Manipular el DOM amb JavaScript

Vicent Moncho Mas
Gemma Subirana Grau

PID_00191129

UOC
Universitat Oberta de Catalunya

Índex

1. Manipular la pàgina web

En aquest apartat es presentarà un seguit d'exemples de codi JavaScript que, basant-se en les tècniques estudiades en els mòduls anteriors, proporcionarà una visió de les possibilitats que ofereix el llenguatge al programador.

L'apartat s'estructura de la manera següent:

- Les característiques del client.
- Les finestres.
- Els marcs.
- La navegació i el pas de variables.

Al llarg de l'apartat es plantejaran exemples senzills que indiquen la base de les tècniques que permeten un control necessari en la creació de webs de certa complexitat.

1.1. Les característiques del client

Malgrat que l'estàndard DOM del W3C s'implementa en els navegadors principals pas a pas, encara no hi ha una situació de convergència o compatibilitat real entre tots els navegadors.

El fet anterior obliga el programador web a ser conscient que hi ha codis que poden funcionar d'una manera diferent o simplement funcionar en una versió concreta del navegador d'un fabricant determinat; per això és molt important de saber, abans d'executar un script, quin és el client que l'interpretarà.

1.1.1. Consultar el navegador

Es pot consultar el fabricant del navegador a partir de la propietat appName de l'objecte Navigator. El retorn d'aquest mètode és el següent:

- Netscape quan el client és Firefox, Safari o Chrome.
- Microsoft Internet Explorer quan el client és Internet Explorer.

En el cas d'Opera, la funció pot tornar qualsevol de les opcions anteriors més Opera, perquè depèn de certs paràmetres de configuració del navegador. Per tant, s'ha de consultar la propietat userAgent, que proporciona més informació.

La informació més completa sobre la versió d'un navegador, però, s'obté a partir del mètode userAgent de l'objecte Navigator. Aquest mètode torna una cadena que conté un seguit de paràmetres que, a més d'identificar-ne el fabricant, n'identifiquen la versió concreta. Per a això, s'ha de dur a terme certa cerca en la cadena d'informació clau. Per exemple:

- La subcadena MSIE identifica Internet Explorer i va seguida d'un espai i el número de versió del navegador.
- La subcadena Gecko identifica el motor implementat en el navegador. Aquest motor s'utilitza en els navegadors Mozilla, Netscape, Firefox i Camino.
- Si apareix la subcadena Safari i no apareix Chrome, s'està indicant que el motor implementat en el navegador és el de Safari.
- La subcadena Opera identifica que el navegador és Opera.
- La subcadena Chrome indica que el navegador és Chrome.

Per tant, la implementació d'aquesta cerca es pot fer amb la sintaxi següent:

```
var agent = navigator.userAgent;
if (agent.indexOf(subcadena) != -1)
```

en què *subcadena* s'ha de substituir per la subcadena del navegador que es vol consultar.

1.1.2. Consultar el sistema operatiu

El mateix codi anterior pot servir per a detectar el sistema operatiu sobre el qual s'executa el navegador, ja que aquest sistema s'identifica en la cadena que torna el mètode userAgent.

La subcadena que s'ha de buscar en cadascun dels casos és la següent:

- Win en el cas d'entorns Microsoft.
- Mac en el cas d'entorns Macintosh.
- X11 en el cas d'entorns basats en Unix.

1.1.3. Consultar si el navegador suporta W3C DOM

El mètode hasFeature() de l'objecte Implementation s'utilitza per a consultar des d'un script si el navegador suporta un dels mòduls de l'estàndard del W3C. En l'exemple següent es carrega un full o un altre d'estils, depenent de si el navegador suporta l'estàndard CSS del DOM2.

Web recomanat

Podeu consultar la llista de tots els mòduls del nivell 2 de w3c a: www.w3.org.

```
var fitxerCSS;
if (document.implementation.hasFeature("CSS2","2.0")){
        fitxerCSS = "estilCSS2.css";
} else {
```

```
        fitxerCSS = "estilCSS1.css";
}
document.write(<link rel='stylesheet' type='text/css' href='"+fitxerCSS +"'>");
```

Web recomanat

Consulteu la llista de tots els mòduls del nivell 2 de W3C en el web de W3C.

1.2. Les finestres

El maneig de les finestres és un aspecte controvertit en la programació amb JavaScript. Des del començament, les finestres han format part d'un seguit de recursos dels programadors de JavaScript per a enriquir la usabilitat o per a entorpir i turmentar l'usuari amb una amalgama de finestres sense sentit.

L'objecte Window no és independent de la divergència dels models de navegador; les propietats i els mètodes que s'implementen depenen força del navegador.

L'abús de les finestres emergents ha arribat a un punt en què els navegadors actuals disposen d'una utilitat per a bloquejar-les que pot configurar l'usuari. Això ja indica que, amb vista a fer una programació amb un cert nivell d'usabilitat, s'ha de mirar d'evitar l'ús de finestres emergents i incloure en la finestra actual tot el contingut.

Amb l'objectiu d'establir mecanismes sòlids de seguretat en la programació amb JavaScript, hi ha una sèrie de restriccions sobre el que es pot fer amb les finestres. Tot seguit s'enumeren algunes de les accions que no es poden fer amb JavaScript:

- Modificar el marc de la finestra actual; encara que es pot redimensionar i situar la finestra del navegador, no es poden agregar o eliminar els components que formen el marc (barra de menú, estat, scroll, etc.). Això només es pot fer en finestres que es creen des d'un script.

- Tancar la finestra principal des d'un script que s'executa en una subfinestra; si es fa la crida, apareix un quadre de diàleg que demana permís a l'usuari per a tancar-la.

- Tancar finestres que no ha creat un script. De fet, no es pot accedir a finestres que no s'han creat des d'un script. No hi ha una array amb les finestres existents.

- Accedir a propietats de Document des d'altres finestres que procedeixen d'altres dominis, és a dir, no es pot accedir a dades d'una pàgina HTML des d'una altra pàgina si aquestes dades no procedeixen del mateix domini.

- Capturar la pulsació dels botons del navegador. Això permet controlar des d'un script el comportament del navegador i és realment perillós.

- Canviar l'adreça de la barra d'adreces; només es pot carregar una pàgina nova, però en canvi amb una pàgina carregada no es pot modificar l'URL d'aquesta pàgina (d'alguna manera, se n'estaria suplantant la identitat).

- Modificar elements de la llista de favorits.

Si s'analitzen les limitacions anteriors a fons, s'arriba fàcilment a la conclusió que són necessàries per a tenir un entorn segur i fiable.

Revisades les limitacions, tot seguit es presentaran algunes accions que es poden fer amb les finestres i que són útils.

1.2.1. Posicionar i maximitzar la finestra principal

Situar la finestra principal del navegador en un punt específic de la pantalla és un procés molt simple. Per a això, tenim el mètode moveTo(x,y), que posiciona la finestra en les coordenades en píxels que s'hi passen com a paràmetres.

Si no es vol situar la finestra en una posició de la pantalla sinó fer un desplaçament respecte a la posició actual, s'ha d'utilitzar el mètode moveBy(x,y).

La implementació dels moviments anteriors no comporta cap canvi en la mida de la finestra. Per a modificar la mida de la finestra, es pot utilitzar el mètode resizeTo(x,y), els paràmetres del qual són els píxels que indiquen la mida que ha de tenir aquesta finestra.

Si es vol maximitzar una finestra del navegador, es pot utilitzar el codi següent:

```
<html>
<head>
<script type="text/javascript">
function obriF()
{
finestra=window.open('','','width=200,height=100');
finestra.document.write("La meva finestra");
}

function maximitza()
{
finestra.moveTo(0,0);
finestra.resizeTo(screen.availWidth,screen.availHeight);
finestra.focus();
}
</script>
```

```
</head>
<body>

<input type="button" value="Crea la finestra" onclick="obriF()" />
<br /><br />
<input type="button" value="Maximitza finestra" onclick="maximitza()" />

</body>
</html>
```

De la funció anterior, se n'han de tenir en compte els detalls següents:

- En primer lloc, se situa la finestra a la part superior esquerra amb la sentència moveTo(0,0).

- En segon lloc, es canvia la grandària al valor màxim disponible d'amplària i alçària. Aquests dos valors s'aconsegueixen a partir de les propietats availWidth i availHeight de l'objecte Screen.

1.2.2. Crear una finestra nova i assignar-hi el focus

La creació d'una finestra nova s'implementa amb el mètode open de l'objecte Window:

```
var novaFinestra = window.open("Pàgina1.html", "Nova pàgina",
"status,menubar,height=400,width=300");
```

En l'exemple de codi anterior, a la finestra nova es carrega la pàgina Pàgina1.html, que es dirà *Nova pàgina*. Les característiques de la pàgina s'especifiquen en la cadena del tercer paràmetre.

Vegeu també

Vegeu la relació completa de les característiques de la pàgina en l'apartat 3 del mòdul "Introducció al DOM" d'aquesta assignatura.

Si es vol crear la pàgina des de zero, s'ha de deixar el primer paràmetre buit, de manera que, tot seguit, s'anirà creant el contingut de la pàgina utilitzant els mètodes write i writeln de l'objecte document.

Quan l'usuari treballa i hi ha diverses finestres, en realitat només treballa amb la pàgina que té el focus actiu o, per exemplificar-ho d'una manera visual, la que hi ha en primer pla. El mètode que permet pujar una finestra que és en un segon pla (amagada darrere la resta) al primer pla és focus().

El codi següent busca si una finestra és oberta: si no ho és, la crea, i si ho és, la porta al primer pla:

```
var novaFinestra;
function portaLaFinestra(url){
       if (!novaFinestra || novaFinestra.closed) {
```

```
                novaFinestra = window.open(url, "Nova", "status, height=200, width=300");
        } else {
                novaFinestra.focus();
        }
}
```

1.2.3. Comunicació entre finestres

En el subapartat anterior, s'ha vist com es pot crear una finestra nova des d'una finestra principal. En aquest veurem com es poden comunicar finestres des de JavaScript que tenen una relació de "parentiu" i que, tal com es deia en la introducció, provenen d'un mateix domini.

En l'exemple següent, es crearà una finestra nova sobre la qual s'escriurà el contingut, i tot això des de la finestra pare:

```
<html>
<script type="text/JavaScript">
//Es defineix una variable global que emmagatzemarà una referència a la finestra
var novaFinestra;
//La funció següent genera i omple de contingut la finestra nova
function creaFinestra(){
        //Comprovem que no estigui creada
        if (!novaFinestra || novaFinestra.closed) {
                novaFinestra = window.open("","Nova", "status, height=200, width=300");
                //Endarrerim l'escriptura 50 ms per evitar problemes
                setTimeout("pintaFinestra()",50);
        } else if (novaFinestra.focus) {
                //La finestra és oberta i és accessible; es porta al primer pla
        novaFinestra.focus();
        }
}
        function pintaFinestra(){
        //Muntem el contingut de la finestra nova
        var contingut = "<html><head><title>Segona finestra</title></head>";
        contingut += "<body><h1>Finestra creada des d'un script</h1>";
        contingut += "</body></html>";
        //S'escriu l'HTML en el document nou de la finestra
        novaFinestra.document.write(contingut);
        novaFinestra.document.close();
}
</script>
</head>
<body>
<form>
<input type="button" value="Crea finestra" id= "BT1" onclick="creaFinestra();"/>
</form>
```

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

Respecte al codi anterior, només es pot comentar que la crida a l'escriptura del contingut a la finestra acabada de crear es duu a terme utilitzant el temporitzador, ja que es pot donar el cas que encara no s'ha acabat de crear la pantalla i ja s'està intentant de pintar-hi a sobre (aquest efecte es pot donar per l'execució en paral·lel del codi que es dóna en els navegadors nous).

Un altre aspecte que s'ha de tenir en compte és que la variable *novaFinestra* emmagatzema una referència a la finestra acabada de crear i és el mecanisme d'accés a la finestra nova des de la principal.

Possiblement, el problema fonamental és com s'ha d'accedir en sentit contrari, és a dir, des de la finestra acabada de crear fer referència a un valor, una propietat o un mètode de la finestra creadora o pare d'aquesta finestra.

Això es pot implementar gràcies a la propietat opener de Window, ja que referencia la finestra que ha estat la responsable de l'execució de la sentència window.open. D'aquesta manera, seguint l'exemple anterior, des de la finestra secundària acabada de crear es pot accedir al valor del botó del formulari amb la sintaxi següent:

```
window.opener.document.forms[0].BT1.value
```

El valor de la propietat opener és null si la finestra no s'ha creat des de cap altra finestra. Aquesta propietat permet comprovar si la finestra actual s'ha creat per codi o l'ha creada l'usuari:

```
if (typeof window.opener == "object") {
          //La finestra ha estat oberta per un script
}
```

1.3. Els marcs

Igual que passa amb les finestres modals, l'ús de marcs també ha generat controvèrsia en els programadors. Les característiques dels marcs els fan útils en alguns casos específics. L'aplicació més comuna és la de dividir la pàgina en un marc de contingut i un marc petit en què se situa l'índex. Amb aquesta estructura, l'usuari navega pel marc de contingut i guarda el marc d'índex de manera estable (fins i tot es pot utilitzar com a contenidor d'algunes variables que han de continuar essent-hi).

Els marcs estableixen entre ells un seguit de relacions jeràrquiques igual que els objectes del DOM. El document inicial que carrega el navegador conté l'element frameset (que defineix l'estructura de marcs de la finestra i és el pare de tots els marcs que es creen).

Els marcs són, al seu torn, un objecte Window i, per tant, es poden utilitzar les propietats i els mètodes de l'objecte Window. Respecte a l'accés als marcs, es pot implementar a partir de l'array frames[ ] de l'objecte Window, que conté una referència a cadascun dels marcs que s'han definit.

L'estructura jeràrquica pot tenir més d'un nivell, és a dir, en un marc es pot crear un frameset en què es defineixi una estructura nova de marcs, amb els quals apareixen relacions de segon nivell entre l'objecte inicial i els marcs finals. Quan es té aquest tipus de relacions, l'accés es pot fer de la manera següent:

- Encadenant les crides window.frames[1].frames[0] s'accedeix a un marc situat en dos nivells en la jerarquia.
- Amb la propietat parent s'accedeix al contenidor superior del marc que fa la crida: parent.frames[1], que fa referència al segon frame de l'estructura.
- Amb la propietat top s'accedeix al contenidor suprem, és a dir, el que és a dalt de tot de la jerarquia.

Amb els mecanismes d'accés anteriors i amb el fet que un marc és en si mateix una finestra, sembla que s'ha de poder fer qualsevol acció des d'un marc sobre la seva pròpia estructura, però no és així. Hi ha un conjunt petit de restriccions que, com és d'esperar, està relacionat amb la seguretat:

- No es pot accedir a les propietats d'un document des d'un altre marc si les pàgines pertanyen a dominis diferents.

- No es pot canviar l'URL de la barra d'adreces; aquesta mostra l'adreça del document que conté l'estructura dels marcs i no l'adreça de la pàgina d'un dels marcs.

- No es pot introduir en l'array de Favorits l'URL d'un dels documents carregat en un dels marcs; l'URL és el de la pàgina que conté l'estructura de marcs.

Tot seguit es presentaran algunes tècniques que són útils quan es treballa amb marcs en JavaScript.

1.3.1. Crear un marc buit

Es pot crear un marc que no tingui contingut sense haver de crear un document HTML en blanc que es carregui en el marc en qüestió. El mecanisme és molt simple, tal com s'aprecia en l'exemple:

```
<html>
<head>
<script type="text/JavaScript">
function marcBuit(){
       return "<html><body></body></html>";
}
</script>
</head>
<frameset rows= "50,*">
      <frame name="frame1" id="frame1" src="menu.html">
      <frame name="frame2" id="frame2"
             src="JavaScript:parent.marcBuit()">
</frameset>
</html>
```

Realment, la tècnica consisteix a substituir l'adreça del fitxer HTML per una funció que crea una pàgina HTML en blanc. Encara més, es pot crear la pàgina de manera dinàmica des de la pròpia funció marcBuit() amb algun contingut.

1.3.2. Assegurar que una pàgina carrega l'estructura de marcs

Un dels inconvenients derivats de l'ús de marcs es produeix quan en una estructura de marcs, en un marc concret, es torna a carregar una altra estructura de marcs:

```
if (top.location.href == self.location.href) {
         top.location.href ="conjuntMarcs.html";
}
```

El codi compara els URL de les finestres referides com a *top* i *self*: si són els mateixos, vol dir que el document és l'únic que es carrega en la finestra del navegador i, per tant, l'estructura de frames ocupa la pàgina inicial del navegador i no està carregada en un marc.

Tal com es veu, en el codi anterior conjuntMarcs.html és la pàgina que conté l'estructura de marcs i el nom del fitxer s'ha de canviar en cada script.

1.3.3. Canviar el contingut d'un marc des d'un altre

Des d'un marc es pot canviar el contingut d'un altre marc canviant l'URL que indica la pàgina web que està carregada en el marc. Per a això, es pot assignar l'URL a la propietat location.href del marc fill amb la sintaxi següent:

```
parent.altreFrame.location.href = "novaPàgina";
```

en què altreFrame és el nom o id del marc al qual es vol modificar el contingut.

És a dir, si no es vol modificar el marc carregant una pàgina HTML nova sinó a partir de l'escriptura des del mateix script. Això es pot fer perquè un marc és en si mateix una finestra window:

```
parent.altreFrame.document.write = "Introduïm aquí el codi HTML que vulguem";
```

Si fa falta canviar simultàniament el contingut de dos marcs o més, es pot implementar d'una manera senzilla a partir d'una funció:

```
function carregaMarcs(url1, url2) {
        parent.altreFrame1.location.href = url1;
        parent.altreFrame2.location.href = url2;
        return false;
}
```

Per tant, es pot cridar la funció anterior amb els dos paràmetres que indiquen les pàgines web que es carregaran en cadascun dels marcs. La funció es pot variar a gust del programador per a introduir més paràmetres (es poden passar com a paràmetre els noms dels frames) i fins i tot es poden modificar perquè carreguin el contingut HTML a partir de codi JavaScript.

1.3.4. Substituir una estructura de marcs per una pàgina

A vegades, hi ha carregada una estructura de marcs en una pàgina web i es vol substituir aquesta estructura per una pàgina simple i, d'aquesta manera, eliminar l'estructura de marcs. Per a portar a terme aquesta substitució, s'ha d'assegurar que l'assignació de l'URL es fa a la finestra del navegador que conté l'estructura de marcs inicial. Per a això, es pot utilitzar la propietat top de l'objecte Window:

```
top.location.href = "novaPàgina.html"
```

Si es tracta d'una estructura de marcs senzilla, es pot utilitzar parent en comptes de top, però fent servir aquest darrer s'assegura que s'accedeix a la finestra pare de totes.

1.3.5. Evitar que una pàgina es carregui en un marc

A vegades, es veu que un lloc web es carrega en un marc d'un altre lloc web i, per tant, apareix desconfigurat, amb scrolls laterals, de manera que és bastant incòmode.

L'efecte anterior es pot evitar inserint codi JavaScript en el head de la pàgina d'inici, que comprova que no es carregui la pàgina en una estructura de marcs. El codi és el següent:

```
if (top != self) {
```

```
            top.location.href = location.href;
    }
```

De fet, el codi anterior es pot utilitzar per a evitar que una estructura de marcs es carregui en un marc d'una altra estructura de marcs. Per a això, el codi anterior només s'hauria de carregar en la pàgina que defineix l'estructura de marcs.

1.4. Navegació i menús

En aquest subapartat, es tindran en compte una sèrie de característiques especials que té el web:

a) S'estudiaran algunes **utilitats de l'objecte Location**, que permeten controlar alguns aspectes de la navegació. En aquest sentit, les polítiques de seguretat implantades deixen poc marge de maniobra.

b) S'estudiaran diverses **tècniques que permeten el pas d'informació entre diverses pàgines web**; concretament, es tracta de la manera de passar dades fent servir galetes, marcs i l'URL.

c) S'estudiaran les **tècniques que permeten de crear menús de navegació més complexos** en la pàgina web.

1.4.1. Carregar una pàgina nova o un enllaç intern

Des de l'script, es vol forçar la càrrega d'una pàgina web nova, és a dir, es vol forçar la navegació en la destinació següent (segurament com a conseqüència d'un esdeveniment d'usuari o de navegador). Per a carregar la pàgina nova a la finestra del navegador o marc actual, simplement s'assigna l'URL a la propietat location.href:

```
location.href = "http://www.uoc.edu";
```

Si es vol anar a un enllaç intern de la pàgina, es pot assignar la cadena del nom de l'enllaç (és a dir, el valor assignat a l'atribut name d'un element a o a l'atribut id de qualsevol element) a la propietat location.hash:

```
location.hash = "fase3";
```

Si en comptes de carregar la pàgina nova amb el mètode location.href es fa amb el mètode location.replace(), s'aconsegueix que la pàgina inicial no surti en l'historial de navegació. Això és perquè el mètode replace no solament modifica la pàgina que hi ha a location, sinó també la que hi ha en l'array de l'historial, de manera que l'anterior és substituïda per la nova i deixa de sortir.

A vegades, s'ha de navegar a una destinació a partir d'una tria que s'ha fet en una llista desplegable de valors. Aquest tipus de navegació necessita les tècniques següents:

- Cada opció de la llista desplegable ha d'emmagatzemar en la propietat value la destinació associada al valor.

- S'ha d'associar un controlador d'esdeveniments a l'objecte Select que llanci la funció que realment obre la destinació nova seleccionada.

- S'ha de crear la funció que manejarà l'esdeveniment anterior i que s'encarregarà de carregar la pàgina nova al navegador.

El codi següent és un exemple de la tècnica plantejada. En primer lloc, es mostra el codi HTML de la llista desplegable:

```
<select name="destinacions" id="destinacions" onchange="navega(this)">
       <option value = "">Tria la teva destinació:</option >
       <option value="http://www.barbados.com/">Illes Barbados</option >
       <option value="http://www.formentera.es">Formentera</option>
</select>
```

Es pot veure en la primera línia com s'ha associat el controlador d'esdeveniments sobre l'objecte Select utilitzant l'esdeveniment onchange (que s'executa quan s'executa una selecció) i que crida la funció navega que es presenta tot seguit:

```
function navega(eleccio){
       var url = destinacions.options[destinacions.selectedIndex].value;
       if (url) {
                 location.href = url;
       }
}
```

Si es vol utilitzar la sintaxi del DOM estàndard, en primer lloc, es pot assignar l'esdeveniment a l'objecte Select de la manera següent:

```
document.getElementById("destinacions").onchange = navega;
```

Nota

S'ha de tenir en compte que aquesta assignació s'ha de fer després de la creació de l'objecte Select "destinacions", ja que en cas contrari s'estaria fent referència a un objecte que encara no existeix.

i, en segon lloc, es pot reescriure la funció controladora de manera que utilitzi l'objecte Event i que sigui compatible tant amb el model d'objectes del DOM estàndard com amb IE de la manera següent:

```
function navega(esdeveniment){
   esdeveniment=(esdeveniment) ? esdeveniment : ((event) ? event : null);
   if (esdeveniment) {
      var element = (esdeveniment.target) ? esdeveniment.target :
      ((esdeveniment.srcElement) ? esdeveniment.srcElement : null);
      if (element && element.tagName.toLowerCase() == "select" && element.value) {
         location.href = element.value;
      }
   }
}
```

1.4.2. Pas de variables entre pàgines

Una primera tècnica que permet el pas de valors o variables entre diverses pàgines utilitza una combinació entre les funcions de maneig de galetes i els esdeveniments onunload i onload de l'objecte document.

Vegeu també

Les funcions de maneig de galetes es tracten en l'apartat 4 del mòdul "Orientació a objectes en JavaScript" d'aquesta assignatura.

Tot seguit, es presenta un exemple en què s'emmagatzema el valor d'un camp d'un formulari i el guarda en una galeta durant trenta dies:

```
<script type="text/JavaScript">
function guardaDades() {
       var dades = document.forms[0].nom.value;
       assignaCookie("Nom",dades,30);
}
</script>
```

de manera que, quan es tanca el document, es crida la funció guardaDades:

```
<body onunload = "guardaDades()">
```

De la mateixa manera, en el document en què vol recuperar els valors, ha de cridar en l'esdeveniment onload la funció que recupera els valors de la galeta:

```
<script type = "text/JavaScript">
function recuperaValors() {
       extreuCookies();
       var nom = cookies["Nom"];
}
</script>
```

Si se situa el codi anterior en el head de la pàgina, en la línia de definició del cos s'ha de fer la crida de la funció recuperaValors quan es produeix l'esdeveniment load:

```
<body onload = "recuperaValors()">
```

El problema principal que pot sorgir quan s'utilitzen galetes són les opcions possibles en la configuració de seguretat del navegador del client.

Una alternativa a l'ús de galetes és l'emmagatzemament dels valors utilitzant les característiques de l'estructura de marcs; concretament, l'estructura Window representa l'estructura de marcs que queda fixa mentre els documents entren als marcs fill i en surten. Aquesta finestra superior és capaç d'emmagatzemar dades JavaScript de tota mena en variables globals.

Es pot utilitzar el controlador d'esdeveniments unload de la pàgina d'un dels marcs per a emmagatzemar les dades en una variable global de la finestra que conté l'estructura dels marcs.

Tot seguit es proposa un exemple de la tècnica que s'ha plantejat:

```
<script type = "text/JavaScript">
        function emmagatzemaDades() {
                top.nom = document.forms[0].nom.value;
}
</script>
```

de manera que, en el cos de la pàgina que necessita emmagatzemar els valors, s'insereix el codi següent:

```
<body onunload="emmagatzemaDades()">
```

En el document que ha de recuperar els valors s'insereix el codi següent:

```
<script type="text/JavaScript">
function llegeixValors(){
       if (typeof top.nom != "undefined") {
                document.forms[0].nom.value = top.nom;
       }
}
</script>
```

i en la definició del cos s'insereix la referència al controlador d'esdeveniments:

```
<body onload="llegeixValors()">
```

Els problemes principals de la tècnica anterior es deuen, en primer lloc, al rebuig d'usar marcs que tenen els programadors i, en segon lloc, al fet que aquesta tècnica funciona mentre la pàgina que conté l'estructura de marcs es manté en la pàgina carregada; per tant, si en algun moment es força una recàrrega de la pàgina utilitzant la funció de recàrrega del navegador, el valor de les variables emmagatzemades es perd.

Per acabar aquest subapartat, una de les tècniques més utilitzades es basa en el pas de les variables mitjançant l'URL que obre la pàgina nova. Aquestes dades es passen com una cadena de cerca que s'afegeix a l'URL de la pàgina següent i s'inclou un script a la pàgina posterior que s'encarrega de llegir la cadena de cerca i recuperar les dades.

Tot seguit es pot veure la tècnica en un exemple senzill:

```
<script type="text/JavaScript">
function navega(url) {
       var dadesApassar = document.forms[0].nom.value;
       location.href = url + "?" + encodeURIComponent(dadesApassar);
}
</script>
```

Per tant, la funció anterior obre la pàgina següent en el navegador, però afegeix a l'URL una cadena de dades. Aquesta funció s'ha de cridar des dels enllaços que facin falta; per tant, s'ha d'implementar en l'esdeveniment onclick dels enllaços:

```
<a href="paginaFinal.html" onclick="navega('paginaFinal.html'); return false">...</a>
```

D'aquesta manera, des de la pàgina receptora s'ha d'implementar un script que sigui capaç de localitzar i tractar la cadena que s'ha passat. Aquesta funció s'ha de cridar en l'esdeveniment load de l'objecte Document perquè els valors estiguin disponibles a l'hora de carregar la pàgina:

```
<script type="text/JavaScript">
function llegeixDades() {
       var buscaCadena = decodeURIComponent(location.search.substring(1,
    location.search.length));
       if (buscaCadena.length > 0) {
               document.forms[0].userName.value = buscaCadena;
       }
}
</script>
```

de manera que en l'esdeveniment load de Document es referencia la funció anterior:

```
<body onload="llegeixDades()">
```

L'avantatge principal del mecanisme anterior és que és del tot independent del navegador, perquè tots els navegadors respecten el pas de cadenes de cerca en l'URL; per tant, de les tres tècniques que s'han vist respecte al pas de variables, aquesta és l'òptima o preferida per la majoria de programadors.

2. Formularis dinàmics

Els formularis en el web van ser el primer objectiu del llenguatge de programació JavaScript i avui dia continuen fent un ús intensiu de scripts, amb l'objectiu d'augmentar la interacció immediata amb l'usuari.

D'aquesta manera, la validació de formularis amb JavaScript continua sent útil, perquè accelera la correcció d'errors, però no implica que la validació que n'ha de fer el servidor no s'hagi de fer, perquè aquesta revisió és inevitable.

Respecte a la validació de formularis, s'han de tenir en compte dos tipus d'estratègia en el procés de validació: en temps real i quan s'envia el formulari.

a) La **validació en temps real** es basa a detectar activitat i a cridar una funció quan hi ha aquesta activitat.

Validació en temps real

Per exemple, l'esdeveniment onChange es produeix quan canvia el contingut d'un quadre de text, i aquest esdeveniment pot cridar una funció que validi el text nou que s'ha introduït.

L'avantatge principal de la validació en temps real és que l'usuari encara té fresca la informació que ha introduït en el camp, és a dir, quan acaba d'introduir el contingut d'un camp, si hi ha hagut un error en aquest contingut, es pot avisar immediatament l'usuari.

b) La **validació quan s'envia el formulari** es basa a comprovar cadascun dels components del formulari just abans d'enviar-lo al servidor. D'aquesta manera, a partir de l'esdeveniment submit de l'objecte Form es llança l'script que porta a cap les validacions.

En aquest apartat s'estudiaran algunes tècniques que permeten de dur a terme les accions bàsiques en el maneig de formularis. Concretament, es veuran tècniques relacionades amb les accions següents:

- Controlar el cursor i el focus.
- Controlar els camps del formulari.
- Validar camps.

2.1. Controlar el cursor i el focus

En aquest subapartat es veuran diverses tècniques que milloraran la usabilitat del formulari, que, si bé es poden considerar detalls petits, ajuden els usuaris de manera gairebé transparent.

2.1.1. Enviar el cursor a un camp determinat

Quan s'entra en el web de Google, l'usuari només ha d'introduir les cadenes clau per a fer la cerca i prémer la tecla Intro. Es tracta d'un procés molt simple i és un dels factors de l'èxit de Google, però darrere d'aquest funcionament hi ha codi que fa que el camp de text de cerca rebi el focus quan s'obre la pàgina d'inici.

El codi que envia el cursor al primer camp de text del formulari és molt simple. En l'exemple que es presenta tot seguit, el codi s'enllaça al controlador de l'esdeveniment onload de l'objecte Document, perquè d'aquesta manera, quan s'ha carregat la pàgina web, la primera acció que es duu a terme és la de moure el focus al camp de destinació:

```
<body onload="document.formulari.camp.focus()">
```

Si el camp de text té un text predeterminat, és una ajuda que aquest text aparegui seleccionat al començament, perquè d'aquesta manera, quan l'usuari pitja qualsevol tecla, se substitueix el contingut predeterminat pel del text nou (i s'evita així el procés d'eliminar el text predeterminat).

La selecció del text en un camp de text es fa amb el mètode select(). Així, doncs, el codi anterior queda de la manera següent:

```
<body onload="document.formulari.camp.focus();
document.formulari.camp.select() " >
```

Per tant, de primer, se situa el focus i, després, se selecciona el text que conté el camp.

Si el camp de text no supera el procés de validació, normalment es torna el cursor al camp erroni seleccionant el text erroni, amb l'objectiu que l'usuari sobreescrigui el text amb el valor correcte.

Els mètodes que permeten aquestes accions són focus() i select(), però s'han de cridar en un cert punt del codi (des de la funció de validació). Per a això, tot seguit es planteja una funció que encapsula aquests dos mètodes:

```
function assignaFocus(nomForm, nomCamp) {
       var element = document.forms[nomForm].elements[nomCamp];
       element.focus();
       element.select();
}
```

Sobre la funció anterior, s'ha de tenir en compte que és vàlida per a camps de tipus text, ja que en el cas de botons d'opció o llistes desplegables la selecció del contingut no té cap sentit.

2.1.2. Avançar el focus amb la tecla Retorn

El moviment entre els diversos camps del formulari es fa amb la tecla de tabulació, però l'usuari està més habituat a fer servir la tecla Retorn, de manera que tot seguit es presentarà una tècnica que permet de navegar pels diversos camps del formulari utilitzant la tecla Retorn.

S'utilitza l'esdeveniment keypress en cada camp des del qual ha d'avançar el focus a un altre camp del formulari. El controlador d'esdeveniments crida la funció seguentFocus(), que es presenta tot seguit:

```
<input type="text" name="camp1" id="camp1" onkeypress="return seguentFocus(this.form, 'camp2',
  event)">
```

La funció seguentFocus() fa el següent:

- En primer lloc, s'ha de comprovar que la tecla pitjada és Retorn, que té els codis ASCII 13 i 3.
- En segon lloc, aplica el focus al camp següent que s'ha passat com a paràmetre de la funció:

```
function seguentFocus(form, seguentCamp, evt) {
       evt = (evt) ? evt : event;
       var codCaracter = (evt.charCode) ? evt.charCode :
              ((evt.which) ? evt.which : evt.keyCode);
       if (codCaracter == 13 || codCaracter == 3) {
              form.elements[seguentCamp].focus();
              return false;
       }
       return true;
}
```

Una de les potencialitats de l'ús de funcions com l'anterior és la possibilitat de poder adaptar la introducció de les dades en un ordre diferent del que se segueix en la distribució dels camps en la pàgina HTML.

2.1.3. Enviar formularis amb la tecla Retorn

En aquest subapartat, es presentarà un seguit de tècniques que permeten, juntament amb el que s'ha presentat en el subapartat anterior, simular un formulari d'aplicació, és a dir, un formulari en què el desplaçament pels diversos camps es fa amb la tecla Retorn i, quan en el darrer camp de text es prem Retorn, es produeix l'esdeveniment de tramesa del formulari.

Per tant, en el darrer camp del formulari s'ha de cridar una funció que s'acabi amb una crida al mètode submit() de l'objecte Form. Tot seguit es presenta un exemple d'aquesta funció:

```
<html>
<head>
</head>
<body>
<form id="formulari">
    <P>
    <LABEL for="nom">Nom: </LABEL>
              <INPUT type="text" id="nombre"><BR>
    <LABEL for=" cognom">Apellido: </LABEL>
              <INPUT type="text" id="cognom"><BR>
    <LABEL for="correuelectronic"> correu electrònic: </LABEL>
              <INPUT type="text" id="email"><BR>
    <INPUT type="radio" name="sexe" value="Home"> Home<BR>
    <INPUT type="radio" name="sexe" value="Dona" onkeypress="return tramesaAmbEnter(event)">
     Dona<BR>
    </P>
 </form>
<script type="text/javascript">
function tramesaAmbEnter(evt){
       evt = (evt) ? evt : event;
       var destinacio = (evt.target) ? evt.target : evt.srcElement;
       var codCaracter = (evt.charCode) ? evt.charCode : ((evt.which) ? evt.which : evt.keyCode);
       if (codCaracter == 13 || codCaracter == 3) {
                        formulari.submit();
               }
}
</script>
</body>
</html>
```

Així, en el darrer camp del formulari es fa l'assignació següent:

```
onkeypress="return tramesaAmbEnter(event)"
```

Un detall que s'ha de considerar és que l'objecte Form ha de tenir definit el controlador d'esdeveniments onsubmit="return false", ja que d'aquesta manera s'obliga a fer que només el mètode submit() de l'script sigui capaç de portar a cap la tramesa.

2.1.4. Tabulació automàtica

Hi ha formularis en què s'introdueix alguna informació en un conjunt combinat de camps de text, que es caracteritzen per una longitud definida en cadascun d'aquests camps.

En aquest tipus de formularis, quan l'usuari ha acabat d'introduir els nombres en un dels camps, automàticament el cursor se situa en el camp de text següent. L'exemple que es presenta tot seguit mostra una de les maneres en què es pot implementar aquest efecte i ho fa en un formulari amb quatre camps de mida màxima 4:

```
<html>
<head>
</head>
<body>
<form id="formulari" onsubmit="return false">
Número de la targeta:
<input type="text" id="nt1" size="5" maxlength="4" onkeyup="mouFocus(this, 'nt2', event)" />
<input type="text" id="nt2" size="5" maxlength="4" onkeyup="mouFocus(this, 'nt3', event)" />
<input type="text" id="nt3" size="5" maxlength="4" onkeyup="mouFocus(this, 'nt4', event)" />
<input type="text" id="nt4" size="5" maxlength="4" />
 </form>
<script type="text/javascript">

function mouFocus(camp, seguent, evt){
       evt = (evt) ? evt : event;
       var codCaracter = (evt.charCode) ? evt.charCode : ((evt.keyCode) ? evt.keyCode :
                         (( evt.which) ? evt.which : 0));
       if (codCaracter >31 &amp;amp;&amp;amp; camp.value.length == camp.maxLength) {
               camp.form.elements[seguent].focus();
       }
}

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

El controlador de la funció keyup crida la funció mouFocus(), que s'encarrega de passar el focus al camp següent:

Es pot veure que el darrer camp no crida la funció mouFocus(). Es podria implementar perquè saltés el focus al camp següent del formulari, simplement passant-hi el valor en el segon paràmetre de la funció.

2.2. Control dels camps del formulari

Hi ha dos mecanismes per a evitar o bloquejar l'accés a certs camps d'un formulari:

a) Desactivar un camp, de manera que apareix visible a l'usuari, però no és actiu, la qual cosa implica que no s'hi pot introduir text ni que aquest text es pot modificar.

b) Ocultar un camp. En aquest cas, el camp desapareix del formulari i no és visible per a l'usuari final, encara que sí que hi és realment.

En aquest subapartat, s'estudiaran els dos mecanismes que permeten desactivar i ocultar camps amb uns exemples senzills.

2.2.1. Desactivar camps

La desactivació de camps de formulari s'implementa a partir de la propietat disabled, que admet un valor booleà:

```
document.formulari.campText.disabled = true;
```

El codi anterior desactiva el camp campText, de manera que l'usuari no hi pot accedir i, en general, mostra un aspecte ombrat de color gris. Encara que els scripts poden llegir i escriure valors en els camps desactivats, aquests camps no s'envien al servidor si no s'activen.

L'activació del camp anterior es fa amb la sentència següent:

```
document.formulari.campText.disabled = false;
```

2.2.2. Ocultar i mostrar controls

Es poden ocultar alguns camps d'un formulari, de manera que només es mostrin si es compleix una condició determinada, com, per exemple, que l'usuari hagi seleccionat alguna opció en algun camp de selecció o marqui una casella de selecció.

Tot seguit, es presenta un exemple de formulari que té un conjunt de camps ocults, que es mostren a l'usuari final quan aquest usuari respon afirmativament a la tercera pregunta:

```
<form name="qüestionari">
```

```
...
<p>3. Vols conèixer els models nous?<br />
<input type="radio" id="resp0" name="resposta" onclick="gestDecision(event)" />No
<input type="radio" id="resp1" name="resposta" onclick="gestDecision(event)" />Sí
<div id="coneixerModels" style="display:none; margin-left:20px">
</p>
<p>
3a. Quina és la teva marca preferida?
<select name="models">
       <option value="">Tria'n un:</option >
       <option value="1">Ferrari</option>
       <option value="2">Mercedes</option>
       <option value="3">BMW</option>
</select>
</p>
</div>
<p>4. Quants anys té el teu vehicle actual?
<select name="edatVehicle">
       <option value="">Tria una opció:</option>
       <option value="1">Menys de 3 anys:</option>
       <option value="2">Entre 3 i 7 anys</option>
       <option value="3">Més de 7 anys</option>
</select>
</p>
...
</form>
```

En el codi anterior s'ha utilitzat una capa per a introduir-hi la llista desplegable de les marques. D'aquesta manera, en principi, la llista queda oculta quan s'estableix l'atribut display a none de la capa, no es mostra al començament i és el controlador de l'esdeveniment click de les opcions radio el que canviarà l'estat de la capa:

```
function gestDecision(evt) {
       evt = (evt) ? evt : event;
       var camp = (evt.target) ? evt.target : evt.srcElement;
       var capa = document.getElementById("coneixerModels");
       if (camp.id =="resp1") {
              capa.style.display = "block";
       } else {
              capa.style.display = "none";
       }
}
```

Tal com es veu en la funció, s'utilitza l'objecte Event per a capturar el camp i, tot seguit, s'assigna a la variable *capa* el contenidor dels camps, de manera que si el camp seleccionat és resp1, es mostra la capa oculta o, si no, els camps continuen ocults.

2.2.3. Permetre només un tipus de dades en un camp

En certes aplicacions web, és interessant restringir el tipus de dades que es poden introduir en un camp de text. Més endavant, es plantegen exemples que defineixen una funció, que es crida des de l'esdeveniment keypress.

En primer lloc, la funció següent restringeix l'entrada de valors als números entre el 0 i el 9 i, a més, permet introduir valors ASCII més petits que el 32, ja que entre aquests valors hi ha els caràcters alfanumèrics que inclouen les tecles Retrocés 8, Tabulador 9 i Retorn 13:

```
function nomesNumeros(evt){
        evt = (evt) ? evt : event;
        var codCaracter = (evt.keyCode) ? evt.keyCode : (evt.which) ?
        evt.which : 0;
        if (codCaracter >31 && (codCaracter <48 || codCaracter >57)) {
                alert("Introdueix només valors numèrics!!!");
                return false;
        }
        return true;
}
```

En l'exemple següent es planteja una funció que valida la introducció de diverses lletres. Per a això, s'ha de tenir en compte el fet que les majúscules i minúscules tenen codis diferents i, a més, no són correlatius. La funció següent comprova la introducció de lletres, tant en majúscules com en minúscules:

```
function nomesLletres(evt){
       evt = (evt) ? evt : event;
       var codCaracter = (evt.keyCode) ? evt.keyCode : (evt.which) ?
       evt.which : 0;
       if (codCaracter >31 && (codCaracter <65 || codCaracter >90) &&
       (codCaracter <97 || codCaracter >122)) {
              alert("Introdueix només lletres!!!");
              return false;
       }
       return true;
}
```

La mateixa estructura de funció serveix per a definir una funció que obligui a entrar-hi certs valors simples, com *Y* o *N*, 0 o 1, o qualsevol combinació que es defineixi. Per exemple, la funció següent només permet d'introduir els valors *Y* o *N*, tant en majúscules com en minúscules:

```
function nomesYoN(evt){
    evt = (evt) ? evt : event;
    var codCaracter = (evt.keyCode) ? evt.keyCode : (evt.which) ? evt.which : 0;
    if (codCaracter > 31 && codCaracter != 78 && codCaracter != 89
    && codCaracter != 110 && codCaracter != 121) {
        alert("Introdueix només els valors 'Y' o 'N'");
        return false;
    }
    return true;
}
```

En un exemple com l'anterior, es pot reforçar la seguretat limitant la grandària del camp a un sol caràcter:

```
<input type="text" name="valor" size="2" maxlength="1" onkeypress="return nomesYoN(event)" /> (Y/N)
```

Tot seguit es presenta la taula de codis ASCII:

Taula 1. Codis ASCII

Caràcters no imprimibles				**Caràcters imprimibles**								
Nom	**Dec.**	**Hex.**	**Car.**	**Dec.**	**Hex.**	**Car.**	**Dec.**	**Hex.**	**Car.**	**Dec.**	**Hex.**	**Car.**
Nul	0	00	NUL	32	20	Espai	64	40	@	96	60	`
Inici de capçalera	1	01	SOH	33	21	!	65	41	A	97	61	a
Inici de text	2	02	STX	34	22	"	66	42	B	98	62	b
Final de text	3	03	ETX	35	23	#	67	43	C	99	63	c
Final de transmissió	4	04	EOT	36	24	$	68	44	D	100	64	d
enquiry	5	05	ENQ	37	25	%	69	45	E	101	65	e
acknowledge	6	06	ACK	38	26	&	70	46	F	102	66	f
Campaneta (beep)	7	07	BEL	39	27	'	71	47	G	103	67	g
backspace	8	08	BS	40	28	(	72	48	H	104	68	h
Tabulador horitzontal	9	09	HT	41	29	)	73	49	I	105	69	i
Salt de línia	10	0A	LF	42	2A	*	74	4A	J	106	6A	j
Tabulador vertical	11	0B	VT	43	2B	+	75	4B	K	107	6B	k
Salt de pàgina	12	0C	FF	44	2C	,	76	4C	L	108	6C	l
Retorn	13	0D	CR	45	2D	-	77	4D	M	109	6D	m

Caràcters no imprimibles				Caràcters imprimibles								
Nom	Dec.	Hex.	Car.	Dec.	Hex.	Car.	Dec.	Hex.	Car.	Dec.	Hex.	Car.
Shift fora	14	0E	SOTA	46	2E	.	78	4E	N	110	6E	n
Shift dins	15	0F	SI	47	2F	/	79	4F	O	111	6F	o
Escapada línia de dades	16	10	DLE	48	30	0	80	50	P	112	70	p
Control dispositiu 1	17	11	DC1	49	31	1	81	51	Q	113	71	q
Control dispositiu 2	18	12	DC2	50	32	2	82	52	R	114	72	r
Control dispositiu 3	19	13	DC3	51	33	3	83	53	S	115	73	s
Control dispositiu 4	20	14	DC4	52	34	4	84	54	T	116	74	t
neg acknowledge	21	15	NAK	53	35	5	85	55	U	117	75	u
Sincronisme	22	16	SYN	54	36	6	86	56	V	118	76	v
Final bloc transmès	23	17	ETB	55	37	7	87	57	W	119	77	w
Cancel·lar	24	18	CAN	56	38	8	88	58	X	120	78	x
Final mitjà	25	19	EM	57	39	9	89	59	Y	121	79	y
Substitut	26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
Escapada	27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
Separador arxius	28	1C	FS	60	3C	<	92	5C	\	124	7C	|
Separador grups	29	1D	GS	61	3D	=	93	5D	]	125	7D	}
Separador registres	30	1E	RS	62	3E	>	94	5E	^	126	7E	~
Separador unitats	31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

2.3. Validar camps

En aquest darrer subapartat, es mostren una sèrie de funcions que fan un conjunt de validacions sobre camps de formulari. El llançament d'aquestes funcions s'executa amb el controlador d'esdeveniment change. Per exemple:

```
<input type = "text" size = "30" id = "correuE" onchange = "esCorreuOK(this)" />
```

Tot seguit es presenten quatre funcions:

- noBuit(): comprova que el camp de text conté com a mínim un caràcter;
- esNumeric(): comprova que el contingut del camp de text és un valor numèric;
- longitud(): comprova que el camp conté exactament dotze caràcters;

- esCorreuE(): comprova que el contingut del camp té el format d'una adreça electrònica.

La funció noBuit() comprova que el camp té com a mínim un caràcter:

```
function noBuit(elem) {
	var cadena = elem.value;
	if(cadena == null || cadena.length == 0) {
		alert("Atenció, el camp és obligatori.")
		return false;
	} else {
		return true;
	}
}
```

La funció anterior fa dues validacions sobre el contingut del camp: ha de contenir un valor nul o una cadena de text buida. Es tracta de dues opcions semblants però s'han de comprovar totes dues.

L'exemple següent comprova que el valor introduït és numèric:

```
function esNumeric(elem) {
	var cadena = elem.value;
	var unDecimal = false;
	var unCaracter = 0;
	//S'assegura que el contingut és en format string
	cadena = cadena.toString();
	for (var i = 0; i < cadena.length; i++) {
		unCaracter = cadena.charAt(i).charCodeAt(0);
		// Es comprova si té un signe negatiu en el primer caràcter
		if (unCaracter == 45) {
			if (i == 0) {
				continue;
			} else {
				alert("Només el primer caracter pot ser el signe negatiu.");
				return false;
			}
		}
		// Es comprova el punt decimal
		if (unCaracter == 46) {
			 if (!unDecimal) {
				unDecimal = true;
				continue;
			 } else {
				alert("Només un punt decimal en un número.");
				return false;
			 }
```

```
                }
                //No s'admeten caràcters fora del rang 0-9
                if (unCaracter <48 || unCaracter >57) {
                        alert("Introdueix només números.");
                        return false;
                }
        }
        return true;
}
```

El refinament de la funció anterior es basa en la validació, si el primer caràcter és el signe negatiu (que marca que el número és negatiu) i hi ha un punt intercalat que indica que el número té valors decimals.

L'exemple que es presenta tot seguit comprova la longitud del contingut d'un camp de text:

```
function longitud(elem,long) {
       var cadena = elem.value;
       var long = parseInt(long);
       if (cadena.length != long) {
               alert("El camp no conté els "+ long +" caràcters requerits");
               return false;
       } else {
               return true;
       }
}
```

L'exemple anterior és molt simple –es basa en la propietat length de l'objecte String–, però pot ser molt útil en segons quins formularis.

Per acabar l'apartat es mostra un exemple de validació d'un camp que ha de contenir una adreça electrònica:

```
function esCorreuE(elem) {
       var cadena = elem.value;
       cadena = cadena.toLowerCase();
       if (cadena.indexOf("@") >1) {
               var addr = cadena.substring(0, cadena.indexOf("@"));
               var domini = cadena.substring(cadena.indexOf("@") +1, cadena.length);
               // Cal almenys un domini en la cadena de l'adreça
               if (domini.indexOf(".") == -1) {
                       alert("Verifica que el domini és correcte");
                       return false;
               }
               // Es revisa caràcter per caràcter
               for (var i = 0; i < addr.length; i++) {
```

```
                        unCaracter = addr.charAt(i).charCodeAt(0);
                        // En aquesta porció no es permet el punt
                        if ((i == 0 && (unCaracter == 45 || unCaracter == 46)) ||
                                 (i == addr.length -1 && unCaracter == 46)) {
                                 alert("Verifica la porció del nom d'usuari");
                                 return false;
                        }
                        //Es comproven caràcters vàlids (- . _ 0-9 a-z)
                        if (unCaracter == 45 || unCaracter == 46 || unCaracter == 95 ||
                               (unCaracter >47 && unCaracter <58) ||
                               (unCaracter >96 && unCaracter <123)) {
                               continue;
                        } else {
                               alert("Verifica la porció del nom d'usuari.");
                               return false;
                        }
                }
                for (i = 0; i < domini.length; i++) {
                        unCaracter = domini.charAt(i).charCodeAt(0);
                        if ((i == 0 && (unCaracter == 45 || unCaracter == 46)) ||
                                 ((i == domini.length -1 || i == domini.length -2) &&
                                 unCaracter == 46)) {
                                 alert("Verifica el domini de l'adreça.");
                                 return false;
                        }
                        if (unCaracter == 45 || unCaracter == 46 || unCaracter == 95 ||
                                 (unCaracter > 47 && unCaracter < 58) ||
                                 (unCaracter > 96 && unCaracter < 123)) {
                                 continue;
                        } else {
                                  alert("Verifica el domini de l'adreça.");
                                  return false;
                        }
                }
                return true;
        }
        alert("L'adreça de correu no és correcta. Si us plau, comproveu-ho.");
        return false;
}
```

La funció anterior es basa en les comprovacions següents:

- En primer lloc, es comprova que la @ és a la cadena de text; si no hi és, no es continua la validació.

- En segon lloc, es comprova que el . que defineix el domini de primer nivell és a la cadena de text; si no hi és, no es continua la validació.

- En tercer lloc, es recorre caràcter per caràcter la primera part de la cadena de text que correspon a l'usuari del compte de correu, i es comprova que la formen caràcters vàlids en un compte de correu.

- En quart lloc, es recorre la part corresponent del domini del compte de correu i es valida que té un domini i un subdomini.

Es tracta d'una validació de correu electrònic molt senzilla. S'hi pot afegir més complexitat introduint-hi més comprovacions, com la validació que els dominis siguin reals a partir d'una llista. Aquesta funció pot ser cridada a partir de l'esdeveniment onblur del camp que ha de contenir l'adreça electrònica.

3. Posicionament dinàmic

La distribució dels elements en una pàgina web es regeix per l'ordre en què s'escriuen en el codi HTML. Fins que no van aparèixer les capes, l'única manera de posicionar elements era fent servir taules. En qualsevol cas, quan s'ha carregat la pàgina, els elements que la formen són estàtics.

L'especificació CSS introdueix certs atributs que permeten posicionar objectes a la pàgina (també coneguda com a *CSS-P* o *DHTML*). Amb l'ús de DHTML o CSS-P es trenca la limitació anterior, ja que:

- Es poden modificar els atributs de posicionament, quan s'ha carregat la pàgina, utilitzant esdeveniments o scripts que ha definit el programador.
- Es poden encavalcar elements HTML a la pàgina web.
- Es poden ocultar i mostrar elements HTML a la pàgina web.
- Es poden moure elements i fins i tot fer algunes animacions.

Aquestes característiques es van introduir a partir de les versions 4.x dels navegadors, cosa que va generar problemes d'interpretació entre els dos navegadors principals d'aquella època. De fet, es tractava de l'àrea en què hi havia una divergència més grossa.

3.1. Propietats CSS de posicionament

Les propietats de posicionament permeten que un element HTML es posicioni en qualsevol coordenada del document. Se solen anomenar **capa** les àrees de posicionament que s'acostumen a crear amb l'etiqueta HTML `<div>`, les propietats o els atributs de la qual es presenten tot seguit:

1) position

Aquesta propietat indica el tipus de posicionament utilitzat en l'element. Els valors disponibles per a aquesta propietat són els següents:

- static: és el valor predeterminat i indica que l'element no es pot posicionar.
- absolute: aquest valor fa que l'element quedi fora del flux d'HTML i la posició queda relativa a la cantonada superior esquerra de la pàgina. Els objectes poden quedar superposats sobre altres elements del flux de la pàgina.

- relative: aquest valor fa que l'element ocupi el mateix lloc que ocuparia si no estigués posicionat i es desplaça de la posició el valor que s'hi indica.

2) top i left

Aquests dos valors defineixen la posició de la capa, de manera que es defineix des de la part superior esquerra de la regió que l'inclou (normalment la regió coincideix amb la finestra del navegador).

3) height i width

Defineixen la grandària de la capa a partir de l'alçària i amplària, que s'ha de mesurar en píxels o en percentatges.

4) clip

La propietat clip permet definir una àrea rectangular dins un element. Tot el que queda dins aquest clip és visible i el de fora, invisible.

La regió es defineix assenyalant les propietats top, right, bottom i left del rectangle, que s'han d'indicar en píxels i han d'agafar com a referència l'element contenidor, i no la pàgina completa. La sintaxi és la següent:

```
clip: rect(top right bottom left);
```

Vegem-ne un exemple d'ús:

```
<div id="tallada" style="position:absolute; left:100; top:250; width: 400; height:
100; background-color: #CC9966; layer-background-color: #CC9966; border: 1px
none #000000; clip: rect(30 330 80 130)" >
```

S'ha de tenir en compte que l'espai reservat per a tota la capa es manté inalterat, encara que no sigui visible. A més, es pot moure el clip modificant els paràmetres de posicionament de la capa sobre la qual és definit per a aconseguir l'efecte que es vol.

5) visibility

L'atribut visibility també té un significat bastant obvi: un element amb la regla visibility: hidden no és visible.

Els valors que pot agafar la propietat visibility són els següents:

- hidden: oculta la capa.
- visible: mostra la capa, si és visible.
- inherit: indica si la visibilitat s'hereta del contenidor superior.

La utilitat de tenir un element invisible no quedaria clara si no fos pel fet que la visibilitat de l'element la poden modificar els esdeveniments o un script que ha definit l'usuari. Una bona part dels menús desplegables que s'han fet en JavaScript es basen en aquesta propietat.

6) z-index

Defineix l'ordre d'encavalcament de les àrees que ocupen els elements, de manera que una capa amb valor z-index més gran se situa sobre una capa amb un valor z-index més petit. Si no s'assigna el valor z-index, l'ordre d'encavalcament el defineix el mateix ordre de creació de les capes, i el darrer element que s'ha definit és el que queda en primer pla.

3.2. Una mica d'història: navegadors 4.x

Amb l'aparició de CSS-P en els navegadors 4.x, la major part dels problemes van venir del fet que Netscape 4 va introduir una etiqueta propietària `<layer>` que no era compatible amb IE4 i que no es va fer servir en les versions posteriors del navegador.

3.2.1. Capes a Netscape 4

Tal com s'ha dit, Netscape va introduir una etiqueta propietària `<layer>` que proporcionava mecanismes de posicionament semblants als de l'estàndard CSS. La taula següent mostra les propietats de l'objecte Layer relacionades amb el posicionament dinàmic.

Taula 2. Propietats de l'objecte Layer

Propietat	Descripció
clip	Referència a l'objecte Clip de retallada de la capa. Aquest objecte es defineix amb les propietats top, right, bottom i left.
left	Posició de la coordenada *x* de la capa.
top	Posició de la coordenada *y* de la capa.
pageX	Posició de la coordenada *x* de la capa relativa a la pàgina.
pageY	Posició de la coordenada *y* de la capa relativa a la pàgina.
visibility	Indica la visibilitat de la capa, i admet els valors show i hide.
zIndex	Indica l'ordre d'encavalcament de la capa.

Tot seguit es presenta un exemple de capa creada per a Netscape 4.x:

```
<layer name="capaNN" pageX="100" pageY="100" width="100" height="50"
bgcolor="#ffff99">Capa creada amb layer! </layer>
```

L'etiqueta `<layer>` va néixer i morir en les versions 4 del Netscape; com s'ha dit, no la van suportar ni les versions posteriors del navegador ni cap navegador de la competència.

D'altra banda, just abans de la presentació del Netscape 4, el navegador va adoptar suport per a les etiquetes `<div>` posicionades; per tant, la capa anterior també es pot crear amb la sintaxi següent:

```
<div id="capaNN" style="position: absolute; top: 100px; left: 100px; width:
100px; height: 50px; background-color: #ffff99">Capa creada amb div!</div>
```

De fet, l'accés a les capes creades amb l'etiqueta `<div>` es feia utilitzant la col·lecció layers[ ]. Per exemple, l'accés a la capa anterior es feia amb la sintaxi següent:

```
document.layers['capaNN'];
```

Sense cap dubte, però, la característica que va fer perdre terreny a Netscape va ser que no reflectia d'una manera dinàmica els canvis produïts en la propietat style, la qual cosa implicava que no es podien modificar les propietats d'una manera dinàmica. És a dir, si es modificava un atribut de la propietat style de l'etiqueta `<div>` o qualsevol propietat de l'etiqueta `<layer>` amb un script, aquesta modificació no es reflectia a la pàgina.

D'aquesta manera, el posicionament dinàmic en Netscape 4.x no es basava a modificar les propietats top i left sinó en uns mètodes implementats a l'objecte Layer. Tot seguit es veuen els mètodes principals:

- moveBy(incrementX,incrementY): provoca un desplaçament de la capa a partir dels valors passats com a paràmetres;
- moveTo(x,y): situa la capa en les coordenades indicades en els paràmetres *x* i *y*;
- moveToAbsolute(x,y): situa la capa en les coordenades absolutes indicades en els paràmetres *x* i *y*;
- moveAbove(capa): situa la capa actual sobre la capa passada com a paràmetre;
- moveBelow(capa): situa la capa actual sota la capa passada com a paràmetre;
- resizeBy(incrementX,incrementY): incrementa la grandària de la capa amb els valors passats en els paràmetres incrementX i incrementY;

- resizeTo(alt,ample): especifica la grandària de la capa amb els valors passats en els paràmetres alt i ample.

3.2.2. Capes a Internet Explorer 4

Les capes a Internet Explorer es defineixen utilitzant l'etiqueta `<div>`, on es defineix la propietat style amb cadascun dels atributs que té. Vegem-ne tot seguit un exemple:

```
<div id="capaIE" style="position: absolute; top: 100px;
left: 100px; width: 100px; height: 100px; background-color: "#ffff99">Capa creada amb div!</div>
```

La diferència principal del navegador Internet Explorer respecte a Netscape es basava en el fet que Internet Explores sí que actualitzava la presentació davant de qualsevol canvi de la propietat style d'una manera dinàmica. Així, doncs, des d'un script es podia modificar qualsevol atribut de l'etiqueta `style` i aquest atribut es representava a la pàgina de manera immediata.

Per tant, a Internet Explorer les modificacions de les capes no es feien a partir de mètodes de l'objecte sinó a partir de modificar les propietats de l'objecte Style. Per exemple, a partir de la capa anterior, el codi següent:

```
document.all['capaIE'].style.left += 10;
document.all['capaIE'].style.backgroundColor = 'orange';
```

fa un desplaçament de la capa a la dreta de 10 píxels i modifica el color de fons d'aquesta capa.

Una de les característiques que cal tenir en compte quan es fan servir els atributs CSS en JavaScript és que la sintaxi dels atributs canvia quan aquests atributs s'utilitzen des de JavaScript. És a dir, en l'exemple anterior, en la creació de l'etiqueta `<div>` s'ha especificat el color de fons amb l'atribut background-color, mentre que en el codi script aquesta propietat de l'objecte Style s'anomena *backgroundColor*.

La regla general consisteix a convertir els atributs compostos de CSS en una única paraula en el DOM, utilitzant el que es coneix com a *gep de camell* (la segona paraula canvia la primera lletra a majúscules i elimina el guió que les separa).

```
background-color - backgroundColor
```

3.3. CSS-P en navegadors DOM W3C

L'accés a les capes en els navegadors que compleixen l'especificació DOM del W3C es fa amb mètodes d'accés que ja s'han explicat. D'aquesta manera, quan el node (o capa) està referenciat en una variable, la manipulació dels atributs CSS que té es fa a partir de l'objecte Style (d'una manera semblant a la que es va implementar en Internet Explorer).

Vegeu també

Dels mètodes d'accés a les capes en els navegadors que compleixen l'especificació DOM del W3C s'ha tractat en l'apartat 2 del mòdul "Introducció al DOM" d'aquesta assignatura.

Per exemple, donada la capa següent:

```
<div id="capaDOM" style="position: absolute; top: 100px; left: 100px; width:
100px; height: 100px; background-color: " ffff99">Capa creada amb div!</div>
```

La modificació del color de fons es fa amb la sintaxi següent:

```
document.getElementById("capaDOM").style.backgroundColor = "orange";
```

D'aquesta manera, es poden manipular les propietats de l'objecte Style de qualsevol objecte del document, però la manipulació de regles definides amb selectors en un full d'estil extern es fa a partir d'una col·lecció styleSheets[ ] definida en un objecte Document i que s'especifica en el DOM2 per a CSS. Mitjançant aquesta col·lecció, s'accedeix a les regles de bloc definides i es manipulen directament amb els mètodes insertRule() i deleteRule().

3.4. Posicionament cross-platform

És necessari que, mitjançant funcions de script o mitjançant esdeveniments que hagi provocat l'usuari, les propietats de posicionament canviïn. Es vol modificar la posició, la mida, la visibilitat o l'índex z dels objectes. A més, és necessari que el que es faci sigui funcional en tots els navegadors o com a mínim en els més utilitzats.

Tal com s'ha vist en els subapartats anteriors, els mecanismes amb què s'accedeix als objectes i s'hi interactua a Internet Explorer 4 i a Netscape 4 i en els navegadors basats en el DOM són diferents. Afortunadament, aquest problema té solució i, a més, la solució és oberta, en el sentit que, si hi ha problemes nous amb versions posteriors dels navegadors, es poden tractar seguint la mateixa estratègia.

La programació cross-platform es basa en la programació condicionada al navegador que llegeix l'script; d'aquesta manera, quan s'ha detectat el navegador, depenent de les característiques que tingui, s'executa un codi o un altre.

3.4.1. Detectar el navegador

En el cas que les funcions que es defineixen depenguin del navegador que les executa, es pot utilitzar la tècnica explicada en l'apartat 1.1.1 per a detectar el navegador i fer servir una estructura condicional a les funcions.

Tot seguit es presentarà una llibreria amb les funcions següents:

```
function hidden(nomCapa) {}
function show(nomCapa){}
function setX(nomCapa,coorX){}
function setY(nomCapa,coorY){}
function setZ(nomCapa, indexZ){}
function setHeight(nomCapa, alt){}
function setWidth(nomCapa, ample){}
```

Per a això, en primer lloc, es crea una funció que la utilitzen les funcions anteriors i l'objectiu de la qual és assignar a una variable la referència de la capa, ja que cada navegador té un mecanisme diferent per a fer una referència a les capes:

```
function getElement(nomCapa){
       return document.getElementById(nomCapa);
}
```

Com es veu, en la funció la recuperació de la capa és immediata.

Definida la funció anterior, les funcions de l'API són molt senzilles:

```
function hide(nomCapa){
       //Es recupera la referència a la capa i s'assigna l'atribut hidden
       var capa = getElement(nomCapa);
       capa.style.visibility = 'hidden';
}

function show(nomCapa){
       //Es recupera la referència a la capa i s'assigna l'atribut visible
       var capa = getElement(nomCapa);
       capa.style.visibility = 'visible';
}

function setX(nomCapa,x){
       //Es recupera la referència a la capa i s'assigna la coordenada
       var capa = getElement(nomCapa);
       capa.style.left=x;
}
```

```
function setY(nomCapa, y){
       //Es recupera la referència a la capa i s'assigna la coordenada
       var capa = getElement(nomCapa);
       capa.style.top=y;
}

function setZ(nomCapa, zIndex){
       //Es recupera la referència a la capa i s'assigna el valor
       var capa = getElement(nomCapa);
       capa.style.zIndex = zIndex;
}

function setHeight(nomCapa, height){
       //Es recupera la referència a la capa i s'assigna el valor
       var capa = getElement(nomCapa);
       capa.style.height = height;
}

function setWidth(nomCapa, width){
       //Es recupera la referència a la capa i s'assigna el valor
       var capa = getElement(nomCapa);
       capa.style.width = width;
}
```

D'aquesta manera, les funcions anteriors s'escriuen en un fitxer extern que és referenciat en l'script quan són necessàries i es poden ampliar amb més funcions o s'hi poden introduir estructures condicionals a dins si l'assignació dels valors no es fa amb la mateixa sintaxi en algun navegador. És a dir, si hi ha versions noves que aporten propietats que no són compatibles amb les anteriors, es poden modificar les funcions anteriors perquè suportin les característiques noves d'aquests navegadors.

3.5. Fonaments de l'animació

L'animació en JavaScript és molt senzilla i es basa en un conjunt petit de tècniques basades en propietats CSS de les capes i en el maneig del temps amb els mètodes que hi ha en el llenguatge. D'aquesta manera, els quatre factors que s'han de tenir en compte són aquests:

- Posicionament.
- Definició del recinte.
- Definició del moviment.
- Temporització.

En els subapartats següents, es mostra com es poden implementar les característiques anteriors.

3.5.1. Posicionament

El posicionament és l'acció de situar o moure la capa per un escenari definit; si l'assignació de posicions absolutes és suficient, les funcions setX() i setY() que s'han definit en el subapartat anterior resolen aquesta primera part del problema.

El problema és que el moviment es basa en l'increment o decrement de la posició actual de la capa; per tant, s'ha d'obtenir la posició actual de la capa i, a partir d'aquests valors, incrementar o disminuir la posició amb les unitats definides per a assignar *a posteriori* aquests valors amb les funcions setX() i setY().

Per tant, s'han de tenir dues funcions, getX(nomCapa) i getY(nomCapa), que proporcionin la posició actual de la capa. El codi d'aquestes funcions és molt simple, com es pot veure tot seguit:

```
function getX(nomCapa){
       var capa = getElement(nomCapa);
       return(parseInt(capa.style.left))
}
function getY(nomCapa){
       var capa = getElement(nomCapa);
       return(parseInt(capa.style.top))
}
```

D'aquesta manera, amb les funcions anteriors es pot desplaçar 10 píxels a la dreta una capa anomenada *bola* amb el codi següent:

```
posX = getX("bola");
setX("bola",posX+10);
```

El desplaçament, però, ha de tenir uns límits pel que fa a la longitud, i també pel que fa al recinte en què es pot moure. Aquests dos aspectes s'estudien tot seguit.

3.5.2. Pas i grandària del recinte

En l'exemple del subapartat anterior s'ha desplaçat la capa 10 píxels cap a la dreta: aquest valor es diu **pas de la capa**. En una animació es defineix una variable amb el valor de pas, de manera que aquest valor es pot modificar dinàmicament a partir d'un esdeveniment d'usuari o quan es compleix alguna condició, com arribar a l'extrem del recinte.

S'ha de tenir en compte que el pas indica el sentit del desplaçament, de manera que en segons quines situacions el sentit del desplaçament s'ha d'invertir, i això és tan senzill com canviar el signe del pas. D'aquesta manera, si el pas = 10, el codi següent:

```
pas = pas*-1;
```

actualitza el pas al valor -10 i inverteix amb aquest codi simple el sentit del moviment, tant en horitzontal com en vertical.

Ara bé, si l'animació necessita que els moviments en els sentits vertical i horitzontal siguin independents, es poden definir dues variables: *pasX* i *pasY*.

Un altre aspecte que s'ha de considerar són els límits del recinte en què es fa l'animació, de manera que si es programa una animació autònoma (sense intervenció de l'usuari), quan la capa arriba al límit del recinte, aquesta animació s'ha de situar en un altre extrem del recinte o bé s'ha de canviar el sentit o la direcció del moviment (variable *pas*).

L'estratègia de definició del recinte es basa en quatre variables noves que defineixen els límits. Per exemple:

```
var limitSup =100;
var limitInf =400;
var limitEsq =100;
var limitDret =600;
```

La funció d'aquestes variables és comprovar en cada modificació de la posició de la capa (moviment) si aquesta capa ha arribat a un d'aquests límits i, si és així, portar a cap les accions que s'han definit perquè no superi els límits (canviar el sentit, saltar a una altra posició, parar el moviment, etc.).

A vegades pot ser que aquests límits depenguin de les dimensions de l'àrea del document o de la pantalla, de manera que s'han d'utilitzar algunes propietats dels objectes Document o Screen per a definir els límits del recinte.

3.5.3. Temporització

La temporització és especialment important en les animacions autònomes, ja que el moviment es basa en la crida de tant en tant a una funció que provoca un canvi de posició d'un objecte. Per a aquesta finalitat, se sol utilitzar la funció setInterval(nomFuncio, milliSegons).

La funció anterior té dos paràmetres: el primer conté el nom de la funció que s'ha d'executar i el segon és el nombre de mil·lisegons que passen entre dues crides a la funció. Per exemple:

```
var repeteix = setInterval("mou()",1000);
```

Executa la funció mou() al cap d'un segon.

D'altra banda, en un moment concret fa falta que la funció mou() es deixi d'executar, i per a això s'ha de cancel·lar la funció setInterval().

La cancel·lació d'aquesta funció es duu a terme amb la funció clearInterval(variableInterval), que ha de rebre com a paràmetre la variable que conté la referència a la funció setInterval().

Per exemple, la cancel·lació de la funció definida en l'exemple anterior es fa de la manera següent:

```
clearInterval(repeteix);
```

3.6. Aplicacions simples de CSS-P

En aquest subapartat, s'estudiaran quatre exemples senzills que utilitzen com a base tècnica el que s'ha explicat en els subapartats anteriors.

3.6.1. Animació en línia recta

Es defineix una animació en què es desplaça un element des d'una posició determinada d'una pàgina a una altra posició, seguint una línia recta entre l'una i l'altra. Per a això, es defineix una funció animaRecta() que necessita els paràmetres d'entrada següents:

- L'identificador de la capa que es mou.
- La coordenada *x* de la posició inicial.
- La coordenada *y* de la posició inicial.
- La coordenada *x* de la posició final.
- La coordenada *y* de la posició final.
- La velocitat del moviment.

D'aquesta manera, la crida a la funció següent:

```
animaRecta("capa1", 100, 100, 300, 300, 10);
```

mou la capa capa1 des de la posició inicial (100, 100) fins a la posició final (300, 300) seguint una línia recta i a 10 de velocitat.

```
// Es crea un objecte anima que disposa d'un conjunt
//de propietats relacionades amb el moviment
var anima = new Object();
//Se inicialitza l'objecte anima
function initAnima() {
```

```
    anima = {capa:"", valorX:0, valorY:0, finX:0, finY:0, pasoX:0, pasoY:0,
    distX:0, distY:0, MoviX:0, MoviY:0, vel:1, cami:1, interval:null };
}
// Es defineix la funció animaRecta
function animaRecta(capa, iniciX, iniciY, finalX, finalY, pas) {
    initAnima();
    anima.capa = capa;
    anima.valorX = iniciX;
    anima.valorY = iniciY;
    anima.finX = finalX;
    anima.finY = finalY;
    anima.distX = Math.abs(finalX - iniciX);
    anima.distY = Math.abs(finalY - iniciY);
    anima.vel = (pas) ? pas : 1;
    //S'assigna la posició inicial de l'element
    document.getElementById(capa).style.left = iniciX + "px";
    document.getElementById(capa).style.top = iniciY + "px";
    //Es calcula la longitud de la línea entre l'inici i el final de les coordenades
    anima.cami = Math.sqrt((Math.pow((iniciX - finalX), 2)) + (Math.pow((iniciY - finalY), 2)));
    //Es calcula la mida en píxels dels passos al llarg dels eixos
    anima.pasX = parseInt(((anima.finX - anima.valorX) / anima.cami) * anima.vel);
    anima.pasY = parseInt(((anima.finY - anima.valorY) / anima.cami) * anima.vel);
    //S'inicia la crida respectiva a l'animació
    anima.interval = setInterval("executaAnimacio()", 10);
}
// Calcula els passos següents i els assigna a les propietats
function executaAnimacio() {
    if ((anima.MoviX + anima.pasX) <= anima.distX &&
    (anima.MoviY + anima.pasY) <= anima.distY) {
        var x = anima.valorX + anima.pasX;
        var y = anima.valorY + anima.pasY;
        document.getElementById(anima.capa).style.left = x + "px";
        document.getElementById(anima.capa).style.top = y + "px";
        anima.MoviX += Math.abs(anima.pasX);
        anima.MoviY += Math.abs(anima.pasY);
        anima.valorX = x;
        anima.valorY = y;
    } else {
        document.getElementById(anima.capa).style.left = anima.finX + "px";
        document.getElementById(anima.capa).style.top = anima.finY + "px";
        clearInterval(anima.interval);
    }
}
```

La funció anterior comença amb la creació del contenidor anima, que s'encarrega d'emmagatzemar els valors per a l'animació. La crida a la funció animaRecta() assigna els valors inicials de l'objecte anima i tot seguit emplena les propietats amb els valors que s'han passat com a paràmetres o s'han calculat a partir d'aquests paràmetres.

La funció s'acaba amb la crida al mètode setInterval(), que crida diverses vegades la funció que realment implementa l'animació executaAnimacio(). Aquesta funció mou l'objecte cap a la destinació que té fins que l'element s'acosta o arriba a les coordenades de destinació; quan no pot anar més enllà, l'element es posiciona explícitament en la destinació i l'identificador de l'interval es neteja per a aturar les iteracions.

3.6.2. Animació circular

L'exemple següent es pot interpretar com una variant de l'anterior, ja que la diferència és que aquesta vegada s'anima l'element fent que dugui a terme un traçat circular en comptes d'un de rectilini.

Per a això, la funció nova animaCirc() necessita els paràmetres següents:

- L'identificador de la capa que es mou.
- La coordenada *x* del punt de començament/final del cercle.
- La coordenada *y* del punt de començament/final del cercle.
- Un valor sencer parell que indiqui el nombre de punts que s'han de pintar en el cercle.
- Un valor sencer del radi relatiu del cercle.

Per exemple, la crida següent de la funció:

```
animaCirc("circula", 120, 120, 18, 5);
```

mou l'element des del punt (120, 120), seguint una circumferència de radi 5 píxels.

```
// Es crea un objecte anima, que disposa d'un conjunt
//de propietats relacionades amb el moviment
var anima = new Object();
//S'inicialitza l'objecte Anima
function initAnima() {
    anima = {capa:"", valorX:0, valorY:0, finX:0, finY:0, pas:1, punts:1, radi:1,
    interval:null };
}
//Es completa l'objecte anima amb els paràmetres i valors calculats necessaris
    function animaCirc(capa, iniciX, iniciY, punts, radio) {
    initAnima();
    anima.capa = capa;
```

```
    anima.finX = anima.valorX = iniciX;
    anima.finY = anima.valorY = iniciY;
    anima.punts = punts;
    anima.radi = radi;
    //Assigna la posició inicial dels elements
    document.getElementById(capa).style.left = iniciX + "px";
    document.getElementById(capa).style.top = iniciY + "px";
    //Comença la repetició de l'animació
    anima.interval = setInterval("ejecutaAnimacionC()", 10);
}
    function executaAnimacioC() {
    if (anima.pas < anima.punts) {
    var x = anima.finX +
    Math.round(Math.cos(anima.pas * (Math.PI/(anima.punts/2)))
    * anima.radi);
    var y = anima.finY +
    Math.round(Math.sin(anima.pas * (Math.PI/(anima.punts/2)))
    * anima.radi);
    document.getElementById(anima.capa). style.left = x + "px";
    document.getElementById(anima.capa). style.top = y + "px";
    anima.finX = x;
    anima.finY = y;
    anima.pas++;
    } else {
    document.getElementById(anima.capa).style.left =
    anima.valorX + "px";
    document.getElementById(anima.capa).style.top =
    anima.valorY + "px";
    clearInterval(anima.interval);
    }
}
```

El codi de l'animació circular és semblant al que s'ha plantejat en l'exemple anterior; aquest es basa en la suavitat del moviment circular, que es fonamenta en el nombre de punts al llarg del cercle, que es converteix en el límit superior d'anima.pas en l'if de la funció executaAnimacioC().

La matemàtica aplicada es basa en el fet que s'ha dividit el valor per PI per a aconseguir un cercle complet; d'aquesta manera, per a qualsevol combinació de valors, el radi del cercle el controla el factor de multiplicació que hi ha al final de les dues línies que contenen Math.PI.

De fet, el valor emmagatzemat en anima.radi no és una mesura recta en píxels sinó un factor que controla el radi, de manera que com més gran és el número més gran és el radi.

Per acabar, es pot indicar que augmentar el valor d'anima.punts fa que l'animació sigui més suau, ja que els arcs entre els punts de refresc són més grans, és a dir, el moviment és més suau, pel fet que el temps de l'interval s'ha fixat en 10 mil·lisegons i això implica que s'executi més ràpid.

3.6.3. Crear un menú desplegable amb jQuery

Seguint els passos següents es crearà un menú desplegable de dos nivells utilitzant CSS i jQuery. L'estructura HTML del menú serà la següent:

```
<ul class="menu">
<li><a href="#">Inici</a></li>
<li><a href="#">Tipus i Requisits</a>
<ul>
   <li><a href="#">Jubilaci&amp;oacute;</a></li>
   <li><a href="#">Pensi&amp;oacute;</a></li>
   <li><a href="#">Jubilaci&amp;oacute; per invalidesa</a></li>
   <li><a href="#">Jubilaci&amp;oacute; per edat avan&amp;ccedil;ada</a></li>
</ul>
</li>
<li><a href="#">Reajustaments</a></li>
<li><a href="#">Contacte</a></li>
</ul>
```

S'utilitza CSS per a aplicar format: en primer lloc, se situa cada element al costat de l'altre i no a sota de l'altre tal com se situen per defecte les llistes. S'assigna el valor *position:relative* a l'element individual perquè quan una nova llista parteixi de l'element es pugui posicionar correctament utilitzant *position:absolute*.

S'aplica *text-transform:uppercase* a l'enllaç per a col·locar-ne el text en majúscules, se'n defineix el color de fons i la tipografia. Després se'n defineix l'estat *:hover* i se'n canvia el color de fons i de tipografia novament per a destacar-lo.

```
ul.menu {
   float:right;
   display:block;
   margin-top: 38px;
   list-style-type:none;
}
.menu li {
   line-height:18px;
   font-size:13px;
   position:relative;
   float:left;
}
.menu li a {
   color: #000;
```

```
    text-transform:uppercase;
    padding: 5px 20px;
    text-decoration:none;
}
.menu li a:hover {
    background: #9c0101;
    color: white;
}
```

En el pas següent es posiciona la nova llista que s'obri com a filla d'un element <li>. Per defecte està oculta *display:none* i amb position i top s'acomoda la llista perquè es desplegui cap avall.

```
.menu li ul {
    display:none;
    position:absolute;
    top:20px;
    width: 240px;
    background-color: #f4f4f4;
    padding:0;
    list-style-type:none;
}
```

To seguit, s'estilitzen una mica els elements fills de la nova llista.

```
.menu li ul li {
    width: 200px;
    border: 1px solid #9c0101;
    border-top:none;
    padding: 10px 20px;
}
.menu li ul li:first-child {
    border-top: 1px solid #9c0101;
}
.menu li ul li a {
    width: 240px;
    margin: 0;
    padding:0;
}
.menu li ul li a:hover {
    width: 240px;
    margin: 0;
    color: #9c0101;
    background:none;
}
```

Per acabar el menú s'utilitza la biblioteca jQuery, ja que simplifica el procés. Amb poc codi es modifica la propietat display de cada ul li que tingui una etiqueta ul filla cada vegada que el punter del ratolí hi passa per dalt i en surt. Ara bé, a continuació s'expliquen uns detalls bàsics de la biblioteca per entendre el codi.

Introducció jQuery

De la mateixa manera que s'han creat biblioteques al llarg del curs, jQuery és una biblioteca JavaScript que simplifica el tractament de documents HTML, el maneig d'esdeveniments, la creació d'animacions i les interaccionis via Ajax. A més, és multiplataforma, de manera que s'ha provat en els navegadors següents: Internet Explorer 6.0+, FireFox 2.0+, Safari 2.0+, Opera 9.0+ i Chrome.

L'última versió de la biblioteca es pot descarregar del web:

http://docs.jquery.com/Downloading_jQuery#Current_Release,

de manera que es fa ús del fitxer utilitzant la sintaxi habitual:

```
<script type="text/javascript" src="jquery-1.7.2.min.js"></script>
```

A partir d'aquest moment ja es poden utilitzar les funcions jQuery. Les funcions bàsiques es presenten a continuació:

La funció central de jQuery és la funció dolar $(""), amb la qual s'accedeix als nodes de l'arbre DOM.

A continuació uns senzills exemples d'ús:

```
$("div"); // obté tots els nodes DIV de l'arbre DOM
$("#layout"); // obté el node amb l'atribut id = "layout" de l'arbre DOM
$(".myStyle"); // obté els nodes els atributs class dels quals tinguin el valor "myStyle" dins
               // de l'arbre DOM
$("&lt;li&gt;home&lt;/li&gt;"); // crea un node de tipus li amb un text embegut "home"
$("div#content"); // obté el node de tipus div l'identificador del qual sigui "content"
$("&lt;li&gt;home&lt;/li&gt;").wrapInner("&lt;a&gt;"); // crea un node li i insereix una àncora
$("div#content").text("Aquest és el nou contingut de la capa"); // obté el node div amb id =
                                                                // content i modifica el text que
                                                                // conté la capa.
```

La funció $("") admet diferents paràmetres:

```
$("li"); //Selectors CSS
$(document);// Elements DOM
$("&lt;li&gt;home&lt;li&gt;"); // HTML
```

```
$(function(){});//Funcions
```

I a més disposa de mètodes per a assignar controlador d'esdeveniments:

- clic() i dbclick(): afegeix un esdeveniment al clic del ratolí.
- keypress() i keydown(): afegeix un esdeveniment quan es prem el teclat.
- hover(): afegeix dos esdeveniments a l'acció de passar el punter del ratolí sobre un component, el primer dels quals és passar-hi pel damunt i el segon, deixar-lo.
- toogle(): intercanvia entre un comportament o un altre quan es produeix l'esdeveniment.

També incorpora mètodes per a modificar els atributs associats als components o nodes de l'arbre DOM:

- text() i html(): retornen o assignen el text com un String o html com si treballéssim amb innerHTML.
- attr() i removeAttr(): accedeix, modifica i elimina un atribut d'un node.
- addClass(), removeClass() i toogleClass(): afegeix, elimina o intercanvia les classes css.
- css(): retorna o assigna una propietat css a un node.

La combinació dels dos tipus de funcions executa codi d'una certa complexitat com s'observa en els exemples següents:

```
<script type="text/javascript">
// la funció ready associada al document  assegura que el codi s'executa.
//quan la pàgina està totalment carregada en el client.
$(document).ready(function() {
//s' obtenen tots els enllaços i s'associa una funció a l'esdeveniment onclick.
$("a").clic(function(event){
// quan es produeixi l'esdeveniment s'assigna un estil css de color de fons a l'enllaç.
$(this).css({backgroundColor:'xarxa'});
});
});
</script>
```

Per tant, el codi restant del menú desplegable és el següent:

```
<script type="text/javascript" src="jquery-1.7.2.min.js"></script>
<script type="text/javascript">
```

```
    $(document).ready(function() {  //s'executa la funció quan la pàgina està carregada,
    $('ul li:has(ul)').hover(  //s'assigna en primer lloc el controlador de l'esdeveniment
                              //en passar el punter del ratolí per l'opció del menú.
    function(i)
    {
       $(this).find('ul').css({display: "block"}); //es mostra l'opció del menú.
    },
    //la funció següent s'executarà quan el punter del ratolí abandona l'enllaç.
    function(i)
    {
       $(this).find('ul').css({display: "none"}); //s'amaga l'opció del menú.
    }
    );
    });
</script>
```

3.7. Manipulació dinàmica dels fulls d'estil

El W3C defineix CSS com "un mecanisme simple per a afegir estil (per exemple, fonts, colors, espais) en documents web". Per tant, els CSS (*Cascading Style Sheets*) són un mecanisme per a aplicar format als documents HTML (i a documents en altres llenguatges estructurats, com XML), i separen el contingut de les pàgines de l'aparença que tenen.

La informació està continguda en el document HTML, però en aquest document no s'especifica com s'ha de visualitzar o representar la informació. Les indicacions sobre la presentació visual del document estan especificades en els CSS.

El W3C, com no podia ser de cap altra manera, és l'organisme que dicta també els estàndards sobre els CSS. El 1995 es va començar a parlar de la utilització dels fulls d'estil CSS per a incloure'ls en les especificacions 3.0 de l'HTML, però la discussió es va allargar tant que al final es van imposar les extensions que havia proposat Netscape, cosa que va donar lloc a la versió 3.2 de l'HTML, encara que la versió 3.0 de l'Explorer ja les suportava en certa manera.

Avui dia, les especificacions CSS són en el nivell 2 (CSS2). Respecte al suport que tenen aquestes especificacions dels navegadors principals, la situació actual és la següent:

Taula 3. Especificacions CSS en els navegadors

Navegador	Suport CSS1	Suport CSS2
Netscape 4	Escàs	No consta
Internet Explorer 4	Escàs	No consta
Internet Explorer 5	Lleugerament, però amb problemes	No consta

Navegador	Suport CSS1	Suport CSS2
Internet Explorer 5.5	Total, però amb problemes	No consta
Netscape 6	Total	Gairebé total
Internet Explorer 6	Total	Gairebé total
Mozilla 1.0	Total	Gairebé total
Opera 7.0	Total	Gairebé total

Com es veu en la taula 3, la tendència dels navegadors, afortunadament, és suportar els estàndards, però a causa dels costos que això comporta no s'aconsegueix sempre prou bé.

3.7.1. Motius per a fer servir CSS

Hi ha molts motius per què el programador d'espais web faci servir fulls d'estil. Tot seguit s'enumeren els més destacats:

- CSS és un llenguatge estàndard de disseny per al web i, per tant, aplicable a altres formats com l'XML. Controla els colors, la tipografia, la mida i el lloc on hi ha elements i imatges.

- CSS és molt senzill de programar, de manera que no és imprescindible fer servir editors especialitzats.

- CSS permet d'estalviar amplada de banda, ja que un sol arxiu CSS pot definir l'estètica de tot un web complex.

- CSS redueix les operacions d'actualització i manteniment, ja que els encarregats de continguts no s'han de preocupar de l'aspecte final. A més, els canvis globals es poden fer en poca estona: només s'ha de modificar un arxiu CSS i es modifica l'aspecte de tot el web.

Si a més es té en compte la convergència dels navegadors actuals a complir l'estàndard, aquest ús és encara més interessant.

3.7.2. Canviar un full d'estil

En aquest subapartat es mostren un seguit d'accions molt comunes relacionades amb la manipulació de CSS des del llenguatge JavaScript. Aquestes accions es presenten en un exemple molt senzill.

En alguns webs es permet als usuaris de definir una skin, que és l'aspecte que té el web. A més, es pot implementar carregant una pàgina CSS nova quan l'usuari selecciona l'aspecte nou.

Tot seguit, es planteja un exemple d'aquesta tècnica. Per començar, es disposa d'aquesta línia que defineix el full CSS per defecte del web:

```
<link id="estilPerDefecte" rel="stylesheet" type="text/css" href="estil.css" />
```

La càrrega d'un full d'estils extern nou es pot fer utilitzant la sintaxi següent:

```
document.getElementById("estilPerDefecte").href="estilNou.css";
```

D'altra banda, es pot activar o desactivar un full d'estil de la pàgina; per a això s'utilitza la propietat disabled de l'objecte StyleSheet:

```
document.styleSheets[1].disabled = true;
```

L'activació és tan simple com l'assignació del valor false en la propietat anterior.

3.7.3. Llegir valors de les propietats del full

Tot seguit definim una funció que cerca el valor d'una propietat del full d'estil establert amb una etiqueta `<style>` o un full d'estil importat.

```
function getElementStyle(elemID, IEStyleProp, CSSStyleProp) {
       var elem = document.getElementById(elemID);
       if (elem.currentStyle) {
              return elem.currentStyle[IEStyleProp];
       } else if (window.getComputedStyle) {
              var compStyle = window.getComputedStyle(elem, "");
              return compStyle.getPropertyValue(CSSStyleProp);
       }
       return "";
}
```

La funció torna el valor de la propietat, sia definida en IE (utilitzant l'objecte CurrentStyle) o en navegadors CSS (utilitzant el mètode window.getComputedStyle() del W3C DOM).

Activitats

Seleccioneu 5 exemples de la pàgina web següent:

http://www.htmlpoint.com/dhtml/index.html

implementeu-los i comenteu al fòrum els problemes amb què us heu trobat i les característiques interessants que heu descobert.