



FACULTAD REGIONAL ROSARIO

Cátedra de: ENTORNOS GRÁFICOS  
"Conceptos Fundamentales de HTML 5"

Prof. Ing. Daniela E. Díaz

Prof. Ing. Julián Butti

UTN-FRRO

## Índice de contenidos:

Introducción .....	1
Historia .....	1
¿Qué es un lenguaje de marcado? .....	3
¿Para que sirve HTML5? ¿Cuáles son los principales usos? .....	3
Ventajas de usar HTML5 .....	4
DOCTYPE.....	6
CANVAS .....	7
AUDIO.....	12
VIDEO .....	14
Elementos HTML semánticos. ....	15
FORM (autofocus).....	17
FORM (placeholder) .....	19
FORM (required) .....	20
FORM (pattern).....	22
FORM (input type="email").....	26
FORM (input type="range") .....	27
FORM (input type="date"/"datetime-local"/"month"/"time","week") .....	31
FORM (input type="color") .....	37
FORM (input type="number") .....	37
FORM (input type="url") .....	39
FORM (input - datalist).....	40
FORM (novalidate) .....	42
WEB STORAGE (localStorage y sessionStorage) .....	44
GEOLOCATION (getCurrentPosition) .....	48
GEOLOCATION (mostrar en un mapa) .....	52
GEOLOCATION (tiempo de espera y captura de errores) .....	55
GEOLOCATION (watchPosition).....	59
DRAW AND DROP (dragstart, dragover, drop) .....	61
DRAW AND DROP (drag, dragend) .....	66
DRAW AND DROP (dragenter, dragleave).....	73

API FILE (lectura de archivo de texto local).....	78
API FILE (lectura de múltiples archivos de texto locales).....	82
API FILE (lectura de una imagen) .....	87
API FILE (drag and drop de una imagen del escritorio).....	91

## Introducción

**HTML 5** es la quinta revisión más importante que se hace al lenguaje HTML. En este trabajo, se introducen nuevas características para ayudar a los autores de aplicaciones Web, y se ha prestado especial atención a la definición de claros criterios de conformidad para los agentes de usuario (navegadores) en un esfuerzo por mejorar la interoperabilidad.

HTML es el lenguaje de marcado principal de la World Wide Web. Originalmente, HTML fue diseñado principalmente como un lenguaje para describir semánticamente documentos científicos. Su diseño general, sin embargo, ha permitido que se adapte, en los años posteriores, para describir varios otros tipos de documentos e incluso aplicaciones.

## Historia

Durante sus primeros cinco años (1990-1995), HTML pasó por una serie de revisiones y experimentó una serie de extensiones, alojadas principalmente en el CERN (*Organización Europea para la Investigación Nuclear*), y luego en el IETF (*Internet Engineering Task Force*).

Con la creación del W3C, el desarrollo de HTML cambió de lugar de nuevo. Un primer intento frustrado de extender el HTML en 1995 conocido como HTML 3.0 luego dio paso a un enfoque más pragmático conocido como HTML 3.2, que se completó en 1997. HTML 4.01 siguió rápidamente más tarde ese mismo año.

Al año siguiente, la membresía del W3C decidió dejar de evolucionar HTML y en su lugar comenzó a trabajar en un equivalente basado en XML, llamado XHTML. Este esfuerzo comenzó con una reformulación de HTML 4.01 en XML, conocido como XHTML 1.0, que no agregó nuevas funciones, excepto la nueva serialización, y que se completó en 2000. Después de XHTML 1.0, el enfoque del W3C se volvió más fácil para otros grupos de trabajo para extender XHTML, bajo el título de XHTML Modularization. Paralelamente, el W3C también trabajó en un nuevo lenguaje que no era compatible con los lenguajes HTML y XHTML anteriores, llamándolo XHTML 2.0.

Alrededor de la época en que la evolución de HTML se detuvo en 1998, parte de la API para HTML desarrollada por los proveedores de navegadores se especificaron y publicaron bajo el nombre DOM Level 1 (en 1998) y DOM Level 2 Core y DOM Level 2 HTML (comenzando en 2000 y culminando en 2003). Estos esfuerzos luego se agotaron, con algunas especificaciones DOM nivel 3 publicadas en 2004, pero el grupo de trabajo se cerró antes de que se completaran todos los borradores de nivel 3.

En 2003, la publicación de XForms, una tecnología que se posicionó como la próxima generación de formularios web, despertó un renovado interés en la evolución del HTML en sí mismo, en lugar de encontrar reemplazos para él. Este interés surgió de la constatación de que la implementación de XML como tecnología web se limitaba a

tecnologías completamente nuevas (como RSS y Atom más tarde), en lugar de un reemplazo de las tecnologías implementadas existentes (como HTML).

Una prueba de concepto para demostrar que era posible extender los formularios de HTML 4.01 para proporcionar muchas de las funciones que introdujo XForms 1.0, sin requerir que los navegadores implementaran motores de procesamiento que fueran incompatibles con las páginas web HTML existentes, fue el primer resultado de este interés renovado. En esta etapa inicial, aunque el borrador ya estaba disponible públicamente, y ya se estaban solicitando comentarios de todas las fuentes, la especificación solo estaba bajo los derechos de autor de Opera Software.

La idea de que la evolución de HTML debería reabrirse se probó en un taller W3C en 2004, donde se presentaron algunos de los principios que subyacen al trabajo HTML, así como el anteproyecto de propuesta antes mencionado que cubre solo las características relacionadas con formularios. El W3C conjuntamente por Mozilla y Opera. La propuesta fue rechazada debido a que la propuesta entraba en conflicto con la dirección previamente elegida para la evolución de la Web; el personal y la membresía del W3C votaron para continuar desarrollando sustitutos basados en XML en su lugar.

Poco después, Apple, Mozilla y Opera anunciaron conjuntamente su intención de continuar trabajando en el esfuerzo bajo el paraguas de un nuevo lugar llamado WHATWG. Se creó una lista de correo pública y el borrador se movió al sitio WHATWG. El derecho de autor se modificó posteriormente para ser propiedad conjunta de los tres proveedores, y para permitir la reutilización de la especificación.

El WHATWG se basó en varios principios básicos, en particular, que las tecnologías deben ser compatibles con versiones anteriores, que las especificaciones y las implementaciones deben coincidir incluso si esto significa cambiar la especificación en lugar de las implementaciones, y que las especificaciones deben ser lo suficientemente detalladas como para que las implementaciones interoperabilidad completa sin ingeniería inversa entre sí.

Este último requisito en particular requiere que el alcance de la especificación HTML incluya lo que se había especificado previamente en tres documentos separados: HTML 4.01, XHTML 1.1 y DOM Nivel 2 HTML. También significó incluir muchos más detalles de los que se habían considerado anteriormente como la norma.

En 2006, el W3C indicó su interés en participar en el desarrollo de HTML 5.0 después de todo, y en 2007 formó un grupo de trabajo autorizado para trabajar con WHATWG en el desarrollo de la especificación HTML. Apple, Mozilla y Opera permitieron al W3C publicar la especificación bajo los derechos de autor del W3C, manteniendo una versión con la licencia menos restrictiva en el sitio WHATWG.

Durante varios años, ambos grupos trabajaron juntos bajo el mismo editor: Ian Hickson. En 2011, los grupos llegaron a la conclusión de que tenían diferentes objetivos: el W3C quería trazar una línea en la arena para las características de una Recomendación HTML 5.0, mientras que WHATWG quería continuar trabajando en un estándar de vida para HTML, manteniendo continuamente la especificación y agregando nuevas características. A mediados de 2012, se presentó un nuevo equipo de edición en el W3C para que se encargue de crear una Recomendación HTML 5.0 y preparar un Borrador de Trabajo para la próxima versión HTML.

Desde entonces, W3C Web Platform WG ha estado recogiendo parches del WHATWG que resolvió errores registrados en la especificación W3C HTML o representó con mayor precisión la realidad implementada en los agentes de usuario. Hasta ahora los parches de la especificación HTML WHATWG se fusionaron hasta el 12 de enero de 2016. Los editores HTML del W3C también han agregado parches que resultaron de discusiones y decisiones tomadas por el W3C Web Platform WG, así como correcciones de errores de errores no compartidos por el WHATWG.

HTML5 es el principal lenguaje de la web. Es uno de los primeros lenguajes que debes dominar si deseas dedicarte al diseño y desarrollo web. Pero, ¿por qué es tan importante? ¿cuál es el alcance de este popular lenguaje? Estas y otras preguntas serán respondidas en este artículo, en el que comenzaremos explicando qué es el lenguaje HTML5 y por qué es tan importante en la actualidad.

### ¿Qué es un lenguaje de marcado?

Un **lenguaje de marcado** hace referencia a aquellos lenguajes que emplean etiquetas. Estas etiquetas ya están predefinidas dentro del lenguaje respectivo y contienen la información que “ayudan” a leer el texto. Es decir, tanto para los desarrolladores como para las plataformas que pueden leer este lenguaje, las etiquetas contienen información adicional de la estructura del texto.

*Su principal diferencia con los lenguajes de programación es que éstos últimos poseen funciones aritméticas o variables, mientras que los lenguajes de marcado no.*

### ¿Para que sirve HTML5? ¿Cuáles son los principales usos?

El lenguaje HTML5 se usa para definir la estructura básica de una página web. Sin embargo, una de sus más grandes adiciones en esta nueva versión es poder añadir audio y video sin necesidad de usar Flash u otro reproductor multimedia.

Por medio de las etiquetas <video> y <audio> de HTML5, permite añadir videos o audio sin necesidad de usar Adobe Flash o cualquier otro plugin de tercero. Toda la acción sucede desde el propio navegador, lo que puede ayudar a disminuir al tamaño del archivo final de tu página. Los desarrolladores pueden tener acceso a una API que les permitirá determinar cómo estas nuevas etiquetas son presentadas a los usuarios. En otras palabras, puedes incluir videos de presentación de algún producto, críticas en video, podcasts, muestras de música, etc. La adición de estas dos etiquetas expande el uso que le puedes dar al lenguaje HTML5.



También, pueden subirse videos a páginas de terceros como Vimeo o Youtube e incrustarlos en el nuevo sitio web, esta es una de las opciones más viables pues a pesar de colocar elementos multimedia, el peso final del archivo no se ve afectado.

La geolocalización permite al sitio detectar la ubicación de cada usuario que ingresa al sitio web. Esto puede tener diversos usos, por ejemplo, para ofrecer opción de idiomas según el lugar de ubicación del usuario o para enlazarlo a la página oficial de la marca en el país en el que se encuentra, entre otras opciones útiles que, dependiendo como la use, pueden mejorar la experiencia de usuario.



Es una característica con la que hay que tener bastante cuidado e informar al cliente al respecto pues de lo contrario, sería una violación a su privacidad. Es por ello que esta opción no se puede activar si el usuario no lo aprueba.

Con HTML5 se pueden crear animaciones en 2D gracias a la etiqueta <canvas>. La API para esta etiqueta permite dibujar elementos en 2D y animarlos. El resultado final bien podría incluirlo en la página de inicio del sitio web pues la API da bastante control sobre los elementos.

También la API permite añadir eventos de teclado, ratón y cualquier otro mando que desee incluir. Esta posibilidad ha emocionado a muchos desarrolladores que se han dedicado a realizar sus juegos en HTML5. Pudiendo jugar desde cualquier navegador.



La gran ventaja de desarrollar aplicaciones HTML5 es que el resultado final es completamente accesible, es decir, se puede acceder a esta aplicación desde un ordenador, tablet o móvil. Incluso al cambiar de dispositivo, se puede acceder a la aplicación web mediante la URL respectiva, cosa que no sucede con una aplicación móvil.



La gran parte de aplicaciones web funcionan desde la nube. Un ejemplo común son los clientes de correo como Gmail, que también cuenta con una aplicación móvil. Es probable que muchos usuarios probablemente prefieran la aplicación móvil, pero le da la facilidad a sus usuarios de elegir la opción que les atraiga más.

## **Ventajas de usar HTML5**

### **Es gratuito**

No necesita ningún tipo de programa especial para empezar a programar en HTML5, incluso puede hacerlo en un bloc de notas, guardar el documento como HTML y podrá visualizarlo desde cualquier navegador. Sin embargo, aunque esto es posible no es realmente recomendable pues en un bloc de notas no separa las etiquetas del contenido y puede ser más complicado realizar correcciones.

Pero no necesita ningún software costoso, puede usar un editor de código gratuito como Notepad++ que ofrece funciones básicas como diferenciación por color entre etiquetas y contenido.

### **Código más ordenado**

Debido a la adición de nuevas etiquetas que ayudan a nombrar partes de la estructura básica de toda página web (como <header>, por ejemplo), así como la eliminación de ciertas etiquetas, el código HTML se puede separar fácilmente entre etiquetas y contenido, permitiendo así que el desarrollador pueda trabajar de manera más efectiva y detectar errores de manera más rápida.

Las etiquetas son claras y descriptivas, de modo que el desarrollador puede comenzar a codificar sin ningún problema. Es realmente un lenguaje bastante sencillo de comprender en esta nueva versión.

### **Compatibilidad en navegadores**

Los navegadores modernos y populares como Chrome, Firefox, Safari y Opera soportan HTML5, es decir, sin importar qué navegador empleen los usuarios el contenido se puede visualizar correctamente. El único problema sería considerar a usuarios que emplean navegadores más antiguos, ya que en éstos no todas las nuevas funciones y etiquetas de HTML5 están disponibles.

### **Almacenamiento mejorado**

Otra nueva adición en HTML5 ha sido el almacenamiento local que se define a sí mismo como “mejor que las cookies” pues la información nunca se transfiere al servidor. De esta manera, la información se mantiene segura. Asimismo, esta nueva característica permite que la información se mantenga almacenada incluso después de haber cerrado el navegador y como funciona desde el lado de cliente, la información se mantiene a salvo incluso si el usuario decide borrar sus cookies.

Ya que la información se guarda en el navegador del usuario, da muchas más posibilidad a las aplicaciones web como por ejemplo el uso de caché que mejora el tiempo de respuesta de la aplicación.

### **HTML5 y el diseño adaptativo**

HTML5 es compatible con los navegadores móviles, de modo que cada página realizada en HTML5 que se ve en ordenadores, también se puede adaptar a los dispositivos móviles. Esta especificación para móviles puede hacerla desde el mismo documento HTML o puede emplear una framework especializada para mejorar tu productividad.

Esta característica del lenguaje HTML5 es probablemente una de las más útiles, pues se puede acceder a cualquier página o aplicación web desde un dispositivo móvil y permite que la experiencia sea igual de buena que al visitar una página web en ordenadores.



## DOCTYPE

La declaración del tipo de documento (DTD Document Type Declaration), esta sección se ubica en la primera línea del archivo HTML, es decir antes de la marca html.

Según el rigor de HTML 4.01 utilizado podemos declararla como:

Declaración transitoria:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  
"http://www.w3.org/TR/html4/loose.dtd">
```

Declaración estricta:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">
```

Ahora con el HTML 5 se simplifica esta sección con la siguiente sintaxis:

```
<!DOCTYPE HTML>
```

Es importante agregar el DOCTYPE, de esta forma el navegador puede saber que estamos utilizando la especificación del HTML 5.

**Ejemplo: Confeccionar una página que muestre el DOCTYPE propuesto para el HTML5.**

```
<!DOCTYPE HTML>  
  
<html>  
  
<head>  
  
  <title>Título de la página</title>  
  
  <meta charset="UTF-8">  
  
</head>  
  
<body>
```

<p>Si vemos el código fuente de esta página veremos la forma de declarar el DOCTYPE en HTML5.</p>

</body>

</html>

## CANVAS

**CANVAS** es una nueva etiqueta del HTML 5, permite dibujar en dicha área mediante JavaScript. El objetivo de este elemento es hacer gráficos en el cliente (navegador), juegos etc. Para hacer un uso efectivo de este nuevo elemento de HTML tenemos que manejar el JavaScript.

La estructura de un programa que accede al **canvas** es:

<html>

<!DOCTYPE HTML>

<head>

<title>Título de la página</title>

<meta charset="UTF-8">

<script type="text/javascript">

```
function retornarLienzo(x)
{
    var canvas = document.getElementById(x);
    if (canvas.getContext)
    {
        var lienzo = canvas.getContext("2d");
        return lienzo;
    }
    else
```

```
        return false;
    }

function dibujar()
{
    var lienzo=retornarLienzo("lienzo1");
    if (lienzo)
    {
        lienzo.fillStyle = "rgb(200,0,0)";
        lienzo.fillRect (10, 10, 100, 100);
    }
}

</script>
</head>
<body onLoad="dibujar()">
<canvas id="lienzo1" width="300" height="200">
Su navegador no permite utilizar canvas.
</canvas>
</body>
</html>
```

La primer función que implementaremos y será de uso común en todos los problemas es la que retorna la referencia al objeto que nos permite dibujar en el canvas:

```
function retornarLienzo(x)
{
    var canvas = document.getElementById(x);
    if (canvas.getContext)
```

```
{  
    var lienzo = canvas.getContext("2d");  
    return lienzo;  
}  
  
else  
    return false;  
}
```

La función recibe el "id" que hemos especificado en el elemento canvas dispuesto en la página (en este ejemplo le llamamos lienzo1):

```
<canvas id="lienzo1" width="300" height="200">
```

Su navegador no permite utilizar canvas.

```
</canvas>
```

En caso que el navegador no implemente la especificación del elemento canvas produce un falso el if (lo que estamos haciendo en este if es comprobar si existe la función getContext, en caso afirmativo el navegador soporta canvas):

```
if (canvas.getContext)
```

En caso de implementar el canvas obtenemos una referencia del mismo llamando a la función getContext y pasando como parámetro un string con el valor "2d":

```
var lienzo = canvas.getContext("2d");
```

La función dibujar es la que se ejecuta luego que se carga la página en el navegador:

```
function dibujar()  
{  
    var lienzo=retornarLienzo("lienzo1");  
    if (lienzo)
```

```
{  
    lienzo.fillStyle = "rgb(200,0,0)";  
    lienzo.fillRect (10, 10, 100, 100);  
}  
}
```

En esta función llamamos a la anterior "retornarLienzo("lienzo1") pasando como parámetro el id del canvas respectivo. En caso que la función retorne la referencia al lienzo el if se verificará como verdadero y estaremos en condiciones de comenzar a dibujar en el mismo:

```
var lienzo=retornarLienzo("lienzo1");  
  
if (lienzo)
```

Mediante la variable lienzo podemos acceder a un conjunto de funciones y propiedades disponibles para dibujar.

En el ejemplo:

```
lienzo.fillStyle = "rgb(200,0,0)";  
  
lienzo.fillRect (10, 10, 100, 100);
```

Activamos como color de relleno de figura el rojo (200,0,0) y seguidamente llamamos a la función fillRect que dibuja un rectángulo desde la posición (10,10) con ancho de 100 píxeles y un alto de 100 píxeles. La coordenada (0,0) se encuentra en la parte superior izquierda

**Ejemplo: Obtener el contexto gráfico del elemento canvas y seguidamente dibujar un cuadrado de color rojo.**

```
<!DOCTYPE HTML>  
  
<html>  
  
<head>  
  
    <title>Título de la página</title>  
  
    <meta charset="UTF-8">
```

```
<script type="text/javascript">

function retornarLienzo(x)
{
    var canvas = document.getElementById(x);
    if (canvas.getContext)
    {
        var lienzo = canvas.getContext("2d");
        return lienzo;
    }
    else
        return false;
}

function dibujar()
{
    var lienzo=retornarLienzo("lienzo1");
    if (lienzo)
    {
        lienzo.fillStyle = "rgb(200,0,0)";
        lienzo.fillRect (10, 10, 100, 100);
    }
}

</script>

</head>

<body onLoad="dibujar()">
```

```
<canvas id="lienzo1" width="300" height="200">
```

Su navegador no permite utilizar canvas.

```
</canvas>
```

```
</body>
```

```
</html>
```

## AUDIO

Otro elemento que se agrega al HTML5 es el AUDIO. El objetivo de esta etiqueta es permitir la carga y ejecución de archivos de audio sin requerir un plug-in de Flash, Silverlight o Java.

El comité de estandarización W3C deja abierto a cada empresa que desarrolla navegadores los formatos que quieran soportar (así tenemos que algunos soportan mp3, wav, ogg, au)

Un ejemplo de disponer el elemento audio dentro de una página sería:

```
<audio src="sonido.ogg" autoplay controls loop></audio>
```

Las propiedades que podemos utilizar con la marca audio son:

**src:** La URL donde se almacena el archivo de audio. Si no definimos la URL la busca en el mismo directorio donde se almacena la página.

**autoplay:** En caso de estar presente el archivo se ejecuta automáticamente luego de cargarse la página sin requerir la intervención del visitante.

**loop:** El archivo de audio se ejecuta una y otra vez.

**controls:** Indica que se deben mostrar la interface visual del control en la página (este control permite al visitante arrancar el audio, detenerlo, desplazarse etc.)

**autobuffer:** En caso de estar presente indica que primero debe descargarse el archivo en el cliente antes de comenzar a ejecutarse.

Como no hay un formato de audio universalmente adoptado por todos los navegadores el elemento audio nos permite agregarle distintas fuentes:

```
<audio controls autoplay loop>
```

```
  <source src="sonido.ogg">
```

```
  <source src="sonido.mp3">
```

```
  <source src="sonido.wav">
```

```
<source src="sonido.au">  
  
</audio>
```

El elemento **source** indica a través de la propiedad **src** la ubicación del archivo de audio respectivo. El orden que disponemos estas fuentes es importante. Primero el navegador busca la primera fuente y verifica que puede reproducir dicho archivo, en caso negativo pasa a la siguiente fuente.

Una página que muestra el control de audio:

```
<!DOCTYPE HTML>  
  
<html>  
  
<head>  
  
  <title>Título de la página</title>  
  
  <meta charset="UTF-8">  
  
</head>  
  
<body>  
  
  <audio controls>  
  
    <source src="http://www.tutorialesprogramacionya.com/audios/gallo.ogg">  
  
    <source src="http://www.tutorialesprogramacionya.com/audios/gallo.mp3">  
  
    <source src="http://www.tutorialesprogramacionya.com/audios/gallo.wav">  
  
  </audio>  
  
</body>  
  
</html>
```

Según el navegador el control de reproducción de audio puede variar.

En el caso que solo necesitemos reproducir un único formato de archivo podemos evitar los elementos HTML "source" y disponer la propiedad src directamente en la etiqueta "audio". Por ejemplo:

```
<audio controls src="http://www.tutorialesprogramacionya.com/audios/gallo.mp3">  
  
</audio>
```



## VIDEO

El elemento VIDEO permite mostrar un video sin la necesidad de plugin (Flash). En este momento los navegadores permiten mostrar formatos como el mp4, webm y ogv.

FireFox permite mostrar videos en formato ogv (formato de vídeo de código abierto Ogg/Theora).

Luego para visualizar un video con este formato en FireFox tenemos:

```
<video width="640" height="360"
src="http://videos.mozilla.org/firefox/3.5/overview/overview.ogv"
controls>
```

Este navegador no permite tag video

```
</video>
```

Las propiedades más importantes de la marca video son:

**src:** Dirección donde se almacena el video.

**controls:** Se visualiza el panel de control del video: botón de inicio, barra de avance del video etc.

**autoplay:** El video se inicia inmediatamente luego que la página se carga en el navegador.

**width:** Ancho en píxeles del video.

**height:** Alto en píxeles del video.

Como no hay un formato de video universalmente adoptado por todos los navegadores el elemento video nos permite agregarle distintas fuentes:

```
<video width="640" height="360" controls>
<source src="http://videos.mozilla.org/firefox/3.5/overview/overview.ogv">
<source src="http://videos.mozilla.org/firefox/3.5/overview/overview.mp4">
</video>
```

Esto es similar al elemento AUDIO-

La siguiente página muestra un video llamado "video1.mp4":

```
<!DOCTYPE HTML>
```

```
<html>

<head>

  <title>Título de la página</title>

  <meta charset="UTF-8">

</head>

<body>

<video width="640" height="360"

  controls

  src="http://www.tutorialesprogramacionya.com/videos/video1.mp4">

</video>

</body>

</html>
```

Nuevamente según el navegador nos muestra una interfaz visual distinta, pero nos permite reproducir el video.

### Elementos HTML semánticos.

El HTML 5 introduce una serie de elementos estructurales que facilitarán tanto el desarrollo de las páginas como también el análisis de las mismas por buscadores.

Los elementos de HTML 5 que ayudan a estructurar la parte semántica de la página son:

**header:** El elemento header debe utilizarse para marcar la cabecera de una página (contiene el logotipo del sitio, una imagen, un cuadro de búsqueda etc)

El elemento **header** puede estar anidado en otras secciones de la página (es decir que no solo se utiliza para la cabecera de la página)

**nav:** El elemento **nav** representa una parte de una página que enlaza a otras páginas o partes dentro de la página. Es una sección con enlaces de navegación.

No todos los grupos de enlaces en una página deben ser agrupados en un elemento **nav**. únicamente las secciones que consisten en bloques de navegación más importantes son adecuados para el elemento de navegación.

**section:** El elemento **section** representa un documento genérico o de la sección de aplicación. Una sección, en este contexto, es una agrupación temática de los contenidos. Puede ser un capítulo, una sección de un capítulo o básicamente cualquier cosa que incluya su propio encabezado.

Una página de inicio de un sitio Web puede ser dividida en secciones para una introducción, noticias, información de contacto etc.

**footer:** El elemento **footer** se utiliza para indicar el pie de la página o de una sección. Un pie de página contiene información general acerca de su sección el autor, enlaces a documentos relacionados, datos de derechos de autor etc.

**aside:** El elemento **aside** representa una nota, un consejo, una explicación. Esta área son representados a menudo como barras laterales en las revistas impresas.

El elemento puede ser utilizado para efectos de atracción, como las comillas tipográficas o barras laterales, para la publicidad, por grupos de elementos de navegación, y por otro contenido que se considera por separado del contenido principal de la página.

**article:** El elemento **article** representa una entrada independiente en un blog, revista, periódico etc.

Cuando se anidan los elementos **article**, los artículos internos están relacionados con el contenido del artículo exterior. Por ejemplo, una entrada de blog en un sitio que acepta comentarios, el elemento **article** principal agrupa el artículo propiamente dicho y otro bloque **article** anidado con los comentarios de los usuario.

Un ejemplo de página añadiendo estos elementos semánticos sería:

```
<!DOCTYPE html>

<html>

<head>

  <meta charset="UTF-8">

  <title>Elementos semánticos del HTML5</title>

</head>

<body>

  <header>

    <h1>Encabezado de la página</h1>

  </header>
```

```
<nav>

  <p>enlaces de navegación</p>

</nav>

<section>

  <p>Sección 1</p>

</section>

<section>

  <p>Sección 2</p>

</section>

<aside>

  <p>Publicidad</p>

</aside>

<footer>

  <p>Pié de página</p>

</footer>

</body>

</html>
```

### FORM (autofocus)

En HTML5 disponemos una propiedad en los controles de los formularios que nos permiten indicar cual control tendrá foco (es decir que aparecerá el cursor en su interior)

Anteriormente esta actividad se la realizaba mediante Javascript, pero como vemos varios tipos de problemas ahora se los puede resolver directamente con HTML.

Para que aparezca un control con foco lo único que tenemos que hacer es disponer la propiedad **autofocus** sin asignarle valor.

Ejemplo: Confeccionar un problema que solicite la carga del nombre de usuario, una clave y la edad. Hacer que cuando aparezca el formulario en pantalla automáticamente tome foco el control donde se debe ingresar la edad.

```
<!DOCTYPE HTML>

<html>

<head>

  <title>Título de la página</title>

  <meta charset="UTF-8">

</head>

<body>

  <form action="#">

    Usuario:

    <input type="text" id="usuario">

    <br>

    Clave:

    <input type="password" id="clave">

    <br>

    Edad:

    <input type="text" id="edad" autofocus>

    <br>

    <input type="submit" value="Confirmar">

  </form>

</body>

</html>
```

El resultado en el navegador al cargar esta página es:

Se puede observar que lo único que se ha agregado al control donde se ingresa la edad es la propiedad **autofocus**:

```
<input type="text" id="edad" autofocus>
```

Si a ningún control se agrega la propiedad autofocus por defecto no aparece ningún control con foco y será el usuario que tendrá que seleccionar uno.

Solo tiene sentido incluir la propiedad **autofocus** a un único control de una página web, sino el resultado es impredecible.

## FORM (placeholder)

Normalmente cuando creamos un formulario previo a cada control disponemos un mensaje indicando qué dato debe cargar el usuario en el mismo, en algunas circunstancias dicha información puede quedar no lo suficientemente clara.

Mediante la propiedad **placeholder** podemos disponer más información directamente dentro del control generalmente con un ejemplo de dato a ingresar. El contenido del control HTML inicial se borra inmediatamente luego que el operador ingresa un caracter.

Es importante tener en cuenta que debemos utilizar la propiedad **placeholder** si la información que disponemos en ella despeja de dudas al operador.

Ejemplo: Confeccionar un formulario que solicite la carga de la patente de un automóvil, tener en cuenta que toda patente está formada por dos letras, tres números y finalmente dos letras:

```
<!DOCTYPE HTML>

<html>

<head>

  <title>Título de la página</title>

  <meta charset="UTF-8">

</head>

<body>

  <form action="#">

    Patente:

    <input type="text" id="patente" placeholder="AB123CD">

    <br>

    <input type="submit" value="Confirmar">
```

```
</form>
```

```
</body>
```

```
</html>
```

Al control que deseamos agregar esta marca de agua (o mensaje inicial) inicializamos la propiedad **placeholder**:

```
<input type="text" id="patente" placeholder="AB123CD">
```

### FORM (required)

Otra facilidad que nos proporciona el HTML5 es la validación de contenido obligatorio de un control.

Mediante la propiedad **required** obligamos e informamos al operador que el control se debe cargar obligatoriamente (por ejemplo el nombre de usuario, clave, estudios etc.)

Esto se hacía únicamente con Javascript pero ahora agregando la propiedad **required** al control la validación de contenido se hace automáticamente.

Nuevamente estamos en presencia de una propiedad que no requiere que le asignemos un valor, con solo hacer referencia a la misma dentro del control el navegador verificará que su contenido esté cargado previo a enviarlo al servidor.

Ejemplo: Confeccionar una página que solicite la carga del nombre de usuario y una clave. No permitir enviar los datos al servidor si alguno de los controles está vacío, emplear para esto la propiedad **required**.

```
<!DOCTYPE HTML>
```

```
<html>
```

```
<head>
```

```
<title>Título de la página</title>
```

```
<meta charset="UTF-8">
```

```
</head>
```

```
<body>
```

```
<form action="#">
```

Usuario:

```
<input type="text" id="usuario" required>
```

```
<br>
```

Clave:

```
<input type="password" id="clave" required>  
  
<br>  
  
<input type="submit">  
  
</form>  
  
</body>  
  
</html>
```

Podemos observar que solo hemos agregado la propiedad **required** a cada control HTML que necesitamos que el operador no deje vacío:

Usuario:

```
<input type="text" id="usuario" required>  
  
<br>
```

Clave:

```
<input type="password" id="clave" required>
```

Luego si el operador carga el nombre e intenta dejar la clave vacía al presionar el botón para enviar los datos al servidor (submit) el navegador se encargará de mostrar un mensaje indicando que no debe dejar el campo de la clave vacío:



The screenshot shows a web form with two input fields. The first field is labeled 'Usuario:' and contains the text 'Diego'. The second field is labeled 'Clave:' and is empty. Below the 'Clave:' field, there is a red-bordered box with a yellow exclamation mark icon and the text 'Completa este campo'. To the left of the 'Clave:' field, there is a button labeled 'Enviar'.

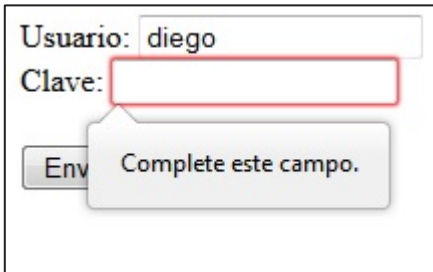


El mensaje que aparece e informa al operador es propio de cada navegador, pero no deja ninguna duda de la acción que debe tomar el operador.

En el Chrome:



En el Firefox:



## FORM (pattern)

Una expresión regular es una técnica que nos permite identificar si un string coincide con un patrón determinado. Un ejemplo clásico es determinar si un email está correctamente estructurado (sabemos que comienza con uno o más caracteres alfanuméricos, luego el carácter @ seguido de uno o más caracteres alfanuméricos, luego el carácter "." y finalmente otros caracteres alfabéticos)

Hay muchas situaciones donde el operador debe cargar datos que tienen que tener una determinada estructura.

Hasta hace poco solo esto podía resolverse empleando Javascript, pero ahora el HTML5 añade a los controles de formulario la propiedad **pattern** donde definimos el patrón que debe seguir la cadena que cargue el operador y en caso que no cumpla dicho patrón informar un mensaje de error.

Veremos que para definir expresiones regulares intervienen una serie de caracteres . \* + ? = ! : | \ / ( ) [ ] { } que tienen un significado especial en la definición del lenguaje de expresiones regulares, estos tipos de caracteres suelen llamarse **metacaracteres**.

El tema de expresión regular es bastante amplio y complejo en un principio. Lo más adecuado es iniciar con pequeños ejemplos que nos muestren el funcionamiento y la sintaxis.

Ejemplo: Confeccionar un formulario que nos solicite la carga obligatoria de un número binario de ocho dígitos.

```
<!DOCTYPE HTML>

<html>

<head>

  <title>Título de la página</title>

  <meta charset="UTF-8">

</head>

<body>

  <form action="#">

    Ingrese un número binario de ocho dígitos:

    <input type="text" id="numero" required pattern="[01]{8}">

    <br>

    <input type="submit">

  </form>

</body>

</html>
```

Entre corchetes encerramos los caracteres permitidos, en este caso [01] indicamos que los dos únicos caracteres posibles de ingresar dentro del control son el cero y el uno. Seguidamente indicamos entre llaves la cantidad de caracteres que se deben ingresar, en nuestro ejemplo indicamos {8}

También hay que tener muy en cuenta que debemos agregar la propiedad **required**. Si no agregamos esta propiedad y el operador deja vacío el control no valida la expresión regular que hemos dispuesto. Es decir si dejamos vacío el control no se analiza si se ingresó un número binario de ocho dígitos.

Ejemplo: Confeccionar un formulario que nos solicite la carga de un número entero decimal que contenga 5 dígitos.

```
<!DOCTYPE HTML>

<html>

<head>

  <title>Título de la página</title>

  <meta charset="UTF-8">

</head>

<body>

  <form action="#">

    Ingrese un número entero que contenga 5 dígitos:

    <input type="text" id="numero" required pattern="[0-9]{5}">

    <br>

    <input type="submit">

  </form>

</body>

</html>
```

Si bien podíamos enumerar del cero al nueve los caracteres posibles, es más cómodo indicar un rango de caracteres [0-9] utilizando el guión.

La expresión regular queda:

```
<input type="text" id="numero" required pattern="[0-9]{5}">
```

Ejemplo: Confeccionar un formulario que nos solicite la carga de un número entero decimal que contenga entre 1 y 5 dígitos.

```
<!DOCTYPE HTML>

<html>

<head>

  <title>Título de la página</title>

  <meta charset="UTF-8">

</head>

<body>
```

```
<form action="#">
```

Ingrese un número entero decimal que contenga entre 1 y 5 dígitos:

```
<input type="text" id="numero" required pattern="[0-9]{1,5}">
```

```
<br>
```

```
<input type="submit">
```

```
</form>
```

```
</body>
```

```
</html>
```

En este problema vemos que cuando queremos cargar una cantidad no exacta de valores indicamos entre llaves primero el valor mínimo y luego de una coma el valor máximo:

```
<input type="text" id="numero" required pattern="[0-9]{1,5}">
```

Ejemplo: Confeccionar un formulario que nos solicite la carga de una patente de un auto. Toda patente cuenta con tres caracteres seguidos de tres números

```
<!DOCTYPE HTML>
```

```
<html>
```

```
<head>
```

```
<title>Título de la página</title>
```

```
<meta charset="UTF-8">
```

```
</head>
```

```
<body>
```

```
<form action="#">
```

Ingrese su patente:

```
<input type="text" id="patente" required pattern="[A-Za-z]{3}[0-9]{3}">
```

```
<br>
```

```
<input type="submit" value="confirmar">
```

```
</form>
```

```
</body>
```

```
</html>
```

Primero debemos indicar entre corchetes que podemos ingresar cualquier caracter alfabético en mayúsculas o minúsculas, luego entre llaves indicamos que solo se pueden cargar 3 de dichos caracteres. Seguimos nuevamente entre corchetes indicando que se puede cargar números entre 0 y 9, indicando entre llaves que deben ser exactamente 3.

### FORM (input type="email")

El HTML5 trae una nueva serie de controles de formulario que complementan los existentes en HTML.

Para utilizar estos nuevos controles hay que definir el tipo en la propiedad **type** del **control input**.

Normalmente cuando queremos ingresar un email utilizamos el control:

```
<input type="text" id="mailusuario">
```

Luego para validarlo en el navegador implementamos una función en Javascript.

En HTML5 tenemos un control especializado para el ingreso de un email por teclado que tiene asociado toda la lógica para validar el ingreso correcto del dato.

Ejemplo: Ingresar por teclado un email verificando su correcta sintaxis.

```
<!DOCTYPE HTML>
```

```
<html>
```

```
<head>
```

```
<title>Título de la página</title>
```

```
<meta charset="UTF-8">
```

```
</head>
```

```
<body>
```

```
<form action="#">
```

Mail:

```
<input type="email" id="emailusuario" required>
```

```
<br>
```

```
<input type="submit" value="Confirmar">

</form>

</body>

</html>
```

Como vemos la sintaxis para el ingreso de un email es utilizar un elemento input y asignar a la propiedad **type** el valor "**email**". También tenemos que agregar la propiedad **required** si queremos que no deje pasar un dato vacío dentro del control.

Si ingresamos un mail con una estructura incorrecta (por ejemplo nos olvidamos el @, el navegador se encargará de informar al usuario que ha cargado un mail mal formado.

### FORM (input type="range")

Otro control que agrega el HTML5 es una barra de selección de un valor.

En lugar de utilizar un control input de tipo text para cargar un número podemos seleccionar el valor con el mouse o con el dedo según el dispositivo que estamos utilizando.

Para definir una barra de selección inicializamos la propiedad **type** con el valor **range**. Luego podemos inicializar las propiedades **min** y **max** para fijar el valor mínimo y el máximo.

Ejemplo: Solicitar la carga de una temperatura entre 0 y 100, utilizar una barra de selección.

```
<!DOCTYPE HTML>

<html>

<head>

  <title>Título de la página</title>

  <meta charset="UTF-8">

</head>

<body>

  <form action="#">
```

    Seleccione una temperatura:

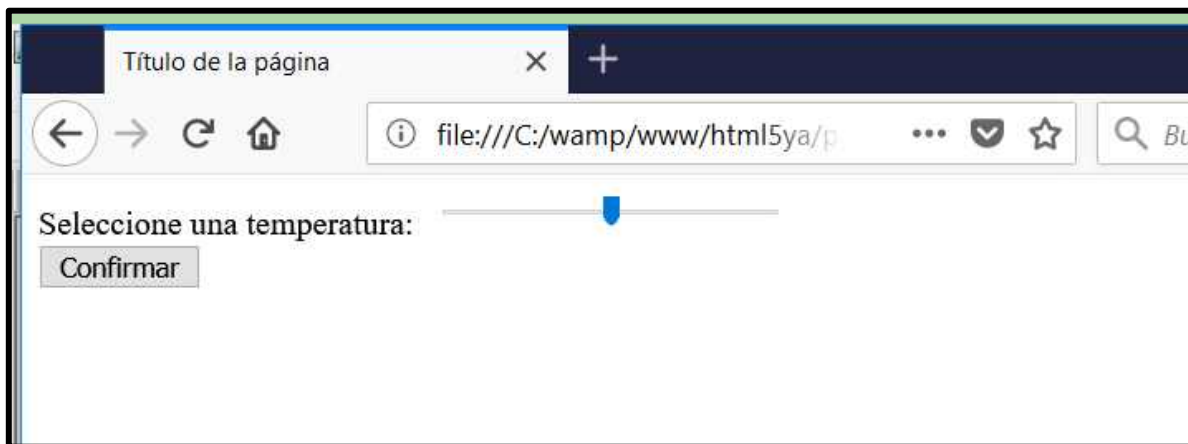
```
<input type="range" id="temperatura" min="0" max="100">  
<br>  
<input type="submit" value="Confirmar">  
</form>  
</body>  
</html>
```

Como podemos ver definimos el elemento input inicializando la propiedad **type** con el valor **range**, algo similar hacemos con las propiedades **min** y **max**:

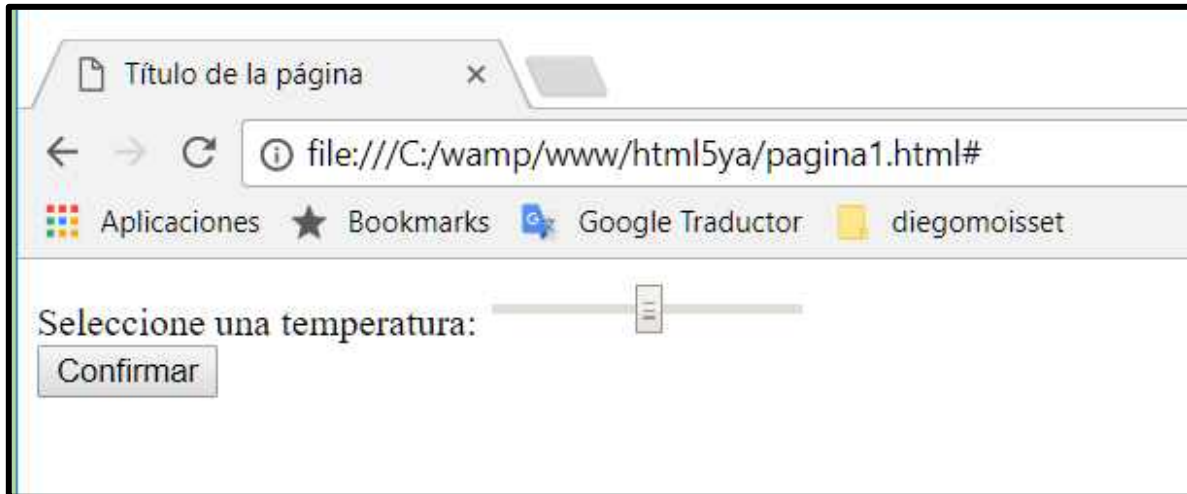
```
<input type="range" id="temperatura" min="0" max="100">
```

Si bien todos los navegadores modernos implementan este control la forma de mostrarlos es diferente

Firefox:



Chrome:



Ejemplo: Solicitar la carga de una temperatura entre 0 y 100, utilizar una barra de selección. Mostrar en un elemento span el valor seleccionado en el range.

```
<!DOCTYPE HTML>
```

```
<html>
```

```
<head>
```

```
<title>Título de la página</title>
```

```
<meta charset="UTF-8">
```

```
<script>
```

```
addEventListener('load',inicio,false);
```

```
function inicio()
```

```
{
```

```
document.getElementById('temperatura').addEventListener('change',cambioTemperatura,false);
```

```
}
```

```
function cambioTemperatura()
```

```
{
```



```
document.getElementById('temp').innerHTML=document.getElementById('temperatura').value;

}

</script>

</head>

<body>

  <form action="#">

    Seleccione una temperatura:

    <input type="range" id="temperatura" min="0" max="100">

    <span id="temp">0</span>

    <br>

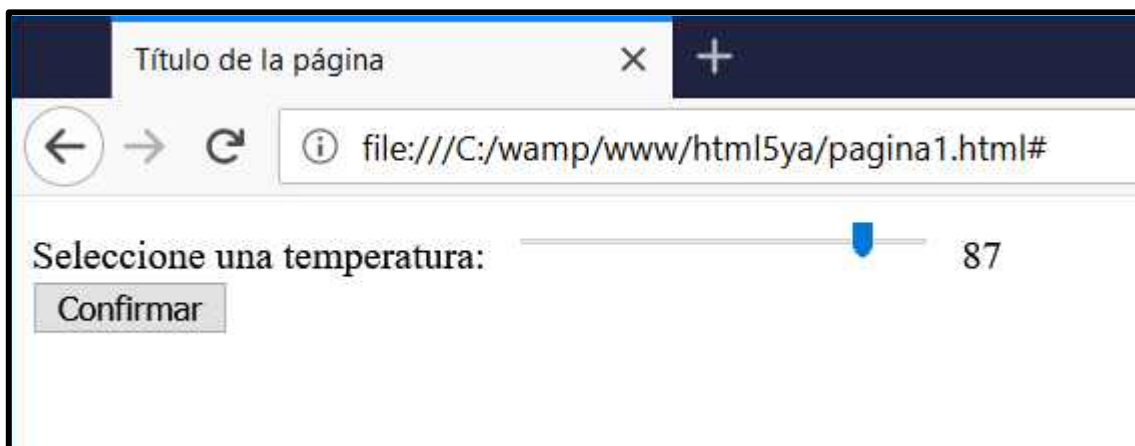
    <input type="submit" value="Confirmar">

  </form>

</body>

</html>
```

El resultado en el navegador al cargar esta página es:



Como vemos disponemos un bloque de Javascript para capturar el evento change de la barra de selección. Cuando se dispara este evento se ejecuta la función

cambioTemperatura donde modificamos el contenido del elemento span con el valor actual de la barra de selección:

```
<script>

  addEventListener('load',inicio,false);

  function inicio()

  {

document.getElementById('temperatura').addEventListener('change',cambioTemperatura,false);

  }

  function cambioTemperatura()

  {

document.getElementById('temp').innerHTML=document.getElementById('temperatura').value;

  }

</script>
```

### FORM (input type="date"/"datetime-local"/"month"/"time","week")

Para la entrada de un dato de tipo fecha podemos definir un control de formulario asignando en la propiedad "**type**" del control "**input**" el valor "**date**":

```
<input type="date">
```

Luego los navegadores modernos cuando el operador seleccione el control aparecerá un calendario para efectuar la selección de la fecha en forma más sencilla.

Ejemplo: Solicitar la carga de dos fecha utilizando el control de tipo "date".

```
<!DOCTYPE HTML>
```

```
<html>
```

```
<head>

  <title>Título de la página</title>

  <meta charset="UTF-8">

</head>

<body>

  <form action="#">

    <p>Primer fecha:

    <input type="date" id="fecha1" name="fecha1">

    </p>

    <p>

    Segundo fecha:

    <input type="date" id="fecha2" name="fecha2">

    </p>

    <p><input type="submit" value="Confirmar"></p>

  </form>

</body>

</html>
```

Dependiendo el navegador mostrará un calendario con un formato particular.

Como ejemplo en Chrome tenemos la siguiente interfaz:

Título de la página x

file:///C:/wamp/www/html5ya/pagina1.html

Primer fecha: 01/01/2018

Segundo fecha: dd/mm/aaaa ▼

Confirmar

enero de 2018 ▼

lu.	ma.	mi.	ju.	vi.	sá.	do.
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4

### datetime-local

Si necesitamos cargar una fecha y una hora podemos utilizar un control **input** con el valor "**datetime-local**":

```
<input type="datetime-local" id="fecha1" name="fecha1">
```

En Chrome aparece con el formato:

Título de la página

file:///C:/wamp/www/html5ya/pagina1.html

Primer fecha: 01/01/2018 13:15

Segundo fecha: dd/mm/aaaa --:--

Confirmar

enero de 2018

lu.	ma.	mi.	ju.	vi.	sá.	do.
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4

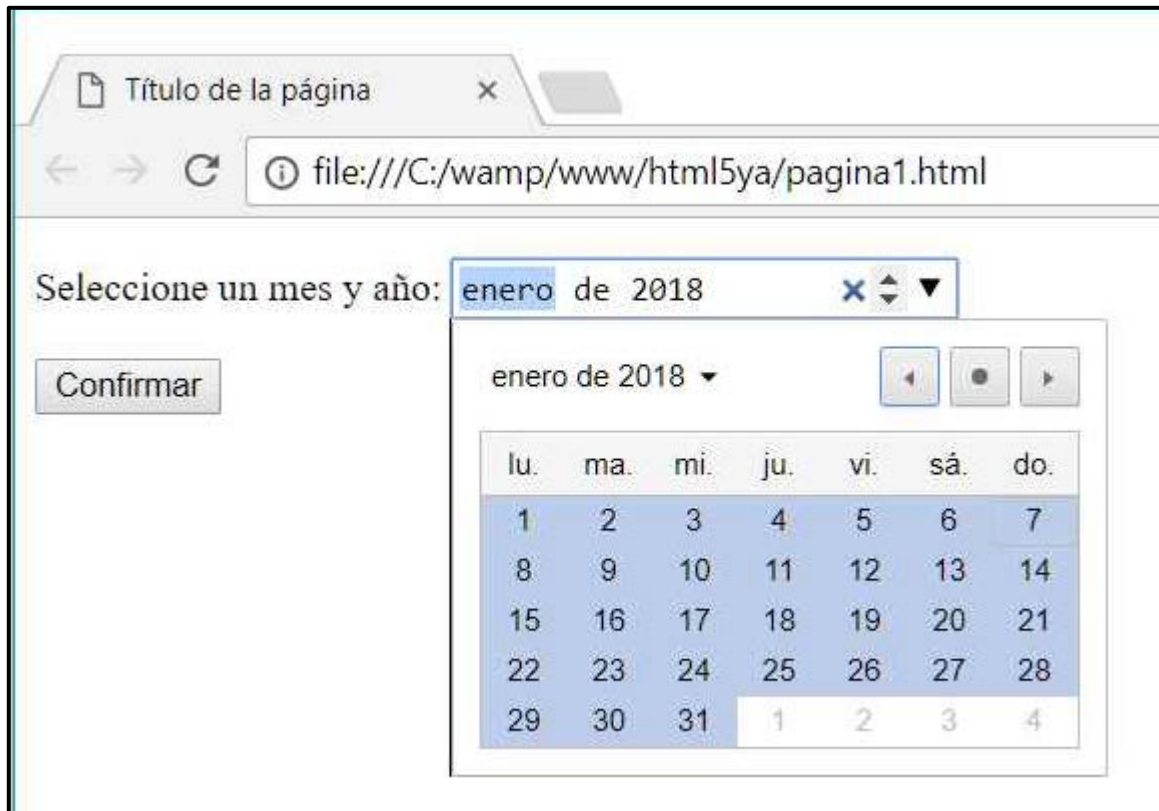
## month

Si necesitamos cargar un mes y año podemos utilizar en la propiedad **type** el valor **"month"**:

Seleccione un mes y año:

```
<input type="month" id="mes1" name="mes1">
```

En Chrome aparece una interfaz similar a esta:



Hasta enero de 2018 FireFox no ha implementado esta funcionalidad.

### time

Si necesitamos cargar una hora cualquiera podemos utilizar en la propiedad **type** el valor **"time"**:

<p>Ingrese hora:

<input type="time" id="hora" name="hora"></p>

Los navegadores principales muestran una interfaz similar a esta:



Título de la página x

file:///C:/wamp/www/html5ya/pagina1.html

Ingrese hora: 12:05 x

Confirmar

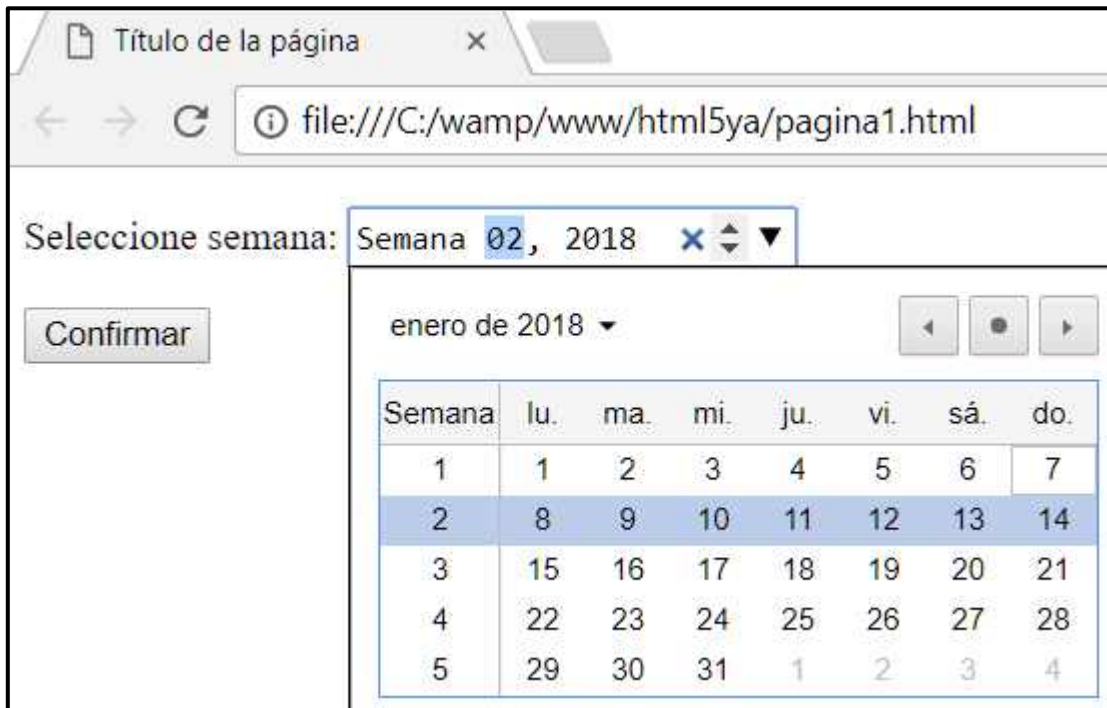
### week

Si necesitamos seleccionar una semana del año podemos utilizar en la propiedad **type** el valor **"week"**:

<p>Semana:

<input type="week" id="semana" name="semana"></p>

Chrome muestran una interfaz similar a esta:



Título de la página x

file:///C:/wamp/www/html5ya/pagina1.html

Seleccione semana: Semana 02, 2018 x

Confirmar

enero de 2018

Semana	lu.	ma.	mi.	ju.	vi.	sá.	do.
1	1	2	3	4	5	6	7
2	8	9	10	11	12	13	14
3	15	16	17	18	19	20	21
4	22	23	24	25	26	27	28
5	29	30	31	1	2	3	4

Firefox a enero de 2018 no implementa esta funcionalidad.

### FORM (input type="color")

Otra actividad común en un sitio web puede ser la selección de un color. Disponemos un control de formulario especial para dicha funcionalidad:

```
<input type="color" id="color1" name="color1">
```

Ejemplo: Solicitar la selección de un color utilizando el control de tipo "color".

```
<!DOCTYPE HTML>
```

```
<html>
```

```
<head>
```

```
<title>Título de la página</title>
```

```
<meta charset="UTF-8">
```

```
</head>
```

```
<body>
```

```
<form action="#">
```

```
<p>Seleccione un color:
```

```
<input type="color" id="color1" name="color1">
```

```
</p>
```

```
<p><input type="submit" value="Confirmar"></p>
```

```
</form>
```

```
</body>
```

```
</html>
```

Los navegadores principales han implementado esta funcionalidad.

Si queremos que aparezca un color seleccionado por defecto debemos inicializar la propiedad "**value**" (con el color rojo):

```
<input type="color" id="color1" name="color1" value="#ff0000">
```

### FORM (input type="number")

Si necesitamos cargar un valor numérico podemos emplear un **input** con el siguiente formato:



```
<input type="number" id="valor" name="valor" min="1" max="10">
```

Las propiedades **min** y **max** definen el valor máximo y mínimo que se puede ingresar.

Ejemplo: Ingresar por teclado un valor numérico comprendido entre 1 y 10.

```
<!DOCTYPE HTML>
```

```
<html>
```

```
<head>
```

```
<title>Título de la página</title>
```

```
<meta charset="UTF-8">
```

```
</head>
```

```
<body>
```

```
<form action="#">
```

```
<p>Ingrese un valor entre 1 y 10:
```

```
<input type="number" id="valor" name="valor" min="1" max="10">
```

```
</p>
```

```
<p><input type="submit" value="Confirmar"></p>
```

```
</form>
```

```
</body>
```

```
</html>
```

Cuando ejecutamos esta página en Chrome aparece un selector numérico con dos botones que podemos incrementar o disminuir el valor:



### FORM (input type="url")

Si se necesita cargar una url por teclado en HTML5 podemos utilizar un control que valida que se ingrese correctamente la dirección del sitio web.

En el caso de no cargar una dirección web correcta aparece un diálogo informando tal situación.

Ejemplo: Ingresar por teclado las direcciones de tres sitios web en forma obligatoria.

```
<!DOCTYPE HTML>

<html>

<head>

  <title>Título de la página</title>

  <meta charset="UTF-8">

</head>

<body>

  <form action="#">

    <p>Dirección Web 1:

    <input type="url" id="direccion1" name="direccion1" size="30" required> </p>

    <p>Dirección Web 2:

    <input type="url" id="direccion2" name="direccion2" size="30" required> </p>

    <p>Dirección Web 3:

    <input type="url" id="direccion3" name="direccion3" size="30" required> </p>

    <p><input type="submit" value="Confirmar"> </p>

  </form>

</body>

</html>
```

Cuando el operador ingresa una dirección web incorrecta el navegador nos muestra con un mensaje tal situación:

Título de la página

file:///C:/wamp/www/html5ya

Dirección Web 1:

Dirección Web 2:

Dirección Web 3:

Confirmar

Ingresa una URL.

### FORM (input - datalist)

Cuando utilizamos un editor de línea:

```
<input type="text">
```

Podemos ingresar cualquier cadena en su interior. Con html5 podemos mostrarle una lista de sugerencias (no obligatorias como un select) para que el operador pueda seleccionarla sin tener que tipear todos los caracteres.

Esta funcionalidad se logra creando un elemento nuevo llamado **datalist** con todas sus opciones y luego asociándolo a un control input de tipo text.

Importante es notar que cuando disponemos un **datalist** no es obligatorio que el usuario seleccione uno de esos elementos, el usuario tiene la libertad de ingresar otra cadena de caracteres distinta a la propuesta por el **datalist**.

Ejemplo: Desarrollar un formulario que solicite la carga del nombre de navegador que utiliza el visitante. Disponer un control de tipo text para el ingreso de dicho dato y mostrar mediante un datalist los navegadores más comunes.

```
<!DOCTYPE HTML>
```

```
<html>
```

```
<head>
```

```
<title>Título de la página</title>
```

```
<meta charset="UTF-8">

</head>

<body>

  <form action="#">

    Ingrese el nombre de navegador que utiliza:

    <input type="text" id="navegador" list="listanavegadores">

      <datalist id="listanavegadores">

        <option label="Chrome" value="Chrome">

        <option label="Firefox" value="Firefox">

        <option label="Internet Explorer" value="Internet Explorer">

        <option label="Microsoft Edge" value="Microsoft Edge">

        <option label="Safari" value="Safari">

      </datalist>

    <br>

    <input type="submit" value="Confirmar">

  </form>

</body>

</html>
```

Por un lado definimos el datalist con todas sus opciones:

```
<datalist id="listanavegadores">

  <option label="Chrome" value="Chrome">

  <option label="Firefox" value="Firefox">

  <option label="Internet Explorer" value="Internet Explorer">

  <option label="Microsoft Edge" value="Microsoft Edge">

  <option label="Safari" value="Safari">

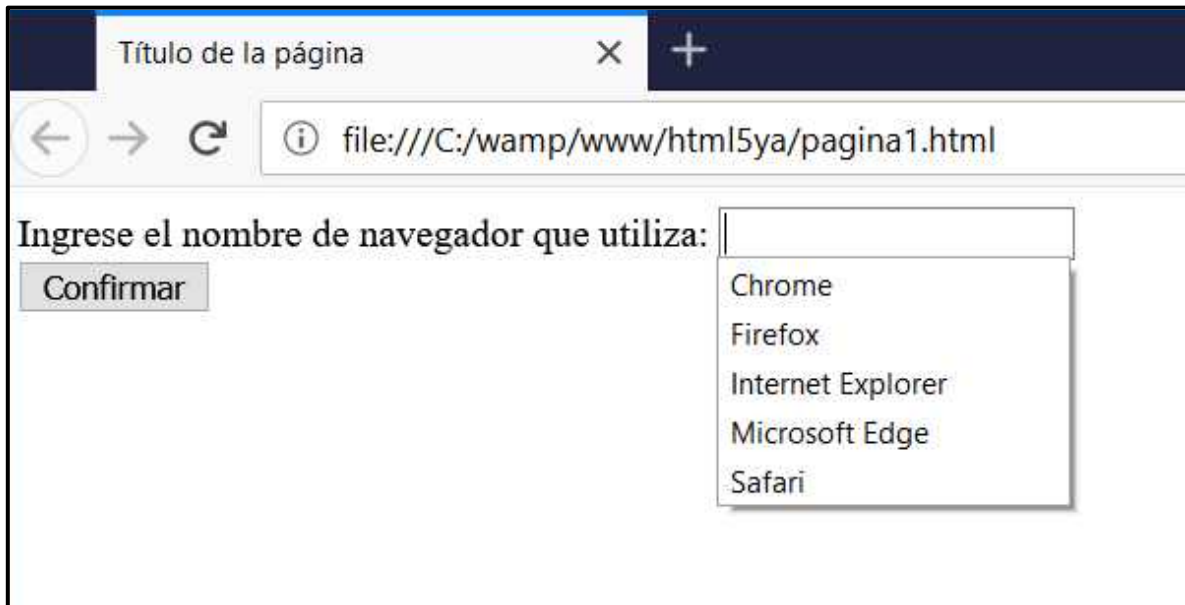
</datalist>
```

Dentro del elemento **datalist** definimos los elementos **option** inicializando la propiedad **label** con el texto que se mostrará y la propiedad **value** con el valor que retornará si se lo selecciona.

Para asociar el **datalist** con un control input debemos inicializar la propiedad **list** del control input con el id del **datalist**:

```
<input type="text" id="navegador" list="listanavegadores">
```

Luego cuando comenzamos a escribir el contenido aparece una lista desplegando las sugerencias:



### FORM (novalidate)

Html5 facilita la validación de datos sin tener en muchos casos la necesidad de utilizar Javascript mediante controles específicos o propiedades especiales como validate, pattern etc.

Hay que tener en cuenta también que normalmente implementamos formularios para que el usuario realice la carga de datos y sean recibidos posteriormente por un servidor web. Es sabido que el programa en el servidor (PHP, ASPNet etc.) debe validar nuevamente los datos enviados desde el navegador. En esta circunstancia es muy útil desactivar todas las validaciones que hemos implementado en el cliente y para no tener que modificar cada uno de los controles contenidos en un formulario existe una propiedad que desactiva todos los controles al mismo tiempo.

Para desactivar las validaciones en el cliente solo tenemos que agregar la propiedad **novalidate** a la marca **form**.

Seguramente no encontraremos ningún sitio que tenga esta propiedad activa en un sitio en producción ya que solo tiene como objetivo permitir la depuración de datos en el servidor cuando recibe datos no validados.

Ejemplo: Confeccionar una página que solicite la carga del nombre de usuario y una clave. No permitir enviar los datos al servidor si alguno de los controles está vacío, emplear para esto la propiedad `required`. Agregar la propiedad `novalidate` y comprobar que la información se envía al servidor independientemente que carguemos o no datos en los controles `input`.

```
<!DOCTYPE HTML>

<html>

<head>

  <title>Título de la página</title>

  <meta charset="UTF-8">

</head>

<body>

  <form action="#" novalidate>

    Usuario:

    <input type="text" id="usuario" required>

    <br>

    Clave:

    <input type="password" id="clave" required>

    <br>

    <br>

    <input type="submit">

  </form>

</body>

</html>
```

Lo único que hacemos es agregar la propiedad `novalidate` a la marca `form`:

```
<form action="#" novalidate>
```

Con esto todas las validaciones que hemos especificado dentro del formulario el navegador las saltea. Recordemos borrar la propiedad novalidate luego. Esto nos es útil para ver que hace nuestro programa en el servidor cuando por ejemplo dejamos vacía la clave.

### WEB STORAGE (**localStorage** y **sessionStorage**)

Con el HTML clásico si necesitamos almacenar datos en el cliente (navegador) se utilizan las cookies. Con HTML5 se han agregado otras tecnologías para almacenar datos en el cliente.

La primer tecnología que vamos a ver para almacenar datos en el navegador empleando HTML5 es la funcionalidad que provee el **objeto localStorage**.

El **objeto localStorage** nos permite almacenar datos que serán recordados por el navegador para siempre, es decir no tienen una fecha de caducidad.

La cantidad de información que podemos almacenar es muy superior a la permitida con las cookies, el **localStorage** permite almacenar por lo menos 5 Mb.

La información que se almacena en el **localStorage** a diferencia de las cookies no se envía al servidor cada vez que se solicita una página. Necesariamente debemos utilizar Javascript para almacenar y recuperar datos.

El objeto **localStorage** cuenta con dos métodos fundamentales para grabar y recuperar datos:

```
localStorage.setItem ( [clave] , [valor])
```

```
localStorage.getItem ( [clave] )
```

El método **setItem** permite almacenar los datos que le enviamos en el segundo parámetro y los guarda con la clave indicada en el primer parámetro.

Para recuperar datos del **localStorage** debemos llamar al método **getItem** pasando como parámetro la clave de referencia. La clave sería como el primary key en el modelo de base de datos relacionales.

Ejemplo: Confeccionar una aplicación que permita administrar un diccionario ingles/castellano, almacenar en forma local dichos datos.

```
<!DOCTYPE HTML>
```

```
<html>
```

```
<head>
```

```
<title>Título de la página</title>
```

```
<meta charset="UTF-8">
```

```
<script>
```

```
    window.addEventListener('load', inicio, false);
```

```
    function inicio() {
```

```
        document.getElementById('guardar').addEventListener('click', guardar, false);
```

```
        document.getElementById('traducir').addEventListener('click', recuperar, false);
```

```
    }
```

```
    function guardar(evt) {
```

```
        localStorage.setItem(document.getElementById('ingles').value,  
document.getElementById('castellano').value);
```

```
        document.getElementById('ingles').value="";
```

```
        document.getElementById('castellano').value="";
```

```
    }
```

```
    function recuperar(evt) {
```

```
        if (localStorage.getItem(document.getElementById('ingles').value) == null)
```

```
            alert('No está almacenada la palabra  
' + document.getElementById('ingles').value);
```

```
        else
```

```
            document.getElementById('castellano').value=localStorage.getItem(document.getElem  
entById('ingles').value);
```

```
    }
```



```
</script>
```

```
</head>
```

```
<body>
```

Palabra en ingles:

```
<input type="text" id="ingles">
```

```
<input type="button" id="traducir" value="Traducir">
```

```
<br>
```

Palabra en castellano:

```
<input type="text" id="castellano">
```

```
<br>
```

```
<input type="button" id="guardar" value="Guardar">
```

```
</body>
```

```
</html>
```

Hemos implementado dos controles input para la carga de datos por parte del operador y dos botones para procesar la grabación de datos en el localStorage y la búsqueda.

Lo primero que hacemos en Javascript es indicar que el método inicio se ejecute una vez que la página este cargada:

```
window.addEventListener('load', inicio, false);
```

En la función inicio asociamos la función que se ejecutará al presionarse cada botón:

```
function inicio() {  
    document.getElementById('guardar').addEventListener('click', guardar, false);  
    document.getElementById('traducir').addEventListener('click', recuperar, false);  
}
```

El método guardar se ejecuta cuando presionamos el botón "Guardar", en este llamamos al método setItem del objeto localStorage pasando como clave la palabra en

ingles y como dato a grabar la palabra en castellano ( por ejemplo si guardamos la traducción de casa luego como clave se almacena 'house' y como dato 'casa'):

```
function guardar(evt) {  
  
    localStorage.setItem(document.getElementById('ingles').value,  
document.getElementById('castellano').value);  
  
    document.getElementById('ingles').value="";  
  
    document.getElementById('castellano').value="";  
  
}
```

El proceso inverso, es decir la recuperación de datos del localStorage la realizamos en el la función 'recuperar', llamamos al método getItem pasando como parámetro la palabra en ingles que buscamos para que nos retorne su traducción. Agregamos un if para los casos que no exista la palabra dentro del diccionario:

```
function recuperar(evt) {  
  
    if (localStorage.getItem(document.getElementById('ingles').value) == null)  
  
        alert('No          está          almacenala          la          palabra  
' + document.getElementById('ingles').value);  
  
        else  
  
document.getElementById('castellano').value=localStorage.getItem(document.getElem  
entById('ingles').value);  
  
}
```

Como podemos analizar con un programa de 15 líneas podemos automatizar el almacenamiento y recuperación de datos del diccionario.

Otro objeto llamado **sessionStorage** cuenta con los mismos métodos pero la diferencia fundamental es que los datos almacenados solo permanecen mientras no cerremos la sesión del navegador, una vez que cerramos el navegador se pierden todos los datos almacenados utilizando el objeto **sessionStorage**.

Dependiendo de la necesidad utilizamos el **localStorage** o el **sessionStorage**.

## GEOLOCATION (getCurrentPosition)

La **geolocalización** es una característica de HTML5 que nos permite acceder a ubicación geográfica del usuario del sitio web con una precisión absoluta en el caso que el dispositivo que utilice tenga GPS.

La precisión va a disminuir si no tiene GPS y debe utilizarse la IP del proveedor de servicios de Internet. De todos modos veremos que podemos detectar dicha precisión.

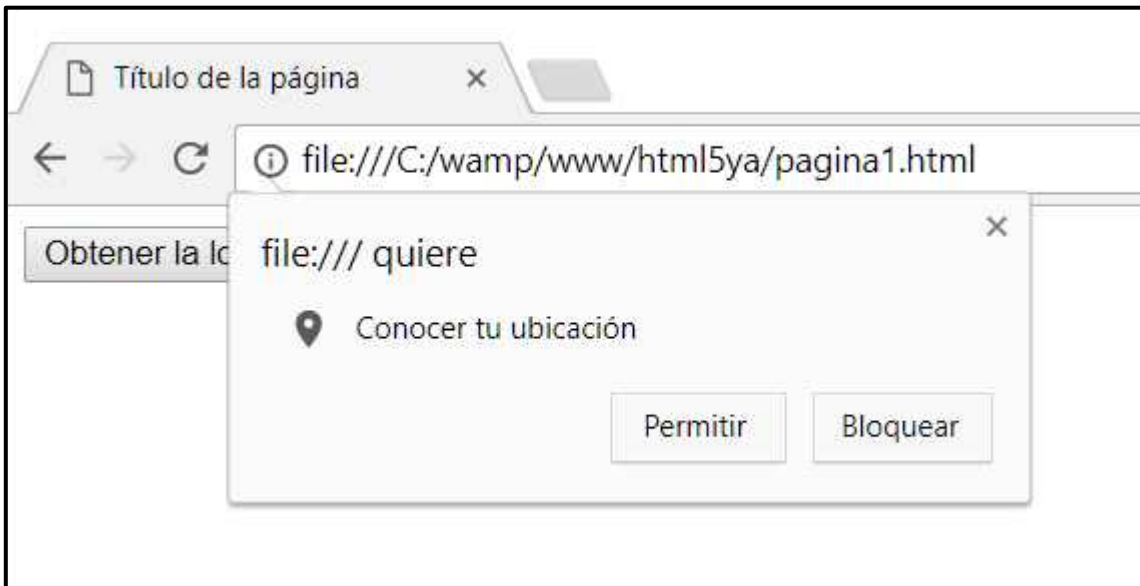
El navegador administra la **geolocalización** mediante un objeto llamado **geolocation** que es un atributo del objeto **navigator**.

El objeto **geolocation** cuenta con un método llamado **getCurrentPosition** que le enviamos el nombre de la función que se llamará cuando se obtenga la posición.

Como la **geolocalización** es una característica que puede invadir nuestra privacidad (tengamos en cuenta que estamos informando al sitio web el lugar exacto donde estamos parados en ese momento) el navegador nos muestra un diálogo para que aceptemos o no informar nuestra posición.

Los mensajes dependiendo del navegador son similares.

Por ejemplo en Chrome:



Ejemplo: Confeccionar una aplicación que permita obtener la latitud, longitud de nuestra posición y además con que precisión se obtuvo en metros.

```
<!DOCTYPE HTML>
```

```
<html>
```

```
<head>
```

```
<title>Título de la página</title>
```

```
<meta charset="UTF-8">
```

```
<script type="text/javascript">
```

```
    window.addEventListener('load', inicio, false);
```

```
    function inicio() {
```

```
        document.getElementById('obtener').addEventListener('click',  
recuperarLocalizacion, false);
```

```
    }
```

```
    function recuperarLocalizacion() {
```

```
        if (navigator.geolocation) {
```

```
            navigator.geolocation.getCurrentPosition(mostrarCoordenada);
```

```
        } else {
```

```
            alert('El navegador no dispone la capacidad de geolocalización');
```

```
        }
```

```
    }
```

```
    function mostrarCoordenada(posicion) {
```

```
        document.getElementById('dato').innerHTML='Latitud: '+
```

```
        posicion.coords.latitude+
```

```
        '<br> Longitud: '+posicion.coords.longitude+
```

```
        '<br>Exactitud: '+posicion.coords.accuracy;
```

```
    }
```

```
</script>
```

```
</head>
```

```
<body>
```

```
  <input type="button" id="obtener" value="Obtener la localización actual">
```

```
  <br>
```

```
  <span id="dato"> </span>
```

```
</body>
```

```
</html>
```

Analicemos el código. Disponemos un botón y un elemento HTML span para informar en tiempo de ejecución las coordenadas:

```
<input type="button" id="obtener" value="Obtener la localización actual">
```

```
<br>
```

```
<span id="dato"> </span>
```

En el evento load ejecutamos la función inicio asociando la función que se debe ejecutar al presionar el botón:

```
window.addEventListener('load', inicio, false);
```

```
function inicio() {
```

```
    document.getElementById('obtener').addEventListener('click',  
recuperarLocalizacion, false);
```

```
}
```

Ahora si analicemos lo nuevo que sucede cuando se presiona el botón:

```
function recuperarLocalizacion() {
```

```
    if (navigator.geolocation) {
```

```
        navigator.geolocation.getCurrentPosition(mostrarCoordenada);
```

```
    } else {
```

```
        alert('El navegador no dispone la capacidad de geolocalización');
```

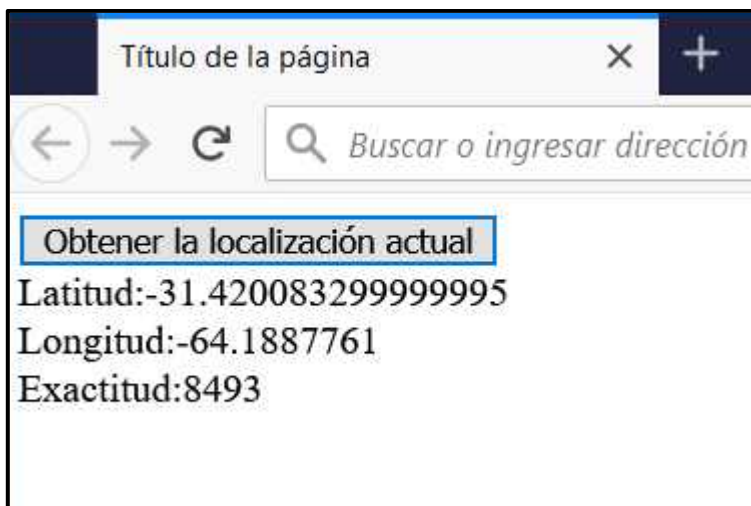
```
}  
  
}
```

Disponemos un if para verificar si nuestro navegador es moderno y dispone de la capacidad de geolocalización mediante el objeto geolocation. Si el if se verifica verdadero llamamos al método `getCurrentPosition` y le pasamos como dato el nombre de la función que se ejecutará cuando el navegador obtenga el dato de la posición.

La función `mostrarCoordenada` se ejecuta cuando el navegador obtuvo la posición actual y la recibe como parámetro esta función. El objeto que llega como parámetro dispone de una propiedad llamada `coords` que contiene entre otros datos la latitud, longitud, precisión en metros etc.:

```
function mostrarCoordenada(posicion) {  
    document.getElementById('dato').innerHTML='Latitud:' +  
        posicion.coords.latitude +  
        '<br> Longitud:' + posicion.coords.longitude +  
        '<br>Exactitud:' + posicion.coords.accuracy;  
}
```

Cuando ejecutamos la aplicación luego de aceptar informar de nuestra posición veremos los valores de la latitud, longitud y precisión:



## GEOLOCATION (mostrar en un mapa)

Es muy sencillo obtener la latitud y la longitud empleando HTML5, pero estos dos números brindan poca información si no los posicionamos en un mapa.

El mejor software actualmente sobre mapas es el Google Maps. Veremos ahora como el servicio de Google nos puede generar un mapa en forma dinámica a partir de la latitud y longitud que obtenemos en nuestro sitio web con el consentimiento del visitante.

Ejemplo: Confeccionar una aplicación que al presionar un botón obtenga la latitud y longitud de nuestra posición actual y luego solicitar a los servicios de Google Maps un mapa de dicha coordenada.

```
<!DOCTYPE HTML>

<html>

<head>

  <title>Título de la página</title>

  <meta charset="UTF-8">


<script type="text/javascript">


  window.addEventListener('load', inicio, false);


  function inicio() {

    document.getElementById('obtener').addEventListener('click',
recuperarLocalizacion, false);

  }


  function recuperarLocalizacion() {

    if (navigator.geolocation) {

      navigator.geolocation.getCurrentPosition(mostrarCoordenada);

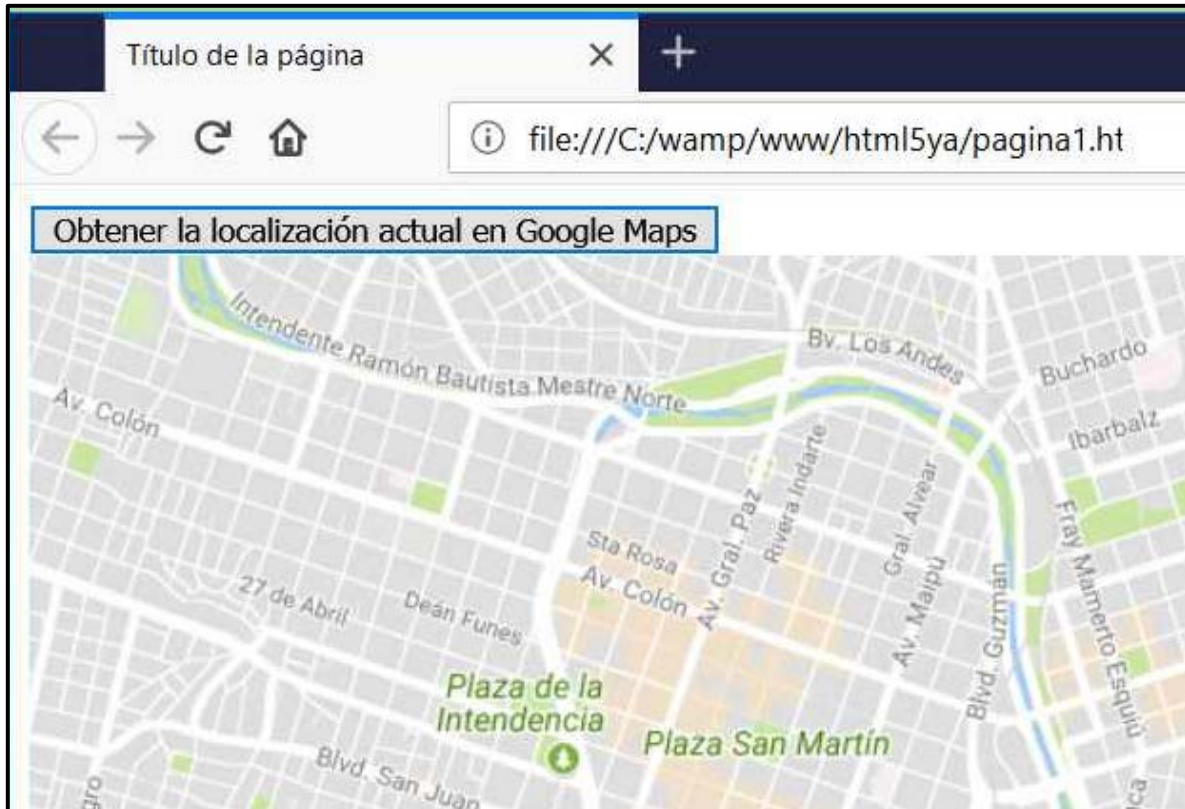
    } else {

      alert('El navegador no dispone la capacidad de geolocalización');
```

```
    }  
  }  
  
  function mostrarCoordenada(posicion) {  
    var direccion = posicion.coords.latitude + "," + posicion.coords.longitude;  
    var mapa = "http://maps.googleapis.com/maps/api/staticmap?center=" +  
      direccion + "&zoom=14&size=500x500&sensor=false";  
    document.getElementById("dato").innerHTML = "<img src='" + mapa + "'>";  
  }  
  
</script>  
  
</head>  
<body>  
  <input type="button" id="obtener" value="Obtener la localización actual en Google  
  Maps">  
  <br>  
  <div id="dato"></div>  
</body>  
</html>
```

Cuando ejecutamos el programa, presionamos el botón y luego de dar el consentimiento de obtener nuestra ubicación se muestra una imagen del mapa:





Ahora si los resultados en pantalla son muy claros.

Primero registramos el evento click para nuestro botón y cuando se presiona procedemos a verificar que se trate de un navegador moderno que acepte el HTML5, con lo que procedemos a llamar al método **getCurrentPosition** indicando el nombre de la función que recibirá la coordenada:

```
window.addEventListener('load', inicio, false);
```

```
function inicio() {  
    document.getElementById('obtener').addEventListener('click',  
recuperarLocalizacion, false);  
}
```

```
function recuperarLocalizacion() {  
    if (navigator.geolocation) {  
        navigator.geolocation.getCurrentPosition(mostrarCoordenada);
```

```
} else {  
    alert('El navegador no dispone la capacidad de geolocalización');  
}  
}
```

En la función `mostrarCoordenada` procedemos a llamar la página **staticmap** de los servidores de Google pasando como parámetro fundamental la coordenada actual (latitud y longitud), esta página genera una imagen que la mostramos en el div `dato`:

```
function mostrarCoordenada(posicion) {  
    var direccion = posicion.coords.latitude + "," + posicion.coords.longitude;  
    var mapa = "http://maps.googleapis.com/maps/api/staticmap?center=" +  
        direccion + "&zoom=14&size=500x500&sensor=false";  
    document.getElementById("dato").innerHTML = "<img src='" + mapa + "'>";  
}
```

### GEOLOCATION (tiempo de espera y captura de errores)

El método **getCurrentPosition** tiene otros dos parámetros opcionales:

```
getCurrentPosition([funcion que recibe la coordenada],  
    [función que captura el error],  
    [objeto que configura parámetros iniciales]);
```

Por ejemplo una llamada válida a `getCurrentPosition`:

```
navigator.geolocation.getCurrentPosition(mostrarCoordenada,errores,{ timeout: 50});
```

El segundo parámetro es el nombre de la función que eventualmente captura un error:

```
function errores(err) {  
    if (err.code == err.TIMEOUT)  
        alert("Se ha superado el tiempo de espera");  
    if (err.code == err.PERMISSION_DENIED)  
        alert("El usuario no permitió informar su posición");  
}
```

```
if (err.code == err.POSITION_UNAVAILABLE)

    alert("El dispositivo no pudo recuperar la posición actual");

}
```

El objeto **err** que llega como parámetro a la función nos informa el tipo de error generado por el intento de obtener la geolocalización. Hay tres posibilidades de errores que se pueden generar:

- **(TIMEOUT)**: El primer error posible es que el tiempo de espera para obtener la posición haya expirado. Podemos especificar al método `getCurrentPosition` el tiempo máximo que tiene para obtener la posición.
- **(PERMISSION\_DENIED)**: Se genera si el usuario no aprueba la obtención de la geolocalización.
- **(POSITION\_UNAVAILABLE)**: Cuando el dispositivo (por ejemplo un GPS) genera error interno.

Otro concepto importante es que al llamar al método **getCurrentPosition** en el tercer parámetro podemos enviarle un objeto donde configuramos todos o alguno de sus atributos:

`timeout` //Especificamos el tiempo máximo de espera (si no se especifica el tiempo es infinito)

`maximumAge` //El atributo `maximumAge` indica que la aplicación está dispuesto a aceptar una posición almacenada en el caché cuya edad no supere la cantidad de milisegundos indicado en esta propiedad.

Por defecto está configurada en cero.

`enableHighAccuracy` //Con el valor `true` activa la alta precisión (por defecto está configurado en `false`)

La sintaxis para configurar algunos de dichos atributos:

```
navigator.geolocation.getCurrentPosition(mostrarCoordenada,errores,{ timeout: 50,maximumAge: 60000});
```

Ejemplo: Confeccionar una aplicación que al presionar un botón obtenga la latitud y longitud de nuestra posición actual y luego solicitar a los servicios de Google Maps un mapa de dicha coordenada. Esperar solo 10 milisegundos y ver si se genera un error por el tiempo de espera. Disponer luego en el atributo `timeout` un valor mayor hasta que no se dispare el error de tiempo de espera.

```
<!DOCTYPE HTML>
```

```
<html>
```

```
<head>
```

```
<title>Título de la página</title>
```

```
<meta charset="UTF-8">
```

```
<script type="text/javascript">
```

```
    window.addEventListener('load', inicio, false);
```

```
    function inicio() {  
        document.getElementById('obtener').addEventListener('click',  
recuperarLocalizacion, false);  
    }
```

```
    function recuperarLocalizacion() {  
        if (navigator.geolocation) {  
  
navigator.geolocation.getCurrentPosition(mostrarCoordenada,errores,{ timeout: 50});  
        } else {  
            alert('El navegador no dispone la capacidad de geolocalización');  
        }  
    }
```

```
    function mostrarCoordenada(posicion) {  
        var direccion = posicion.coords.latitude + "," + posicion.coords.longitude;  
        var mapa = "http://maps.googleapis.com/maps/api/staticmap?center=" +  
            +direccion+"&zoom=14&size=500x500&sensor=false";  
        document.getElementById("dato").innerHTML = "<img src='"+mapa+"'>";  
    }
```

```
function errores(err) {  
    if (err.code === err.TIMEOUT)  
        alert("Se ha superado el tiempo de espera");  
    if (err.code === err.PERMISSION_DENIED)  
        alert("El usuario no permitió informar su posición");  
    if (err.code === err.POSITION_UNAVAILABLE)  
        alert("El dispositivo no pudo recuperar la posición actual");  
}
```

</script>

</head>

<body>

<input type="button" id="obtener" value="Obtener la localización actual en Google Maps">

<br>

<div id="dato"></div>

</body>

</html>

Hemos fijado el tiempo de espera en 50 milisegundos, con lo que es muy probable que no demos tiempo al dispositivo a generar la posición:

```
navigator.geolocation.getCurrentPosition(mostrarCoordenada,errores,{ timeout: 50});
```

La función errores la hemos pasado como parámetro cuando llamamos a `getCurrentPosition` y es en donde analizamos el tipo de error disparado:

```
function errores(err) {  
    if (err.code === err.TIMEOUT)  
        alert("Se ha superado el tiempo de espera");
```

```
if (err.code == err.PERMISSION_DENIED)

    alert("El usuario no permitió informar su posición");

if (err.code == err.POSITION_UNAVAILABLE)

    alert("El dispositivo no pudo recuperar la posición actual");

}
```

### GEOLOCATION (**watchPosition**)

Hemos empleado el método **getCurrentPosition** para obtener la posición, existe otro método llamado **watchPosition** que tiene los mismos parámetros que **getCurrentPosition** con la diferencia que la función que recibe la posición se llama cada vez que se detecta que nos hemos desplazado.

Ejemplo: Confeccionar una aplicación que inmediatamente arranque muestra la longitud, latitud y precisión. Actualizar dicha posición cada vez que cambie empleando para ello la llamada al método **watchPosition**.

Para probar este algoritmo es mejor hacerlo en un celular y desplazarnos para que se obtengan distintas posiciones.

```
<!DOCTYPE HTML>
```

```
<html>
```

```
<head>
```

```
<title>Título de la página</title>
```

```
<meta charset="UTF-8">
```

```
<script type="text/javascript">
```

```
window.addEventListener('load', inicio, false);
```

```
function inicio() {
```

```
    if (navigator.geolocation) {
```

```
        navigator.geolocation.watchPosition(mostrarCoordenada);
```

```
    } else {  
        alert('El navegador no dispone la capacidad de geolocalización');  
    }  
}
```

```
function mostrarCoordenada(posicion) {  
    document.getElementById('dato').innerHTML='Latitud: '+  
        posicion.coords.latitude+  
        '<br> Longitud: '+posicion.coords.longitude+  
        '<br>Exactitud: '+posicion.coords.accuracy;  
}
```

```
</script>
```

```
</head>
```

```
<body>
```

```
<span id="dato"></span>
```

```
</body>
```

```
</html>
```

En la función inicio que se dispara una vez que la página se ha cargado completamente es que llamamos a **watchPosition** indicando la función que se ejecutará cada vez que cambiemos de posición:

```
function inicio() {  
    if (navigator.geolocation) {  
        navigator.geolocation.watchPosition(mostrarCoordenada);  
    } else {  
        alert('El navegador no dispone la capacidad de geolocalización');  
    }  
}
```

```
}
```

El algoritmo para mostrar la coordenada actual no cambia:

```
function mostrarCoordenada(posicion) {  
    document.getElementById('dato').innerHTML='Latitud: '+  
        posicion.coords.latitude+  
        '<br> Longitud: '+posicion.coords.longitude+  
        '<br>Exactitud: '+posicion.coords.accuracy;  
}
```

### DRAG AND DROP (dragstart, dragover, drop)

El HTML5 nos permite fácilmente implementar el concepto de drag and drop (arrastrar y soltar).

Debemos seguir una serie específica de pasos para indicar que un elemento HTML se le permite que sea arrastrado y depositado en otra parte de la página.

Parte se resuelve con HTML y otra mediante Javascript.

Debemos indicar primero que elementos HTML se les permitirá ser arrastrados asignándole a la propiedad **draggable** el valor true:

```
<span id="palabra1" draggable="true">the </span>
```

Los eventos fundamentales que tenemos que capturar para procesar el arrastrar y soltar son:

**dragstart:** Se dispara cuando el usuario selecciona el elemento que quiere arrastrar. La función recibe como parámetro la referencia al elemento HTML que está siendo arrastrado.

**dragover:** Se dispara cuando el elemento se ha dispuesto dentro del contenedor. El parámetro de la función hace referencia al elemento contenedor. Como el comportamiento por defecto es denegar el drop, la función debe llamar al método **preventDefault** para indicar que se active el soltar elemento.

**drop:** El elemento arrastrado se ha soltado en el elemento contenedor. El parámetro de la función hace referencia al elemento contenedor.



Ejemplo: Confeccionar un programa que muestre una oración en Ingles con las palabras desacomodadas. Permitir mediante drag and drop disponer las palabras dentro de un div.

```
<!DOCTYPE HTML>

<html>

<head>

  <title>Título de la página</title>

  <meta charset="UTF-8">


<style>

#recuadro {

  width: 300px;

  height: 50px;

  background-color: yellow;

  border-style: solid;

  border-color: red;

  font-size: 1.5em;

}

#palabras span {

  font-size: 1.5em;

}

</style>


<script>

  window.addEventListener('load', inicio, false);


  function inicio() {

    document.getElementById('palabra1').addEventListener('dragstart', drag, false);
```

```
document.getElementById('palabra2').addEventListener('dragstart', drag, false);  
document.getElementById('palabra3').addEventListener('dragstart', drag, false);  
document.getElementById('palabra4').addEventListener('dragstart', drag, false);  
document.getElementById('recuadro').addEventListener('dragover', permitirDrop,  
false);  
  
document.getElementById('recuadro').addEventListener('drop', drop, false);  
  
}
```

```
function drag(ev)  
{  
    ev.dataTransfer.setData("Text",ev.target.id);  
}
```

```
function drop(ev)  
{  
    ev.preventDefault();  
    var dato=ev.dataTransfer.getData("Text");  
    ev.target.appendChild(document.getElementById(dato));  
}
```

```
function permitirDrop(ev)  
{  
    ev.preventDefault();  
}
```

</script>

</head>

```
<body>

  <p>Arrastre en orden las palabras para formar la oración correcta.</p>

  <div id="recuadro"></div>

  <div id="palabras">

    <span id="palabra1" draggable="true">the </span>

    <span id="palabra2" draggable="true">is </span>

    <span id="palabra3" draggable="true">What </span>

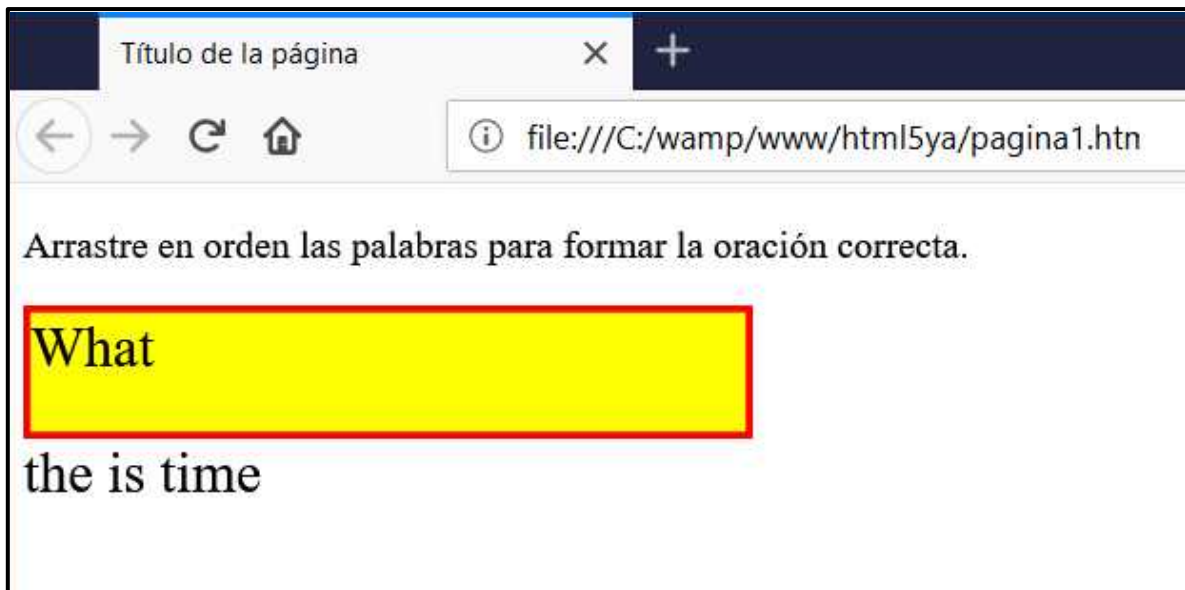
    <span id="palabra4" draggable="true">time </span>

  </div>

</body>

</html>
```

El resultado en el navegador al cargar esta página es:



En el cuerpo de la página definimos un div llamado recuadro donde dispondremos las palabras que se arrastrarán a su interior:

```
<div id="recuadro"></div>
```

Se definió en la hoja de estilo del div el color amarillo de fondo con rojo como borde:

```
#recuadro {
```

```
width: 300px;

height: 50px;

background-color: yellow;

border-style: solid;

border-color: red;

font-size: 1.5em;

}
```

Por otro lado definimos cuatro elementos **span** que contienen las palabras que pueden ser arrastradas, para permitir el arrastre debemos inicializar la propiedad **draggable** con el valor true:

```
<span id="palabra1" draggable="true">the </span>

<span id="palabra2" draggable="true">is </span>

<span id="palabra3" draggable="true">What </span>

<span id="palabra4" draggable="true">time </span>
```

Cuando se termina de cargar la página se ejecuta el método inicio donde asociamos para cada elemento **span** el evento **dragstart** indicando que se ejecute la función drag.

Por otro lado debemos indicar los nombres de las funciones que se ejecutarán para los eventos **dragover** y **drop** del elemento contenedor (es decir el elemento HTML que recibirá en su interior los elementos arrastrados):

```
window.addEventListener('load', inicio, false);
```

```
function inicio() {

    document.getElementById('palabra1').addEventListener('dragstart', drag, false);

    document.getElementById('palabra2').addEventListener('dragstart', drag, false);

    document.getElementById('palabra3').addEventListener('dragstart', drag, false);

    document.getElementById('palabra4').addEventListener('dragstart', drag, false);

    document.getElementById('recuadro').addEventListener('dragover', permitirDrop, false);

    document.getElementById('recuadro').addEventListener('drop', drop, false);

}
```

```
}
```

La función **drag** se ejecuta cuando el operador selecciona el objeto a arrastrar y tiene como objetivo mediante el atributo **dataTransfer** indicar en este caso el id del objeto que acaba de seleccionarse, para luego poder recuperarlo en el evento **drop**:

```
function drag(ev)
{
    ev.dataTransfer.setData("Text",ev.target.id);
}
```

Por otro lado debemos indicar al objeto que recibe el elemento permitir recibir objetos:

```
function permitirDrop(ev)
{
    ev.preventDefault();
}
```

Finalmente la función **drop** se ejecuta cuando soltamos el elemento dentro del contenedor. En esta llamamos a **preventDefault** del contenedor y seguidamente recuperamos la referencia del objeto que estábamos arrastrando mediante el método **getData** del atributo **dataTransfer**. Creamos un elemento dentro del div:

```
function drop(ev)
{
    ev.preventDefault();
    var dato=ev.dataTransfer.getData("Text");
    ev.target.appendChild(document.getElementById(dato));
}
```

## DRAG AND DROP (drag, dragend)

Como mínimo cuando implementamos el **drag and drop** capturamos los eventos de:

**dragstart:** Se dispara cuando el usuario selecciona el elemento que se quiere arrastrar. La función recibe como parámetro la referencia al elemento HTML que está siendo arrastrado.

**dragover:** Se dispara cuando el elemento se ha dispuesto dentro del contenedor. El parámetro de la función hace referencia al elemento contenedor. Como el comportamiento por defecto es denegar el drop, la función debe llamar al método **preventDefault** para indicar que se active el soltar elemento.

**drop:** El elemento arrastrado se ha soltado en el elemento contenedor. El parámetro de la función hace referencia al elemento contenedor.

Otros dos eventos factibles de capturar son:

**drag :** Se dispara cada vez que el elemento se mueve. El parámetro de la función hace referencia al elemento HTML que se está arrastrando. Debemos configurar este evento para cada elemento factible de mover.

**dragend:** Se dispara cuando el elemento se suelta, indistintamente que se suelte dentro o fuera del contenedor.

Ejemplo: Confeccionar un programa que muestre una oración en Ingles con las palabras desacomodadas. Permitir mediante drag and drop disponer las palabras dentro de un div. Inmediatamente luego que el usuario selecciona una palabra cambiar el fondo del div que recibirá la palabra con colores aleatorios. Cuando se suelte la palabra volver el fondo del div al color amarillo.

```
<!DOCTYPE HTML>

<html>

<head>

  <title>Título de la página</title>

  <meta charset="UTF-8">


<style>

#recuadro {

  width: 300px;

  height: 50px;

  background-color: yellow;
```

```
border: 2px solid #ff0000;

font-size: 1.5em;
}

#palabras span {

font-size: 1.5em;
}

#palabras {

padding-top: 50px;
}

</style>

<script>

window.addEventListener('load', inicio, false);

function inicio() {

    document.getElementById('palabra1').addEventListener('dragstart', dragInicio,
false);

    document.getElementById('palabra2').addEventListener('dragstart', dragInicio,
false);

    document.getElementById('palabra3').addEventListener('dragstart', dragInicio,
false);

    document.getElementById('palabra4').addEventListener('dragstart', dragInicio,
false);

    document.getElementById('palabra1').addEventListener('drag', dragMueve,
false);

    document.getElementById('palabra2').addEventListener('drag', dragMueve,
false);

    document.getElementById('palabra3').addEventListener('drag', dragMueve,
false);
```

```
document.getElementById('palabra4').addEventListener('drag', dragMueve,
false);

document.getElementById('palabra1').addEventListener('dragend', dragSolto,
false);

document.getElementById('palabra2').addEventListener('dragend', dragSolto,
false);

document.getElementById('palabra3').addEventListener('dragend', dragSolto,
false);

document.getElementById('palabra4').addEventListener('dragend', dragSolto,
false);

document.getElementById('recuadro').addEventListener('dragover', permitirDrop,
false);

document.getElementById('recuadro').addEventListener('drop', drop, false);
}

function dragInicio(ev)
{
    ev.dataTransfer.setData("Text",ev.target.id);
}

function dragMueve(ev)
{
    var ale=200+parseInt(Math.random()*55);

    var rojo=ale;

    var verde=ale;

    var azul=ale;

    document.getElementById('recuadro').style.background =
"rgb("+rojo+", "+verde+", "+azul+")";
}
```



```
function dragSolto(ev)
{
    document.getElementById('recuadro').style.background = "rgb(255,255,0)";
}

function drop(ev)
{
    var dato=ev.dataTransfer.getData("Text");
    ev.target.appendChild(document.getElementById(dato));
    ev.preventDefault();
    document.getElementById(dato).removeEventListener('dragstart', dragInicio,
false);
}

function permitirDrop(ev)
{
    ev.preventDefault();
}

</script>

</head>

<body>

<p>Arrastre en orden las palabras para formar la oración correcta.</p>

<div id="recuadro"></div>

<div id="palabras">

    <span id="palabra1" draggable="true">the </span>
```

```
<span id="palabra2" draggable="true">is </span>
<span id="palabra3" draggable="true">What </span>
<span id="palabra4" draggable="true">time </span>
</div>
</body>
</html>
```

En la función inicio registramos los eventos dragstart, drag y dragend para todos los elementos HTML capacitados para ser arrastrados:

```
document.getElementById('palabra1').addEventListener('dragstart', dragInicio,
false);

document.getElementById('palabra2').addEventListener('dragstart', dragInicio,
false);

document.getElementById('palabra3').addEventListener('dragstart', dragInicio,
false);

document.getElementById('palabra4').addEventListener('dragstart', dragInicio,
false);

document.getElementById('palabra1').addEventListener('drag', dragMueve,
false);

document.getElementById('palabra2').addEventListener('drag', dragMueve,
false);

document.getElementById('palabra3').addEventListener('drag', dragMueve,
false);

document.getElementById('palabra4').addEventListener('drag', dragMueve,
false);

document.getElementById('palabra1').addEventListener('dragend', dragSolto,
false);

document.getElementById('palabra2').addEventListener('dragend', dragSolto,
false);

document.getElementById('palabra3').addEventListener('dragend', dragSolto,
false);
```

```
document.getElementById('palabra4').addEventListener('dragend', dragSolto, false);
```

También registramos los eventos a capturar del div receptor de objetos:

```
document.getElementById('recuadro').addEventListener('dragover', permitirDrop, false);
```

```
document.getElementById('recuadro').addEventListener('drop', drop, false);
```

En la función **dragInicio** almacenamos el id del objeto que comienza a arrastrarse:

```
function dragInicio(ev)
{
    ev.dataTransfer.setData("Text",ev.target.id);
}
```

La función **dragMueve** se ejecuta cada vez que desplazamos el objeto por la pantalla y como actividad cambiamos el color de fondo del div contenedor con un valor aleatorio (tener en cuenta que esta función se comenzará a ejecutar mientras tengamos agarrado el objeto y no lo hallamos soltado):

```
function dragMueve(ev)
{
    var ale=200+parseInt(Math.random()*55);
    var rojo=ale;
    var verde=ale;
    var azul=ale;

    document.getElementById('recuadro').style.background =
    "rgb("+rojo+","+verde+","+azul+")";
}
```

La función **dragSolto** se ejecuta cuando se suelta el objeto indistintamente estemos o no dentro del contenedor, en nuestro problema fijamos nuevamente el color de fondo del div a color amarillo:

```
function dragSolto(ev)
{
    document.getElementById('recuadro').style.background = "rgb(255,255,0)";
}
```

En la función **drop** agregamos la palabra dentro del div y suprimimos el evento que pueda moverse nuevamente dicha palabra llamando a `removeEventListener`:

```
function drop(ev)
{
    var dato=ev.dataTransfer.getData("Text");
    ev.target.appendChild(document.getElementById(dato));
    ev.preventDefault();

    document.getElementById(dato).removeEventListener('dragstart', dragInicio,
false);
}
```

La última función es permitir disponer objetos dentro del div contenedor:

```
function permitirDrop(ev)
{
    ev.preventDefault();
}
```

### DRAG AND DROP (**dragenter**, **dragleave**)

Falta analizar dos eventos más que podemos capturar cuando implementamos algoritmos de **drag and drop**.

**dragenter**: se ejecuta este evento cuando un objeto que es arrastrado entra en el objeto contenedor. El parámetro de esta función hace referencia al objeto contenedor.

**dragleave:** se ejecuta este evento cuando un objeto que es arrastrado sale del objeto contenedor. El parámetro de esta función hace referencia al objeto contenedor.

Ejemplo: Confeccionar un programa que muestre una oración en Inglés con las palabras desacomodadas. Permitir mediante drag and drop disponer las palabras dentro de un div. Cambiar el color de fondo del recuadro donde se disponen las palabras una vez que el usuario entra a dicho recuadro y volverlo al color original si sale del recuadro sin soltar el objeto.

```
<!DOCTYPE HTML>

<html>

<head>

  <title>Título de la página</title>

  <meta charset="UTF-8">


<style>

  #recuadro {

    width: 300px;

    height: 50px;

    background-color: rgb(255,255,0);

    border-style: solid;

    border-color: red;

    font-size: 1.5em;

  }

  #palabras span {

    font-size: 1.5em;

  }

</style>


<script>

  window.addEventListener('load', inicio, false);
```

```
function inicio() {  
    document.getElementById('palabra1').addEventListener('dragstart', dragInicio,  
false);  
    document.getElementById('palabra2').addEventListener('dragstart', dragInicio,  
false);  
    document.getElementById('palabra3').addEventListener('dragstart', dragInicio,  
false);  
    document.getElementById('palabra4').addEventListener('dragstart', dragInicio,  
false);  
    document.getElementById('recuadro').addEventListener('dragover', permitirDrop,  
false);  
    document.getElementById('recuadro').addEventListener('drop', drop, false);  
    document.getElementById('recuadro').addEventListener('dragenter', entra,  
false);  
    document.getElementById('recuadro').addEventListener('dragleave', sale, false);  
}
```

```
function dragInicio(ev)  
{  
    ev.dataTransfer.setData("Text",ev.target.id);  
}
```

```
function drop(ev)  
{  
    ev.preventDefault();  
    var dato=ev.dataTransfer.getData("Text");  
    ev.target.appendChild(document.getElementById(dato));  
    document.getElementById(dato).removeEventListener('dragstart', dragInicio,  
false);
```

```
        document.getElementById('recuadro').style.background = "rgb(255,255,0)";
    }

    function permitirDrop(ev)
    {
        ev.preventDefault();
    }

    function entra(ev)
    {
        document.getElementById('recuadro').style.background = "rgb(255,0,0)";
    }

    function sale(ev)
    {
        document.getElementById('recuadro').style.background = "rgb(255,255,0)";
    }
</script>

</head>
<body>
    <p>Arrastre en orden las palabras para formar la oración correcta.</p>
    <div id="recuadro"></div>
    <div id="palabras">
        <span id="palabra1" draggable="true">the </span>
        <span id="palabra2" draggable="true">is </span>
        <span id="palabra3" draggable="true">What </span>
```

```
<span id="palabra4" draggable="true">time </span>
</div>
</body>
</html>
```

Como podemos ver lo nuevo que presenta este problema es que registramos los eventos **dragenter** y **dragleave** para el objeto contenedor (es decir el div donde el usuario arrastrará las palabras):

```
document.getElementById('recuadro').addEventListener('dragenter',      entra,
false);

document.getElementById('recuadro').addEventListener('dragleave', sale, false);
```

Cuando el usuario arrastra una palabra y entra al div contenedor se dispara la función 'entra' donde cambiamos el color de fondo del div:

```
function entra(ev)
{
    document.getElementById('recuadro').style.background = "rgb(255,0,0)";
}
```

Y cuando el usuario arrastrando el objeto sale del div se dispara la función 'sale' donde regresamos al color amarillo el fondo del div:

```
function sale(ev)
{
    document.getElementById('recuadro').style.background = "rgb(255,255,0)";
}
```

Lo único nuevo que agregamos a la función 'drop' es volver al color amarillo de fondo del div:

```
function drop(ev)
```



```
{  
    ev.preventDefault();  
    var dato=ev.dataTransfer.getData("Text");  
    ev.target.appendChild(document.getElementById(dato));  
    document.getElementById(dato).removeEventListener('dragstart',    dragInicio,  
false);  
    document.getElementById('recuadro').style.background = "rgb(255,255,0)";  
}
```

## API FILE (lectura de archivo de texto local)

Otra funcionalidad que tenemos con el HTML5 es el acceso solo de lectura de los archivos que hay en nuestro equipo.

Veremos cual es la estructura de Javascript que debemos implementar para poder seleccionar un archivo de texto e inmediatamente poder acceder a su contenido en el navegador.

Para trabajar con archivos que se encuentran en el disco duro del equipo el usuario es el responsable de seleccionar uno. Hay varias formas para la selección del archivo, veremos primero la más común que es disponer un control HTML:

```
<input type="file" name="archivo">
```

Este control dispone en la página web un botón que al ser presionado por el usuario aparece un diálogo para seleccionar un archivo de nuestro disco duro.

Para poder identificar cuando el usuario seleccionó un archivo debemos implementar el evento **change** de dicho control.

Ejemplo: Confeccionar un programa que nos permita seleccionar de nuestro disco duro un archivo de texto y posteriormente leer su contenido y mostrarlo en un control textarea. Mostrar además su nombre, tamaño y tipo.

```
<!DOCTYPE HTML>
```

```
<html>
```

```
<head>
```

```
<title>Título de la página</title>
```

```
<meta charset="UTF-8">
```

```
<script>

    window.addEventListener('load', inicio, false);

    function inicio() {

        document.getElementById('archivo').addEventListener('change', cargar, false);

    }

    function cargar(ev) {

document.getElementById('datos').innerHTML='Nombre                               del
archivo: '+ev.target.files[0].name+'<br>'+

                                'Tamaño del archivo: '+ev.target.files[0].size+'<br>'+

                                'Tipo MIME: '+ev.target.files[0].type;

        var arch=new FileReader();

        arch.addEventListener('load',leer,false);

        arch.readAsText(ev.target.files[0]);

    }

    function leer(ev) {

        document.getElementById('editor').value=ev.target.result;

    }

</script>

</head>

<body>

<input type="file" id="archivo"><br>

<textarea rows="10" cols="80" id="editor"></textarea>
```

```
<br>
```

```
<p id="datos"></p>
```

```
</body>
```

```
</html>
```

Veamos detenidamente cual debe ser la estructura de nuestro algoritmo para poder leer el contenido de un archivo de texto que se encuentra en el disco duro del equipo.

Primero definimos un control de tipo **file** que es el encargado de permitirnos seleccionar un archivo:

```
<input type="file" id="archivo"><br>
```

Disponemos un control de tipo **textarea** donde mostraremos el contenido del archivo que seleccione el operador:

```
<textarea rows="10" cols="80" id="editor"></textarea>
```

Finalmente disponemos un párrafo donde mostraremos el nombre de archivo que se leerá, su tamaño y tipo:

```
<p id="datos"></p>
```

Ahora veamos donde se encuentra el programa en Javascript que nos permite acceder al archivo, en la función inicio registramos el evento **change** del control de tipo file que se disparará cuando el usuario seleccione un archivo del disco duro:

```
window.addEventListener('load', inicio, false);
```

```
function inicio() {
```

```
    document.getElementById('archivo').addEventListener('change', cargar, false);
```

```
}
```

Cuando el usuario selecciona un archivo se ejecuta la función cargar (que registramos en la función inicio), la función recibe un objeto de la clase File que lo accedemos: `ev.target.files[0]`, este objeto tiene tres atributos `name` (nombre del archivo que acabamos de seleccionar), `size` (tamaño en bytes del archivo) y `type` (tipo de archivo).

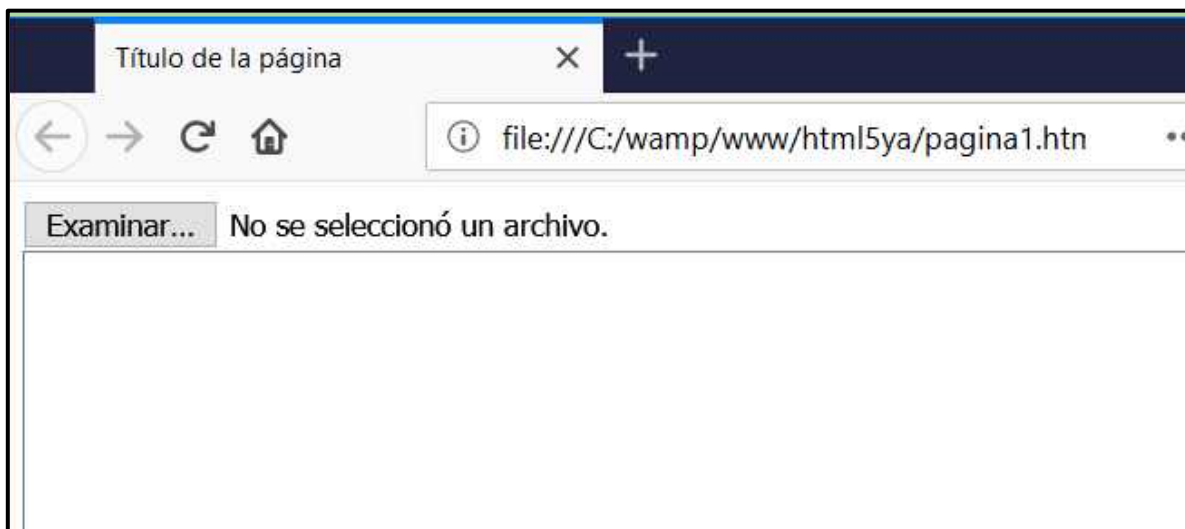
Luego de mostrar las tres propiedades fundamentales del archivo procedemos a crear un objeto de la clase `FileReader` para poder acceder al contenido del archivo. Mediante la llamada al método `readAsText` procedemos a leer el contenido del archivo y registramos el evento `load` para indicar la función que se disparará cuando tengamos todo el archivo en memoria:

```
function cargar(ev) {  
    document.getElementById('datos').innerHTML='Nombre del  
    archivo:'+ev.target.files[0].name+'<br>'+  
    'Tamaño del  
    archivo:'+ev.target.files[0].size+'<br>'+  
    'Tipo MIME:'+ev.target.files[0].type;  
    var arch=new FileReader();  
    arch.addEventListener('load',leer,false);  
    arch.readAsText(ev.target.files[0]);  
}
```

Por último la función leer recibe un objeto que almacena todos los datos contenidos en del archivo:

```
function leer(ev) {  
    document.getElementById('editor').value=ev.target.result;  
}
```

Inicialmente se muestra el botón para la selección del archivo y el textarea vacío:



Luego de seleccionar un archivo de texto del disco duro podemos ver el contenido del archivo y sus propiedades (nombre, tamaño y tipo):



## API FILE (lectura de múltiples archivos de texto locales)

El concepto anterior de HTML5 nos permite seleccionar un archivo de texto del disco duro local y seguidamente acceder a sus propiedades como el nombre, tamaño, tipo así como a su contenido.

Otra variante de este algoritmo es acceder a un conjunto de archivos en forma simultánea. Esto lo logramos cuando definimos un **control file** e indicamos la propiedad **multiple**:

```
<input type="file" id="archivo" multiple>
```

Esta propiedad no es necesario asignarle valor. Con esto cuando el operador presione el botón le permitirá seleccionar múltiples archivos.

Ejemplo: Confeccionar un programa que nos permita seleccionar un conjunto de archivos de texto de una sola vez y muestre sus contenidos en un control textarea (separando el contenido de cada archivo por varios guiones). Mostrar además todos los nombres de archivos leídos.

```
<!DOCTYPE HTML>

<html>

<head>

  <title>Título de la página</title>
```

```
<meta charset="UTF-8">

<script>

    window.addEventListener('load', inicio, false);

    function inicio() {

        document.getElementById('archivo').addEventListener('change', cargar, false);

    }

    function cargar(ev) {

        document.getElementById('datos').innerHTML="";

        document.getElementById('editor').value="";

        for(var f=0;f<ev.target.files.length;f++) {

            document.getElementById('datos').innerHTML=document.getElementById('datos').innerHTML+

                'Nombre'

                del

            archivo: '+ev.target.files[f].name+'<br>';

            var arch=new FileReader();

            arch.addEventListener('load',leer,false);

            arch.readAsText(ev.target.files[f]);

        }

    }

    function leer(ev) {

        document.getElementById('editor').value=document.getElementById('editor').value+ev.target.result+
```

```
        '\n-----\n';
    }
</script>
</head>

<body>
<p>Selecciona un conjunto de archivos de texto del disco duro.</p>
<input type="file" id="archivo" multiple><br>
<textarea rows="10" cols="80" id="editor"></textarea>
<br>
<p id="datos"></p>
</body>
</html>
```

En el bloque de HTML solo hemos dispuesto la propiedad **multiple** al control de tipo **file** (esto informa al navegador que está permitido la selección de más de un archivo):

```
<body>
<p>Selecciona un conjunto de archivos de texto del disco duro.</p>
<input type="file" id="archivo" multiple><br>
<textarea rows="10" cols="80" id="editor"></textarea>
<br>
<p id="datos"></p>
</body>
</html>
```

La función cargar se ejecuta una vez que el usuario seleccionó los archivos del disco duro, lo primero que hacemos es borrar el contenido del párrafo y el textarea por si hay datos de selecciones anteriores:

```
function cargar(ev) {  
    document.getElementById('datos').innerHTML="";  
    document.getElementById('editor').value="";
```

Ahora viene lo distinto, como se pueden haber seleccionado más de un archivo debemos disponer una estructura repetitiva for. Para saber la cantidad de archivos seleccionados la propiedad files es un vector por lo que podemos acceder a length que almacena la cantidad de elementos del mismo:

```
for(var f=0;f<ev.target.files.length;f++) {
```

Dentro del for vamos concatenando en el párrafo el nombre de archivo que accedemos del vector según su subíndice ev.target.files[f].name:

```
document.getElementById('datos').innerHTML=document.getElementById('datos').innerHTML+  
    'Nombre' del  
    archivo: '+ev.target.files[f].name+'<br>';
```

Por otro lado creamos por cada archivo un objeto de la clase FileReader y le asociamos la función que se ejecutará cuando se termine de cargar el contenido del archivo en memoria:

```
var arch=new FileReader();  
arch.addEventListener('load',leer,false);  
arch.readAsText(ev.target.files[f]);  
}  
}
```

La función leer se ejecuta por cada uno de los archivos que se lee, aquí procedemos a acumular el contenido de cada archivo en el textarea y agregarles una serie de guiones de separación:



```
function leer(ev) {  
  
document.getElementById('editor').value=document.getElementById('editor').value+e  
v.target.result+  
  
                '\n-----  
                \n';  
  
}
```

Inicialmente se muestra el botón para la selección de archivos y el textarea vacío:



Luego de seleccionar archivos de texto del disco duro podemos ver el contenido de los archivos y sus nombres:



## API FILE (lectura de una imagen)

Las API para manejo de archivos no se limitan solo al formato de texto, sino que podemos leer cualquier tipo de archivo y luego mediante Javascript procesarlo en forma local. Un formato muy fácil de leerlo y mostrarlo son las imágenes.

Para la lectura de un archivo con formato de imagen la clase **FileReader** dispone de un método llamado **readAsDataURL** que transforma los datos de la imagen en una URL de datos.

Ejemplo: Permitir seleccionar una imagen del disco duro y posteriormente mostrarla en el fondo de un div.

```
<!DOCTYPE HTML>

<html>

<head>

  <title>Título de la página</title>

  <meta charset="UTF-8">

<style>
```

```
#caja {  
    margin: 10px;  
    width: 350px;  
    height: 350px;  
    border: 5px dashed gray;  
    border-radius: 8px;  
    background: rgb(230,230,230);  
    background-repeat: no-repeat;  
    background-size: 100%;  
}  
</style>
```

```
<script>  
    window.addEventListener('load', inicio, false);  
  
    function inicio() {  
        document.getElementById('archivo').addEventListener('change', cargar, false);  
    }  
  
    function cargar(ev) {  
        var arch=new FileReader();  
        arch.addEventListener('load',leer,false);  
        arch.readAsDataURL(ev.target.files[0]);  
    }  
  
    function leer(ev) {
```

```
        document.getElementById('caja').style.backgroundImage="url(" +
ev.target.result + "");

    }

</script>

</head>

<body>

<p>Lectura de una imagen almacenada en el disco duro.</p>

<input type="file" id="archivo"><br>

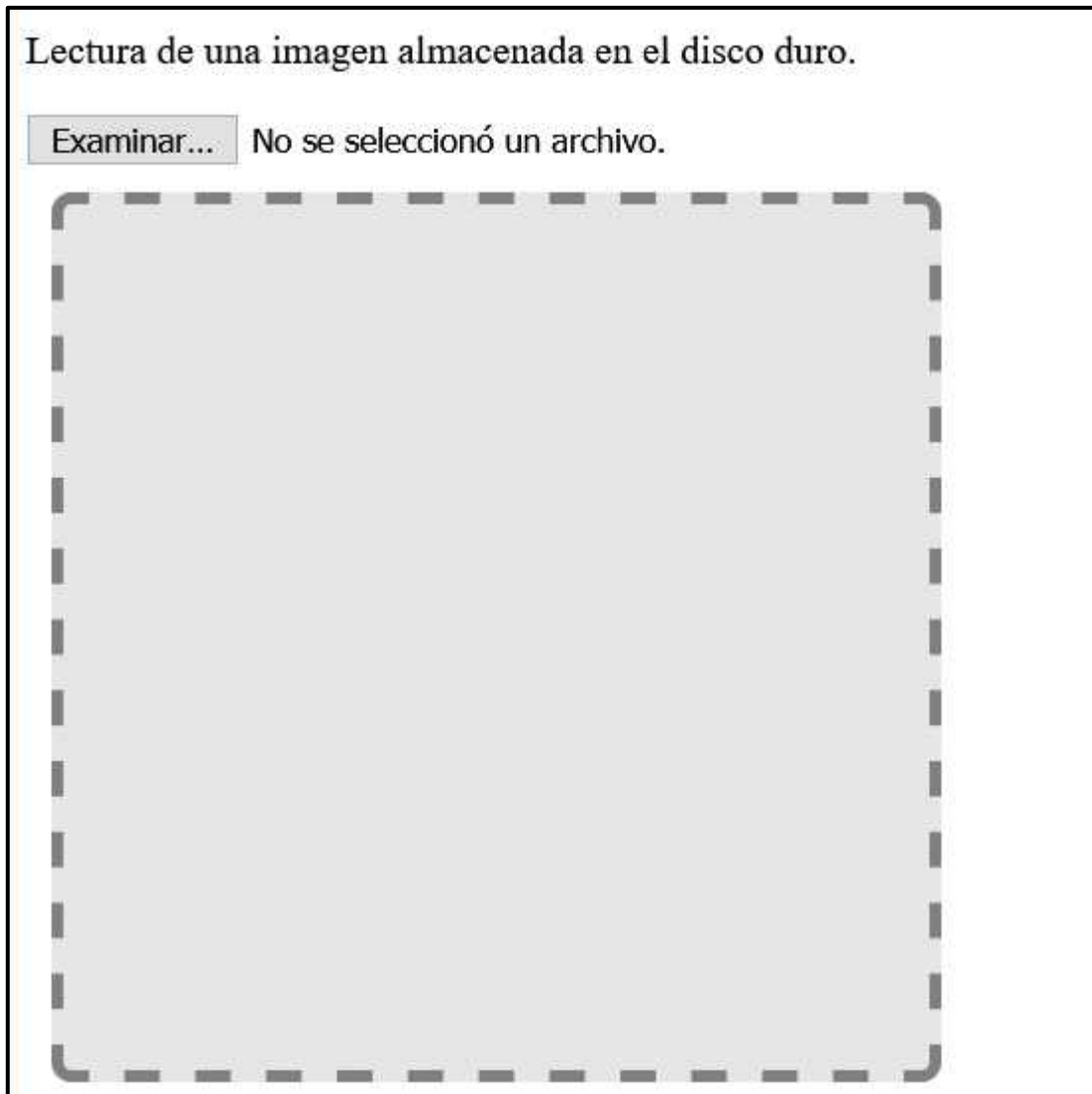
<div id="caja">

</div>

</body>

</html>
```

El resultado en el navegador al cargar esta página es:



Para resolver este problema hemos dispuesto un div llamado caja que configuraremos la propiedad **backgroundImage** cuando el usuario seleccione una imagen del disco duro:

```
<body>  
  
<p>Lectura de una imagen almacenada en el disco duro.</p>  
  
<input type="file" id="archivo"><br>  
  
<div id="caja">  
  
</div>  
  
</body>
```

En la función inicio registramos el evento **change** para el control HTML de tipo file:

```
function inicio() {  
    document.getElementById('archivo').addEventListener('change', cargar, false);  
}
```

Cuando el usuario terminó de seleccionar la foto se ejecuta la función cargar donde creamos un objeto de la clase **FileReader**, registramos el evento load para indicar que se ejecute la función leer una vez que el archivo esté en memoria, y finalmente llamamos al método **readAsDataURL** que procede a cargar la imagen:

```
function cargar(ev) {  
    var arch=new FileReader();  
    arch.addEventListener('load',leer,false);  
    arch.readAsDataURL(ev.target.files[0]);  
}
```

En la función leer inicializamos la propiedad **backgroundImage** con la imagen que acabamos de leer (como está con formato de URL de datos debemos llamar a la función url previo a la asignación):

```
function leer(ev) {  
    alert(ev.target.result);  
    document.getElementById('caja').style.backgroundImage="url(" +  
ev.target.result + ")";  
}
```

## API FILE (drag and drop de una imagen del escritorio)

Con la API FILE podemos leer archivos que se encuentran en forma local solo disponiendo un control de tipo file y mediante el botón que muestra el navegador seleccionar el archivo que queremos leer.

Pero veremos ahora que las API de DRAG AND DROP que provee HTML5 nos permiten arrastrar archivos que se encuentran en el escritorio de la computadora (es decir fuera del navegador) y proceder a su lectura.

Ejemplo: Permitir arrastrar una imagen que se encuentre en el escritorio de la computadora y posteriormente mostrarla en el fondo de un div.

```
<!DOCTYPE HTML>

<html>

<head>

  <title>Título de la página</title>

  <meta charset="UTF-8">


<style>

#caja {

  margin: 10px;

  width: 350px;

  height: 350px;

  border: 5px dashed gray;

  border-radius: 8px;

  background: rgb(230,230,230);

  background-repeat: no-repeat;

  background-size: 100%;

}

</style>


<script>

  window.addEventListener('load', inicio, false);


  function inicio() {

    document.getElementById('caja').addEventListener('dragover',    permitirDrop,
false);
```

```
document.getElementById('caja').addEventListener('drop', drop, false);
}

function drop(ev)
{
    ev.preventDefault();
    var arch=new FileReader();
    arch.addEventListener('load',leer,false);
    arch.readAsDataURL(ev.dataTransfer.files[0]);
}

function permitirDrop(ev)
{
    ev.preventDefault();
}

function leer(ev) {
    document.getElementById('caja').style.backgroundImage="url('"
ev.target.result + "')";
}
</script>
</head>

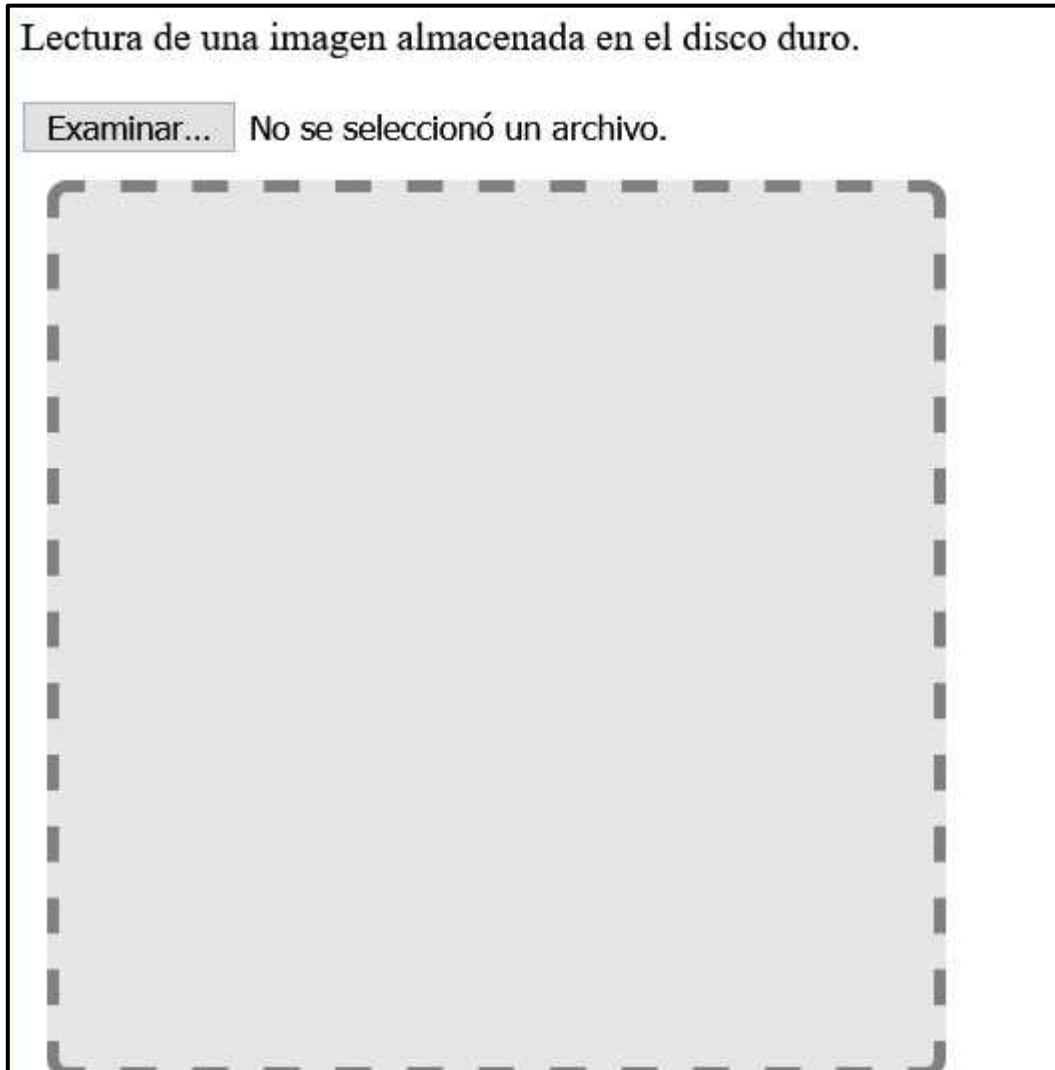
<body>
<p>Arrastrar una imagen desde el escritorio.</p>
<div id="caja">
</div>
```



```
</body>
```

```
</html>
```

El resultado en el navegador al cargar esta página es:



Hemos dispuesto que el archivo solo se pueda arrastrar desde el escritorio.

Nuestro bloque HTML ahora queda muy cortito:

```
<body>
```

```
<p>Arrastrar una imagen desde el escritorio.</p>
```

```
<div id="caja">
```

```
</div>
```

```
</body>
```

Registramos los eventos dragover y drop de nuestro div:

```
function inicio() {  
    document.getElementById('caja').addEventListener('dragover', permitirDrop,  
false);  
    document.getElementById('caja').addEventListener('drop', drop, false);  
}
```

Cuando el usuario suelta la imagen dentro del div procedemos a leer el archivo y registrar el evento load para que se muestre posteriormente:

```
function drop(ev)  
{  
    ev.preventDefault();  
    var arch=new FileReader();  
    arch.addEventListener('load',leer,false);  
    arch.readAsDataURL(ev.dataTransfer.files[0]);  
}
```

Mostramos la imagen en la función leer:

```
function leer(ev) {  
    document.getElementById('caja').style.backgroundImage="url('"  
ev.target.result + "')";  
}
```

Desactivamos en la función permitirDrop la actividad por defecto que hace un navegador cuando se arrastra una imagen (que por defecto muestra la imagen ocupando toda la ventana del navegador):

```
function permitirDrop(ev)  
{  
    ev.preventDefault(); }
```

Bibliografía

Sitios web visitados:

<http://w3c.github.io/html/>

<https://www.fing.edu.uy/tecnoinf/mvd/cursos/ria/material/teorico/ria-03-HTML5-CSS3.pdf>

<http://desarrolloweb.dlsi.ua.es/cursos/2011/html5-css3-es/introduccion>

<https://gutl.jovenclub.cu/wp-content/uploads/2013/10/El+gran+libro+de+HTML5+CSS3+y+Javascrrip.pdf>