3

3 Funcions i arrays.

En aquesta lliçó aprofundirem en el coneixement de les funcions i veurem algunes característiques avançades que ens permetran modelar alguns dels conceptes propis d'una bona programació, com és la modularitat dels nostres programes.

En aquests moments ja coneixem els valors de tipus primitius: números, booleans, cadenes de caràcters, **`null`** i **`undefined`**. En aquesta lliçó veurem una col·lecció de dades, és a dir, una agrupació de valors. Hi ha dos grans col·leccions: els arrays (llistes indexades) que veurem en aquesta lliçó i els objectes (llistes associatives) que veurem en la propera lliçó.

3.1 Funcions: conceptes avançats.

3.1.1 Definició i crida a una funció.

Ja hem vist que les funcions són petits programes que resolen problemes petits. Cal diferenciar entre la definició de la funció i la crida a la funció i, al igual que les variables, podem declarar una variable o declarar (definir) una funció més tard que la inicialització de la variable o la crida a la funció, respectivament. A això s'anomena *hoisting*, és a dir, el intèrpret primer busca totes les declaracions de variables i definicions de funcions abans de procedir a les inicialitzacions o crides.

Per exemple, aquests dos codis són correctes, tot i que el segon és més endreçat,

```
var a;              // declaració de variable
b=minim(3, 5.6);    // crida i inicialització

function minim(x,y){ // declaració de funció
  if(x<y)
     return x;
  else
    return y;
}

var b;              // declaració de variable
a=minim(4.5, minim(2,Math.random()*5));
```

```
function minim(x,y){ // declaració de funció
  if(x<y)
     return x;
  else
    return y;
}

var a, b;  // declaració de variables global
a=minim(4.5, minim(2,Math.random()*5));
b=minim(3, 5.6);
```

Recordeu que tots els arguments de tipus primitiu es passen per valor i els arrays i objectes que veurem en aquesta lliçó es passen per referència.

3.1.2 Expressions funció.

Hi ha una segona manera de declarar una funció, són les expressions funció,

```
var quadrat=function(x){return x*x;};  // expressió funció
console.log(quadrat(3)); // 9
```

En aquest cas sí que cal escriure abans l'expressió funció que la crida.

Es pot crear la funció, amb el seu codi, en temps d'execució a través d'un objecte `Function`, per exemple:

```
var quadrat=new Function("x","return x*x;"); // el darrer paràmetre és el codi de la funció
console.log(quadrat(3)); // 9
```

Normalment, les expressions funció són anònimes (no tenen identificador) però el poden tenir, per exemple, si és una funció que s'ha de cridar a si mateixa (recursivitat),

```
var factorial=function fact(n){
  if(n>1)
    return n*fact(n-1);
  else
    return 1;
};
console.log(factorial(5)); // 120
```

Les funcions expressió són útils per passar-les com argument a una altra funció, com per exemple, en les funcions *callback* o en la composició de funcions, $f(x)\circ g(x)=f(g(x))$, per exemple

$$\left(\frac{1}{x}\right)^2+2$$

```
function f(x,g){
  return g(x)*g(x)+2;

}
console.log(f(2,function(x){return 1/x;})); // 2.25  passem una funció anònima
```

De la mateixa manera, una funció pot retornar una altra funció,

```
function f(x){
  if(x<0)
    return function(x){return 1/x;};
  else
    return function(x){return x*x;};
}
var y=f(-1); console.log(y(-4)); // -0.25
console.log(f(1)(4));            // 16
```

Podem fer que s'executi una funció una única vegada amb la següent instrucció,

```
(function(){/* codi de la funció */})();
```

Amb la nova especificació EMACScript 6 es pot definir una funció expressió **anònima** amb la notació fletxa,

```
function f(x){
  if(x<0)
    return (x) => {return 1/x};
  else
    return x => x*x;  // si només hi ha una instrucció return, no cal {}
}
var y=f(-1); console.log(y(-4)); // -0.25
console.log(f(1)(4));            // 16
```

Les funcions expressió fletxa no estan vinculades amb **`this`** ni **`super()`**, per tant no es poden fer servir per a mètodes ni per a la funció **`constructor()`**.

3.1.3 Funcions internes i clausures.

Es pot declarar una funció dins una altra funció, es parla de funcions internes. L'àmbit d'ús de la funció interna és el propi bloc de la funció que la conté. Veiem l'àmbit en el següent exemple,

```
var m,n;

// es pot utilitzar m, n i f1

function f1(x,y){
  var z;

  // es pot utilitzar m, n, f1,
  // també x, y, z i f2

  function f2(a,b){
    var c;

    // es pot utilitzar m, n, f1,
    // també x, y, z, f2
    // i també a, b, c i f3

    function f3(i,j){
      var k;

      // es pot utilitzar m, n, f1,
      // també x, y, z, f2
      // també a, b, c, f3
      // i també i, j, k


    }

  }
}
```

com veieu, la funció externa no pot veure el que hi ha dins de la funció interna però a l'inrevés sí, la funció interna pot usar tot l'entorn de les funcions que la contenen. Es parla que la funció interna és una **clausura**, ella veu tot el que està fora però, des de fora no es pot accedir a l'interior de la funció interna. La clausura "lliga" a la funció interna tot l'àmbit extern que utilitza:

```
function f(x){
   return function g(y){return x+y;}
}
var funcioInterna=f(3);
console.log(funcioInterna(5)); // 8, funcioInterna recorda la variable x=3, per la clausura
console.log(f(3)(5); // 8
```

Fixem-nos en el següent exemple,

```
var persona=function(nom){
  var edat;
  return {  // retornem un objecte literal amb quatre mètodes
    setNom : function(nouNom){nom= nouNom;},
    getNom : function(){return nom;},
    setEdat: function(novaEdat){edat=parseInt(novaEdat);},
    getEdat: function(){return edat;}
  }
}
```

```
var p=persona("Joan");  // atès que retorna funcions internes que utilitzen l'entorn de la funció
                        // cridada, l'entorn d'aquesta es conserva (nom, edat) mentre es pugui
                        // utilitzar la funció externa. Serà una manera de crear
                        // variables "privades".

p.getNom();             // Joan
p.setNom("Joanot");
p.setEdat("3.5anys");
p.getNom();             // Joanot
p.getEdat();            // 3
```

cada cop que declarem una variable i li assignem la funció externa, es crearà un entorn de dos variables: `nom` i `edat` que es conservaran encara que la funció externa ja s'ha executat perquè a través de la nova variable es pot accedir a funcions internes que utilitzen aquest entorn. Això permet crear **encapsulació**, és a dir, variables privades que només es podran modificar amb l'us extern de funcions internes.

3.1.4 Paràmetres múltiples i paràmetres per defecte.

Totes les funcions tenen un paràmetre `arguments` que és un objecte similar a un array: té la propietat `length` i es pot utilitzar la notació claudàtor, però, no té els mètodes de l'objecte `Array`.

Així, tota funció podrà saber quants arguments s'han passat en la crida. Per exemple,

```
function sumar(){
  var resultat=0;
  for(var i=0; i<arguments.length; i++) // també sumar.arguments.length però, sumar. és opcional
      resultat += arguments[i];
  return resultat;
}
sumar(2,4,6,10); // 22
sumar(-1,5);     // 4
```

Si volem tenir paràmetres amb valors per defecte podem fer el següent:

EMACScript 5

```
function multiplicar(a,b){
  b=(typeof b !== "undefined")?b:1;
  return a*b;
}
multiplicar(5,3);  // 15
multiplicar(24);   // 24
```

EMACScript 6

```
function multiplicar(a,b=1){
  return a*b;
}
multiplicar(5,3);  // 15
multiplicar(24);  // 24
```

En EMACScript 6 també tenim la possibilitat d'agrupar els paràmetres des d'una posició fins al final en un array, per exemple,

```
function multiples(x, ...laResta) {  // el paràmetre laResta és un Array (els veurem tot seguit)
  return laResta.map(function(elementActual) {
                       return x * elementActual;
                     });
}
console.log(multiples(3, 1,2,3,4,5,6,7,8,9)); // Array [ 3, 6, 9, 12, 15, 18, 21, 24, 27 ]
```

3.2 Objecte Array.

Un array és un tipus especial d'objecte. És una col·lecció de variables però que permeten ser accedides a través d'un índex. Es parla de "llista indexada".

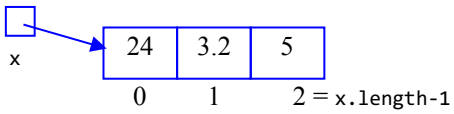
La col·lecció pot ser heterogènia però el més normal és que sigui homogènia, és a dir, totes les variables de l'array contenen el mateix tipus de valor.

3.2.1 Creació d'arrays.

Hi ha dues maneres de crear un array:

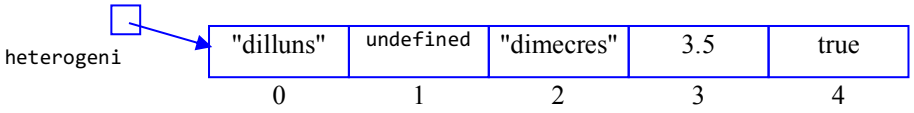
1. **Usant una constant literal**.
   Per exemple:

```
var x=[24, 3.2, 5] // ha creat x[0]=24, x[1]=3.2 i x[2]=5
```



   tenim doncs, quatre variables, la col·lecció o array, `x`, i les variables individuals `x[0]`, `x[1]` i `x[2]`

```
var heterogeni=["dilluns",,"dimecres",3.5,true];
```



   tenim, sis variables, l' array, `heterogeni`, i les variables individuals `heterogeni[0]`, `heterogeni[1]`, `heterogeni[2]`, `heterogeni[3]` i `heterogeni[4]`

   Per crear un array "buit" faríem:

```
var a=[]; // l'array a està buit, la col·lecció està buida
```

2. **Usant un constructor d'objectes Array**. Hi ha dos constructors, un que explicita els valors inicials de l'array i un altre que només indica la grandària (quan només hi ha un argument).

   Per exemple:

```
var x = new Array(24,3.2,5);         // crea el mateix array que abans
var diesLaborables = new Array(5);   // crea un array de 5 posicions undefined
diesLaborables[0]="dilluns;
diesLaborables[1]="dimarts;
diesLaborables[2]="dimecres";
diesLaborables[3]="dijous";
diesLaborables[4]="divendres";
var aBuit= new Array();              // l'array aBuit està buit
```

Podem fer créixer l'array assignant en una posició que no existeix:

```
x[4]=2; // s'ha creat dues posicions més, x[3]=undefined i x[4]=2
```

és a dir, les posicions han de ser consecutives començant per l'índex 0.

3.2.2 Accés als elements de l'array.

Per accedir a un element de l'array fem servir la notació claudàtor,

```
nomArray[index]
```

on `index` és un nombre enter des de 0 fins a `nomArray.length-1.`

Tot objecte array té la propietat **`length`** que ens informa de la grandària de la col·lecció.

Per recórrer tots els elements de l'array podem fer servir la sentència **`for(var i in nomArray)`** que serà equivalent a **`for(var i=0; i<nomArray.length; i++)`**, per exemple,

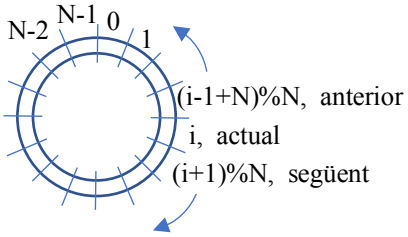
```
function suma(a){
  var resultat=0;
  for(var i in a){      // equivalent a for(var i=0; i<a.length; i++)
    resultat += a [i];
  }
  return resultat;
}
console.log(suma([1,2,3,4,5]));  // 15
```

Hi ha una altra manera d'iterar tot l'array, a través del mètode `forEach()`,

```
var colors=["vermell", "verd", "blau"];
colors.forEach(function(color){
  console.log(color);
});
```

on, la funció passada al `forEach` s'executa una vegada per cada ítem de l'array el qual és passat com argument a la funció. Els ítems `undefined` no són iterats.

Podem recórrer l'array de forma cíclica de la següent manera:



3.2.3 Arrays multidimensionals.

Un element d'un array pot ser un altre array. Això ens permet treballar amb arrays multidimensionals. Per exemple, anem a definir una matriu de 4 files per 3 columnes,

```
var matriu=new Array(4);
for(let i=0; i<4; i++){     // recordeu que let defineix la variable en l'àmbit del bloc
  matriu[i]=new Array(3);
  for(let j=0; j<3; j++){
    matriu[i][j]=i+" , "+j;
  }
}
console.log(matriu);
```

dóna el següent resultat,

```
[["0,0", "0,1", "0,2"],
 ["1,0", "1,1", "1,2"],
 ["2,0", "2,1", "2,2"],
 ["3,0", "3,1", "3,2"]]
```

Sovint però, és millor treballar l'estructura matriu amb un únic array fent el següent,

```
           j=0    1    nColumnes-1
   i=0   [["0,0", "0,1", "0,2"],
     1    ["1,0", "1,1", "1,2"],  --> ["0,0", "0,1", "0,2", "1,0", "1,1",...,"3,1", "3,2"]
     2    ["2,0", "2,1", "2,2"],       k=0    1     2     3     4              nFiles * nColumnes -1
 nFiles-1  ["3,0", "3,1", "3,2"]]
```

```
var matriu = new Array(nFiles*nColumnes);
for(let i=0; i<nFiles; i++){
  for(let j=0; j<nColumnes; j++){
    matriu[i*nColumnes+j]=i+" , "+j;
  }
}
```

Amb aquesta tècnica, es podrà utilitzar tots els mètodes de `Array()` que veurem tot seguit.

3.2.4 Mètodes dels arrays.

Recordeu que els mètodes són les funcions pròpies de l'objecte que serveixen per consultar o canviar l'estat del mateix. Per "cridar" a un mètode d'un objecte fem:

```
nomObjecte.nomMetode(llista_arguments)
```

Els mètodes més importants de qualsevol objecte array són:

`toString()`	Expressa el contingut de l'array com un string mostrant tots els valors
	
	Exemple:
	`var a=[3,9,5,20,7,6,10];`
	`a.toString();  // "3,9,5,20,7,6,10"`

`indexOf(valor[,index])`
`lastIndexOf(valor[,index])`
	Retorna la posició en l'array on troba per primer/darrer cop el valor.
	Retorna `-1` si no l'ha trobat. Comença la cerca a partir de `index`. Si no es posa `index`, comença la cerca des de la posició `0`. Un valor negatiu de index, comença a buscar anant cap a l'esquerra
	
	Exemple:
	`var a=["pera", "poma", "pera", "poma", "tomaquet"];`
	`var x=a.indexOf("poma");  // x==1`
	`x=a.indexOf("poma",2);  // x==3`
	`x=a.indexOf("taronja");  // x==-1, no l'ha trobat`
	`x=a.lastIndexOf("pera");  // x==2`

`concat(llista_arrays)`	afegeix els arrays subministrats al final de l'array actual.
	
	Exemple:
	`var a1=["a","b"];  // construim amb un literal`
	`var a2=new Array("c","d"); // construim amb un constructor`
	`var a=a1.concat(a2);  // a==["a","b","c","d"]`

`sort([funcio_Comp])`	Ordena els elements de l'array alfabèticament (com strings). Si es subministra una funció de comparació, s'ordena usant aquesta funció. Aquesta funció ha d'acceptar dos paràmetres i retornar 0 si són iguals, < 0 si el primer és més petit que el segon, > 0 si el primer és més gran que el segon. Per ordenar números sempre s'haurà de subministrar aquesta funció
	
	Exemple:
	`var a=[3,9,5,20,7,6,10];`
	`a.sort();  // a==[10,20,3,5,6,7,9]`
	`a.sort(function(a,b){return a-b;});  // a==[3,5,6,7,9,10,20]`
	`// per tornar a desordenar podem fer,`
	`a.sort(function(a,b){return Math.random()-0.5}`

`reverse()`	Inverteix l'array, el primer element passa a ser el darrer i el darrer passa a ser el primer.
	
	Exemple:
	`var b=["b","d","a","t"];`
	`b.reverse();  // b==["t","a","d","b"]`

`slice(indexI[,indexF])`	Crear un nou array amb els elements que hi ha des de la posició `indexI` fins a la posició `indexF-1`. Si no es posa aquest segon argument, s'agafa fins al final de l'array.
	
	Exemple:
	`var a=[2,4,1,7,9,5,2,7,6,1];`
	`var b=a.slice(3,7); //  b=[7,9,5,2]`
	`b=a.slice(3);  //  b=[7,9,5,2,7,6,1]`

`push(llista_valors)`	Afegeix els nous elements al final de l'array. Retorna la nova dimensió de l'array.
	
	Exemple:
	`var a=["abril","maig","juny"];`
	`a.push("juliol","agost"); // a==["abril","maig","juny","juliol","agost"]`

`pop()`	Treu i retorna el darrer element de l'array
	
	Exemple:
	`var a=[3,9,5,2,7,6,1];`
	`var x=a.pop(); // x==1  a=[3,9,5,2,7,6]`

`shift()`	Treu i retorna el primer element de l'array
	
	Exemple:
	`var a=[3,9,5,2,7,6,1];`
	`var x=a.shift(); // x==3  a=[9,5,2,7,6,1]`

`unshift(valor)`	Afegeix un nou element al principi de l'array. Retorna la nova dimensió de l'array
	
	Exemple:
	`var a=["abril","maig","juny"];`
	`a.unshift("febrer","març"); // a==["febrer","març","abril","maig","juny"]`

`forEach(callback)`	Permet accedir a tots elements de l'array i executar la funció amb cadascun d'ells. Aquesta funció no hauria de canviar els valors l'array.
	
	Exemple:
	`var a=[10,2,13,7,5],suma=0;`
	`a.forEach(function(x){`
	`  suma+=x;`
	`});`
	`console.log(suma);  // 37`

`map(callback)`	Retorna un nou array amb els valors retornats d'aplicar la funció a tots els elements del primer array,
	
	Exemple:
	`var a=[1,2,3];`
	`var b=a.map(function(x){`
	`  return x*x;`
	`});`
	`console.log(b);  // Array [1,4,9]`

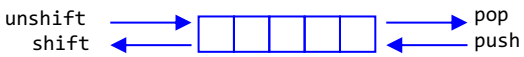
`filter(callback)`	Retorna un nou array amb els valors del primer array que la funció a retornat true,
	
	Exemple:
	`var a=[10,2,13,7,5];`
	`var b=a.filter(function(x){`
	`  return x>5;`
	`});`
	`console.log(b);  // Array [10,13,7]`

`reduce(callback)`	Retorna un valor després d'anar "acumulant" un resultat aplicant la funció a tots els elements de l'array,
	
	Exemple:
	`var a=[10,2,13,7,5];`
	`var suma=a.reduce(function(acumulat,x){`
	`  return acumulat+x;`
	`}, 100);  // valor inicial de acumulat, per defecte 0`
	`console.log(suma);  // 137`

Els mètodes `map`, `filter` i `reduce` permet implementar la **programació funcional**.
Per exemple, volem escriure una funció que calculi l'array intersecció de dos donats,

```
var a=[4,2,7,5,9,1];
var b=[4,3,7,6,8,2,5];
function interseccio1(a,b){
  var resultat=[];
  for (let i = 0; i < a.length; i++) {
    for (let j = 0; j < b.length; j++) {
      if(a[i]===b[j]){
        resultat.push(a[i]);
        break;
      }
    }
  }
  return resultat.sort();
}
function interseccio2(a,b){
  return a.filter(function(x){
    return b.indexOf(x)>-1;
  }).sort();
}
const interseccio3=(a,b)=>a.filter(x=>b.indexOf(x)>-1).sort();
/* es llegueix de la següent manera: creem una funció expressió de nom interseccio3 a la qual se
li passa dos arrays a, b. Aquesta funció retorna un array ordenat amb tots els elements de a que
també es troben a b.
*/
console.log("interseccio1(a,b) = "+interseccio1(a,b));  // [2,4,5,7]
console.log("interseccio2(a,b) = "+interseccio2(a,b));  // [2,4,5,7]
console.log("interseccio3(a,b) = "+interseccio3(a,b));  // [2,4,5,7]
```

Es pot crear una pila o una cua fent servir els mètodes adequats:



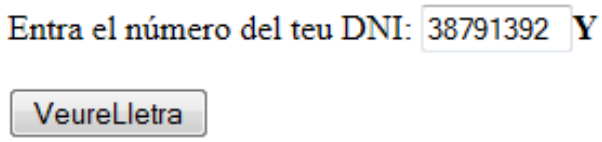
Pila: `var p=new Array(); p.push(e); var e=p.pop(); e.length; e.length=0`
Cua: `var c=new Array(); c.push(e); var e=p.shift(); e.length; e.length=0`

Si volem un diccionari de parelles clau-valor o un conjunt d'elements cal fer servir objectes de les classes `Map` o `Set`, respectivament:

Diccionari:
```
var m=new Map(); m.set("clau","valor"); var valor=m.get("clau");
m.has("clau"); m.delete("clau"); m.size; m.clear();
for(var [clau,valor] of m){console.log('('+clau+','+valor+')');}
```
Conjunt:
```
var s=new Set(); s.add(item); s.delete(item); s.has(item); s.size; s.clear();
for(var item of s){console.log(item);}
```

3.3 Exercicis.

Exercici 1. Escriu un programa que calculi la lletra del DNI donat el seu número. La interfície que es proposa és la següent:



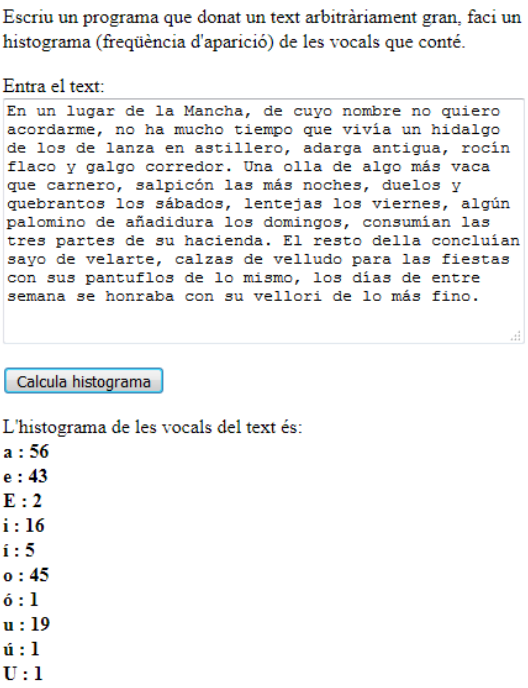

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>exemple 3, Calcul de la lletra del DNI</title>
</head>

<body>
<p>
<label for="dni">Entra el número del teu DNI:</label>
<input type="text" id="dni" size="8" maxlength="9"><strong id="lletraNIF"></strong>
</p>
<p><input type="button" value="VeureLletra" onClick="calcularLletraDni()"></p>

<script>
function calcularLletraDni(){
  var dni = parseInt(document.getElementById('dni').value);
  if (isNaN(dni)){
    alert("Has d'entrar un número");
    return;
  }
  var lletres=
      ["T","R","W","A","G","M","Y","F","P","D","X","B","N","J","Z","S","Q","V","H","L","C","K","E"];
  var lletraNif = lletres[dni%23];
  document.getElementById("lletraNIF").innerHTML=lletraNif;
}
</script>
</body>
</html>
```

http://ssh.eupmt.tecnocampus.cat/~jou/LM/exemples/exemple4a.html

Exercici 2. Escriu un programa que compti la freqüència d'aparició de cadascuna de les vocals en un text donat. La interfície que es proposa és:




```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>histograma (freqüència d'aparició) de les vocals</title>
</head>

<body>
<p>Escriu un programa que donat un text arbitràriament gran, faci un histograma (freqüència
d'aparició) de les vocals que conté.</p>
<p>
  Entra el text: <br />
  <textarea id="text" cols="45" rows="5"></textarea>
</p>
<p> <input type="button" value="Calcula histograma" onClick="programa()"> </p>
<p>L'histograma de les vocals del text és:  </br>
 <strong id="histograma"> </strong></p>


<script>
function programa(){
  var text=document.getElementById("text").value;
  var resposta=histogramaCaracters(text);
  document.getElementById("histograma").innerHTML=resposta;
}
```

```
function histogramaCaracters(text){
  var resultat="";

   // solució treballant amb arrays
  var histogramaVocal = ["a","à","A","À",
                         "e","é","è","E","É","È",
                         "i","í","I","Í",
                         "o","ó","ò","O","Ó","Ò",
                         "u","ú","ü","U","Ú","Ü"];
  var histogramaFrequencia = [0,  0,  0,  0,
                              0,  0,  0,  0,  0,  0,
                              0,  0,  0,  0,
                              0,  0,  0,  0,  0,  0,
                              0,  0,  0,  0,  0,  0];
   function posicioVocal(v){  // funció interna a una altra funció
     for(var p=0; p<histogramaVocal.length; p++){
       if(histogramaVocal[p]==v)
         return p;
     }
     return -1; // quan no és una vocal
  }

  var p;
   for(var i=0; i<text.length; i++){
     p=posicioVocal(text.charAt(i));
     if(p!=-1)
         histogramaFrequencia[p]++;

   }
   for(p=0; p<histogramaVocal.length; p++){
      if(histogramaFrequencia[p]!=0){
         resultat += histogramaVocal[p]+" : "+histogramaFrequencia[p]+"</br>";
       }
   }

   /*
   // solució treballant amb objectes
   var histograma = {"a":0, "à":0, "A":0, "À":0,
                     "e":0, "é":0, "è":0, "E":0, "É":0, "È":0,
                     "i":0, "í":0, "I":0, "Í":0,
                     "o":0, "ó":0, "ò":0, "O":0, "Ó":0, "Ò":0,
                     "u":0, "ú":0, "ü":0, "U":0, "Ú":0, "Ü":0 };

   for(var i=0; i<text.length; i++){
      if(histograma[text.charAt(i)]!=undefined)  // accedim a la propietat amb ["propietat"]
           histograma[text.charAt(i)]++;
   }
   for(var c in histograma){
       if(histograma[c]!=0)                    // accedim a la propietat amb [variable]
      resultat += c + " : " + histograma[c] + "</br>";
   }
   */
   return resultat;
}
</script>
</body>
</html>
```

http://ssh.eupmt.tecnocampus.cat/~jou/LM/exemples/exemple4b.html