Orientació a objectes en JavaScript

Vicent Moncho Mas

PID_00191131

UOC
Universitat Oberta de Catalunya

Índex

1. La programació orientada a objectes en JavaScript

1.1. Introducció

La programació orientada a objectes (POO) és actualment el paradigma de programació més utilitzat. Fa servir l'abstracció per a crear un model basat en el món real i es tracta d'un model organitzat, d'una banda, entorn dels objectes en comptes de les accions i, de l'altra, entorn de les dades en comptes de la lògica.

Històricament, un programa era vist com un conjunt de processos lògics, que, a partir d'unes dades d'entrada, es processaven i produïen unes dades de sortida.

La programació orientada a objectes utilitza objectes (estructures de dades i mètodes) i les interaccions d'aquests objectes per a dissenyar aplicacions. En aquest paradigma cada objecte s'ha de veure com una màquina senzilla que és responsable de fer un seguit de feines per a les quals ha estat implementada.

JavaScript implementa els quatre principis bàsics de la programació orientada a objectes (abstracció, encapsulació, herència i polimorfisme), però a diferència d'altres llenguatges, com Java, és possible programar en JavaScript sense haver d'utilitzar exclusivament aquestes característiques.

Aquesta darrera particularitat, juntament amb el fet que la majoria dels scripts no tenen massa complexitat, ha afavorit que una bona part dels programadors no utilitzin les característiques de la POO en JavaScript.

Es poden **classificar els objectes en els tipus següents:**

1) Objectes definits per l'usuari. Aquests objectes els defineix i els crea el programador amb l'objectiu d'estructurar la informació que es manipularà en els programes.

Vegeu també

El segon apartat d'aquest mòdul se centrarà a estudiar la creació d'objectes personalitzats.

2) Objectes natius de JavaScript. En aquest conjunt hi ha els objectes següents:

- Objectes associats als tipus de dades primàries, com String, Number i Boolean, que proporcionen propietats i mètodes a aquests tipus de dades.
- Objectes associats a tipus de dades compostes, com Array i Object.

- Objectes que proporcionen utilitats, com Date, Math i RegExp.

Les característiques d'aquests objectes les especifica l'estàndard ECMA-262, encara que, com és habitual, les companyies hi afegeixen les seves pròpies particularitats.

3) Objectes del navegador. Com els objectes Windows i Navigator, aquests objectes permeten manipular les finestres i interaccionar amb l'usuari.

Aquest tipus d'objectes se solen conèixer com a *BOM* (*browser object model*). El fet que estiguin relacionats íntimament amb el navegador fa que no siguin estàndard i variïn depenent de la versió que s'utilitza.

4) Objectes del document. Formen una part de l'especificació del DOM (*document object model*) i defineixen l'estructura de la interfície de la pàgina HTML. El consorci W3C ha dut a terme una especificació estàndard del DOM, per mirar de fer-hi convergir les diverses estructures implementades.

La classificació anterior pot ser una ajuda per a estudiar el model de programació, però evidentment els diversos tipus d'objectes tenen un cert encavalcament. No hi ha un estàndard que defineixi tots els aspectes de JavaScript: l'ECMA-262 regula els aspectes bàsics del llenguatge i l'especificació DOM de W3C defineix com s'han de presentar els documents estructurats (pàgines web) en un llenguatge de programació.

ECMA International

Es tracta d'una organització basada en filiacions d'estàndards per a la comunicació i la informació. L'organització es va fundar el 1961 per a estandarditzar els sistemes computats a Europa. El Consorci World Wide Web (W3C) és un consorci internacional que produeix recomanacions per al World Wide Web. El dirigeix Tim Berners-Lee.

Encara que la tendència actual de les companyies és complir els estàndards internacionals ECMA-262 i DOM de W3C, aquestes companyies continuen definint els seus propis models d'accés a la interfície d'usuari i creant les seves pròpies extensions del DOM.

En aquest primer apartat, es presentaran els conceptes i la metodologia que s'utilitzen en JavaScript per a la programació orientada a objectes.

1.2. Principis bàsics en JavaScript

Tal com s'ha dit en el subapartat anterior, JavaScript compleix els principis bàsics de la programació orientada a objectes. Tot seguit es presenten aquests principis i les tècniques que ho fan possible.

Vegeu també

Els principis i les tècniques que permeten la programació orientada a objectes s'estudiaran en detall en l'apartat 2 d'aquest mòdul.

1) Abstracció. El concepte d'*abstracció* es basa en la idea que un objecte ha de representar una certa idea o tasca, de manera que l'objecte ha de tenir una interfície que proporcioni les característiques o propietats i les accions o mètodes que se n'esperen.

Aquest principi s'aconsegueix a partir de la combinació de l'herència, els mètodes i les propietats dels objectes de JavaScript.

2) Encapsulació. El concepte d'*encapsulació* es basa en el fet que els objectes han de mantenir internament l'estat que els permet definir el comportament que tenen. Aquestes dades han d'estar ocultes per a la resta d'objectes i, si són accessibles, només ho són a partir de la interfície pública que l'objecte mateix proporciona.

En JavaScript, l'encapsulació s'aconsegueix amb l'herència i la definició de variables locals que només es modifiquen o a les quals només s'accedeix a partir de mètodes que ho permeten.

3) Herència. El concepte d'*herència* es basa en la possibilitat de crear objectes especialitzats a partir d'objectes més generals, de manera que aquests disposin de les propietats i dels mètodes dels objectes ascendents.

La tècnica que s'utilitza per a implementar herència es basa en l'ús de prototips.

4) Polimorfisme. El concepte de *polimorfisme* es basa en el fet que diferents objectes han de respondre de manera diferent a la crida d'un mateix mètode, però no solament això, també s'interpreta que un mateix objecte s'ha de comportar de manera diferent davant d'un mateix mètode, depenent del context en què s'ha cridat.

En JavaScript el polimorfisme s'implementa d'una manera senzilla, ja que en el primer cas els mètodes, encara que tinguin el mateix nom en dos objectes diferents, pertanyen als objectes mateixos, de manera que no es crea cap conflicte.

Pel que fa al polimorfisme basat en el comportament dependent del context, s'implementa a partir de la programació d'un comportament diferent en una funció, depenent dels paràmetres que rep la crida.

1.3. Implementació de la programació orientada a objectes

Els llenguatges principals de programació que implementen la programació orientada a objectes (Java, JavaScript, C#, C++, PHP, etc.) no ho fan de la mateixa manera. Cada llenguatge ha adaptat els principis de la POO a la seva pròpia sintaxi i als seus objectius principals.

En el cas que ens concerneix, JavaScript suporta les característiques de la programació orientada a objectes, encara que té algunes particularitats que s'han d'estudiar amb deteniment.

Tot seguit es presentaran els components principals de la programació orientada a objectes i un exemple breu de la implementació que tenen en JavaScript.

1.3.1. La classe en JavaScript

A diferència de Java, C++ i altres llenguatges, JavaScript no té una sentència class que defineixi les classes. En JavaScript, la definició d'una classe es fa amb una simple funció, en què s'incorporen les propietats i els mètodes.

En l'exemple següent es defineix una classe nova anomenada *Cotxe*:

```
//Defineix la classe Cotxe
function Cotxe() { }
```

Evidentment, la classe Cotxe no té encara cap mètode ni cap propietat; es tracta d'una classe buida de contingut.

1.3.2. L'objecte en JavaScript

La creació d'objectes o instàncies de classe s'implementa, en JavaScript, utilitzant la sentència new. Per exemple, en el codi següent es creen dues instàncies de la classe Cotxe que s'assignen a les variables *cotxe1* i *cotxe2*:

```
//Defineix la classe Cotxe
function Cotxe() { }

var cotxe1 = new Cotxe();
var cotxe2 = new Cotxe();
```

Ens trobem dos objectes o dues instàncies de la classe Cotxe que encara no tenen cap propietat o mètode.

1.3.3. El constructor en JavaScript

En el paradigma de la programació orientada a objectes, el mètode constructor s'utilitza per a inicialitzar les propietats de la instància d'una classe (crear l'objecte). És a dir, el constructor és cridat quan la classe és instanciada.

Ara bé, tal com hem avançat, en JavaScript no hi ha la sentència class i la definició de classe es fa amb la definició d'una funció; per tant, no cal definir un constructor d'una manera explícita. La mateixa funció que ha definit la classe actua de constructor pròpiament dit.

Per exemple, seguint el codi anterior:

```
//Defineix la classe Cotxe
function Cotxe() {
        alert("Classe Cotxe instanciada");
}

var cotxe1 = new Cotxe();
var cotxe2 = new Cotxe();
```

En la definició de la funció de la classe s'ha inserit una crida al mètode alert() que avisa que s'ha instanciat la classe Cotxe. En l'exemple s'executa dues vegades: la primera, en la creació de cotxe1, i la segona, en la creació de cotxe2.

1.3.4. Les propietats en JavaScript

Les propietats són variables que formen part de l'objecte i que són les encarregades de definir l'estat de l'objecte com a instància de la classe. Dues instàncies d'una mateixa classe es diferencien pel contingut de les propietats que tenen.

Les propietats d'una classe poden ser de dos **tipus**, depenent de l'àmbit d'influència que tenen:

- **Propietats públiques:** quan són accessibles des de fora de la classe mateixa.

- **Propietats privades:** quan no són accessibles des de fora de la classe.

La implementació de la característica anterior es duu a terme a partir de la mateixa definició de la variable, és a dir, si es defineix en la classe com a *global*, la variable defineix una propietat pública, mentre que si es defineix una *variable local*, s'implementa una propietat privada.

La diferència principal és que les propietats privades només es poden consultar o modificar definint mètodes interns de la classe.

En el mateix codi de les classes, es pot accedir a les propietats utilitzant la sentència this. En l'exemple següent, es pot veure el procés de creació de dues propietats de la classe Cotxe:

```
//Defineix la classe Cotxe
function Cotxe(velocitat) {
        alert("Classe Cotxe instanciada");
        var color = null;
        this.velocitat = velocitat;
}
```

```
var cotxe1 = new Cotxe(40);
var cotxe2 = new Cotxe(60);
```

Si ens fixem en el codi anterior, s'hi aprecia el següent:

- En primer lloc, s'ha introduït un paràmetre d'entrada en la funció que defineix la classe Cotxe; l'objectiu d'aquest paràmetre d'entrada és assignar, en el moment de construir l'objecte, el valor de la propietat velocitat.

- En segon lloc, s'ha introduït al cos de la funció la sentència this.velocitat = velocitat;, que defineix una propietat pública velocitat en la classe i que li assigna el valor que es passa com a paràmetre a la funció quan és cridada. És a dir, és passat a la funció que duu a terme l'objecte. A més, s'ha assignat una propietat privada color que no es pot modificar des de fora de la classe.

- En tercer lloc, s'han creat dues instàncies de la classe Cotxe, en què cotxe1 es diferencia de cotxe2 en la propietat velocitat, ja que la primera s'ha creat amb un valor 40, mentre que la segona s'ha creat amb un valor 60.

1.3.5. Els mètodes en JavaScript

En JavaScript, la definició dels mètodes de la classe és molt simple, ja que s'implementen a partir de la definició de propietats a les quals s'assignen funcions que tenen definides les accions que ha de portar a cap el mètode.

Tot seguit s'introdueix en la classe Cotxe el mètode setColor, que assigna el valor de la propietat color:

```
//Defineix la classe Cotxe
function Cotxe(velocitat) {
        alert("Classe Cotxe Instanciada");
        this.velocitat = velocitat;
        this.setColor = function(color) {
                this.color = color;
                alert("Cotxe: color modificat");
        }
}
var cotxe1 = new Cotxe(40);
var cotxe2 = new Cotxe(60);

cotxe1.setColor("Vermell");
cotxe2.setColor("Blau");
```

Si ens fixem en el codi anterior, s'hi aprecia el següent:

- En primer lloc, s'hi ha introduït una línia nova en què es defineix setColor com una funció amb un paràmetre d'entrada. Aquest paràmetre s'assigna a dins de la funció a la propietat privada color de la classe Cotxe. És a dir, s'ha definit un mètode que permet modificar propietats dels objectes.

- En segon lloc, les dues darreres sentències modifiquen la propietat color dels objectes cotxe1 i cotxe2, als quals s'assigna vermell i blau, respectivament.

En l'exemple anterior es presenta una característica fonamental de la POO: l'encapsulació. Això és així perquè setColor es pot interpretar com el mètode públic de la classe Cotxe que permet modificar la propietat color. D'aquesta manera no es modifica directament la propietat, sinó que es fa d'una manera controlada a partir d'un mètode definit per a això.

1.3.6. L'herència en JavaScript

JavaScript té un mecanisme que permet crear objectes i dur a terme herència entre ells. Aquesta tècnica s'anomena *prototypal inheritance*. Es basa en el fet que un objecte pot heretar mètodes i propietats d'altres objectes utilitzant la propietat prototype.

L'herència de prototip s'implementa amb la propietat prototype que hi ha en tots els objectes del llenguatge, però s'ha de tenir en compte que la propietat només pot heretar altres objectes i no altres prototips o funcions constructores.

L'objecte Prototype

En JavaScript tot objecte té una propietat anomenada *prototype* que permet afegir propietats i mètodes a tots els objectes que s'han creat d'una mateixa classe i a tots els que es creïn després.

Vegem l'ús d'aquesta propietat en l'exemple següent:

```
//Definim la classe Cotxe
function Cotxe() {
}
var cotxe1 = new Cotxe();
Cotxe.prototype.velocitat =120;
var cotxe2 = new Cotxe();
alert(cotxe1.velocitat); // Aquest alert mostra 120
alert(cotxe2.velocitat); // Aquest alert mostra 120
```

Com es pot veure en l'exemple, l'objecte Prototype afegeix la propietat velocitat a tots els objectes de la classe Cotxe; el fet interessant de debò és que afegeix aquesta propietat tant als objectes que es crearan *a posteriori* (per exemple, cotxe2) com als que s'han creat abans (per exemple, cotxe1).

Implementació de l'herència

La possibilitat d'assignar a l'objecte Prototype variables, funcions o objectes és el que permet implementar l'herència. Tot seguit, es presenta un exemple molt senzill que ajuda a comprendre la tècnica:

```
function Vehicle(color) {
       this.rodes = 4;
       this.maximPassatgers = 4;
       this.color = color;
}

function Cotxe(color) {
       this.rodes =4;
       this.maximPassatgers =4;
       this.color = color;
       this.tePortes = true;
}

function Moto(color) {
       this.rodes =2;
       this.maximPassatgers =2;
       this.color = color;
       this.tePortes = false;
}
```

En l'exemple anterior es defineixen tres classes diferents, que tenen una particularitat: les classes Cotxe i Moto tenen en comú propietats amb la classe Vehicle i, a més, afegeixen una propietat: tePortes.

L'herència permet crear les classes Cotxe i Moto a partir de la classe Vehicle, a fi d'evitar la definició de totes les propietats que ja s'han definit i les que són comunes entre les tres classes. Així, la classe Vehicle es considera la classe pare, mentre que les classes Cotxe i Moto es consideren classes filla de la classe Vehicle.

Tot seguit es presenta, amb un exemple, l'ús d'herència per a crear les classes anteriors:

```
//En la classe pare es defineixen totes les propietats comunes
function Vehicle(color) {
       this.rodes =4;
```

```
        this.maximPassatgers =4;
        this.color = color;
}
//En les classes que hereten es canvien els valors de les variables que ho necessitin
function Cotxe(color) {
        this.color = color;
        this.tePortes = true;
}
//Es reemplaça l'objecte Prototype per un objecte Vehicle perquè la classe
// Cotxe adquireixi tots els seus mètodes i propietats
Cotxe.prototype = new Vehicle();

//Ara s'implementa igual amb la classe Moto
function Moto(color) {
        this.rodes =2;
        this.maximPassatgers =2;
        this.color = color;
        this.tePortes = false;
}
Moto.prototype = new Vehicle();
```

Pel que fa al codi anterior, s'ha de tenir en compte el següent:

- Al començament, la propietat prototype no conté propietats ni mètodes, però, quan s'afegeixen a l'objecte Prototype, automàticament s'afegeixen a totes les instàncies de la classe.

- A més, en comptes d'assignar propietats o mètodes a Prototype, es reemplaça aquest objecte per un altre que ja té propietats i mètodes (un objecte de la classe Vehicle) i, per tant, s'afegeixen automàticament totes les propietats a totes les instàncies de la classe nova.

L'herència basada en prototips permet identificar les classes pare quan s'utilitza la propietat **instanceOf**.

Aquesta tècnica per a implementar l'herència té un avantatge sobre les altres, perquè aconsegueix que l'operador instanceOf funcioni com cal, ja que aquest operador permet saber si una instància d'un objecte pertany a una classe determinada.

```
var moto1 = new Moto("vermell");
alert(moto1 instanceof Moto); //Mostra true
alert(moto1 instanceof Cotxe); //Mostra false
alert(moto1 instanceof Vehicle); //També mostra true
```

Com es veu en l'exemple anterior, s'obté que l'objecte que es crea és de la classe Moto, però també és de la classe Vehicle, tal com s'esperaria en qualsevol altre llenguatge de programació orientada a objectes.

1.3.7. Exemples de programació orientada a objectes en JavaScript

Tot seguit es planteja un exemple en què es crea una classe que representa un alumne de la UOC, que disposa de les propietats nom, edat, numMatricula i els mètodes matricula i imprimeix. L'exemple s'acaba creant una instància d'aquesta classe i s'utilitzen els mètodes d'aquesta classe:

```
//Es defineix el mètode matrícula per a la classe AlumneUOC
function matricula(num_matricula){
       this.numMatricula = num_matricula;
}

//Es defineix el mètode imprimeix per a la classe AlumneUOC
function imprimeix(){
        document.write("<br>Nom: " + this.nom);
        document.write("<br>Edat: " + this.edat);
        document.write("<br>Número de matrícula: " + this.numMatricula);
}
//Es defineix el constructor de la classe AlumneUOC
function AlumneUOC(nom, edat){
        this.nom = nom;
        this.edat = edat;
        this.numMatricula = null;
        this.matricula = matricula;
        this.imprimeix = imprimeix;
}

//Es crea una instància
elmeuAlumne = new AlumneUOC("Pau Ferrer",34);

//Es crida el mètode d'imprimir
elmeuAlumne.imprimeix();

//Es crida el mètode de matriculació
elmeuAlumne.matricula(305);

//Es torna a cridar el mètode d'imprimir (amb el número de matrícula seleccionat)
elmeuAlumne.imprimeix();
```

En l'exemple següent es defineix una classe Persona, que defineix una propietat pública, nom, i una de privada, edat. En el cas d'aquesta darrera propietat, es defineixen dos mètodes que permeten modificar i consultar la propietat. L'exemple s'acaba creant una classe nova, Sanitari, que hereta de Persona els mètodes i les propietats.

```
//Definició de la classe persona
function Persona(nom) {
        //Definició de les propietats
        //Publica
                this.nom = nom;
        //Privada
                var edat = null;
};
//Definició dels mètodes
//Mètode que modifica l'edat
Persona.prototype.mEdat = mEdat;
function mEdat(Edat){
        this.edat = Edat;
};
//Mètode que consulta l'edat
Persona.prototype.cEdat = cEdat;
function cEdat(){
       return this.edat;
};

// Es crea un objecte nou de la "classe" Persona
var au = new Persona("Carles");

// Aquesta funció modifica el nom directament, ja que es tracta
// d'una propietat pública, i l'edat a partir del mètode, ja que és
// una propietat privada
function modifica(){
       au.nom = "Pere";
       au.mEdat(22);
};

// Tot seguit, es defineix una classe nova Sanitari a partir de la classe Persona
function Sanitari(categoria) {
       this.estudis = "Sanitat";
       this.categoria = categoria;
}
Sanitari.prototype = new Persona();
```

En el codi anterior s'aconsegueix crear la classe Sanitari a partir de les propietats noves que fan que aquesta classe sigui una especialització de la classe Persona i que hereti totes les propietats i els mètodes definits en la classe Persona. Se'n pot comprovar el funcionament amb el codi següent:

```
san = new Sanitari("Infermer","Vicent");
san.nom="Vicent";
san.mEdat(18);
alert(san.estudis +" "+san.categoria+" "+san.nom+" "+san.cEdat() )
```

2. Crear i usar objectes en JavaScript

En l'etapa anterior s'han presentat les tècniques principals de què es disposa en JavaScript per a crear i usar objectes. En aquest apartat s'aprofundirà en l'estudi d'aquestes tècniques i les variants possibles d'aquestes. Per a això, s'estudiaran abans certes característiques especials del llenguatge Java Script que enriqueixen el procés de crear objectes.

2.1. Crear objectes

Abans que res, cal introduir un seguit de tècniques que proporcionen diferents mecanismes que permeten crear objectes i manipular-los.

2.1.1. Literals de funció

Els literals de funció utilitzen la paraula clau *function*, però sense un nom de funció explícita. Aquest procés es fa servir per a crear mètodes d'objectes que defineix l'usuari, de manera que la funcionalitat del mètode s'insereix en el codi mateix de la funció constructora.

Tot seguit se'n presenta en un exemple:

```
function Automata(nom){
       this.nom = nom;
       this.saluda = function(){alert("Hola, em dic:" + this.nom);};
       this.missatge = function(missatge) {alert(missatge);};
}
```

En el codi anterior, es pot veure el següent:

- En primer lloc, es defineix el mètode saluda amb una funció sense nom que obre una finestra alert() i es mostra el nom de l'objecte que es passarà en la construcció.

- En segon lloc, es defineix el mètode missatge amb una funció que, a més, té un paràmetre d'entrada. Aquest paràmetre s'ha de passar quan es crida el mètode de l'objecte.

Tot seguit es mostra un exemple en què es crea una instància de la classe anterior i es fa una crida als dos mètodes que té:

```
//Es crea una instància que s'emmagatzema en la variable r2d2
var r2d2 = new Automata("S2000");
//Es crida el mètode "saluda", que obre la finestra amb el missatge "Hola,
```

```
//em dic: S2000"
r2d2.saluda();
//Es crida el mètode "missatge", que obre la finestra amb el missatge
//"Hola, Món"
r2d2.missatge("Hola, Món");
```

2.1.2. Literals d'objecte

Els literals d'objecte es construeixen a partir d'una llista de parells propietat/valor separats per comes i tancats entre claus. Els parells s'identifiquen indicant un nom de propietat seguit de dos punts i, després, el valor que té assignat.

Igual que en el cas dels literals de funció, l'ús de literals d'objecte pot simplificar el procés de creació de classes. Tot seguit se'n planteja un exemple:

```
var Automata = {
       nom : "S2000",
       model : "P5+",
       concurrencia : "Sí",
       saluda : function() { alert("Hola, Món");}
}
```

En el codi anterior s'ha creat una classe formada per tres propietats i un mètode, en el qual l'ús de literals d'objecte i de funcions fa que el codi sigui més intuïtiu.

L'exemple següent afegeix certa complexitat al codi anterior:

```
var mod = "P6-";
var Automata = {
       nom: null,
       model: mod,
       concurrencia: "Sí",
       saluda: function() {alert("Hola, Món");},
       pantalla: {
               nom: "Trinitron",
               model: "40 polzades",
               concurrencia: "Sí",
               saluda: function() {alert("Hola, espectador");};
        };
 };
 Automata.nom="S2000";
```

En el codi anterior, s'observen les característiques següents:

- S'ha assignat a la propietat nom el valor null, de manera que quan es crea l'objecte aquesta propietat no té cap valor.

- S'ha assignat a la propietat model el valor d'una variable que s'ha definit abans en la funció constructora.

- S'ha definit una propietat nova, pantalla, que, al seu torn, és un objecte format per les propietats nom, model, concurrencia i la funció saluda.

El codi anterior és ben correcte, però hi ha un seguit de qüestions que es plantegen en interpretar-lo: què passa si es modifica *a posteriori* el valor de la variable *mod*? Varia el valor de la propietat model de l'objecte quan aquest model s'ha creat? El codi que introdueix l'objecte Pantalla, és clar o una mica recarregat?

Un literal d'objecte pot simplificar el procés de definició de classes perquè aporta claredat al codi.

En els subapartats següents es mirarà d'aclarir les qüestions anteriors i s'aprofundirà més en el potencial de JavaScript com a llenguatge de programació orientat a objectes.

2.1.3. Tipus de dades primitives i de referència

Els tipus de dades en JavaScript es classifiquen en aquests dos tipus:

- **Tipus primitius:** són els tipus de dades numèriques, de cadena, lògiques, no definides i nul·les; són primitius en el sentit que es restringeixen a un conjunt de valors definit i es pot pensar que aquests valors estan emmagatzemats en la variable mateixa amb què els manipulem.

- **Tipus de referència:** són objectes que es poden haver creat amb Object, Array, Function, etc. Pel fet de ser objectes poden tenir una bona quantitat de dades heterogènies i, per tant, la variable que inclou un tipus de referència no conté el valor real que té, sinó una referència o un punter al lloc de la memòria en què s'emmagatzema el conjunt de valors.

Tot seguit es presenten dos exemples que mostren la diferència de comportament que tenen els dos tipus de dades:

```
var x =18;
var y = x;
x = 22;
```

En aquest exemple, el valor que adquireix la variable *y* és 18, ja que, com que es tracta d'un tipus de dades primitiu, el valor s'emmagatzema en la mateixa variable *x*. Després, aquest valor s'assigna a la variable *y* en la segona línia de codi (que l'emmagatzema); per tant, no l'afecta l'assignació nova que s'ha fet en la tercera línia. Els valors finals són aquests: $x = 22$ i $y = 18$.

En l'exemple següent, hi ha una cosa que canviarà:

```
var x = [18, 22, 26];
var y = x;
x[0] = 20;
```

La clau d'aquest exemple és que s'ha assignat a la variable *x* un vector, i aquest vector és un tipus per referència, de manera que en la segona línia l'assignació de la variable *y* es fa per referència, és a dir, aquesta variable apunta el lloc de la memòria en què s'emmagatzema el vector.

Per aquest motiu, en la tercera línia de codi es modifica directament en la posició de memòria en què hi ha emmagatzemat el primer element del vector, cosa que implica que, quan s'acabarà el codi, tant el valor de la variable *x* com el de *y* serà el vector [20, 22, 26].

L'efecte que s'aprecia en el codi anterior s'ha de tenir en compte, ja que pot causar errors no esperats. Per exemple, els paràmetres de les funcions en JavaScript són passats per valor, però, si el paràmetre és un tipus per referència, no es passa una còpia del paràmetre sinó la referència, de manera que la modificació al cos de la funció provoca una modificació directa de l'atribut fora de la funció.

En l'exemple següent, s'observa una funció a la qual es passen dos valors: un tipus primitiu i un altre de referència:

```
//Es defineixen dues variables
var vector = ["Alacant", "Albacete", "Almeria"];
var edat =150;

//Es defineix la funció que modificarà les dues variables
function modifica(x,y){
        x[0] = "Àlaba";
        y = 100;
}

//Es crida la funció amb les dues variables definides
modifica(vector, edat);
```

El resultat del codi anterior és que la variable *vector* emmagatzema els valors ["Àlaba", "Albacete", "Almeria"], mentre que la variable *edat* continua amb el valor 150.

2.1.4. Crear objectes utilitzant arrays associatives

Una array associativa és una estructura vectorial en què els elements del vector, en comptes de ser organitzats amb índexs numèrics depenent de la posició que tenen, són organitzats per claus no numèriques.

```
var provincia = new Array();
provincia['u'] = 'Àlaba';
provincia['dos'] = 'Albacete';
provincia['tres'] = 'Alacant';
```

Si es vol recuperar el valor d'Alacant, s'ha de referenciar l'array amb la sintaxi següent:

```
provincia['tres'];
```

JavaScript proporciona arrays associatives pel fet que les dues instruccions següents són equivalents:

```
object.property;
object["property"];
```

de manera que el codi anterior equival al següent:

```
var provincia = new Object();
provincia.u = 'Àlaba';
provincia.dos = 'Albacete';
provincia.tres = 'Alacant';
```

En tots dos casos, es pot accedir a les propietats utilitzant la notació de parèntesis o la notació de punts.

Les arrays associatives s'utilitzen amb freqüència quan els noms de les propietats no se saben fins a l'hora d'execució del codi; per exemple, una funció que sol·licita a l'usuari que hi introdueixi noms de clients i les seves preferències.

L'emmagatzemament de les dades es pot dur a terme amb una estructura del tipus *array associativa* de la manera següent:

```
clients[nom] = preferències;
```

en què nom és una cadena i preferències poden ser des d'una cadena fins a un objecte nou amb una certa estructura.

El problema principal d'aquest tipus de vectors ens el trobem quan fa falta portar-hi a cap una iteració, ja que com que no és organitzat per índexs no es pot utilitzar una estructura repetitiva del tipus for.

Per a aquest tipus d'iteracions, l'estructura més indicada és for/in de JavaScript. Tot seguit, es presenta un exemple d'ús d'aquesta estructura:

```
//Es defineix l'array associativa
var provincia = new Object();
provincia.u = 'Àlaba';
provincia.dos = 'Albacete';
provincia.tres = 'Alacant';
//Es defineix el bucle que recorre cada element de l'array
for (var ordre in provincia) {
        document.writeln("La província número "+ordre+ "és"+provincia[ordre]);
}
```

Amb el codi anterior, cada nom que té dades que hi estan associades s'assigna a *ordre* a cada pas del bucle. Amb el valor d'aquesta variable s'obtenen els valors de l'array en el cos del bucle utilitzant la sentència provincia[ordre].

2.2. Propietats dels objectes

2.2.1. Propietats de les instàncies

Es poden afegir propietats d'una manera dinàmica als objectes. Aquestes propietats són **propietats d'instància**, és a dir, només es modifica la propietat en l'objecte sobre el qual s'actua, no en tots els objectes. Per exemple:

```
var salutacio = new String("Hola, Món");
salutacio.idioma = "Català";
```

En el codi anterior s'ha afegit la propietat idioma a l'objecte salutacio, que és una instància de la classe String. La propietat no s'afegeix a tots els objectes o totes les instàncies de la classe String.

L'eliminació de propietats s'implementa amb la sentència delete de JavaScript. Aquesta sentència es fa servir per a eliminar propietats d'instàncies i també en el cas d'elements d'arrays.

En l'exemple següent s'elimina la propietat idioma de l'objecte anterior:

```
delete salutacio.idioma;
```

Si es consulta el valor de la propietat després d'haver-ne fet l'eliminació, aquesta propietat torna el valor undefined, que indica que ja no hi és.

2.2.2. Propietats de les classes

Tal com s'ha presentat en l'etapa anterior, tots els objectes tenen una propietat prototype que en defineix l'estructura. Es pot emplenar el prototip del constructor amb el codi i les dades que han de compartir totes les instàncies de la classe.

Si es parteix de l'exemple següent plantejat al començament de l'etapa:

```
function Automata(nom){
       this.nom = nom;
       this.saluda = function(){alert("Hola, em dic:" + this.nom);};
       this.missatge = function(missatge) {alert(missatge);};
}
```

es pot traslladar la propietat nom i el mètode saluda al prototip:

```
Automata.prototype.nom = "Sense nom";
Automata.prototype.saluda = function() {alert("Hola, em dic:" + this.nom);};

function Automata(nom){
        if (nom) this.nom = nom;
        this.missatge = function(missatge) {alert(missatge);};
}
```

El funcionament del codi quan un objecte té propietats definides o mètodes en el prototip és el següent:

- Si es mira d'accedir a una propietat o un mètode, de primer l'intèrpret el busca en l'objecte o la instància mateixos de la classe.

- Si no troba la propietat o el mètode a l'objecte, el busca en el prototip de l'objecte.

Si es té en compte la lògica anterior, es proposen dos casos d'ús:

- Es crea una instància de la classe Automata, però sense que hi passi cap paràmetre que defineixi la propietat nom. Si *a posteriori* es prova d'accedir a la propietat nom, com que no la troba en la instància, la va a buscar en el prototip i, per tant, torna el valor Sense nom.

- Si es crea una instància de la classe Automata i s'hi passa el paràmetre que defineix la propietat nom, aquest paràmetre sobreescriu el que s'ha definit en el prototip.

- Quan es crida el mètode saluda, com que no és en la classe, es busca en el prototip de la classe.

Una de les característiques més interessants dels prototips és que són compartits, és a dir, només hi ha una còpia del prototip que fan servir tots els objectes creats amb el mateix constructor. Això implica que si hi ha un canvi en el prototip el podran veure tots els objectes que el comparteixen, i per això els valors predeterminats del prototip són sobreescrits per les variables de les instàncies i no són modificats directament.

Si es canvia el valor d'un prototip, implica el canvi en tots els objectes que comparteixen aquest prototip. L'exemple següent s'aprofita d'aquesta característica:

```
String.prototype.getPrimerCaracter = function(){
        return this.charAt(0);
};
```

En l'exemple anterior s'ha modificat el prototip de la classe String i s'ha definit un mètode nou que permet accedir al primer caràcter de la cadena. Com que es tracta d'un mètode en el prototip, afecta tots els objectes:

```
var cadena = "S2000".getPrimerCaracter();
```

A més de propietats d'instància i de prototip, es poden definir propietats estàtiques o de classe. Com que els constructors són funcions i les funcions són instàncies de l'objecte Function, es poden afegir propietats als constructors:

```
Automata.intelligent = true;
```

La línia anterior ha definit una propietat estàtica de l'objecte Automata quan s'ha definit una variable d'instància al constructor. Aquestes propietats només existeixen en un sol lloc com a membres dels constructors, la qual cosa implica que l'accés s'ha de fer amb el constructor.

Les propietats estàtiques han de contenir dades o codi que no ha de dependre del contingut de cap instància en particular. Per exemple, el mètode toLowerCase() de l'objecte String no pot ser un mètode estàtic perquè la cadena que torna depèn de l'objecte des del qual s'ha cridat. Tanmateix, la propietat PI de l'objecte Math sí que pot ser estàtica, perquè no depèn de cap instància específica.

2.3. Encadenament de prototips

L'herència en JavaScript s'aconsegueix amb els prototips; les instàncies d'un objecte hereten el codi i les dades definides en el prototip del constructor. També es pot obtenir un tipus d'objecte nou, però, a partir d'un tipus que ja hi és, quan hereta les propietats de l'ascendent i hi afegeix propietats noves. L'exemple següent mostra aquesta tècnica:

```
function Robot(tipologia){
       if (tipologia){
                 this.tipologia = tipologia;
       }
}
Robot.prototype = new Automata();
Robot.prototype.tipologia = "Cuina";
```

El concepte que s'introdueix en l'exemple és l'establiment del prototip Robot a una instància nova d'un objecte Automata. D'aquesta manera, els objectes Robot contenen tant les propietats o els mètodes de la classe Robot com els de la classe Automata.

L'accés a les propietats es resol de la mateixa manera que s'ha presentat abans: les propietats d'instància de l'objecte es revisen de primer per a cercar-hi una coincidència; després, si no se'n troba cap, es busca en el prototip. Si tampoc no se n'hi troba cap, es comprova el prototip pare. Aquest procés es continua repetint de manera recursiva.

El procés anterior explica el motiu que tots els objectes tinguin certes propietats, com toString(). Aquest mètode és una propietat del prototip Object i, com que tots els objectes s'obtenen d'Object, el fet de cridar toString() a qualsevol objecte obre finalment el camí per encadenament d'herència fins a la propietat toString() d'Object (si no ha estat sobreescrita en alguna classe intermèdia).

2.4. Destrucció d'objectes

Quan es creen objectes, s'assigna d'una manera automàtica un espai en la memòria per a emmagatzemar-los i es passa una referència de l'objecte nou al constructor que s'ha cridat. Però no solament s'assigna la memòria, sinó que també s'allibera la memòria que no s'utilitzarà *a posteriori*.

Així, doncs, hi ha una vigilància sobre les dades, de manera que quan un conjunt de dades ja no és accessible al programa, l'espai que ocupa és recuperat o és alliberat per a assignacions futures a altres objectes.

JavaScript, igual que altres llenguatges de programació com Java, utilitza el que es coneix com a *recuperador de memòria* (*garbage collector*) per a eliminar memòria no utilitzada.

Es pot facilitar o accelerar aquesta feina assignant el valor null a cadascuna de les referències de l'objecte que es vol alliberar (sempre que quedi una referència que apunta l'objecte, no s'allibera la memòria). D'aquesta manera, el recuperador de memòria detecta que l'objecte no s'utilitzarà i marca l'espai que s'ha fet servir com a lliure.

3. Objectes predefinits

Tal com s'ha dit abans, en JavaScript hi ha un conjunt d'objectes incorporats que permet accedir a moltes de les funcions que hi ha en qualsevol altre llenguatge de programació. Ens referim als objectes Object, Array, Boolean, Date, Function, Math, Number, String i RegExp.

En aquest mòdul es presentaran les propietats que es fan servir més i que suporten comunament els navegadors principals.

Web recomanat

Es pot consultar en línia l'especificació completa dels objectes predefinits en l'estàndard ECMA-262.

3.1. Els objectes Object i Function

3.1.1. L'objecte Object

Es tracta de l'objecte pare o avi a partir de qual hereten tots els objectes que hi ha en el llenguatge o que es crearan en el llenguatge. Així, aquest objecte defineix les propietats i els mètodes que són comuns a tots els objectes, de manera que cada objecte particular podrà reescriure mètodes o propietats si ho necessita per adequar-lo a l'objectiu que té.

Un ús de l'objecte, encara que no és gaire comú o recomanable, és com a mètode de creació alternatiu d'objectes. Per exemple:

```
var cotxe = new Object();
cotxe.marca = "Ford";
cotxe.model = "Focus";
cotxe.aireAcon = true;
cotxe.mostra = function()
               {
               alert(this.marca +" "+this.model);
               }
```

En l'exemple anterior s'ha creat un objecte cotxe a partir d'Object.

Una altra de les estructures que es generen amb l'ús d'Object són les arrays associatives. A diferència de les bàsiques, en les associatives cada element de l'array es referencia pel nom que s'hi ha assignat i no amb l'índex que n'indica la posició.

Vegeu també

Les arrays associatives s'han tractat en l'apartat 2 d'aquest mòdul.

Tot seguit es presenta un exemple de l'ús d'aquestes arrays:

```
var adreces = new Object();
adreces["Víctor"] = "Santiago de Cuba";
```

```
adreces["Pablo"] = "Madrid";
adreces["Miquel"] = "València";
```

D'aquesta manera, es pot recuperar l'adreça de Miquel utilitzant la sintaxi següent:

```
var adrMiq = adreces["Miquel"];
```

Per tant, com es pot veure, simplement se simula una array amb l'assignació dinàmica de propietats a un objecte del tipus Object.

Taula 1. Propietats de l'objecte Object

Nom	Descripció
constructor	Referència a la funció que s'ha cridat per a crear l'objecte genèric.
prototype	Representa el prototip per a la classe.

Taula 2. Mètodes de l'objecte Object

Nom	Descripció
toString()	Torna a l'objecte una cadena, de manera predeterminada "[objecte Object]".
valueOf()	Torna el valor primitiu associat amb l'objecte, de manera predeterminada "[objecte Object]".

3.1.2. L'objecte Function

És l'objecte de què deriven les funcions de JavaScript; proporciona propietats que transmeten informació útil durant l'execució de la funció. Un exemple d'aquestes propietats és l'array arguments[ ].

El constructor per a l'objecte Function és aquest:

```
new Function (arg1, arg2..., argN, funció)
```

en què:

- arg1, arg2,..., argN: paràmetres opcionals, de la funció.
- funció: és una cadena que conté les sentències que componen la funció.

La propietat arguments [ ] permet saber el nombre d'arguments que s'han passat en la crida a la funció.

En l'exemple següent, es crea una funció, a dins de la qual hi ha un bucle for l'extrem superior del qual el defineixen el nombre d'arguments de la funció mateixa, ja que l'objectiu del bucle és manipular els arguments que s'han passat en la crida a la funció, però dels quals *a priori* no se sap el nombre.

```
function llista(tipus) {
        document.write("<" + tipus + "l>");
        for (var i=1; i<llista.arguments.length; i++) {
                document.write("<li>" + llista.arguments[i]);
                document.write("</" + tipus + "l>");
        }
}
```

La crida a la funció amb els arguments següents:

```
llista("u", "U", "Dos", "Tres");
```

té com a resultat el següent:

```
<ul>
<li>u
<li>Dos
<li>Tres
</ul>
```

Taula 3. Propietats de l'objecte function

Nom	Descripció
arguments	Array amb els paràmetres que s'han passat a la funció.
caller	Nom de la funció que ha cridat la que s'executa.
constructor	Referència a la funció que s'ha cridat per a crear l'objecte.
length	Nombre de paràmetres que s'han passat a la funció.
prototype	Valor a partir del qual es creen les instàncies d'una classe.

La propietat caller

Aquesta propietat només és accessible des del cos de la funció. Si es fa servir fora de la funció mateixa, el valor és null.

Taula 4. Mètodes de l'objecte function

Nom	Descripció	Sintaxi	Paràmetres
apply	Crida la funció.	apply(obj[,arg1,arg2...argN])	obj: nom de la funció arg1,..., argN: llista d'arguments de la funció
call	Crida la funció.	call(obj[,args])	obj: nom de la funció args: array d'arguments de la funció
toString	Retorna una string que representa l'objecte especificat.	toString()	
valueOf	Retorna el valor primitiu associat a l'objecte.	valueOf()	

3.2. Els objectes Array, Boolean i Date

3.2.1. L'objecte Array

Les arrays emmagatzemen llistes ordenades de dades heterogènies. Les dades s'emmagatzemen en índexs enumerats començant des de zero, als quals s'accedeix utilitzant l'operand d'accés a arrays ([ ]).

Crear arrays

El constructor de l'objecte admet les sintaxis següents:

```
var vector1 = new Array();
var vector2 = new Array(longitud);
var vector3 = new Array(element0, element1,..., elementN);
```

La primera sintaxi del constructor crea una array buida sense una dimensió definida; en la segona sintaxi, s'hi passa la grandària de l'array com a paràmetre, i en el tercer constructor, s'hi passen els valors que s'emmagatzemen en l'array.

Per exemple, es pot definir la longitud de l'array i després emplenar cadascuna de les posicions que té:

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

o bé començar l'array en el moment mateix de crear-la:

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

També hi ha la possibilitat de crear arrays utilitzant literals d'array. Per exemple:

```
var vector1 = [ ];
var vector2 = [,,,,,,,,,,,,];
var vector3 = ["element0", "element1",..., "elementN"];
```

Accedir als elements

L'accés als elements d'una array es fa utilitzant el nom de l'array seguit de l'índex de l'element que s'ha de consultar tancat entre claudàtors. Sobre el valor de l'índex s'han de tenir en compte els punts següents:

- Les arrays s'indexen a partir del valor zero, de manera que el primer valor de l'array té l'índex 0.

- Si es consulta un element de l'array que no s'ha assignat, l'array torna el valor undefined.

Tot seguit es mostra un exemple d'accés als elements d'una array:

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

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

Afegir i modificar elements a una array

A diferència d'altres llenguatges de programació, en JavaScript no fa falta augmentar la memòria de manera explícita si s'augmenta la grandària de l'array. La grandària la gestiona directament el llenguatge i, per tant, se simplifica la feina del programador.

D'aquesta manera, en el codi següent:

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

s'afegeix un element a l'array vector1 i, com es veu, no s'ha fet de manera consecutiva, és a dir, es poden deixar espais buits en una array si fa falta.

En modificar o afegir elements a una array, s'ha de tenir en compte que les arrays són objectes o tipus per referència, de manera que si s'ha assignat una array a dues variables, la modificació d'un element de l'array afecta les dues variables i no solament aquella des de la qual s'ha modificat:

```
var vector2 = [2, 4, 6, 8];
var vector3 = vector2;
vector3[0] =1;
```

L'assignació del valor 1 a l'array vector3 no solament modifica el contingut d'aquesta array sinó també el de l'array vector2.

Eliminar elements d'una array

Els elements d'una array es poden eliminar utilitzant la sentència delete:

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

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

La propietat length

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

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

Si bé es podria esperar que length tornés el valor 1, tal com s'ha explicat, length torna el valor 51, ja que aquest és el primer índex lliure després del darrer.

Aquest comportament de la propietat length fa que sigui poc recomanable assignar buits en els elements d'una array, ja que en aquests casos length no representa el nombre d'elements reals.

A més de la informació que proporciona la propietat length, es tracta d'un mecanisme realment interessant d'eliminació d'elements d'una array, perquè qualsevol índex que conté un valor més gran del que s'ha assignat a length s'estableix a undefined. D'aquesta manera, l'eliminació de tots els valors d'una array es pot fer assignant a la propietat length el valor zero.

El mètode sort

El mètode sort requereix que s'indiqui la funció de comparació per a ordenar l'array. Si no s'especifica aquesta funció, els elements de l'array es converteixen en strings i s'ordenen alfabèticament. Per exemple, Barcelona aniria abans que Saragossa, i 80 abans que 9.

En el cas de no especificar la funció de comparació, els elements s'ordenen segons el retorn d'aquesta funció:

- Si compara(a,b) és més petit que 0, b té un índex en l'array més petit que a.
- Si compara(a,b) és 0, no modifica les posicions.
- Si compara(a,b) és més gran que 0, b té un índex en l'array més gran que a.

La funció de comparació té la forma següent:

```
function compara(a, b) {
       if (a < b per un criteri d'ordenació)
                return -1;
```

```
        if (a > b per un criteri d'ordenació)
                return 1;
        return 0; //són iguals
}
```

El criteri d'ordenació definirà si l'ordenació serà numèrica, alfanumèrica o qualsevol que el programador pugui definir.

Per a ordenar números, la funció es defineix de la manera següent:

```
function compara(a, b){
        if (a < b)
                return -1;
        else if (a === b)
                return 0;
        else
                return 1;
}
```

En una array amb els valors 1, 2 i 11, si s'aplica el mètode sort() sense modificar la funció de comparació, el resultat de l'ordenació és 1, 11, 2, és a dir, s'ordenen els valors alfabèticament.

En l'exemple següent es mostra com s'ha de fer per a modificar la funció de comparació:

```
numeros = new Array(1, 2, 11);
function compara(a, b){
        if (a < b)
                return -1;
        else if (a === b)
                return 0;
        else
                return 1;
}

function ordena(){
        numeros.sort(compara);
        document.write(numeros);
}
```

El resultat d'aquest exemple és el següent: 1, 2, 11.

Simulació de piles LIFO

Una pila LIFO és una estructura que s'utilitza per a emmagatzemar dades seguint l'ordre de darrer d'entrar, primer de sortir. L'analogia és una pila de documents sobre una taula de manera que el primer que retirem per estudiar coincideix amb el darrer que posem a la taula.

Per a simular l'ús de piles amb arrays, s'utilitzen els mètodes push() i pop(); quan es crida el mètode push(), s'afegeixen els arguments donats al final de l'array i s'incrementa la grandària d'aquesta array, que es reflecteix en la propietat length.

El mètode pop() elimina el darrer element de l'array, el torna i disminueix la propietat length en una unitat.

En l'exemple següent es veuen aquests mètodes en acció:

```
var pila = [ ];          //[ ]
pila.push("primer");   //["primer"]
pila.push(15, 30);     //["primer",15, 30]
pila.pop();            //["primer",15] i torna el valor 30
```

El fet que aquests dos mètodes es facin servir per a simular piles no implica que no es puguin utilitzar els mètodes per a inserir i eliminar valors situats al final de l'array.

Simular cues FIFO

Una cua FIFO es basa en l'ordre de primer d'entrar, primer de sortir. L'analogia és, per exemple, qualsevol cua per a pagar en un comerç o a l'entrada del cinema. JavaScript disposa dels mètodes unshift() i shift(), que, a diferència dels dos anteriors, afegeixen i eliminen dades del començament d'array.

D'aquesta manera, unshift() introdueix els arguments al principi de l'array i fa que els elements que hi ha canviïn a índexs més alts, de manera que, com és d'esperar, augmenta el valor de la propietat length.

El mètode shift(), per la seva banda, elimina el primer element de l'array, el torna i redueix l'índex de la resta d'elements de l'array, i acaba amb la disminució obligatòria de la propietat length.

En l'exemple següent se'n veu el comportament:

```
var cua = ["Joan", "Pere", "Andreu", "Vicenç"];
cua.unshift("Clàudia", "Raquel");    //cua contindrà ["Clàudia", "Raquel", "Joan",
                                     //"Pere", "Andreu", "Vicenç"]
var primer = cua.shift();            //primer contindrà el valor "Clàudia"
```

La simulació d'una cua es fa combinant els mètodes push(), que afegeix elements al final de la cua, i shift(), que extreu el primer element d'aquesta cua:

```
var cua = ["Joan", "Pere", "Andreu", "Vicenç"];
cua.unshift("Clàudia");   //cua contindrà ["Clàudia", "Joan",
                      //"Pere", "Andreu", "Vicenç"]
var seguent = cua.pop();             //seguent contindrà el valor "Vicenç"
```

Concatenar arrays

El mètode concat() torna l'array resultat d'afegir els arguments a l'array sobre la qual s'ha fet la crida. Per exemple:

```
var color = ["Vermell", "Verd", "Blau"];
var colorAmp = color.concat("Blanc", "Negre");
```

En l'exemple anterior s'obté una array colorAmp nova amb el contingut següent: ["Vermell", "Verd", "Blau", "Blanc", "Negre"]. L'array color, en canvi, no s'ha modificat.

Es poden concatenar dues arrays simplement si es compleix la condició que l'argument del mètode concat() sigui una array.

Conversió en una cadena

Per a convertir una array en una cadena, s'utilitza el mètode join(), que a més de la conversió permet especificar per mitjà del paràmetre que té com se separaren els elements de la cadena. Es fa servir join() per a mostrar els elements d'una array amb un separador específic:

```
var color = ["Vermell", "Verd", "Blau"];
var cadena = color.join("-");
```

En l'exemple anterior, la variable *cadena* adquireix el contingut següent: Vermell-Verd-Blau.

Invertir l'ordre dels elements

El mètode reverse() inverteix l'ordre dels elements d'una array i, a diferència dels dos mètodes anteriors, la mateixa array emmagatzema els elements, amb l'ordre invertit:

```
var color = ["Vermell", "Verd", "Blau"];
color.reverse();
```

En l'exemple anterior l'array color, després d'executar el mètode, té el contingut següent: ["Blau", "Verd", "Vermell"].

Extreure fragments d'una array

El mètode slice() torna un fragment de l'array sobre la qual es crida; no actua realment sobre l'array (tal com fa reverse()), i queda intacta. El mètode agafa dos arguments, l'índex inicial i el final, i torna una array que conté els elements que hi ha entre l'índex inicial i el final (exclòs el final).

En cas que només rebi un argument, el mètode torna l'array que componen tots els elements des de l'índex indicat fins al final de l'array.

Una característica interessant és el fet que admet valors negatius per als índexs i, quan aquests índexs són negatius, es compten des del final de l'array. En els exemples següents es veu l'ús d'aquest mètode:

```
var carrera = [21, 25, 12, 23];
carrera.slice(2);      //torna [12, 23]
carrera.slice(1,3);    //torna [25, 12]
carrera.slice(-2, -1)  //torna [12]
```

Afegir, eliminar i modificar elements d'una array

El mètode splice() es pot utilitzar per a afegir, reemplaçar o eliminar elements en una array i torna els elements que s'han eliminat. Dels arguments que pot agafar, l'únic que és obligatori és el primer, la sintaxi del qual és la següent:

```
splice(inici, elementsEsborrar, elementsAfegir);
```

El significat dels arguments és el següent:

- inici: indica l'índex a partir del qual es fa l'operació.
- elementsEsborrar: nombre d'elements que s'eliminen, començant pel que marca el primer paràmetre. Si s'omet aquest paràmetre, s'eliminen tots els elements des del començament fins al final de l'array i, tal com s'ha dit, són tornats (es poden emmagatzemar en una variable).
- elementsAfegir: llista d'elements separats per comes, no obligatòria, que substitueixen els eliminats.

Tot seguit, se'n mostra un exemple d'ús:

```
var carrera = [21, 25, 12, 23];
carrera.splice(2, 2);  //torna els elements eliminats [12, 23] i l'array
                       //contindrà els valors [21, 25]
```

```
carrera.splice(2, 0, 31, 33);  //no elimina cap valor i per tant torna [ ]
                               //i afegeix els valors [31, 33] a la cadena
```

Arrays multidimensionals

Una array multidimensional és una array en què cada element és, al seu torn, una array. En l'exemple següent es defineix una array bidimensional:

```
var matriu = [[1,2,3],[4,5,6],[7,8,9]];
```

L'accés a les arrays multidimensionals es fa a partir de la concatenació de claudàtors que indiquen els elements a què s'accedeix:

```
matriu[0][1];   //torna el valor 2, ja que accedim al segon
                //element de la primera array
```

Ús de prototips

Tal com s'ha explicat, es poden afegir mètodes i propietats nous a qualsevol objecte utilitzant els prototips. Tot seguit es defineix un mètode mostra() nou, que ensenya en una finestra el contingut d'una array:

```
function mMostra(){
       if (this.length != 0)
                 alert(this.join());
       else
                 alert("L'array és buida");
}
Array.prototype.mostra = mMostra;
```

Per acabar l'objecte Array, es mostra en una taula les propietats i mètodes principals de l'objecte.

Taula 5. Propietats de l'objecte Array

Nom	Descripció
constructor	Referència a la funció que s'ha cridat per a crear l'array actual.
length	És el nombre d'elements de l'array.
prototype	Representa el prototip per a la classe.

Taula 6. Mètodes de l'objecte Array

Nom	Descripció	Sintaxi	Paràmetres	Retorn
concat	Concatena dues arrays.	concat(array2)	array2: nom de l'array que s'ha de concatenar a la que ha cridat el mètode.	Array nova, unió de totes dues.

Nom	Descripció	Sintaxi	Paràmetres	Retorn
join	Uneix tots els elements de l'array en una string, separats pel símbol indicat.	join(separador)	separador: signe que separa els elements de l'string.	String.
pop	Esborra el darrer element de l'array.	pop()		L'element esborrat.
push	Afegeix un element o més d'un al final de l'array.	push(elt1,..., eltN)	elt1,..., eltN: elements que s'han d'afegir.	El darrer element afegit.
reverse	Trasllada els elements de l'array.	reverse()		
shift	Elimina el primer element de l'array.	shift()		L'element eliminat.
slice	Extreu una part de l'array.	slice(inici,final)	inici: índex inicial de l'array que s'ha d'extreure. final: índex final de l'array que s'ha d'extreure.	Una array nova amb els elements extrets.
splice	Canvia el contingut d'una array, de manera que hi afegeix elements nous i, alhora, elimina els que ja hi havia.	splice(índex, quants, nouEl1,..., nouElN)	índex: índex inicial a partir del qual es comença a canviar. quants: nombre d'elements que s'han d'eliminar. nouEl1,..., nouElN: elements que s'han d'afegir.	Array d'elements eliminats.
sort	Ordena els elements de l'array.	sort(compara)	compara: funció que defineix l'ordre dels elements.	
toString	Converteix els elements de l'array a text.	toString()		String que conté els elements de l'array passats a text.
unshift	Afegeix un element o més d'un al començament de l'array.	unshift(elt1,..., eltN)	elt1,..., eltN: elements que s'han d'afegir.	La longitud nova de l'array.

3.2.2. L'objecte Boolean

És l'objecte incorporat en el llenguatge que representa les dades de tipus lògic. Es tracta d'un objecte molt simple, ja que no disposa de propietats ni mètodes exceptuant els que hereta de l'objecte Object. El constructor de l'objecte és el següent:

```
new Boolean(valor)
```

Si el paràmetre s'omet, o té els valors 0, null o false, l'objecte agafa el valor inicial com a false. En qualsevol altre cas, agafa el valor true. Aquestes característiques fan que aquest objecte es pugui usar per a convertir un valor no booleà a booleà.

Taula 7. Propietats de l'objecte Boolean

Nom	Descripció
constructor	Referència a la funció que s'ha cridat per a crear l'objecte.

Nom	Descripció
prototype	Representa el prototip per a la classe.

Taula 8. Mètodes de l'objecte Boolean

Nom	Descripció	Sintaxi	Retorn
toString	Representa un objecte mitjançant una string.	toString()	"true" o "false" segons el valor de l'objecte.
valueOf	Obtenir el valor que té l'objecte.	valueOf()	string "true" o false" segons el valor que té.

3.2.3. L'objecte Date

L'objecte Date proporciona una varietat extensa de mètodes que permeten manipular dates i hores. L'objecte, però, no conté un rellotge en funcionament, sinó un valor de data estàtic que, a més, internament s'emmagatzema com el nombre de mil·lisegons des de les dotze de la nit de l'1 de gener de 1970.

El constructor de l'objecte pot rebre un ventall interessant de paràmetres. Tot seguit es presenten els que es fan servir més:

```
new Date()
new Date(any_num, mes_num, dia_num)
new Date(any_num, mes_num, dia_num, hora_num, min_num, seg_num)
```

El significat dels paràmetres és el següent:

any_num, mes_num, dia_num, hora_num, min_num, seg_num: són enters que formen part de la data. En aquest cas, s'ha de tenir en compte que el mes 0 correspon al gener, i el mes 11, al desembre.

Treballar amb dates

En l'exemple següent, es pot veure l'ús de les funcions que permeten mostrar la data actual en la pàgina del navegador:

```
mesos = new Array("Gener", "Febrer", "Març", "Abril", "Maig", "Juny", "Juliol",
"Agost", "Setembre", "Octubre", "Novembre", "Desembre");
var data = new Date();
var mes = data.getMonth();
var any = data.getFullYear();
document.write("Avui és" + data.getDate() + "de" + mesos[mes] + "de" + any);
```

Tal com es veurà tot seguit, els objectes Date disposen d'un conjunt molt ampli de mètodes que permeten establir o llegir una propietat directament de l'objecte fent internament les conversions que facin falta, ja que, tal com s'ha indicat abans, l'objecte emmagatzema els valors en mil·lisegons.

Taula 9. Propietats de l'objecte Date

Nom	Descripció
constructor	Referència a la funció que s'ha cridat per a crear l'objecte.
prototype	Representa el prototip per a la classe. És de lectura i prou.

Taula 10. Mètodes de l'objecte Date

Nom	Descripció	Sintaxi	Paràmetres	Retorn
getDate	Retorna el dia del mes per a una data.	getDate()		Enter entre 1 i 31.
getDay	Retorna el dia de la setmana per a una data.	getDay()		Enter entre 0 i 6. El 0 és diumenge, l'1 és dilluns, etc.
getHours	Retorna l'hora per a una data.	getHours()		Enter entre 0 i 23.
getMinutes	Retorna els minuts per a una data.	getMinutes()		Enter entre 0 i 59.
getMonth	Retorna el mes per a una data.	getMonth()		Enter entre 0 i 11.
getSeconds	Retorna els segons per a una data.	getSeconds()		Enter entre 0 i 59.
getTime	Retorna un nombre que correspon al temps transcorregut per a una data.	getTime()		Nombre de mil·lisegons transcorreguts des de l'1 de gener de 1970 a les 00.00.00 hores.
getTimezoneOffset	Retorna la diferència horària, entre l'hora local i l'hora GMT (*Greenwich mean time*).	getTimezoneOffset()		Nombre de minuts que marca la diferència horària.
getFullYear	Retorna l'any per a una data.	getFullYear()		Retorna tots quatre dígits.
parse	String que conté els mil·lisegons transcorreguts per a la data.	Date.parse(dataString)	dataString: data en format string.	Mil·lisegons transcorreguts des de l'1 de gener de 1970 a les 00.00.00 hores.
setDate	Posa el dia a una data.	setDate(valordia)	valordia: enter entre 1 i 31 que representa el dia.	
setDay	Posa el dia a una data.	setDate(valordia)	valordia: enter entre 1 i 31 que representa el dia.	
setHours	Posa l'hora a una data.	setHours(valorhora)	valorhora: enter entre 0 i 23 que representa l'hora.	
setMinutes	Posa els minuts a una data.	setMinutes(valorminuts)	valorminuts: enter entre 0 i 59 que representa els minuts.	

Nom	Descripció	Sintaxi	Paràmetres	Retorn
setMonth	Posa el mes a una data.	setMonth(valormes)	valormes: enter entre 0 i 11 que representa el mes.	
setSeconds	Posa els segons a una data.	setSeconds(valorsegons)	valorsegons: enter entre 0 i 59 que representa els segons.	
setTime	Posa el valor a una data.	setTime(valorhorari)	valorhorari: mil·lisegons transcorreguts des de l'1 de gener de 1970 a les 00.00.00 hores.	
setYear	Posa l'any a una data.	setYear(valorany)	valorany: enter que representa l'any.	
toGMTString	Converteix una data a string, fent servir les convencions GMT d'Internet.	toGMTString()		
toLocaleString	Converteix una data a string, fent servir les convencions locals. Vegeu retorn.	toLocaleString()		Exemple: dissabte, 07 de desembre de 2002 21.22.59.
toLocaleDateString	Converteix una data a string, fent servir les convencions locals. Vegeu retorn.	toLocaleDateString()		Exemple: Sat Dec 7 21:22:59 UTC +0100 2002.
toLocaleTimeString	Converteix una data a string, fent servir les convencions locals.			
UTC	Mil·lisegons transcorreguts per a la data indicada.	Date.UTC(any, mes, dia, hores, min, s)	Enters. Els tres darrers són opcionals.	Mil·lisegons transcorreguts des de l'1 de gener de 1970 a les 00.00.00 hores.

3.3. Els objectes Math, Number i String

3.3.1. L'objecte Math

L'objecte Math és format per un conjunt de constants i mètodes que permeten fer operacions matemàtiques de certa complexitat. Aquest objecte és estàtic, de manera que no es pot crear una instància i l'accés a les constants i els mètodes que té es fa directament utilitzant l'objecte mateix.

L'exemple següent mostra un conjunt de càlculs que s'han fet amb l'objecte Math:

```
function calcula(nombre){
    cosValue = Math.cos(nombre); //Emmagatzema a cosValue el valor del cosinus
    sinValue = Math.sin(nombre); //Emmagatzema a sinValue el valor del sinus
    tanValue = Math.tan(nombre); //Emmagatzema a cosValue el valor de la tangent
    sqrtValue = Math.sqrt(nombre); //Emmagatzema a sqrtValue el valor de l'arrel
    powValue = Math.pow(nombre,3); //Emmagatzema a powValue el valor d'elevar a 3 el valor
    expValue = Math.exp(nombre); //Emmagatzema a expValue el valor de e elevat al valor del nombre
```

```
}
```

Taula 11. Propietats de l'objecte Math

Nom	Descripció	Exemple

E	Constant d'Euler. El valor aproximat que té és 2,718. És només de lectura.	function valorE() {
		return **Math.E**
		}

LN10	Logaritme neperià de 10. El valor aproximat que té és 2,302. És només de lectura.	function valorLN10() {
		return **Math.LN10**
		}

LN2	Logaritme neperià de 2. El valor aproximat que té és 0,693. És només de lectura.	function valorLN2() {
		return **Math.LN2**
		}

LOG10E	Logaritme en base 10 del nombre E. El valor aproximat que té és 0,434. És només de lectura.	function valorLog10e() {
		return **Math.LOG10E**
		}

LOG2E	Logaritme en base 2 del nombre E. El valor aproximat que té és 1,442. És només de lectura.	function valorLog2e() {
		return **Math.LOG2E**
		}

PI	Nombre pi. El valor aproximat que té és 3,1415. És només de lectura.	function valorPi() {
		return **Math.PI**
		}

SQRT1_2	1 dividit per l'arrel quadrada de 2. El valor aproximat que té és 0,707. És només de lectura.	function valorSQRT1_2() {
		return **Math.SQRT1_2**
		}

SQRT2	Arrel quadrada de 2. El valor aproximat que té és 1,414. És només de lectura.	function valorSQRT2() {
		return **Math.SQRT2**
		}

Taula 12. Mètodes de l'objecte Math

Nom	Descripció	Sintaxi	Paràmetres	Retorn
abs	Calcula el valor absolut d'un nombre.	abs(x)	x: un nombre.	Valor absolut de *x*.
acos	Calcula l'arc cosinus d'un nombre.	acos(x)	x: un nombre.	Valor de l'arc cosinus de *x* en radiants. Valor entre [0, PI].
asin	Calcula l'arc sinus d'un nombre.	asin(x)	x: un nombre.	Valor de l'arc sinus de *x* en radiants. Valor entre [-PI/2,PI/2].
atan	Calcula l'arc tangent d'un nombre.	atan(x)	x: un nombre.	Valor de l'arc tangent de *x* en radiants. Valor entre [-PI/2,PI/2].
atan2	Calcula l'arc tangent del quocient de dos nombres.	atan2(y,x)	y: un nombre (coordenada y). x: un nombre (coordenada x).	
ceil	Retorna, més gran o igual que un nombre, l'enter més petit.	ceil(x)	x: un nombre.	El primer enter més gran o igual que *x* (per a 9,8 en retorna 10).
cos	Calcula el cosinus d'un nombre.	cos(x)	x: un nombre.	Valor del cosinus de *x*. Valor entre [-1,1].
exp	Calcula el valor del nombre E elevat a un nombre.	exp(x)	x: un nombre.	Valor de E^x.
floor	Retorna, més petit o igual que un nombre, l'enter més gran.	floor(x)	x: un nombre.	El primer enter més petit o igual que *x* (per a 9,8 en retorna 9).

Nom	Descripció	Sintaxi	Paràmetres	Retorn
log	Calcula el logaritme en base E d'un nombre.	log(x)	x: un nombre.	El logaritme en base E de *x*.
max	Retorna el més gran de dos nombres.	max(x,y)	x: un nombre. y: un nombre.	Si *x* > *y*, retorna *x*. Si *y* > *x*, retorna *y*.
min	Retorna el més petit de dos nombres.	min(x,y)	x: un nombre. y: un nombre.	Si *x* > *y*, retorna y. Si *y* > *x*, retorna *x*.
pow	Calcula la potència entre dos nombres.	pow(x,y)	x: un nombre que representa la base. y: un nombre que representa l'exponent.	Valor de x^y.
random	Retorna un nombre aleatori entre 0 i 1.	random()		Nombre aleatori entre [0,1].
round	Arrodoneix un nombre a l'enter més proper.	round(x)	x: un nombre.	El valor arrodonit de *x*.
sin	Calcula el sinus d'un nombre.	sin(x)	x: un nombre.	Valor del sinus de *x*. Valor entre [-1,1].
sqrt	Calcula l'arrel quadrada d'un nombre.	sqrt(x)	x: un nombre.	Arrel quadrada de *x*.
tan	Calcula la tangent d'un nombre.	tan(x)	x: un nombre.	Valor de la tangent de *x*.

3.3.2. L'objecte Number

Aquest objecte es fa servir sobretot per a accedir a mètodes de formatació de nombres, a més de proporcionar propietats estàtiques que defineixen constants numèriques. El constructor de l'objecte Number és el següent:

```
new Number (valor)
```

en què valor és el valor numèric que hi haurà en l'objecte creat. Si el paràmetre és omès, la instància s'inicialitza amb el valor 0.

Taula 13. Propietats de l'objecte Number

Nom	Descripció	Exemple d'ús
Constructor	Referència a la funció que s'ha cridat per a crear l'objecte.	

MAX_VALUE	És el valor numèric més gran que es pot representar en JavaScript. És només de lectura.	if (nombre < =
		Number.MAX_VALUE)
		lamevaFuncio(nombre)
		else
		alert("nombre massa gran")

MIN_VALUE	És el valor numèric més petit que es pot representar en JavaScript. És només de lectura.	if (nombre >=
		Number.MIN_VALUE)
		lamevaFuncio(nombre)
		else
		lamevaFuncio(0)

NaN	Indica que no es tracta d'un valor numèric (Not-A-Number). És només de lectura.	
NEGATIVE_INFINITY	Representa el valor negatiu infinit. És només de lectura.	

Nom	Descripció	Exemple d'ús
POSITIVE_INFINITY	Representa el valor positiu infinit. És només de lectura.	
Prototype	Representa el prototip per a la classe.	

Valors de l'objecte Number

- El nombre més gran que es pot representar en JavaScript és 1.79E+308. Els valors més grans que aquest adquireixen el valor Infinity.
- El nombre més petit que es pot representar en JavaScript és 5E-324. La propietat MIN_VALUE no representa el nombre negatiu més petit, sinó el positiu, més pròxim al 0, que pot representar el llenguatge. Els valors més petits que aquest es converteixen en 0.
- Les propietats MAX_VALUE, MIN_VALUE, POSITIVE_INFINITY i NEGATIVE_INFINITY són estàtiques. Per tant, s'han d'utilitzar amb l'objecte mateix, per exemple: Number.MAX_VALUE.
- Per a veure el valor de la propietat NaN, s'ha de fer servir la funció isNaN.
- Un valor més petit que NEGATIVE_INFINITY es visualitza com a -Infinity.
- Un valor més gran que POSITIVE_INFINITY es visualitza com a Infinity.

Taula 14. Mètodes de l'objecte Number

Nom	Descripció	Sintaxi	Paràmetres	Retorn
toExponential	Expressa un nombre en notació exponencial.	toExponential(n)	n: nombre de decimals	String
toFixed	Per a arrodonir a partir del nombre de decimals indicat.	toFixed(n)	n: nombre de decimals	String
toString	String que representa l'objecte especificat.	toString([base])	base: nombre entre 2 i 16 que indica la base en què es representa el nombre.	String
toPrecision	La dada expressada segons la precisió indicada.	toPrecision(n)	n: nombre de dígits de precisió.	String
valueOf	Obté el valor que té l'objecte.	valueOf()		String

3.3.3. L'objecte String

L'objecte String és el contenidor de tipus primitius de tipus cadena de caràcters. Proporciona un conjunt molt ampli de mètodes que permeten manipular i extreure cadenes i convertir-les a text HTML.

La sintaxi del constructor de l'objecte String és la següent:

```
new String(text)
```

en què text és una cadena de caràcters opcional en el constructor.

Els mètodes de l'objecte es poden cridar en cadenes primitives, és a dir, sense haver creat l'objecte amb el constructor. Aquesta característica fa que la creació de cadenes amb la sintaxi anterior sigui poc comuna:

```
var cadena = "Hola, Montse";
longitud = cadena.length;
```

La propietat length de l'objecte String, a diferència de l'objecte Array, no la pot establir el programador; es tracta d'una propietat de lectura i prou i canvia quan la cadena modifica la grandària que té en caràcters.

En general, tots els mètodes de l'objecte String no modifiquen el contingut de l'objecte, sinó que tornen el resultat de l'acció, encara que sense modificar el contingut inicial de la cadena; per a modificar una cadena, s'hi ha d'assignar un valor nou:

```
var cadena = "Hola, Món";
cadena.toUpperCase();
```

La variable *cadena* continua emmagatzemant el valor "Hola, Món"; la funció ha actuat i ha tornat la cadena canviant els caràcters a majúscules, però no ha modificat la variable que conté la cadena original:

```
var cadena = "Hola, Món";
cadena = cadena.toUpperCase();
```

En aquest cas, s'ha modificat la variable *cadena* i ara té el contingut "Hola, Món".

Treballar amb cadenes

El mètode charAt(enter) torna el caràcter que hi ha en la posició que indica el nombre que s'ha passat com a paràmetre; s'ha de tenir en compte que, igual que en les arrays, el primer caràcter d'una cadena té l'índex 0:

```
var cadena = "Hola, Món";
caracter = cadena.charAt(2); //La variable caracter recupera el valor "l".
```

El mètode indexOf(cadena) torna l'índex de la primera aparició de l'argument a la cadena. Seguint l'exemple anterior:

```
var cadena = "Hola, Món";
posicio = cadena.indexOf("Món"); //La variable posicio recupera el valor 6.
```

En cas que l'argument no sigui a la cadena, el mètode torna el valor -1. El mètode accepta un segon paràmetre opcional que especifica l'índex des del qual s'ha de començar la cerca:

```
var cadena = "Hola, Montse";
posicio = cadena.indexOf("ó",3); //La variable posicio recupera el valor 7.
```

El mètode lastIndexOf(cadena) torna l'índex de la darrera aparició de la cadena passada com a argument. Igual que l'anterior, disposa d'un argument opcional que indica l'índex en què s'ha d'acabar la cerca:

```
var cadena = "Hola, Món";
posicio = cadena.lastIndexOf("o",3); //La variable posicio recupera el valor 1.
```

Igual que l'anterior, torna el valor -1 quan no troba la cadena que busca.

El mètode substring(inici,final) fa una extracció de la cadena, de manera que el primer argument especifica l'índex en què comença la cadena que es vol extreure i el segon argument (opcional) assenyala el final de la subcadena.

En cas que no s'hi passi el segon argument, s'extreu la subcadena fins al final de la cadena original:

```
var cadena = "Hola, Món";
var subCadena = cadena.substring(5); //subCadena adquireix el valor " Món"
var subCadena2 = cadena.substring(5,7); //subCadena2 adquireix el valor " M"
```

El mètode concat() fa la concatenació de totes les cadenes que s'hi passen com a paràmetres (accepta qualsevol quantitat d'arguments) i torna la cadena resultat de concatenar la cadena original amb les que s'hi han passat com a paràmetres:

```
var cadena = "Hola, Món";
var cadena2 = cadena.concat(" lliure ", "i feliç"); //cadena2 adquireix el valor
"Hola, Món lliure i feliç"
```

La concatenació, però, és més comuna de fer-la amb l'operador +:

```
var cadena2 = cadena + " lliure " + "i feliç";
```

El mètode split() divideix la cadena en cadenes separades segons un delimitador passat com a argument del mètode; el resultat s'emmagatzema en una array:

```
var vector = "Hola, Món lliure".split(" ");
```

L'exemple anterior assigna a la variable *vector* una array amb tres elements: "Hola,", "Món", "lliure".

Taula 15. Propietats de l'objecte String

Nom	Descripció	Exemple
constructor	Referència a la funció que s'ha cridat per a crear l'objecte.	

Nom	Descripció	Exemple
length	Longitud de l'string	var tema = "Programació" alert("La longitud de la paraula Programació és " + tema.length)

Taula 16. Mètodes de l'objecte String

Nom	Descripció	Sintaxi	Paràmetres	Retorn
anchor	Crea una àncora HTML.	anchor(nom_ancora)	nom_ancora: text per a l'atribut NAME de l'etiqueta `< A NAME=`.	
big	Mostra un text en lletres grans.	big()		
blink	Mostra el text parpellejant.	blink()		
bold	Mostra el text en negreta.	bold()		
charAt	Retorna un caràcter del text, el que és en la posició indicada.	charAt(índex)	índex: nombre entre 0 i la longitud-1 del text.	Caràcter del text que és en la posició indicada pel paràmetre.
charCodeAt	Retorna el codi ISO-Latin-1 del caràcter que és en la posició indicada.	charCodeAt(índex)	índex: nombre entre 0 i la longitud-1 del text.	Codi del caràcter que és en la posició indicada pel paràmetre.
concat	Uneix dues cadenes de text.	concat(text2)	text2: cadena de text que s'ha d'unir a la que crida el mètode.	String resultat de la unió de les altres dues.
fixed	Mostra el tipus amb font de lletra teletip.	fixed()		
fontcolor	Mostra el text en el color especificat.	fontcolor(color)	color: color per al text.	
fontsize	Mostra el text en la grandària especificada.	fontsize(size)	size: grandària per al text.	
fromCharCode	Retorna el text creat a partir de caràcters en codi ISO-Latin-1.	fromCharCode(num1, ..., numN)	numN: codis ISO-Latin-1	String
indexOf	Retorna l'índex en què hi ha per primera vegada la seqüència de caràcters especificada.	indexOf(valor, inici)	valor: caràcter o caràcters que s'han de buscar. inici: índex a partir del qual comença a buscar.	Nombre que indica l'índex. Retorna -1 si no es troba el valor especificat.
italics	Mostra el text en cursiva.	italics()		
lastIndexOf	Retorna l'índex en què hi ha per darrera vegada la seqüència de caràcters especificada.	lastIndexOf(valor, inici)	valor: caràcter o caràcters que s'han de buscar. inici: índex a partir del qual comença a buscar.	Nombre que indica l'índex. Retorna -1 si no es troba el valor especificat.
link	Crea un enllaç HTML.	link(href)	href: string que especifica la destinació de l'enllaç.	
match	Retorna les parts del text que coincideixen amb l'expressió regular indicada.	match(exp)	exp: expressió. Pot incloure els indicadors /f (global) i / i (ignorar majúscules i minúscules).	Array que conté els textos que s'han trobat.

Nom	Descripció	Sintaxi	Paràmetres	Retorn
replace	Substitueix una part del text pel text nou indicat.	replace(exp, text2)	exp: expressió regular per a fer la cerca del text que s'ha de substituir. text2: text que substitueix el que s'ha trobat.	String nova.
search	Retorna l'índex del text que s'ha indicat en l'expressió regular.	search(exp)	exp: expressió per a fer la cerca.	
slice	Extreu una porció de l'string.	slice(inici,[final])	inici: índex del primer caràcter que s'ha d'extreure. final: índex del darrer caràcter que s'ha d'extreure. Pot ser negatiu, i llavors indica quants en cal restar des del final. Si no s'hi indica, extreu fins i tot el final.	String que conté els caràcters que hi havia entre inici i final.
small	Mostra el text en font petita.	small()		
split	Crea una array, i separa el text segons el separador indicat.	split([separador], [límit])	separador: caràcter que indica per on s'ha de separar. Si s'omet, l'array només contindrà un element, que és l'string completa. límit: indica el nombre màxim de parts per a posar en l'array.	Array.
strike	Mostra el text ratllat.	strike()		
sub	Mostra el text com a subíndex.	sub()		
substr	Retorna una porció del text.	substr(inici, [longitud])	inici: índex del primer caràcter per extreure. longitud: nombre de caràcters per extreure. Si s'omet, s'extreu fins al final de l'string.	String.
substring	Retorna una porció del text.	substring(inici, final)	inici: índex del primer caràcter que s'ha d'extreure. final: índex+1 del darrer caràcter que s'ha d'extreure.	
sup	Mostra el text com a superíndex.	sup()		
toLocaleLowerCase	Converteix el text a minúscules, tenint en compte el llenguatge de l'usuari.	toLocaleLowerCase()		String nova en minúscules.
toLocaleUpperCase	Converteix el text a majúscules, tenint en compte el llenguatge de l'usuari.	toLocaleUpperCase()		String nova en majúscules.
toLowerCase	Converteix el text a minúscules.	toLowerCase()		String nova en minúscules.
toUpperCase	Converteix el text a majúscules.	toUpperCase()		String nova en majúscules.
toString	Obté l'string que representa l'objecte.	toString()		String.

Nom	Descripció	Sintaxi	Paràmetres	Retorn
valueOf	Obté l'string que representa el valor de l'objecte.	valueOf()		String.

Codis ISO-Latin-1

El conjunt de codis ISO-Latin-1 agafa valors de 0 a 255. Els 128 primers es corresponen amb el codi ASCII.

4. Expressions regulars i ús de galetes

4.1. Les expressions regulars

Les expressions regulars són un mecanisme que permet fer cerques, comparacions i certs reemplaçaments complexos. Per exemple, si s'escriu en la línia d'ordres de Microsoft Windows l'ordre següent:

```
dir *.exe,
```

s'utilitza una expressió regular que defineix totes les cadenes de caràcters que comencin per qualsevol valor seguit de .exe, és a dir, tots els arxius executables independentment del nom que tenen.

L'acció anterior, en què es compara la cadena de text amb el patró (expressió regular), s'anomena *reconeixement de patrons* (*pattern matching*).

En JavaScript, les expressions regulars es basen en les del llenguatge Perl, de manera que s'hi assemblen molt i es representen per l'objecte RegExp (*regular expresion*).

Es pot crear una expressió regular amb el constructor de l'objecte RegExp o bé utilitzant una sintaxi creada especialment per a això. En l'exemple següent es veu això darrer:

```
var patro = /Cubo/;
```

L'expressió regular anterior és molt simple: en una comparació amb una cadena tornaria true si es comparava amb la cadena Cubo.

Escriptura

Totes les expressions regulars s'escriuen entre barres invertides.

De la mateixa manera, es pot crear l'expressió regular anterior utilitzant l'objecte RegExp:

```
var patro = new RegExp("Cubo");
```

En aquest cas, però, el que es passa al constructor és una cadena; per tant, en comptes de fer servir (/), es tanca entre cometes dobles.

Nota

Les expressions regulars es poden crear utilitzant la sintaxi específica que tenen o mitjançant l'objecte RegExp.

Per a complicar una mica més l'exemple anterior, imaginem-nos que es vol comprovar si la cadena és Cubo o Cuba. Per a això es fan servir els claudàtors, que indiquen opció, és a dir, la comparació amb /[ao]/ tornaria cert, en cas que la cadena fos la lletra *a* o la lletra *o*.

```
var patro = /Cub[ao]/;
```

I si es volgués comprovar si la cadena és Cub0, Cub1, Cub2,..., Cub9? En comptes d'incloure els deu dígits amb els claudàtors, s'utilitza el guió, que serveix per a indicar rangs de valors. Per exemple, 0-9 indica tots els nombres de 0 a 9 inclusivament:

```
var patro = /Cub[0-9]/;
```

Si, a més, es busca que el darrer caràcter sigui un dígit (0-9) o una lletra minúscula (a-z), s'aconsegueix de manera fàcil escrivint dins els claudàtors un criteri darrere de l'altre:

```
var patro = /Cub[0-9a-z]/;
```

I què passaria si en comptes de tenir només un nombre o una lletra minúscula es volgués comprovar que n'hi pugui haver més d'un, però sempre minúscules o nombres?

En el cas anterior s'ha de recórrer als marcadors següents:

- +: indica que el que té a l'esquerra hi pot ser 1 vegada o més.
- *: indica que hi pot ser 0 vegades o més (en el cas de + el nombre o la minúscula hi ha de ser com a mínim una vegada; amb * Cub també s'acceptaria).
- ?: indica opcionalitat, és a dir, el que es té a l'esquerra hi pot ser o no (hi pot aparèixer 0 o 1 vegades).
- {}: serveix per a indicar exactament el nombre de vegades que hi pot aparèixer o un rang de valors. Per exemple: {3} indica que hi ha d'aparèixer exactament 3 vegades; {3, 8} indica que hi ha d'aparèixer de 3 a 8 vegades, i {3,} indica que hi ha d'aparèixer com a mínim tres vegades.

S'ha d'anar amb compte amb els {}, perquè exigeixen que es repeteixi el darrer, de manera que si no està segur del que faran, s'han d'utilitzar els (). Per a il·lustrar-ho, es planteja l'exemple següent d'ús d'expressions regulars:

```
var patro = /Cub[ao]{2}/;
document.write("Cubocuba".search(patro));
document.write("Cuboa".search(patro));
```

Funció search i mètode replace

La funció search de String comprova si la cadena que representa el patró que s'hi passa com a argument és dins de la cadena sobre la qual es crida. Si hi és, torna la posició (per exemple, per a la cadena Cubo amb el patró /C/ torna 0, 1 si el patró és u, 2 si és b...) i si no hi és, torna -1.

Un altre ús interessant és l'ús del mètode replace de l'objecte String, la sintaxi del qual, cadena.replace(patro, substitut), indica que se substitueix la cadena sobre la qual es criden les ocurrències del patró per la cadena especificada en la crida.

Un dels elements més interessants de les expressions regulars és l'especificació de la posició que ha d'ocupar la cadena. Per a això, s'utilitzen els caràcters ^ i $, que indiquen que l'element sobre el qual actuen ha d'anar al començament de la cadena o al final de la cadena, respectivament. En l'exemple següent, se'n pot veure l'ús:

```
var patro = /^aa/; //Es busca la cadena aa a l'inici de la cadena
var patro = /uu$/; //Es busca la cadena uu al final de la cadena
```

Algunes altres expressions interessants són les següents:

- \d: un dígit; equival a [0-9].
- \D: qualsevol caràcter que no sigui un dígit.
- \w: qualsevol caràcter alfanumèric; equival a [a-zA-Z0-9_].
- \W: qualsevol caràcter no alfanumèric.
- \s: espai.
- \t: tabulador.

Web recomanat

Hi ha un portal web dedicat a les expressions regulars. Es pot destacar que aquestes expressions són compatibles en la majoria dels llenguatges de programació que les implementen: www.regular-expressions.info/

4.1.1. L'objecte RegExp

L'objecte RegExp conté el patró d'una expressió. El constructor per a l'objecte RegExp és el següent:

```
new RegExp("patró", "indicador")
```

en què:

a) patró, text de l'expressió regular.

b) indicador, és opcional i pot agafar tres valors:

- g: es tenen en compte totes les vegades que l'expressió apareix en la cadena.
- i: ignora majúscules i minúscules.
- gi: tenen efecte les dues opcions: g i i.

En el patró de l'expressió regular, es poden fer servir caràcters especials. Els caràcters especials substitueixen una part del text. Tot seguit es mostren els caràcters especials que es poden utilitzar:

Taula 17. Caràcters especials en expressions regulars

\	Per als caràcters que normalment s'interpreten com a literals, indica que el caràcter que el segueix no s'ha d'interpretar com un literal. Per exemple: /b/ s'interpretaria com a *b*, però /\b/ s'interpretaria com a indicador de límit de paraula. Per als caràcters que normalment no s'interpreten com a literals, indica que en aquest cas sí que s'ha d'interpretar com un literal i no com un caràcter especial. Per exemple, el caràcter * és un caràcter especial que s'utilitza com a comodí, /a*/ pot voler dir cap o diverses *a*. Si l'expressió conté /a*/ s'interpreta com el literal "a*".

^	Indica inici de línia. Per exemple: /^A/ no trobarà la *A* en la cadena *HOLA*, però sí que la trobarà en *Alarma*.
$	Indica final de línia. Per exemple: /$A/ no trobarà la *A* en la cadena *ADÉU*, però sí que la trobarà en *HOLA*.
*****	Indica que el caràcter que el precedeix pot aparèixer cap vegada o diverses vegades. Per exemple: /ho*/ es trobarà en *hola*, *hoooola* i també en *heura*, però no en *camell*.
+	Indica que el caràcter que el precedeix pot aparèixer una vegada o diverses vegades. Per exemple: /ho+/ es trobarà en *hola* i *hoooola*, però no en *heura* ni en *camell*.
?	Indica que el caràcter que el precedeix pot aparèixer cap vegada o una vegada. Per exemple: /ho?/ es trobarà en *hola* i *heura*, però no en *hoooola* ni en *camell*.
.	Indica un sol caràcter a excepció del salt de línia. Per exemple: / .o/ es trobarà en *hola* però no en *camell*.
(x)	Indica que, a més de buscar el valor *x*, es repetirà la cerca entre el resultat de la primera cerca. Per exemple: en la frase "hola, t'espero a l'hotel d'holanda", /(holan*)/ trobaria *hola*, *holan* i *hola*. (El darrer *hola* és part d'*holan*.)
x|y	Indica el valor de *x* o el de *y*. Per exemple: /sol|vent/ trobaria *sol* en la frase "Fa sol a Sevilla".
{n}	Indica quantes vegades exactes ha d'aparèixer el valor que el precedeix (*n* és un enter positiu). Per exemple: /o{4}/ es trobaria en *hoooola* però no en *hola*.
{n,}	Indica quantes vegades ha d'aparèixer com a mínim el caràcter que el precedeix (*n* és un enter positiu). Per exemple: /o{2,}/ es trobaria en *hoooola* i *hoola* però no en *hola*.
{n,m}	Indica el nombre mínim i màxim de vegades que pot aparèixer el caràcter que el precedeix (*n* i *m* són enters positius). Per exemple: /o(1,2)/ es trobaria en *hola* i *hoola*, però no en *hoooola*.
[xyz]	Indica qualsevol dels valors entre claudàtors. Els elements continguts expressen un rang de valors. Per exemple: [abcd] es pot expressar també com a [a-d].
[^xyz]	Busca qualsevol valor que no aparegui entre claudàtors. Els elements continguts expressen un rang de valors. Per exemple: [^abcd] es pot expressar també com a [^a-d].
[\b]	Indica un retrocés o *backspace*.
\b	Indica un delimitador de paraula, com un espai. Per exemple: /\bn/ es troba en *nyu* però no en *mico*, i /c\b/ en *camaleó* però no en *mico*.
\B	Indica que no hi pot haver delimitador de paraula, com un espai. Per exemple: /\Bn/ es troba en *manel* però no en *nyu* ni en *camaleó*.
\cX	Indica un caràcter de control (X és el caràcter de control). Per exemple: /cm/ indica Ctrl + *M*.
\d	Indica que el caràcter és un dígit. També es pot expressar com a /[0-9]/. Per exemple, /\d/ en "carrer peix, n. 9" trobaria 9.
\D	Indica que el caràcter no és un dígit. /\D/ també es pot expressar com a /[^0-9]/.
\f	Indica salt de pàgina (*form-feed*).
\n	Indica salt de línia (*linefeed*).
\r	Indica tecla de retorn.
\s	Indica un espai en blanc que pot ser l'espai, el tabulador, el salt de pàgina i el salt de línia. Per tant, equival a posar [\f\n\r\t\v].
\S	Indica un sol caràcter diferent de l'espai, del tabulador, del salt de pàgina i del salt de línia. Per tant, equival a posar [^\f\n\r\t\v].
\t	Indica el tabulador.
\v	Indica un tabulador vertical.
\w	Indica qualsevol caràcter alfanumèric, inclòs el caràcter _. Equival a posar [A-Za-z0-9_].

n	*n* és un valor que fa referència al parèntesi anterior (compta els parèntesis oberts). Per exemple: a "poma, taronja, pera, préssec", l'expressió /poma(,)\staronja\1/ trobaria "poma, taronja".
\ooctal **\xhex**	Permet d'incloure codis ASCII en expressions regulars. Per a valors octals i hexadecimals. o i x agafarien els valors, per exemple, \2Fhex.

Taula 18. Propietats de l'objecte RegExp

Nom	Descripció
$1,..., $9	Contenen les parts de l'expressió que hi ha entre parèntesis. Només és de lectura.
$_	Vegeu la propietat input.
$*	Vegeu la propietat multiline.
$&	Vegeu la propietat lastMatch.
$+	Vegeu la propietat lastParen.
$`	Vegeu la propietat leftContext.
$'	Vegeu la propietat rightContext.
global	Indica si es fa servir g en l'expressió regular. Es poden tenir els valors true, si es fa servir l'indicador g, i false, si no es fa servir. Només és de lectura.
ignoreCase	Indica si es fa servir l'indicador i en l'expressió regular. Es poden tenir els valors true, si es fa servir l'indicador i, i false, si no es fa servir. Només és de lectura.
input	Representa l'string sobre el qual s'aplica l'expressió regular. També es diu $_. És estàtica i, per tant, es fa servir de la manera següent: RegExp.input.
lastIndex	Especifica l'índex a partir del qual s'ha d'aplicar l'expressió regular.
lastMatch	Representa el darrer ítem que s'ha trobat. També es diu $&. És estàtica i, per tant, es fa servir de la manera següent: RegExp. lastMatch. Només és de lectura.
lastParen	Representa el darrer ítem que ha trobat una expressió de parèntesis. També es diu $+. És estàtica i, per tant, es fa servir de la manera següent: RegExp. lastParen. Només és de lectura.
leftContext	Representa la substring que precedeix el darrer ítem que s'ha trobat. També es diu $'. És estàtica i, per tant, es fa servir de la manera següent: RegExp. leftContext. Només és de lectura.
multiline	Indica si s'aplicarà la cerca en diverses línies. Els valors possibles que té són true i false. També es diu $*. És estàtica i, per tant, es fa servir de la manera següent: RegExp. multiline.
prototype	Representa el prototip per a la classe.
rightContext	Representa la substring que segueix el darrer ítem que s'ha trobat. També es diu $'. És estàtica i, per tant, es fa servir de la manera següent: RegExp. rightContext. Només és de lectura.
source	Representa l'string que conté el patró per a l'expressió regular, excloses les \ i els indicadors i i g. Només és de lectura.

Taula 19. Mètodes de l'objecte RegExp

Nom	Descripció	Sintaxi	Paràmetres	Retorn
compile	Compila l'expressió regular durant l'execució d'un script.	regexp.compile(patró, [indicadors])	patró: text de l'expressió regular. indicadors: g o i	
exec	Executa la cerca.	regexp.exec(str) regexp(str)	str: string sobre la qual s'aplica la cerca.	Una array amb els ítems que s'han trobat.

Nom	Descripció	Sintaxi	Paràmetres	Retorn
test	Executa la cerca.	regexp.test(str)	str: string sobre la qual s'aplica la cerca.	true si troba algun ítem; false si no en troba cap.

4.2. Les galetes

Les galetes (*cookies*) neixen amb l'objectiu de solucionar una limitació del protocol HTTP 1.0, que ocorre pel fet que HTTP és un protocol sense estat. Això fa que no hi hagi una manera de mantenir comunicació o informació de l'usuari durant les diverses peticions que es fan al servidor en una mateixa connexió o visita a un lloc web o una pàgina web.

Les galetes aporten un mecanisme que permet emmagatzemar en l'equip del client un conjunt petit de dades de tipus text que estableix el servidor web. D'aquesta manera, en cada connexió el client torna la galeta amb el valor emmagatzemat al servidor que processa el valor i actua de manera que duu a terme les accions pertinents.

L'assignació d'una galeta segueix la sintaxi següent:

```
nom=valor [;expires=dataGMT] [;domain=domini] [;path=ruta] [;secure]
```

en què:

- nom=valor: defineix el nom de la galeta i el valor que aquesta galeta emmagatzemarà.

- expires=dataGMT: estableix la data de caducitat de la galeta; aquesta data s'ha d'establir en format GMT i, per tant, és útil fer servir el mètode toGMTString() de l'objecte Date. Aquest paràmetre és opcional, de manera que quan una galeta no té establerta una data de caducitat, aquesta galeta es destrueix quan l'usuari tanca el navegador; en aquest cas, es diu que la *galeta és de sessió*. Les galetes que tenen establerta una data de caducitat es coneixen com a *galetes persistents*.

- domain=domini: estableix el domini que ha assignat la galeta, de manera que aquesta galeta només es torna davant d'una petició d'aquest domini. Per exemple: domain=www.uoc.edu implica que la galeta només es torna al servidor www.uoc.edu quan s'hi estableix una connexió. Si no s'estableix cap valor, el mateix navegador estableix el valor del domini que ha creat la galeta.

- path=ruta: estableix una ruta específica del domini sobre la qual es torna la galeta. Si no s'estableix, la ruta per defecte que ha assignat el navegador és la ruta des de la qual s'ha creat la galeta.

- secure: si s'indica aquest valor, la galeta només es torna quan s'ha establert una comunicació segura amb HTTPS. Si no s'assigna aquest valor, el navegador torna la galeta en connexions no segures HTTP.

Per tant, el funcionament bàsic és el següent: un usuari es connecta a un lloc web; el navegador, a partir de l'URL, revisa el conjunt de galetes amb la intenció de cercar-ne una que coincideixi amb el domini i la ruta. Si n'hi ha una d'aquestes característiques, el navegador l'envia al servidor i, si n'hi ha més d'una, les envia separades pel caràcter punt i coma:

```
nom=Victor; email=vriospazos@uoc.edu
```

Perquè les galetes funcionin, s'han de tenir habilitades al navegador. Hi ha usuaris que les consideren un mecanisme d'invasió a la intimitat, perquè es poden establir galetes persistents que facin un seguiment de certes accions en el procés de navegació dels usuaris.

4.2.1. Maneig de galetes

En JavaScript es pot treballar amb galetes a partir de la propietat cookie de l'objecte document. El funcionament és molt simple: només s'ha d'assignar a aquesta propietat una cadena que representi la galeta, de manera que el navegador analitza la cadena com a galeta i l'afegeix a la llista de galetes que té.

```
document.cookie="nom=Victor; expires=Sun, 14-Dec-2010 08:00:00 GMT; path=/fitxers";
```

En l'assignació anterior, es crea una galeta persistent amb data caducitat del 14.12.2010; en aquest cas, s'assigna una ruta que s'utilitzarà juntament amb el domini per defecte de la pàgina que ha creat la galeta.

Aquest mecanisme de domini/ruta fa que una galeta només la pugui recuperar el domini/ruta que s'indica, cosa que impedeix que s'accedeixi a la informació emmagatzemada des d'altres dominis.

L'anàlisi que fa el navegador sobre la cadena assignada a document.cookie comprova que el nom i el valor no tenen caràcters com ara espais en blanc, comes, la lletra *ñ*, accents i altres caràcters.

Per a solucionar aquest problema, s'utilitzen les funcions escape() i unescape(), que porten a cap la conversió de cadenes en cadenes URL que el validador del navegador dóna per bones.

D'aquesta manera, s'utilitza el mètode escape() per a convertir una cadena en format URL abans d'emmagatzemar-la en la galeta i, quan la galeta es recupera, s'utilitza la funció unescape() sobre el valor de la galeta.

4.2.2. Escriure, llegir i eliminar galetes

L'escriptura de galetes és molt senzilla: només s'ha d'assignar a la propietat cookie una cadena en què s'especifiqui el nom, el valor i els atributs de caducitat, el domini, la ruta i la seguretat que es volen aplicar.

En l'exemple següent, es mostra una funció que rep com a paràmetres el nom, el valor i, com a paràmetre opcional, el nombre de dies durant els quals la galeta serà activa, de manera que si no es passa d'aquest nombre de dies, la galeta serà de sessió i, per tant, s'eliminarà quan l'usuari tancarà el navegador.

```
function assignaCookie(nom,valor,dies){
       if (typeof(dies) == "undefined"){
                //Si no es passa del paràmetre dies, la cookie és de sessió
                document.cookie = nom + "=" + valor;
       } else {
                //Es crea un objecte Date al qual s'assigna la data actual
                //i s'hi afegeixen els dies de caducitat transformats en
                //mil·lisegons
                var caduca = new Date;
                caduca.setTime(caduca.getTime() + dies*24*3600000);
document.cookie = nom + "=" + valor + ";expires=" + caduca.toGMTString();
       }
}
```

La lectura de galetes es fa examinant la cadena emmagatzemada en la propietat cookie de l'objecte Document, però s'ha de tenir en compte que aquesta cadena és formada per tants parells nom=valor com galetes de valor diferent s'han establert en el document actual. Per exemple, si es fa l'assignació següent:

```
document.cookie= "nom=Victor";
document.cookie="email=vriospazos@uoc.edu"
```

Cadascuna de les cadenes anteriors s'afegeix a la galeta, de manera que el valor de document.cookie és el següent:

```
"nom=Victor; email=vriospazos@uoc.edu"
```

Normalment, només s'ha de recuperar alguna galeta concreta o treballar-hi. Per exemple, fa falta la cadena que indica el correu electrònic, però la galeta conté totes les cadenes emmagatzemades.

Això fa que s'hagi d'implementar un mecanisme que recuperi cada cadena de manera separada a partir d'una anàlisi de la cadena que ha tornat document.cookie.

L'exemple següent utilitza una array associativa per a emmagatzemar els noms i els valors de cadascun dels components de la galeta:

```
//Es crea l'objecte que contindrà l'array associativa cookies[nom] = valor
var cookies = new Object();

//Es defineix la funció que analitza la cadena i crea l'array a partir de la cadena
//document.cookie, i es comproven certes situacions especials
function extreuCookies(){

       //Variables que emmagatzemaran les cadenes nom i valor
       var nom, valor;

       //Variables que controlaran els límits que marquen la posició de les
       //diverses cookies a la cadena
       var inici, mig, final;

       //El bucle següent comprova si hi ha alguna entrada en l'array
       //associativa, de manera que si es així, es crea una instància nova de
       //l'objecte cookies per a eliminar-les
       for (nom in cookies){
              cookies = new Object();
              break;
       }
       inici = 0;

       //Es fa un bucle que captura a cada pas el nom i valor de
       //cada cookie de la cadena i l'assigna a l'array associativa
       while (inici< document.cookie.length){
                //la variable mig emmagatzema la posició del pròxim caràcter "="
                mig = document.cookie.indexOf('=', inici);

                //la variable final emmagatzema la posició del pròxim caràcter ";"
                final = document.cookie.indexOf(';',inici);

                //El següent if comprova si final adquireix el valor -1 que indica
                //que no s'ha trobat cap caràcter ";", cosa que indica que
                //s'ha arribat a la darrera cookie i, per tant, s'assigna a la
                //variable final la longitud de la cadena
                if (final == -1) {
                          final = document.cookie.length;
                }

                //El següent if fa dues comprovacions; en primer lloc, si
                //mig és més gran que final o mig és -1 (que indica que no
                //s'ha trobat cap caràcter "="), la cookie té nom però
                //no valor assignat
```

```
                //En segon lloc, el nom de la cookie és entre els
                //caràcters que hi ha entre inici i mig i el valor de la cookie
                //entre els caràcters que hi ha entre mig+1 i final
                if ( (mig > final) || (mig == -1) ) {
                        nom = document.cookie.substring(inici, final);
                        valor = "";
                } else {
                        nom = document.cookie.substring(inici,mig);
                        valor = document.cookie.substring(mig+1,final);
                }

                //Una vegada recuperat el nom i el valor, s'assigna a l'array
                //associativa aplicant la funció de conversió unescape()
                cookies[nom] = unescape(valor);

                //En el pas següent del bucle while, la variable inici adquireix
                //el valor final +2, i d'aquesta manera salten el punt i coma i
                //l'espai que separa les diverses cookies en la cadena
                inici = final +2;
        }
}
```

L'eliminació d'una galeta es fa assignant una data de caducitat del passat; per exemple, es pot utilitzar la data "01-Jan-1970 00:00:01 GMT".

L'únic problema apareix quan la galeta no té un valor assignat; aquests casos s'han de tractar de manera especial, com es veu en l'exemple següent:

```
function eliminaCookie(nombre){
        //S'actualitza la cookie modificant la data de caducitat i assignant
        //un valor qualsevol, en l'exemple esborrada
        document.cookie = nom + "=esborrada; expires=Thu, 01-Jan-1970 00:00:01 GMT";
        //S'actualitza la cookie sense valor indicant una data de caducitat //anterior
        document.cookie = nom + "; expires=Thu, 01-Jan-1970 00:00:01 GMT";
}
```

De la funció anterior, es pot dir que, en cas que la galeta s'hagi creat amb informació sobre el domini i la ruta, s'ha d'introduir aquesta informació a la cadena que actualitza o esborra la galeta.

Les funcions anteriors proporcionen prou mecanismes per a controlar les galetes, encara que aquestes galetes es poden modificar depenent de les necessitats del programador.

Tot seguit es crearan dues galetes, de nom "nom" i "email" i els valors "Victor" i "vriospazos@uoc.edu", respectivament:

```
assignaCookie("nom", "Victor");
assignaCookie("email","vriospazos@uoc.edu");
```

S'obté l'array associativa amb les galetes emmagatzemades cridant la funció extreuCookies(), i aquestes galetes es poden utilitzar:

```
extreuCookies();
var nom = cookies["nom"];
var corr = cookies["email"];
```

S'eliminen les dues galetes utilitzant la funció eliminaCookie():

```
eliminaCookie("nom");
eliminaCookie("email");
```

4.2.3. Usos principals de les galetes

Actualment, s'utilitzen les galetes en els casos següents:

- Per a emmagatzemar l'estat de l'usuari, adaptant la presentació o el contingut de la pàgina i basant-se en les preferències de l'usuari.

- Per a tornar a encaminar l'accés a una pàgina diferent quan l'usuari compleix certes condicions; per exemple, un usuari registrat passa directament a les pàgines de contingut, mentre que si és la primera vegada que hi accedeix, s'encamina a la pàgina de registre.

- Com en el cas anterior, a un usuari que accedeix per primera vegada a una pàgina se li pot obrir una finestra d'informació inicial, de manera que aquesta finestra només s'obre la primera vegada que hi accedeix.

4.2.4. Limitacions

Cada navegador imposa un seguit de limitacions sobre la grandària i el nombre de galetes que es poden establir. En general, són les següents:

- En un moment concret, un navegador pot emmagatzemar diversos centenars de galetes.

- Cada lloc web pot tenir com a màxim 20 galetes.

- Cada galeta té un límit de 4.000 caràcters en el seu valor.

Evidentment, es tracta de limitacions aproximades, ja que cada navegador aplica les seves pròpies limitacions, que evolucionen amb cada versió nova.

Activitats

1. Descriviu tres exemples d'objectes del món real:

- Per a cadascun dels objectes, definiu la classe a la qual pertanyen.
- Assigneu a cada classe un identificador descriptiu adequat.
- Enumereu diversos atributs i operacions per a cadascuna de les classes.

2. Creeu una classe per a cadascun dels objectes plantejats en l'exercici anterior utilitzant el llenguatge Javascript.

3. Creeu una pàgina web en la qual es creen almenys 2 objectes de les classes anteriors i s'utilitzen els seus mètodes i propietats.

4. Creeu un rellotge que l'usuari del web actualitzi manualment fent un clic en un botó.

5. Utilitzant l'objecte String creeu dues funcions:

- encripta(text): que substitueix cadascun dels caràcters de la cadena per d'altres, de manera que el text resultant quedarà inintel·ligible.
- desencripta(text): que realitza la substitució inversa a la de la funció anterior.

Utilitzeu les dues funcions anteriors en una pàgina web on perquè un text introduït per l'usuari, s'encripti o desencripti.