CSS3 i JavaScript avançat

Jordi Collell Puig

PID_00176148

UOC
Universitat Oberta de Catalunya

Índex

1. CSS, especificació tercera

1.1. Introducció al CSS

El CSS és un llenguatge d'estils, emprat per a definir la presentació, el format i l'aparença d'un document de marcatge, tant HTML com XML o qualsevol altre. Comunament, s'empra per a donar format visual a documents HTML o XHTML que funcionen com a espais web. També es pot emprar en formats XML o altres tipus de documents de marcatge per a generar documents.

Els fulls d'estil neixen de la necessitat de dissenyar la informació de tal manera que puguem separar el contingut de la presentació, i així, per a una mateixa font d'informació, generalment definida mitjançant un llenguatge de marcatge, oferir diferents presentacions segons dispositius, serveis, contextos o aplicacions. Per exemple, un mateix document HTML, mitjançant diferents fulls d'estil, es pot presentar per pantalla, per impressora, per lectors de veu o per tauletes Braille. Separem el contingut de la forma, la composició, els colors i les fonts.

El World Wide Web Consortium (W3C, http://www.w3c.org) manté l'especificació del CSS.

1.2. L'especificació tercera (CSS3)

Les diferents revisions dels fulls d'estil tenen l'origen en la **primera especificació** publicada pel W3C el desembre del 1996, que pretenia unificar la sintaxi i la manera de definir un full d'estil per als diferents llenguatges derivats de l'SGML. Així, les principals característiques que es poden definir mitjançant aquesta primera especificació són:

1) Propietats del tipus i de l'estil de lletra.

2) Colors dels textos i dels fons.

3) Atributs del text com espai entre caràcters, paraules i línies.

4) Alineació de taules, blocs de text, imatges, paràgrafs.

5) Marges externs i interns, filets i posició de la majoria d'elements.

6) Definició única d'elements mitjançant ***ids*** i agrupament d'atributs mitjançant ***classes***.

A partir d'aquesta primera especificació, el maig del 1998, i com una extensió de la primera revisió, apareix la segona, popularment coneguda com l'*especificació **CSS2*** i adoptada per la majoria de navegadors. Com a característiques fonamentals de la segona revisió, hi ha la possibilitat de definir posicions de forma absoluta, relativa i fixa, la profunditat dels elements (***z-index***) quan hi ha superposicions i també suport per a formats de veu i textos bidireccionals. D'aquesta revisió, n'apareix una altra, la **2.1**, que incorpora moltes de les millores fetes pels fabricants de navegadors i que actualment és emprada i estandarditzada.

La tercera revisió de l'especificació del **CSS** pel W3C comença el 2005 i encara està en procés de definició. Aquest cop, però, les diferents implementacions dels motors de renderització (*rendering*) dels navegadors no estan esperant a tenir una especificació, sinó que implementen certes coses a la seva manera, i per tant, moltes són utilitzables en entorns de producció web. Cal partir sempre, però, de la consideració que les diferents implementacions dels navegadors no és exacta, i que, per tant, quan dissenyem un full d'estil, el principi que l'ha de regir és que funcioni a tot arreu, i no pas que funcioni igual.

A diferència de les altres especificacions, aquesta vegada s'ha dividit la tercera revisió en temes. D'aquesta manera disposem de diferents temes que poden créixer i evolucionar en paral·lel, i no pas com un de gran i monolític, amb moltíssimes revisions (cada vegada hi ha més gent que hi està interessada i evoluciona més de pressa).

En la tercera revisió hi trobem grups com els de selectors, unitats de mesura, model de caixa, colors i gammes, model de línia, text, fonts, llenguatges verticals i *page-media*. I d'aquesta manera, diferents mòduls tenen un estatus diferent i són adoptats pels fabricants de programari a diferent velocitat.

1.2.1. CSS en el World Wide Web

Lluny dels usos més abstractes, els fulls d'estil han esdevingut l'eina per a donar format i color als continguts del World Wide Web (WWW). Així, qualsevol document HTML és formatat amb estils CSS. La principal característica del web semàntic és aquesta separació de continguts i visualització, en què el contingut té sentit en si mateix, i la visualització s'adapta a cada dispositiu i medi. D'aquesta manera i com que és l'eina amb la qual donem forma i color al contingut, els fabricants de programari (navegadors) han passat a ser els implementadors de les funcionalitats especificades pel W3C.

A grans trets, tots implementen l'especificació, però en les excepcions hi ha matisos i cada navegador té les seves característiques a l'hora de dibuixar (***render***) el contingut amb el full d'estil.

1.3. Famílies de navegadors

Parlem de famílies de navegadors per especificar les diferents tecnologies de dibuix que contenen, i poder agrupar les diferents versions de navegadors d'una manera senzilla per al desenvolupador. Destaquem quatre grans famílies:

1) Família basada en el **motor de dibuix de l'Internet Explorer**. Fonamentalment, són navegadors en el sistema operatiu Windows, en les diferents versions, 6, 7, 8, 9, i molts dels encastats en programes com el Microsoft Office.

2) Família Geko, que utilitza el **motor de dibuix Geko**, desenvolupat pel Mozilla. El més popular és el Firefox, però hi ha múltiples implementacions en diferents dispositius.

3) Família Webkit. Basada en el **motor de dibuix *opensource* Webkit**, desenvolupat pel Konkeror, i actualment millorat pel Safari d'Apple, i el Google Chrome. També esdevé el motor de dibuix de les dues plataformes de dispositius mòbils més esteses: l'iOS (iPhone/iPad) i l'Android.

4) Família de navegadors basada en l'**Opera**.

Hem de destacar que només parlem de la **tecnologia de dibuix**, o sigui, la part del programari que llegeix el document **SGML**, hi aplica els estils que troba al full d'estil i el dibuixa a la pantalla (o qualsevol altra sortida). A més d'això, un navegador (client web) conté un intèrpret de JavaScript que ens permet executar seqüències d'ordres i interactuar amb el **DOM** (***document object model***) o conjunt d'SGML amb els estils aplicats.

Aquest aclariment el fem perquè a partir d'aquí, quan parlem de les diferents novetats i tècniques disponibles amb la tercera revisió, especificarem en quins motors de dibuix funciona i així extraurem les propietats més comunes que podem utilitzar avui dia en el procés de confecció d'espais web.

1.4. Beneficis de l'ús del CSS3

1) Reduir el temps de desenvolupament i manteniment

Utilitzar propietats i mètodes de CSS3 pot ser un benefici directe a l'hora de desenvolupar, ja que ens estalviem força feina, com per exemple a l'hora de fer fons amb cantonades arrodonides, ja que abans s'havien de fer amb imatges.

També ens estalviem molta feina a l'hora de fer ombres, ja que ens tornem a estalviar la imatge que havíem de fer servir abans (normalment un gràfic en format PNG).

D'altra banda, podem millorar el rendiment, ja que tenim menys codi, *divs* dintre *divs*, etc.

2) Incrementar el rendiment de les pàgines

Menys etiquetes HTML indiquen menys codi a l'hora de baixar-se del servidor i menys codi a l'hora d'interpretar i dibuixar el navegador. Això implica dos estalvis: un d'amplada de banda i un altre de rendiment de l'ordinador. A més, moltes de les tècniques de CSS3 ens estalvien imatges, que alhora compleix la doble premissa de rendiment.

1.5. La millora progressiva

Un dels elements clau a l'hora d'emprar CSS és utilitzar una tècnica de desenvolupament anomenada *millora progressiva*, que consisteix a començar a generar un codi genèric que funcioni en tots els navegadors i, de mica en mica, introduir millores per a navegadors més moderns. Això és possible perquè els intèrprets de CSS dels navegadors ignoren les propietats que no coneixen.

Emprant aquesta tècnica assolim un control total òptim de l'aspecte, ja que com més bones prestacions tingui el navegador, més bona visualització tindrà.

2. Novetats CSS3

2.1. Nous selectors

Els selectors CSS són l'eina més potent del llenguatge d'estils, ja que ens permeten seleccionar diferents elements del contingut HTML depenent de l'etiqueta o dels seus atributs, sense haver de fer ús de la seva classe, el seu **ID** o JavaScript.

2.1.1. Selectors d'atributs

- **[att^="valor"]**

Selecciona elements amb un atribut que comença per *valor*.

- **[att$="valor"]**

Selecciona elements amb un atribut que acaba amb *valor*.

- **[att*="valor"]**

Selecciona elements que contenen un atribut que conté *valor*.

Aquests selectors els poden utilitzar tots els fulls d'estil destinats a navegadors, l'Internet Explorer 7 o un de superior, l'Opera, el Webkit i els navegadors basats en el Geko.

Exemple:

```
<style>
   a[title$="sample"] {
      color:#ea1d1d;
   }
</style>
<p>Lorem ipsum <a href="#" title="this is a sample">dolor sit amet</a>, consectetur adipisicing
elit, sed do eiusmod tempor incididunt ut labore et dolore <a href="#">magna aliqua</a>. Ut
enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse <a href="#"
title="this is a sample">cillum dolore</a> eu fugiat nulla pariatur. Excepteur sint occaecat
cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.</p>
```

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

En l'exemple anterior amb el selector d'atributs acabat en ($=), marquem els enllaços que contenen l'atribut *title* acabat amb *sample* de color vermell.

2.1.2. Combinadors (no funciona amb l'IE6)

- **~ *Sibling* general (germà)**

Selecciona els germans d'un element. Una etiqueta germana és la que existeix al mateix nivell, o que té un pare en comú i que és a continuació de la referenciada. Així, en l'exemple següent podem dir que són etiquetes germanes els *p* i els *h2*, ja que tots tenen el mateix pare (*div*). Només se seleccionaran, però, els elements *p* posteriors a la definició del *h2*.

Exemple:

```
<style>
h2~p { color:green; }
</style>
<div id="un">
 <p>Aquest element no serà visible, perquè és germà, però és anterior.</p>
 <h2>This is a test</h2>
 <p>Aquest p és germà de h2.</p>
 <p>Aquest altre p també és germà de h2.</p>
</div>
<div id="dos">
   <p>Aquest p no es pintarà.</p>
</div>
```

Pintarà de verd els *p* del *div id=un* posteriors a *h2*, ja que són germans del *h2*. Tenen en comú el pare *#un*, mentre que el *p* del *div id=dos* no es pintarà, ja que no té cap germà *h2*.

2.1.3. Pseudoclasses

Les pseudoclasses són un dels afegitons més estesos i utilitzats en el CSS3. Les classes més útils són les següents:

- **:nth-child(n)**

Selecciona elements basant-se en la posició dels fills. Pot utilitzar nombres, expressions i les paraules *odd* i *even* ('senar' i 'parell'). Podem fer fàcilment el típic efecte zebra a les taules.

Per exemple, donada una llista de <li>:

```
li:nth-child(2) { color: red; }
Pintarà de vermell el segon element de la llista.
li:nth-child(even) { color: red; }
Pintarà de vermell els elements parells.
li:nth-child(n+4) { color: red; }
Pintarà de vermell tots els elements a partir del quart.
li:nth-child(3n) { color: red; }
Pintarà de vermell cada tres elements.
li:nth-child(2n+5) { color: red; }
Pintarà de vermell cada dos elements a partir del cinquè.
```

- **:nth-last-child(n)**

Segueix la mateixa idea que l'anterior, però selecciona a partir del darrer element.

- **:last-child**

Selecciona el darrer element d'una llista.

- **:checked**

Selecciona elements que estan marcats, com ara caselles de selecció o *checkboxes*.

- **:empty**

Selecciona elements que són buits.

- **:not**

Selecciona elements que no compleixen la declaració especificada.

```
p:not([class*="lead"]) { color: black; }
Marcarà tots els p que no tinguin una classe assignada que contingui la cadena lead.
```

Els navegadors basats en el Webkit i l'Opera admeten tots els pseudoselectors CSS3; els basats en el Firefox 2 i 3 només admeten el *not*, *last-child only-child*, *root*, *empty*, *target*, *checked*, *enabled* i *disabled*; els que es basen en el Firefox 4 admeten tots els pseudoselectors, i els navegadors Microsoft no admeten pseudoselectors.

2.1.4. Pseudoelements

Els pseudoelements ens permeten seleccionar parts de contingut dintre d'una etiqueta. Així, per exemple, podem seleccionar la primera línia emprant el pseudoelement *:first-line*, o la primera lletra emprant el pseudoelement *:first-letter*. Així:

```
P:first-letter { font-size: 35px }
```

Ens canviarà la primera lletra. Una de les novetats més interessants del CSS3 és el pseudoelement *:selection*, aplicat als camps de formulari: ens permet introduir modificacions en el moment que un camp de formulari té el focus.

```
input:selection { background-color:#eaeaea }
```

Canviarà el color del camp de formulari, quan aquest tingui el focus seleccionat.

2.2. Colors RGBA i opacitat

2.2.1. RGBA

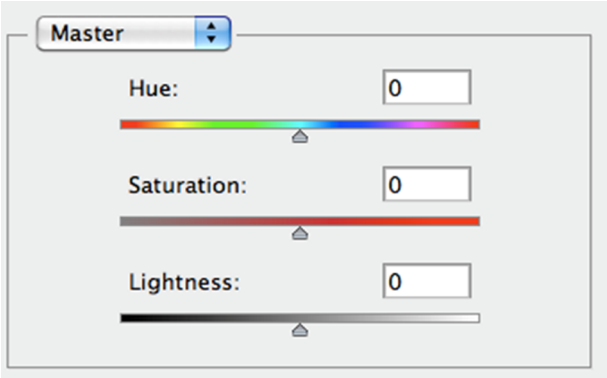
La possibilitat d'especificar colors en mode RGB ja existeix des de l'especificació 2, però en la versió 3 és ampliada amb un nou model de color, el basat en saturació (HSL), i la possibilitat d'especificar **canal alpha** sobre els colors. D'aquesta manera, podem crear expressions CSS del tipus:

```
p { color: rgba(0,0,0, 0.5) }
```

Crearà un color negre amb una transparència del 50%. Per a veure'n l'efecte caldrà tenir una imatge per sota, o si el nostre color de fons és blanc, ens apareixerà color gris.

2.2.2. Colors HSL

El model de color **HSL** (to, saturació i llum o ***hue***, ***saturation***, ***light***) defineix el color a partir dels tres paràmetres. Així, per a una tonalitat de color determinada, podem alterar la saturació a partir del segon paràmetre, i la quantitat de llum al tercer paràmetre.



Per exemple:

Serà definida pels colors:

```
background:hsla(320, 100%, 10%);
background:hsl(320, 80%, 30%);
background:hsl(320, 60%, 50%);
background:hsl(320, 40%, 70%);
background:hsl(320, 20%, 90%);
```

També podem afegir el component d'*alpha* si definim com a ***hsla(320, 80%, 30%, 0.7)***.

El model de color HSL és implementat en els navegadors basats en el Webkit i el Firefox.

2.2.3. Opacitat

Com indica el nom, l'opacitat (*opacity*) ens permet definir el nivell de transparència per a un selector. A diferència del model RGB, la transparència s'aplica a l'objecte i tots els fills o continguts. Així, per exemple:

```
<img src="img_02.jpg" style="opacity:0.2">
<img src="img_02.jpg" style="opacity:0.4">
<img src="img_02.jpg" style="opacity:0.6">
<img src="img_02.jpg" style="opacity:0.8">
<img src="img_02.jpg" style="opacity:1">
```

Produirà:

Totes les propietats són admeses pels navegadors basats en tecnologia Webkit, Geko (Mozilla) i Opera, mentre que l'Explorer no l'admet. De tota manera, la família Microsoft admet una altra propietat complementària, que ens permet fer el mateix efecte:

```
filter: alpha(opacity = 50);
```

Podem utilitzar totes aquestes propietats sempre que tinguem en compte un mecanisme alternatiu per als navegadors que no l'admeten. La millor mecànica de treballar és definir primer un color admès i afegir després les propietats CSS3; d'aquesta manera el navegador que no l'admet utilitzarà la primera, mentre que els que l'admetin, la podran utilitzar.

2.3. Noves propietats

2.3.1. Cantonades arrodonides

Segurament les cantonades arrodonides (***border-radius***) és la propietat CSS3 més esperada i desitjada pels dissenyadors HTML. Permet arrodonir les cantonades d'una caixa (*div*, *span*...) donat un radi, i tot sense la necessitat d'utilitzar una imatge de fons, produïda amb una eina d'edició gràfica.

Podem manipular les cantonades d'una caixa de manera uniforme utilitzant la propietat ***border-radius: 15px;*** de la manera que mostrem en l'exemple següent:

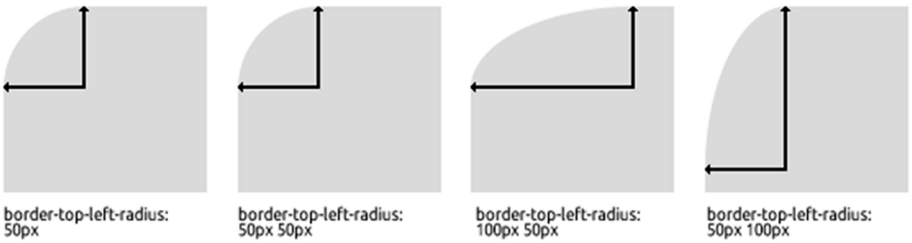
```
#example1 {
   -moz-border-radius: 15px;
   border-radius: 15px;
}
```

Aquesta caixa te cantonades rodones en tots els navegadors disponibles

Cal fixar-nos en la presència de la propietat ***-moz-border-radius: 15px;*** específica dels navegadors basats en el Geko.

També podem definir les propietats per a cadascuna de les cantonades utilitzant les propietats ***border-bottom-left-radius***, ***border-bottom-right-radius***, ***border-top-left-radius*** i ***border-top-right-radius***. Aquestes propietats admeten dos paràmetres per a definir el radi de curvatura de la cantonada. Si en definim

només un, aconseguim cantonades simètriques, mentre que si els definim tots dos, estem tractant independentment el paràmetre *x* del *y*, com es pot apreciar en el gràfic següent:



Cal dir que, per als navegadors basats en el Geko, s'han d'emprar les propietats següents: ***-moz-border--moz-bottom-left-radius***, ***-moz-border-bottom-right-radius***, ***-moz-border-top-left-radius*** i ***-moz-border-top-right-radius***.

D'altra banda, la propietat ***border-radius*** també es pot emprar per a definir les quatre cantonades de la caixa alhora:

```
border-radius: 5px 10px 5px 10px / 10px 5px 10px 5px;
border-radius: 5px;
border-radius: 5px / 10px;
```

En la primera definició, el primer grup de quatre mesures defineix les propietats horitzontals dels quatre radis, mentre que el segon grup utilitzat en separat pel caràcter /, defineix els radis verticals. En el segon exemple, es defineixen tots amb un radi de *5px*. En el tercer, es defineix un radi horitzontal de *5px* i un de vertical de *10px* per a les quatre cantonades.

Totes aquestes propietats són disponibles en els navegadors basats en el Webkit, el Geko, l'Opera en la darrera versió del navegador de Microsoft (**IE9**), que finalment intenta complir les especificacions del W3C.

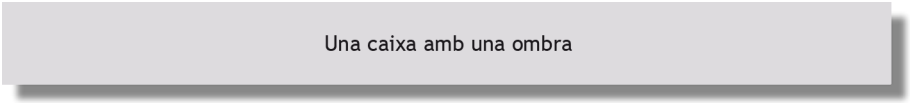
2.3.2. Ombres

En la versió 2 de l'especificació es va introduir la possibilitat de generar ombres (*box-shadow*, *text-shadow*) directament des de codi CSS, es va treure en la versió 2.1 i s'ha tornat a introduir en la versió 3.

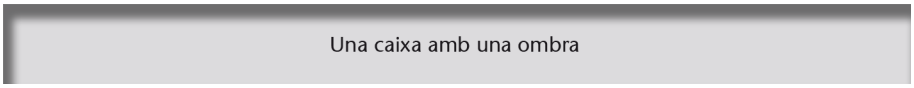
Les propietats per a generar ombres en una caixa són les següents:

```
#example1 {
-moz-box-shadow: 10px 10px 5px #888;
-webkit-box-shadow: 10px 10px 5px #888;
box-shadow: 10px 10px 5px #888;
}
```

Generarà:

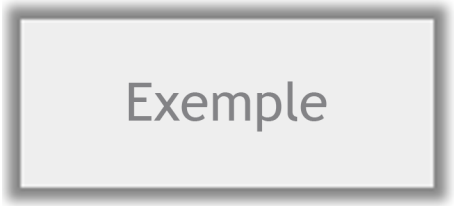


La propietat ***box-shadow*** admet una llista de cinc elements que defineixen en ordre, desplaçament horitzontal, desplaçament vertical, desenfocament, extensió i color de l'ombra. Opcionalment, s'hi pot afegir la paraula clau ***inset***, que ens permetrà fer una ombra interna en lloc d'externa:



Vegem-ne alguns exemples:

```
#Example_F {
-moz-box-shadow: 0 0 5px 5px #888;
-webkit-box-shadow: 0 0 5px 5px#888;
box-shadow: 0 0 5px 5px #888;
}
```



La propietat ***text-shadow*** es comporta de la mateixa manera, i ens permet generar ombres als textos.

D'altra banda, podem crear més d'una ombra per element, o sigui que les podem superposar i obtindrem un efecte de paper de ceba. Per a aconseguir aquest efecte hem de fer el següent:

```
box-shadow: 0 0 10px 5px black, -20px 0px 50px blue;
```

Generarà dues ombres, una de negra i una de blava. En podem encadenar de diferents, simplement separant la llista per comes.

2.3.3. Múltiples imatges de fons

Una altra característica interessant del CSS3 és la possibilitat d'incloure múltiples imatges de fons en un mateix element. Així, el codi següent és completament funcional:

```
#example1 {
width: 500px;
height: 250px;
background-image: url(fons1.png), url(fons2.png);
background-position: center bottom, left top;
background-repeat: no-repeat;
}
```

Fixeu-vos que definim dues imatges de fons, *fons1.png* i *fons2.png*, i també dues posicions diferents per a cadascuna: una posició centrada i a la part inferior, i una altra a la part superior esquerra.

2.3.4. Filets amb imatges

Podem definir imatges per als filets (*borders*) dels nostres blocs de la manera següent:

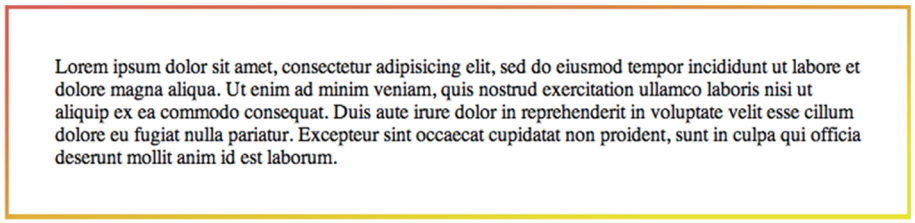
```
border-image: url(border-image.png) 25% repeat;
```

Definirem la imatge que volem emprar com a filet. El percentatge és relatiu a la part que volem emprar i finalment si volem que repeteixi el mode patró, o volem que només aparegui una sola vegada. A primer cop d'ull sembla complicat, però ens hi podem endinsar una mica, amb un exemple. Donada la imatge següent:

Si la volem emprar com a fons, utilitzem el codi següent:

```
#un {
   padding:30px;
   border-width:10px 10px 10px 10px;
   -webkit-border-image: url('css3_imatge_fons.png') 10 100 10 10 repeat stretch;
   background-color:#fff;
   }
```

N'obtenim el resultat següent:



Si ens hi fixem, primer definim una amplada de filet (10 píxels) amb la propietat ***border-width***, després definim la imatge que utilitzarem i les proporcions en píxels (també admet percentatges) d'imatge per a cadascuna de les cantonades. Finalment, tenim la propietat (***stretch/repeat***) per si volem que repeteixi o s'ajusti.

2.3.5. Columnes de text

Una altra de les novetats és la possibilitat de treballar amb columnes de text. Realment, amb l'amplada actual de les pantalles ha esdevingut necessari, ja que una amplada massa gran en els paràgrafs de text n'afecta la llegibilitat. Així, des de l'especificació 3, podem utilitzar dissenys amb columnes. Les propietats que hem d'emprar són:

```
-moz-column-count:3;
-webkit-column-count: 3;
-moz-column-width: 200px;
-webkit-column-width:200px;
-moz-column-gap: 20px;
-webkit-column-gap: 20px;
```

Ens generarà una estructura de tres columnes sempre que puguin complir l'amplària de columna preferida fixada amb ***-column-width***; si no la fixàvem, empraria la part proporcional de l'àrea disponible. Amb la propietat ***-column-gap: 20px*** definim el marge que volem entre columnes, i també disposem de la propietat ***column-rule***, amb la qual podem definir un filet que divideixi les columnes.

Ens generarà el següent:

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

2.3.6. WebFonts

Una altra de les possibilitats que resulten molt atractives a l'hora d'emprar els CSS3 és la capacitat d'adjuntar tipografies a un document HTML. Com totes les coses relatives a la indústria dels navegadors, hi ha diferents sistemes, formats i evidentment llicències de les tipografies.

A partir de l'especificació 3, disposem de l'etiqueta *@font-face*, que com veiem en l'exemple següent ens permet definir una tipografia:

```
@font-face {
   font-family: Gentium;
   src: url(gentium.otf);
}
```

Com hem vist en l'exemple anterior, aquesta etiqueta ens permet adjuntar al document una tipografia, però els problemes vénen en els diferents formats que cada navegador i versió entén de l'arxiu de tipografies. El resultat per a una visualització correcta en tots els navegadors és el següent:

```
@font-face {
font-family: 'UbuntuRegular';
src: url('webfont.eot');
src: url('webfont.eot?#iefix') format('embedded-opentype'),
   url('webfont.woff') format('woff'),
   url('webfont.ttf') format('truetype'),
   url('webfont.svg#UbuntuRegular') format('svg');
font-weight: normal;
font-style: normal;}
```

I d'aquesta manera obtenim una visualització correcta en totes les versions dels navegadors. Per a generar totes les versions de la font, cal que emprem una eina com la següent:

http://www.fontsquirrel.com/fontface/generator

Donat un arxiu TTF, o OpenType, aquesta eina ens generarà tant les versions que hem de pujar al servidor com la definició CSS que haurem d'emprar en el nostre document.

Veient l'embolic existent i la necessitat dels dissenyadors de treballar amb tipografies personalitzades, han aparegut serveis en línia (*on-line*) que ens permeten adjuntar tipografies als nostres documents sense haver-nos de preocupar ni de llicències ni de conversions d'arxius. Actualment els dos serveis més estesos són:

- El servei comercial TypeKit (http://typekit.com/): es preocupen de llicenciar-nos i servir-nos l'ús d'una tipografia comercial.

- El servei de Google WebFonts (http://www.google.com/webfonts): podem adjuntar tipografies *opensource* als nostres documents.

Tots dos serveis s'encarreguen de servir les diferents versions de la tipografia per a cada versió del navegador, facilitant-nos un petit codi CSS adjuntable al nostre CSS i que ens donarà accés a l'ús de la tipografia.

2.3.7. *MediaQueries*

Des de la versió 2.1 del CSS, hi ha la possibilitat de definir estils segons l'ús del full: un per a pantalla (***screen***), un altre per a impressió (***print***) i un altre per a lectors de veu (***voice***). A partir de l'especificació 3 es fa un pas més enllà, i ens permet definir estils específics per a diferents mides de pantalla. La definició següent:

```
@media screen and (max-width: 600px) {
   .class { background: #ccc; }
}
```

Defineix estils específics per a navegadors amb una amplària de pantalla més petita que 600 píxels. També ho podem fer directament amb un full d'estils, emprant l'etiqueta:

```
<link rel="stylesheet" media="screen and (min-width: 600px)" href="small.css" />
```

En aquest cas, ens referim a mides de pantalla a partir de 600 píxels.

Hi ha una petita i subtil diferència si emprem ***min-width*** o ***min-device-width***. Amb la primera fem referència a l'àrea visible en aquests moments al dispositiu, mentre que amb ***device-width*** fem referència a la resolució del dispositiu.

La possibilitat de poder treballar així realment ens ofereix solucions òptimes, ja que podem emprar només CSS per a crear una versió mòbil d'un lloc web.

Web recomanat

En podem veure molts exemples al web:
http://www.mediaqueri.es

2.4. Transicions CSS

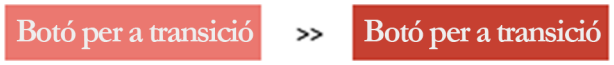
Les transicions CSS són petits canvis en propietats dels fulls d'estil, desencadenats per interaccions de l'usuari, com per exemple quan el ratolí passa per sobre d'alguna cosa (***:hover***) o un camp de formulari canvia. En una transició aquests canvis en les propietats es produeixen de manera progressiva durant un interval de temps.

Suposem un exemple simple, per a entendre com funcionen les transicions CSS:

```
<a href="#" class="boto">Botó per a transició</a>
a.boto {
   text-decoration:none;
   color:#fff;
   padding: 5px 10px;
   background: #f76c6c;
   -webkit-transition-property:background;
   -webkit-transition-duration: 0.5s;
   -webkit-transition-timing-function:ease;
```

```
}
a.boto:hover {
   background: #d22828;
}
```

Generarà la imatge següent, i en passar-hi el ratolí per sobre (***hover***) desencadenarà la transició corresponent:



Del codi següent cal que expliquem el següent:

- La transició no s'ha de posar en l'esdeveniment, que en aquest cas és el selector ***hover***, ja que si la té ja la pot fer quan toqui.

- El que definim en la transició és la propietat de la transició que volem animar, en aquest cas el ***background***.

- Podem definir la durada i també la interpolació de temps de la transició. En aquest cas utilitza una funció d'acceleració, però disposem de les funcions següents: ***ease***, ***linear***, ***ease-in***, ***ease-out***, ***ease-in-out*** i ***cubic-bezier***.

- També podem definir un retard en l'execució de l'animació amb la propietat ***-webkit-transition-delay: 0.5s;***.

De tota manera, hi ha un accelerador per a les transicions que té la forma següent:

```
-webkit-transition: background 0.3s ease 0.5s;
```

A efectes pràctics, fa el mateix que la definició anterior, però d'una manera més compacta. També s'ha de dir que es poden definir transicions de més d'una propietat separant-les amb una coma (,):

```
-webkit-transition: background .3s ease, color 0.2s linear;
```

Les transicions amb CSS només són disponibles als navegadors basats en el Webkit, en l'Opera i en la versió 4 del Geko. Per això, cal que afegim l'accelerador per comptabilitat:

```
-webkit-transition: background .3s ease, color 0.2s linear;
-moz-transition: background .3s ease, color 0.2s linear;
-o-transition: background .3s ease, color 0.2s linear;
transition: background 3s ease, color 0.2s linear;
```

Una altra opció disponible és fer la transició de totes les propietats que canviïn amb la instrucció ***all***:

```
transition: all 3s ease, color 0.2s linear;
```

Web recomanat

Aquí hi trobarem una llista de les propietats que poden ser animades (transicions):

http://www.w3.org/TR/css3-transitions/#properties-from-css-

2.5. Exercicis

1. Feu uns botons CSS amb efectes de degradat intern i ombra exterior i animació al *hover*.

2. Maqueteu un petit *layout* fent servir *mediaquery*, *webfonts*, cantonades rodones (*borders*) en les imatges i amb *fallback*.

3. Biblioteques JavaScript

3.1. Per què una biblioteca?

El JavaScript és el llenguatge de programació utilitzat en el desenvolupament d'aplicacions web per la banda del client. Recordant-ne una mica la història, el JavaScript neix com a llenguatge el 1995 gràcies a la Netscape Corporation, que l'incorpora com a llenguatge de ***script*** en la primera versió del client de WWW. Paral·lelament, **Microsoft** inicia el desenvolupament del seu client de WWW, l'**Internet Explorer**, i copia el llenguatge de Netscape i en canvia el nom: l'anomena *jScript*. Realment els dos llenguatges són molt semblants, però són diferents.

Des de bon principi, es generen diferencies en l'ús, en la manera com s'interactua amb el DOM, el sistema d'esdeveniments i moltes altres petites peculiaritats que els fan diferents. Així, ens trobem amb un llenguatge, però que ha d'interactuar amb models de classes diferents i utilitza sistemes d'esdeveniments diferents.

Al principi la programació de client era terriblement difícil, ja que calia treballar amb cadascuna de les especificacions per als diferents navegadors, i això feia que el codi que es generava fos poc robust i mantenible. En el desenvolupament d'una petita funció era fàcil trobar-se amb dos condicionals per a cada especificació de navegador.

Per a solucionar aquests problemes d'interacció del llenguatge amb els navegadors, neixen **biblioteques** amb l'objectiu d'aconseguir una API (***application programming interface***) comuna en els diferents navegadors.

En aquest mòdul exposarem el que al nostre entendre són les principals biblioteques JavaScript en el desenvolupament d'aplicacions web. Abans, però, especificarem les tasques que ens calen per a desenvolupar aplicacions web en el client i així avaluar i veure com tracten això les diferents biblioteques.

3.2. Què ens ha d'oferir una biblioteca JavaScript?

Fonamentalment, una solució als dos reptes bàsics que afrontem quan desenvolupem aplicacions web:

- Interactuar amb el **DOM**: seleccionar, afegir, modificar i esborrar nodes, seleccionar conjunts de nodes i aplicar-hi estils CSS i generar contingut nou.

- Interactuar amb l'usuari mitjançant el sistema d'esdeveniments: capturar accions de ratolí, de teclat i processar-les de la manera corresponent.

També ens hauria d'oferir un sistema unificat de comunicació amb el servidor de manera asíncrona (Ajax), un sistema per a treballar amb formularis de dades (una forma d'interacció amb el **DOM**) i especificar components de programari tipus *widget* que ens permetin expandir les funcionalitats pròpies del navegador.

Usualment una **biblioteca** també ofereix un conjunt de convencions a l'hora de desenvolupar programari, és a dir, una manera de fer, una filosofia de treball.

Podem afirmar que una biblioteca per al JavaScript ens ha de permetre desenvolupar les nostres aplicacions web de tal manera que no ens hem de preocupar per les diferències i les incompatibilitats entre navegadors.

3.3. Quines biblioteques estudiarem?

Actualment, hi ha multitud de biblioteques per al JavaScript, però aquí ens interessa estudiar només les més importants. Així, bona part del treball el farem amb una de les **biblioteques** més esteses: la biblioteca *jQuery*, creada per John Resig.

S'estima que tres de cada quatre llocs web que utilitzen una biblioteca fan servir *jQuery*, i la utilitzen empreses com Amazon, Microsoft, BBC i Twitter. Segurament, també és la més fàcil d'utilitzar. Amb aquesta biblioteca, aprendrem a manipular el DOM, a treballar amb esdeveniments, a generar peticions al servidor no síncrones i a utilitzar-ne la capa de components (*jquery-ui*).

Estudiarem el ***Prototype***, que va ser una de les primeres biblioteques que es van utilitzar i que fan servir empreses com Apple. També veurem de passada les biblioteques **YUI**(escrita per Yahoo) i MooTools.

De totes aquestes biblioteques en veurem els dos pilars bàsics: la interacció amb el DOM i el sistema d'esdeveniments.

4. *jQuery*

4.1. Obtenir *jQuery*

Podem baixar *jQuery* del seu web (www.jquery.com), però també la podem utilitzar directament des dels **CDN (*content delivery network*)** de Google. Si accedim al seu web, veiem que disposem de dues versions diferents: la versió ***production, minified and gziped*** està preparada per a entorns de producció, amb el codi comprimit i optimitzat perquè ocupi molt pocs kilobytes de baixada; i la versió, de desenvolupament. Només hem de baixar aquesta darrera si el que volem és revisar i llegir el codi de la biblioteca (és a dir, veure com està feta), ja que la versió de producció és completament utilitzable.

La comunitat recomana utilitzar *jQuery* des d'un **CDN**. Un **CDN** és una xarxa global de distribució de continguts, amb servidors geolocalitzats de manera òptima, per a millorar els temps de baixada. Si enllacem la biblioteca des d'aquí, quan el client de web intenta baixar-la, moltes vegades es troba que ja la té a la **memòria cau** (***cache memory***).

Sigui com sigui, caldrà que enllacem la biblioteca en el nostre document HTML utilitzant l'etiqueta estàndard d'HTML <script> fent:

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

O bé utilitzarem l'etiqueta amb una adreça CDN.

```
<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/1.6.1/
jquery.min.js"></script>
```

Una vegada fet això, ja podrem començar a utilitzar la biblioteca completa per a fer les nostres aplicacions.

4.2. Com funciona *jQuery*: l'objecte *$*

Com veurem més endavant, algunes de les biblioteques del JavaScript embolcallen a partir d'un objecte global. D'aquesta manera l'espai de memòria queda net i podem implementar la biblioteca en espais complexos amb moltes altres funcions.

jQuery utilitza el símbol *$()* com a funció que ens permetrà interactuar amb la biblioteca. De fet, el símbol *$* és un sinònim de la vertadera funció *window.jQuery*. Així:

```
var jQuery = window.jQuery = window.$
```

A partir d'aquesta funció, s'embolcallen totes les funcionalitats de la biblioteca, i podem fer qualsevol operació. Per a compatibilitzar-la amb altres biblioteques (***Prototype*** utilitza el mateix símbol), *jQuery* ens ofereix la funció ***jQuery.noConflict();***, que desactiva el símbol *$()* i el deixa amb *jQuery()*.

4.3. Interactuar amb el DOM

El nucli de *jQuery* és interactuar amb el DOM. De fet, la funció *$()* ens permet **seleccionar**, recuperar referències a elements del DOM i a partir d'aquí començar a interactuar amb el llenguatge de programació. No és convencional, ja que no segueix l'estructura usual de programació d'aplicacions, però sí que és molt pràctic, ja que emprem algun element del DOM en el 80% del que programem al navegador. Per aclarir-ho una mica tot plegat, vegem-ne un exemple.

```
<html>
<head>
<title>Exercici jQuery 2</title>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.6.1/jquery.min.js"> </script>
<script type="text/javascript">
<!--
$(document).ready(function(){ // 1
   var resultat = $('#p1').text(); // 2
alert(resultat)
})
//-->
</script>
</head>
<body>
   <p id="p1">Hola, mon!</p>
</body>
</html>
```

El codi que apareix a **1** encara no és rellevant, però serveix per a programar l'esdeveniment ***DOM disponible***. Si ens fixem en la part de **2**, amb ***$('#p1')***, seleccionem l'element amb ID igual a ***"p1"***. De fet, el que ens retorna la funció ***$*** és una instància de la biblioteca *jQuery* inicialitzada amb l'element *p* del DOM. A partir d'aquí, el mètode *text()* ens retorna el contingut del node com a cadena de text, el guardem a la variable resultant i la mostrem fent un avís (*alert*).

4.3.1. Selectors, filtres i CSS

Com es veu molt bé en l'exemple anterior, la funció *$()* rep un selector CSS i ens retornarà la instància de l'objecte. La qüestió és veure quins selectors podem utilitzar.

El primer selector és el #, igual que amb CSS, cosa que ens permet seleccionar elements per ID. De la mateixa manera, podem utilitzar selectors amb classes CSS. Així, la crida ***$('.estil1')*** seleccionarà tots els elements que tinguin associada la classe *estil1*. Com veiem, utilitzem el mateix llenguatge que ja coneixem amb CSS a l'hora d'interactuar amb el DOM.

Cal fer notar que ambdues crides retornen una instància de la biblioteca, a punt per a emprar-la, amb la diferència que en la primera ens retorna la instància, inicialitzada amb un sol element, i en la segona, amb un conjunt d'elements. La manera com hem seleccionat els elements del DOM era una tècnica habitual per a manipular en temps d'execució el contingut de l'HTML.

Gairebé sempre podem aplicar la mateixa regla de CSS als selectors de *jQuery*:

$('*')	Tots els elements
$('#propietat)	Element amb *id=propietat*
$('.classe')	Elements que tinguin la classe *css*
$('img')	Elements
$('img,p,selectorN')	Combina els elements *img*, *p* i *selectorN*

Igual que amb CSS3, podem seleccionar per atribut:

$('a[rel="nofollow"]')	Tots els *a* amb l'atribut *rel="nofollow"*
$('[atribut|="valor"]')	Atribut que sigui *valor* o comenci per *valor*
$('[attribute~="value"]')	Atribut que contingui *valor* delimitat per espais
$('[atribut!="valor"]')	Que l'atribut no tingui el valor
$('[atribut^="valor"]')	Que el valor de l'atribut comenci per *valor*
$('[atribut$="valor"]')	Que el valor de l'atribut acabi amb *valor*
$('[atribut]')	Que l'element contingui l'atribut
$('[atribut1="1"][atribut2="2]')	Que continguin tots els atributs amb valors

De la mateixa manera que en CSS3, disposem de filtres, però en tots els casos no cal que ens preocupem per la compatibilitat de les diferents versions dels navegadors admetent certes propietats, ja que això és feina de la biblioteca. Així, podem utilitzar filtres com els següents:

$('tr:first-child')	La primera fila d'una taula. També podem emprar *:last-child*, i *:nth-child(4n)*
$("tr:even")	Fila senar, també parell amb ***odd***

$('tr:eq(3)')	El tercer element del conjunt seleccionat. En aquest cas la tercera fila del conjunt de totes les taules de la pàgina. També podem emprar: • ***gt(n)***: conjunt més gran que *n* • ***lt(n)***: conjunt més petit que *n* • ***not(n)***: tots, excepte *n*
$('tr:first')	El primer element. També el darrer emprant ***:last***

També podem filtrar elements depenent del contingut que tenen:

$("div:contains('Llibre')")	Seleccionarà els elements *div* que continguin la paraula *Llibre*
$("div:has(p)")	Elements *div* que **continguin** un altre element *p*
$("td:parent")	Elements que són parells d'un altre node, inclosos nodes de text

Finalment, per a completar l'apartat de selectors, ens cal veure una nova funcionalitat i detallar una diferència en els objectes que ens retorna.

Quan seleccionem múltiples objectes, com per exemple ***$('img') per a seleccionar totes les imatges***, l'objecte que ens retorna *jQuery* és una instància normal augmentada amb les funcions de llista (***iterable***). *jQuery* ofereix una funció, ***.each***, que ens permet iterar pels diferents elements que hem seleccionat. Així, podem escriure el codi següent:

```
$('img').each(function(index, element) {
console.log( element.src )
});
```

Iterarà per totes les imatges del document. Per a saber si un selector ens ha retornat un conjunt o només un element, hem de consultar la propietat *length*:

```
$('img').length; // Retornarà el total d'imatges que conté el document.
```

La funció ***$()*** admet un segon paràmetre que ens permet definir el context en el qual volem que es faci la cerca del selector. Per defecte, si el paràmetre no existeix, la funció fa la cerca a tot el document. Si apliquem un context només la farà al context; així:

```
<div><p>Content</p></div>
<div id="p1">
<p>Content 2</p>
</div>
$('p', $('#p1') );
```

Retornarà només el segon *p*, ja que és l'únic que trobem dintre el context de *#p1*.

4.4. Manipular el DOM

Ara que ja sabem seleccionar elements d'un document, podem fer un pas enllà i començar a manipular-los. Podem canviar-hi propietats CSS, llegir i modificar-ne els atributs, copiar-los, eliminar-los i afegir-ne.

4.4.1. Propietats CSS

Podem manipular les propietats CSS d'un conjunt d'elements seleccionats amb el mètode *.css*:

```
$('p').css('color', 'red);
```

Canviarà el color del text de tots els paràgrafs a vermell. També, però, podem canviar moltes propietats a la vegada; per a fer això, la funció rebrà un objecte *javascript* que quedarà en el format següent:

```
$('p').css({ 'color': 'grey',
'padding': '5px',
'background-color': 'yellow' });
```

La majoria de mètodes de l'objecte *jQuery* que poden admetre un parell de paràmetres també poden admetre un objecte, i així fer molta més feina d'una sola vegada.

De la mateixa manera que els podem canviar també els podem consultar, demanant la propietat:

```
$('p').css('color')
```

Ens retornarà el color de l'element *p*.

Cal dir que hi ha alguns mètodes a la manera d'accés directe, com per exemple:

.show()	Ens mostrarà l'element si era amagat
.hide()	L'amagarà
.toggle()	Actua a la manera d'interruptor. Si l'element és amagat el mostra i si l'element és visible l'amaga

Des del JavaScript, amb ***jQuery*** podem afegir una nova classe CSS a un element seleccionat amb l'ordre:

```
$('#id').addClass('nomdelaclasse')
```

O podem consultar si té assignada la classe amb:

```
$('#id').hasClass('nomdelaclasse')
```

O bé la podem eliminar:

```
$('#id').removeClass('nomdelaclasse')
```

També podem manipular i consultar l'alçària i l'amplària d'un element mitjançant els mètodes ***.height()*** i ***.width()***, que tant ens serveixen per a assignar una alçària i una amplària com per a consultar l'alçària i l'amplària reals. S'ha de tenir en compte que la mida d'un element que ens donarà les funcions no computa els ***paddings***. La mida amb ***padding***, però sense el ***border***, la podem saber si consultem els mètodes ***innerHeight()*** i ***innerWidth()***.

4.4.2. Atributs dels nodes

De la mateixa manera que podem consultar o modificar propietats CSS, *jQuery* ens facilita una fórmula per a interactuar amb els atributs de cada node. Emprant la funció:

```
<img width="34" />
$('img').attr('width', '44')
```

De la mateixa manera que el CSS, el mètode admet un objecte com a paràmetre, però si només hi enviem un paràmetre que és una propietat, ens retornarà el valor pertinent.

4.4.3. Afegir contingut al DOM

Per a inserir contingut dintre del **DOM**, disposem de tres fórmules bàsiques en relació amb la selecció corresponent:

1) Afegir un node embolcall

L'acció d'afegir un node embolcallant ***.wrap*** del selector en curs, generarà el següent:

```
<p id="un">Una prova</p>
$('.inner').wrap('<div class="new" />');
<div class="new">
   <p id="un">Una prova</p>
</div>
```

D'aquesta manera, quan fem un ***.wrap*** d'un selector, l'embolicarem amb l'etiqueta inserida. De la mateixa manera podem fer un ***.unwrap()***.

També hi ha el mètode ***.wrapAll***, que embolcallarà tot el conjunt de selectors en un sol *div*.

2) Afegir a dintre

Ens afegirà el contingut, passat com a paràmetre, en diferents posicions del selector segons el mètode que fem servir:

.append('content')	Afegeix el contingut al final del selector
.prepend()	Afegeix el contingut al principi del selector
.html()	Recupera el contingut d'un selector o l'hi assigna. Sempre contingut HTML
.text()	Recupera el text d'un selector, sense etiquetes. També l'assigna

3) Afegir a fora

$().after() i ***$().before()*** insereixen els elements seleccionats just abans o després del node seleccionat. Si la selecció ens retorna diferents nodes, farà l'operació per a cadascun d'aquests nodes.

4.4.4. Esborrar nodes

Com que podem esborrar qualsevol selector simplement executant el mètode ***.remove()***, també podem utilitzar el mètode ***.empty()*** per a buidar el contingut d'un node, inclòs text pla.

4.5. Funcions generals

Com a biblioteca o marc de treball, *jQuery* ens ofereix un seguit d'extensions, mètodes o funcions que ens permeten afegir petites utilitats i funcionalitats al JavaScript:

$.each(collection, callback(indexInArray, valueOfElement))

És una funció que ens permet programar iteracions. Suposem que:

```
a = [1,2,3,4,5]
resultat = 0
$.each(a, function(index, valor) { resultat += a }
alert('el resultat es' + a)
```

Com a ***collection***, podem passar qualsevol element que sigui iterable en el JavaScript, com per exemple una *array*, o el resultat d'un selector. De fet, un patró comú quan volem fer coses amb un conjunt és:

```
$('.item').each(function(ind) {
$(ind).text()
 })
```

En l'exemple anterior, iterem per la col·lecció de nodes que tenen assignat l'estil ***item***. En aquest patró, és molt interessant saber que ***$(ind)*** és el node actual.

També podem iterar sobre les propietats d'un objecte o ***hash***:

```
$.each( { name: "Pere", lang: "JS" }, function(k, v){
 alert( "Clau: " + k + ", Valor: " + v );
 });
```

$.extend({}, objecte1, objecte2)

El mètode *extend* ens permet afegir propietats a un objecte definit prèviament:

```
a = {un: 'hola', dos:'dos' }
b = {un:'si', tres:'quepassa'}
$.extend(a, b)
a == {un:'si', tres:'quepassa', dos:'dos'}
```

És de gran utilitat per a escriure codi modular, ja que ens permet ampliar d'una manera molt fàcil objectes amb propietats i mètodes d'altres objectes. Una aplicació pràctica és la creació de ***mixins***, objectes que amplien la funcionalitat d'altres objectes, com per exemple:

```
var a = {
'hola': function() { return 1; },
'adéu': function() { return 2; } }
var mixin = { 'hola': function() { return 2; } }
$.extend(a, mixin)
console.info( a.hola() == 2 )
console.info( a.adeu() == 2 )
```

Si ens fixem en l'exemple, disposem d'un objecte *A*, que té una funció *hola*, que retorna 1. Però després hi apliquem, amb el mètode ***$.extend***, l'objecte *mixin*, sobreescrivint la funció *hola*.

Al final de l'exemple, veiem que ***a.hola()*** ja no retorna 1, sinó que retorna 2, ja que ha estat sobreescrita (***estesa***).

Si revisem el codi font de ***jQuery***, apreciarem que la majoria de mètodes nous els instal·len d'aquesta manera. També ho podem fer estenent la cadena de prototipatge:

```
$.extend(a.prototype, mixin)
```

Ens generaria mètodes públics de l'objecte *A*. Mentre que l'objecte *A* de l'exemple simplement és un paquet de funcions (***namespace***).

$.inArray(valor, array)

Cerca valor a la llista. Si no el troba retorna –1. Si el troba, retorna la posició >0.

$.isArray(p)

Retorna ***true*** si *p* és una *array*.

$.merge(a, b)

Barreja el contingut de *a* i *b* i retorna una nova ***array***.

Web recomanat

Trobareu moltes més utilitats a l'adreça:
http://api.jquery.com/category/utilities/

4.6. Sistema d'esdeveniments

A banda de poder manipular fàcilment el **DOM** des dels nostres programes, l'altre gran pilar de les biblioteques del JavaScript és gestionar els **esdeveniments** (***events***). Totes les persones que s'han volgut enfrontar a aquesta tasca sense utilitzar cap biblioteca saben que no és trivial, ja que cada navegador ens ofereix un sistema d'esdeveniments diferent.

La manera genèrica d'escoltar un esdeveniment en *jQuery* és mitjançant l'ordre ***.bind***. Per a capturar els clics a tots els enllaços podem escriure:

```
$('a').bind('click', function() {
   contador += 1
})
```

Sempre segueix el mateix patró: seleccionem el node al qual volem assignar l'esdeveniment, i amb l'ordre *bind* hi podem assignar l'esdeveniment que vulguem. Per exemple, per a assignar un esdeveniment *change* a un camp desplegable (***select***), escriurem:

```
$('#idselect').bind('change', manipulador)
```

En què ***manipulador*** pot ser, com en el cas anterior, una funció anònima, o pot ser el nom d'una funció (la variable que conté l'objecte *funció*).

Cal dir que a partir de la versió de ***jQuery*** 1.4, podem assignar a la funció *bind* un objecte que contingui una llista d'esdeveniments mapats com per exemple:

```
$('a').bind({
   'mousenter': function(evt) {
   $(this).css('color', 'black')
},
   'click': function(evt) {
$(this).toggle();
}
});
```

És important fer notar que, en la funció manipuladora de l'**esdeveniment**, la variable *this* apunta al node del DOM que la desencadena, i que, per tant, l'ordre ***$(this)*** selecciona el node mateix.

Així mateix, la funció rep un paràmetre que conté un objecte de tipus ***event***. Aquest paràmetre sovint és omès, però amb aquest paràmetre podem accedir a informació addicional en el moment de generar l'esdeveniment i durant la propagació d'aquest esdeveniment. Amb ***jquery*** el sistema d'esdeveniments segueix una regla de propagació en bombolla (*bubbling*) per la qual quan un esdeveniment és generat en un node es propaga cap als seus pares. Imaginem-nos l'exemple següent:

```
$(document).ready(function(){
   $('tr').click(function(){
      alert('clic a tr');})
      .height(30)
      .css( 'background-color', '#eaeaea');
   $('a').click(function(e){
      alert('clic a a');
      e.stopPropagation()
      });
   });
   </script>
</head>
<body>
<table style="border:1px solid black;">
   <tr>
      <td><a>Hola</a></td>
   </tr>
</table>
```

Si fèiem clic a l'enllaç *a*, el capturaríem des de la funció de l'esdeveniment, però com que l'esdeveniment seguiria el seu procés de propagació, també executaria el manipulador assignat al node pare *tr* i ens generaria un doble ***alert()***, el de l'element *a* i el de l'element *tr*. O sigui, s'executarien tots dos esdeveniments.

Per a impedir això i trencar la cadena de propagació, utilitzaríem l'objecte *event* que rep el manipulador i cridaríem al mètode ***.stopPropagation()*** tal com hem fet en l'exemple.

En l'objecte *event* també hi podem trobem altres dades com les propietats ***pageX*** i ***pageY***, que són les coordenades de ratolí, on s'ha desencadenat l'***event***. També dintre d'aquest objecte hi trobem propietats com ***target***, que és l'objecte que genera l'esdeveniment, o bé ***currentTarget***, que és l'objecte des d'on s'ha iniciat la propagació.

De la mateixa manera que assignem un esdeveniment amb el mètode ***bind***, el podem desassignar amb el mètode ***.unbind***:

```
$('tr').unbind('click');
```

Els esdeveniments es poden executar de manera manual emprant l'ordre ***.trigger****('nomevent')*:

```
$('tr').trigger('click');
```

S'executarà el *handler* assignat a l'esdeveniment (sempre que l'hàgim assignat *a priori*). I de la mateixa manera, podem generar els nostres propis esdeveniments.

Imaginem-nos que tenim un rellotge i volem que ens notifiqui el temps cada segon. Des de la funció que controla el temps que passa, podem generar un esdeveniment de tipus ***'segon'*** (el nom és a voluntat nostra), com fem en l'exemple següent:

```
var segons = 0
setInterval(function(){
$('#p1').text( ++segons )
      .trigger('segon', [segons])

}, 1000)
$('#p1').bind('segon', function(esdeveniment, data){
   if(data==10) segons = 0
})
```

Utilitzem l'element *#p1* perquè ens generi un esdeveniment de segon, després el capturem i fem que la variable que compta els segons que passen s'inicialitzi quan arriba a 10. Perquè això funcioni, necessitem tenir un node amb *id #p1* al nostre HTML.

4.6.1. Taula d'esdeveniments

blur	Quan en un camp de formulari perdem el focus del teclat
focus	Quan un element d'un formulari rep un clic del ratolí
load	Un element extern acaba el procés de càrrega, com per exemple una imatge
resize	La finestra canvia de mida, pertany al Window
scroll	Fem *scroll* a la finestra o a un element *div*
click	Fem clic sobre un node
dblclick	Fem doble clic sobre un node
mouseover	Passem el ratolí per sobre de l'element
mouseout	El ratolí surt de l'element
change	El valor d'un camp de formulari canvia
submit	Un formulari és enviat cap al servidor
keydown, keyup	Premem una tecla del teclat, que podem trobar a la propietat *wich*, de l'objecte *event* passat al manipulador
error	Es desencadena, per exemple, quan des del servidor no es carrega una imatge

Hi ha accessos directes a la majoria de propietats que podem utilitzar amb un *bind*, amb el seu nom de funció directament. Així, podem cridar als mètodes d'un selector ***.click***, ***.change***, ***.error***, etc.

4.7. Formularis

Una altra de les tasques clau en la programació d'aplicacions web per al JavaScript és la manipulació, construcció i sobretot validació de formularis. ***jQuery*** ens ofereix tota una sèrie de mètodes dissenyats especialment per a consultar i validar formularis. Així, podem accedir al valor de qualsevol camp de formulari a partir del seu selector fent:

```
$('#id').val()
```

El mateix mètode ens serveix per a assignar un valor. Per a fer-ho hi enviarem com a paràmetre el valor que s'ha d'assignar.

També podem capturar el moment d'enviar el formulari i programar una funció que decideixi si es pot enviar o no, segons el contingut dels camps, utilitzant l'esdeveniment ***submit***. Així, podem fer:

```
$('formulari#un').bind('submit', function(){
```

```
    // Si la validació és correcta.
    return true;
    // Si hi ha errors a la validació,
    // els podem mostrar i s'ha de retornar false.
    return false;
}
```

També hi ha mètodes per a treballar amb **Ajax** que ens permeten serialitzar d'una vegada tot el contingut del formulari fent ***$('form').serialize()***. Ens generarà el formulari preparat per a enviar-lo per GET. Altrament i segons ens interessi, podem utilitzar ***.serializeArray()*** i ens permetrà obtenir el formulari dintre una llista.

4.8. Ajax

Ajax (***asynchronous JavaScript and XML***) consisteix en una tècnica mitjançant la qual podem fer crides al servidor sense recarregar el contingut de la nostra pàgina. Així, podem enviar i rebre dades des del servidor de manera interactiva des del JavaScript. Com sempre, per a utilitzar la tècnica, cada navegador ho implementa d'una manera diferent i, per sort, ***jQuery*** ens facilita una interfície unificada.

Per a fer una petició contra el servidor disposem de la funció ***.ajax***, de caire genèric i que admet molts paràmetres diferents, o dues alternatives molt més directes que ens permeten sol·licitar documents amb **GET** o **POST**.

```
$.get( url, [ data ], success(data))
```

url: És l'adreça que sol·licitarem al servidor.

data: És un objecte JavaScript amb els paràmetres que vulguem enviar al servidor.

success: És un *handler* que s'executarà amb la resposta que ens envia el servidor.

Un exemple senzill de la manera com hem de carregar un document extra des del servidor pot quedar així:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Exercici de càrrega d'Ajax</title>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.5.1/jquery.min.js"> </script>
<script>
$(document).ready(function(){
   $('#bt').click(function(){ // 1
      $.get('arxiu.txt', {hola:3, mon:4}, function(data){ // 2
```

```
            $('#desti').html(data); // 3
        });
    })
});
</script>
<style>
    #desti { color:blue; }
</style>
</head>
<body>
<p>En prémer el botó, carregarem a dintre del paràgraf blau el contingut de l'arxiu .txt del
servidor.</p>
<input type="button" id="bt" value="Carrega" />
<p id="desti">Aquí carregarem un contingut amb Ajax des del servidor.</p>
</body>
</html>
```

En l'exemple, programem (1) l'esdeveniment clic del botó *#bt* que llanci una petició Ajax per mètode GET (2) *$.get*, baixant l'arxiu *.txt* i enviant-hi dos paràmetres (*hola* i *mon*). Com que la petició és asíncrona (no sabem quan ens respondrà el servidor), hi enviem una funció perquè l'executi en aquest moment (2); la funció té un paràmetre *data*, que és el contingut de la petició, i que finalment (3) el col·loquem al node *#destinacio*.

Web recomanat

En podem veure l'exemple a:
http://multimedia.uoc.edu/~jcollell/ejercicio_ajax.html

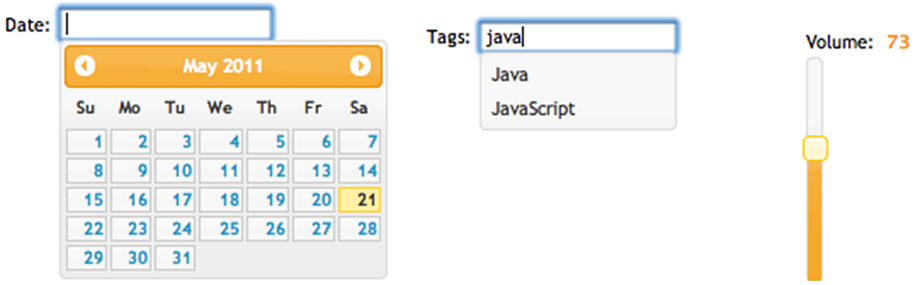
També podem fer peticions al servidor per mitjà de **POST** amb la funció *$.post* i els mateixos paràmetres.

Podem consultar en línia la documentació perquè les tres funcions ens ofereixen multitud d'opcions més per a simplificar el treball.

4.9. Components

Internament, ***jQuery*** no ens ofereix cap sistema de components (*widgets*), però sí que hi ha un projecte paral·lel que ofereix determinats components. El projecte s'anomena *jqueryUI*, i el podem revisar a www.jqueryui.com. Encara que molt lligat a *jQuery*, funciona de manera independent amb calendari propi de versions.

Fonamentalment, ens ofereixen components visuals, components interactius, efectes i una utilitat. Pel que fa a components visuals, tenen un petit component de calendari, un camp d'autocompleció (mentre s'escriu filtra resultats d'una llista) i pestanyes, entre d'altres.



A part dels components visuals, les ***jqueryUI*** també ens ofereixen un seguit de components funcionals com per exemple els ***draggables*** i els ***sortables***: els primers ens permeten programar objectes arrossegables per la pantalla, i els segons, objectes ordenables gracies a ordenació.

Tots els components segueixen la mateixa nomenclatura i funcionament que la biblioteca mare. Hi ha un web, amb exemples, referència i també un selector de pells, que ens permetrà configurar el component adaptat a les necessitats de disseny del nostre projecte.

4.10. Patrons d'ús

Com ja hem dit anteriorment, un entorn de treball o una biblioteca, a més d'una sèrie d'instruccions, objectes i mètodes per a facilitar-nos la feina, impliquen certs patrons de treball a manera de convencions que faran que l'execució d'aplicacions sigui més robusta i més convencional per a la majoria de programadors del marc.

```
$(document).ready(function() {
   // Codi de programació.
})
```

Aquest primer patró empra l'esdeveniment ***ready***, un esdeveniment especial que ens notifica que el DOM del document HTML ja està a punt per a emprar-lo, o dit d'una altra manera, podem emprar de manera segura qualsevol dels elements del document des del nostre programa. L'esdeveniment estàndard per a fer això en el JavaScript és el ***window.onload***. En la tècnica de *jQuery* l'esdeveniment s'executa sense haver baixat les imatges associades amb el DOM, mentre que amb la convencional s'ha d'esperar a la resta i és una mica menys eficient! Cal dir també que podem crear tants *document.ready* com vulguem durant el document, mentre que només hi pot haver un *window.onload*.

Un altre patró d'ús molt interessant és la capacitat de *jQuery* d'encadenar mètodes. Funcionalment no implica gaire, però sí a l'hora d'escriure codi molt llegible, ja que, sobre un mateix selector, podem anar aplicant diferents mètodes. Per tant, és habitual trobar construccions com aquesta:

```
$('#lelemtn')
   .css('color', 'black')
```

```
    .bind('click', alferclic)
    .bind('mouseover', rollover)
```

Un altre patró d'ús molt comú quan es programa en el JavaScript modern consisteix a generar funcions anònimes que s'executen per a "amagar" aquest codi de la resta de codi de la pàgina o del navegador. Per exemple:

```
// Creem una funció anònima per a utilitzar com a embolcall.
(function(){
   // Aquesta variable normalment és global.
   var msg = "Thanks for visiting!";
   // I l'assignem a un mètode global.

   window.onunload = function(){
      // Podem utilitzar la variable definida globalment.
      alert( msg );
   }; // Tanquem la funció i l'executem. A partir d'aquí, la
   // resta de variables són invisibles.
})();
```

5. *Prototype*

Creada el 2005, va ser una de les primeres biblioteques a aparèixer i encapsular tota la potència del JavaScript per a navegadors moderns. Destaca perquè és la primera en moltes coses i perquè és la biblioteca d'ús del potent entorn de treball web Ruby on Rails. També disposa d'una biblioteca de components gràfics i d'efectes visuals anomenada *script.aculo.us*.

5.1. Interactuar amb el DOM

De la mateixa manera que el *jQuery*, la biblioteca ens ofereix una funció per a accedir al **DOM** fàcilment, ***$('idelement')***, que obté una referència a l'element i a més hi afegeix tots els mètodes propis de la classe ***Element.Methods***. També hi ha la funció ***$$('classecss')***, que fa el mateix però a partir de classes CSS. En aquest cas, en lloc de retornar una referència retorna una llista d'elements en el mateix ordre que apareixen al **DOM**. També ens ofereix una funció per a accedir a formularis ***$F()***.

Un cop obtinguda la referència als elements, podem navegar pels diferents nodes veïns en el DOM amb tot un seguit de mètodes:

$("clar").adjacent("li.dona");

Ens retornarà una *array* de nodes veïns al que té *id #clar* i que compleixin el selector.

$("clar").ancestors();

Recol·lecta tots els pares en ordre.

up/down/next/previos

Selecciona el pare, el fill, el següent i l'anterior.

descendants

Retorna els fills.

També disposem d'elements per a modificar el contingut de la pàgina. Podem inserir elements utilitzant la funció ***$("MainDiv").insert({top:"Afegit a l'inici de l'element"})***, que primer obté la referència a l'element per a inserir després el contingut en la posició superior. L'objecte que rep com a paràmetre pot contenir propietats com *top*, *before*, *after* i *bottom*.

Podem esborrar elements amb el mètode ***.remove()***, i també podem emprar ***.update()*** per a actualitzar l'element amb el contingut donat, i ***.replace()***, que el canvia.

Altres funcionalitats que ens ofereix *de facto* permeten calcular posicions relatives i absolutes d'un element (***absolute***, ***relativize***), clonar posicions (***clonePosition***), i també amagar i mostrar (***show***, ***hide***), canviar opacitats (***getOpacity***, ***setOpacity***), mides (***getDimensions***, ***getHeight***, ***getWidth***) i fer retallades (*clipping*) (***makeClipping***, ***undoClipping***).

També ens ofereix mètodes per a treballar amb CSS: ***addClassName***, ***removeClassName*** i ***toggleClassNames*** serveixen per a afegir, treure o intercanviar classes; ***hasClassName*** i ***classNames***, per a consultar, i ***setStyle***, ***getStyle***.

5.2. Sistema d'esdeveniments

La manera general de capturar esdeveniments amb ***Prototype*** és utilitzant la funció:

```
Event.observe(element, eventName, handler [,useCapture = false]);
```

Element pot ser una *string* amb un ID, o directament un element seleccionat. El nom de l'esdeveniment ha de ser l'estipulat pel W3CDOM nivell 2: el *handler* és una funció i la bandera (*flag*) ***useCapture*** determina si utilitzem el sistema de bombolles o no.

Un bon exemple d'ús és:

```
<ul id="PeopleList">
   <li class="noia" id="judy">Joana</li>
   <li class="noi" id="sam">Sam</li>
   <li class="noia" id="amelia">Amelia</li>
</ul> </div>
<textarea id="results" cols="50" rows="10"></textarea>
<script type="text/javascript" src="prototype-1.6.0.2.js"></script>
<script type="text/javascript">
$("results").value = "";
Event.observe("PeopleList", "click", function(e) {
$("results").value += "clicked on " + e.target.id + "\n";
});
</script>
```

En fer clic sobre els elements s'executarà la funció, que escriurà l'ID de l'element premut al camp *textarea*.

La funció *handler* té un paràmetre de tipus esdeveniment, del qual veiem diferents mètodes i propietats: ***Event.target*** (és el node que ha generat l'esdeveniment), ***this*** (és el node on s'ha capturat l'esdeveniment) i ***Event.element*** (és el node que ha generat l'esdeveniment).

.stopObserving() elimina les referencies als esdeveniments i ***unloadCache()*** elimina tots els esdeveniments capturats. ***Event.stop()*** evita que un esdeveniment faci la bombolla.

5.3. Ajax

```
var url = "http://myserver/api/get";
var ajaxCall = new Ajax.Request(url, {
method: 'get',
parameters: {a:1, b:2},
onSuccess: function() { alert('resposta correcta.2xx') },
onFailure: function() { alert('Fallida') },
onComplete: function() { alert('Completada') },
});
```

Amb aquesta simple signatura, ja podem fer peticions contra el servidor. Hem de destacar l'ús de diferents *handlers* que ens permeten capturar la resposta des del servidor.

5.4. Utilitats generals

Una de les mancances de la biblioteca i que sovint rep crítiques és la construcció, ja que basa part del funcionament en extensions que pengen de la cadena de prototipatge dels objectes normals del llenguatge; així, ens ofereix eines per a estendre i personalitzar encara més la biblioteca.

Object.clone(objecte): fa una còpia exacta d'un objecte.

Object.extend(desti, source): amplia l'objecte de destinació amb les propietats (i mètodes d'origen).

isArray, isElement, isFunction, isHash, isNumber, isString, isUndefined

Retornen *true*, si són del tipus.

Object.keys(c): retorna una llista (***array***) de les claus d'un objecte.

Object.values(c): retorna una llista (***array***) dels valors d'un objecte.

toHTML, toJSON, toQueryString: converteix un objecte en HTML, JSON o format *query string* per a enviar-lo o desar-lo en format text.

Una de les mancances que té el JavaScript (tot i que quan es coneix en profunditat no ho és) és l'existència d'un sistema per a treballar amb classes (OOP) i herència. *Prototype* defineix un petit sistema amb:

```
var myParentClass = Class.create({
   parentFunction: function() { return "parent";}
});
var myClass = Class.create(myParentClass,{
   classFunction: function() { return "class";}});
var c = new myClass();
```

També hi ha funcions afegides a l'objecte ***String.blank()*** i ***String.empty()***, ***startsWith***, ***endsWith***, entre molts altres.

Web recomanat

En podeu consultar en línia una àmplia documentació:
http://api.prototypejs.org/

5.5. Components

A partir de la biblioteca *Prototype* i utilitzant-la com a base neix una extensió, http://script.aculo.us/, que hi afegeix tot un seguit de components (*widgets*) que ens permeten generar components ***drag&drop*** de manera simple o components per a ordenar, tota una galeria d'efectes i components com caselles d'autocompleció o un sistema d'edició de continguts en el document HTML mateix.

6. YUI

Web recomanat

Podem baixar la biblioteca YUI des de l'adreça següent:
http://developer.yahoo.com/yui/

Nascuda a l'equip de desenvolupament de Yahoo, la *Yahoo User Interface Library* és un conjunt de més de 50 MB de components per al desenvolupament d'aplicacions de client. Inclou arxius, *assets*, components i una potent biblioteca JavaScript que aquí analitzarem de manera breu.

La biblioteca YUI segurament és de les més completes i complexes, realment pot fer de tot i fins i tot inclou una biblioteca CSS.

Per a emprar-la podem utilitzar un sistema en línia que ens permet generar una biblioteca personalitzada per a cada ús a http://developer.yahoo.com/yui/3/configurator/

També hi ha l'opció, però, d'emprar la biblioteca baixant-ne els components directament des del **CDN** de Yahoo. Per a fer-ho hem d'incloure l'arxiu següent:

```
<script src="http://yui.yahooapis.com/3.2.0/build/yui/yui-min.js"></script>
```

A partir d'aquí, podem demanar i carregar per comanda els mòduls que necessitem, com per exemple:

```
YUI().use('node', 'anim','yui2-calendar', function(Y) {
   var YAHOO = Y.YUI2;
   Y.one('#test');
});
```

6.1. Treballar amb el DOM

Amb la YUI es treballa amb el DOM amb el mòdul *Node*. Aquest mòdul conté mètodes per a seleccionar nodes de l'HTML i interactuar-hi.

Mitjançant el mètode *one*, podem utilitzar un selector CS per a seleccionar un node, però només un, i amb aquest node obtindrem una referència al mateix estil que *jQuery*:

```
Y.one('#test').setStyle('display', 'none')
```

També podem seleccionar un conjunt o una llista utilitzant el mètode:

```
Y.all('.test')
```

A partir de tenir un node seleccionat hi podem demanar coses i assignar coses amb els mètodes ***get/set***. Per exemple, podem demanar l'ID fent ***Y.one('#test').get('id')***, o podem fer ***Y.one('#test').get('parentNode')*** perquè ens retorni un nou node que serà el pare. També podem configurar propietats com l'HTML d'un node fent ***Y.one('#test').set('html', 'blabbla')***.

Lectura recomanada

Tota la referència de la classe *node* és a:
http://developer.yahoo.com/yui/3/api/Node.html

6.2. Sistema d'esdeveniments

Per a afegir un esdeveniment escoltador (***Listener***) a una instància de node, s'ha d'emprar el mètode:

```
Y.one('#demo').on('click', function(e) {
   e.preventDefault();
alert('event: ' + e.type);
});
```

D'altra banda, també podem emprar la funció *delegate*, que ens capturarà esdeveniments que passin dintre del node fent:

```
Y.one('#demo').delegate('click', function(e) { });
```

Si *#demo* fos un *ul*, i dintre hi hagués elements *li*, la funció ens capturaria clics, dintre d'aquesta jerarquia.

Eliminem un ***listener*** amb el mètode *deatach*:

```
Y.one('#demo').deatach('click');
```

Podem desencadenar esdeveniments des de programa (simular-los) fent:

```
Y.one('#demo').simulate('click');
```

O podem llançar esdeveniments propis amb:

```
Y.one("#test").fire("events:elmeuevent");
```

```
Y.one("#test").on('touchstart', function() { });
```

Podem emprar esdeveniments tàctils, ja que hi són presents:

6.3. Eines i utilitats

Segurament la YUI és l'entorn de treball de desenvolupament més complet, i conté moltíssimes utilitats i mòduls. Repassarem com fer peticions amb Ajax i com construir petites animacions.

6.3.1. Peticions Ajax

Les peticions Ajax es fan emprant el mòdul:

```
Y.io('/url/de/lacrida', {
      method: 'POST',
      // Serialitzem el formulari.
      form: {
id: "elform", useDisabled: true },
      // Esdeveniments.
      on: {
complete: function (id, response) {
         // Handler de la resposta.
      }
   }
});
```

En aquest cas serialitzem el formulari *#elform*, fem una petició per **POST**, i quan obtinguem la resposta la processem a la funció del ***handler***.

Com veieu, pràcticament segueixen la mateixa estructura que la *Prototype* o la *jQuery*; juguem amb configuracions i paràmetres nous.

6.3.2. Animació

La YUI també té un mòdul d'animació.

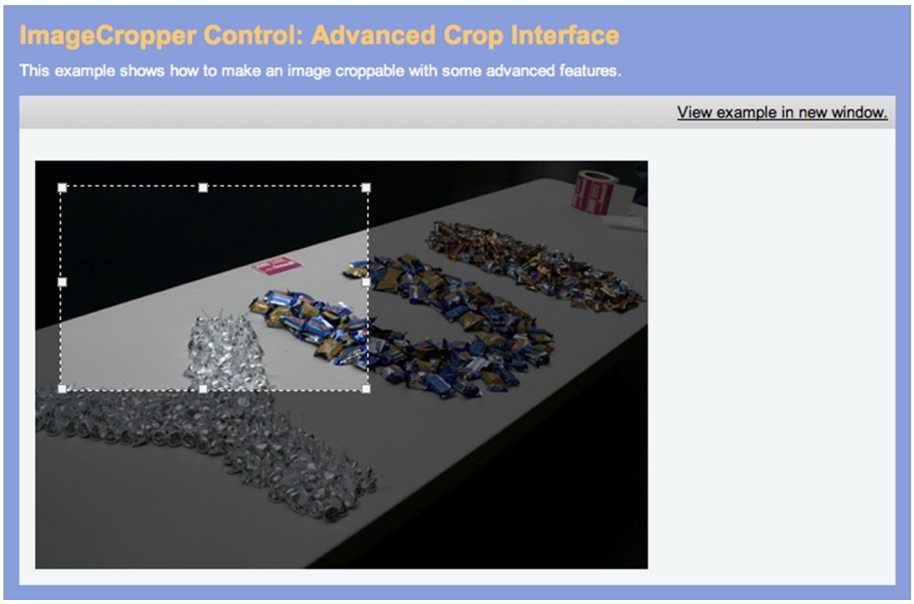
```
var animation = new Y.Anim({
node: '#my-div', // Selector a animar.
// Origen animació.
from: {
height: 0,
width: 0
},
// desti
to: {
height: 100,
width: 100
},
duration: 0.5, // durada
easing: Y.Easing.easeOut // Tipus d'interpolació de moviment.
});

animation.run();
```

Totes les propietats de l'animació es poden canviar utilitzant ***getters*** i ***setters***. També cal dir que les animacions desencadenen esdeveniments que es poden capturar.

6.4. Components

Realment la YUI destaca pel fet de ser una biblioteca completa amb multitud de components, entre els quals destaquen el camp d'autocompleció, l'editor de text enriquit, un sistema per a fer gràfics i una eina per a retallar imatges.



7. MooTools

Tal com esmenten al seu web, *MooTools* és una **biblioteca** JavaScript modular, compacta i orientada a objectes. A diferència de *jQuery*, de la qual molts diuen que implementa un DSL (***domain specific language***), la *MooTools* utilitza només JavaScript i el completa amb eines potents i robustes.

Moltes vegades la comparen amb la ***Prototype***, ja que totes dues basen la potència a augmentar els objectes natius per mitjà de la cadena de prototipatge. *MooTools*, però, és molt més compacta i modular.

MooTools significa '***my object oriented JavaScript tools***'.

Web recomanat

Podem baixar la biblioteca de l'adreça següent:

http://mootools.net/download

Podem fer una baixada personalitzada amb components assenyalats de la manera següent:

http://mootools.net/more/

7.1. Treballar amb el DOM

Té dues funcions bàsiques: ***$('id')***, que ens serveix per a seleccionar nodes per ID, i ***$$('css')***, que ens permet emprar selectors CSS.

Per exemple:

$$('E F')	Selecciona elements *F* que són fills de l'element *E*
$$('E[foo="bar"]')	Selecciona elements *E* que continguin un atribut *foo=="bar"*
$$('E:first-child')	Selecciona elements *E* que són el primer fill dels seus pares

La biblioteca ens ofereix eines per a caminar pels diferents nodes. Així, una vegada seleccionat un node, disposem dels mètodes ***getParent()*** i ***getParents()***: el primer ens retorna el pare, i el segon, una llista de tots els pares fins a l'arrel.

Al mateix temps, disposem dels mètodes ***el.getElement(selector)*** i ***el.getElements(selector)***, que ens permeten recuperar els fills del node que compleixin el selector. També disposem d'altres mètodes com ***el.getFirst(tag_name)*** o ***el.getLast(tag_name)***, que retornen el primer fill i l'últim que compleixen el *tag* definit com a paràmetre.

Per a manipular els estils CSS d'un node, disposem també de mètodes: ***el.hasClass('nom')***, consulta si el node té assignada la classe CSS; ***el.addClass('nom')***, òbviament afegirà la classe *nom*; o ***.removeClass('borders')***, que eliminarà la classe *borders*. Així:

```
$$('li.foo').addClass('bordered');
```

```
$$('li.foo').removeClass('bordered');
```

També ens ofereix el mètode ***toggleClass('class')***, per a posar o treure la classe en qüestió alternativament.

També podem manipular directament els estils associats amb un node emprant la funció ***$$('li.bar').setStyle('font-weight: bold')***; o de la mateixa manera ***setStyles***, que admet un objecte de paràmetres. També el ***getStyle*** i el ***getStyles***.

Disposem també d'un sistema per a treballar amb els atributs (o propietats) dels nodes. Així, ***.getProperty('type')*** o ***setProperty('type')*** ens mirarà o configurarà l'atribut tipus d'un camp de formulari. En aquest cas, també disposem de les versions múltiples, ***setProperties***, i d'una per a eliminar propietats, ***removeProperty()***.

Com la resta de biblioteques, des de *MooTools*, també podem crear o injectar elements en el DOM. Així:

```
var item1 = document.newElement('li',
{ class: 'bordered',
text: 'Second list item'
});
```

Generarà un element, que tindrem preparat per a injectar-lo. Per a fer-ho:

```
item2.inject(item1, 'inside');
```

També podrem fer servir aquestes altres fórmules equivalents a l'anterior:

.injectBefore(el),.
injectAfter(el),.injectTop(el),.injectBottom(el),.injectInside(el)

Podem clonar nodes amb ***$('id').clone()*** i els podem substituir:

```
var new_title = new Element('span', { text: 'New sublist' });
new_title.replaces(span1);
```

O hi podem fer un embolcall (*wrap*):

```
var span1 = $$('#div4>ul>li>span')[0];
var strong = new Element('strong');
strong.wraps(span1, 'top');
```

També podem eliminar nodes amb els mètodes ***.destroy()***, que elimina el node seleccionat, i ***.empty()***, que ens neteja tot l'arbre de fills.

7.2. Sistema d'esdeveniments

Les eines de *MooTools* per a gestionar els esdeveniments són bàsiques. Fonamentalment el que fan és apedaçar i solucionar les inconsistències entre els diferents navegadors i així capturar un esdeveniment d'un node:

```
var first_handler = function(ev) {
   ev.stopPropagation();
   ev.preventDefault();
   var el = ev.target;
   el.toggleClass('clickcolor');
};
$('link1').addEvent('click', first_handler);
```

Mitjançant ***stopPropagation*** trenquem la cadena de propagació i ***preventDefault*** cancel·la el comportament per defecte de l'esdeveniment. La propietat *Event.target* dóna accés a l'element involucrat en l'esdeveniment.

Podem afegir múltiples esdeveniments a un element o a una llista utilitzant el mètode ***addEvents*** i emprant com a paràmetre un objecte tipus, ***propietat/funció manipuladora***:

```
{
'click': function() {},
'rollover': function() {}
}
```

Per a eliminar un esdeveniment o una llista d'esdeveniments, podem emprar ***removeEvent('click', handler)*** o ***removeEvents('click')*** sense necessitar una referència al manipulador.

7.3. Ajax

MooTools té una classe *Request* que serveix per a fer peticions Ajax. Per a utilitzar-la:

```
var req1 = new Request({
   method: 'GET',
   url: 'data1.txt',
   onRequest: function() {
      $log("*** Firing request for " + this.options.url);
   },
   onSuccess: function(result_text, result_xml) {
      $log("*** Loaded " + this.options.url);
      $log("*** Data: " + result_text);
   },
   onFailure: function() {
```

```
        $log("*** Request failed: " + this.status);
    },
    onException: function() {
        $log("*** Exception occurred!");
        $log(arguments);
    })
req1.send();
```

Les dues primeres opcions de l'objecte *paràmetres* faciliten el mètode de fer la petició i l'URL que s'ha de demanar. La resta són *handlers* que s'han de desencadenar en els esdeveniments que es generaran. Recordem que el JavaScript és asíncron (el fil d'execució no espera).

Notem que, per a tots els manipuladors, la referència ***this*** apunta a l'objecte *Request*.

També podem fer una petició amb Ajax utilitzant el sistema d'esdeveniments:

```
var req2 = new Request();
req2.addEvent('success', function(txt, xml) {
    $log("*** Data loaded: " + txt);
})
req2.addEvent('success', function(txt, xml) {
    $log("*** Me too! Data loaded: " + txt);
});
req2.GET('data1.txt', { /* options */ })
```

Quan la petició es completi s'executaran les dues funcions.

7.4. Animació

Podem manipular la posició d'un objecte, un node amb propietats absolutes CSS, amb les funcions ***$('#exemple').position({ x: 150, y: 10 })***, i també podem consultar la posició actual amb ***el.getPosition()***, que retorna un objecte de $\{x,y\}$.

Aconseguirem fer animacions amb el mòdul ***Fx.Tween***:

```
var myFx = new Fx.Tween('myElement', {
duration: 'long',
transition: 'bounce:out',
property: 'height'
});
myFx.start('left', 0, 300)
```

Generem una nova instància d'animació, hi donem els paràmetres de durada i el mètode d'interpolació del moviment, i després utilitzem el mètode ***start*** per a dirigir l'animació des de la coordenada *x*:0 fins a la coordenada *x*:300.

També, però, hi ha un accés directe a la funció i que penja de la ***Classe Element***; recordem que la funció *$()* en retorna instàncies.

```
$('myElement').tween('width', '100').
   addEvent('onComplete', function() {
   alert('fi');
});
```

Les animacions també responen a esdeveniments; en aquest darrer exemple, fem que quan acabi l'animació executem un avís de sistema.

Una altra funcionalitat important és la possibilitat d'encadenar animacions amb el mètode ***chain()***:

```
$('myElement').tween('left', '100')
   .chain(function() {
      $('myElement').tween('top', '200')
   })
```

Primer movem l'objecte fins a la coordenada 100, després el reposicionem a la 200 vertical.

7.5. Components

De la mateixa manera que en la biblioteca *jQuery*, en el nucli de *MooTools* no hi ha component (*widgets*), però la seva natura compacta i modular i la possibilitat d'exhibir-los en la mateixa pàgina de la biblioteca ha fet que molts desenvolupadors programin components per a la biblioteca.

Web recomanat

En podem veure exemples aquí:
http://mootools.net/forge/

Personalment destaquem l'arbre d'arxius, un sistema de plantilles, un sistema per a retallar imatges i un altre per a fer-hi zoom, etc.

8. Biblioteques JavaScript amb l'objecte *canvas*

8.1. Introducció

8.1.1. Què és l'objecte *canvas*?

Com indica el nom, és l'equivalent d'una tela de pintor per a dibuixar (***canvas***). Així, emprant JavaScript, hi podrem pintar i dibuixar, crear objectes vectorials, afegir-hi imatges i textos, i capturar-hi esdeveniments.

> De fet, podem dir que amb l'objecte *canvas* tenim un potent punt de partida per a generar animacions i interactivitat.

Com a avantatges, podem esmentar que ja hi ha aplicacions que ens permeten pintar gràfics de dades, d'altres que fan dibuixos vectorials i fins i tot hi ha jocs interactius en dues i tres dimensions.

Quant als desavantatges, la família de l'**Internet Explorer** no és compatible amb aquest objecte fins a la revisió 9 (tot i que es pot emprar amb biblioteques que simulen l'API), i que ens queda una mica primari pel que fa a eines de programació per a la interacció. No disposa d'una **API** avançada per a treballar amb capes, objectes, esdeveniments o *sprites* com permeten altres tecnologies semblants com el **Silverlight** o l'**Adobe Flash**.

El més adient per a treballar amb aplicacions és construir o emprar una biblioteca que ens simplifiqui el procés de generació de codi. Segurament, la millor manera de pensar en l'objecte *canvas* és en un element que ens pot permetre substituir el Flash, en la versió primària, quan encara no era un **entorn de treball** (***framework***) **potent de desenvolupament d'aplicacions web**.

8.1.2. *Canvas* sense res

Per a explorar de manera còmoda els diferents beneficis d'emprar diferents biblioteques per a *canvas*, d'entrada, veurem com s'utilitza sense cap biblioteca, emprant directament l'**API** que ens ofereix el navegador. Així, començarem generant un nou objecte *canvas* amb l'etiqueta:

```
<canvas id="tutorial" width="150" height="150"></canvas>
```

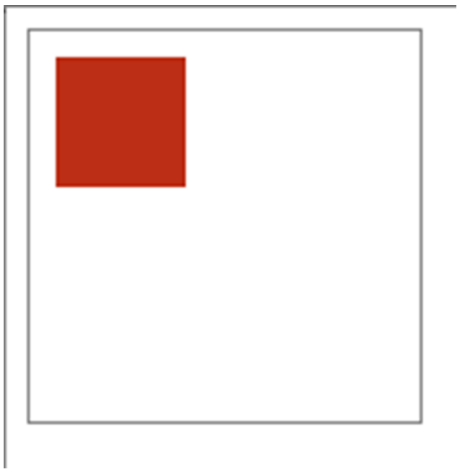
Amb aquesta etiqueta HTML, generarem un objecte de dibuix quadrat de 150 píxels d'amplària. Com tots els elements (objectes) HTML, el podem decorar amb estils. Hi podem definir un filet d'1 píxel de color gris de la manera següent:

```
<canvas id="tutorial" width="150" height="150" style="border:1px solid grey"></canvas>
```

Per a començar a dibuixar, primer hem d'obtenir una referència a l'objecte *canvas* amb ***document.getElementById('tutorial')*** per a obtenir després una referència al context de dibuix en 2D amb ***getContext('2d')***. També hi ha un mètode experimental que ens ofereix l'accés a dibuix en 3D (WebGL) mitjançant *getContext('3d')*.

```
<script>
window.onload = function() {
var canvas = document.getElementById('tutorial');
var ctx = canvas.getContext('2d');
ctx.fillStyle = "rgb(200,0,0)";
ctx.fillRect (10, 10, 50, 50);
}
</script>
```

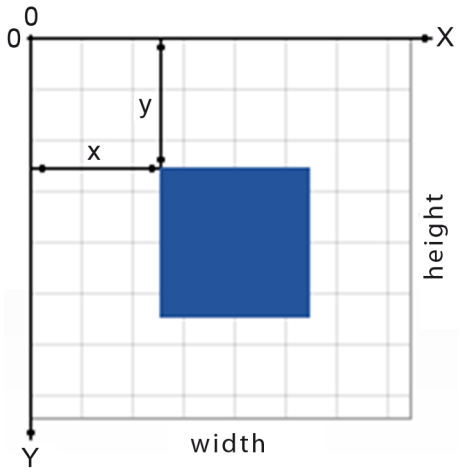
A partir d'aquí ja podem començar a emprar mètodes de l'objecte *canvas*. Amb l'ordre ***fillStyle*** definirem el tipus d'estil de contingut i amb l'ordre ***fillRect*** generarem un quadrat a les coordenades 10,10 i amb 50 píxels d'amplària i alçària. El resultat és el següent:



8.1.3. Dibuixar coses

1) Sistema de coordenades

La definició de coordenades en *canvas* funciona de dalt a baix i d'esquerra a dreta. Per tant, la coordenada 10,50 indica un punt a la columna 10 i un punt a la fila 50.



2) Quadrats

A diferència d'altres tecnologies, ***canvas*** només admet un tipus de forma primitiva: els quadrats. També disposem, però, d'un sistema potent per a generar línies i recorreguts que poden ser oberts o tancats.

Les instruccions per a generar quadrats són:

```
ctx.fillRect(x,y,width,height): dibuixa un rectangle ple de color.
ctx.strokeRect(x,y,width,height): dibuixa un rectangle a línia.
ctx.clearRect(x,y,width,height): neteja i torna transparent l'àrea especificada.
```

Les tres funcions reben els mateixos paràmetres (***x,y***) per a l'origen i (***width,height***) per a l'amplària i l'alçària de la caixa.

3) Línies i camins

La manera de dibuixar línies és diferent de la de dibuixar quadrats. De fet, funciona com si és tractés d'un llapis. Primer la posicionem i l'acostem al paper, i l'anem movent mentre volem que dibuixi.

```
ctx.beginPath(); // Inicia l'operació de dibuix.
ctx.moveTo(40, 40); // Col·loca el punter a 40, 40.
ctx.lineTo(340, 40); // Dibuixa una línia fins a 340, 40.
ctx.closePath(); // Tanca el camí.
ctx.stroke(); // El transforma en dibuix a línia.
```

4) Cercles

No hi ha un mètode específic per a dibuixar cercles amb ***canvas***, però sí que tenim les eines per a dibuixar-los. D'aquesta manera podem utilitzar el mètode Arc:

```
ctx.beginPath(); // Iniciem el dibuix d'un camí.
ctx.arc(100, 90, 50, 0, Math.PI*2, false); // Dibuixem un cercle.
ctx.closePath(); // Tanquem el camí.
ctx.fill(); // L'omplim de contingut.
```

El mètode Arc admet sis paràmetres: els dos primers són el punt central de la circumferència, el tercer el radi, el quart l'angle inicial, el cinquè l'angle final (**els angles en radiants**) i finalment un de booleà per a marcar si s'ha de dibuixar en la direcció de les agulles del rellotge o no.

Per a convertir graus en radiants, podem emprar la fórmula següent:

```
var degrees = 1; // 1 grau.
var radians = degrees * (Math.PI / 180); // 0,0175 radiants.
```

360 graus

360 graus són ***2*Math.PI***.

5) Estils de línia (*stroke*) i de farciment (*fill*)

Una altra de les eines fàcils és la possibilitat de personalitzar els estils de les línies i farciments que dibuixem de la manera següent:

```
ctx.fillStyle = "rgb(255, 0, 0)"; // Definirem un color de farciment vermell.
ctx.strokeStyle = "rgb(255, 0, 0)"; // Definirem un color de línia.
ctx.lineWidth = 20; // Definirem el gruix de la línia.
```

6) Text

Disposem d'un mètode que ens permet escriure coses al *canvas*:

```
var text = "Hello, World!";
ctx.font = "italic 20px serif";
ctx.fillText(text, 30, 80);
```

7) Imatges

Podem afegir imatges directament a un objecte *canvas* de la manera següent:

```
var img = new Image();
img.src = "pathalaimatge.gif"
ctx.drawImage(img,0,0);
```

Com que la imatge és un objecte remot, convé esperar a baixar-lo utilitzant *jQuery* i programant un petit *preload* de la manera següent:

```
var image = new Image();
image.src = "prova.jpg";
$(image).load(function() {
   ctx.drawImage(image, 0, 0);
});
```

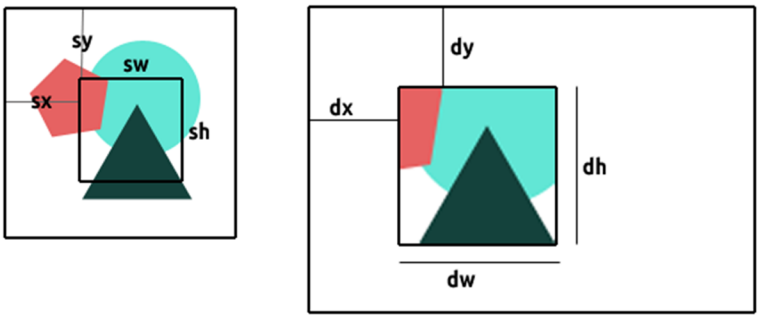
Primer creem un objecte tradicional amb JavaScript per emprar-lo com a receptor de la baixada. Mitjançant l'*image.src*, n'inicialitzem la baixada. Finalment, amb la funció *$(image).load*, programem l'esdeveniment (*onLoad* a la imatge), que ens la dibuixarà (col·locarà) a l'objecte *canvas*. Podem **escalar les imatges** directament quan les dibuixem sobre el ***canvas*** emprant la funció ***drawImage()*** de la manera següent:

```
ctx.drawImage(image, x, y, width, height);
```

En què ***width*** i ***height*** són l'amplària i l'alçària respectives que volem. També podem **retallar imatges** directament en inserir-les al ***canvas*** amb el següent:

```
ctx.drawImage(image, sx, sy, sw, sh, dx, dy, dw, dh);
```

En què ***sx***, ***sy***, ***sw*** i ***sh*** marquen el reenquadrament de la imatge original; ***dx*** i ***dy***, la posició del reenquadrament al *canvas*, i ***dw*** i ***dh***, la grandària.



Sobre les imatges hi podem aplicar transformacions pel que fa a la rotació, l'escala i l'origen. En l'exemple següent:

```
ctx.translate(20, 20);
ctx.rotate(0.7854); // Rotate 45 degrees
var image = new Image();
image.src = "example.jpg";
$(image).load(function() {
   ctx.drawImage(image, 0, 0, 50, 50, -10, -10, 30, 30);
});
```

Traslladarem l'eix de rotació a **250,250** i després aplicarem una rotació en radiants de **0,7854**. A partir d'aquí, s'aplicaran aquests canvis als objectes que dibuixem abans de dibuixar-los.

Una altra característica molt interessant de l'objecte *canvas* i les imatges és la possibilitat d'accedir directament als píxels que configuren una imatge; tenim accés a una matriu dels colors que formen cada píxel i els podem modificar. Amb això, s'obre la porta a la possibilitat de confeccionar amb JavaScript multitud d'efectes sobre imatges directament des del codi. Per tenir accés als píxels, cridarem a la funció:

```
a = ctx.getImageData(x, y, width, height);
```

Ens retornarà un objecte amb tres propietats, el ***width***, el ***height*** i un ***data*** llista de píxels del tipus ***CanvasPixelArray***, una llista en què cada quatre posicions representen un color en la forma RGBA.

```
a.data[0]; // component red
a.data[1]; // component green
a.data[2]; // component blue
a.data[3]; // component alpha
```

El component ***a[5] a a[8]*** serà el píxel següent. En aquesta *array* no hi ha files ni columnes, i les haurem de deduir nosaltres a partir del ***width*** d'imatge.

Podem confeccionar el típic efecte de soroll d'una manera senzilla, generant una imatge des de 0 que escriurem a l'objecte *canvas* amb el següent:

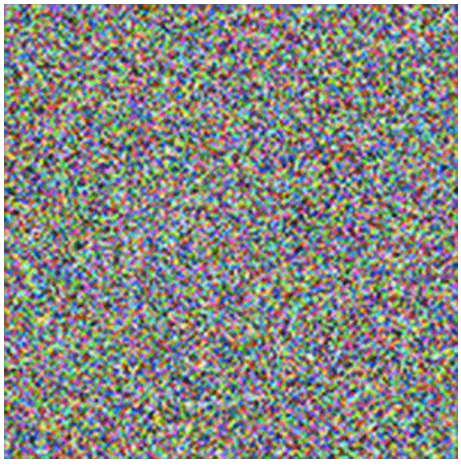
```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<script type="text/javascript">



window.onload = function()
{
   var canvas = document.getElementById('exercici');
   var pt = canvas.getContext('2d');
   setInterval(function() {
      // Generem una array de píxels aleatoris.
      var image_data = dibuixa_soroll( pt );
      // La pintem a l'objecte canvas.
      pt.putImageData( image_data, 0, 0);
   }, 200);
}
```

```
dibuixa_soroll = function(pt)
{
   var imageData = pt.createImageData(150, 150);
   var pixels = imageData.data;
   var numPixels = imageData.width*imageData.height;
   // Per cada píxel generem un color aleatori.
   for (var i = 0; i < numPixels; i++) {
      pixels[i*4] = Math.floor(Math.random()*255); // Red
      pixels[i*4+1] = Math.floor(Math.random()*255); // Green
      pixels[i*4+2] = Math.floor(Math.random()*255); // Blue
      pixels[i*4+3] = 255; // Alpha
   };
   return imageData
}
</script>
</head>
<body>
   <canvas width="150" height="150" style="border:1px solid grey" id="exercici"> </canvas>
</body>
</html>
```

El resultat és una animació com de televisió dessintonitzada:



8.1.4. Transformacions

1) Transformacions i escales

Podem transformar la manera com dibuixem un objecte, o certs objectes, amb els mètodes ***translate*** i ***rotate***: amb el primer, transportem les coordenades 0,0 a un altre punt. Així, si fem:

```
ctx.translate(100,100);
```

La coordenada 0,0 passarà a ser la 100,100, i si nosaltres dibuixem un quadrat a la 0,0,10,10, ens apareixerà a la 100,100,110,110.

Passa el mateix amb la rotació. Si nosaltres fem:

```
ctx.rotate(Math.PI/4)
```

Aplicarem una rotació de 45° en el *canvas* i tot el que hi dibuixem ens apareixerà amb la mateixa rotació.

2) Composició (*compositing*)

Compondre elements es refereix a la manera com es combinaran els nous elements que afegim al *canvas* amb els que ja hi ha. Així, quan dibuixem un objecte al *canvas*, podem decidir com volem que se superposi sobre la resta d'objectes existents.

Quan afegim un nou objecte, podem decidir si volem que se superposi o se sotaposi. I també si volem que en superposar-se s'apliqui un canal ***alpha*** o volem que faci un ***xor*** de píxels. Podem definir el model de composició de la manera següent:

```
ctx.globalCompositeOperation = "";
```

En què el valor pot ser:

source-over	La imatge origen se situarà sobre el *canvas*
destination-over	La imatge se situarà a sota dels objectes existents al *canvas*
source-in	L'origen es dibuixarà a la zona on hi ha superposició
destination-in	La destinació es dibuixarà a la zona on hi ha superposició
lighter, copy, xor	*Lighter* genera la superposició com a color 255, blanc *Copy* dibuixa l'origen en comptes de la destinació Qualsevol part que se superposi quedarà transparent

3) Ombres

També podem definir i pintar ombres en un objecte mentre el definim. Per a fer-ho, cal que definim les propietats adients perquè apareguin correctament quan generem l'objecte:

```
ctx.shadowBlur = 20;
ctx.shadowColor = "rgb(0, 0, 0)";
```

```
ctx.shadowOffsetX = 10;
ctx.shadowOffsetY = 10;
ctx.fillRect(50, 50, 100, 100);
```

En aquest cas, generem una ombra de color negre amb un desenfocament de 20 píxels i desplaçament de 10 píxels.

4) Guardar l'estat actual com a imatge

El ***canvas***, a efectes pràctics, és com si tinguéssim una gran imatge, i com a tal, la podem guardar com a imatge amb l'ordre següent:

```
var dataURL = canvas.get(0).toDataURL();
```

Ens generarà una imatge en format PNG i codificada en base 64. Al mateix temps, aquest *string*, que podem enviar a un servidor i descodificar com a imatge, el podem assignar a un objecte imatge i obrir-lo directament amb el navegador:

```
var img = $("<img></img>"); img.attr("src", dataURL);
canvas.replaceWith(img);
```

Si l'usuari es vol guardar la imatge, simplement ha de fer clic amb el botó dret del ratolí, ja que hem convertit l'objecte *canvas* (vectorial) en una imatge.

També podem generar un arxiu per a baixar-lo enviant primer la imatge al servidor, convertint-la en binari allà i llançant les capçaleres adients juntament amb els bytes de la imatge.

8.1.5. Animació

L'objecte *canvas* no ens proporciona un mecanisme per a generar animacions de manera directa, però sí que ho podem fer gràcies al JavaScript. El principi amb pseudocodi és el següent:

1) Dibuixem.

2) Esperem un interval de temps.

3) Esborrem la pantalla.

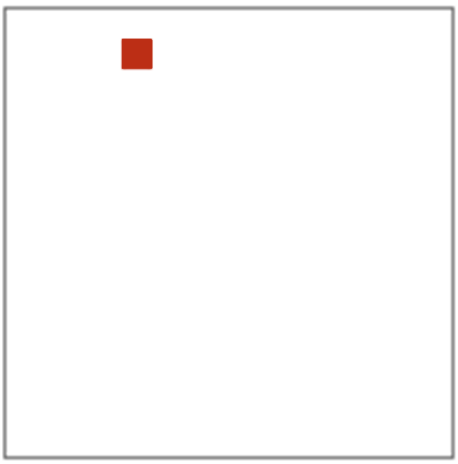
4) Actualitzem els valors.

Tornem al pas 1.

D'aquesta manera podem fer un petit exemple que demostri la manera com podem animar coses:

```
<html>
<head>
   <title>Exercici Moviment canvas</title>
   <script type="text/javascript">
   window.onload = function()
   {
      var canvas = document.getElementById('exercici');
      var pt = canvas.getContext('2d');
      pt.fillStyle = "rgb(200,0,0)"; // Format del dibuix.
      var inc = 1; // Velocitat del moviment.
      // Objecte que movem ...
      var o = { x: 10, y:10, width:10, height:10 };
      // Execució periòdica.
      setInterval(function() {
         // Si l'objecte sobrepassa els límits del canvas,
         // en canviem l'orientació.
         if(o.x >= 140 || o.x <=0)
            inc = -inc
         o.x += inc
         neteja(pt)
         dibuixa( o, pt );
      }, 1);
   }
   dibuixa = function(obj, pt)
   {
      // Dibuixa l'objecte.
      pt.fillRect (obj.x, obj.y, obj.width, obj.height);
   }
   neteja = function(pt)
   {
      // Neteja la pantalla i la prepara per al pas següent.
      pt.clearRect(0,0,150,150)
   }
   </script>
</head>
<body>
   <canvas width="150" height="150" style="border:1px solid grey" id="exercici"> </canvas>
</body>
</html>
```

El resultat és:



Si ens fixem en el codi, veurem que generem una execució periòdica amb ***setInterval***, que anirà cridant al nostre programa. Primer mourem l'objecte, després netejarem la pantalla i finalment el tornarem a dibuixar.

8.2. *Raphael.js*

És una de les primeres biblioteques gràfiques de vectors per al Web, independent del ***canvas***, que també funciona a l'Internet Explorer 6. És una bona solució per a crear petits components interactius. És una barreja de les funcionalitats del *canvas*, amb el *svg*, però l'"esperit" del document és molt interessant.

Web recomanat

Podem consultar-ne el web a: http://raphaeljs.com

8.2.1. Inicialització

Per a utilitzar la biblioteca convé baixar-la del seu web i després incloure-la a l'HTML. També la podem emprar per a fer proves utilitzant el *jsfiddle.com*.

Una vegada inclosa a l'HTML, cal que inicialitzem un nou objecte de tipus *Raphael* amb el codi següent:

```
<html>
<head>
<title>Raphael Play</title>
<script type="text/javascript" src="raphael.js"></script>
<script>
window.onload = function() {
   element = document.getElementById('container')
   var paper = new Raphael(element, 500, 500);
}
   </script>
</head>
<body>
   <div id="container" style="border:1px solid grey"> </div>
</body>
```

```
</html>
```

Si ens hi fixem, l'objecte *Raphael* rep un element del DOM on inserirem l'element *canvas*. En aquest cas ho fem amb el JavaScript estàndard emprant la funció ***document.getElementById(id)***, però també es pot fer amb qualsevol biblioteca del tipus ***jQuery***, ***Prototype*** o d'altres.

També podem inserir directament l'element al DOM emprant, en lloc d'una referència, un element del **DOM** amb les coordenades on el volem:

```
var paper = new Raphael(0,0, 500, 500);
```

8.2.2. Dibuix i formes

Disposem d'una API senzilla i completa per a generar formes i dibuixar camins (***paths***). Així, amb:

```
paper.circle(x,y,radi);
```

Dibuixarem un cercle a les coordenades ***x,y*** del corresponent ***radi***.

```
paper.rect(x,y,width, height, angle)
```

Dibuixarem un rectangle en què ***x,y*** són les coordenades inicials, ***width*** és l'amplària, ***height*** és l'alçària i el darrer paràmetre, que és opcional, és l'angle de les cantonades.

```
var c = paper.ellipse(x,y, rh, rv)
```

Ens permetrà generar el·lipsis en què *x,y* és el punt inicial, *rv* el radi vertical i *rh* el radi horitzontal.

Un aspecte important és que les diferents ordres que generem retornen punters a l'objecte creat. Així, en la darrera ordre, podem accedir a l'objecte *el·lipsi* i generar-hi un color de cos amb:

```
c.attr({'fill':'red'});
```

Mitjançant els atributs, podem assignar i canviar multitud de paràmetres a una llista d'objectes.

fill	Un color o un gradient per a fer el farciment linear gradient: "‹angle›-‹colour›[-‹colour›[:‹offset›]]*-‹colour›", exemple: "90-#fff-#000" – 90° degradat de blanc a negre o "0-#fff-#f00:20-#000" – 0° degradat de blanc a vermell (i 20%) a negre degradat radial: "r[(‹fx›, ‹fy›)]‹colour›[-‹colour›[:‹offset›]]*-‹colour›", exemple: "r#fff-#000" – degradat de blanc a negre "r(0.25, 0.75)#fff-#000" – degradat de blanc a negre amb un punt de focus a 0,25, 0,75 Les coordenades de punt de focus són amb rang de 0...1 Els degradats radials només es poden aplicar a cercles i el·lipsis
fill-opacity	De 0 a 1. Percentatge d'opacitat de l'objecte
font ***font-family*** ***font-size*** ***font-weight***	Nom del tipus de lletra assignat Família del tipus de lletra Grandària del tipus de lletra Tipus de lletra gruixuda ***bold***?
height ***width***	Grandària horitzontal i vertical
href	Si s'especifica, enllaç de l'objecte
opacity	De 0 a 1. Percentatge d'opacitat de l'objecte
path	*pathString* SVG path string format
r	Radi de l'objecte
rotation	Rotació de l'objecte
scale	Escala de l'objecte
src	En el cas d'imatges, font de la imatge
stroke ***stroke-opacity stroke-width***	Color del filet Opacitat del filet Amplària del filet
x,y	Coordenades *x,y* de l'objecte

Web recomanat

Podem visualitzar tota la referència de propietats i mètodes de la biblioteca *Raphael* al seu web:

http://raphaeljs.com/reference.html

Donada una referència a un objecte, també el podem manipular directament amb determinats mètodes disponibles: així, el podem amagar amb ***c.hide()*** o mostrar amb ***c.show()***. El podem fer rodar o escalar, amb ***c.rotate(angle)*** o ***c.scale(sx,sy)*** respectivament, en què *sx* i *sy* són percentatges de 0 a 2 d'escalat de la vertical i l'horitzontal.

També podem moure l'objecte a una altra posició, del *canvas*, amb el mètode ***c.translate(x,y)***.

Una altra ordre important és ***set***, que ens permet generar conjunts d'objectes, agrupacions que podrem manipular de manera independent. Així:

```
var st = paper.set();
st.push(
```

```
paper.circle(10, 10, 5),
paper.circle(30, 10, 5)
);
st.attr({fill: "red"});
```

Crearem dos cercles que es podran manipular des de la mateixa referència *st*.

També podem eliminar elements creats amb l'ordre ***st.remove()***.

Els elements tenen profunditat i tenim dues ordres per a manipular-ne la posició. Si cridem al mètode ***.toFront()*** l'element es posicionarà a la part superior de la pila d'elements, mentre que si cridem al mètode ***.toBack()*** l'element es posicionarà al fons. També podem generar els nous elements en una posició emprant els mètodes ***insertBefore()*** i ***insertAfter()***.

També podem inserir textos amb l'ordre ***.text(x,y,'cadena de text')***. Amb la tecnologia Cufon (https://github.com/sorccu/cufon/wiki/about) podrem emprar fonts, que ens generarà la font en format de conjunt de camins (***paths***) en ***json***.

Hem de parlar també de la potència de l'ordre ***.path()*** per a generar recorreguts. El format de l'ordre és el del *svg* (http://www.w3.org/TR/SVG/paths.html#PathData).

```
paper.path('M10 10L90 90');
```

Generarà una línia diagonal des de la coordenada 10,10 fins a la coordenada 90,90.

I per acabar la part de dibuix de formes amb *Raphael.js*, podem incloure imatges dintre d'un objecte *canvas* amb l'ordre:

```
paper.image("apple.png", 10, 10, 80, 80);
```

En què les dues primeres indiquen les coordenades del punt d'inserció i les segones són l'amplària i l'alçària.

8.2.3. Animació

Un altre dels factors interessants per a emprar *Raphael* és la capacitat per a generar animacions complexes amb una sola ordre. Disposem del mètode ***animate***, que ens permet generar animacions de manera fàcil i còmoda. Així:

```
var d = paper.circle(400, 40, 20);
d.animate({cy: 480, r: 20}, 2000, "bounce");
```

Generarem un cercle de 10 píxels d'amplària a la posició ***10,10***. L'animació que generarem a continuació animarà les propietats a un radi de 20 píxels i la posició del centre *x* també a 20 píxels durant 2.000 mil·lisegons amb una equació de moviment ***bounce*** rebot. Així, la forma general serà:

```
c.animate({objecte_propietats}, durada, equació de moviment, onEnd);
```

En què l'equació pot ser:

```
[">;", "<", "<>", "backIn", "backOut", "bounce", "elastic", "cubic-bezier(p1, p2, p3,
p4)"]
```

Les propietats que podem animar són ***clip-rect***, ***cx***, ***cy***, ***fill***, ***fill-opacity***, ***font-size***, ***height***, ***opacity***, ***path***, ***r***, ***rotation***, ***rx***, ***ry***, ***scale***, ***stroke***, ***stroke-opacity***, ***stroke-width***, ***translation***, ***width***, ***x*** i ***y***.

També podem definir un darrer paràmetre que pot ser una funció que volem que s'executi al final de l'animació.

Finalment, podem fer animacions amb fotogrames clau (***keyframes***), animacions on marquem passos per a moments de temps; per exemple:

```
c.animate({
"20%": {cx: 20, r: 20, easing: ">"},
"50%": {cx: 70, r: 120, callback: function () {...}},
"100%": {cx: 10, r: 10}}, 2000);
```

L'animació anterior mourà l'objecte fins a un radi de 20, durant el 20% del temps, o sigui 400 mil·lisegons, mentre que durant 600 el farà créixer a 120, executant el *callback* en arribar. Finalment, els darrers 1.000 mil·lisegons, evolucionarà cap al punt d'inici.

8.3. EaselJS

Una biblioteca per a *canvas* relativament nova, però amb molt bona perspectiva i una base tècnica sòlida al darrere, és EaselJS. Va ser creada per gskinner.com, com a ajuda al desenvolupament del joc en HTML5 *Pirates Love Daisies*.

La idea fonamental de la biblioteca és afegir al component ***canvas*** mètodes per a interactuar amb objectes sobre aquest, a l'estil del Flash i l'Actionscript. Així, el que han començat a fer és reescriure l'estructura fonamental del **DOM del Flash i *actionscript*** emprant com a sistema de renderització del ***canvas*** **HTML5**.

No és el nostre objectiu exposar com funciona l'estructura d'objectes d'*actionscript* 3, ni la potència que té per a generar estructures visuals complexes, jocs i d'altres. Per tant, el que farem en aquest mòdul és enumerar les principals funcionalitats que ha aportat l'*easelJS*.

8.3.1. Classes clau

- ***DisplayObject***

Classe abstracta per a tots els elements visuals o inseribles al *canvas*. Proporciona accés a totes les propietats (*x,y*, *rotation*, *scaleX*, *scaleY*, *skewX*, *skewY*, *alpha*, *shadow*, etc.) comunes a tots els objectes visuals.

- ***Stage[[[***

És el nivell inicial i contenidor de tots els elements *display*. Cada vegada que es crida a la funció *update* a l'*stage*, s'encarrega de renderitzar tots els elements enganxats al *canvas*.

- ***Container***

És una classe contenidora que permet treballar amb diferents objectes dintre d'un i manipular-los com un sol text.

- ***Text***

Dibuixa text en el context d'un objecte *display*.

- ***Bitmap***

Dibuixa una imatge o vídeo al *canvas*, i disposa de les propietats heretades de *DisplayObject*.

- ***BitmapSequence***

Dibuixa una seqüència d'imatges (***sprites***) diferents de fotogrames en una reixeta i disposa d'una petita API per a manipular la reproducció de la seqüència.

- ***Graphics***

Una API potent per a renderitzar objectes gràfics al *canvas*.

- ***Shape***

Renderitza gràfics vectorials, com a objecte *display list*, amb les propietats de l'anterior.

8.4. *CanvaScript*

A l'estil de *jQuery*, *CanvaScript* ens ofereix un mètode que ens serveix d'eina per a inicialitzar la biblioteca i també com a mètode per a dibuixar les operacions al *canvas*. Per a començar a fer coses, podem emprar l'exemple següent:

Web recomanat

http://jcscript.com/

```
<html>
<head>
   <title>Exercici jcanvascript</title>
   <script type="text/javascript" src="jCanvaScript.js" > </script>
   <script type="text/javascript">
   window.onload = function()
   {
      jc.start('ejercicio');
      jc.circle(50,50,50,'rgba(255,255,0.1)',1);
      jc.start('exercici');
   }
   </script>
</head>
<body>
   <canvas width="150" height="150" style="border:1px solid grey" id="exercici"> </canvas>
</body>
</html>
```

Hi incloem la biblioteca i la inicialitzem amb la funció ***jc.start('id_delcanvas')***. A partir d'aquí, dibuixem un cercle amb el mètode ***jc.circle***. Finalment, tornem a cridar al mètode *start* per pintar les manipulacions fetes.

Una de les característiques importants de la biblioteca és que ens permet treballar de manera encadenada. Així, un cop generem una primitiva (element), hi podem assignar esdeveniments, canviar-ne propietats o donar-hi un ID de manera fluida (sense haver de tornar a seleccionar l'objecte) i de manera encadenada. Trobem construccions de codi com aquesta:

```
   jc.start('exercici', 25);
   jc.circle(50,50,20,'rgba(255,255,0.1)',1)
      .draggable()
      .click(function(){
            alert('hola')
      });
```

Seleccionem un objecte *canvas*, hi dibuixem un cercle, que tornem arrossegable i hi programem un *event clic* que generarà una notificació.

La biblioteca *canvaScript* ens ofereix una sèrie de primitives que ens permeten dibuixar multitud de formes. Bàsicament, augmenta i millora l'**API** per defecte de l'objecte ***canvas***. Així, podem fer el següent:

```
jc.circle(x, y, radius, [color], [farciment])
```

Dibuixa un cercle.

```
jc.rect(x, y, width, height, [color], [farciment])
```

Dibuixa un quadrat.

```
jc.arc(x, y, radius, iAngle, fAngle, [agullesrellotge], [color], [farciment])
```

Dibuixa un arc des d'*iAngle* fins a *fAngle* en la direcció de les agulles del rellotge (*true*) o al revés (*false*). El color del filet i del farciment són paràmetres opcionals.

```
jc.line(points, [color], [farciment])
```

Dibuixa una línia. El primer paràmetre és una llista de parells de punts en la forma [[0,0], [0,10]].

```
jc.qCurve(points, [color], [farciment])
```

Dibuixa una corba amb equació quadràtica.

```
jc.bCurve(points, [color], [fill])
```

Dibuixa una corba de Bézier.

```
jc.imageData(float width, float height)
```

Crea una nova imatge, d'amplària i alçària.

```
jc.image(img, sX, sY, sWidth, sHeight, dX, dY, dWidth, dHeight)
```

És un clon de la funció *image* nativa de *canvas*.

```
jc.text(text, x, y, [maxWidth], [color], [fill])
```

Clon de la funció *text* de *canvas*, amb el color i el farciment extres.

```
lGradient(x1, y1, x2, y2, colors)
```

Dibuixa un degradat de lineal, des de *x1,y1* fins a *x2,y2*, en què els colors són una llista de percentatges de cada color.

```
[ [0,'rgb(0,0,0)'],
[0.5,'rgb(255,0,0)'],
[0.75,'rgb(255,255,0)'] ];
```

Cada element de la llista és una petita llista amb el percentatge i el color.

rGradient(x1, y1, radius1, x2, y2, radius2, colors)

Dibuixa un degradat de forma radial.

```
jc.start(idCanvas);
var colors=[[0,'rgb(0,0,0)'],
[0.5,'rgb(255,0,0)'],
[0.75,'rgb(255,255,0)']];
var gradient=jc.rGradient(10,30,20,10,30,250,colors);
jc.rect(1,1,248,263,gradient,1)
jc.start(idCanvas);
```

jc.layer(idLayer)

L'ordre *layer* genera un conjunt d'objectes que es poden tractar com a unitat. Per exemple, el codi següent:

```
jc.layer('grup1')
.animate({translate:{x:100}},2000)
.draggable()
.clone('2')
.animate({translate:{x:100,y:140}},1500);
```

Emprem el conjunt ***grup1*** i en fem una animació, el tornem arrossegable i finalment el clonem.

```
jc.addObject(name, parameters, fn)
```

Permet generar noves primitives a partir d'un nom, una llista de paràmetres i una funció de dibuix.

Vistos aquests objectes (en el llenguatge de la seva documentació, *primitives*), veurem tot un seguit de mètodes disponibles per a cada objecte, un cop creat o si el seleccionem. Una vegada hem dibuixat un objecte, hi podem assignar un ID, amb la funció ***id()***, que després podrem emprar per a seleccionar l'objecte en qüestió.

```
.animate(parameters, [duration], [easing], [onstep], [fn])
```

Una vegada hem seleccionat un objecte, el podem associar i generar-hi una animació amb el codi següent:

```
jc.circle(50,50,20,'rgba(255,255,0,1)',1)
.animate({x:175,radius:5,color:'#000000'},1000);
```

Generarem una animació del cercle canviant-ne la coordenada *x*, el radi i el color. La durada és en mil·lisegons i podem assignar una funció de moviment i esdeveniments en finalitzar i a cada pas.

```
.opacity([float])
```

Canviem l'opacitat d'un objecte. També ens permet consultar l'existent.

```
.shadow({ x:5, y:5, blur:15, color:'#ff0000' })
```

Permet assignar una ombra.

```
.visible()
```

Canvia la visibilitat d'un objecte.

```
.id()
```

Assigna o consulta l'ID d'un element.

```
.name()
```

Assigna un nom, que després podrem utilitzar a l'estil dels selectors de classe de CSS.

```
.down() / .up() / .level()
```

Manipula la profunditat dels objectes: ***down()*** el fa baixar, ***up()*** el fa pujar i ***level()*** ens retorna la profunditat, la qual sempre es calcula relativa al total d'objectes generats en un espai.

```
.rotate() / .scale() / .transform()/ .translate() / .translateTo()
```

Ens permet alterar i canviar propietats de l'objecte seleccionat.

```
.attr()
```

Permet canviar atributs d'un objecte generat, com per exemple:

```
.attr('fill',0);
```

```
.buffer()
```

Retorna una referència a l'objecte en qüestió. Amb aquesta referència el podem escriure com a imatge.

```
.clip(objecte)
```

Ens permet generar un objecte que n'emmascararà un altre.

```
.clone()
```

Clona l'objecte seleccionat i en genera una còpia.

```
.del()
```

Elimina l'objecte seleccionat.

```
isPointIn(x,y)
```

És l'objecte sobre el punt *x,y*?

```
color()
```

Ens permet canviar el color del farciment d'un objecte.

```
lineStyle()
```

Ens permet canviar l'estil de línia d'un objecte.

Events

També podem assignar esdeveniments a l'estil *jQuery* a qualsevol objecte, fent la crida al nom de l'esdeveniment.

```
click()
dblclick()
mousedown()
mousemove()
mouseout()
mouseover()
mouseup()
blur()
focus()
keydown()
keypress()
```

```
keyup()
```

En la forma:

```
objecte.click(function(){ })
// Dintre la funció, el <emphasis>this</emphasis> es refereix a l'objecte que ha generat l'esdeveniment.
draggable() / droppable()
```

Ens permet transformar un objecte en arrossegable i destí de ser arrossegat.

Mentre la primera actua com a activador de l'arrossegable, la segona actua com a assignador d'esdeveniment. Per exemple:

```
jc.id('#ele').droppable( function() {
// Desencadena l'esdeveniment pertinent.
})
```

8.5. Processing

El Processing neix com un **port** del llenguatge de creació artística per a Java al JavaScript, per part del mateix creador que ***jQuery***, **John Resig** (**www.ejohn.com**).

En aquest manual només el mencionarem, ja que la base del sistema és l'objecte *canvas*, però no el tractarem, perquè el Processing és un llenguatge en si mateix. Així, al JavaScript el port esdevé un intèrpret del llenguatge Processing i estableix un sistema per a processar (*parse*) els *scripts*.

Webs recomanats

Podem trobar-ne exemples i veure com funciona als webs del llenguatge:
http://www.processingjs.com
http://www.processing.com

8.6. Exercici

Rellotge amb *canvas*.

```
<html>
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
   <title>Rellotge JavaScript</title>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.5.1/jquery.min.js"> </script>
<script src="raphael-min.js"> </script>
<script type="text/javascript">
<!--
$(document).ready(function(){
   function Rellotge()
   {
      $(document).trigger('tiempo', [new Date()]);
      setInterval(function(){
         $(document).trigger('tiempo', [new Date()]);
      }, 1000);
```

```
}
var TextVisor = function() {
   var t = this.t = this;
   var tick = function(event, extra) {
      t.pinta( event, extra );
   }
   this.pinta = function(event, extra) {
      $( '#rellotge' ).html( t.format_string(extra) );
   }
   this.format_string = function(dat) {
      d = dat;
      hora = d.getHours();
      minuto =d.getMinutes();
      if(minuto<=9)
         minuto = "0"+minuto;
      segundo = d.getSeconds();
      if(segundo<=9)
         segundo = "0"+segundo;
      return hora + ":" + minuto + ":" + segundo;
   }
   $(document).bind('tiempo', tick);
}

var RaphaelVisor = function(object_id)
{
   var t = this.t = this
   var pt = this.pt = new Raphael(document.getElementById(object_id), 200, 200);
   // Dibuixem el rellotge inicial.
   var PX = 100;
   var PY = 100;
   var tick = function(event, d)
   {
      h = d.getHours();
      if(h>12)
         h = h/2
      m =d.getMinutes();
      // Afegim a l'hora la part proporcional de minuts.
      h += (m*100/60)/100
      // Transformem l'hora a angles.
      s = d.getSeconds();
      ah = (h*360)/12
      am = (m*360)/60
      as = (s*360)/60
      neteja()
      //console.info(h,m,s,">>> ", ah, am, as)
      t.hora = pinta_agulla( 50, ah)
      t.minut = pinta_agulla( 60, am)
```

```
            t.segon = pinta_agulla( 70, as)
            t.hora.attr( {'stroke-width': 7, 'stroke': '#ac360b' })
            t.minut.attr( {'stroke-width': 5, 'stroke': '#a0573c' })
            t.segon.attr( {'stroke-width': 3, 'stroke': '#666666' })
        }
        var neteja = function()
        {
            if(t.hora)
                t.hora.remove();
            if(t.minut)
                t.minut.remove();
            if(t.segon)
                t.segon.remove();
        }
        var pinta_agulla = function ( radi, an )
        {
            // Convertim l'angle en radiants.
            angle = (an-90)*(2*Math.PI/360);
            // calculem coordenades x, y
            x = PX + Math.cos(angle)*radi
            y = PY + Math.sin(angle)*radi
            svg = "M"+ PX + " " + PY + " L" + x + " " + y
            return pt.path(svg)

        }
        this.init=function() {
            fons = pt.circle(100,100, 90)
            fons.attr({'fill': '#fbf8e1'})

            $(document).bind('tiempo', tick);
        }
    }
    // Inicialitzem el rellotge.
    Rellotge();
    // Aquest tercer visor connecta el rellotge amb la finestra del navegador.
    var vis3 = new TextVisor();
    vis3.pinta = function(ev, extra){
        $(document).attr('title',
            this.format_string(extra) );
    }
    // Inicialitzem el rellotge fet amb la biblioteca Raphael.
    vi = new RaphaelVisor('rellotge')
    vi.init()
});
//-->
</script>
</head>
```

```
<body>
   <div id="rellotge" style="width:200px; border:1px solid grey"> </div>
</body>
</html>
```

9. Exercicis

Construirem un rellotge de manera modular. D'entrada, generarem un petit nucli que desencadenarà un esdeveniment que ens informi del pas del temps i al qual afegirem tres components diferents que generaran visualitzacions de l'hora actual:

```
<html>
<head>
   <title>Rellotge JavaScript</title>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.5.1/jquery.min.js"> </script>
<script type="text/javascript">
<!--
// Emprem l'esdeveniment de DOM disponible.
$(document).ready(function(){
   /*
   Creem un primer objecte que ens generarà
   els esdeveniments de temps. Hi connectem els diferents visors de rellotges.
   */
   function Rellotge()
   {
      // Generem un primer esdeveniment.
      $(document).trigger('tiempo', [new Date()]);
      // Fem que cada segon executi un esdeveniment de temps.
      setInterval(function(){
         $(document).trigger('tiempo', [new Date()]);
      }, 1000);
   }

   // Generem una "classe" text visor.
   var TextVisor = function() {
      // Creem un punter al this de l'objecte per
      // emprar-lo en la funció a la qual es crida des de l'esdeveniment, ja que aquesta
      // funció té el this de l'esdeveniment, i no pas de la classe. Com que és una clouser,
      // t és visible a la funció.
      var t = this.t = this;
      // Connectem l'esdeveniment a aquesta funció i
      // des d'aquesta funció executarem el manipulador.
      var tick = function(event, extra) {
         t.pinta( event, extra );
      }
      // Aquesta funció renderitza el resultat de l'esdeveniment.
      this.pinta = function(event, extra) {
         $( '#rellotge' ).html( t.format_string(extra) );
```

```
    }
    // Aquesta funció ens ajuda a pintar una data donada en format
    // correcte 00:00:00.
    this.format_string = function(dat) {
        d = dat;
        hora = d.getHours();
        minuto =d.getMinutes();
        if(minuto<=9)
            minuto = "0"+minuto;
        segundo = d.getSeconds();
        if(segundo<=9)
            segundo = "0"+segundo;
        return hora + ":" + minuto + ":" + segundo;
    }
    // Connectem l'esdeveniment del rellotge.
    $(document).bind('tiempo', tick);
}

// Pintem el div amb alguns estils.
$('#rellotge').css({
    'font-size': '86px',
    'font-family': 'Arial',
    'text-align': 'center',
    'padding-top': '200px'
});

// Creem una instància del visor, que
// funcionarà amb el mètode per defecte que pinta el rellotge.
var visor = new TextVisor();

// Una nova instància, que utilitza el valor del desplegable
// per a mostrar-nos la data en la zona horària seleccionada.
var vis = new TextVisor();
vis.pinta = function(ev, extra) {
    // Convertim la data en mil·lisegons per unificar les conversions
    // i hi restem la diferència horària del nostre país.
    temps = extra.getTime()-3600000;
    // Recuperem la diferència horària del desplegable.
    zona = Number( $('#timezone').val() );
    // El sumem a l'hora actual.
    temps += 3600000 * zona;
    $('#camp_form').val( this.format_string(new Date(temps)) );
}

// Aquest tercer visor connecta el rellotge amb la finestra del navegador.
var vis3 = new TextVisor();
vis3.pinta = function(ev, extra){
```

```
      $(document).attr('title',
         this.format_string(extra) );
   }
   // Inicialitzem el rellotge.
   Rellotge();
});
//-->
</script>
</head>
<body>
   <div id="rellotge"></div><br /><br />
   <div style="text-align:center">
      <select name="DropDownTimezone" id="timezone">
      <option value="-12.0">(GMT -12:00) Eniwetok, Kwajalein</option>
      <option value="-11.0">(GMT -11:00) Midway Island, Samoa</option>
      <option value="-10.0">(GMT -10:00) Hawaii</option>
      <option value="-9.0">(GMT -9:00) Alaska</option>
      <option value="-8.0">(GMT -8:00) Pacific Time (US & Canada)</option>
      <option value="-7.0">(GMT -7:00) Mountain Time (US & Canada)</option>
      <option value="-6.0">(GMT -6:00) Central Time (US & Canada), Mexico City</option>
      <option value="-5.0">(GMT -5:00) Eastern Time (US & Canada), Bogota, Lima</option>
      <option value="-4.0">(GMT -4:00) Atlantic Time (Canada), Caracas, La Paz</option>
      <option value="-3.5">(GMT -3:30) Newfoundland</option>
      <option value="-3.0">(GMT -3:00) Brazil, Buenos Aires, Georgetown</option>
      <option value="-2.0">(GMT -2:00) Mid-Atlantic</option>
      <option value="-1.0">(GMT -1:00 hour) Azores, Cape Verde Islands</option>
      <option value="0.0">(GMT) Western Europe Time, London, Lisbon, Casablanca</option>
      <option value="1.0" selected>(GMT +1:00 hour) Brussels, Copenhagen, Madrid, Paris</option>
      <option value="2.0">(GMT +2:00) Kaliningrad, South Africa</option>
      <option value="3.0">(GMT +3:00) Baghdad, Riyadh, Moscow, St. Petersburg</option>
      <option value="3.5">(GMT +3:30) Tehran</option>
      <option value="4.0">(GMT +4:00) Abu Dhabi, Muscat, Baku, Tbilisi</option>
      <option value="4.5">(GMT +4:30) Kabul</option>
      <option value="5.0">(GMT +5:00) Ekaterinburg, Islamabad, Karachi, Tashkent</option>
      <option value="5.5">(GMT +5:30) Bombay, Calcutta, Madras, New Delhi</option>
      <option value="5.75">(GMT +5:45) Kathmandu</option>
      <option value="6.0">(GMT +6:00) Almaty, Dhaka, Colombo</option>
      <option value="7.0">(GMT +7:00) Bangkok, Hanoi, Jakarta</option>
      <option value="8.0">(GMT +8:00) Beijing, Perth, Singapore, Hong Kong</option>
      <option value="9.0">(GMT +9:00) Tokyo, Seoul, Osaka, Sapporo, Yakutsk</option>
      <option value="9.5">(GMT +9:30) Adelaide, Darwin</option>
      <option value="10.0">(GMT +10:00) Eastern Australia, Guam, Vladivostok</option>
      <option value="11.0">(GMT +11:00) Magadan, Solomon Islands, New Caledonia</option>
      <option value="12.0">(GMT +12:00) Auckland, Wellington, Fiji, Kamchatka</option>
      </select> <input type="text" name="" value="" id="camp_form">
   </div>
</body>
```

```
</html>
```

La funció *Rellotge* inicialitza un temporitzador que genera esdeveniments cada segon, afegint-hi l'hora actual com a paràmetre. La funció anota l'esdeveniment en l'àmbit de document.

A aquest esdeveniment s'hi subscriu la classe *TextVisor*, que representa un visor en mode text. El visor té una funció *pinta*, que ens escriu l'hora actual en un *Div*, amb *id #rellotge*. Hi ha una segona instància de la classe *TextVisor*, a la qual se sobreescriu el mètode *pinta*, i l'associa al contingut d'un desplegable que ens retornarà la diferència horària seleccionada.

Finalment, hi ha una darrera instància en què també se sobreescriu la funció *pinta* i escriu el rellotge en el títol de la finestra del navegador.