DISSENY D'INTERFÍCIES WEB

2. Continguts interactius al web. Introducció a jQuery

Per desenvolupar un lloc web no tan sols cal preparar els elements multimèdia com són les imatges, el vídeo i el so, sinó que s'ha de ser capaç d'integrar-los. Per fer-ho, molt sovint són necessaris coneixements de JavaScript, a banda d'HTML i CSS.

> Podeu trobar una guia bàsica de JavaScript en la secció "Annexos" del web del mòdul.

> **JavaScript** és l'únic llenguatge disponible nadiuament als navegadors i per aquesta raó és el que es fa servir per afegir comportaments dinàmics i interactius al web. Atesa la seva importància, és imprescindible adquirir uns coneixements bàsics sobre aquest.

Habitualment, es fan servir biblioteques que faciliten molt la feina perquè incorporen dreceres per a les tasques més habituals o implementacions genèriques de controls que es poden fer servir en diferents projectes.

2.1. Elements interactius bàsics. Introducció a JQuery

La biblioteca jQuery és una de les més utilitzades i ofereix moltes possibilitats, la qual cosa fa que hi hagi una quantitat extensa de recursos de consulta, tutorials i documentació a Internet que podeu consultar per avançar més en aquest tema pel vostre compte.

> AJAX és un conjunt de tècniques de programació que permeten enviar peticions al servidor i rebre la resposta sense haver de recarregar la pàgina ni obrir-ne una de nova.

Entre d'altres, jQuery facilita la modificació de l'estructura del document, afegir i modificar classes CSS i afegeix moltes funcions que faciliten moltes de les tasques més comunes en treballar amb la interfície del lloc web.

2.1.1. Configuració dels navegadors per a la visualització d'elements interactius

Abans de l'arribada d'HTML5 i les millores aportades per CSS3, quan es volien afegir elements interactius a un lloc web s'havia de recórrer a tecnologies alienes al navegador com Java (amb unes miniaplicacions anomenades *applets*) o molt més freqüentment a

Això simplifica molt la configuració dels navegadors, ja que no cal descarregar cap connector especial, com requeria Flash, ni la màquina virtual de Java. En ser tecnologies nadiues del navegador, poden fer-se servir directament.

Encara que no us heu de preocupar del fet que l'usuari hagi d'instal·lar cap connector en particular, sí que hi ha dues opcions que es poden modificar o desactivar que poden fer que el lloc web deixi de funcionar i mostrar-se correctament:

- Desactivar l'ús de *cookies*: com que no es podrà *recordar* la informació de l'usuari, alguns llocs seran directament inaccessibles, per exemple els llocs web de comerç electrònic.
- Desactivar l'ús de JavaScript: en alguns casos el lloc serà inservible. Per exemple, una web que inclogui mapes de Google Maps no pot funcionar sense JavaScript; en canvi, d'altres només l'utilitzen per afegir alguns efectes, i per a aquests sí que es pot presentar una solució alternativa.

> Les *cookies* són petits fitxers que guarda el navegador amb informació.

Aquests casos s'han de tenir en compte i, com a mínim, s'ha de mostrar un missatge avisant l'usuari que el lloc no pot funcionar o té limitades les seves funcionalitats, perquè aquestes opcions estan desactivades o no estan disponibles.

Per altra banda, amb la inclusió de nous dispositius d'entrada i de sortida s'han afegit alguns controls que limiten a quins d'aquests dispositius es pot accedir. Per exemple, les següents API demanen permís a l'usuari abans de permetre utilitzar-les:

- l'API de geolocalització
- l'API de càmera
- l'API del micròfon

Aquests permisos s'han d'atorgar individualment per a cada pàgina, de manera que, per exemple, un lloc maliciós no pot fer servir la *webcam* sense el consentiment previ.

Tot i així, s'ha de tenir en compte que no tots els navegadors suporten totes les característiques previstes per HTML5, sobretot si es consideren les Web API que van afegint-se contínuament o si voleu fer servir alguna API o biblioteca experimental, com per exemple la WebVR API, que encara està disponible només com a esborrany i només és suportada per Google Chrome i les versions més recents de Mozilla Firefox.

> Podeu trobar més informació sobre la WebVR API en el següent enllaç: goo.gl/4NP6xB.

Un altre exemple més habitual és oferir un mètode alternatiu per reproduir àudio i vídeos quan el navegador no suporta HTML5. Actualment no hi ha cap solució suportada universalment, així que l'únic que es pot fer és mostrar un missatge a l'usuari:

```
  <source src= fitxer_video.webm  type= video/webm >
  Ho sento, el teu navegador no suporta vídeo d'HTML5
</video>
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/xZjRVN.

Com es pot veure en aquest exemple:

1. Primerament, s'intenta fer servir com a font del vídeo un fitxer en format MP4.
2. Si aquest format no és reconegut, s'intenta amb el format WEBM.
3. Si cap dels dos és reconegut, es mostra un missatge informant l'usuari.

Aquests tipus de situacions es poden trobar sempre que es treballa amb HTML5, CSS3 i JavaScript. Poden afrontar-se des de dos angles diferents:

- ***Progressive enhancement***: es comença a desenvolupar el lloc amb les funcionalitats bàsiques que no requereixen cap tecnologia i es va millorant l'experiència d'usuari afegint nous comportaments (i comprovant-los) pas a pas.
- ***Graceful degradation***: amb aquesta tècnica es treballa de manera inversa, primer es desenvolupa el lloc amb totes les tecnologies que es necessiten i després s'afegeixen alternatives de visualització/execució per a les parts que no siguin disponibles globalment.

> Podeu trobar més informació sobre les diferències entre *progressive enhancement* i *graceful degradation* en aquest enllaç: goo.gl/89UPNm.

> Es pot **desactivar JavaScript** des de les opcions (pestanya *General* a Google Chrome) de les eines de desenvolupador, no cal canviar les preferències del navegador.

Vegeu un exemple de totes dues aproximacions. Suposeu que teniu una pàgina amb un botó que crida un *script* que fa uns càlculs:

```
<script>
function calcula() {
  var a = 12,
    b = 30,
    resultat = a + b;

  alert("Resultat: " + resultat);
}
</script>

<button onclick="calcula();">Calcular</button>
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/ONPvgP.

Com que no hi ha manera de fer el càlcul si no es troba activat el JavaScript al navega-

alguns usuaris amb dispositius mòbils.

Aplicant la tècnica de la ***graceful degradation*** podríeu fer el següent:

```
<script>
function calcula() {
  var a = 12,
    b = 30,
    resultat = a + b;

  alert("Resultat: " + resultat);
}
</script>

<button onclick="calcula();">Calcular</button>
<noscript>
  <p>
    Per realitzar el càlcul és necessari tenir JavaScript activat. Si us p
  </p>
</noscript>
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/RaNMxz (malauradament, si desactiveu el JavaScript al vostre navegador CodePen deixarà de funcionar, ho heu de provar en local).

Com podeu veure, s'ha fet servir l'element `noscript` per mostrar el missatge en cas que el navegador no tingui el JavaScript activat.

Tot i que ara es preveuen totes dues possibilitats, en el moment de la implementació s'ha fet sabent que és possible que alguns usuaris no tinguin activat JavaScript, i que fins i tot pot ser que no tinguin opció d'activar-lo, de manera que aquests usuaris es troben amb una pàgina amb un botó que no fa res i un missatge que tampoc no els ajuda a solucionar el seu problema.

En aquest cas en concret no seria possible aplicar el ***progressive enhancement*** sense recórrer a fer servir algun llenguatge al servidor, per exemple PHP, fer l'enviament de les dades a la pàgina i que aquesta retornés el resultat.

Un altre cas molt habitual en què és interessant aplicar la ***graceful degradation*** és quan es volen aplicar efectes CSS més avançats, per exemple un degradat:

> Hi ha moltes eines en línia que permeten generar automàticament el codi CSS aplicant la tècnica del ***graceful degradation***, per exemple: www.color-zilla.com /gradient-editor.

```
<style>
  div {
    width: 100px;
    height: 100px;

    /* Degradat */
    background: #f0f9ff; /* Navegadors antics, color pla */
    background: -moz-linear-gradient(-45deg,  #f0f9ff 0%, #cbebff 47%, #a1
    background: -webkit-linear-gradient(-45deg,  #f0f9ff 0%,#cbebff 47%,#a
    background: linear-gradient(135deg,  #f0f9ff 0%,#cbebff 47%,#a1dbff 10
    filter: progid:DXImageTransform.Microsoft.gradient( startColorstr='#f0
  }
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/RaNEvV.

Com que els fulls CSS s'apliquen en cascada, la primera declaració és sobreescrita per les posteriors, de manera que l'última en aplicar-se és l'última que sigui suportada. Així doncs, en els navegadors més antics s'aplicarà un color pla, en versions antigues dels navegadors que ho suporten es crida afegint el prefix (era necessari en versions anteriors), i els navegadors moderns fan servir la declaració de l'especificació, com indica el W3C.

A continuació veureu un exemple en què aplicar la tècnica del ***progressive enhancement*** sí que és una molt bona alternativa. Suposeu que teniu una versió més avançada de l'exemple anterior que permet introduir dos valors i mostrar el resultat.

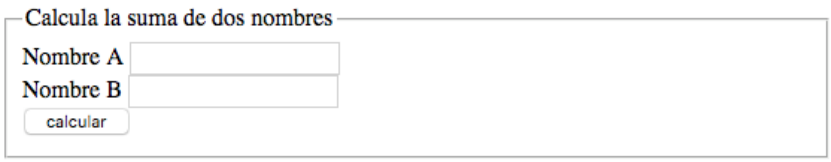
Es comença afegint només l'estructura del formulari amb el marcat d'HTML:

```
<form action="" method="GET" id="principal">
  <fieldset>
    <legend>Calcula la suma de dos nombres</legend>
    <label>Nombre A</label>
    <input name="number_a" type="text">
    <br>
    <label>Nombre B</label>
    <input name="number_b" type="text">
    <br>
    <input value="calcular" type="submit">
  </fieldset>
</form>
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/pyvLGW. En aquest cas en tractar-se d'un exemple no cal indicar l'URL a la que s'enviarà el formulari a través de la propietat `action`.

Com que només inclou HTML bàsic, podeu estar segurs que aquest exemple funcionarà en tots els navegadors (vegeu la figura 2.1).

Figura 2.1. Implementació simple amb HTML



S'afegeix una segona capa, aquest cop amb el codi CSS:

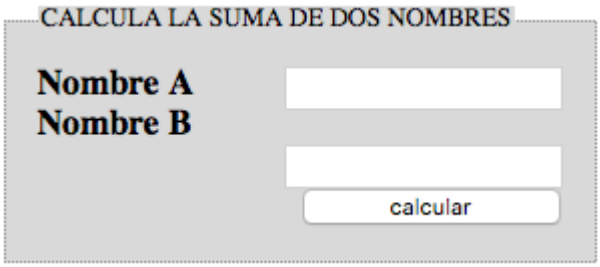
> En alguns navegadors és possible desactivar el CSS, i en els casos dels lectors de pantalles utilitzats per les persones amb discapacitats visuals, els fulls CSS no són aplicables.

```
<style>
  fieldset {
    max-width: 300px;
    border: 1px dotted grey;
```

```
    text-transform: uppercase;
    font-weight: italic;
    font-size: 0.8em;
  }

  label {
    font-weight: bold;
    font-size: 1.1em;
  }

  input {
    width:50%;
    float: right;
  }
</style>
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/VaYXRo i la seva visualització a la figura 2.2.

Figura 2.2. Implementació afegint CSS



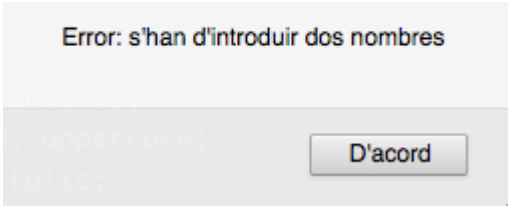
Es conserva tota la funcionalitat, però ara s'ha millorat l'aspecte bàsic del formulari. El següent pas és afegir una nova capa amb codi JavaScript, que serà l'encarregat de validar la informació del formulari abans de fer l'enviament:

```
<script>
  var formulari = document.getElementById('principal'),
  numberA = document.getElementsByName('number_a')[0],
  numberB = document.getElementsByName('number_b')[0];

formulari.addEventListener("submit", function(e) {

  if (!isNumeric(numberA.value) || !isNumeric(numberB.value)) {
    e.preventDefault();
    alert("Error: s'han d'introduir dos nombres");
  } else {
    alert("Correcte: enviant formulari");
  }
});

function isNumeric(n) {
  return !isNaN(parseFloat(n)) && isFinite(n);
}
</script>
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/ONPadN i la seva visualització a la figura 2.3.

Figura 2.3. Implementació afegint JavaScript

DISSENY D'INTERFÍCIES WEB



Aplicant el ***progressive enhancement*** s'ha fet que el codi sigui funcional per a tots els usuaris, millorant l'experiència d'usuari a mesura que les capacitats dels navegadors dels usuaris augmenten.

2.1.2. Eines per a la inclusió de contingut multimèdia

Com que actualment tot el contingut multimèdia que es troba al web és gestionat i manipulat fent servir JavaScript, HTML i CSS, no hi ha gaires eines que alleugereixin la nostra tasca.

Existeixen eines que exporten continguts com a HTML, CSS i JavaScript. Principalment es tractarà de *software* privatiu, com Adobe Animate CC, que permet la creació de continguts multimèdia i que exporta aquests com a codi HTML, CSS i JavaScript. O eines com Unity 3D, que permet desenvolupar jocs fent servir el llenguatge C# i exportar-los a diferents plataformes, entre les quals HTML5 i WebGL. Però com a desenvolupadors serà més habitual que us trobeu treballant amb codi.

El que sí que utilitzareu molt sovint són biblioteques que us facilitaran enormement la vostra tasca, ja que automatitzen les accions més repetitives, simplifiquen la sintaxi i s'encarreguen de gestionar les diferències entre navegadors, fent servir internament diferents implementacions segons siguin o no suportades determinades característiques.

Un altre tipus d'eina molt útil són les aplicacions en línia que es poden trobar per resoldre problemes molt concrets i que us generen codi automàticament, per exemple els generadors de codi CSS3 que faciliten la tasca d'afegir ombres, degradats i altres efectes a partir d'una interfície gràfica. Al llarg d'aquest apartat utilitzareu algunes d'aquestes eines per ajudar-vos a desenvolupar el vostre codi.

> Podeu trobar més informació sobre jQuery a la seva pàgina oficial: jquery.com.

Si accediu a la pàgina de descàrrega de jQuery (jquery.com/download) veureu que hi ha diverses opcions.

> Es poden trobar versions les versions 1.x (suport per navegadors antics) i 2.x al següent enllaç: code.jquery.com

Per una banda es troba la versió anomenada *compressed* (comprimida) i per altra la versió *uncompressed* (descomprimida). Quan estigueu desenvolupant sempre heu

primida és inintel·ligible per depurar, però té un pes menor i és la que s'ha de fer servir en l'entorn de producció.

Compressors en línia

Una de les solucions més simples per comprimir els nostres fitxers CSS i JavaScript és fer servir un **compressor en línia**, com per exemple jscompress.com, que permet comprimir el codi JavaScript només enganxant-lo i clicant un botó.

Els **fitxers de JavaScript**, en ser text pla, realment no es comprimeixen, el que es fa és modificar-los fent servir altres eines de manera que s'eliminin tots els comentaris, els espais innecessaris i els salts de línia, i se substitueix el nom de les variables per d'altres més curts (normalment, una sola lletra). D'aquesta manera, la mida del fitxer es redueix i el temps de descàrrega és menor.

Una vegada decidiu quina versió voleu utilitzar només l'heu de descarregar i copiar al mateix directori que la resta de fitxers del nostre lloc web o en algun dels seus subdirectoris.

Per exemple, si heu descarregat la versió descomprimida, tindreu un fitxer amb un nom semblant a jquery-3.1.1.js. Suposeu que el copieu dins del subdirectori *js/biblioteques* dins del directori que conté tots els vostres fitxers. Per carregar-lo a la pàgina hauríeu d'afegir:

```
<script src="js/biblioteques/jquery-3.1.1.js"></script>
```

Xarxes de lliurament de continguts (CDN)

Un CDN és una xarxa de servidors localitzats en diferents punts geogràfics però amb els mateixos continguts, de manera que quan es rep una petició del fitxer aquest és enviat des del servidor més proper a l'usuari. D'aquesta manera, s'augmenta la velocitat de la resposta. Podeu trobar més informació sobre les xarxes de lliurament de continguts en el següent enllaç: ca.wikipedia.org/wiki/Xarxa_de_lliurament_de_continguts.

En alguns casos pot interessar-vos no allotjar la biblioteca en el vostre servidor. Per exemple, si la pàgina web es preveu que sigui visitada per usuaris internacionals, serà una millor opció fer servir un CDN (xarxa de lliurament de continguts); en el cas de jQuery, ells mateixos ens ofereixen aquest servei, i és tan simple d'utilitzar com afegir la ruta code.jquery.com/jquery-3.1.1.js (substituint el nom del fitxer per la versió que correspongui) en lloc de la del fitxer descarregat, per exemple:

No es pot **carregar una biblioteca** fent servir un CDN si obrim la pàgina directament al nostre ordinador com un fitxer local. En aquests casos haureu de descarregar el fitxer i carregar-lo localment, o bé accedir a la pàgina a través d'un servidor web que tinguem configurat en el nostre equip.

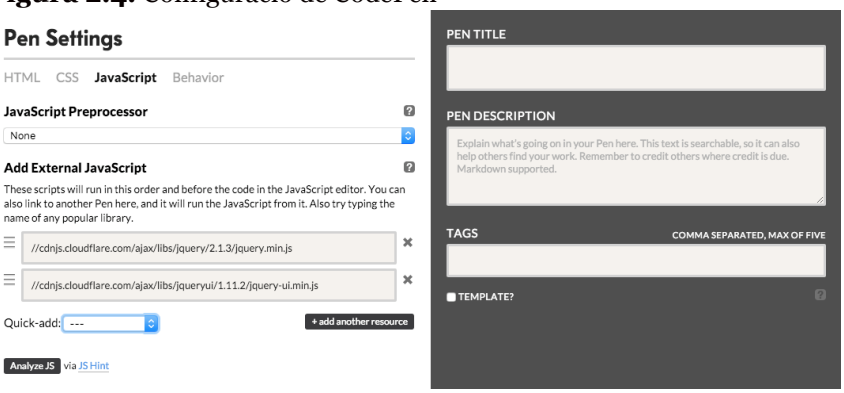
2.1.3. Introducció a jQuery

El primer pas per poder fer servir jQuery al codi és carregar la biblioteca corresponent. En els exemples que es veuran a continuació es fa servir la versió 3.1.1 i és càrrega a través CDN proporcionat pels desenvolupadors de jQuery:

```
<script src="//code.jquery.com/jquery-3.1.1.js"></script>
```

CodePen inclou l'opció de carregar automàticament algunes de les biblioteques externes més populars (vegeu la figura 2.4), com és el cas de JQuery i jQuery UI. Per aquesta raó no s'inclou la càrrega d'aquestes al codi d'exemple, perquè són carregades automàticament.

Figura 2.4. Configuració de CodePen



S'han de tenir en compte dues coses abans de treballar amb jQuery:

- S'ha de carregar la biblioteca abans de fer-la servir.
- El **DOM** (l'estructura del document) ha d'estar completament carregat.

El model d'objectes del document, més conegut com a **DOM** (*Document Object Model*) per les seves sigles en anglès, és una interfície que ens facilita treballar amb documents HTML, entre d'altres.

El DOM defineix els mètodes i les propietats que ens permeten accedir i manipular el document com si es tractés d'un arbre de nodes, on l'arrel és el node `document` que pot contenir qualsevol quantitat de nodes fills i les fulles, els nodes que no tinguin cap des-

Una de les funcionalitats que inclou jQuery és la capacitat de seleccionar i manipular aquests elements fàcilment, ja sigui per modificar-los, per eliminar-los o per afegir-ne de nous.

Encara que el primer punt sembla obvi, com que és possible carregar els fitxers amb el codi de manera asíncrona és molt probable que el vostre fitxer amb el codi JavaScript acabi de descarregar-se abans que la biblioteca, pel fet de ser més lleuger.

El segon punt pot fer que apareguin problemes de difícil detecció, ja que pot ser que comenci a executar-se el codi abans que s'hagi generat el DOM completament.

Per aquesta raó, sempre s'ha d'afegir el vostre codi dins de la funció que és cridada pel mètode `ready` de jQuery, que és cridat una vegada s'ha generat el DOM completament:

```
<script src="//code.jquery.com/jquery-3.1.1.js"></script>
<script>
  $(document).ready(function() {
    // Aquí va el nostre codi
  });
</script>
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/KzwEXN.

> Generalment, les biblioteques fan servir variables i mètodes amb aquests símbols per ressaltar que es tracta d'alguna funció, propietat o objecte especial.

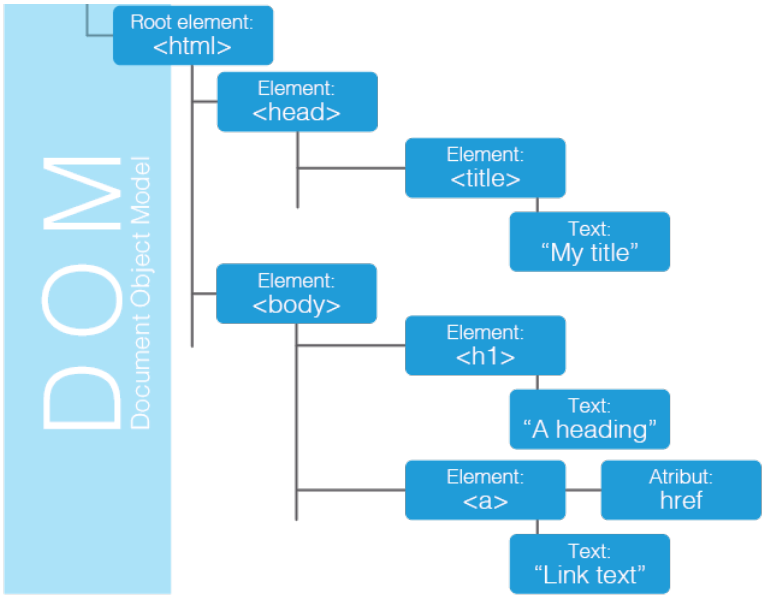
El primer que haureu vist és que hem fet servir el símbol del dòlar (`$`). A JavaScript es pot fer servir com a nom o part del nom de les variables i funcions. És a dir, **és possible, però no recomanable**, posar com a nom a una variable `$$variable$$`.

> Hi ha casos en què cal utilitzar `jQuery` com a nom per evitar conflictes amb altres biblioteques.

En aquest cas, el símbol `$` és una drecera per a `jQuery`, així que `$(document)` el que fa és cridar la funció `jQuery(document)`.

A continuació, cal tenir clar què és aquest paràmetre que s'ha passat a la funció `jQuery`. Quan una pàgina web és carregada en un navegador, el codi es converteix en una estructura de dades similar a un arbre: el DOM. El node arrel d'aquest arbre és l'anomenat `document`, i la resta d'elements pengen d'aquest dins d'aquesta estructura. Podeu veure una representació d'aquest arbre a la figura 2.5.

Figura 2.5. Representació del DOM per a un document HTML



Així doncs, quan es fa servir el mètode `ready` sempre passarem l'objecte `document` com a paràmetre, ja que aquest és el responsable d'indicar quan s'ha acabat de generar l'arbre perquè tots els elements de la pàgina són descendents seus.

En síntesi, aquest exemple el que fa és *dir-li* a jQuery (`$`) que es vol executar el codi (`function() {};`), una vegada el document (`document`) estigui llest (`ready`).

> El mètode `ready()`, encara que té una funció similar a `<body onload="funcioACridar();">`, **és incompatible amb aquest**.
> Si es fa servir un no s'ha de fer servir l'altre.

S'han de distingir dues parts en aquest codi. Per una banda, es fa una crida a la funció `jQuery`, que retorna un objecte, i després es crida un mètode d'aquest objecte (en aquest cas es tractava de `ready`).

La funció `jQuery` converteix el paràmetre en un objecte jQuery, i aquest paràmetre pot ser un node com `document`, una cadena de text amb un **selector CSS**, com per exemple `p`, o una cadena de codi HTML, com per exemple `<p>Hola, món</p>`.

Vegem aquests objectes amb el següent exemple:

```
<script src="//code.jquery.com/jquery-3.1.1.js"></script>
<script>
$(document).ready(function() {
  console.log('document:', $(document));
  console.log('p: ', $('p'));
  console.log('<p>Hola, Món</p>: ', $('<p>Hola, Món</p>'));
});
</script>
<p></p>
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/GZJJzY.

El mètode `console.log`

El mètode `console.log` mostra els valors dels paràmetres per a la consola de les eines de desenvolupador. Per poder veure el resultat heu d'anar a les opcions del vostre navegador:

- Google Chrome: botó desplegable d'*Opcions del navegador / Més eines / Eines de desenvolupador*.
- Mozilla Firefox: menú *Eines / Desenvolupador web / Mostra les eines*.

Una vegada es mostri el panell d'eines, veureu una pestanya anomenada *Console* o *Consola*; si la seleccioneu, podreu veure el resultat, similar a la figura 2.6.

Figura 2.6. Visualització de la consola de Google Chrome

Si desplegueu els continguts de cada objecte podreu veure que la seva estructura és diferent. Cal destacar que:

- L'objecte creat pel `document` no té la propietat `prevObject` perquè no hi ha cap objecte anterior, ni la propietat `selector`.
- L'objecte generat a partir del selector `p` conté tant un objecte anterior (`prevObject`), que es correspon amb el document (és lògic, perquè l'element està inclòs en el document), i la propietat `selector`.
- L'objecte generat a partir de la cadena de text no té cap element previ perquè no s'ha afegit al document; tampoc no té context, per la mateixa raó, i no es pot seleccionar.

Tots tenen la propietat `length` amb valor 1, això és perquè s'ha trobat només un element de cada tipus. Aquesta és una propietat important, perquè quan es treballa amb selectors l'objecte jQuery habitualment contindrà seleccionats múltiples elements:

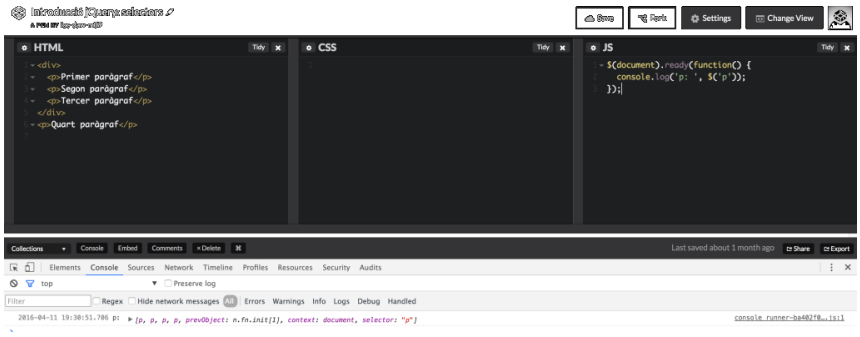
```
<script src="//code.jquery.com/jquery-3.1.1.js"></script>
<script>
  $(document).ready(function() {
    console.log('p: ', $('p'));
  });
</script>
<div>
  <p>Primer paràgraf</p>
  <p>Segon paràgraf</p>
  <p>Tercer paràgraf</p>
</div>
<p>Quart paràgraf</p>
```

Selectors jQuery

jQuery suporta tots els selectors de CSS3, a més de XPath i alguns propis, com es pot veure en el següent enllaç: goo.gl/v1HEBN.

En aquest exemple podeu veure que la funció ha retornat un objecte amb mida 4 perquè s'han trobat quatre elements que coincideixen amb el selector `p` que correspon al paràgraf, com es pot veure a la figura 2.7.

Figura 2.7. Selecció d'elements de tipus paràgraf



En canvi, si canvieu el selector per `div > p`, el resultat és diferent, ja que només són seleccionats els paràgrafs que siguin descendents directes d'un element `div`. Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/grpaxN.

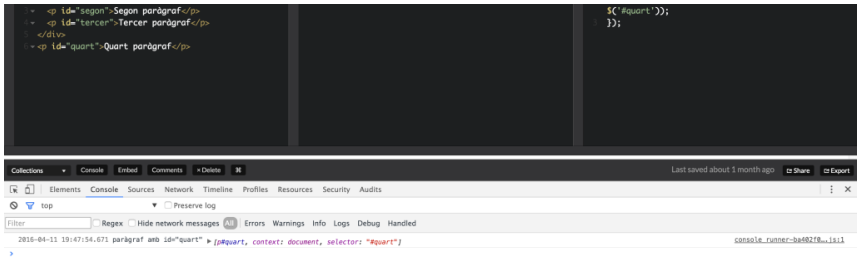
Per descomptat, poden seleccionar-se també elements per a `id`, només cal afegir el símbol # a l'identificador:

```
<script>
  $(document).ready(function() {
    console.log('paràgraf amb id="quart"', $('#quart'));
  });
</script>
<div>
  <p id="primer">Primer paràgraf</p>
  <p id="segon">Segon paràgraf</p>
  <p id="tercer">Tercer paràgraf</p>
</div>
<p id="quart">Quart paràgraf</p>
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/GZJmaY.

A la consola aquest cop la mida és 1, i només conté l'element amb `id="quart"`, com es pot veure a la figura 2.8.

Figura 2.8. Selecció d'un element per la seva id



Ara que ja heu vist com seleccionar un o més elements de la pàgina, vegeu com podeu aplicar un mateix canvi a tots aquests, per exemple per modificar el seu estil CSS:

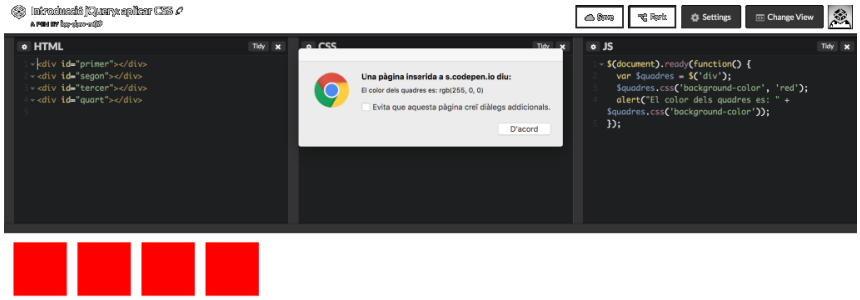
```
<style>
div {
  width: 100px;
  height: 100px;
  margin: 10px;
  float: left;
  background-color: lightgrey;
}
</style>
<script>
$(document).ready(function() {
  var $quadres = $('div');
  $quadres.css('background-color', 'red');
  alert("El color dels quadres és: " + $quadres.css('background-color'));
});
</script>
<div id="primer"></div>
<div id="segon"></div>
<div id="tercer"></div>
<div id="quart"></div>
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/Mywowy.

> Quan es fa servir una variable que emmagatzema un objecte jQuery s'acostuma a prefixar amb el símbol `$`, de manera que és molt clar que es tracta d'un objecte jQuery.

Primerament, s'ha guardat el resultat de la funció en `$quadres`, que contindrà l'objecte retornat per jQuery. A continuació s'ha cridat el mètode `css()` per establir la propietat `background-color` al color vermell, i finalment es mostra un quadre d'alerta recuperant el valor de la propietat `background-color` de l'objecte jQuery, com es pot apreciar a la figura 2.9.

Figura 2.9. Modificació del color mitjançant jQuery



nombre de paràmetres, de manera que si se'n passen dos intentarà establir-la, i si només se'n passa un intentarà recuperar-la. Molts dels mètodes de jQuery funcionen d'aquesta manera.

Però com ho faríeu per aplicar canvis a cada element individualment? Una opció seria afegir un `id` diferent a cada element i després declarar una variable amb un objecte jQuery per a cadascun, però això no seria gens eficient. Per a aquests casos, jQuery ens proporciona un mètode per recórrer a tots els elements i aplicar-hi els canvis de manera individual, substituint el codi JavaScript pel següent:

```
$(document).ready(function() {
  var $quadres = $('div');

  $quadres.css('background-color', 'red');

  $quadres.each(function(i) {
    var alcada = 25 + i * 25;
    $(this).css('height', alcada + 'px');
  });
});
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/GZJEpP, i el resultat a la figura 2.10.

Figura 2.10. Quadres amb la mida modificada individualment

Diferents usos d'each a jQuery

jQuery té dos mètodes anomenats `each` que fan coses diferents. Si es crida a partir d'un objecte amb una selecció d'elements, el que fa és recórrer a aquests elements i aplicar la funció cridada a cada element. En canvi, si es crida directament sobre jQuery (`$.each(array, funcio)`), el que fa és recórrer a l'*array* i aplicar la funció a cada element d'aquest.

En aquest últim cas seria més adient dir que el que hem fet és cridar la **funció** `each` de la biblioteca per diferenciar-lo del primer cas, en què s'ha cridat el mètode `each` d'una instància d'un objecte concret.

S'ha afegit una crida al mètode `each` i s'ha passat com a paràmetre una funció. Aquesta funció és cridada per a cadascun dels elements de l'objecte jQuery, en aquest cas els quatre elements `div`. El valor del paràmetre `i` es correspondrà en cada cas amb l'índex de l'objecte (sent 0 el primer i 3 l'últim en aquest exemple).

d'unitat a fites de la línia, per exemple: px o em.

Dins de la funció, el que es fa és calcular l'alçada de cada element amb la fórmula `25 + i * 25`; a continuació es crea un nou objecte jQuery que fa referència a aquest element amb `$(this)` (com que no es fa res més amb aquest no cal guardar-lo a cap variable), i es crida el mètode `css` com en el cas anterior, però aquest cop per a la propietat `height`.

Fixeu-vos que per poder manipular cadascun d'aquests elements s'ha hagut de cridar de nou la funció `jQuery` passant l'element (`this` fa referència a l'element processat).

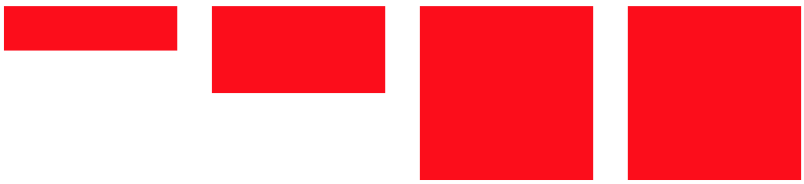
Però en alguns casos pot interessar-vos aturar el recorregut, potser cerqueu un element en concret i una vegada el trobeu voleu sortir. En aquest cas només cal retornar `false` quan es doni la condició per finalitzar el recorregut; per exemple, afegint aquest codi dins de la funció cridada:

```
if (i === 1) {
  return false;
}
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/MywoBo.

Aquest canvi fa que una vegada el valor de `i` sigui igual a 1 (just després de canviar la mida del segon element) es retorni `false` i els dos elements restants no siguin processats, i per tant la seva alçada es mantingui igual, com es pot veure a la figura 2.11.

Figura 2.11. Només la meitat dels quadres són processats



Fins ara heu vist com treballar amb tots els elements seleccionats, però com ho faríeu si només volguéssiu treballar amb un en concret? Hi ha dues maneres d'accedir a aquests elements: a través del mètode `get` o directament com si es tractés d'un *array* especificant l'índex. Substituïu el contingut del bloc de JavaScript pel següent:

```
$(document).ready(function() {
  var $quadres = $('div');

  $($quadres[2]).css('background-color', 'red');

  $($quadres.get(0)).css('background-color','green');

});
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/KzpqGa, i el resultat a la figura 2.12.

DISSENY D'INTERFÍCIES WEB

S'ha de tenir en compte que encara que és possible, **no és recomanable accedir a l'element com si es tractés d'un *array***. S'ha de fer servir el mètode `get` perquè és el que la interfície de jQuery ens garanteix que funcionarà correctament.

En el primer cas s'ha seleccionat el tercer element com si es tractés d'un *array*. Fixeu-vos que per poder cridar el mètode `css` s'ha hagut de tornar a cridar la funció jQuery passant-li com a argument el node emmagatzemat a la tercera posició de l'*array*: `$($quadres[2])`.

> Si no es passa cap argument al mètode `get` retorna un *array* amb tots els nodes seleccionats.

En el segon s'ha fet correctament, cridant el mètode `get` passant com a argument la posició 0 de l'*array*, a continuació s'ha passat el node retornat a la funció `jQuery` i s'ha cridat el mètode `css` per canviar la propietat `background-color` a verd.

El mètode `get` retorna el node seleccionat de la posició; no es tracta d'un objecte jQuery. Un dels tipus de paràmetres que es pot passar a la funció `jQuery` és precisament un node. Així doncs, el que s'ha fet és crear un nou objecte jQuery a partir d'aquest node i seguidament cridar el mètode `css`.

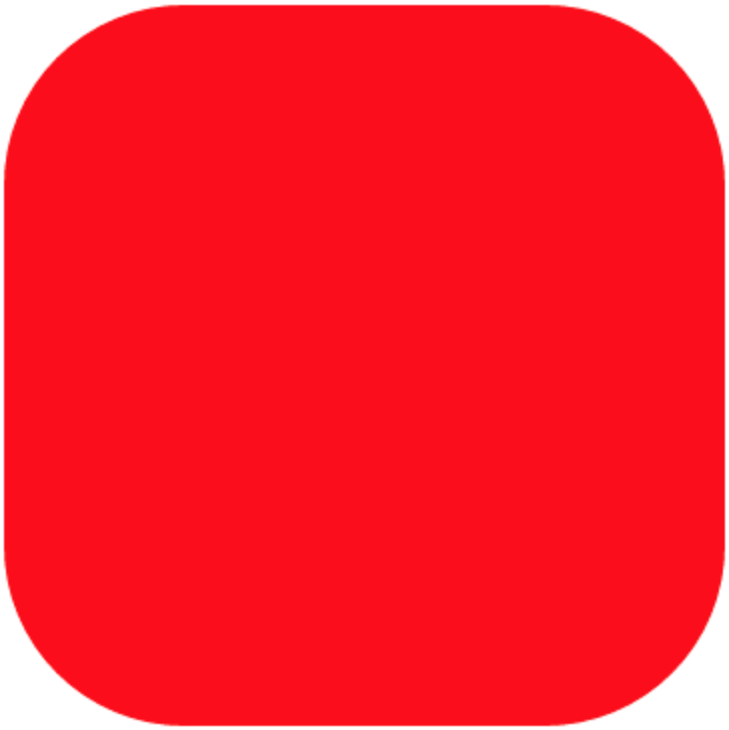
A banda d'afegir o modificar propietats de l'estil CSS, també podeu afegir o eliminar classes d'aquests nodes. Per exemple:

```
<style>
  div {
    width: 100px;
    height: 100px;
    margin: 10px;
    background-color: lightgrey;
  }

  .vores {
    border-radius: 50px;
  }

  .vermell {
    background-color: red;
  }

  .gran {
    width: 200px;
    height: 200px;
  }
</style>
<script>
  $(document).ready(function() {
    var $quadre = $('div');
    $quadre.addClass('vermell');
    $quadre.addClass('gran');
  });
</script>
```

DISSENY D'INTERFÍCIES WEB

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/WwvXMr, i el resultat a la figura 2.13.

Figura 2.13. Múltiples classes afegides a un element



En aquest cas s'han afegit dues classes CSS amb el mètode `addClass`, una anomenada `vermell` i una `gran`, que són afegides en temps d'execució i fan que el quadre tingui `200×200` px de mida i color vermell en lloc de gris.

Per eliminar-les és igual de fàcil, només s'ha de fer servir el mètode `removeClass`. Afegiu aquesta línia dintre de la funció:

```
$quadre.removeClass('vores');
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/jqPaKM, i el resultat a la figura 2.14.

Figura 2.14. Classe eliminada d'un element

DISSENY D'INTERFÍCIES WEB

Com que s'ha eliminat la classe que tenia associada inicialment, ja no són visibles les vores arrodonides. En cas de voler eliminar totes les classes només heu de cridar `removeClass` sense passar cap argument. Afegiu aquesta línia dintre de la funció:

```
$quadre.removeClass();
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/BKNmPy.

Ara el quadre es veu de color gris clar i de la mida original, 100×100 px.

Events de jQuery

A banda del mètode `on` poden afegir-se *handlers* (funcions que són cridades en disparar-se l'*event* fent servir mètodes específics com `ready` o `click`). Podeu trobar una llista de totes aquestes dreceres en el següent enllaç: goo.gl/U8yxFW.

Altra característica molt important de jQuery és la facilitat amb la qual podeu afegir la detecció d'*events* (esdeveniments), com per exemple: si s'ha clicat un element o s'ha passat el cursor per sobre. Substituïu el codi JavaScript de l'últim exemple per aquest:

```
$(document).ready(function() {
  var $quadre = $('div');

  $quadre.on('mouseover', function(e) {
    $(this).addClass('vermell');
  })

  $quadre.on('mouseout', function(e) {
    $(this).removeClass('vermell');
  })
```

DISSENY D'INTERFÍCIES WEB

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/KzpyGa.

Si passeu el cursor per sobre del cercle veureu que canvia a color vermell, això és perquè es dispara l'*event* `mouseover`; en canvi, en sortir el cursor de sobre es dispara `mouseout` i s'elimina la classe `vermell`.

Primer es crida el mètode `on` i se li passa com a paràmetres una cadena amb el nom de l'*event* (o *events* separats per espais) que es volen escoltar, per exemple `mouseover`, i a continuació la funció que es cridarà.

Aquesta funció rep com a paràmetre un objecte que conté tota la informació de l'*event* (en aquest cas s'ha anomenat `e`, però podeu posar el nom que vulgueu), encara que no es fa servir en aquest exemple. Com ja heu vist en parlar del mètode `each`, `this` fa referència al node on s'ha produït l'*event*, així que per treballar amb aquest com un objecte jQuery heu de cridar la funció `jQuery` i passar el node com a paràmetre: `$(this)`.

A continuació crideu el mètode `addClass` o `removeClass` per afegir o treure la classe, respectivament, com ja s'ha vist a l'exemple anterior.

Tots els *events* funcionen de manera similar; vegeu com funciona l'*event* `click` i el mètode `toggleClass` per alternar l'activació de la classe cada vegada que es clica a sobre, i substituïu el codi JavaScript de l'últim exemple per aquest:

```
$(document).ready(function() {
  var $quadre = $('div');

  $quadre.on('click', function(e) {
    $(this).toggleClass('vermell');
  })

});
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/RaPjvz.

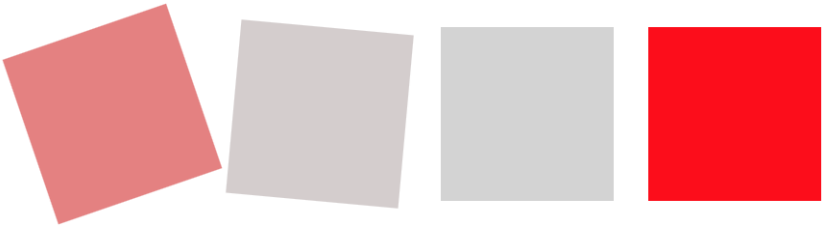
Ara, en fer clic sobre l'element es crida la funció que al seu torn crida el mètode `toggleClass` sobre l'element i afegeix o treu la classe `vermell` segons si aquesta es troba present o no. Aquesta funcionalitat es pot combinar amb les animacions i transicions de CSS3; canvieu a l'exemple anterior el codi CSS pel següent:

```
div {
  width: 100px;
  height: 100px;
  margin: 10px;
  float: left;
  background-color: lightgrey;
  transition: 1s ease;
}

.vores {
  border-radius: 50px;
}
```

```
  transform: rotateZ(180deg);
}
```

Podeu veure aquest exemple en següent enllaç: codepen.io/ioc-daw-m09/pen/aNOxNR, i el resultat a la figura 2.15.

Figura 2.15. Efecte en clicar els elements



Al primer bloc s'afegeix la propietat `transition`, amb una durada d'un segon, i `ease` com a funció d'interpolació per suavitzar-la. Això fa que en canviar qualsevol propietat, el canvi es produeixi progressivament al llarg d'un segon.

A la classe `vermell` s'han canviat les propietats `border-radius` per eliminar les vores arrodonides i s'ha afegit una rotació de 180º sobre l'eix Z.

Ara, en clicar sobre el cercle es pot veure com canvia de color, rota i es converteix en un quadrat. Tot això sense haver de modificar el codi JavaScript, només modificant el CSS. Com es pot apreciar en aquest petit exemple, podeu tenir dos llocs web amb exactament el mateix codi i només canviant el CSS aconseguir efectes i experiències completament diferents.

Vegeu un altre exemple de com combinar CSS i jQuery afegint animacions de CSS3 combinades amb els *events* `mouseover` i `mouseout`:

```
<style>
  div {
    width: 100px;
    height: 100px;
    float: left;
    border: 1px solid lightgrey;
    margin: 10px;
    opacity: 0.5;
    transition: 0.5s ease-out;
  }

  .pols {
    background-color: blue;
    animation-name: pols;
    animation-duration: 0.5s;
    animation-iteration-count: infinite;
    animation-direction: alternate;
  }

  @keyframes pols {
    from {
      transform: scale(1.0);
      opacity: 0.5;
    }
    to {
      transform: scale(1.2);
      opacity: 1;
    }
  }
</style>
<script>
  $(document).ready(function() {
    var $quadres = $('div');
```

```
    })

    $quadres.on('mouseout', function(e) {
      $(this).removeClass('pols');
    })
  });
</script>
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/reVbXb, i el resultat a la figura 2.16.

Figura 2.16. Efecte en clicar als elements

En aquest cas s'ha tornat a fer servir el mètode `on` amb els *events* `mouseover` i `mouseout` per detectar quan entra i quan surt el cursor dels quadres, i els mètodes `addClass` i `removeClass` per afegir o eliminar la classe `pols`, que afegeix:

- Un canvi de color del fons a blau.
- Una animació que s'ha definit amb el nom de `pols`, amb una durada de 0.5 s, que es repetirà indefinidament i amb la direcció de l'animació alterna, és a dir, que quan acaba torna enrere.

Queda pendent veure una altra de les característiques més potents de jQuery: com afegir i eliminar nodes del document i manipular el DOM:

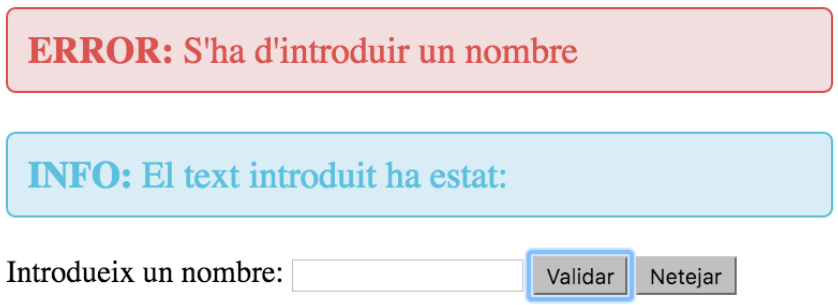
```
<style>
  .missatge {
    border-radius: 5px;
    padding: 10px;
    font-size: 1.2em;
  }

  .error {
    background-color: #f2dede;
    color: #d9534f;
    border: 1px solid #d9534f;
  }

  .exit {
    background-color: #dff0d8;
    color: #5cb85c;
    border: 1px solid #5cb85c;
  }

  .info {
    background-color: #d9edf7;
    color: #5bc0de;
    border: 1px solid #5bc0de;
  }
</style>

<script>
 $(document).ready(function() {
  var $avisos = $('#avisos'),
    $entrada = $('#entrada'),
    $validar = $('#validar'),
    $netejar = $('#netejar');

  $validar.on('click', function(e) {
```

```
      afegirAvis('exit', 'Has introduït un nombre!');
    } else {
      afegirAvis('error', "S'ha d'introduir un nombre");
    }

    afegirAvis('info', 'El text introduït ha estat: ' + $entrada.val());
  });

  $netejar.on('click', function(e) {
    esborrarAvisos();
  });

  function afegirAvis(tipus, missatge) {
    var definicioNode = '<p class="missatge ' + tipus + '"><b>' + tipus.to

    $avisos.append(definicioNode);
  }

  function esborrarAvisos() {
    $avisos.find('p').remove();
  }

});
</script>

<div id="avisos"></div>
<label>Introdueix un nombre:
<input type="text" id="entrada"/>
</label>
<button id="validar">Validar</button>
<button id="netejar">Netejar</button>
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/eZVVmd.

Si proveu aquest exemple veureu que us mostrarà diferents missatges segons si la informació afegida ha passat la validació o no, com es pot apreciar a la figura 2.17.

Figura 2.17. Missatges de validació



Primerament, s'ha definit l'estructura del document, establint els `id` necessaris per poder identificar clarament on s'han de mostrar els `avisos`, quin és el text amb l'`entrada` de l'usuari, i quin és el botó per `validar`.

A continuació s'han afegit les classes CSS, una genèrica per a tots els avisos, anomenada `missatge`, que estableix les vores arrodonides, el `padding` i la mida de la font, i altres específics per canviar el color de l'avís segons si es tracta d'un tipus o un altre: `error`, `exit` i `info`.

> JavaScript no proporciona cap mètode infal·lible per comprovar si una cadena es correspon a un nombre o no; per aquesta raó, és millor fer servir

DISSENY D'INTERFÍCIES WEB

El codi JavaScript no és gaire diferent del que s'ha vist fins ara, però inclou un parell de funcions pròpies:

- `afegeixMissatge(`***`tipus, missatge`***`)`: afegeix el missatge del tipus passat com a argument dins del `div` amb `id="avisos"`.
- `esborrarAvisos()`: esborra tots els avisos.

Al principi es troba la declaració de les variables i l'assignació als tres elements que es volen controlar: el contenidor dels avisos, l'entrada de text i el botó.

A continuació es fa servir el mètode `on` per escoltar quan es dispara l'*event* `click` sobre el botó, i una vegada clicat es produeixen les següents accions:

- S'esborren tots els avisos `esborrarAvisos()`.
- Es comprova si el valor introduït és o no un nombre amb `$.isNumeric($entrada.val())`.
- Si és numèric, es crida `afegeixMissatge` indicant que el tipus serà `exit` i el missatge.
- Si no ho és, es crida també `afegeixMissatge`, però aquest cop el tipus serà `error`.
- En tot cas, es cridaun cop més a `afegeixMissatge` per afegir un avís de tipus `info` indicant el valor introduït.

Com es pot apreciar, `$entrada` és l'objecte jQuery que conté el node corresponent al quadre de text. Per accedir al valor d'aquest (la propietat `value`) accedim cridant el mètode `val`, en aquest cas amb `$entrada.val()`.

> Hi ha moltes maneres d'afegir nodes amb jQuery, i fer servir una cadena de text és només una d'aquestes.

Tota la feina referida a afegir els nodes es troba dins de la funció `afegeixMissatge`. En primer lloc, es crea la cadena de text que conté el codi HTML que es vol afegir, en aquest cas és:

- Un paràgraf (`<p>`).
- Amb les classes `missatge` i la que correspongui al tipus (`error`, `exit` o `info`).
- I el missatge passat com a argument, amb el tipus en majúscules (`tipus.toUpperCase()`) i negreta (`<b>`).

En concret, la cadena composta quedaria així per a un missatge d'error: `<p class="missatge error"><b>ERROR: </b>S'ha d'introduir un nombre</p>`.

amb jQuery en el següent enllaç: api.jquery.com/append.

Una vegada creada la cadena, s'afegeix dins del node contingut en l'objecte jQuery `$avisos`, que correspon a l'element `div` amb `id="avisos"`. Per fer això només s'ha de cridar el mètode `append`, com es veu en l'exemple: `$avisos.append(definicioNode)`. En aquests exemples s'afegeixen sempre dos missatges; fixeu-vos que **no se sobreescriuen**, sinó que s'afegeix un darrere de l'altre.

Mètode "find"

El mètode `find` aplica un selector (com el de la funció `jQuery`), però en lloc de cercar a tot el document només cerca a partir de l'objecte a partir del qual es crida, per exemple: per cercar tots els paràgrafs dins d'un contenidor.

La primera acció que es fa en clicar el botó *Validar* i el botó *Netejar* és esborrar els avisos; vegem que passa quan es crida la funció `esborrarAvisos`:

- Es crida el mètode `find` al qual es passa el paràmetre `'p'` per seleccionar tots els paràgrafs que es trobin dins de l'element contingut a l'objecte guardat a `$avisos`, que en aquest cas és l'element `div` amb `id="avisos"`.
- Com que el retorn de `find` és un objecte jQuery, es pot continuar cridant mètodes de jQuery directament, així que es crida el mètode `remove`, que fa que s'eliminin tots els nodes seleccionats, en aquest cas tots els paràgrafs.

Convé subratllar que quan **un objecte jQuery** conté més d'un node i es crida qualsevol dels seus mètodes, els canvis produïts (o cerques) s'aplicaran a tots ells, per exemple en cridar els mètodes `css` o `remove`.

Separar el comportament en funcions com `esborrarAvisos` i `afegirAvis` proporciona molts avantatges, entre d'altres:

- S'evita la duplicació de codi, la qual cosa fa més fàcil el manteniment, ja que si voleu fer canvis a com s'esborren els elements només heu de fer els canvis en un lloc de l'aplicació.
- Encapsula la implementació de com es porten a terme els canvis, de manera que es pot modificar sense haver de tocar el codi de l'aplicació (penseu que una aplicació complexa podeu tenir aquest codi repartit entre diferents fitxers).

Vegeu ara alguns exemples alternatius de com afegir i eliminar nodes, relacionats amb aquests últims canvis. Començant amb un exemple més estructurat de com crear nodes i modificar-los abans d'afegir-los al document; substituïu el codi de la funció `afegirAvis` pel següent:

```
function afegirAvis(tipus, missatge) {
    var $ressaltat = $('<b></b>'),
      $avis = $('<p></p>');
```

```
  $avis.addClass('missatge');
  $avis.addClass(tipus);

  $avis.append($ressaltat)
  $avis.append(missatge);

  $avisos.append($avis);
}
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/bpVeZV.

> El mètode `append` accepta com a paràmetres cadenes de text, nodes i objectes jQuery.

El codi ara és molt més llarg. Fixeu-vos en el que s'ha fet, pas per pas:

1. S'han creat dos objectes jQuery, un que conté un node de tipus `b` (ressaltat) i un altre de tipus `p`, paràgraf. El codi que es passa com a paràmetres és HTML.
2. Per crear l'etiqueta ressaltada amb el tipus del missatge s'ha fet servir el mètode `text`, que afegeix com a text dins de l'element seleccionat el que s'ha passat per paràmetre; en aquest cas, el tipus concatenant-li el símbol ':' i un espai en blanc.
3. A l'objecte que conté l'avís s'hi ha afegit primerament la classe `missatge` i a continuació la classe corresponent al tipus d'avís amb el mètode `addClass`.
4. A continuació s'ha afegit l'objecte jQuery amb el ressaltat (`$ressaltat`) al node que contindrà tot l'avís (`$avis`) fent servir el mètode `append`.
5. Es crida un altre cop el mètode `append` per afegir a continuació el missatge, que és una cadena de text, i s'afegeix com a text pla.
6. Finalment, s'afegeix al contenidor d'avisos emmagatzemat a la variable `$avisos` l'objecte jQuery amb les classes, el ressaltat i el missatge afegits.

A primera vista pot semblar massa complicat fer-ho així, ja que amb el codi anterior només s'havia de passar una cadena de text, però serveix per adonar-se de la potència de jQuery per crear qualsevol estructura que necessiteu afegint nous elements, continguts i classes abans d'afegir-lo al document.

El següent exemple és molt més simple: es modificarà la funció `esborrarAvisos` per buidar el contingut del node contingut a l'objecte `$avisos` en lloc d'eliminar tots els seus fills un per un; substituïu el codi de la funció `esborrarAvisos` de l'exemple anterior per aquesta:

```
function esborrarAvisos() {
  $avisos.empty();
}
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/dMYpaP.

Com es pot apreciar, en aquest exemple s'ha fet servir un mètode nou, el mètode `empty`. Aquest no accepta cap argument i s'encarrega d'eliminar tots els descendents dels elements continguts a l'objecte jQuery. És més simple que l'anterior i més efectiu,

DISSENY D'INTERFÍCIES WEB

2.1.4. Elements interactius avançats: biblioteques i connectors

La biblioteca jQuery és molt popular, i per aquesta raó es poden trobar biblioteques i altres tipus de connectors basats en jQuery que el requereixen per al seu funcionament i que augmenten les seves funcionalitats, per exemple afegint nous mètodes als objectes jQuery retornats per la funció.

Documentació jQuery UI

Si voleu obtenir més informació sobre aquesta biblioteca i descobrir totes les seves opcions podeu visitar la pàgina oficial en el següent enllaç: jqueryui.com, on també podeu trobar el codi per descarregar, personalitzar la descàrrega i veure els tutorials que ofereixen per als diferents components.

Una d'aquestes biblioteques és jQuery UI, que inclou nous elements per millorar les interfícies d'usuari.

Afegir un connector o una biblioteca és molt fàcil, s'han de carregar dos fitxers:

- El full d'estils del connector: aquest conté tots els estils necessaris; en connectors molt simples pot ser que no existeixi.
- El fitxer amb el codi de la biblioteca o connector, igual que es fa amb la biblioteca jQuery o amb qualsevol altre fitxer amb codi JavaScript.

> A l'hora de carregar recursos externs sempre s'han de carregar primer els fulls d'estil, a continuació les biblioteques JavaScript i finalment la resta de fitxers amb codi JavaScript, encara que és recomanable afegir tant les biblioteques com el codi propi al final del codi HTML abans del tancament de l'element `body`.

Vegeu primer com afegir la biblioteca jQuery UI al vostre codi fent servir una CDN (encara que això no és necessari a CodePen perquè es pot fer la carrega des de les opcions):

```
<head>
  <link rel="stylesheet" href="//code.jquery.com/ui/1.11.2/themes/smoothne
  <script src="//code.jquery.com/jquery-3.1.1.js"></script>
  <script src="//code.jquery.com/ui/1.11.2/jquery-ui.js"></script>
  <script>
    $(document).ready(function() {
      $("input[type=submit], a, button")
      .button()
      .on('click', function(e) {
        e.preventDefault();
      });
    });
  </script>
</head>
```

```
  <input type="submit" value="Un 'element' input de tipus 'submit'"/>

  <a href="#">Un enllaç</a>
</body>
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/oxjBMM.

Si executeu el següent codi veureu que s'ha afegit dins de la capçalera de la pàgina:

- La càrrega del fitxer amb els fulls d'estil de la biblioteca fent servir el CDN de jQuery: `jquery-ui.css`.
- La càrrega de les biblioteques jQuery (sempre primer) i jQuery UI, també a partir del CDN.
- El codi JavaScript que comentarem a continuació.

> És molt habitual representar els enllaços com botons, són molt més fàcils de modificar amb CSS i es poden afegir els *handlers* que necessiteu.

El contingut del bloc HTML és un botó, un element `input` de tipus `submit` i un enllaç, però la seva representació és idèntica per a tots tres i no s'ha fet servir cap etiqueta de classe ni codi CSS.

Encara que potser no sembla gran cosa, aquest és un canvi important. L'aspecte dels botons no està controlat completament per CSS, sinó pel navegador, i cada fabricant ha fet la seva pròpia implementació. Això provoca que l'aspecte de botons per defecte no sigui homogeni entre navegadors i és molta feina fer els canvis pel vostre compte perquè s'han de tenir en compte molts aspectes.

Vegeu què fa el bloc de codi JavaScript: fins que el document no ha carregat completament no s'executa res, per això tota la implementació s'afegeix dins de la funció que és cridada pel mètode `ready` de jQuery, que una vegada disparada crea un objecte jQuery amb una selecció de botons, enllaços i els elements `input` de tipus submit: `$("input[type=submit], a, button").`

Com que jQuery presenta una **interfície fluida**, permet enllaçar crides a diferents mètodes una darrere l'altra; per veure-ho més clar s'acostuma a posar cada crida en una línia diferent.

> En una **interfície fluida** els mètodes retornen el mateix objecte que fa la crida, de manera que poden encadenar-se crides a diferents mètodes una darrere l'altra. Podeu trobar més informació sobre les interfícies fluides en el següent enllaç: |en.wikipedia.org/wiki/Fluent_interface.

Seguidament, es crida el mètode `button`, que és un dels mètodes afegits per la biblio-

> Cridar el mètode `preventDefault` és molt habitual en el cas dels botons i enllaços per poder afegir-los una funcionalitat diferent a la habitual.

A continuació s'afegeix el mètode `on` per escoltar l'*event* `click` com s'ha vist en exemples anteriors, però dins de la funció l'únic que es fa és cridar `e.preventDefault()`, sent `e` l'objecte que conté tota la informació de l'*event* i `preventDefault` el mètode que atura el comportament per defecte, de manera que en clicar a qualsevol dels tres elements no es fa cap acció.

Si inspeccioneu el codi HTML del primer botó amb les eines de desenvolupador (o qualsevol d'ells) veureu que s'han afegit una sèrie de classes i atributs que no hi són en el nostre codi HTML; per exemple, el contingut de l'element `button` modificat:

```
<button class="ui-button ui-widget ui-state-default ui-corner-all ui-butto
  <span class="ui-button-text">Un element 'button'</span>
</button>
```

En cridar el mètode `button`, aquest s'ha encarregat d'afegir totes les classes CSS (que pertanyen al full d'estils de jQuery UI) i elements necessaris per donar aquest aspecte.

Aquesta biblioteca ofereix una gran quantitat d'opcions, dividides en quatre grups:

- **Interaccions**: interaccions basades en el comportament del ratolí, com arrossegar, deixar anar o canviar la mida d'un element.
- ***Widgets***: controls que es reemplacen o afegeixen noves opcions a les interfícies d'usuari, com són els botons, les barres de progrés, les pestanyes, etc.
- **Efectes**: mètodes que modifiquen els aspectes i les posicions dels elements, incloent-hi els que afegeixen i eliminen classes.
- **Utilitats**: usades per jQuery UI per a la creació d'interaccions i *widgets*.

> Podeu trobar una llista amb milers de connectors per a la biblioteca jQuery en el següent enllaç: plugins.jquery.com.

En algunes ocasions necessitareu incorporar funcionalitats complexes molt determinades, per exemple per mostrar una galeria fotogràfica o afegir un reproductor de vídeo. En aquests casos, el que heu d'afegir és un **connector** (programa en JavaScript amb una funcionalitat molt concreta) especialitat per resoldre aquest tipus de problemes.

A diferència de les biblioteques, els connectors només inclouen una quantitat limitada d'opcions de configuració, de manera que són molt més simples d'utilitzar i molt més lleugers.

2.1.5. Desenvolupament de connectors per a jQuery

sar convertir-lo en un connector per facilitar la seva reutilització i manteniment.

Creareu un connector anomenat `arrodonir` que afegirà vores arrodonides a tots els elements als quals s'apliqui. Feu una primera prova per veure el comportament que es vol recrear:

```
<style>
  div {
    width: 100px;
    height: 100px;
    background-color: blue;
  }
</style>
<script>
  $(document).ready(function() {
    $('div').css('border-radius', '50px');
  });
</script>
<div></div>
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/GZpOMP.

S'ha d'esperar que l'estructura del document estigui correctament carregada, se seleccionen tots els elements `div` i se'ls aplica l'estil `border-radius` amb un valor de 50 píxels.

Convertir aquesta funcionalitat en un connector és molt fàcil, només s'ha d'afegir una funció anomenada `arrodonir` a `$.fn`, i a partir d'aquest moment aquesta funció passarà a estar disponible a tots els objectes jQuery. Reemplaceu el codi JavaScript pel següent:

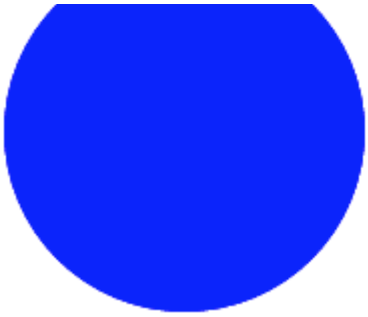
```
$.fn.arrodonir = function() {
  this.css('border-radius', '50px');
}

$(document).ready(function() {
  $('div').arrodonir();
});
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/mPeqXa.

Ara, quan es crida el mètode `arrodonir` als elements seleccionats per l'objecte jQuery, en aquest cas els elements `div`, s'aplicarà la propietat `border-radius` amb un valor de `50px`, com es pot veure a la figura 2.18.

Figura 2.18. Element al qual se li ha aplicat el connector creat

DISSENY D'INTERFÍCIES WEB



Vegeu un exemple una mica més elaborat. Es calcularà l'alçada i l'amplada, i s'ajustarà el radi de les vores, de manera que si són iguals s'obtindrà un cercle, i si no, les bandes més estretes formaran un semicercle:

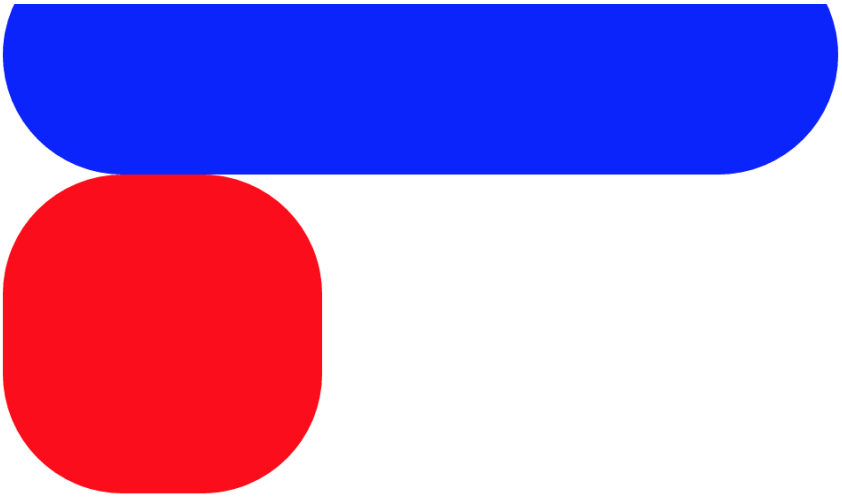
```
<style>
  div {
    width: 150px;
    height: 120px;
    background-color: blue;
  }

  div+div {
    width: 250px;
    height: 150px;
    background-color: red;
  }
</style>
<script>
  $.fn.arrodonir = function() {
  var height = this.height(),
      width = this.width(),
      radi = Math.min(height,width)/2;

  this.css('border-radius', radi + 'px');
}

$(document).ready(function() {
  $('div').arrodonir();
});
</script>

<div></div>
<div></div>
```

> Si proveu a canviar la mida dels elements mitjançant el codi CSS a CodePen tingueu en compte que l'arrodoniment de les vores pot ser que no s'actualitzi correctament; això és una limitació de CodePen i no el comportament normal.

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/vGNWjV.

Fixeu-vos en aquest exemple que si la mida dels elements són diferents, els càlculs seran només correctes per al primer dels elements processats, encara que s'aplicarà la vora arrodonida a tots ells però amb el mateix valor, com es pot apreciar a la figura 2.19.

Figura 2.19. El mateix connector aplicat a múltiples elements



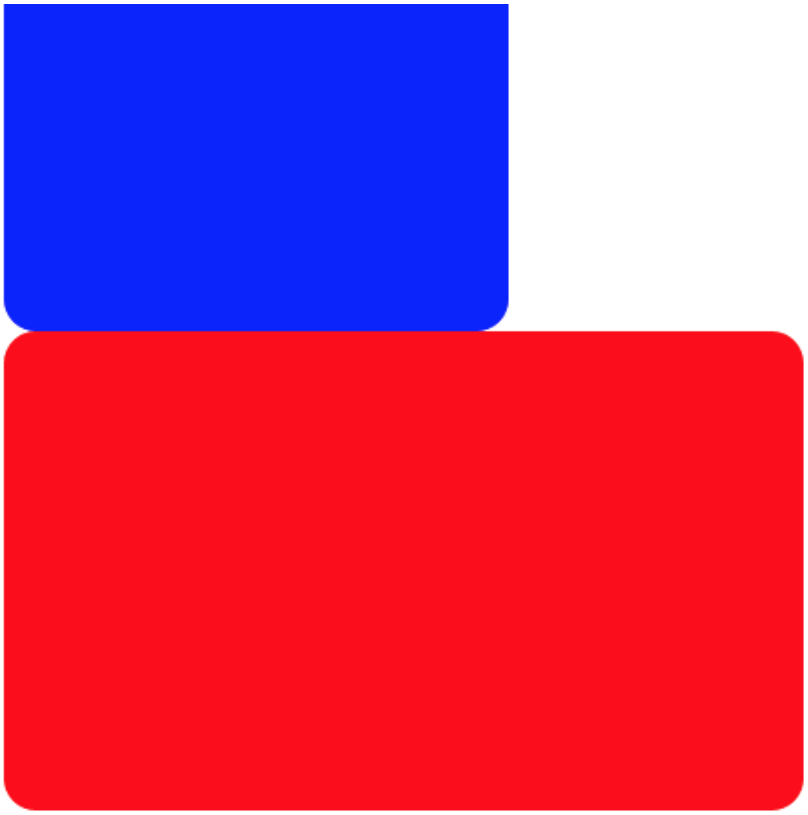
Vegeu un últim exemple: com ho faríeu si necessiteu poder passar arguments a aquest mètode? Molt fàcil, de la mateixa manera que a qualsevol altre mètode o funció. Substituïu el codi del bloc JavaScript pel següent:

```
$.fn.arrodonir = function(radi) {
  var height, width, radi;

  if (!radi) {
    height = this.height();
    width = this.width();
    radi = Math.min(height, width) / 2;
  }

  this.css('border-radius', radi + 'px');
}

$(document).ready(function() {
  $('div').arrodonir(5);
});
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/ZWbame.

Simplement s'ha afegit un argument al mètode `arrodonir` anomenat `radi`, i dins del mètode es comprova si aquest **no està definit** (no s'ha passat cap valor en cridar el mètode) amb `if (!radi) {}`. Si no ho està, es fan els càlculs com a l'exemple anterior, i en cas que sí que estigui definit, es fa servir el valor per establir el radi. Si es passa el valor 5 (com a l'exemple) a la funció, el resultat seria el que es pot apreciar a la figura 2.20.

Figura 2.20. Elements modificats pel connector parametritzat



Només falta afegir-hi un petit detall. Per mantenir la interfície fluida de jQuery, el connector ha de retornar `this`: afegiu al final de la funció del connector `return this`. I per comprovar que realment es manté la interfície fluida, canvieu el codi de la funció cridada per `ready` per aquest:

```
$(document).ready(function() {
  $('div')
    .arrodonir(10)
    .css('background-color', 'green');
});
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/ONyQmO.

Les vores dels elements `div` seran arrodonides 10 píxels, i el fons de tots dos elements serà de color verd.

Ja teniu el vostre connector preparat per fer-lo servir; podeu guardar-lo com un fitxer independent i carregar-lo com si es tractés d'una altra biblioteca, **sempre després d'haver carregat jQuery**, i el mètode `arrodonir` estarà disponible per a tots els objectes jQuery.

2.1.6. Introducció a JSON

El llenguatge de marques **JSON** (abreviatura de JavaScript Object Notation) permet guardar una col·lecció de dades de manera que és fàcilment interpretable pels humans. Vegeu aquest exemple, que es tracta d'un objecte de JavaScript creat a partir de la declaració literal:

```
   cognoms:  Palau Torres ,
  "dni": "12345689X"
}
```

Fixeu-vos ara en com és aquest objecte en format JSON:

```
{
  "nom": "Pere",
  "cognoms: "Palau Torres",
  "dni": "12345689X"
}
```

Origen de JSON

El llenguatge JSON va ser popularitzat i convertit en un estàndard per Douglas Crockford. Podeu trobar més informació sobre JSON i el seu promotor en el següent enllaç: en.wikipedia.org/wiki/JSON.

Són idèntics. Això és perquè el format d'intercanvi de dades JSON es basa en la declaració literal dels objectes JavaScript i, per tant, permet estructurar les dades de la mateixa manera. Per exemple, afegint *arrays*:

```
{
  "nom": "Pere",
  "cognoms: "Palau Torres",
  "dni": "12345689X"
  "assignatures":[
    "Programació bloc 1",
    "Bases de dades bloc 1",
    "Entorns de desenvolupament"
  ]
}
```

O fins i tot altres objectes amb notació literal:

```
{
  "periode": "2016/2",
  "alumnes": [
    {
      "nom": "Pere",
      "cognoms: "Palau Torres",
      "dni": "12345689X"
      "assignatures":[
        "Programació bloc 1",
        "Bases de dades bloc 1",
        "Entorns de desenvolupament"
      ]
    },
    {
      "nom": "Lluís",
      "cognoms: "Martorell Ferrer",
      "dni": "223353389D"
      "assignatures":[
        "Programació bloc 2",
        "Bases de dades bloc 2"
      ]
    }
  ]
```

El **format JSON** només treballa amb les propietats de l'objecte; en

DISSENY D'INTERFÍCIES WEB

Si ho compareu amb altres llenguatges de marcat, com per exemple XML, es pot apreciar que un dels avantatges és que aquest és molt més simple i entenedor. Fixeu-vos com és la representació de l'exemple anterior amb format XML:

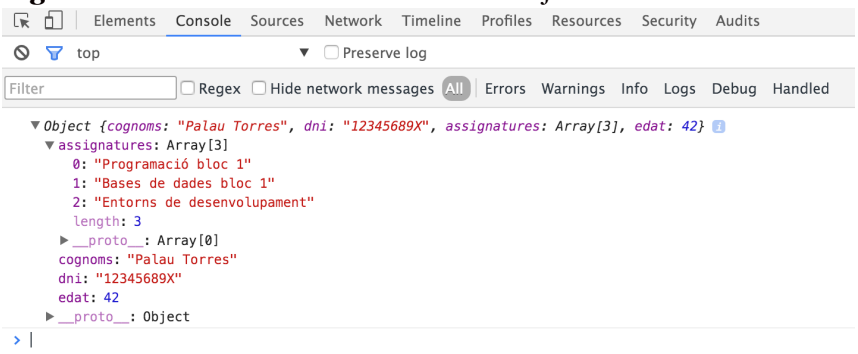
```
<institut>
  <cursos>
    <curs>
      <periode>2016/2</periode>
      <alumnes>
        <alumne>
          <nom>Pere</nom>
          <cognoms>Palau Torres</cognoms>
          <dni>12345689X</dni>
          <assignatures>
            <assignatura>
              <nom>Programació bloc 1</nom>
            </assignatura>
            <assignatura>
              <nom>Bases de dades bloc 1</nom>
            </assignatura>
            <assignatura>
              <nom>Entorns de desenvolupament</nom>
            </assignatura>
          </assignatures>
        </alumne>
        <alumne>
          <nom>Lluis</nom>
          ...
        </alumne>
      </alumnes>
    </curs>
  </cursos>
</institut>
```

Com podeu apreciar, en l'exemple anterior només s'ha afegit un alumne, però la quantitat de codi necessari ha estat molt més gran que per fer-ho en JSON i a més és més feixuc llegir la informació.

Un altre avantatge que presenta treballar amb JSON a JavaScript és que en tractar-se d'objectes nadius del llenguatge és molt fàcil afegir, modificar o eliminar dades, com es pot veure en el següent exemple:

```
// Es crea l'objecte alumne, que conté l'estructura de dades
var alumne = {
  "nom": "Pere",
  "cognoms": "Palau Torres",
  "dni": "12345689X",
  "assignatures":[
    "Programació bloc 1",
    "Bases de dades bloc 1",
    "Entorns de desenvolupament"
  ]
};

// S'afegeix una nova propietat i li assignem un valor
alumne.edat = 24;

// Es modifica aquest valor
alumne.edat = 42;

// Finalment, s'elimina la propietat nom amb l'operador delete de JavaScri
delete alumne.nom

// Podeu trobar el contingut final a la consola
console.log(alumne);
```

Figura 2.21. Visualització a la consola de l'objecte alumne



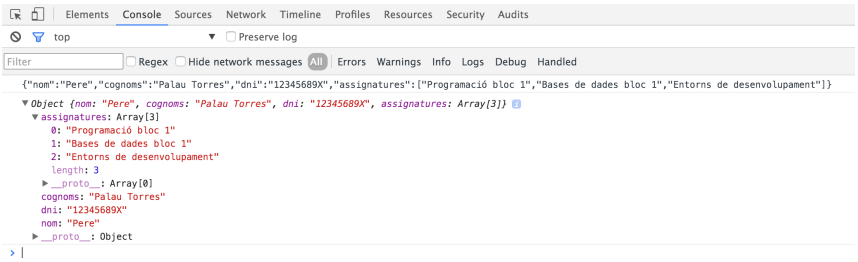
Com podeu veure, el valor final per a la propietat `edat` és 42, i no existeix la propietat `nom` perquè l'hem eliminat.

Ara bé, encara que en aquest apartat no es treballa amb l'enviament ni la recepció de dades, és necessari saber que si voleu enviar aquest objecte a través de la xarxa, per exemple, fent servir un formulari, s'ha de convertir aquest en una cadena de text. I al contrari, pot ser que rebeu un objecte JSON com una cadena de text des del servidor i l'hàgiu de convertir en un objecte.

```
// Es crea l'objecte alumne, que conté l'estructura de dades
var alumne = {
  "nom": "Pere",
  "cognoms": "Palau Torres",
  "dni": "12345689X",
  "assignatures": [
    "Programació bloc 1",
    "Bases de dades bloc 1",
    "Entorns de desenvolupament"
  ]
};

// Conversió de l'objecte a cadena de text
var alumneAText = JSON.stringify(alumne);
console.log(alumneAText);

// Conversió  la cadena de text a objecte
var textAAlumne = JSON.parse(alumneAText);
console.log(textAAlumne);
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/dMdwMO, i el resultat a la figura 2.22.

Figura 2.22. Visualització de l'objecte a la consola com a objecte i com a cadena de text



Com heu vist, JavaScript ens proporciona l'objecte global `JSON`, que exposa dos mètodes:

- `stringify`: converteix un objecte en una cadena de text en format JSON.

Amb aquests dos mètodes, i tenint clar com afegir, modificar i eliminar propietats d'un objecte, no heu de tenir cap problema per utilitzar-los com a estructures de dades en les vostres aplicacions.

2.1.7. Execució i verificació

Un dels majors problemes que us trobareu a l'hora de verificar el funcionament d'una aplicació web és la fragmentació del nombre de dispositius en el que s'ha de visualitzar.

Fins fa uns anys, la visualització de pàgines web estava lligada als equips d'escriptori, i per tant només calia preocupar-se de comprovar les pàgines i aplicacions web en un nombre limitat de navegadors. Tot i així, representava força dificultats, perquè sovint en un mateix equip no es podien instal·lar diferents versions d'un mateix navegador, o fins i tot els navegadors no eren compatibles amb diferents versions del mateix sistema operatiu.

Amb la proliferació dels telèfons intel·ligents, el problema ha empitjorat. No tan sols s'ha de comprovar en una combinació major de navegadors i sistemes operatius, sinó que és possible trobar casos en què fins i tot amb el mateix programari els dispositius poden tenir tipus de pantalles i sensors molt diferents, ja que tant pot tractar-se d'un dispositiu mòbil amb pantalla de 3" com d'una televisió intel·ligent amb una pantalla de 50".

> Podeu trobar més informació sobre com muntar un laboratori de dispositius en el següent enllaç: goo.gl/tt8hcG.

Hi ha dues possibles solucions a aquest problema: la creació d'un **laboratori de dispositius** o **la simulació**. Cap de les dues és exhaustiva, i només permeten obtenir una certa seguretat que la nostra aplicació es visualitzarà correctament en determinades combinacions.

> Podeu trobar més informació sobre la sincronització multidispositiu en el següent enllaç: goo.gl/zkHpW1.

La creació d'un laboratori de dispositius no està a l'abast de tothom, ja que suposa una gran inversió en maquinari (s'han d'adquirir tots els dispositius per testejar) i, opcionalment, en programari. Existeixen alternatives gratuïtes i de pagament, tant per muntar la infraestructura per sincronitzar els dispositius com per al programari que els gestiona.

La simulació, en canvi, és més assequible, però té el desavantatge de no tractar directament amb dispositius reals. Quan parlem de simulació tenim opcions molt diferents:

- **Serveis remots de proves**: aquests serveis generalment són de pagament i ens permeten testejar la nostra web en una quantitat molt gran de dispositius. Per exemple, www.browserstack.com permet fer proves en més de 700 navegadors

amb diferents configuracions de sistema operatiu i navegadors. A banda de ser poc pràctic a l'hora de realitzar les proves, estem molt limitats quant a les possibilitats, ja que no tots els equips poden simular tots els sistemes operatius i navegadors. Per exemple, per emular el sistema operatiu macOS el vostre equip ha de tenir un processador Intel i3 o superior; per emular iOS cal fer servir macOS, i per emular Android de manera efectiva cal que el vostre equip faci servir processadors Intel, ja que són els únics que estan optimitzats per funcionar amb l'emulador proporcionat per Google (en el moment de redacció d'aquests materials).

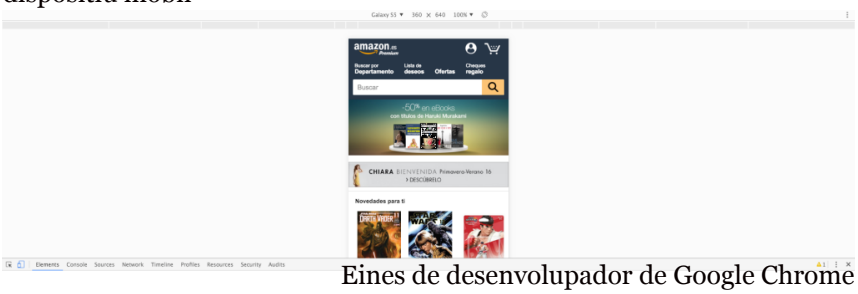
- **Eines de desenvolupador**: les eines de desenvolupador dels diferents navegadors permeten simular la visualització en diferents mides; encara que no és ni de bon tros el mateix que simular tot un sistema operatiu ni els navegadors, sovint és suficient per cobrir la major part de les plataformes que us poden interessar.

Un dels objectius que s'han de plantejar en tots els projectes és que la pàgina o aplicació es visualitzi correctament quan es visiti a través de dispositius mòbils, fet que suposa comprovar-la com a mínim en els següents formats:

- dispositiu mòbil vertical
- tauleta vertical
- tauleta apaïsada
- escriptori

Afortunadament, aquestes quatre combinacions es poden comprovar fàcilment a través de les eines de desenvolupador, com es pot veure a la figura 2.23.

Figura 2.23. Visualització d'una pàgina simulant les dimensions d'un dispositiu mòbil



Eines de desenvolupador de Google Chrome

Si us fixeu en la figura 2.23 veureu que a la cantonada inferior dreta hi ha una icona que representa un mòbil i una tauleta; clicant sobre aquesta icona es passa al mode de disseny responsiu i podem seleccionar la visualització de la pàgina com si fossin diferents dispositius i resolucions.

A banda de **comprovar les resolucions**, heu d'assegurar-nos que la nostra aplicació funciona correctament com a mínim en els navegadors d'escriptori més utilitzats, que en el moment de redactar aquests materials són: Google Chrome, Mozilla Firefox, Safari, Opera i Internet Explorer.

En tractar amb elements multimèdia heu de tenir molt en compte que els dos principals problemes que us trobareu serà la incompatibilitat de formats de vídeo i àudio; per

- **Àudio**: format MP3 i OGG.
- **Vídeo**: format MP4 i WebM.

Per pal·liar les limitacions que us trobareu en treballar amb navegadors antics disposeu de les següents opcions:

- **HTML**: entre els elements afegits a HTML5 hi ha alguns que només tenen funció semàntica, com per exemple `aside` o `article`, que es poden fer servir indistintament en navegadors antics (ja que es tracta de codi XML vàlid), i d'altres com `audio` i `video`, que inclouen la seva pròpia API. Aquests últims no poden funcionar en navegadors antics, i l'única solució actual és mostrar un missatge si no és possible utilitzar l'etiqueta:

```
<video>
  <source src="fitxer_video.mp4" type="video/mp4">
  <source src="fitxer_video.webm" type="video/webm">
  Ho sento, el teu navegador no suporta vídeo d'HTML5
</video>
```

- **CSS**: en el cas dels fulls d'estil, la solució és afegir una propietat que funcioni universalment i a continuació afegir les funcions més avançades, de manera que s'aplicarà l'última acceptable. Per exemple, en el cas de voler afegir un degradat a un fons podeu fer servir la propietat `background-color` per afegir un color pla, i a continuació el codi de degradat propi dels diferents navegadors, acabant amb el de l'especificació de CSS3:

```
body {
    background: #f0f9ff; /* Navegadors antics, color pla */
    background: -moz-linear-gradient(-45deg,  #f0f9ff 0%, #cbebff 47%, #a1
    background: -webkit-linear-gradient(-45deg,  #f0f9ff 0%,#cbebff 47%,#a
    background: linear-gradient(135deg,  #f0f9ff 0%,#cbebff 47%,#a1dbff 10
    filter: progid:DXImageTransform.Microsoft.gradient( startColorstr='#f0
}
```

> Podeu trobar més informació sobre *polyfill* en el següent enllaç: goo.gl/h9NAhH.

- **JavaScript**: en el cas de JavaScript teniu dues opcions, la primera és implementar les funcions no disponibles en navegadors antics com a *polyfill*, de manera que si no són detectades les hi afegiu. S'ha de tenir en compte que aquesta solució sempre és més lenta que fer servir les funcions nadiues dels navegadors que sí les suportin. L'altra opció és fer servir les funcionalitats de biblioteques que ja implementin solucions per a navegadors antics, com jQuery amb les versions 1.x. Per exemple, per donar suport a Internet Explorer 6 i anteriors a l'hora d'enviar peticions AJAX al servidor s'havia de fer la següent comprovació:

```
if (window.XMLHttpRequest) { // Mozilla, Safari, IE7+ ...
    httpRequest = new XMLHttpRequest();
} else if (window.ActiveXObject) { // IE 6 i anteriors
    httpRequest = new ActiveXObject("Microsoft.XMLHTTP");
}
```

En canvi, si les peticions AJAX es fan a través de jQuery, no us heu de preocupar d'aquestes variacions, ja que ho gestiona la biblioteca.

DISSENY D'INTERFÍCIES WEB

Quan feu servir **biblioteques com Query** es recomanable emprar sempre les funcions i els mètodes que us proporcionen en lloc d'utilitzar JavaScript, perquè la biblioteca us assegura que el comportament estarà homogeneïtzat entre diferents navegadors i versions. Tenint en compte que jQuery és la biblioteca de JavaScript més utilitzada i que ofereix una gran quantitat de funcionalitats, és recomanable que l'estudieu pel vostre compte més enllà del que cobreixen aquests materials.

2.2. Casos pràctics d'integració de continguts multimèdia

Gràcies als nous elements afegits amb HTML5 (`audio` i `video`) és possible crear reproductors personalitzats fent servir només les tecnologies proporcionades pels navegadors, afegint llistes de reproducció que carreguin la informació de fonts externes com per exemple un servidor de vídeo extern o un fitxer de dades.

Un altre ús molt freqüent dels recursos multimèdia és la creació de bàners publicitaris, ja que aprofitant les característiques del llenguatge, en lloc de fer servir bàners estàtics amb codi HTML, es poden crear bàners dinàmics que mostrin un anunci o un altre segons les dades que s'hagin carregat (o incrustat).

En particular, els dos dels llenguatges de marcat més populars per transmetre aquesta informació són XML i JSON (JavaScript Object Notation). Un avantatge d'aquest últim és que es tracta del mateix format que fa servir JavaScript i per tant una estructura de dades en JavaScript és idèntica a la seva representació en JSON.

2.2.1. Creació d'un reproductor de vídeo

En primer lloc, es desenvoluparà un reproductor de vídeo, pas a pas, al qual s'integraran efectes de so i animacions amb CSS. Les característiques d'aquest reproductor seran les següents:

- A la secció esquerra mostrarà una llista generada dinàmicament a partir d'una estructura de dades JSON (en el nostre cas, un objecte literal de JavaScript)
- A la secció dreta mostrarà una caixa on es reproduirà el vídeo i a sota els botons de reproducció.
- En passar el cursor sobre qualsevol botó o element de la llista es reproduirà un so i s'executarà una petita animació.
- En fer clic sobre un element de la llista es reproduirà un so i començarà a reproduir-se el vídeo associat.
- En fer clic sobre un botó es reproduirà un so i es realitzarà una acció diferent, segons el botó clicat:
    - **Anterior**: es desplaçarà el vídeo seleccionat una posició cap enrere, de manera que si era seleccionat el primer vídeo, passarà a seleccionar-se l'últim, i començarà la reproducció.

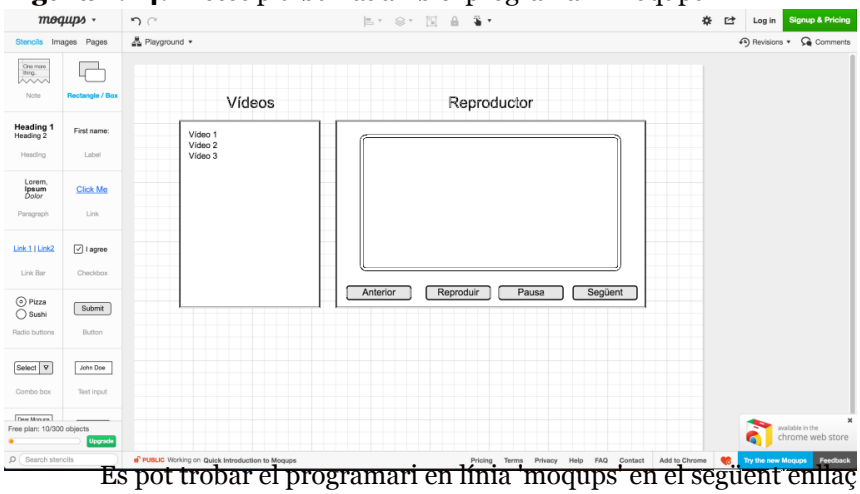
ducció.

- **Reproduir**: inicia la reproducció del vídeo seleccionat des del començament, i, si ja s'estava reproduint, tornarà a començar.
- **Aturar**: si el vídeo estava reproduint-se l'atura, i, si estava aturat, continua reproduint des del mateix punt.

Prototip de la interfície del reproductor

El primer que s'ha de fer abans de començar a codificar la solució és fer un prototip de la interfície. D'aquesta manera, a l'hora de portar a terme la implementació tindreu clar com ha de funcionar l'aplicació. Aquest prototip el podeu dissenyar directament sobre paper o fer servir alguna eina de *wireframes* o *mockups* com la que es pot veure a la figura 2.24. Es recomana fer servir programari específic per a la creació d'aquests, ja que ofereixen elements neutres per crear les interfícies que no distreuen del seu objectiu.

Figura 2.24. Prototip dibuixat amb el programari 'moqups'



Es pot trobar el programari en línia 'moqups' en el següent enllaç (https://moqups.com/).

Una vegada tingueu clara la distribució de la interfície de la vostra aplicació podeu començar la següent fase de preparació.

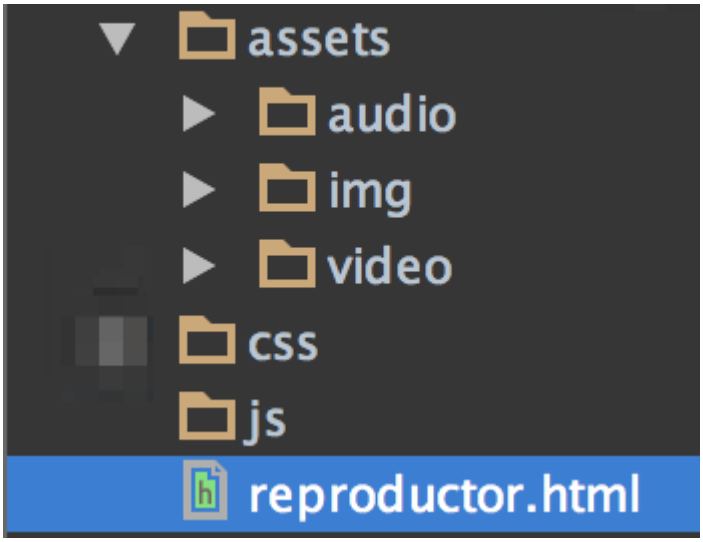
Estructura de directoris i preparació de recursos

El següent pas és crear la estructura de directoris, preparar els recursos multimèdia necessaris i copiar-los als directoris pertinents.

És possible que l'estructura de directoris us vingui donada per les tecnologies que empreu; en aquest cas, fareu servir una estructura pròpia, com es pot veure a la figura 2.25.

Figura 2.25. Estructura de fitxers i directoris del reproductor

DISSENY D'INTERFÍCIES WEB



- El fitxer HTML es trobarà a l'arrel del projecte.
- Els fitxers CSS es trobaran dins del directori */css*.
- Els fitxers amb codi JavaScript aniran dins del directori */js*.
- Els fitxers de recursos multimèdia es trobaran dins del directori */assets*, i dins d'aquests:
  - Els fitxers de vídeo dins de la carpeta */video*.
  - Els fitxers d'àudio dins de la carpeta */audio*.
  - Els fitxers d'imatge dins de la carpeta */img* (encara que en aquest projecte no es farà servir cap).

Pas 1: preparació de l'estructura de la pàgina HTML

Arribats a aquest punt, ja podeu començar a codificar. Primer haureu de crear el fitxer HTML sense aplicar cap estil ni cap identificador, això ho fareu en el següent pas. Només heu de fixar-vos en el disseny del prototip i pensar com ha d'estar estructurat el codi.

Temporalment, es farà servir una llista (amb els elements `ul` i `li`) per representar els vídeos que poden seleccionar-se, ja que no s'afegirà la creació dinàmica fins més endavant.

Aquesta seria una possible implementació:

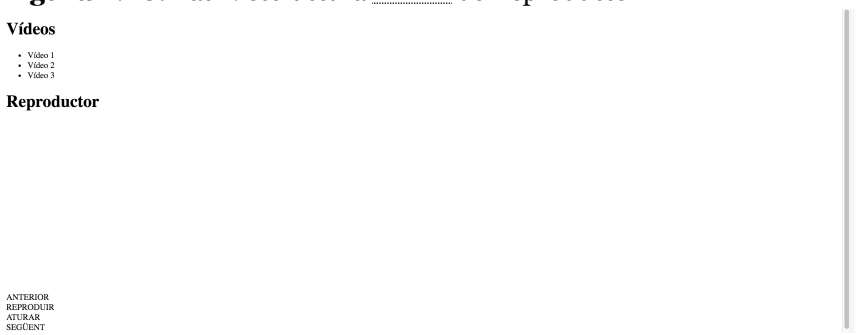
```
<!DOCTYPE html>
<html>
<head>
    <title>Reproductor</title>
</head>
<body>
<head>
  <title>Reproductor</title>
</head>

<body>
  <main>
    <div>
      <h1>Vídeos</h1>
      <ul>
        <li>Vídeo 1</li>
        <li>Vídeo 2</li>
        <li>Vídeo 3</li>
```

```
        <hl>Reproductor</hl>
        <div>
          <video width="430px" height="315px" type="video/mp4"></video>
        </div>
        <div>
          <div>ANTERIOR</div>
          <div>REPRODUIR</div>
          <div>ATURAR</div>
          <div>SEGÜENT</div>
        </div>
      </div>
    </main>
  </body>
  </html>
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/GZmpWa.

I el resultat obtingut seria similar al que es pot veure a la figura 2.26.

Figura 2.26. Pas 1: estructura HTML del reproductor



Pas 2: afegir el full d'estil

Una vegada tingueu l'estructura llesta fareu servir el full d'estil per donar-li un format que s'ajusti al nostre prototip.

El primer que heu de fer és crear un fitxer de text pla anomenat reproductor.css, que guardareu dins del directori */css*. A continuació, l'enllaçareu amb el document HTML afegint el següent codi dins de l'element `head`:

```
<link rel="stylesheet" href="css/reproductor.css" type="text/css" />
```

> Recordeu que als **exemples de CodePen** no cal enllaçar els fitxers amb el codi CSS ni JavaScript, ja que aquest es troba a les diferents columnes. Per altra banda, tampoc no cal enllaçar la càrrega de la biblioteca jQuery, perquè està afegida a la configuració del Pen.

Seguidament, procediu a afegir els identificadors i classes a diferents seccions del codi HTML que us facilitarà l'assignació d'estils:

- identificador per a la llista
- identificador per al visor
- identificador per al reproductor

> Freqüentment, en treballar amb elements flotants, els elements que apareixen a continuació no es mostren correctament. Una de les solucions possibles és afegir un element `div` buit amb una classe que aplique l'estil CSS `clear:both`. Per convenció, a aquesta classe se l'anomena `clear` o `clearfix`.

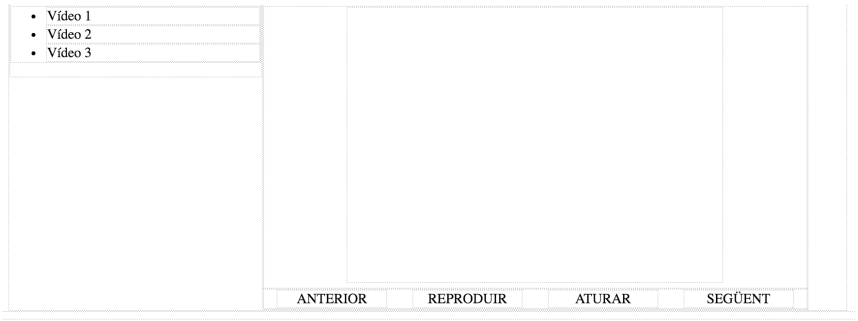
Ara només resta aplicar els estils apropiats:

```
* {
  border: 1px dotted lightgray;
}

main {
  max-width: 960px;
  margin: 0 auto;
}

#llista,
#visor {
  float: left;
}

#llista {
  width: 30%;
}

#visor {
  width: 65%;
  text-align: center;
}

#controls {
  display: flex;
  justify-content: space-around;
}

.boto {
  flex-basis: 20%;
}

.clearfix {
  clear: both;
  border: none;
}
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/oxWjWP.

A la figura 2.27 podeu veure el resultat obtingut, molt més semblant al prototip.

Figura 2.27. Pas 2: reproductor després d'afegir el codi CSS



Fixeu-vos que s'ha afegit una vora a tots els elements del document:

```
* {
   border: 1px dotted lightgray;
 }
```

Aquesta vora s'ha afegit de manera temporal per ressaltar les delimitacions de tots els elements, i no formarà part del disseny final.

Pas 3: afegir fonts amb Google Fonts

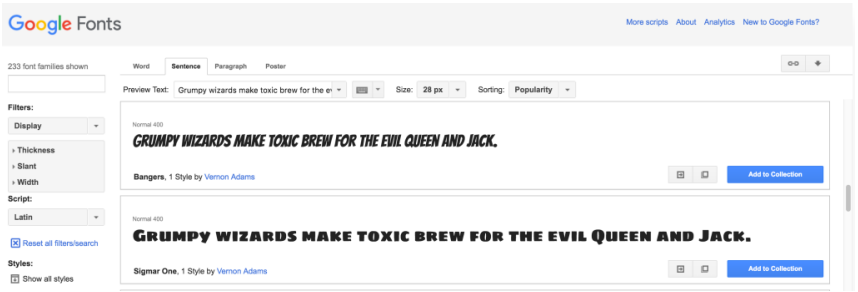
Una manera molt simple de fer els vostres títols i capçaleres més atractives és fer servir fonts externes, com les que ofereix Google Fonts.

> Les llicències de les fonts oferides per Google Fonts generalment són de tipus lliure o gratuïtes.

Per afegir alguna d'aquestes fonts heu de visitar la pàgina de Google Fonts (vegeu la figura 2.28), i a continuació:

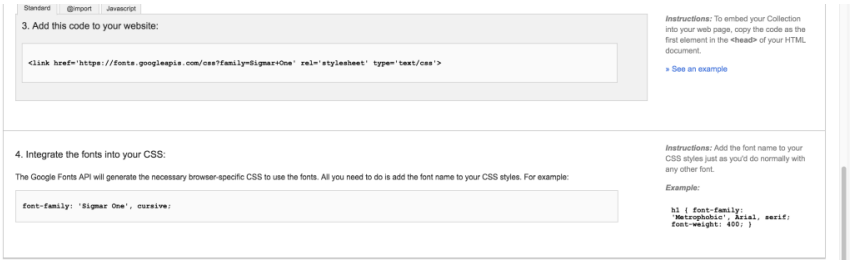
- Seleccioneu una de les fonts llistades, per exemple Sigmar One, de Vernon Adams.
- Afegiu-la a la col·lecció, fent clic al botó *Add to Collection*.

Figura 2.28. Selecció de la font de Google Fonts



- Feu clic al botó *use*, que us portarà a una altra pàgina on podreu veure més informació sobre la font, el seu pes i el codi per afegir tant al fitxer HTML i el codi CSS per fer-la servir, com es pot veure a la figura 2.29.

Figura 2.29. Visualització del codi HTML i CSS per utilitzar la font



Ara que ja teniu tota la informació que necessiteu, actualitzeu el vostre fitxer HTML amb el codi obtingut de la pàgina, i més a més afegiu el joc de caràcters per evitar problemes de representació:

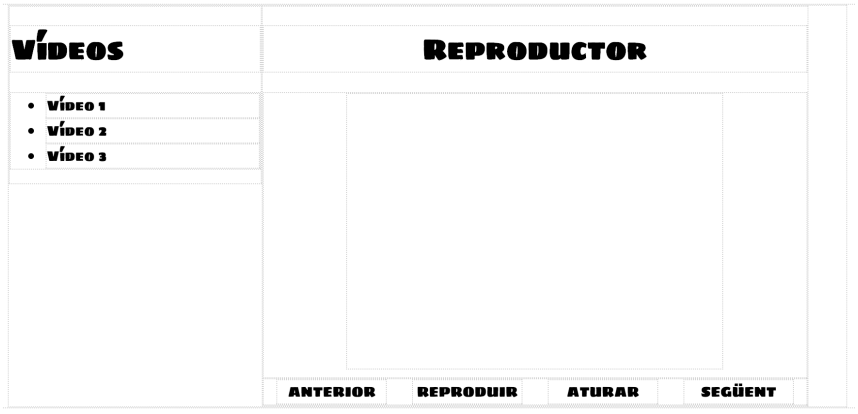
```
<head>
    <meta charset="utf-8">
    <title>Reproductor</title>
    <link href="css/reproductor.css" rel="stylesheet" type="text/css"/>
    <link href="https://fonts.googleapis.com/css?family=Sigmar+One" rel="s
</head>
```

> Per assegurar que tots els caràcters de la pàgina es mostren correctament s'ha d'afegir a la capçalera l'etiqueta: `<meta charset="utf-8">`.

També s'ha de modificar el fitxer CSS per fer servir aquesta font a tots els elements, ja que no s'inclourà cap altre tipus d'element de text a banda de les capçaleres i els botons, i s'obtindrà el resultat que es pot veure a la figura 2.30.

```
* {
    border: 1px dotted lightgray;
    font-family: 'Sigmar One', cursive;
}
```

Figura 2.30. Pas 3: reproductor amb la font canviada



Pas 4: afegir icones amb Font Awesome

Per fer més clara la utilitat dels botons afegireu icones de Font Awesome. Com que precisament inclou un joc d'icones per a reproductors, facilita molt la feina.

L'avantatge de fer servir icones en lloc d'imatges és que les fonts funcionen com a imat-

Podeu trobar més informació sobre com utilitzar les icones de Font Awesome en el següent enllaç: goo.gl/54lXEX.

El primer que heu de fer per poder emprar les icones de Font Awesome és enllaçar amb el fitxer que conté el codi CSS:

```
<link href="https://maxcdn.bootstrapcdn.com/font-awesome/4.5.0/css/font-aw
```

> Tant **Google Fonts** com **Font Awesome**, a banda del fitxer CSS, realitzaran peticions extres per accedir als fitxers de fonts.

Una vegada enllaçat, ja podeu afegir les icones al reproductor; en aquest cas, s'ha decidit fer servir les icones:

- `step-backward`
- `play`
- `pause`
- `step-forward`

Per afegir un botó directament al codi HTML només hem de cercar la icona que us interessi dins del lloc web de Font Awesome i fer-hi clic. Us portarà a una altra pàgina, on trobareu el codi HTML que heu de fer servir.

La icona `step-backward` es pot trobar en el següent enllaç: goo.gl/MGm4PD. Allà es pot trobar el codi corresponent, `class="fa fa-step-backward"></i>` (com es pot veure a la figura 2.31), i la visualització en diferents mides.

Figura 2.31. Informació completa d'una icona de Font Awesome

Per augmentar la mida d'una icona només heu de

per triplicar-la, etc.

Reemplaceu el codi dels controls pel següent, on s'han afegit les icones per al reproductor i un salt de línia per fer que el text quedi sempre en una nova línia:

```
<div id="controls">
  <div class="boto"><i class="fa fa-step-backward fa-3x"></i><br>ANTERIOR<
  <div class="boto"><i class="fa fa-play fa-3x "></i><br>REPRODUIR</div>
  <div class="boto"><i class="fa fa-pause fa-3x"></i><br>ATURAR</div>
  <div class="boto"><i class="fa fa-step-forward fa-3x"></i><br>SEGÜENT</d
</div>
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/eZWpxV i la seva visualització a la figura 2.32.

Figura 2.32. Pas 4: icones de Font Awesome afegides al reproductor

Pas 5: canviar l'estil dels botons

El següent pas consisteix en donar estil als botons per fer-los més atractius. S'ha decidit fer servir dos tons de blau: un de clar per a l'estat normal i un de més fosc quan el cursor sigui a sobre del botó, i aplicar cantonades arrodonides. Al fitxer CSS canvieu l'estil dels botons pel següent:

```
.boto {
  flex-basis: 20%;
  border-radius: 10px;
  border: 2px solid #105F85;
  padding: 5px;
  color: white;
  background-color: #2897E8;
  cursor: pointer;
}

.boto:hover {
  background-color: #105F85;
}
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/jqmWJv i la seva visualització a la figura 2.33.

Figura 2.33. Pas 5: Afegir estil als botons

Pas 6: canviar l'estil de la llista

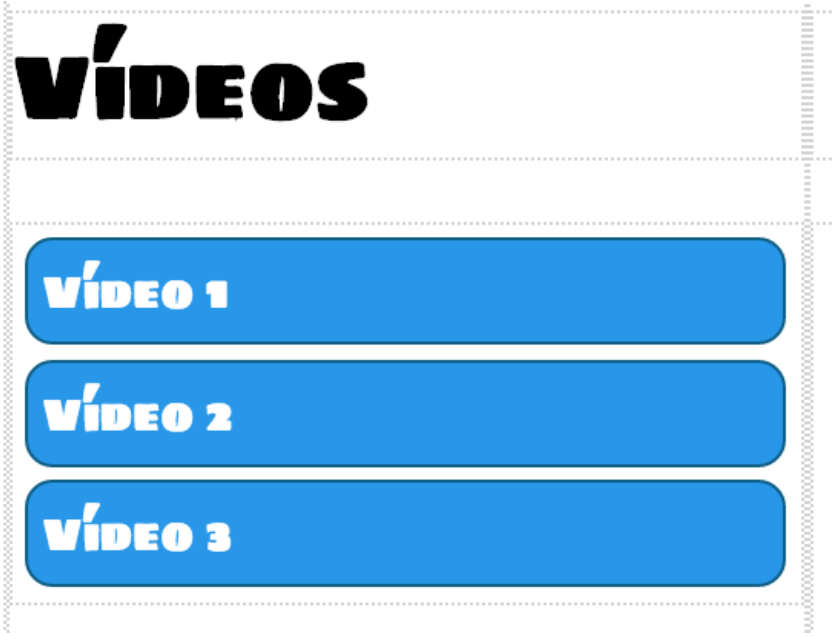
De manera semblant, modifiqueu l'estil de la llista aplicant el mateix tipus de vora, cantonades i colors:

```
#llista ul {
  padding: 0;
```

```
  list-style: none;
  border-radius: 10px;
  border: 2px solid #105F85;
  padding: 5px;
  color: white;
  background-color: #2897E8;
  cursor: pointer;
  margin: 5px;
}

#llista li:hover {
  background-color: #105F85;
}
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/eZWJwa i la seva visualització a la figura 2.34.

Figura 2.34. Pas 6: nou aspecte de la llista



Pas 7: refacció del full d'estil

> La refacció consisteix a reestructurar un codi per millorar-lo sense modificar el seu comportament extern.

Us haureu adonat que el codi CSS tant del botó com dels elements de la llista és pràcticament idèntic. Arribats a aquest punt, és una bona idea fer una refacció per evitar repetir els blocs de codi. Una possible solució seria la següent:

```
#llista ul {
  padding: 0;
  margin: 0;
}

#llista li {
  list-style: none;
  margin: 5px;
}

.boto {
  flex-basis: 20%;
}
```

```
  border: 2px solid #105F85;
  padding: 5px;
  color: white;
  background-color: #2897E8;
  cursor: pointer;
}

.boto:hover, #llista li:hover {
  background-color: #105F85;
}
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/BKRKar.

Pas 8: afegir efectes amb CSS3

Ara que ja teniu definits la llista i els botons, us resta afegir un fons per a tota la pàgina i fer algun petit retoc.

S'ha decidit fer servir un degradat mitjançant CSS3, i per ajudar-vos a generar el codi CSS podeu fer servir una de les múltiples eines en línia, per exemple la que es troba a www.colorzilla.com/gradient-editor. Només heu de seleccionar el tipus de degradat, modificar-lo i copiar el codi que es genera com a propietat `background` de l'element `body` al fitxer CSS:

Cal tenir en compte que si només heu de preocupar-vos per navegadors moderns, el codi per generar el degradat es reduiria a `body {background: linear-gradient(top, rgba(167,207,223,1) 0%,rgba(35,83,138,1) 100%)}`.

```
body {
  /* Permalink - use to edit and share this gradient: http://colorzilla.co
    background: rgb(167,207,223); /* Old browsers */
    background: -moz-linear-gradient(top, rgba(167,207,223,1) 0%, rgba(35,
    background: -webkit-linear-gradient(top, rgba(167,207,223,1) 0%,rgba(3
    background: linear-gradient(to bottom, rgba(167,207,223,1) 0%,rgba(35,
}
```

Hi ha un inconvenient: en casos com aquest, en què la llargària de la pàgina no és suficient per cobrir-la sencera, el degradat s'atura on acaba el contingut. Una possible solució és fer servir un degradat lineal, i com a color del fons de l'element `html` el mateix color amb el qual acaba el degradat. D'aquesta manera, la transició és inapreciable:

```
html {
    background: rgb(35,83,138);
}

body {
    margin: 0;
    padding-top: 20px;

    /* Permalink - use to edit and share this gradient: http://colorzilla.
    background: rgb(167,207,223); /* Old browsers */
    background: -moz-linear-gradient(top, rgba(167,207,223,1) 0%, rgba(35,
    background: -webkit-linear-gradient(top, rgba(167,207,223,1) 0%,rgba(3
    background: linear-gradient(to bottom, rgba(167,207,223,1) 0%,rgba(35,
}
```

Podeu trobar un generador d'ombres de text en el següent enllaç: css3gen.com/text-shadow.

```
h1 {
  text-align: center;
  color: white;
  text-shadow: 2px 2px 2px black;
}
```

Podeu trobar un generador d'ombres de caixa en el següent enllaç: www.cssmatic.com/box-shadow.

Seguidament s'aplica com a fons un color de tipus RGBA (color de 24 bits amb transparència) i s'afegeix una ombra a tot el panell:

```
#llista, #visor {
  background: rgba(0, 0, 0, 0.75);
  border-radius: 10px;
  border: 1px solid black;
  padding: 5px;
  margin: 5px;

  /* Ombra */
  -webkit-box-shadow: 10px 10px 20px 0px rgba(0,0,0,0.75);
  -moz-box-shadow: 10px 10px 20px 0px rgba(0,0,0,0.75);
  box-shadow: 10px 10px 20px 0px rgba(0,0,0,0.75);
}
```

En canvi, per al reproductor de vídeo s'afegeix un fons negre sòlid, de manera que queda molt clar on es reproduiran els vídeos:

```
video {
  background:black;
  margin: 10px;
}
```

Finalment, elimineu la vora que es va afegir per comprovar fàcilment les delimitacions de cada element:

```
* {
  font-family: 'Sigmar One', cursive;
}
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/aNWNvv.

Es pot veure el resultat obtingut després d'aplicar els canvis anteriors a la figura 2.35.

Figura 2.35. Pas 8: aspecte final del reproductor

DISSENY D'INTERFÍCIES WEB

Pas 9: enllaçar la biblioteca jQuery i afegir efectes de so

> Podeu trobar un generador de sons de 8 bits amb llicència CC0 (domini públic) en el següent enllaç: sfbgames.com/chiptone.

Per continuar afegireu efectes de so als vostres botons: un quan el cursor estigui a sobre i un altre en fer-hi clic.

Primerament, heu d'obtenir els sons, per qualsevol mitjà; per exemple, sons descarregats de biblioteques de so amb llicència lliure, creats per vosaltres mateixos a través d'eines en línia o enregistrats i modificats amb Audacity.

Necessitareu obtenir dos fitxers d'àudio, que haureu de copiar dins del directori *assets/audio* del vostre projecte amb aquests noms:

- **selecciona.mp3**: aquest so es reproduirà en fer clic sobre un botó o element de la llista.
- **a-sobra.mp3**: aquest so es reproduirà quan passem el cursor per sobre d'un botó o element de la llista.

Una vegada tingueu els vostres fitxers d'àudio preparats, enllaceu la biblioteca jQuery i el fitxer amb el codi JavaScript, ja que fareu servir un fitxer extern per tenir el codi millor organitzat.

El primer que heu de fer és crear un nou fitxer de text buit dins de la carpeta *js* anomenat `reproductor.js`, i a continuació enllaceu tant la biblioteca jQuery com el vostre fitxer afegint el següent codi dins de la capçalera:

```
<script src="//code.jquery.com/jquery-3.1.1.min.js"></script>
<script src="js/reproductor.js"></script>
```

Sempre han d'enllaçar-se tots els **fitxers amb codi CSS** abans que els fitxers amb JavaScript, i quan treballem amb biblioteques, aquestes han de carregar-se sempre abans que els fitxers amb el codi que les fan servir.

Primerament s'afegirà la classe `sonor` als elements que vulgueu que produeixin so en passar el cursor per sobre. Comenceu afegint-la als elements de la llista de vídeos:

```
<ul>
  <li class="sonor">Vídeo 1</li>
  <li class="sonor">Vídeo 2</li>
  <li class="sonor">Vídeo 3</li>
</ul>
```

I a continuació, afegiu també la classe dels botons:

```
<div id="controls">
  <div class="boto sonor"><i class="fa fa-step-backward fa-3x"></i><br>ANT
  <div class="boto sonor"><i class="fa fa-play fa-3x "></i><br>REPRODUIR</
  <div class="boto sonor"><i class="fa fa-pause fa-3x"></i><br>ATURAR</div
  <div class="boto sonor"><i class="fa fa-step-forward fa-3x"></i><br>SEGÜ
```

Seguidament, afegiu el següent codi JavaScript al fitxer reproductor.js que detectarà quan es dispara l'*event* `mouseenter` i reproduirà el so:

```
$(document).ready(function () {

    var a_sobra = new Audio('assets/audio/a-sobra.mp3');

    $('.sonor').on('mouseenter', function() {
        a_sobra.play();
    });

});
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/xVdVpR.

Si proveu ara veureu que funciona, però no tal com s'espera. El so es reprodueix, però només torna a començar quan s'ha acabat la reproducció i no tan aviat com el cursor es col·loca sobre altre element.

Pas 10: creació d'un 'sound pool'

El so no sembla reproduir-se correctament. Aquesta és una limitació de l'element `audio` d'HTML. La solució és crear un conjunt de sons (*sound pool* en anglès), de manera que cada vegada que es demani reproduir un so s'agafarà el següent de la llista.

Utilització de 'pools'

En casos de jocs que poden tenir desenes de sons reproduint-se pràcticament al mateix temps, com IOC Invaders (github.com/XavierGaro/ioc-invaders), o d'imatges que es repeteixen, la reutilització d'aquests recursos representa una

DISSENY D'INTERFÍCIES WEB

Haureu de substituir el codi del fitxer reproductor.js pels següents fragments. En primer lloc, afegireu la crida a la funció `inicialitzarSoundPool`, a la qual passareu dos arguments: el nom pel qual volem identificar el so i l'URL on es troba el fitxer d'àudio.

Un altre canvi és que en lloc de reproduir directament el so, ara es cridarà una altra funció, també pròpia, amb el nom `reproduirSo`, passant com a argument l'identificador que hem assignat al so.

```
$(document).ready(function() {

  inicialitzarSoundPool('a_sobra', 'assets/audio/a-sobra.mp3')

  $('.sonor').on('mouseenter', function() {
    reproduirSo('a_sobra');
  });

});
```

> Reproduir un nombre elevat de sons al mateix temps pot afectar negativament el rendiment de l'aplicació.

> **Ús de constants a JavaScript**
>
> Encara que a JavaScript 5 no existeix el concepte de constant, ens pot ser útil fer servir la convenció d'emprar algunes variables com si ho fossin i posar el seu nom en majúscules, com es fa en altres llenguatges.

Seguidament s'afegeixen dues variables globals:

- `soundPool`: un objecte literal de JavaScript buit que es farà servir com a diccionari de dades.
- `MAX_SOUNDS`: una variable que es farà servir com si fos una constant, i que té la finalitat de limitar el nombre de sons idèntics que poden reproduir-se al mateix temps.

```
var soundPool = {},
  MAX_SOUNDS = 10;
```

Vegeu ara la implementació de la funció `inicialitzarSoundPool`, que té la finalitat de crear l'estructura de dades necessària per gestionar els sons i crear tots els elements d'àudio necessaris:

```
function inicialitzarSoundPool(nom, url) {
  soundPool[nom] = {
    sons: [],
    actual: 0
```

```
    soundPool[nom].sons.push(new Audio(url));
  }
}
```

Com podeu veure, la funció és molt senzilla, però pot resultar una mica estranya si encara no es domina el llenguatge JavaScript.

El primer que s'ha fet és afegir a l'estructura de dades global `soundPool` un nou objecte literal. La clau per accedir a aquest objecte dins de `soundPool` serà el `nom` que s'ha passat com a argument, i l'objecte creat tindrà dues propietats:

- **sons**: un *array* que emmagatzemarà tots els elements d'àudio creats per reproduir aquest so.
- **actual**: un *enter* que servirà per saber quin és l'element d'àudio que s'ha de reproduir en cada moment.

> El mètode `push` és propi dels *arrays* de JavaScript i permet afegir un element al final de l'*array*.

A continuació s'utilitza un bucle `for` per crear tants elements d'àudio com indica la variable global `MAX_SOUNDS`, i s'afegeixen a l'*array* de l'objecte creat amb el mètode `push`: `soundPool[nom].sons.push(new Audio(url));`.

Fixeu-vos que tots els elements d'àudio així creats s'afegeixen a l'objecte guardat al *sound pool* amb el `nom` passat com a argument i amb el fitxer d'àudio especificat com a `url`.

Per exemple, si el `nom` és *a_sobra* i la `url` fos *assets/audio/a-sobra.mp3*, internament cada línia del bucle s'interpretaria així:

```
soundPool['a_sobra'].sons.push(new Audio('assets/audio/a-sobra.mp3'));
```

Per acabar, afegiu el mètode `reproduirSo`, que agafarà un nou element de so del *sound pool* cada vegada que es cridi fins a arribar a l'últim element, on tornarà a començar:

```
function reproduirSo(nom) {
  var index = soundPool[nom].actual;
  soundPool[nom].sons[index % MAX_SOUNDS].play();
  soundPool[nom].actual++;
}
```

Podeu veure l'exemple complet en el següent enllaç: codepen.io/ioc-daw-m09/pen/RaVQNV.

El primer que es fa és obtenir l'índex del so corresponent al nom passat com a argument que toca reproduir.

A continuació, s'aprofita la propietat del mòdul per evitar haver de fer la comprovació quan s'arriba al final de l'*array*. Atès que la mida d'aquest és igual a MAX_SOUNDS, es

Finalment, s'incrementa el valor del so actual per deixar llest el *sound pool* per reproduir el següent.

Pas 11: reutilització de funcions

Ara que ja funciona correctament, afegiu el so per a l'*event* `click`, afegint al final de la funció cridada per `ready` el següent codi:

```
inicialitzarSoundPool('selecciona', 'assets/audio/selecciona.mp3')

$('.sonor').on('click', function () {
  reproduirSo('selecciona');
});
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/bpWLBj.

Com podeu veure, una vegada implementat el sistema de *pools* d'àudio és molt fàcil afegir qualsevol quantitat de sons. Només s'ha hagut de canviar el nom del so, l'URL del fitxer que es vol reproduir i l'*event* al qual es vol associar aquest nou so.

Pas 12: afegir animacions CSS

> Podeu trobar una explicació detallada de com crear animacions amb CSS a l'apartat "Elements multimèdia al web: imatges, vídeo i àudio" de la unitat "Creació i integració d'elements multimèdia al web".

Per acabar amb els efectes dels botons i de la llista de vídeos s'inclourà una petita animació fent servir CSS; afegiu el següent codi al final del fitxer reproductor.css:

```
.sonor {
    transition: 0.5s ease-out;
}

.sonor:hover {
    animation-name: zoom;
    animation-duration: 0.5s;
    animation-direction: alternate;
}

@keyframes zoom {
    from {
        transform: scale(1.0) rotate(5deg);
    }

    to {
        transform: scale(1.1) rotate(-5deg);
    }
}
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/YqVeaE.

Pas 13: creació d'elements de la llista i reproducció de vídeo

Fins ara, la llista de vídeos que es mostrava era fixada pel codi HTML, però això no és

DISSENY D'INTERFÍCIES WEB

En un cas real, segurament aquesta llista es carregaria fent servir AJAX, però per simplificar afegireu la informació dels vídeos directament al vostre fitxer reproductor.js amb una estructura de dades de tipus JSON, i es generaran els elements de la llista via codi.

El primer que fareu és eliminar els elements de la llista de vídeos i afegir un identificador per facilitar la feina d'afegir-los. La llista al vostre codi HTML ha de quedar així:

```
<div id="llista">
  <h1>Vídeos</h1>
  <ul id="videos">
  </ul>
</div>
```

Objectes a partir de notació literal

A JavaScript és possible crear objectes de forma literal, això facilita la creació d'estructures que es poden utilitzar com a diccionaris de dades, permetent tractar els elements emmagatzemats com a parells de *clau-valor*.

A continuació afegiu a la declaració de variables general del fitxer reproductor.js la informació dels vídeos com un *array* d'objectes de JavaScript creats amb *notació literal*, juntament amb una variable anomenada `selecciona` que inicialment tindrà el valor `0` per establir com a seleccionat el primer vídeo:

```
var soundPool = {},
    MAX_SOUNDS = 10,
    videos = [
        {
            titol: 'El primer vídeo',
            descripcio: 'Aquesta és la descripció del primer vídeo',
            url: 'assets/video/video-1.mov'
        },
        {
            titol: 'El segon vídeo',
            descripcio: 'Aquesta és la descripció del segon vídeo',
            url: 'assets/video/video-2.mov'
        },
        {
            titol: 'El tercer vídeo',
            descripcio: 'Aquesta és la descripció del tercer vídeo',
            url: 'assets/video/video-3.mov'
        }
    ],
    seleccionat = 0;
```

Seguidament, afegireu una funció per inicialitzar la llista de vídeos, la finalitat de la qual serà la següent:

- Recórrer a totes les posicions de l'*array* de vídeos.
- Crear un element de tipus `li` per a cada element de l'*array*.
- Afegir la classe `sonor` per conservar els efectes CSS afegits anteriorment.
- Afegir un identificador únic que estarà format pel mot `video-` i l'índex que li correspongui a l'element de l'*array*.

sobre l'element es mostri la descripció del vídeo.

- Afegir el nou element a la llista de `videos`.
- Afegir la detecció de l'*event* `click` per establir aquest vídeo com el seleccionat i iniciar la seva reproducció.

Una vegada es té clar en què consisteix la implementació, fent servir jQuery trobareu que és pràcticament més simple que redactar-la, ja que per a cada una d'aquestes accions hi ha un mètode o funció de jQuery que facilita la tasca. Afegiu el següent codi al final del fitxer reproductor.js:

```
function inicialitzarLlistaVideos() {
    $.each(videos, function (index, value) {
        // Creem un nou node de tipus LI fent servir jQuery
        var $node = $('<li></li>');

        // S'afegeix la classe sonor
        $node.addClass('sonor');

        // S'afegeix un identificador únic basat en el seu índex
        $node.attr('id', 'video-'+index);

        // S'afegeix el contingut de l'element
        $node.text(value.titol);

        // S'afegeix la descripció per mostrar quan deixem el cursor a sob
        $node.attr('title', value.descripcio);

        // S'afegeix el node a la llista de vídeos
        $('#videos').append($node);

        // S'afegeix la detecció del esdeveniment click per establir aques
        $node.on('click', function() {
            seleccionat = index;
            reproduirVideo();
        })
    });
}
```

Ara que ja teniu la funció que inicialitza la llista de vídeos només cal cridar-la. Haureu d'afegir la crida a aquesta funció just abans de la inicialització dels *pools*, perquè si ho afegiu després els elements de la llista no quedaran enllaçats amb aquests:

```
$(document).ready(function () {
    inicialitzarLlistaVideos();
```

Finalment, només resta implementar la funció `reproduirVideo` i obtindreu un resultat com el de la figura 2.36:

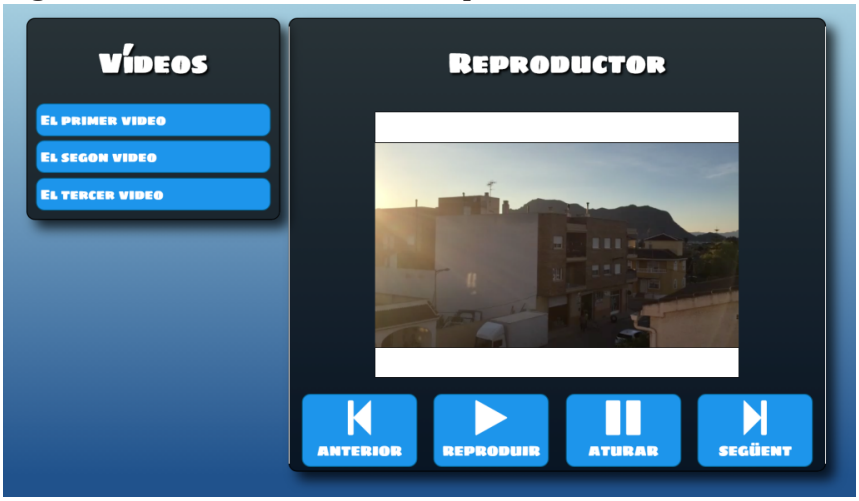
```
function reproduirVideo() {
    var video = videos[seleccionat];
    $('video').attr('src', video.url);
    $('video').get(0).play();
}
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/ZWKPjy.

Aquesta funció és molt simple: primer s'agafen les dades del vídeo seleccionat de l'*array*, a continuació se selecciona l'element `video` i es canvia el seu atribuït `src` a l'URL del vídeo.

retorna el primer element de tipus `video` que es trobi a la pàgina, i a continuació cridar el seu mètode `play` per reproduir el vídeo.

Figura 2.36. Pas 13: versió final del reproductor amb vídeo



Pas 14: gestió del reproductor a través dels botons

Ja gairebé heu enllestit el vostre reproductor i només cal afegir la funcionalitat als botons. El primer que haureu de fer és afegir un identificador únic a cadascun d'aquests botons, modificant el codi HTML:

```
<div id="controls">
  <div id="anterior" class="boto sonor"><i class="fa fa-step-backward fa-3
  <div id="reproduir" class="boto sonor"><i class="fa fa-play fa-3x "></i>
  <div id="aturar" class="boto sonor"><i class="fa fa-pause fa-3x"></i><br
  <div id="seguent" class="boto sonor"><i class="fa fa-step-forward fa-3x"
</div>
```

A continuació, heu d'afegir la detecció de l'esdeveniment `clic` per a cadascun d'ells; ho podeu fer a continuació de la inicialització dels efectes de so, associant cada botó amb una funció diferent:

```
$('#anterior').click(anteriorVideo);
$('#reproduir').click(reproduirVideo);
$('#aturar').click(aturarVideo);
$('#seguent').click(seguentVideo);
```

> Es pot consultar la llista completa de dreceres per gestionar *events* en el següent enllaç: goo.gl/U8yxFW.

Aquest cop, en lloc de fer servir el mètode `on` de jQuery, s'ha fet servir el mètode `click`. En aquesta situació és indiferent, i es pot fer servir l'un o l'altre indistintament. jQuery facilita una extensa llista de dreceres per escoltar *events*: `click`, `dblclick`, `mousedown`, etc.

Si necessiteu escoltar **múltiples *events*** simultàniament heu de fer

```
});.
```

Una altra diferència que es pot apreciar és que en lloc de cridar una funció com en els exemples anteriors s'ha posat només el nom de les funcions corresponents. En tots dos casos es tracta del que anomenem *callback*, funcions que són cridades per altres funcions. Per exemple, quan es dispara l'*event* associat com en aquest cas, quan es finalitza un temporitzador (amb les funcions `setInterval` o `setTimeout`) o quan retorna una petició AJAX del servidor amb èxit.

El següent pas és implementar aquestes funcions. Com que la funció `reproduirVideo` s'ha implementat anteriorment, només cal codificar les noves:

```
function aturarVideo() {
    var video = $('video').get(0);

    if (video.paused) {
        video.play();
    } else {
        video.pause();
    }
}

function anteriorVideo() {
    seleccionat--;
    if (seleccionat < 0) {
        seleccionat = videos.length - 1;
    }
    reproduirVideo();
}

function seguentVideo() {
    seleccionat++;
    if (seleccionat === videos.length) {
        seleccionat = 0;
    }
    reproduirVideo();
}
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/mPmgMV.

> Recordeu que l'element `video` és un element afegit a HTML5 i compta amb una API pròpia que exposa mètodes com `play` i `pause`, i propietats com `paused`.

Vegeu amb detall aquestes funcions. En primer lloc, teniu la funció `aturarVideo`. El seu comportament és diferent del de les altres, ja que ha de controlar l'estat de reproducció; per aquesta raó, el que s'ha fet és obtenir l'element de vídeo, i a partir de la propietat `paused` determinar si cal aturar-lo (`video.pause()`) o reproduir-lo (`video.play()`).

Les funcions `anteriorVideo` i `seguentVideo` són pràcticament idèntiques, només redueixen o augmenten el valor de l'element seleccionat, canviant a l'última o primera posició respectivament si el valor es troba fora de rang, i a continuació inicien la repro-

Pas: 15. afegir funcionalitat al teclat

Per acabar amb aquest exemple s'afegirà una funcionalitat més: quan es premi la tecla espai, el vídeo alternarà entre pausa i reproducció. Afegiu el següent codi a continuació del codi per gestionar els *events* dels botons:

```
$(document).keypress(function(evt) {
  if (evt.charCode==32) {
    aturarVideo();
  }
});
```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m09/pen/qZmwed.

> `keypress` és una drecera per `(on("keypress", function(evt) {…}`.

Com veieu, ha estat molt simple afegir aquesta funcionalitat. S'ha fet una subscripció per escoltar l'*event* `keypress` de tot el document, i quan es dispara es comprova si la propietat `charCode` té el valor `32` (que correspon a la barra d'espai). Si és així, es crida `aturarVideo`, que l'aturarà o el reproduirà segons l'estat actual.

Cal tenir en compte que no tots els navegadors retornen la mateixa informació a l'objecte *event*, sobretot en referencia als *events* de teclat i per tant és recomanable fer servir biblioteques per controlar aquest comportament.

A continuació podeu veure el llistat complet dels fitxers que componen el reproductor començant per reproductor.html:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Reproductor</title>
    <link href="css/reproductor.css" rel="stylesheet" type="text/css"/>
    <link href="https://fonts.googleapis.com/css?family=Sigmar+One" rel="s
    <link href="https://maxcdn.bootstrapcdn.com/font-awesome/4.5.0/css/fon

    <script src="//code.jquery.com/jquery-3.1.1.min.js"></script>
    <script src="js/reproductor.js"></script>

</head>
<body>
<main>
    <div id="llista">
        <h1>Vídeos</h1>
        <ul id="videos">
        </ul>
    </div>
    <div id="visor">
        <h1>Reproductor</h1>
        <video width="430px" height="315px"  type="video/mp4">
        </video>
        <div id="controls">
            <div id="anterior" class="boto sonor"><i class="fa fa-step-bac
            <div id="reproduir" class="boto sonor"><i class="fa fa-play fa
            <div id="aturar" class="boto sonor"><i class="fa fa-pause fa-3
            <div id="seguent" class="boto sonor"><i class="fa fa-step-forw
        </div>
    </div>
```

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

Aquest és el contingut de reproductor.css:

```
* {
    font-family: 'Sigmar One', cursive;
}


h1 {
    text-align:center;
    color: white;
    text-shadow: 2px 2px 2px black;
}

html {
    background: rgb(35,83,138);
}

body {
    margin: 0;
    padding-top: 20px;

    /* Permalink - use to edit and share this gradient: http://colorzilla.
    background: rgb(167,207,223); /* Old browsers */
    background: -moz-linear-gradient(top, rgba(167,207,223,1) 0%, rgba(35,
    background: -webkit-linear-gradient(top, rgba(167,207,223,1) 0%,rgba(3
    background: linear-gradient(to bottom, rgba(167,207,223,1) 0%,rgba(35,
}

main {
    max-width: 960px;
    margin: 0 auto;
}

video {
    background:black;
    margin: 10px;
}

#llista, #visor {
    float: left;
}

#llista {
    width: 30%;
}


#visor {
    width: 65%;
    text-align: center;
}

#controls {
    display: flex;
    justify-content: space-around;
}

#llista ul {
    padding: 0;
    margin: 0;
}

#llista li {
    list-style: none;
    margin: 5px;
}

.boto {
    flex-basis: 20%;
}
```

```
    padding: 5px;
    color: white;
    background-color: #2897E8;
    cursor: pointer;
}

.boto:hover, #llista li:hover {
    background-color: #105F85;
}

.clearfix {
    clear: both;
    border: none;
}

#llista, #visor {
    background: rgba(0, 0, 0, 0.75);
    border-radius: 10px;
    border: 1px solid black;
    padding: 5px;
    margin: 5px;

    /* Ombra */
    -webkit-box-shadow: 10px 10px 20px 0px rgba(0,0,0,0.75);
    -moz-box-shadow: 10px 10px 20px 0px rgba(0,0,0,0.75);
    box-shadow: 10px 10px 20px 0px rgba(0,0,0,0.75);
}

.sonor {
    transition: 0.5s ease-out;
}

.sonor:hover {
    animation-name: zoom;
    animation-duration: 0.5s;
    animation-direction: alternate;
}

@keyframes zoom {
    from {
        transform: scale(1.0) rotate(5deg);
    }

    to {
        transform: scale(1.1) rotate(-5deg);
    }
}
```

I finalment, el contingut de reproductor.js:

```
$(document).ready(function () {

    inicialitzarLlistaVideos();

    inicialitzarSoundPool('a_sobra', 'assets/audio/a-sobra.mp3')

    $('.sonor').on('mouseenter', function () {
        reproduirSo('a_sobra');
    });


    inicialitzarSoundPool('selecciona', 'assets/audio/selecciona.mp3')

    $('.sonor').on('click', function () {
        reproduirSo('selecciona');
    });

    $('#anterior').click(anteriorVideo);
    $('#reproduir').click(reproduirVideo);
    $('#aturar').click(aturarVideo);
    $('#seguent').click(seguentVideo);

    $(document).keypress(function(evt) {
        if (evt.charCode==32) {
```

DISSENY D'INTERFÍCIES WEB

```
});

var soundPool = {},
    MAX_SOUNDS = 10,
    videos = [
        {
            titol: 'El primer vídeo',
            descripcio: 'Aquesta és la descripció del primer vídeo',
            url: 'assets/video/video-1.mov'
        },
        {
            titol: 'El segon vídeo',
            descripcio: 'Aquesta és la descripció del segon vídeo',
            url: 'assets/video/video-2.mov'
        },
        {
            titol: 'El tercer vídeo',
            descripcio: 'Aquesta és la descripció del tercer vídeo',
            url: 'assets/video/video-3.mov'
        }
    ],
    seleccionat = 0;

function inicialitzarSoundPool(nom, url) {
    soundPool[nom] = {
        sons: [],
        actual: 0
    };

    for (var i = 0; i < MAX_SOUNDS; i++) {
        soundPool[nom].sons.push(new Audio(url));
    }
}

function reproduirSo(nom) {
    var index = soundPool[nom].actual;
    soundPool[nom].sons[index % MAX_SOUNDS].play();
    soundPool[nom].actual++;
}

function inicialitzarLlistaVideos() {
    $.each(videos, function (index, value) {
        // Creem un nou node de tipus LI fent servir jQuery
        var $node = $('<li></li>');

        // Afegim la classe sonor
        $node.addClass('sonor');

        // Afegim un identificador únic basat en el seu índex
        $node.attr('id', 'video-' + index);

        // Afegim el contingut de l'element
        $node.text(value.titol);

        // Afegim la descripció per mostrar quan deixem el cursor a sobre
        $node.attr('title', value.descripcio);

        // Afegim el node a la llista de vídeos
        $('#videos').append($node);

        // Afegim la detecció de l'esdeveniment click per establir aquest
        $node.on('click', function () {
            seleccionat = index;
            reproduirVideo();
        })
    });
}

function reproduirVideo() {
    var video = videos[seleccionat];
    $('video').attr('src', video.url);
    $('video').get(0).play();
}

function aturarVideo() {
    var video = $('video').get(0);
```

```
        } else {
            video.pause();
        }
    }

    function anteriorVideo() {
        seleccionat--;
        if (seleccionat < 0) {
            seleccionat = videos.length - 1;
        }
        reproduirVideo();
    }

    function seguentVideo() {
        seleccionat++;
        if (seleccionat === videos.length) {
            seleccionat = 0;
        }
        reproduirVideo();
    }
```

Per finalitzar, podeu veure l'aspecte final del reproductor en funcionament a la figura 2.37.

Figura 2.37. Pas 15: resultat final

2.2.2. Creació d'un bàner amb carrega dinàmica de dades

Seguidament desenvolupareu un bàner que permetrà mostrar diferents anuncis de manera rotativa configurats a partir d'una estructura de dades JSON (més concretament un objecte literal de JavaScript). Les característiques que inclourà són:

- Creat a partir de l'element `canvas` existent al codi HTML.
- Configuració dels anuncis a partir de l'objecte JSON amb:
  - Imatge a mostrar.
  - Text que es mostrarà si es posa el cursor a sobre.
  - URL al qual es redirigirà en fer clic sobre l'anunci.
  - Canvi d'anunci periòdicament.

Aquest exemple és força interessant perquè soluciona un problema amb el qual us podeu trobar en el món laboral: les **còpies de pàgina a la memòria cau** (*cache*, en anglès). Per optimitzar el temps de càrrega de les pàgines és habitual fer servir algun sistema de memòria cau a la banda del servidor, i configurar les directives del servidor

- Els continguts de la pàgina no s'actualitzen: la primera vegada que es demana la pàgina, el servidor fa totes les gestions necessàries per crear-la (per exemple, les consultes a la base de dades) i a partir d'aquest moment retorna sempre la mateixa pàgina. El sistema, per refrescar aquestes pàgines, depèn de l'aplicació del servidor i no hi ha res que pugui fer l'usuari en aquest sentit.
- Els fitxers que es descarreguen amb la pàgina, com poden ser imatges, fitxers amb codi JavaScript o CSS, fonts, etc., poden ser servits pel servidor amb una data d'expiració (per exemple, 30 dies a partir del moment que s'ha descarregat) i fins que no ha passat aquest temps o l'usuari esborra les dades del navegador no tornen a descarregar-se.

Si només necessiteu mostrar una sèrie d'anuncis fixos no hi hauria problema, però actualment el cas més habitual és haver de mostrar una gran quantitat d'anuncis diferents, i per tant incrustar els anuncis directament al codi HTML no és una solució vàlida.

> Podeu trobar més informació sobre els servidors d'anuncis o *Ad Servers* en el següent enllaç: en.wikipedia.org /wiki/Ad_serving.

Una possible solució és fer servir un servidor d'anuncis, això us permet aprofitar els avantatges de la memòria cau i mostrar diferents anuncis cada vegada, ja que la gestió de quins anuncis s'han de mostrar recau sobre el servidor que ens enviarà tota la informació necessària per mostrar els anuncis.

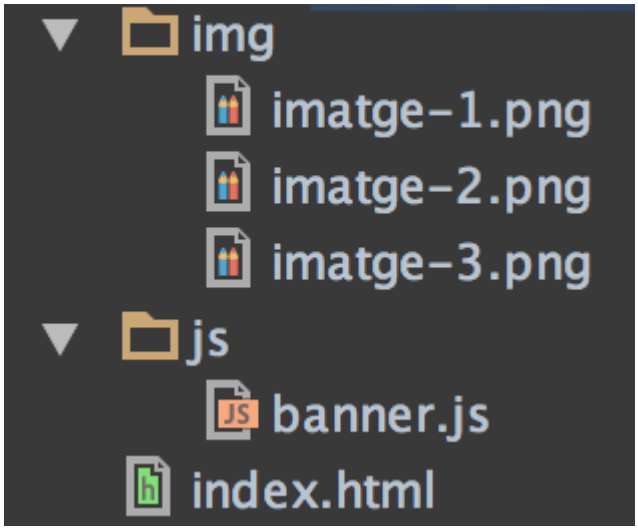
En el aques cas, el que e farà és afegir l'estructura de dades directament del fitxer amb el codi JavaScript, ja que com s'ha comentat anteriorment la implementació de peticions al servidor queda fora de l'abast d'aquests materials.

Una vegada es té clar el concepte hi ha moltes maneres de portar-lo a terme: es podria afegir directament codi HTML a la pàgina, o bé afegir un element `iframe` que contingués l'anunci. Aquesta implementació consisistirà en utilitzar un element `canvas` per mostrar els anuncis.

En aquest exemple es farà servir una estructura simple que es pot veure a la figura 2.38. Els fitxers d'imatge aniran al directori */img* i el fitxer JavaScript, a la carpeta */js*.

Figura 2.38. Estructura de fitxers i directoris per al bàner

DISSENY D'INTERFÍCIES WEB



Afegiu aquest codi al fitxer banner.html; com podeu veure en aquest exemple, no es farà servir CSS, només s'enllaçarà amb la biblioteca jQuery i el fitxer de codi JavaScript:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Pàgina que conté el bàner</title>

    <script src="//code.jquery.com/jquery-3.1.1.min.js"></script>
    <script src="js/banner.js"></script>
</head>
<body>
<canvas id="banner" width="300" height="300">El teu navegador no suporta l
</body>
</html>
```

Vegeu ara el contingut del fitxer banner.js amb deteniment:

```
var TEMPS_ENTRE_ANUNCIS = 5 * 1000, // Temps en mil·lisegons
    anuncis = [
        {titol: 'Aquest és el primer anunci', imatge: 'img/imatge-1.png',
        {titol: 'Aquí podeu veure el segon anunci', imatge: 'img/imatge-2.
        {titol: 'I finalment, el tercer anunci', imatge: 'img/imatge-3.png
    ],
    anunciActual = 0;
```

El primer que s'ha afegit és una variable (emulant una constant), anomenada `TEMPS_ENTRE_ANUNCIS`, per facilitar la lectura del codi. Aquesta variable guardarà el temps que ha de passar entre en anunci i el següent en mil·lisegons.

A continuació podeu veure l'estructura de dades que s'ha fet servir per als anuncis d'exemple, un *array* d'objectes amb notació literal que contenen el títol, la imatge a mostrar i l'URL al qual redirigeixen en fer-hi clic.

I finalment, vegeu la **variable global** que indica quin és l'anunci actual.

> Podeu trobar més informació sobre *ES.next* a es6-features.org.

del *DOM* s'hagi completat:

```
$(document).ready(function () {
    var canvas = $("#banner").get(0), // Equivalent a: document.getElement
        context = canvas.getContext("2d");

    rotarAnuncis(context);
});
```

En el bloc d'inicialització només ha calgut obtenir el context de l'element `canvas` que s'haurà de passar a la funció `rotarAnuncis` i que l'utilitzarà per iniciar la rotació d'anuncis.

Fixeu-vos que seria molt fàcil ampliar aquest codi per suportar múltiples àrees d'anuncis, només caldria canviar el comptador de l'anunci actual per suportar múltiples àrees i començar a rotar els anuncis dins de la funció d'inicialització amb les diferents àrees.

```
function rotarAnuncis(context) {
    establirAnunci(context);
    setTimeout(function () {
        anunciActual++;
        rotarAnuncis(context);
    }, TEMPS_ENTRE_ANUNCIS);
}
```

La funció `rotarAnuncis` també és molt simple: estableix l'anunci `canvas` del qual s'ha passat el seu `context` com a argument, i després inicia un temporitzador per incrementar el comptador d'anunci actual i cridar-se a si mateix.

Vegeu la implementació d'`establirAnunci`, on es presenta un nou mètode jQuery, el mètode `unbind`:

```
function establirAnunci(context) {

    var anunci = anuncis[anunciActual % anuncis.length],
        $banner = $('#banner').attr('title', anunci.titol);

    // Ens assegurem que no existeixi ja un event de tipus click
    $banner.unbind('click');

    // Lliguem la funció a l'event click
    $banner.click(function () {
        window.open(anunci.url, '_blank');
    });

    dibuixarAnunci(anunci.imatge, context);
}
```

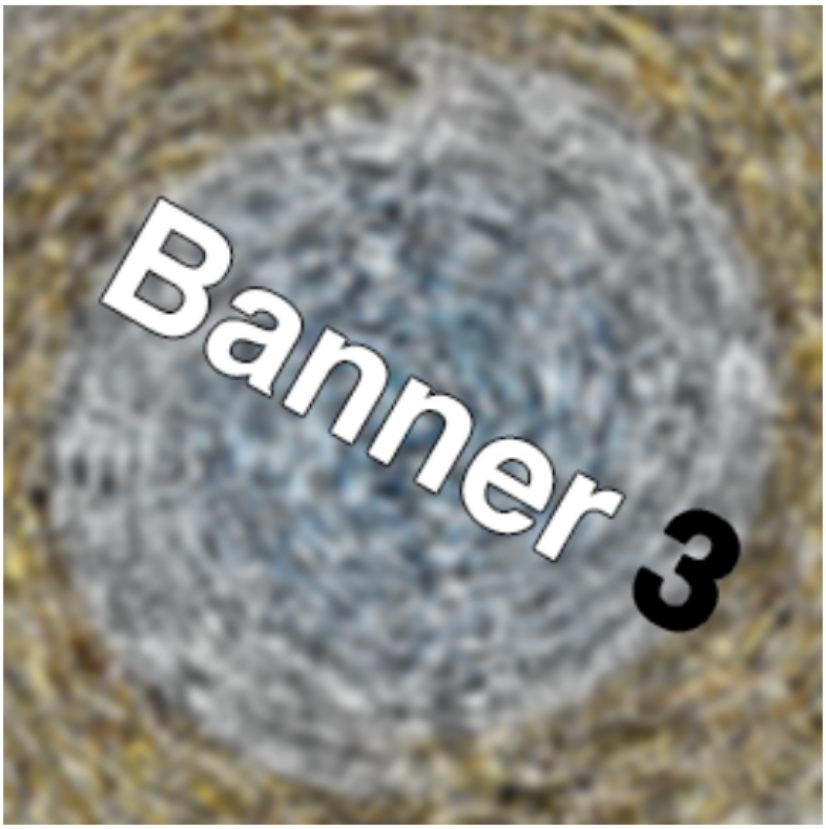
Fixeu-vos que s'ha fet ús de l'operació mòdul per evitar haver de controlar si el comptador d'anunci actual es troba fora de rang. A continuació podeu veure que es crida el mètode jQuery `unbind`, això serveix per eliminar qualsevol detector d'*events* lligat a l'*event* passat com a paràmetre (`click`). Si es fa així es poden provocar conflictes en fer clic a l'anunci, ja que es produeixen crides a les funcions anteriors.

Una vegada deslligades totes les funcions de l'*event* `click`, es procedeix a lligar-lo a la funció que correspon, i que provoca l'apertura d'una nova pàgina, a una pestanya o finestra diferent, amb l'URL de l'anunci: `window.open(anunci.url, '_blank');`.

```
function dibuixarAnunci(imatge, context) {
    var img = new Image();

    img.onload = function () {
        context.drawImage(img, 0, 0);
    };

    img.src = imatge;
}
```

Podeu veure l'exemple complet en el següent enllaç: codepen.io/ioc-daw-m09/pen/pywrJP i la seva visualització a la figura 2.39.

Figura 2.39. Bàner mostrant un dels anuncis



Aquest codi és molt simple, però conté un element molt important. Primer es crea un element de tipus `image` d'HTML5, i a continuació, **abans d'establir l'origen de la imatge**, s'afegeix una funció que serà cridada una vegada finalitzi la càrrega de la imatge, aquí és on afegim el codi que la dibuixarà al `canvas`. Si no s'afegeix el codi de la funció `onload` abans d'establir l'origen és possible que aquesta acabi de descarregar-se sense cridar la funció, especialment si aquesta es troba en local o emmagatzemada a la memòria cau del navegador.

S'ha de tenir en compte que els recursos de la pàgina es continuen carregant encara que hagi finalitzat la càrrega del DOM. Això vol dir que el codi pot començar a executar-se abans que les imatges s'hagin descarregat, la qual cosa provocarà que no es mostrin al `canvas` i per tant que no funcioni correctament. Per aquesta raó, **sempre que treballeu amb imatges i l'element `canvas` heu de gestionar l'esdeveniment `onload` de la imatge**, ja sigui creant un *gestor de descàrregues* o directament.

Finalment, s'estableix l'origen de la imatge, que es carregarà automàticament; a conti-

DISSENY D'INTERFÍCIES WEB

2.3. Els drets d'autor i l'ús de les llicències

Cal tenir en compte la qüestió dels drets d'autor i de l'ús de les llicències, ja que sense aquest coneixement podeu trobar-vos molts problemes a l'hora d'utilitzar elements que no hagin estat creats exclusivament per vosaltres.

Un punt molt important a l'hora de treballar amb elements multimèdia és que de vegades no es té clar si es pot fer servir un element en les nostres pàgines web o aplicacions, si podem editar-lo o si algú pot fer servir el vostre treball sense el vostre consentiment.

Fins i tot en el cas d'haver comprat una fotografia o cançó, això no us dóna automàticament el dret de fer-la servir, i per aquesta raó s'ha de tenir molt en compte sota quina llicència s'ha adquirit l'element i en cas de dubte consultar amb l'autor.

2.3.1. Drets de la propietat intel·lectual

Més informació sobre la propietat intel·lectual

Podeu trobar més informació sobre els drets de la propietat intel·lectual a la web del Ministeri de Educació, Cultura i Esport: goo.gl/83dWgO.

Tota obra està subjecta als drets d'autor, de manera que en cas de dubte sempre es pot donar per suposat que una obra determinada està protegida i no es pot fer servir sense contactar primer amb el seu autor.

Per aquesta raó és imprescindible consultar sempre la llicència de qualsevol material que vulgueu fer servir, allà trobareu què se'n pot fer i què no. Un exemple són els llocs web de descàrregues de recursos que permeten als autors distribuir les seves creacions com a mostres del seu de treball, de demostració o de manera completament lliure.

Pel mateix motiu també és important incorporar la llicència d'ús als vostres treballs, ja que segons la llicència que feu servir se'n podrà fer un ús o un altre del nostre treball o les vostres publicacions.

2.3.2. Llicències

En el cas del disseny d'aplicacions web trobareu que a la vostra feina es poden aplicar diferents tipus de llicències; per una banda, part de la nostra feina pot ser creativa (dissenys d'imatges), i per una altra, normalment inclourà codi, i per tant s'aplicarien llicències de programari.

En tot cas, es poden distingir dos grans grups:

Drets d'autor i programari

S'ha de tenir en compte que en el cas del programari, pel que fa als drets d'autor, s'ha d'enregistrar

- **Llicències amb *copyright***: si no s'especifica una altra cosa, tota obra és considerada automàticament amb *copyright*, no cal fer cap acció específica ni indicar-ho. Aquesta protecció permet a l'autor explotar la seva obra i cedir part dels seus drets a altres.
- **Llicències amb *copyleft***: aquests tipus de llicències permeten a un autor renunciar als seus drets sobre l'obra, de manera que aquesta pot ser copiada, derivada, modificada i redistribuïda. Sempre s'ha d'incloure la llicència on s'especifica què pot i què no pot fer la persona que rep una còpia, per evitar per exemple que la redistribueixin sense aplicar-hi el *copyleft*. Dintre del *copyleft* podem trobar també llicències amb *copyleft* parcial. Una de les més conegudes d'aquest tipus és Creative Commons.

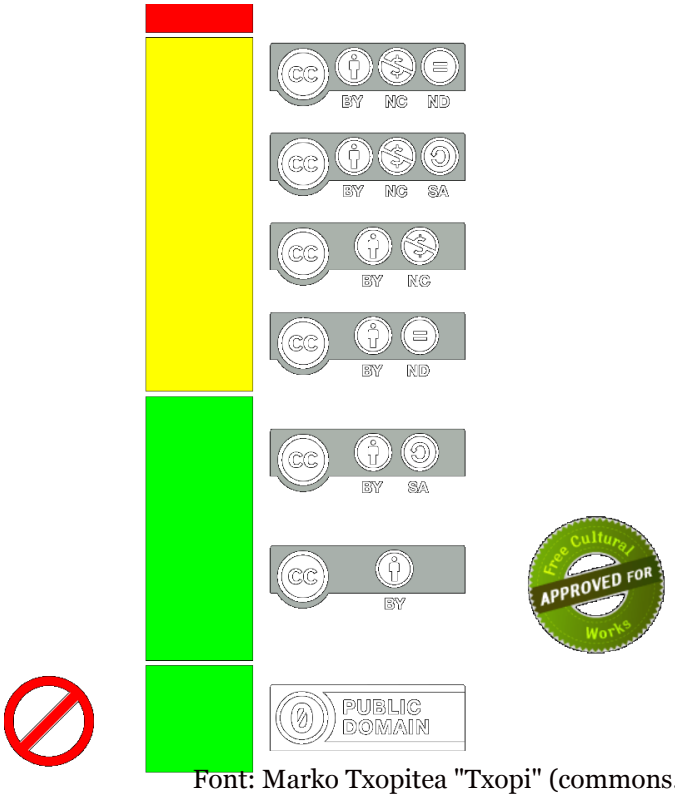
Podeu trobar més informació sobre les llicències Creative Commons a cat.creativecommons.org.

Un tipus de llicències molt utilitzades actualment són les **Creative Commons**. Aquestes comparteixen alguns punts amb el *copyleft*, ja que permeten als autors renunciar a part dels seus drets sobre les seves obres, de manera que poden ser compartides molt fàcilment.

Aquestes llicències tenen dos punts forts: per una banda, són molt fàcils d'entendre (com pot apreciar-se a la figura 2.40), i per una altra poden combinar-se per formar els següents sis tipus, més el de domini públic:

- **CC0**: domini públic. Renunciem a tot dret sobre l'obra (excepte als irrenunciables).
- **CC BY**: reconeixement. Permet distribuir, remesclar, retocar i crear a partir de l'obra, fins i tot amb fins comercials, sempre que s'acrediti l'autor.
- **CC BY-SA**: reconeixement-compartir igual. Com l'anterior, però l'obra s'ha de distribuir amb les mateixes condicions.
- **CC BY-ND**: reconeixement-sense obra derivada. Aquesta llicència no permet crear obres derivades, es pot distribuir l'obra comercialment però de manera íntegra i acreditant l'autor.
- **CC BY-NC**: reconeixement-no comercial. Amb aquesta llicència no és possible fer un ús comercial de l'obra.
- **CC BY-NC-SA**: reconeixement-no comercial-compartir igual. Aquest tipus no permet fer un ús comercial i la distribució s'ha de fer sota les mateixes condicions.
- **CC BY-NC-ND**: reconeixement-no comercial-sense obra derivada. Aquesta és l'opció més restrictiva, atès que no permet modificar l'obra ni comercialitzar-la.

Figura 2.40. Semàfor Creative Commons



Font: Marko Txopitea "Txopi" (commons.wikimedia.org)

Per què no s'utilitza Creative Commons per al programari?

Encara que pot ser temptador fer servir una llicència Creative Commons per al programari, és desaconsellat per la mateixa organització. El que fan servir, i recomanen, per al programari lliure són les llicències GPL.

Es poden trobar una gran quantitat de llicències de programari lliure, això és perquè no existeix un acord global sobre què és el programari lliure i el programari obert. Alguns desenvolupadors són partidaris que tot ha de ser lliure i gratuït, mentre que d'altres opinen que el codi ha de ser lliure però s'ha de pagar. És a dir, si una persona compra un programari determinat, aquest ha d'anar acompanyat del codi font.

Rendibilitzar el programari lliure

Una pràctica habitual per rendibilitzar els projectes de programari lliure és facilitar la base i cobrar després els serveis de consultoria, d'instal·lació, vendre llibres, desenvolupament de *plugins*, etc.

Un exemple de programari obert d'aquest tipus és **Wordpress**; per una banda, tenim el codi de forma gratuïta i, per una altra, es poden contractar els seus serveis d'allotjament i consultoria, comprar temes i *plugins* de tercers, llibres sobre diferents aspectes de Wordpress, etc.

GNU GPL i programari privatiu

una mateixa obra o codi sota diferents llicències, de manera que seria possible distribuir el programari sota GPL i vendre'l a una companyia sota una llicència amb *copyright*.

Entre aquestes llicències, les més utilitzades són:

- **GNU General Public License (GPL) 2.0 i 3.0**: escrita originàriament per Richard Stallman, aquesta llicència garanteix a l'usuari final executar, estudiar, compartir i modificar el programari, raó per la qual s'ha d'adjuntar o oferir el codi font. Aplica el *copyleft*, de manera que les obres derivades només poden ser distribuïdes sota els mateixos termes de llicència, encara que permet vendre les obres derivades a qualsevol preu. Com que obligatòriament ha d'adjuntar el codi font, no permet incloure aquest codi junt amb programari privatiu.
- **MIT License**: creada pel Massachussetts Institute of Technology, aquesta llicència és molt semblant a la GNU GPL, però es pot distribuir com a programari privatiu i no adjuntar el codi font.
- **Apache License 2.0**: aquesta és com la del MIT, garanteix els mateixos drets que la GNU GPL amb la possibilitat de no adjuntar el codi font, però afegeix expressament els drets sobre patents per fer-lo servir.
- **BSD License 2.0**: molt similar a les dues anteriors, no és gaire recomanable usar-la, perquè pot ser confosa amb la versió BSD original, que és defectuosa. Per aquesta raó és més recomanable fer servir Apache License 2.0.

En resum, el més habitual és que es faci servir una llicència GPL 3.0 si no voleu reservar-vos cap dret o una Apache License 2.0 si es vol reservar el dret de distribuir el codi font amb el programari lliure.

Es pot consultar una llista completa de llicències lliures al lloc web de GNU: www.gnu.org/licenses/license-list.html.

En cas de voler combinar diferents elements o codi (per exemple, llibreries), s'ha de tenir en compte que les seves llicències han de ser compatibles, aplicant-se sempre l'aspecte més restrictiu, però en cap cas es podran fer servir si una obliga a fer una cosa i l'altra la contrària. Per exemple, les quatre llicències de programari comentades anteriorment són compatibles amb GPL.

Combinar dues imatges sota llicència Creative Commons

En el cas de tenir dues imatges sota llicència Creative Commons, si una és CC0 (domini públic) i l'altra és CC-BY (atribució), l'obra derivada podrà distribuir-se sempre que s'acrediti l'autor, perquè hi obliguen les restriccions CC-BY.

Combinar una llibreria amb llicència GPL i una altra de

> En aquest cas hi ha una incompatibilitat, ja que la llicència GPL ens obliga a distribuir el codi font del programari i la llicència del *software* primitiu no ens permet distribuir-lo (i segurament tampoc ens l'han facilitat).
>
> No seria possible combinar aquestes dues llibreries, i s'hauria de cercar una altra solució alternativa.

2.3.3. Recursos amb llicències fàcilment localitzables

A Internet podem trobar molts llocs de descàrregues, però només una minoria posen a la nostra disposició quines llicències s'apliquen a aquests recursos o si podeu reutilitzar-los pels vostres propis interessos.

Alguns d'aquests llocs estan dedicats a la venda d'aquests recursos, siguin dibuixos, música o fotografies, proporcionant una part de manera gratuïta per aconseguir més visites, i la resta de pagament (generalment lliures de regalies, *Royalty Free*).

Altres llocs on es poden trobar imatges amb llicències permissives són llocs dedicats a promocionar els treballs dels artistes.

A continuació teniu una llista de llocs on podeu trobar diferents tipus de recursos que o bé mostren les llicències d'aquests, en gran part permissives, o són de domini públic:

- **Fonts**:
  - Google Fonts: www.google.com/fonts
  - Dafont: www.dafont.com
- **Música i sons**:
  - *as3sfxr* (generador de sons de 8 bits): www.superflashbros.net/as3sfxr
  - Freesound (sons): www.freesound.org/browse
  - Free Music Archive (música): www.freemusicarchive.org
  - incompetech (música): incompetech.com
- **Imatges**:
  - flaticon (imatges vectorials): www.flaticon.es
  - Unsplash (fotografies en alta resolució, CC0): unsplash.com
  - Behance: www.behance.net
  - DeviantArt: www.deviantart.com
- **Codi**:
  - CodePen (tots els *pens* són codi lliure): codepen.io
  - GitHub (consultar llicències per a cada projecte): github.com

DISSENY D'INTERFÍCIES WEB