

# 1

---

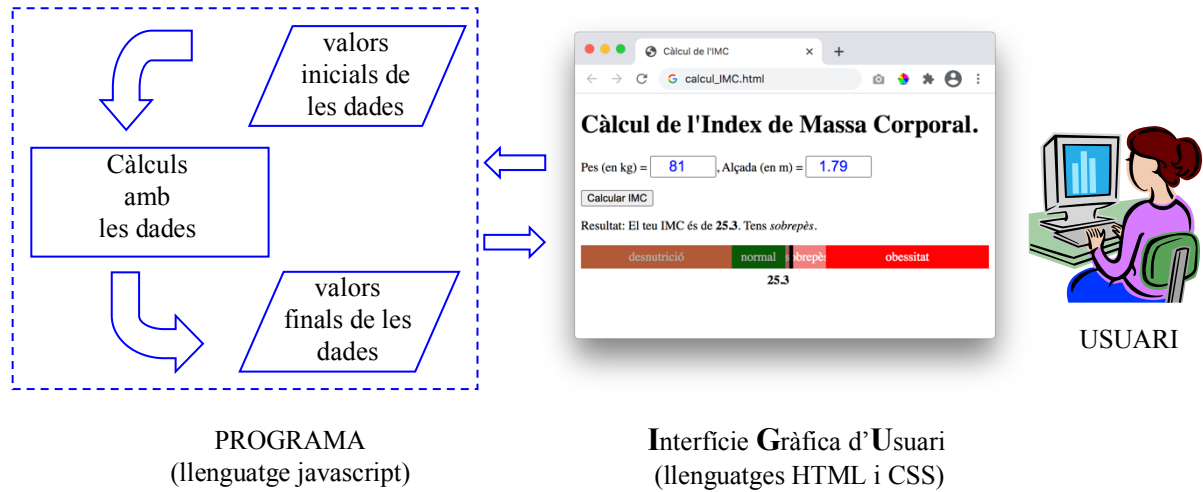
## 1 PROGRAMACIÓ BÀSICA EN JAVASCRIPT

En aquest capítol començarem a estudiar el llenguatge de programació javascript, bàsicament, per dos motius:

- fer petits càlculs,
- augmentar la interacció de l'usuari en les nostres pàgines HTML (*events*) i prendre control total del Model d'Objectes del Document (DOM) que ens permetrà dissenyar jocs i aplicacions multimèdia a la web (*Dynamic HTML*).

## 1.1 Què és un programa?.

Un programa permet a l'usuari fer uns càlculs amb unes dades subministrades. L'estructura general d'un programa és:



L'usuari usa el ratolí i el teclat per interactuar amb els elements d'entrada de la Interfície Gràfica d'Usuari (HTML+CSS): entra els valors inicials de les dades, dona l'ordre de càlcul (events) i, finalment, mira els resultats o valors finals de les dades.

El programa (javascript) agafa els valors inicials entrats i realitza uns càlculs obtenint uns valors finals per a les dades.

Els valors finals (resultats) seran mostrats a l'usuari a través dels elements de sortida de la Interfície Gràfica d'Usuari.

## 1.2 Estructura d'un programa javascript.

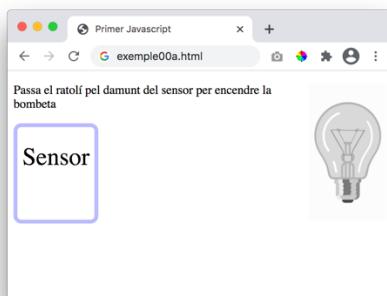
El javascript és un llenguatge de programació, interpretat pel navegador, que permet afegir interactivitat a les pàgines web que no seria possible només amb l'ús dels llenguatges HTML i CSS.

El javascript incorpora el codi de programa a través de la marca HTML `<script>`, per tant, el codi javascript sol estar barrejat amb el propi HTML i CSS. És un llenguatge que usa objectes per emmagatzemar la informació. No hi ha codi compilat (binari) sinó que tot el codi és font (text javascript) i viatja amb el document HTML. Sempre es podrà veure i, per tant, analitzar el codi font.

Per a una guia completa de la sintaxi javascript consulteu

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide>

**Exemple 0.** Un primer programa javascript. Quan passem el ratolí pel damunt del sensor, s'engega la bombeta.



```
<!DOCTYPE html>
<html lang="ca">
<head>
<title>Primer Javascript</title>
<meta charset="utf-8">
<style>
  #sensor{
    width: 100px; height: 100px;
    border: 5px solid #bbf; border-radius: 10px;
    padding-top: 20px;
    font-size: 2em; text-align: center;
    cursor: pointer;
  }
</style>
</head>

<!-- Interfície Gràfica d'Usuari -->
<body>
<div style="float: right;">
  
</div>
<p>Passa el ratolí pel damunt del sensor per encendre la bombeta</p>
<div id="sensor" onMouseOver="encendre();" onMouseOut="apagar();">Sensor</div>

<script>

/* Programa javascript */
function encendre(){
  document.getElementById("bombeta").src="img/on.jpg"
}
function apagar(){
  document.getElementById("bombeta").src="img/off.jpg"
}

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

Prova-ho >>> [http://ssh.eupmt.tecnocampus.cat/~jou/FP/capitol\\_1/exemples/exemple00a.html](http://ssh.eupmt.tecnocampus.cat/~jou/FP/capitol_1/exemples/exemple00a.html)

Observant aquest primer exemple veiem que s'han declarat dues funcions, de nom `encendre` i `apagar`, en una marca `<script>`. La funció `encendre()` és invocada quan succeeix l'esdeveniment, el ratolí entra dins la regió del sensor, `onMouseOver`. La funció `apagar()` s'executa quan el ratolí surt del sensor, `onMouseOut`. Noteu que han aparegut uns atributs nous en la marca `div` del sensor. Són els esdeveniments que escolta el navegador i que provoquen la resposta que desitja el programador, en aquest cas, que s'engegui la bombeta.

### 1.2.1 On es col·loquen els "scripts"?

Els scripts es poden col·locar en la capçalera o en el cos del document. La marca `<script>` és tractada com un element HTML normal, i per tant, es carrega en memòria seguint el flux normal del codi font de la pàgina.

Si posem els scripts en la capçalera, aquests es carreguen en el navegador abans que aquest mostri el contingut del cos. Això assegura que no seran invocats per esdeveniments que produeixi l'usuari en la IGU abans de que s'hagin carregat en memòria, la qual cosa podria comportar un mal funcionament,

```
:  
<script>  
  /*  
    funcions, variables globals i sentències  
  */  
</script>  
</head>  
:
```

Però, cal anar en compte en no posar molt de codi a la capçalera doncs, podria endarrerir la càrrega dels elements de la pàgina. En aquests casos podria ser millor posar els `<script>` just abans de la marca `</body>`, és a dir;

```
:  
<script>  
  /*  
    funcions, variables globals i sentències  
  */  
</script>  
</body>  
</html>
```

També hi ha la possibilitat de col·locar tots aquests scripts en un fitxer independent i vincular-lo al document HTML,

```
<script src="funcions_sentencies.js"></script>
```

on l'atribut `src` és el fitxer extern.

El contingut del fitxer `.js` ha de ser codi javascript. No està permès codi HTML ni CSS en aquest fitxer. Quan hi ha moltes línies de codi és millor aquest segon mètode doncs:

- separem l'HTML i CSS del javascript la qual cosa el fa més llegible i mantenible
- guanyem rapidesa i eficiència quan el mateix codi és compartit per més d'una pàgina web atès que quan s'ha descarregat a la memòria cau de l'ordinador, en visitar una altra pàgina que utilitzi el mateix codi, aquest no es tornarà a descarregar sinó que s'agafarà de la memòria cau.

Barrejat amb l'HTML i dins del seu `<body>` hi tindrem les crides a les funcions anteriors. La crida és la invocació d'execució de la funció. Aquesta invocació, sovint, la desencadena un esdeveniment de l'usuari: ha premut un botó, ha posat el ratolí damunt una paraula (com en l'exemple primer), etc.

El gestor de l'esdeveniment és la funció que s'encarrega de donar resposta a l'esdeveniment que ha provocat l'usuari. Els esdeveniments estan associats a les marques HTML del document. En l'exemple, la marca `<div id="sensor">` pot generar dos esdeveniments: quan el ratolí entra dins la divisió, `onMouseOver`, i quan surt de la divisió, `onMouseOut`,

```
<div id="sensor" onMouseOver="encendre();" onMouseOut="apagar();">Sensor</div>
```

i la resposta a l'esdeveniment és cridar al seu gestor, les funcions `encendre()` i `apagar()`, respectivament.

### 1.2.2 Concepte de funció.

Una funció és com un petit programa que resol un problema molt concret. Per tant, l'estructura d'una funció és la mateixa que la d'un programa qualsevol: donem uns valors inicials, fem un procés o càlculs amb aquests valors i obtenim uns resultats que retornem. La única diferència és que una funció és un programa petit i que ha de funcionar per a qualsevol conjunt de valors de les dades inicials.

**Exemple 1.** Escriviu una funció que calculi la suma de quatre valors donats.

```
function suma4(a,b,c,d){
    var resultat=0;
    resultat=a+b+c+d;
    return resultat;
}
```

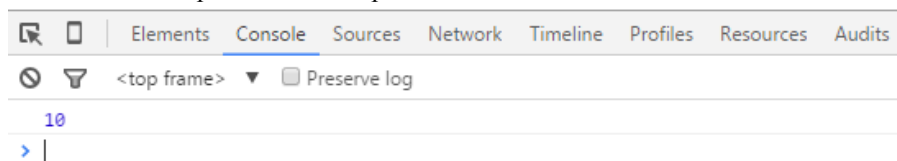
Les dades inicials són a, b, c i d amb valor arbitrari. Se sobreentén que seran valors numèrics els quals es podran sumar.

Dada final o valor final després del procés

Prova-ho >>> [http://ssh.eupmt.tecnocampus.cat/~jou/FP/capitol\\_1/exemples/exemple01.html](http://ssh.eupmt.tecnocampus.cat/~jou/FP/capitol_1/exemples/exemple01.html)

Per tant, tenim el problema de sumar quatre números resol't per a qualsevol combinació d'aquests quatre números. Si volem calcular la suma 1+2+3+4, per exemple, només cal invocar a `suma4(1,2,3,4)` com s'ha fet en l'exemple anterior.

En aquest exemple, el programa mostra el resultat a l'usuari a través de la consola del navegador, usant la funció de javascript `console.log(expressió)`. Per veure la consola del navegador cal fer Ctrl+Maj+I o anar al menú i seleccionar "Eines per al desenvolupador".



Quan escrivim codi font és important anar comentant el codi per a possibles revisions. Els comentaris en javascript són:

```
// tot el que escrivim des de // fins a final de línia és un comentari

/*
    també podem fer comentaris de
    varies línies
*/
```

Els comentaris també es poden fer servir per inhabilitar temporalment una part del codi quan estem en l'etapa de depuració (*debugging*).

En javascript les sentències acaben en ; o final de línia. Per tant, els ; a final de línia són opcionals però és molt recomanable posar-los sempre.

Tots els identificadors, nom de funció o nom de variable, ha de començar: per una lletra majúscula (A..Z) o minúscula (a..z) o el caràcter dolar (\$) o el caràcter subratllat (\_). Després del primer caràcter pot haver també dígit (0..9). No pot haver espais en blanc en l'identificador (ha de ser una única paraula) i eviteu vocals accentuades. Trieu identificadors que aportin significat a la funció o variable.

## 1.3 Entrada i sortida de dades amb elements de formulari.

El llenguatge HTML té previst un conjunt de marques per demanar dades a l'usuari (Entrada). Són els elements de formulari.

### Camps de text i passwords.

Serveixen per introduir una única línia de text, per tant, les tecles "return" i "enter" no són acceptades dins la seqüència de caràcters que conformen el text.

Per exemple, si volem demanar el nom fariem,

Nom (\*)

```
Nom(*)<input id="nom" type="text" size="25" maxlength="35" required  
placeholder="Escriu el teu nom..." autofocus>
```

Observeu que el camp surt de color vermell, indicant que és un camp obligatori i encara no s'ha entrat el valor. També se sol posar (\*) per indicar que és un camp obligatori.

El programa pot llegir el valor entrat per l'usuari (un `String`) i emmagatzemar-lo en la variable `nom` de la següent manera,

```
var nom=document.getElementById("nom").value;
```

nom atribut	descripció
value	valor del camp. El valor escrit en la marca actua com a valor per defecte però, l'usuari el pot editar i canviar. El valor final editat és el que s'entra.
size	especifica l'amplada del camp expressat en el nombre de caràcters visibles (en promig, per la font utilitzada). Si s'entra més caràcters, automàticament s'aplica un scroll per poder veure la part final del text entrat.
maxlength	nombre màxim de caràcters que es podran introduir
disabled	el camp no es pot editar per l'usuari, és a dir, no és una entrada sinó una sortida del programa (no se sol fer servir)
autofocus	el cursor es col·loca en el camp perquè es pugui editar directament. Només pot haver un <code>input</code> amb <code>autofocus</code> .
readonly	el camp no es pot editar per l'usuari, només el programa podrà canviar el seu valor.
required	el camp s'ha d'omplir per part de l'usuari, no es pot deixar buit.
autocomplete	el camp mostrarà una llista amb les entrades anteriors per si s'ha de repetir una mateixa entrada.
placeholder	text que apareixerà escrit dins del camp que serveix per orientar a l'usuari envers quina entrada s'espera. En entrar el primer caràcter, aquest text desapareix.

El password es pot demanar de la següent forma,

password

```
password: <input id="password" type="password" size="12" maxlength="10">
```

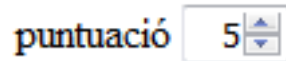
és com un camp de text amb la particularitat de que l'usuari no veu el caràcter que s'ha escrit (queden substituïts per \*).

Podem llegir el valor entrat per l'usuari (un `String`) i emmagatzemar-lo en la variable `password` de la següent manera,


```
var password=document.getElementById("password").value;
```

## Números.

Serveix per entrar un número enter dins un rang de valors predeterminat. Per exemple, volem entrar la puntuació amb notes del 0 al 10:



```
puntuació <input id="puntuacio" type="number" min="0" max="10" step="1" value="5" style="width:35px; text-align:right;">
```

nom atribut	descripció
value	valor del camp. El valor escrit en la marca actua com a valor per defecte però, l'usuari el pot editar i canviar. El valor final editat és el que s'entra.
min	especifica el valor mínim que es pot entrar.
max	especifica el valor màxim que es pot entrar.
step	especifica l'increment o decrement que s'aplicarà al valor actual en interactuar en els botons de pujar i baixar,  . Admet valors decimal com step="0.1", o el valor step="any" en el qual cas, es permetrà qualsevol valor dins del rang establert.

També podem entrar un número dins un rang amb una barra lliçadora ,



```
volum: <input id="volum" type="range" min="0" max="10" step="1" value="5">
```

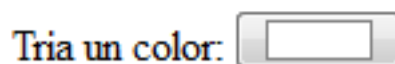
però, l'inconvenient, en aquest cas, és que no es veu el valor actual seleccionat sinó la posició relativa. Podria servir, per exemple, per pujar o baixar un volum acústic, nivell d'il·luminació, etc.

En els dos casos podem llegir el valor entrat per l'usuari (un String) i emmagatzemar-lo en la variable corresponent: puntuacio o volum, respectivament:

```
var puntuacio=document.getElementById("puntuacio").value;
```

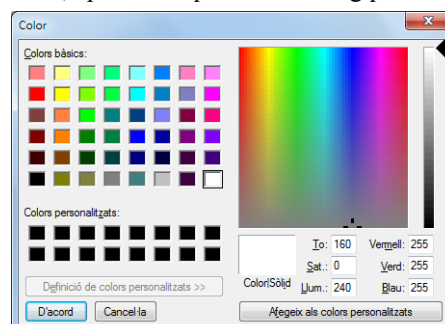
## Colors.

També es pot demanar que l'usuari triï un color qualsevol.



```
Tria un color:<input id="color" type="color" value="#ffffff">
```

quan l'usuari fa clic en el botó mostra, apareix el quadre de diàleg per escollir el nou color,



Podem llegir el valor entrat per l'usuari (un String que representa el valor del color en CSS) i emmagatzemar-lo en la variable color de la següent manera,

```
var color=document.getElementById("color").value;
```

### Dates i hora.

També podem llegir, en format `String`, una data o una hora concreta. Per tal de no entrar una data o hora impossible, amb HTML5 es presenta una finestra perquè l'usuari triï una data o hora sense equivocar-se en el format,

Tria una data: `<input id="data" type="date">`

Quan l'usuari faci clic en l'element, apareixerà una finestra per escollir una data vàlida,

Tria una data:

diciembre de 2020

L	M	X	J	V	S	D
30	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3
4	5	6	7	8	9	10

Hoy

Un cop escollida la data, queda reflectida en l'element,

Tria una data:

De forma similar, podem triar una hora,

Tria una hora: `<input id="hora" type="time">`

Tria una hora:

11	50
12	51
13	52
14	53
15	54
16	55
17	56

Un cop triada l'hora, aquesta queda reflectida en l'element,

Tria una hora:

### Botons.


Els botons s'usen per disparar l'execució d'una funció Javascript determinada.

Executa el programa

`<input type="button" value="Executa el programa" onClick="programa()">`

En HTML5, també s'admet botons amb imatges,

`<button type="button" onClick="dades()">  
   
 Veure les dades entrades  
</button>`

 Veure les dades entrades



### Opcions independents, CheckBox.

Els botons "checkbox" (quadrats) permeten a l'usuari seleccionar un nombre determinat d'opcions independents, és a dir, no mútuament excloents.

```

☒ uses cotxe
☐ t'agrada el cinema

colors favorits (marquem fins a tres)
☐ groc
☐ verd
☐ blau
☐ vermell

<input type="checkbox" id="cotxe" checked=""> uses cotxe <br>
<input type="checkbox" id="cinema"> t'agrada el cinema <br>
<p>colors favorits (marquem fins a tres) <br>
  <input id="groc" type="checkbox"> groc <br>
  <input id="verd" type="checkbox"> verd <br>
  <input id="blau" type="checkbox"> blau <br>
  <input id="vermell" type="checkbox"> vermell <br>
</p>

```

Per saber si l'usuari ha marcat una opció cal comprovar la propietat `checked` de l'element que és un booleà (`true` o `false`),

```

☐ groc      document.getElementById("groc").checked == false
☒ groc  document.getElementById("groc").checked == true

```

### Opcions mútuament excloents, Radio Button.

Els botons "radio" (circulars) es diferencien dels "checkbox" en que només un botó del grup pot estar seleccionat. Es diu que són mútuament excloents. El grup queda definit pels botons que tenen el mateix `name`. Quan l'usuari seleccioni una targeta de crèdit, automàticament es deselecciona l'anterior.

```

Targeta de crèdit:
☒ Visa
☐ MasterCard
☐ American Express

<p>Targeta de crèdit:</p>
<p>
  <input type="radio" name="targ" id="visa" checked="">Visa<br>
  <input type="radio" name="targ" id="mastercard">MasterCard<br>
  <input type="radio" name="targ" id="amex">American Express<br>
</p>

```

Per saber si l'usuari ha marcat una opció cal comprovar la propietat `checked` de l'element tal com hem fet pels CheckBox.

### Àrees de text.

Serveixen per introduir text de més d'una línia, per tant, s'accepten les tecles "return" i "enter" dins la cadena de text.

Per exemple, si volem demanar una opinió fariem,

opinió

```

<textarea id="opinio" cols="50" rows="5"
  placeholder="Escriu la teva opinió aquí.">
</textarea>

```

Observeu que no hi ha l'atribut `value`. En el seu lloc, el text entre les marques d'inici i fi s'utilitza com el contingut inicial de l'àrea de text.

nom atribut	descripció
rows	nombre de línies de text visibles. Si s'entren més línies apareix, de forma automàtica, una barra de "scroll" vertical
cols	especifica l'amplada del camp expressat en el nombre de caràcters visibles (en promig, per la font utilitzada). Si s'entra més caràcters, automàticament es passa a la línia següent.

Podem llegir el valor entrat per l'usuari (un `String`) i emmagatzemar-lo en la variable `opinio` de la següent manera,

```
var opinio=document.getElementById("opinio").value;
```

### Triar una opció.

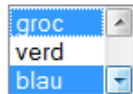
Un element `select` presenta un conjunt d'opcions a l'usuari. Si només es pot seleccionar una i no s'ha especificat una mida visible, les opcions es presenten amb un menú desplegable (drop-down menu). Si es pot seleccionar més d'una opció o s'ha permès més d'una opció visible es presenten les opcions amb una llista.

nom atribut	descripció
size	nombre d'opcions visibles. Si s'usa l'atribut <code>size</code> , el camp <code>select</code> es representa amb una llista. Quan no hi ha l'atribut <code>size</code> ni l'atribut <code>multiple</code> , la representació és amb una persiana (drop-down menu).
multiple	especifica si es pot seleccionar més d'una opció.

### Llistes. Triar varies opcions.

La funcionalitat és similar a la dels botons `CheckBox` però ara, normalment tenim moltes opcions i poc espai per presentar-les a l'usuari. Per això s'obre una petita finestra on presentar aquestes opcions.

colors favorits (seleccionen fins a tres)



```
colors favorits (seleccionen fins a tres) <br>
<select id="color" size="3" multiple>
  <option id="groc" selected>groc</option>
  <option id="verd">verd</option>
  <option id="blau" selected>blau</option>
  <option id="vermell">vermell</option>
</select>
```

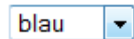
Si des de javascript volem saber si l'usuari ha escollit una opció cal preguntar per la propietat `selected` que és de tipus booleà (`true` o `false`)

```
document.getElementById("groc").selected==true;
```

### Menú desplegable. Escollir una única opció.

La funcionalitat és similar a la dels botons `Radio` però ara, normalment tenim moltes opcions i poc espai per presentar-les a l'usuari. Per això es selecciona a través d'una persiana desplegable.

colors favorits (seleccionen un)



```
color favorit (seleccionen un)<br>
<select id="color">
  <option id="groc">groc</option>
  <option id="verd">verd</option>
  <option id="blau" selected>blau</option>
  <option id="vermell">vermell</option>
</select>
```

## Etiquetes i agrupacions.

Les etiquetes ens permetran associar un text al camp corresponent. Aquest text, per tant, serà el contingut d'un element html que podrem dissenyar amb CSS. Fer clic a l'etiqueta és equivalent a fer-ho en l'element HTML associat. Així, serà especialment útil en `<input>` de mida petita com els `checkbox` i els `radio`. Per exemple:

```
<p>
<input type="radio" name="targ" id="visa" checked="checked">
<label for="visa">Visa</label><br>
<input type="radio" name="targ" id="mastercard">
<label for="mastercard">MasterCard</label><br>
<input type="radio" name="targ" id="amex"><label for="amex">American Express</label>
</p>
```

També podem fer agrupacions de camps i posar una etiqueta al grup. Per exemple, si volem el següent formulari,

caldria el següent html,

```
<fieldset>
  <legend>Dades personals</legend>
  <label for="nom"> Nom </label> <br>
  <input type="text" id="nom"> <br>
  <label for="cognoms"> Cognoms </label> <br>
  <input type="text" id="cognoms"> <br>
  <label for="dni">DNI </label> <br>
  <input type="text" id="dni" size="10" maxlength="9">
</fieldset>

<fieldset>
  <legend>Dades de connexió</legend>
  <label for="nom_usuari"> Nom de l'usuari</label> <br>
  <input type="text" id="nom_usuari" maxlength="10"> <br>
  <label for="clau_de_pas"> Clau de pas</label> <br>
  <input type="password" id="clau_de_pas" maxlength="10"> <br>
  <label for="clau_de_pas2"> Repeteix la clua de pas</label> <br>
  <input type="password" id="clau_de_pas2" maxlength="10">
</fieldset>
```

Tots els elements d'un formulari són d'entrada de dades per part de l'usuari. Per escriure el resultat del programa, que tenim emmagatzemat, per exemple, en la variable `el resultat`, hem d'agafar un element HTML de la Interfície Gràfica d'usuari que permeti mostrar el seu valor en format text, per exemple,

```
<span id="solucio"></span>, utilitzant la següent instrucció:
document.getElementById("solucio").innerHTML=resultat;
```

Prova-ho >>> [http://ssh.eupmt.tecnocampus.cat/~jou/FP/capitol\\_1/exemples/exempleEntradaFormulari.html](http://ssh.eupmt.tecnocampus.cat/~jou/FP/capitol_1/exemples/exempleEntradaFormulari.html)

### 1.4 Tipus de dades: nombres, Strings i booleans.

En javascript el tipus de la dada (variable o constant) és implícit al valor que emmagatzema la dada o al valor representat en les constants literals. Per tant, a diferència d'altres llenguatges de programació, no hi ha cap paraula reservada pels tipus de les dades i el tipus de la dada variable pot canviar al llarg de l'execució.

Hi ha 5 tipus de valors primitius (indivisibles) que reconeix el javascript:

- nombres (**Number**), no diferenciant entre enters i reals: 3, -1.5, 5e2, .5E-1
- booleans (**Boolean**), amb els valors **true** o **false**
- cadenes de caràcters (**String**), inclou els caràcters individuals, i els valors s'escriuen entre cometes dobles "això és un string" o entre cometes simples 'això és un altre string'.
- **null**, objecte no creat
- **undefined**, valor que té una variable declarada però no inicialitzada.

A més, el llenguatge té els objectes (els veurem en el següent capítol) i les funcions. Una dada primitiva es pot convertir en un objecte: **Number**, **Boolean** o **String**.

#### 1.4.1 Variables: declaració, accés i modificació.

Els valors de les dades o bé s'emmagatzemen en variables per poder-les usar posteriorment, o bé són constants literals (s'escriu el seu valor "literalment" en el codi font). Podeu pensar que una variable és un calaix al que li heu posat una etiqueta (identificador), i on hi emmagatzemeu un valor.

Totes les variables que usa el programa han d'estar declarades (creades). Per declarar una variable cal utilitzar la paraula reservada **var**.

Per exemple,

```
var edat;           // declaració. En aquest moment la variable edat té el valor undefined
                    edat undefined

edat=34;           // modificació. Ara la variable edat s'ha modificat (inicialitzat) al valor 34
                    edat 34

var nom="Hola";    // es pot declarar i inicialitzar en la mateixa instrucció.
                    nom "Hola"

var x, y, z=5;     // es pot declarar i/o inicialitzar una llista de variables separades
                    // per comes en la mateixa instrucció
                    x undefined      y undefined      z 5
```

**Modificació:** el valor que conté una variable és susceptible de ser canviat en qualsevol moment (d'aquí el nom de variable). Per assignar un valor a una variable utilitzem la instrucció d'assignació:

`nomVariable = expressió;`

per exemple, `x=35`, i és diu que "la variable x pren per valor el número 35"

Observeu que el nom de la variable va a l'esquerra del igual i a la dreta hi posarem una expressió (vegeu apartat 1.5) que pot ser tan senzilla com un valor literal a una altra variable. Per exemple,

```
var x, y=35;       // declarem les variables x, y i inicialitzem la y al valor numèric 35
                    x undefined      y 35

x=2;               // inicialitzem la variable x al valor numèric 2
                    x 2              y 35

x=x+y              // modifiquem el valor de la variable x amb el resultat de l'expressió x+y.
                    // L'expressió s'executa amb els valors de les variables (accés)
                    x 37              y 35
```

### 1.4.2 Constants **ES6** i valors literals: declaració i accés.

Es poden declarar (crear) dades que no canviaran al llarg de l'execució del programa, és a dir, seran constants. La constant pot ser anònima (sense identificador) o pot tenir un nom. En el primer cas, sempre apareixen dins una expressió i s'anomenen valors literals. En el segon cas, es declara un identificador per a la constant, normalment en majúscules, i s'inicialitza de la següent manera:

```
const PREFIX=93; // 93 és el valor literal de la constant. És una característica ES6
```

Els valors literals són seqüències de caràcters que representen, literalment, el valor de la dada dins d'una expressió.

nombres	els <b>enters</b> es poden representar en decimal o hexadecimal. Els valors en decimal no poden començar per 0 i els valors hexadecimals comencen per 0x. Per exemple, el valor <i>dotze</i> es pot escriure: 12 en decimal o 0xC en hexadecimal.  els <b>reals</b> es poden representar en coma flotant o notació exponencial. Per exemple: el valor <i>menys catorze coma cinc</i> és pot escriure: -14.5 en coma flotant o -0.145E2 en notació exponencial.
booleans	<b>false true</b>
String	"qualsevol seqüència de caràcters" o 'qualsevol seqüència de caràcters' La seqüència de caràcters pot contenir caràcters especials: \t (tabulador), \n (nova línia), \' (apòstrof), \" (cometes dobles), \\ (barra inclinada invertida), ...  <b>ES6</b> També es poden fer strings parametritzats, x=`El resultat es \${a+b}`; on <i>a+b</i> és una expressió

### 1.4.3 Àmbit de les dades: local i global.

L'àmbit (*scope*) és el lloc del programa on es pot fer servir una variable (accedir o modificar el seu valor).

L'àmbit d'accés de les variables pot ser:

- **global**, quan es declara la variable fora de qualsevol funció. Serà accessible en tot el document HTML.
- **local**, quan es declara dins d'una funció. Només serà accessible dins de la definició de la funció independentment del lloc concret on s'ha declarat. Per exemple, aquestes dues definicions de funció són equivalents:

```
function f() {
  var x=5, y;
  y=x+z;
  var z=6;
  return y;
}

function f() {
  var x=5, z=6, y;
  y=x+z;
  return y;
}
```

Per facilitar la lectura del codi és millor el segon cas, és a dir, declarar totes les variables que farem servir abans de fer els càlculs amb els seus valors.

- **block**, **ES6** quan es declara dins un bloc {...} amb la paraula reservada **let**.

Si tenim una variable local amb el mateix nom que una variable global, la local "tapa" a la global. És a dir, la global no es podrà fer servir dins del cos de la funció.

Per exemple,

```
var x=3 // variable global perquè està declarada fora de qualsevol funció
function f(){
  var x=0 // variable local perquè està declarada dins de la funció
  return x; // aquesta x és la variable local. Aquesta funció sempre retorna 0,
            // independentment del valor de la variable global x
}
```

Cal triar bé doncs, els noms de les variables locals i globals per no confondre-les. De fet, és una bona pràctica no usar variables globals. Però, hi ha situacions en que l'ús de variables globals, pot millorar el rendiment del programa (execució més ràpida).

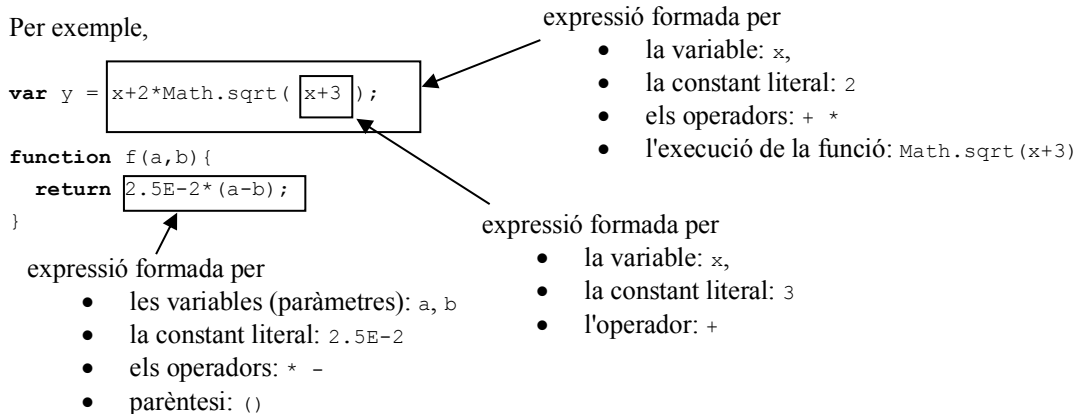
## 1.5 Expressions i operadors.

Una expressió és un conjunt de: variables, constants, valors literals, operadors, execució (crida) de funcions i parèntesi ( ) que avalua un únic valor. Aquest és el valor de l'expressió.

Les expressions apareixen:

- en una instrucció d'assignació (és el cas més habitual): `nomVariable=expressió;`
- com a argument en l'execució d'una funció: `nomFuncio(expressió);`
- en una sentència `return` d'una funció: `return expressió;`

Per exemple,



Cal conèixer quins són els operadors que poden aparèixer en una expressió i quines són les regles per interpretar correctament l'ordre de les operacions involucrades en l'expressió.

### 1.5.1 Operadors aritmètics.

Els operands i el resultat són números.

operadors aritmètics	nom de l'operador	tipus dels operands	tipus del resultat	Exemples on: <code>var x=3, y=5.5, z=10, r;</code>
++	increment	enter	enter	<code>x++; // x val 4, és equivalent a <code>x=x+1</code>;</code>
--	decrement			<code>x--; // x val 2, és equivalent a <code>x=x-1</code>;</code>
+	suma	enter o real	real	<code>r = x+y; // r val 8.5</code>
-	resta o canvi de signe			<code>r = x-y; // r val -2.5</code> <code>r = -x; // r val -3</code>
*	multiplicació			<code>r = x*y; // r val 16.5</code>
/	divisió			<code>r = x/y; // r val 0.545454</code>
%	residu de la divisió entera (mòdul)	enter	enter	<code>r = z%x; // r val 1 doncs <code>10=3*3+1</code></code>

### 1.5.2 Operadors lògics.

Els operands i el resultat són valors booleans.

operadors booleans	nom de l'operador	tipus dels operands	tipus del resultat	Exemples on: <code>var x=true, y=false, r;</code>
&&	I (AND)	booleà	booleà	<code>r = x&amp;&amp;y; // r és false</code>
	O (OR)			<code>r = x  y; // r és true</code>
!	NO (NOT)			<code>r = !x; // r és false</code>

Totes les combinacions possibles són:

<b>x&amp;&amp;y</b>	x=true	x=false
y=true	true	false
y=false	false	false

És a dir, `x&&y` és **true** sí i només sí els dos valors `x` i `y` són **true**

<b>x  y</b>	x=true	x=false
y=true	true	true
y=false	true	false

És a dir, `x||y` és **true** si `x` o `y` són **true**, qualsevol dels dos o ambdós.

	<b>!x</b>
x=true	false
x=false	true

Propietats de De Morgan:

`!(x && y) == !x || !y`

`!(x || y) == !x && !y`

Les expressions booleanes s'avaluen en curtcircuit, és a dir,

**false** && *qualsevolExpressió* és sempre **false** i no s'avalua *qualsevolExpressió*

**true** || *qualsevolExpressió* és sempre **true** i no s'avalua *qualsevolExpressió*

En una expressió booleana, els valors `null`, `0`, `NaN`, `""`, `undefined` es converteixen a **false**

### 1.5.3 Operadors relacionals.

Els operadors relacionals serveixen per comparar valors. El resultat sempre serà un booleà: **true** o **false**.

Els operands acostumen a ser valors numèrics o cadenes de caràcters (*String*). Si són cadenes de caràcters la comparació és lexicogràfica (alfabètica)

operadors relacionals	nom de l'operador	tipus dels operands	tipus del resultat	Exemples de valors true on: var x=3, y=4;
==	igual	qualsevol	Booleà	3==x "3"==x // "3" es converteix abans a número 3=='3' // '3' es converteix abans a número
!=	diferent			x!=4 x!="4" // "4" es converteix abans a número x!=y
>	més gran que	Numèric o String		y>x "12">x // "12" es converteix abans a número "12"<"6" // comparació lexicogràfica "Tara">"Tapa" // comparació lexicogràfica
>=	més gran o igual que			y>=x "4">=y // "4" es converteix abans a número
<	més petit			x<y
<=	més petit o igual que			x<=y y<="4" // "4" es converteix abans a número
===	igual en valor i tipus	qualsevol		3===x // mateix tipus i mateix valor
!==	diferent en valor o tipus			"3"!==x //mateix valor però diferent tipus

En molts casos, si els dos operands no són del mateix tipus, javascript intenta convertir-los a un valor apropiat per a la comparació, normalment a un valor numèric. Amb els operadors `===` i `!==` no es fa cap conversió, per tant, si els operands són de diferent tipus ja no poden ser iguals.

### 1.5.4 Operador concatenació.

Hi ha un únic operador binari per a operands de tipus `string`, és l'operador concatenació `+`. Aquest operador serveix per ajuntar (concatenar) dues cadenes de caràcters en una de sola, per exemple,

```
var missatge="Hola. "+"Això és un missatge."; // missatge == "Hola. Això és un missatge."
```

Aquest operador el farem servir molt al llarg del curs perquè nosaltres treballarem molt amb cadenes de caràcters.

L'operador `+` unari aplicat a una dada cadena de caràcters, la converteix a un nombre.

### 1.5.5 Conversió de tipus.

En l'avaluació d'expressions es poden produir canvis implícits de tipus. Com hem vist, l'operador `+` pot ser suma de nombres o concatenació de cadenes. Si un operand és un nombre i l'altre una cadena, es convertirà el nombre a una cadena i es farà una concatenació de cadenes. En el cas de que l'operador sigui `-`, la resta, si afecta a un nombre i una cadena, es convertirà la cadena a un nombre perquè no es poden restar cadenes.

Per exemple:

```
var s="7"; // cadena
var n=42; // nombre
var r=n-s; // resta, r és un número, r==35
r=s-n; // resta, segueix sent un número, r==35
r=n+s; // concatenació, ara és una cadena, r=="427"
r=s+n; // concatenació, segueix sent una cadena, r=="742"
n=n+""; // concatenació, ara n és una cadena, n=="42"
```

Per convertir una cadena a un número, **explícitament**, cal usar les funcions `parseInt()` i `parseFloat()`.

Per exemple:

```
var r=parseInt("345"); // r és número i val 345
r=parseInt("FF",16); // ara r val 255, el segon argument de la funció és la base de
// numeració. Si la base és 10 no cal posar-lo.
r=parseInt("Hola"); // r val NaN perquè "Hola" no pot representar cap valor numèric
r=parseFloat("3.14"); // r val 3.14
r+="3.14"; // equivalent a l'anterior, r val 3.14
```

### 1.5.6 Avaluació d'expressions. Ordre de les operacions.

La precedència dels operadors determina l'ordre en que s'apliquen quan avaluem una expressió que conté molts operadors. Els parèntesis serveixen per indicar quin és l'ordre desitjat, de manera que els parèntesis més profunds són els que s'avaluen primer. Es recomana posar parèntesis quan no quedi clar l'ordre de les operacions.

En la següent taula resumeix l'ordre de precedència dels operadors, de major a menor precedència:

Tipus d'operador		Operadors
parèntesi		() més profund
unaris	negació, canvi de signe, de String a Number, increment, decrement	! - + ++ --
aritmètics	multiplicar, dividir, mòdul	* / %
	sumar, restar	+ -
relacionals	menor que, menor o igual que, major que, major o igual que	< <= > >=
	igual a, diferent de, estrictament igual a, estrictament diferent de	== != === !==
lògics	i lògica	&&
	o lògica	
assignació		= += -=

Els operadors de la mateixa precedència s'avaluen d'esquerra a dreta.



### 1.5.7 Objecte Math.

És un objecte predefinit en javascript que agrupa les constants i funcions matemàtiques més important.

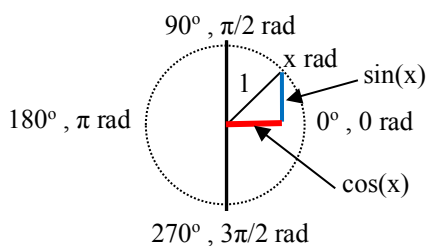
*Constants matemàtiques.*

	nom	valor aproximat
<b>Math.E</b>	nombre <i>Euler</i> , $e$	$e = 2.718$
<b>Math.LN10</b>	logaritme neperià de 10	$\ln(10) = 2.302$
<b>Math.LN2</b>	logaritme neperià de 2	$\ln(2) = 0.693$
<b>Math.PI</b>	nombre Pi, $\pi$	$\pi = 3.14159$

*funcions matemàtiques.*

<b>Math.abs</b> (nombre)	valor absolut, $ \text{nombre} $
<b>Math.sin</b> (nombre_Radians)	sinus
<b>Math.cos</b> (nombre_Radians)	cosinus
<b>Math.tan</b> (nombre_Radians)	tangent
<b>Math.exp</b> (nombre)	exponencial, $e^{\text{nombre}}$
<b>Math.log</b> (nombre)	logaritme neperià (base $e$ ), $\ln(\text{nombre})$
<b>Math.ceil</b> (nombre)	Retorna l'enter més petit major o igual a nombre
<b>Math.floor</b> (nombre)	Retorna l'enter més gran menor o igual a nombre
<b>Math.round</b> (nombre)	Retorna l'enter més a prop a nombre
<b>Math.min</b> (nombre,...)	Retorna el nombre més petit dels subministrats
<b>Math.max</b> (nombre,...)	Retorna el nombre més gran dels subministrats
<b>Math.pow</b> (base,exp)	Retorna $\text{base}^{\text{exp}}$
<b>Math.sqrt</b> (nombre)	Retorna l'arrel quadrada, $\sqrt{\text{nombre}}$

Recordeu:



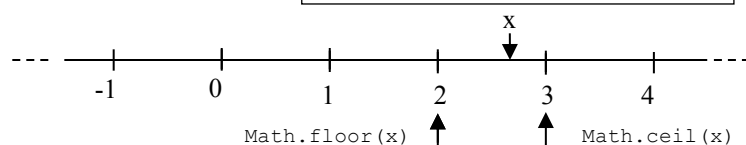
$$2\pi \text{ radians} = 360^\circ$$

$$\log_a(b) = \log_c(b) / \log_c(a)$$

per tant,

$$\log_{10}(x) = \text{Math.log}(x) / \text{Math.LN10}$$

$$\log_2(x) = \text{Math.log}(x) / \text{Math.LN2}$$



## 1.6 Sentències de control de l'execució.

Un programa és una seqüència de sentències. Fins ara hem vist només les sentències de declaració, inicialització d'una variable o constant i assignació d'una variable. Recordem la sintaxi d'aquesta última,

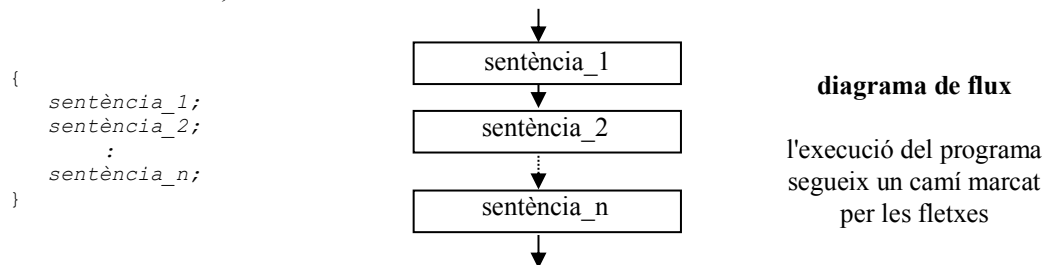
```
nomVariable = expressió; // nomVariable pren per valor el resultat de l'expressió
```

Ara és l'hora de veure altres sentències que permetran controlar el flux de l'execució del programa, és a dir, quines sentències s'executen, quantes s'executen i quines no s'executen.

Totes les sentències acaben en ; o final de línia. És bo però, acostumar-se a posar el ; al final de totes les sentències.

### 1.6.1 Estructura seqüencial. Sentència bloc.

L'estructura més senzilla és la seqüencial, és a dir, s'executen totes les sentències, una darrera l'altre. La sentència bloc és una seqüència de sentències i queda definit per les claus, { }. És l'única sentència que no necessita acabar en ;



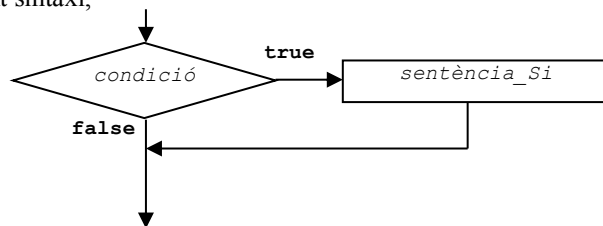
### 1.6.2 Estructura condicional: if..., if... else..., if... else if...

L'estructura condicional permet triar quines sentències s'executen segon el valor d'una condició. Una condició és una expressió booleana, és a dir, una expressió que quan és avaluada, el seu valor és un booleà.

Hi ha diferents estructures condicionals. Veurem les més importants.

La primera és la sentència **if...**, amb la següent sintaxi,

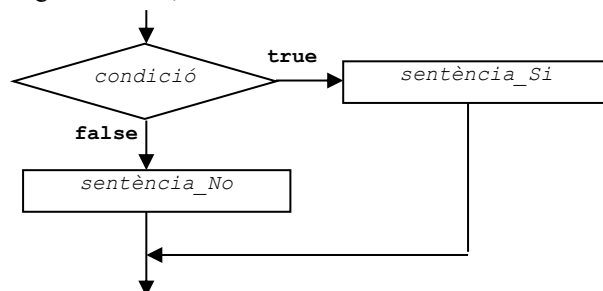
```
if(condició){
  sentència_Si;
}
```



Observeu que la *sentència\_Si* només s'executarà si la condició és certa (**true**).

Una altra sentència és la **if... else...** amb la següent sintaxi,

```
if(condició){
  sentència_Si;
}
else{
  sentència_No;
}
```



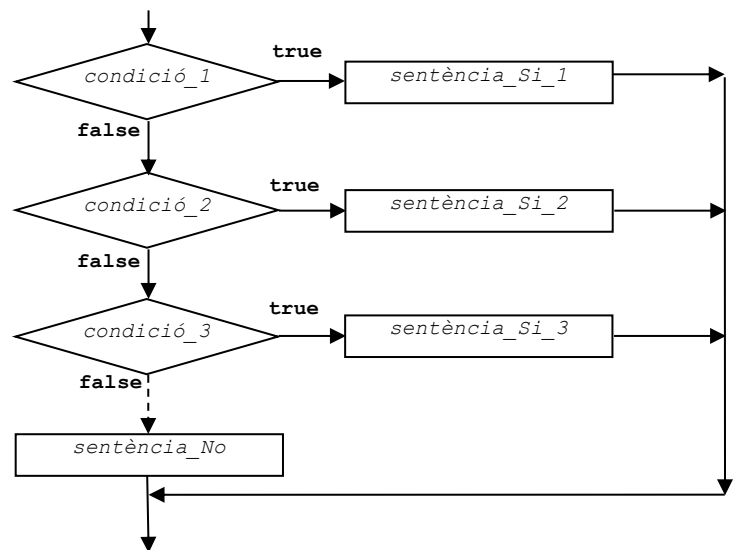
En aquest cas, si la *condició* és certa (**true**) s'executa la *sentència\_Si*, sinó s'executa la *sentència\_No*

**Exemple 2.** Escriu una funció que retorni el més gran de dos números donats .

```
function mesGran(a, b){ // a, b són els números donats (dades d'entrada)
  if(a>b){
    return a;
  }
  else{
    return b;
  }
}
```

Una altra estructura és la `if... else if...`, en que sempre és decideix amb una condició quina és la sentència que s'executa,

```
if(condició_1){
  sentència_Si_1;
}
else if(condició_2){
  sentència_Si_2;
}
else if(condició_3){
  sentència_Si_3;
}
:
else{
  sentència_No;
}
```

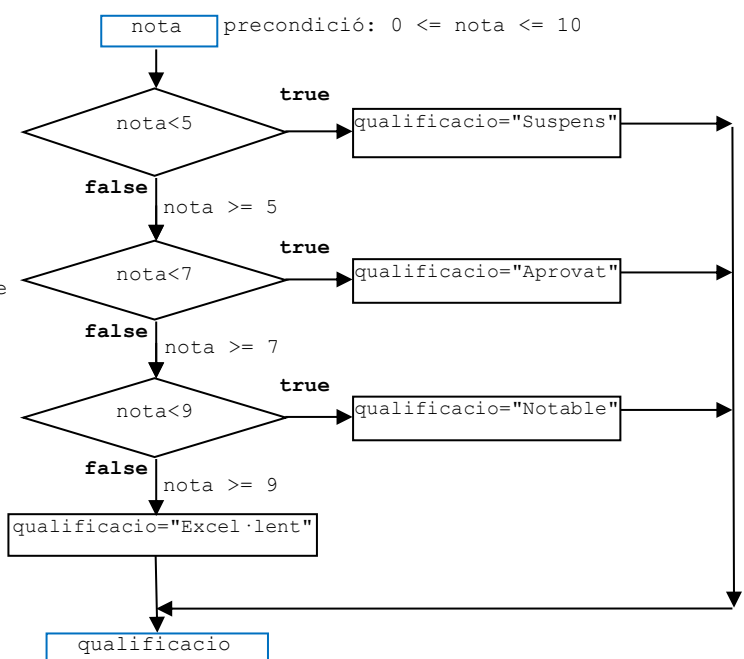


Per exemple, si volem qualificar una nota segons el barem següent:

Suspens,       $0 \leq \text{nota} < 5.0$   
 Aprovat,       $5.0 \leq \text{nota} < 7.0$   
 Notable,       $7.0 \leq \text{nota} < 9.0$   
 Excel·lent,  $9.0 \leq \text{nota} \leq 10.0$

faríem,

```
function calcularQualificacio(nota){
  // el valor de nota ha de complir que
  // 0 <= nota <=10
  var qualificacio="";
  if(nota<5)
    qualificacio="Suspens";
  else if(nota<7)
    qualificacio="Aprovat";
  else if(nota<9)
    qualificacio="Notable";
  else
    qualificacio="Excel·lent";
  return qualificacio;
}
```



**Exemple 3.** Escriu un programa que calculi l'Índex de Massa Corporal,  $IMC = \text{Pes (kg)} / \text{alçada}^2 \text{ (m)}$ . Cal que el programa indiqui a més si hi ha desnutrició ( $IMC < 18.5$ ), normalitat ( $18.5 \leq IMC < 25$ ), sobrepès ( $25 \leq IMC < 30$ ) o obesitat ( $IMC \geq 30$ ). La IGU que es proposa és la següent,

## Càlcul de l'Index de Massa Corporal.

Pes (en kg) =  , Alçada (en m) =

Resultat: El teu IMC és de **27.2**. Tens *sobrepès*.

```
<!DOCTYPE HTML>
<html lang="ca">
<head>
<meta charset="utf-8">
<title>Càlcul de l'IMC</title>
</head>
<body>
<h1>Càlcul de l'Index de Massa Corporal.</h1>
<p>
  <label for="pes"> Pes (en kg) = </label><input type="text" id="pes" size="5">,
  <label for="alçada">Alçada (en m) = </label><input type="text" id="alçada" size="5">
</p>
<p>
  <input type="button" value="Calcular IMC" onClick="programa()">
</p>
<p id="resultat"></p>
<script>
function programa() {
  // Entrada de dades per part de l'usuari
  var pes=parseFloat(document.getElementById("pes").value);
  var alçada=parseFloat(document.getElementById("alçada").value);
  // Càlcul del resultat: IMC i missatge: desnutrició, pes normal, sobrepès o obesitat.
  var resultat=calcul_IMC(pes, alçada);
  // Sortida dels resultat cap a l'usuari
  document.getElementById("resultat").innerHTML=resultat;
}
function calcul_IMC(pes,alçada){
  var imc, missatge;
  imc=pes/(alçada*alçada);
  imc=Math.round(imc*10)/10;          // per representar el valor només amb un decimal
  // imc=Number(imc).toFixed(1);      // també es pot fer amb un objecte Number
  // imc=(pes/(alçada*alçada)).toFixed(1); // o directament en l'expressió
  if(imc<18.5)
    missatge="desnutrició";
  else if(imc<25)
    missatge="un pès normal";
  else if(imc<30)
    missatge="sobrepès";
  else
    missatge="obesitat";
  return "Resultat: El teu IMC és de <strong>"+imc+
    "</strong>. Tens <em>"+missatge+"</em>.";
}
</script>
</body>
</html>
```

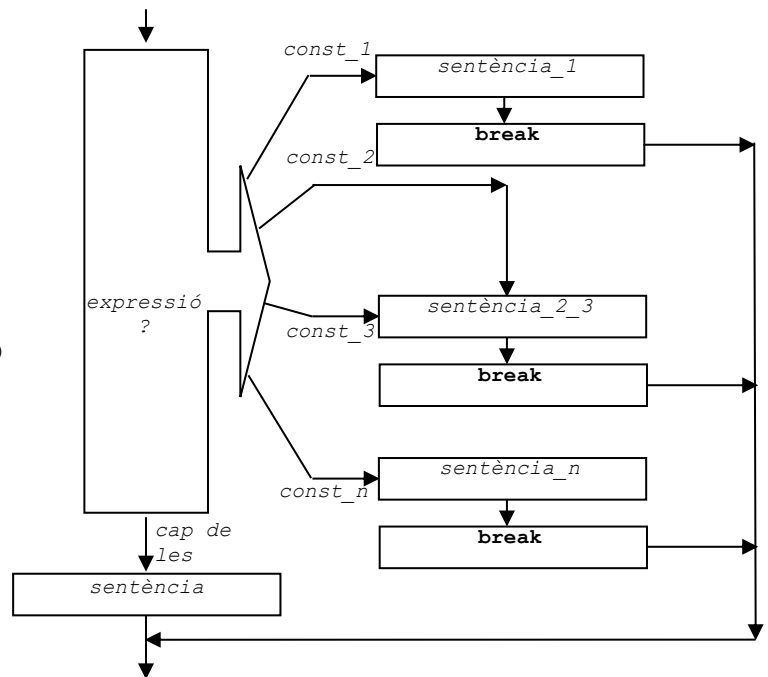
Prova-ho >>> [http://ssh.eupmt.tecnocampus.cat/~jou/FP/capitol\\_1/exercicis/exercici07.html](http://ssh.eupmt.tecnocampus.cat/~jou/FP/capitol_1/exercicis/exercici07.html)

Una variant d'aquesta darrera estructura és quan avaluem una expressió entera, i decidim executar una o altra sentència en funció de quin valor avalua l'expressió. Seria el cas, per exemple, d'un menú d'opcions.

```
switch(expressió){
  case const_1: sentència_1; break;
  case const_2:
  case const_3: sentència_2_3; break;
  :
  case const_n: sentència_n; break;
  default: sentència;
}
```

equivalent a

```
if(expressió==const_1)
  sentència_1;
else if(expressió_entera==const_2 ||
        expressió_entera==const_3 )
  sentència_2_3;
:
else if(expressió_entera==const_n)
  sentència_n;
else
  sentència;
```



#### Exemple 4. Moure un rectangle al prémer una tecla fletxa.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>Exemple de switch</title>
<style>
#rectangle{
  width: 30px; height: 20px;
  position: absolute;
  background-color:#bfb;
}
</style>
</head>

<body onkeydown="programa(event)"> <!-- cal posar la paraula event -->
<h2>Prem qualsevol tecla fletxa per moure el rectangle</h2>
<div id="rectangle" style="top:100px; left:100px;"> </div>
<script>
function programa(e){
  const LEFT=37, UP=38, RIGHT=39, DOWN=40;
  var rect=document.getElementById("rectangle");
  var top =parseInt(rect.style.top);
  var left=parseInt(rect.style.left);

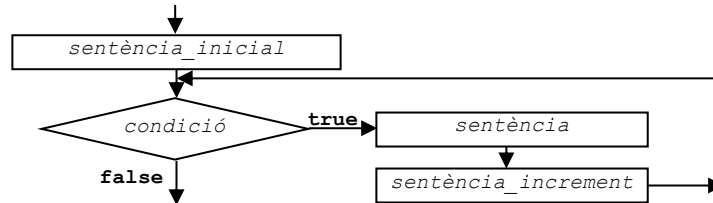
  switch(e.keyCode){
    case LEFT : rect.style.left=(left-5)+"px"; break;
    case RIGHT: rect.style.left=(left+5)+"px"; break;
    case UP    : rect.style.top =(top -5)+"px"; break;
    case DOWN  : rect.style.top =(top +5)+"px"; break;
  }
}
</script>
</body>
</html>
```

[http://ssh.eupmt.tecnocampus.cat/~jou/FP/capitol\\_1/exemples/switch.html](http://ssh.eupmt.tecnocampus.cat/~jou/FP/capitol_1/exemples/switch.html)

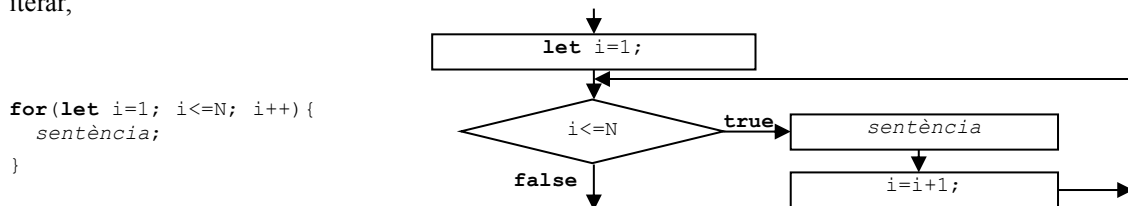
### 1.6.3 Estructures iteratives for, while.

Quan tinguem que repetir l'execució d'un conjunt de sentències un nombre conegut de vegades, farem servir la sentència **for**, amb la següent sintaxi,

```
for(sentència_inicial; condició; sentència_increment){  
    sentència;  
}
```



Farem servir una variable com a comptador del nombre d'iteracions que cal fer. Aquesta variable la inicialitzarem a 1 i a cada iteració la incrementarem una unitat fins arribar al nombre de vegades que cal iterar,



Nota: Si en l'execució de la sentència dins del **for** s'executa la sentència **break**; llavors s'avorta el **for** i el programa continua després d'aquest **for**.

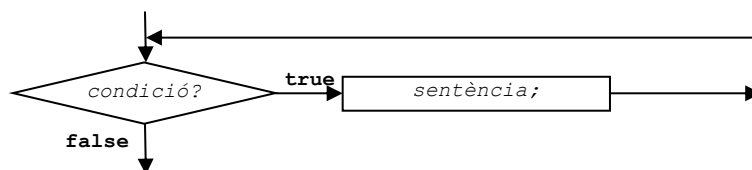
**Exemple 5.** Escriu una funció que retorni la suma dels primers 100 números, 1+2+3+...+100 .

```
function suma100(){  
    var suma=0;           // resultat  
    for(let i=1; i<=100; i++){ // i és una variable de bloc  
        suma=suma+i;  
    }  
    return suma;  
}
```

Prova-ho >>> [http://ssh.eupmt.tecnocampus.cat/~jou/FP/capitol\\_1/exemples/exemple09.html](http://ssh.eupmt.tecnocampus.cat/~jou/FP/capitol_1/exemples/exemple09.html)

Quan no sabem quantes vegades hem d'iterar però sí coneixem una condició que és necessària per seguir iterant, fem servir la sentència **while** amb la següent sintaxi

```
while(condició){  
    sentència;  
}
```



Aquesta sentència és molt versàtil però, alhora, molt perillosa perquè cal controlar que la sentència faci evolucionar la condició cap a l'estat **false** en alguna iteració. Sinó, es tractaria de un "bucle infinit", és a dir, el programa no s'aturaria mai.

**Exemple 6.** Escriu una funció que retorni la mínima quantitat de nombres consecutius que cal sumar, començant pel 1, de forma que la suma sigui superior a un nombre n donat .

```
function quantsSumen(n){  
    var i=0, suma=0; // variables auxiliars  
    while(suma<=n){  
        i++;  
        suma=suma+i;  
    }  
    return i;  
}
```

Prova-ho >>> [http://ssh.eupmt.tecnocampus.cat/~jou/FP/capitol\\_1/exemples/exemple10.html](http://ssh.eupmt.tecnocampus.cat/~jou/FP/capitol_1/exemples/exemple10.html)

## 1.7 Estructuració del programa en funcions.

Una funció és un programa petit que resol un problema més o menys elemental.

Quan el Programa ha de resoldre un problema gran, és millors atacar-lo amb un conjunt de funcions que resolen problemes més petits, però que, convenientment “cridades”, resoldran el problema gran. A aquesta manera de resoldre problemes de programació s’anomena disseny descendent (*top-down design*). Aquestes funcions que resolen petits problemes però que apareixen sovint en els programes, s’emmagatzemen en format de llibreria i serveixen per no haver de repetir codi en futur programes. El llenguatge Javascript és molt ric en llibreries de funcions que resolen petits problemes.

Totes les funcions pròpies (que no són llibreria) s’han de **definir** abans d’usar-les en el nostre programa amb la següent sintaxi,

```
function identificador(paràmetres) {
  // variables locals auxiliars
  // procés
  return expressió; // resultat
}
```

La sentència **return**, si hi és, sempre ha de ser la última instrucció que s’executa de la funció. A vegades, el programa fa uns càlculs (procés) amb unes dades inicials però no calcula cap resultat. En aquest cas no es posa la sentència **return**.

Un cop definida, aquesta funció es pot fer servir, és parla de **cridar** (*call*) a la funció, passant-li els valors de les dades inicials (arguments). La crida sol estar dins d’una expressió, de manera que el resultat de la funció agafa el lloc en l’expressió on s’ha cridat a la funció.

Tots els paràmetres de la funció, que són de tipus primitiu, es passen per valor (còpia). Per tant, la funció no pot canviar el valor dels arguments, només podrà canviar el valor dels paràmetres (còpia dels arguments). Per contra, els arguments de tipus objecte es passen als paràmetres per referència, és a dir, es passa una còpia de l’adreça de l’objecte argument, per tant, la funció, a través del seu paràmetre podrà canviar el valor de l’objecte argument perquè el paràmetre i l’argument estan “apuntant” al mateix objecte.

### 1.7.1 Funcions predefinides.

El llenguatge javascript disposa d’un conjunt de funcions predefinides (llibreria) que resolen problemes petits però molt freqüents. Algunes d’elles ja han aparegut en els exemples del capítol, sobre tot les de canviar el tipus de la dada:

<b>isNaN</b> ( <i>nombre</i> )	<i>is Not a Number</i> , funció booleana que retorna <b>true</b> si l’argument <b>no és un nombre vàlid</b> i <b>false</b> en qualsevol altre cas. Exemples:  <code>isNaN(3.5)</code> <b>ÉS false</b> <code>isNaN("hola")</code> <b>ÉS true</b>
<b>parseInt</b> ( <i>string</i> [, <i>base</i> ])	Converteix el <i>string</i> a un nombre enter a la <i>base</i> indicada (per defecte la <i>base</i> és 10). Si no pot fer la conversió retorna <b>NaN</b> . Exemples:  <code>parseInt("FF",16)</code> és el número 255, el pas contrari seria <code>(255).toString(16)=="ff"</code> <code>parseInt("-3.14")</code> és el número 3 <code>parseInt("4r")</code> és 4, és a dir, deixa de convertir quan ja no pot ser un número <code>parseInt("h3")</code> <b>ÉS NaN</b>
<b>parseFloat</b> ( <i>string</i> )	Converteix el <i>string</i> a un nombre real. Si no pot fer la conversió retorna <b>NaN</b> Exemple: <code>parseFloat("-1.4e-2")</code> és el nombre -0.014 <code>parseFloat("-1,4e-2")</code> és -1, és a dir, deixa de convertir quan ja no pot ser un número
<b>Math.random</b> ()	Retorna un número real aleatori entre 0 (inclòs) i 1.0 (exclòs), és a dir, $0 \leq \text{Math.random}() < 1.0$
<b>setInterval</b> ( <i>funcio</i> , <i>mseg</i> )	Executa la <i>funcio</i> repetidament després de passar <i>mseg</i> milisegons. Retorna un número que és l’identificador del interval per poder-lo parar amb <code>clearInterval</code> .
<b>clearInterval</b> ( <i>id</i> )	Atura l’interval <i>id</i> creat amb <code>setInterval</code>

Totes les funcions aritmètiques que hem vist de l’objecte `Math` també es consideren funcions predefinides.

## 1.8 Objecte String

Hi ha dues maneres de crear una cadena de caràcters o String:

1. Usant un valor literal.

Per exemples:

```
var cad1="hola"; // es poden fer servir cometes dobles o simples
```

2. Usant la classe String.

Per exemple:

```
var cad2 = new String(" que tal"); // crea un objecte String, " que tal"
```

Quan s'ha creat amb un valor literal, si sol·licitem un servei (mètode) a la cadena, aquesta es tradueix a un objecte String.

Existeix un únic operador pels Strings i és la concatenació, +. Recordeu que si els operands són una cadena i un número, l'operador + és concatenació de cadenes.

Per exemple:

```
var cad3=cad1+cad2; // cad3=="hola que tal"
```

### 1.8.1.1.1 Propietats.

Els String tenen la propietat `length` la qual ens informa de la quantitat de caràcters de la cadena:

<code>length</code>	És el nombre de caràcters de la cadena. Exemple: <pre>var x="Hola, que tal?".length; // x==14</pre>
---------------------	---

### 1.8.1.1.2 Mètodes.

Els mètodes més importants de qualsevol String són:

<code>charAt(index)</code> <code>charCodeAt(index)</code>	retorna el caràcter/codi numèric de la cadena que ocupa la posició indicada. El primer caràcter és a la posició 0 i el darrer és a <code>length-1</code> Exemple: <pre>var a="camió"; var x=a.charAt(1); // x=="a" var y=a.charCodeAt(1); // y==97</pre>
<code>indexOf(subcadena)</code> <code>lastIndexOf(subcadena)</code>	Retorna la posició en la cadena actual on troba per primer/darrer cop la subcadena. Retorna -1 si no l'ha trobat. Exemple: <pre>var a="hola, que tal estàs?"; var x=a.indexOf("que"); // x==6 x=a.indexOf("poma"); // x==-1, no l'ha trobat x=a.lastIndexOf("a"); // x==11</pre>
<code>toLowerCase()</code>	Transforma la cadena a tots els caràcters en minúscules. Exemple: <pre>var a="Hola. Que tal estàs, Joan?"; a.toLowerCase(); // a=="hola. que tal estàs, joan?";</pre>
<code>toUpperCase()</code>	Transforma la cadena a tots els caràcters en majúscules Exemple: <pre>var a="Hola. Que tal estàs, Joan?"; a.toUpperCase(); // a=="HOLA. QUE TAL ESTÀS, JOAN?";</pre>
<code>substr(index,longitud)</code>	Retorna la subcadena de <code>longitud</code> caràcters a partir de <code>index</code> en la cadena actual Exemple: <pre>var a="Hola. Que tal estàs, Joan?"; var b=a.substr(6,3); // b=="Que"</pre>



<b>substring</b> (index1,index2)	Retorna la cadena des de la posició <code>index1</code> fins a <code>index2</code> en la cadena actual Exemple: <pre>var a="Hola. Que tal estàs, Joan?"; var b=a.substring(6,18); // b="Que tal estàs"</pre>
<b>replace</b> (strVell,strNou)	Substitueix la primera vegada que troba <code>strVell</code> per <code>strNou</code> . Si <code>strVell</code> és l'expressió regular <code>/strVell/g</code> es reemplacen totes les vegades que troba <code>strVell</code> per <code>strNou</code> Exemple: <pre>var a="la le li lo lu la le li lo lu"; a.replace("la", "pa"); // "pa le li lo lu la le li lo lu" a.replace(/la/g, "pa"); // "pa le li lo lu pa le li lo lu"</pre>
<b>split</b> ("separador")	Retorna un array amb tots els strings que troba separats per <code>separador</code> . Exemples: <pre>var a="la le - li - lo - lu la le"; var x=a.split("-"); // x=["la le "," li "," lo "," lu la le"] "Hola".split(""); // ["H","o","l","a"]</pre>

## 1.9 Objecte Date .

Són objectes per treballar amb l'hora i les dates.

Hi ha 4 constructors diferents:

Date ()	data actual del sistema
Date (milisegons)	data expressada en milisegons des de 1 de gener de 1970
Date (String)	data expressada amb un String
Date (any,mes,dia [,hora,minut,segon])	data explícita. Els mesos van des de 0 (gener) fins a 11 (desembre)

### 1.9.1.1.1 Mètodes.

Els més important són:

getTime ()	obté la data de l'objecte expressada en milisegons
setTime (milisegons)	estableix la data expressada en milisegons des de 1 de gener de 1970 a les 0 hores

Existeixen els mètodes set i get per: Year, Month, Day, Hours, Minutes, Seconds.

**Exemple 7.** Escriviu un programa que mostri un rellotge digital amb l'hora actual.

**Rellotge digital: 11:49:26**

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>Exemple 5. Rellotge digital</title>
</head>

<body >
Rellotge digital: <strong id="rellotge"></strong>

<script>
var cronometre=setInterval(horaActual,1000);

function horaActual() {
    var data=new Date();
    var hora=data.getHours();
    var minuts=data.getMinutes();
    var segons=data.getSeconds();

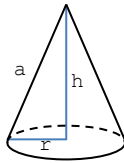
    if(hora<10) hora="0"+hora;
    if(minuts<10) minuts="0"+minuts;
    if(segons<10) segons="0"+segons;
    var hms=hora+":"+minuts+":"+segons;

    document.getElementById("rellotge").innerHTML=hms;
}
</script>
</body>
</html>
```

[http://ssh.eupmt.tecnocampus.cat/~jou/FP/capitol\\_2/exemples/exemple5.html](http://ssh.eupmt.tecnocampus.cat/~jou/FP/capitol_2/exemples/exemple5.html)

## 1.10 Exercicis resolts.

**Exercici 1.** Escribe dues funcions que calculin l'àrea i el volum d'un con recte donats el radi de la base,  $r$ , i l'altura del con,  $h$ .



$$A = \pi r^2 + \pi r a \quad \text{on } a = \sqrt{h^2 + r^2}$$

$$V = \frac{\pi r^2 h}{3}$$

```
function areaCon(r, h){
  var a=Math.sqrt(h*h+r*r);
  return Math.PI*r*r + Math.PI*r*a;
}

function volumCon(r, h){
  return Math.PI*r*r*h/3;
}
```

**Exercici 2.** Escribe un programa que generi un nombre enter aleatori dins del rang  $[a,b]$ . La IGU que es proposa és la següent

Generació de números aleatoris

Entra el rang: A = , B =

3

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>exercici 2</title>
<script>
function nombreAleatori(a,b){
  return Math.floor(Math.random()*(b+1-a))+a;
}
function programa6(){
  var A=parseInt(document.getElementById("A").value);
  var B=parseInt(document.getElementById("B").value);

  document.getElementById("resultat").innerHTML=nombreAleatori(A,B);
}
</script>
</head>

<body>
<fieldset>
<legend> Generació de números aleatoris</legend>
<p>
<label for="A">Entra el rang: A = </label> <input id="A" type="text" size="3" maxlength="3">,
<label for="B">B = </label> <input id="B" type="text" size="3" maxlength="3">
</p>
<p>
<input type="button" id="Executa" value="Genera un valor aleatori" onClick="programa6();">
<span id="resultat"></span>
</p>
</fieldset>
</body>
</html>
```

Prova-ho >>> [http://ssh.eupmt.tecnocampus.cat/~jou/FP/capitol\\_1/exercicis/exercici06.html](http://ssh.eupmt.tecnocampus.cat/~jou/FP/capitol_1/exercicis/exercici06.html)

### Exercici 3. Moure un rectangle al prémer una tecla fletxa.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>Exercici 3</title>
<style>
#rectangle{
  position: absolute;
  width: 30px; height: 20px;
  background-color:#bfb;
}
</style>
</head>

<body onkeydown="programa(event)"> <!-- cal posar la paraula event -->
<h2>Prem qualsevol tecla fletxa per moure el rectangle</h2>
<div id="rectangle" style="top:100px; left:100px;"> </div>
<script>
function programa(e){ <!-- el paràmetre e serà l'event passat com a argument -->
  const LEFT=37, UP=38, RIGHT=39, DOWN=40;
  var rect=document.getElementById("rectangle");
  var top =parseInt(rect.style.top); // podem accedir a les propietats CSS de l'element
  var left=parseInt(rect.style.left);

  // keyCode és el codi numèric (ASCII) de la tecla que ha produït l'event
  if(e.keyCode == LEFT){
    rect.style.left=(left-5)+"px";
  }
  else if(e.keyCode == RIGHT){
    rect.style.left=(left+5)+"px";
  }
  else if(e.keyCode == UP){
    rect.style.top =(top -5)+"px";
  }
  else if(e.keyCode == DOWN){
    rect.style.top =(top +5)+"px";
  }
}
</script>
</body>
</html>
```

Prova-ho >>> [http://ssh.eupmt.tecnocampus.cat/~jou/FP/capitol\\_1/exemples/switch.html](http://ssh.eupmt.tecnocampus.cat/~jou/FP/capitol_1/exemples/switch.html)

**Exercici 4.** Escriu un programa que donats un nombre  $n$  ( $n > 0$ ) escrigui la taula de multiplicar de  $n$ . La IGU que es proposa és la següent,

Entra el valor de  $n =$

Resultat:

9 x 0 = 0
9 x 1 = 9
9 x 2 = 18
9 x 3 = 27
9 x 4 = 36
9 x 5 = 45
9 x 6 = 54
9 x 7 = 63
9 x 8 = 72
9 x 9 = 81
9 x 10 = 90

```
<!DOCTYPE HTML>
<html lang="ca">
<head>
<meta charset="utf-8">
<title>Exercici 6</title>
<style>
.solucio {
    background-color: #FFC;
    padding: 10px;
}
</style>
</head>

<body>
<div class="solucio">
    <p>
        <label for="n">Entra el valor de n = </label><input id="n" type="text" size="3">
    </p>
    <p><input type="button" value="Executa" onClick="Prova_Exercici()" /></p>
    <p>Resultat: </p>
    <div id="resultat"> </div>
</div>

<script>
function Exercici(n){
    var resultat="<table border='1'>\n";
    for(let i=0; i<=10; i++){
        //resultat += "    <tr><td>"+n+" x "+i+" = "+(n*i)+"</td></tr>\n"; // ES5
        resultat += `    <tr><td>${n} x ${i} = ${n*i}</td></tr>\n`; // ES6
    }
    resultat += "</table>\n";
    return resultat;
}
function Prova_Exercici(){
    var n=parseInt(document.getElementById("n").value);
    document.getElementById("resultat").innerHTML = Exercici(n);
}
</script>
</body>
</html>
```

Prova-ho >>> [http://ssh.eupmt.tecnocampus.cat/~jou/FP/capitol\\_1/exercicis/exercici10.html](http://ssh.eupmt.tecnocampus.cat/~jou/FP/capitol_1/exercicis/exercici10.html)

**Exercici 5.** Escriu un programa que donat el nombre  $n$  ( $n > 1$ ) dibuixi amb asterisc (\*): un quadrat, un triangle equilàter o un triangle equilàter invertit. La figura mostra la IGU que es proposa. Per exemple, si el nombre es 3 el resultat seria:

- a. un quadrat

```
* * *
* * *
* * *
```

- b. un triangle equilàter

```
*
* *
* * *
```

- c. un triangle equilàter invertit

```
* * *
* *
*
```

IGU que es proposa.

Si el nombre  $n$  que se li dona és negatiu o zero, el programa hauria d'escriure un missatge indicant que el nombre donat no és vàlid.

```
<!DOCTYPE HTML>
<html lang="ca">
<head>
<meta charset="utf-8">
<title>Exercici 7</title>
<style>
.solucio {
  background-color: #FFC;
  padding: 10px;
}
</style>
</head>

<body>
<div class="solucio">
  <p>
    <label for="n">Entra el valor de n = </label><input id="n" type="text" size="3">
  </p>
  <p>
    <input type="button" value="quadrat" onClick="Prova_Exercici('quadrat') "><br>
    <input type="button" value="triangle equilàter"
      onClick="Prova_Exercici('triangle') " /><br>
    <input type="button" value="triangle equilàter invertit"
      onClick="Prova_Exercici('triangle_invertit') ">
  </p>
  <p>
    Resultat: <br>
    <textarea id="resultat" cols="45" rows="5"></textarea>
  </p>
</div>
```

```
<script>

function Prova_Exercici(tipus){
  var n=parseInt(document.getElementById("n").value);
  if(!(n>0)){
    alert ("El valor de n no és més gran que zero");
  }
  else{
    if(tipus=="quadrat"){
      document.getElementById("resultat").innerHTML=Exercici_a(n);
    }
    else if(tipus=="triangle"){
      document.getElementById("resultat").innerHTML=Exercici_b(n);
    }
    else if(tipus=="triangle_invertit"){
      document.getElementById("resultat").innerHTML=Exercici_c(n);
    }
  }
}

function Exercici_a(n){
  var resultat="";
  for(let f=1; f<=n; f++){ // per tota fila
    for(let c=1; c<=n; c++){ // per tota columna
      resultat += " * ";
    }
    resultat += "\n"; // canviem de fila
  }
  return resultat;
}

function Exercici_b(n){
  var resultat="";
  for(let f=1; f<=n; f++){ // per tota fila
    for(let c=1; c<=f; c++){ // per tota columna
      resultat += " * ";
    }
    resultat += "\n"; // canviem de fila
  }
  return resultat;
}

function Exercici_c(n){
  var resultat="";
  for(let f=1; f<=n; f++){ // per tota fila
    for(let c=1; c<=n-f+1; c++){ // per tota columna
      resultat += " * ";
    }
    resultat += "\n"; // canviem de fila
  }
  return resultat;
}

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

Prova-ho >>> [http://ssh.eupmt.tecnocampus.cat/~jou/FP/capitol\\_1/exercicis/exercici11.html](http://ssh.eupmt.tecnocampus.cat/~jou/FP/capitol_1/exercicis/exercici11.html)

**Exercici 6.** Escriu un programa que donat un text, compti el nombre de paraules que conté. La IGU que es proposa és la següent,

Entra el text:

En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivía un hidalgo de los de lanza en astillero, adarga antigua, rocín flaco y galgo corredor. Una olla de algo más vaca que carnero, salpicón las más noches, duelos y quebrantos los sábados, lentejas los viernes, algún palomino de añadidura los domingos, consumían las tres partes de su hacienda. El resto della concluían sayo de velarte, calzas de velludo para las fiestas con sus pantuflos de lo mismo, los días de entre semana se honraba con su vellori de lo más fino.

Compta paraules

Hi ha 100 paraules en el text.

Són aquestes paraules, ordenades alfabèticament:

acordarme adarga algo algún antigua astillero añadidura calzas carnero con con concluían consumían corredor cuyo de de de de de de de de de de della domingos duelos días el en en entre fiestas fino flaco galgo ha hacienda hidalgo honraba la lanza las las lentejas lo lo los los los los lugar mancha mismo mucho más más más no no noches nombre olla palomino pantuflos para partes que que quebrantos quiero resto rocín salpicón sayo se semana su su sus sábados tiempo tres un un una vaca velarte vellori velludo viernes vivía y y

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>Exercici 2</title>
</head>
<body>
<p>
  <label for="text">Entra el text: </label><br />
  <textarea id="text" cols="45" rows="5"></textarea>
</p>
<p> <input type="button" value="Compta paraules" onClick="programa()"> </p>
<p>Hi ha <strong id="quantos"> </strong> paraules en el text. </p>
<p>Són aquestes paraules, ordenades alfabèticament: <br>
  <strong id="paraules"> </strong></p>
<script>
function programa() {
  var text=document.getElementById("text").value;
  var resposta=comptaParaules(text);
  document.getElementById("quantos").innerHTML=resposta.quantos;
  document.getElementById("paraules").innerHTML=resposta.paraules;
}
function comptaParaules(text){
  var resposta = {paraules:"", quantos:0};
  /*
  // eliminem els caràcter separadors , . ; :
  text=text.toLowerCase(); text=text.replace(/,/g,""); text=text.replace(/\./g,"");
  text=text.replace(/;/g,""); text=text.replace(/:/g,"");
  */
  // es pot posar en una sola línia atès que el mètode retorna un string
  text=text.toLowerCase().replace(/,/g,"").replace(/\./g,"").replace(/;/g,"")
    .replace(/:/g,"");
  /*
  var textArray=text.split(" ");
  textArray=textArray.sort();*/
  // també es pot posar en una sola línia
  var textArray=text.split(" ").sort();
  for(var i=0; i<textArray.length; i++){
    resposta.paraules = resposta.paraules + textArray[i]+" ";
  }
  resposta.quantos=textArray.length;
  return resposta;
}
</script>
</body>
</html>
```

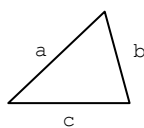
[http://ssh.eupmt.tecnocampus.cat/~jou/FP/capitol\\_2/exercicis/exercici2.html](http://ssh.eupmt.tecnocampus.cat/~jou/FP/capitol_2/exercicis/exercici2.html)



## 1.11 Exercicis enunciat.

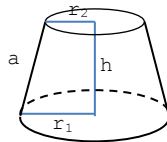
(★) fàcil  
 (★★) normal  
 (★★★) difícil

**Enunciat 1.** (★★★) Escriviu una funció que calculi l'àrea d'un triangle, donades les longituds dels seus tres costats amb la fórmula de Heró,



$$A = \sqrt{s(s-a)(s-b)(s-c)} \quad \text{on} \quad s = \frac{a+b+c}{2}$$

**Enunciat 1b.** (★★★) Escriviu dues funcions que calculin l'àrea i el volum d'un tronc de con donats el radi de la base,  $r_1$ , el radi de la tapa,  $r_2$ , i l'altura del con,  $h$ .



$$A = \pi[r_1^2 + r_2^2 + (r_1 + r_2)a] \quad \text{on} \quad a = \sqrt{h^2 + (r_1 - r_2)^2}$$

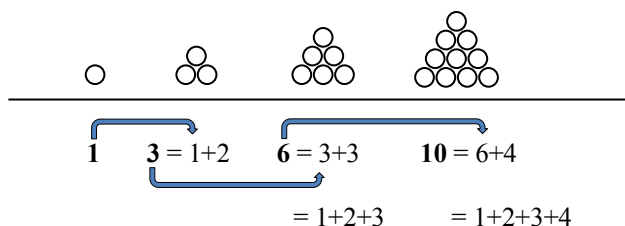
$$V = \frac{\pi h}{3}(r_1^2 + r_2^2 + r_1 r_2)$$

**Enunciat 2.** (★) Escriviu una funció, de nom `FtoC`, que donada una temperatura en graus Fahrenheit, calculi la temperatura equivalent en graus Celsius.

$$T_C = (T_F - 32) \frac{5}{9}$$

**Enunciat 3.** (★★★) Escriviu una funció que donat un número enter positiu més gran que 1, determini si el número és primer. Recordeu que un número enter és primer si és més gran que 1 i només es pot dividir de forma exacte per 1 i per si mateix. *Ajuda: Utilitza l'operador %*

**Enunciat 4.** (★★★) Escriviu una funció que escrigui els primers  $n$  números triangulars. Recordeu que un número enter positiu és triangular si el número "es pot apilar" en forma de triangle. Per exemple,



per tant,  $n$  és triangular si  $n = 1 + 2 + 3 + 4 + \dots + (k-1) + k$  i es diu que és el número triangular de  $k$  files. També es compleix que, si  $n$  és triangular de  $k$  files, el número triangular de  $k+1$  files és el número  $n+k$ .

**Enunciat 5.** (★★★) Escriviu una funció que calculi el valor de  $\pi \approx 3.14159$  de forma aproximada. Una manera de calcular el valor de  $\pi$  és calculant l'àrea d'un polígon regular de  $n$  costats inscrit en un cercle de radi 1. *Ajuda: utilitzeu la funció creada a l'enunciat 1. Recordeu el teorema del cosinus.*

**Enunciat 6.** (★★★) Escriviu una funció que escrigui tots els números forts de tres dígit. Un número fort és un número enter positiu que coincideix amb la suma dels seus dígitos elevats al cub. Per exemple, 371 és un número fort perquè  $371 = 3^3 + 7^3 + 1^3$ . *Ajuda: per trobar els dígit 7 del número 371 podeu fer* `n=371; x=n.toString(); digit=parseInt(x.charAt(1));`

**Enunciat 7.** (★) Escriviu una funció que donats dos números enters  $n$  i  $d$ , indiqui si  $d$  és un divisor propi de  $n$ . Perquè  $d$  sigui un divisor propi de  $n$  cal que  $n$  sigui divisible per  $d$  (divisió exacta) però  $d$  no pot ser  $n$ . *Ajuda:* %

**Enunciat 8.** (★★) Escriviu una funció que determini si dos números enters positius són amics. Dos números són amics si la suma dels divisors propis d'un és igual a l'altre i a l'inrevés. Per exemple, 220 i 284 són amics perquè els divisors propis de 220 són: 1, 2, 4, 5, 10, 11, 20, 22, 44, 55, 110 i la suma dona 284. Per altra banda, els divisors propis de 284 són: 1, 2, 4, 71, 142 i la suma dona 220.

**Enunciat 9.** (★★) Escriviu una funció que escrigui tots els números perfectes de quatre xifres. Un número perfecte és un número amic de si mateix. Per exemple, 28 és perfecte perquè els divisors propis de 28 són: 1, 2, 4, 7, 14 i la suma dona 28.

**Enunciat 10.** (★) Escriviu una funció que calculi el factorial d'un nombre enter  $n$ . El factorial de  $n$ , expressat  $n!$  dins d'una fórmula matemàtica, es calcula de la següent manera:  $n! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot \dots \cdot (n-1) \cdot n$ , és a dir, el factorial de  $n$  és el producte de tots els enters positius des de 1 fins a  $n$ . *Ajuda:* `for`

**Enunciat 11.** (★★) Escriviu una funció que calculi el sinus d'un angle expressat en graus. La funció  $\sin(x)$  es pot aproximar, per a angles expressats en radians, per la següent fórmula:

$$\sin(x) \cong x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!}$$

*Ajuda:* Utilitzeu la funció `factorial(n)` de l'exercici anterior. Recorda que  $\pi$  radians =  $180^\circ$

**Enunciat 12.** (★★★★) Escriviu una funció `toBase(n, b)` que donats un nombre natural  $n$  i una base  $b$ : 2..16 retorni un `String` que sigui la representació del valor  $n$  en base  $b$ .

**Enunciat 13.** (★★) Escriviu un programa que permeti veure un valor numèric expressat en base decimal a base binària i hexadecimal. La IGU que es proposa és,

entra el número decimal	escull base	resultat
256	<input type="radio"/> binari <input checked="" type="radio"/> hexadecimal	100

**Enunciat 14.** (★★) Escriviu un programa que permeti veure un valor numèric expressat en base hexadecimal a base decimal i binària. La IGU que es proposa és,

entra el número hexadecimal	escull base	resultat
A2F	<input checked="" type="radio"/> decimal <input type="radio"/> binari	2607

**Enunciat 15.** (★★) Escriviu un programa que permeti donat un import en €, calculi el nombre mínim de monedes en que es pot pagar. Les monedes en Euros són: 1cèntim, 2cèntims, 5cèntims, 10cèntims, 20cèntims, 50cèntims, 1€, 2€,



**Enunciat 16.** (★★★★) Escriviu una funció que verifiqui que el número de **Ramanujan** 1729 és *interessant*, és a dir, és el número natural més petit que es pot expressar com a suma de dos cubs positius de dues maneres diferents:

$$1729 = 1^3 + 12^3 = 9^3 + 10^3$$