setMilliseconds(milisegon)**: Estableixen les hores, minuts, segons i milisegons respectivament.
  - **toString(), toDateString(), toTimeString()**: Retornen la data i hora.
  - **toLocaleString(), toLocaleDateString(), toLocaleTimeString()**: Retornen la data i hora amb format de sistema.

- **Funcions matemàtiques:**
  - **pi**: Nombre PI.
  - **sin(x), cos(x), tan(x)**: Sinus, cosinus i tangent.
  - **asin(x), acos(x), atan(x)**: Arcosinus, arcocosinus i arcotangent.
  - **max([x[,y[,…]]]), min([x[,y[,…]]])**: Màxim i mínim d'una llista de números.
  - **sqrt(x)**: Arrel quadrada.
  - **round(x)**: Arrodoniment a sencer.
  - **random()**: Genera un número aleatori entre [0 i 1].

- **Altres funcions:**

- **alert()**: És un mètode de l'objecte "window" i s'utilitza molt quan se inicia en JavaScript. Mostra per pantalla una petita finestra d'avís. Si volem mostrar el text en vàries línies, escriurem "\n.
- **confirm()**: Permet mostrar un quadre de diàleg per demanar a l'usuari alguna cosa, que com a resultat serà o true o false, i que quedarà guardat dins la variable que s'indiqui.
- **prompt()**: S'utilitza per demanar una dada a l'usuari. També pertany a l'objecte "window".
- **onload**: S'utilitza quan volem que s'executi una funció de JavaScript quan es carrega la pàgina web. S'escriu a l'atribut de l'etiqueta <body> (<body onload="funcioexemple()">). Exemple onload

Podeu ampliar JavaScript, consultant:

Referència JavaScript