Programación Front End (nivel inicial)

Ing. Leonardo Torres, Lic. Gustavo Lafuente

FACUTLAD DE INGENIERÍA
Universidad Nacional de la Pampa

Argentina programa 4.0

Lenguaje **HTML 5.0**

HTML 5

Introduce e incrementa un amplio rango de nuevas características

Controles de formularios, APIs, multimedia, estructura, y semántica.

- Semantic Elements
- New Input Types
- HTML Graphics
- HTML Video / Audio
- HTML Geolocation
- HTML Drag / Drop
- HTML Local Storage
- HTML Web Workers

Lenguaje **HTML 5.0**

HTML 5

Introduce e incrementa un amplio rango de nuevas características

Controles de formularios, APIs, multimedia, estructura, y semántica.

Semantic Elements ✓
New Input Types ✓
HTML Graphics
HTML Video / Audio ✓
HTML Geolocation
Cookies

HTML Local Storage
HTML Web Workers

HTML 5.0
Geolocalización

La API de geolocalización HTML5 se utiliza para obtener la posición geográfica de un usuario.

Dado que esto puede comprometer la privacidad del usuario, la posición no está disponible a menos que el usuario lo apruebe.

el método **getCurrentPosition()** para obtener la posición del usuario.

HTML 5.0
Geolocalización

```
<script>
 var x=document.getElementById("demo");
 function getLocation()
  {
    if (navigator.geolocation)
     {
       navigator.geolocation.getCurrentPosition(showPosition);
     }
    else {x.innerHTML="Geolocation is not supported by this browser.";}
  }
 function showPosition(position)
  {
  x.innerHTML="Latitude: " + position.coords.latitude +
  "<br>Longitude: " + position.coords.longitude;
  }
</script>
```

geolocalizacion.htm

Geolocalización

Gestión de errores

El segundo parámetro del método **getCurrentPosition()** es usado para el manejo de errores

Especifica una función (**showError**) para ejecutarse en caso de no obtener la ubicación del usuario:

Códigos de error:

- ***PERMISSION_DENIED*** *Permiso denegado - el usuario no permitió Geolocalización*
- ***POSITION_UNAVAILABLE*** *Posición no disponible - No es posible obtener la ubicación actual*
- ***TIMEOUT*** *Tiempo de espera - La operación ha agotado*

Geolocalización

Gestión de errores

```
function showError(error)
  {
  switch(error.code)
    {
      case error.PERMISSION_DENIED:
          x.innerHTML="User denied the request for Geolocation."
          break;
      case error.POSITION_UNAVAILABLE:
          x.innerHTML="Location information is unavailable."
          break;
      case error.TIMEOUT:
          x.innerHTML="The request to get user location timed out."
          break;
      case error.UNKNOWN_ERROR:
          x.innerHTML="An unknown error occurred."
           break;
    }
  }
```

geolocalizacionError.htm

Geolocalización

Resultados en un mapa

```
function showPosition(position)
 {
 var latlon=position.coords.latitude+","+position.coords.longitude;

 var img_url="http://maps.googleapis.com/maps/api/staticmap?center="
 +latlon+"&zoom=14&size=400x300&sensor=false";

 document.getElementById("mapholder").innerHTML="<img src='"+img_url+"'>";
 }
```

geolocalizacionMapa.html
(usando Google Maps con una imagen estática)
geolocalizacionMapa2Dinamico.html
(usando Google Maps con controles y mapa interactivo)

Html **Graphics**

HTML Canvas

<canvas> se utiliza para dibujar gráficos, a través de secuencias de comandos (normalmente JavaScript).

HTML SVG

SVG significa *Scalable Vector Graphics*

Canvas

El elemento **<canvas>** es sólo un *contenedor para gráficos*.

Se debe utilizar una secuencia de comandos (JS) para dibujar los gráficos.

Canvas tiene varios métodos para dibujar *trazados, cajas, círculos, texto y agregar imágenes*.

Un Canvas es un área rectangular en una página HTML, y se especifica con el elemento **<canvas>**.

Nota: De forma predeterminada, el elemento **<canvas>** no tiene bordes ni contenido.

```
<canvas id="miCanvas" width="200" height="100"
    style="border:2px solid #abc123;">
</canvas>
```

miCanvas.html

Canvas
Dibujar con JS

Todo dibujo en el Canvas se debe hacer dentro de un JavaScript

```
<canvas id="miCanvas" width="200" height="100"
    style="border:1px solid #c3c3c3;">
Su Browser no soporta el tag HTML5 canvas.
</canvas>
```

```
<script>
    var c = document.getElementById("miCanvas");
    var ctx = c.getContext("2d");
    ctx.fillStyle = "#FAA200";
    ctx.fillRect(0,0,150,75);
</script>
```

miCanvasRectangulo.htm
l

Canvas
Dibujar con JS

El elemento Canvas

```
var c = document.getElementById("miCanvas");
```

- **getContext ()** (hay que pasar la cadena "2d" al método getContext ()):

```
var ctx = c.getContext("2d");
```

- El ("2d") objeto **getContext** es un objeto incorporado en HTML5, con muchas propiedades y métodos para dibujar Líneas, cajas, círculos, texto, imágenes y más.
- Las siguientes dos líneas dibujan un rectángulo rojo:

```
ctx.fillStyle = "#FAA200";
ctx.fillRect(0,0,150,75);
```

miCanvasRectangulo.html

Canvas: Paths (Líneas)

Para dibujar líneas rectas en un Canvas, utilizamos 2 métodos:

moveTo (x, y) define el punto inicial de la línea
lineTo (x, y) define el punto final de la línea

Para dibujar la línea en realidad, tenemos que usar uno de los métodos de "pintura", como **stroke()**

```
var c = document.getElementById("miCanvas");
var ctx = c.getContext("2d");
ctx.moveTo(0,0);
ctx.lineTo(200,100);
ctx.stroke();
```

miCanvasLinea.html

Canvas: Circulo

Para dibujar Círculos usamos el método:

arc(x,y,r,start,stop)

Para círculos la línea en realidad, tenemos que usar uno de los métodos de “pintura", como **stroke()** o **fill()**.

```
var c = document.getElementById("miCanvas");
var ctx = c.getContext("2d");
ctx.beginPath();
ctx.arc(95,50,40,0,2*Math.PI);
ctx.stroke();  //o usar el método fill()
```

miCanvasLinea.html

Canvas: Textos

Para dibujar texto en un cavas los métodos son:

font define las propiedades de font para el texto

fillText(text,x,y) – Dibuja el texto “relleno" sobre el canvas

strokeText(text,x,y) – Dibuja el texto sobre el canvas (no relleno)

```
var c = document.getElementById("miCanvas");
ctx.font = "30px Arial";
ctx.fillText("Hola POW",10,50);
ctx.strokeText("Hola POW",10,80);
```

miCanvasTexto.html

Canvas: Degradados (Gradients)

Los degradados se pueden utilizar para rellenar rectángulos, círculos, líneas, texto, etc. Las formas en el canvas no se limitan a los colores sólidos.

Hay dos tipos diferentes de gradientes:

createLinearGradient(x, y, x1, y1) - Crea un gradiente lineal

createRadialGradient(x, y, r, x1, y1, r1) - Crea un radial / degradado circular

Una vez que tenemos un objeto degradado, hay que añadir dos o más niveles de color.

El método **addColorStop()** especifica los niveles de color, y su posición a lo largo del gradiente. Posiciones gradiente puede estar en cualquier lugar entre 0 a 1.

Para usar el degradado, establezca la propiedad **fillStyle** o **strokeStyle** al degradado, y luego dibujar la forma, como un rectángulo, texto, o una línea.

miCanvasGradientes.htm

Canvas: Degradados (Gradients)

```
var c = document.getElementById("miCanvas");
var ctx = c.getContext("2d");

// Crea Gradiente
var grd = ctx.createLinearGradient(0,0,200,0);
grd.addColorStop(0,"red");
grd.addColorStop(1,"white");

// Rellenar con gradiente
ctx.fillStyle = grd;
ctx.fillRect(10,10,150,80);
```

```
var c = document.getElementById("miCanvas2");
var ctx = c.getContext("2d");

// Crea Gradiente
var grd = ctx.
ctx.createRadialGradient(75,50,5,90,60,100);
grd.addColorStop(0,"red");
grd.addColorStop(1,"white");

// Rellenar con gradiente
ctx.fillStyle = grd;
ctx.fillRect(10,10,150,80);
```

miCanvasGradientes.htm

Canvas: Images

Para dibujar imágenes:

drawImage(image,x,y)

```
 var c = document.getElementById("miCanvas");
var ctx = c.getContext("2d");
var img = document.getElementById(“imagenID");
ctx.drawImage(img,10,10);
```

miCanvasImagen.html
CanvasRotarImagen.html

HTML5: **SVG**

- SVG significa **Scalable Vector Graphics**
- Se utiliza para definir los gráficos basados en vectores para la web
- Los gráficos son definidos en formato XML
- Gráficos SVG no pierden ninguna calidad si se amplían o cambian de tamaño
- Cada elemento y cada atributo en archivos SVG se pueden animar
- SVG es una recomendación de W3C
- Ventajas sobre otros formatos de imagen (como JPEG y GIF):
  - pueden ser creados y editados con cualquier editor de texto
  - se pueden buscar, indexar, usar con script, y se comprimen
  - son escalables
  - se pueden imprimir con alta calidad en cualquier resolución
  - son ampliables (y la imagen se pueden ampliar sin que se deforme)

HTML5: **SVG**


```
<!DOCTYPE html>
<html>
<body>

    <svg width="300" height="200">
        <polygon points="100,10 40,198 190,78 10,78 160,198"
        style="fill:purple;stroke:lime;stroke-width:5;fill-
    rule:evenodd;" />
        Lo siento, su navegador no soporta el formato SVG.
    </svg>

</body>
</html>
```

miImagenSVG.html
/HTML_SVG/index.html
/HTML_SVG/elipses.htm

Lenguaje **HTML 5.0**

Hasta aquí que vimos y como continuamos…..

- Semantic Elements ✓
- New Input Types ✓
- HTML Graphics ✓
- HTML Video / Audio ✓
- HTML Geolocation ✓
- HTML Cookies
- HTML Local Storage
- HTML Web Workers
- HTML Application cache

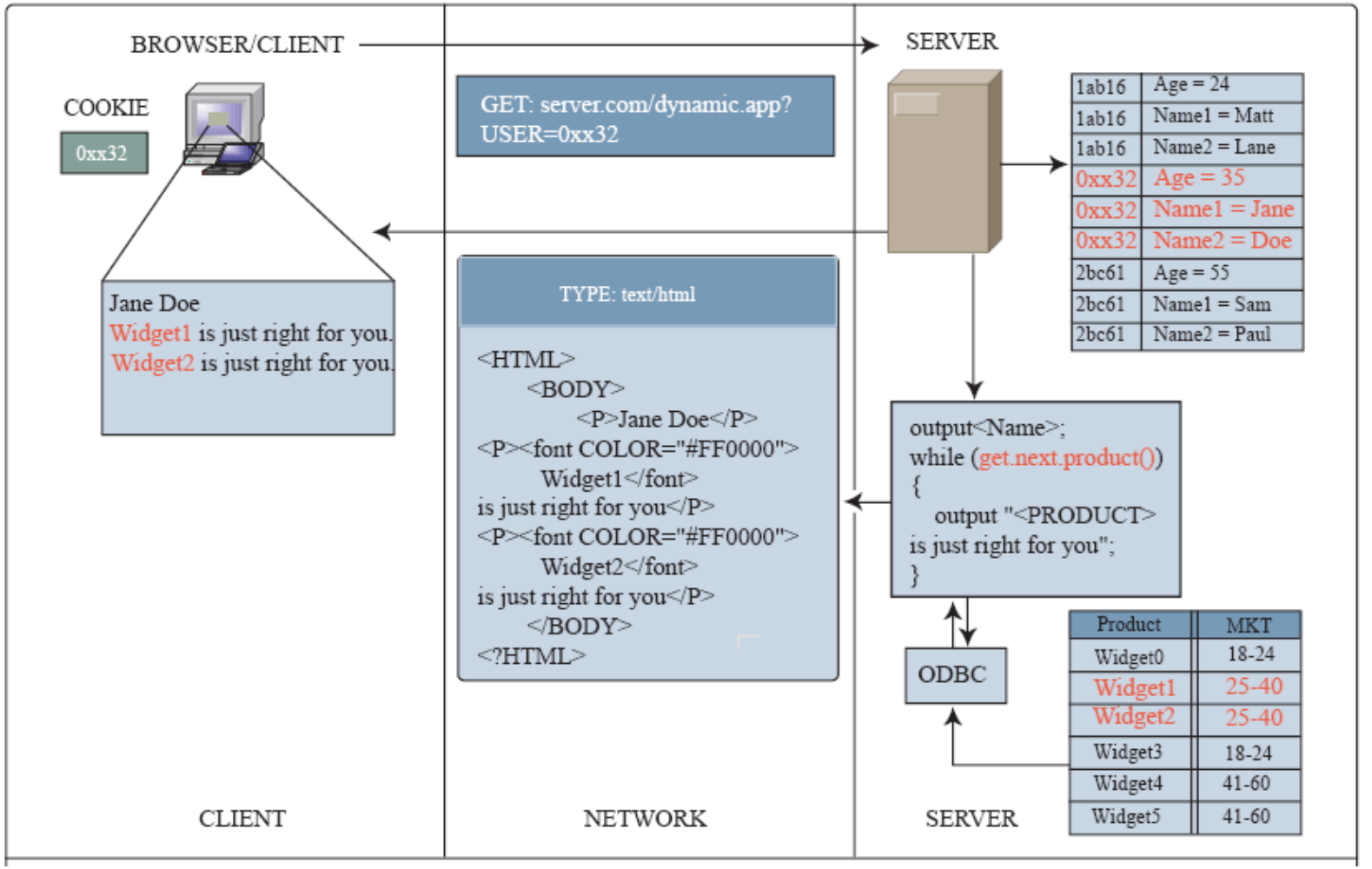
Almacenamiento de Datos del lado Cliente

- HTML5 ofrece dos nuevos objetos para el almacenamiento de datos en el cliente:
  - **localStorage** - almacena los datos sin límite de tiempo (window.localStorage)
  - **sessionStorage** - almacena los datos para un período de sesiones (code.sessionStorage)
- Anteriormente, esto se hizo con las **cookies**.
  - Las cookies no son adecuadas para grandes cantidades de datos, ya que se transmiten por cada petición al servidor, por lo que es muy lento y poco efectivo.
- En HTML5, los datos no se transmiten por cada petición del servidor, pero es usado solamente cuando se le consulta por ellos.
  - Es posible almacenar grandes cantidades de datos sin afectar el rendimiento de la página web.
- Los datos se almacenan en diferentes áreas para diferentes sitios web y un sitio web puede sólo acceder a los datos almacenados por él mismo.
  - ˜5MB (puede configurarse).
- HTML5 utiliza JavaScript para almacenar y acceder a los datos.

Cookies

- Son un método para identificar a los usuarios web y proporcionarles páginas web personalizadas.
  - La primera vez que el usuario accede al portal web se recopila información personalizada del mismo.
  - El servidor "empaqueta" la información en una "cookie" y la envía en forma de archivo al navegador del cliente, que la almacena en el sistema de archivos.
  - Cada vez que el usuario se conecta al mismo portal web el navegador envía la "cookie" (almacenada en su PC) al servidor como identificación personalizada.
  - El servidor utiliza la información de la "cookie" para personalizar las páginas enviadas al usuario.

Cookies: Ejemplo



Cookies

- El uso de cookies permite entre otras cosas:
  - Manejar información relacionada con los elementos seleccionados.
  - Los sitios pueden almacenar preferencias del usuario.
  - Las aplicaciones de registro de usuarios pueden liberar al cliente de reingresar su identificación en la siguiente conexión.

Cookies

- Una cookie es introducida en el cliente a través de la inclusión de un encabezado HTTP como parte de la respuesta del servidor
- Típicamente esto es realizado por código dinámico o por un módulo propio del servidor web
- **Set-Cookie: NAME=VALUE; expires=DATE;path=PATH;domain=DOMAIN_NAME; secure**

Cookies

- **NAME=VALUE**
  - Especifica el nombre de la cookie y su valor

- **Expires=DATE**
  - Especifica una fecha que define el tiempo de vida de la cookie. Al llegar a la fecha, la cookie debe ser eliminada por el cliente
  - El formato es Wdy, DD-Mon-YYYY HH:MM:SS
  - GMT

Cookies

- **domain=DOMAIN_NAME**
  - Especifica el dominio al que se permite enviar la cookie
- **path=PATH**
  - Especifica la ruta dentro del servidor en donde se permite enviar la cookie
- **secure**
  - Especifica que la cookie solo se puede enviar por un canal seguro

Cookies

- En JavaScript la cadena que se utiliza para la consulta y manipulación de *cookies* es:

- **document.cookie**.

- Dicha cadena representa todas las *cookies* asociadas al documento. Al ser esta una variable de tipo cadena se recomiendan utilizar métodos tales como **substring**, **charAt**, **indexOf** y **lastIndexOf** para determinar los valores guardados en la *cookie*.

Cookies

- Una forma básica para escribir una cookie:

```
document.cookie='ejemplo=cookie'
```

- Forma básica para leer una cookie:

```
alert(document.cookie);
document.write(document.cookie);
```

Cookies

Como ejemplos de uso de *cookies:*

- Número de veces que algún usuario ha visitado una página.
- Llevar una historia personalizada de los elementos que se llevan en un carrito electrónico.
- Guardar preferencias del usuario.

Ejemplo del uso de cookies.

Código de la página que guarda (escribe) la cookie

```
<head>
    <title>Escribir una Cookie</title>
<script language="javascript" type="text/javascript">
// <!CDATA[

function Button1_onclick() {
    var nombre, mascota, cook1, cook2;
    nombre = Text1.value;
    cook1 = "usuario=" + nombre + ";expires=Mon,6-Jul-09 12:00:00 GMT;"
    mascota = Text2.value;
    cook2 = "mascota=" + mascota + ";expires=Mon,6-Jul-09 12:00:00 GMT;"
    document.cookie = cook1;
    document.cookie = cook2;
}
```

Código de la página que lee la **cookie**

```
<head>
    <title>Leer Cookie</title>
<script language="javascript" type="text/javascript">
// <!CDATA[

function Button1_onclick() {
  Text1.value = document.cookie;
}

// ]]>
</script>
```

HTML 5: El Objeto **localStorage**

El objeto **localStorage** almacena los datos sin límite de tiempo.

Antes de utilizar el almacenamiento local, compruebe la compatibilidad con los navegadores para **localStorage** y **sessionStorage**

```
if(typeof(Storage) !== "undefined") {
    // Código para localStorage/sessionStorage.
} else {
    // Lo siento su navegador no soporta Storage.
}
```

```
<script type="text/javascript">
            localStorage.apellido=“Perez";
            document.write(localStorage.apellido);
</script>
```

```
// Para almacenar
localStorage.apellido = “Perez";
// Recuperar
document.getElementById("result").innerHTML = localStorage.apellido;
//Para remover
localStorage.removeItem(“apellido");
```

El Objeto localStorage

El siguiente ejemplo cuenta el número de veces que un usuario ha visitado una página:

```
<script type="text/javascript">
  if (localStorage.contarclick)
   {
     localStorage. contarclick=Number(localStorage. contarclick) +1;
    }
  else
   {
     localStorage.contarclick=1;
   }
  document.write("cliqueastes el botón "+ localStorage. contarclick + " vece(s).");
</script>
```

localstore.html

El objeto sessionStorage

El objeto **sessionStorage** almacena los datos para una sesión.

Los datos se eliminan cuando el usuario cierra la ventana del navegador.

```
<script type="text/javascript">
        sessionStorage.apellido=“Perez";
document.write(sessionStorage.apellido);
</script>
```

El objeto sessionStorage

El siguiente ejemplo se cuenta el número de veces que un usuario ha visitado una página, en la sesión actual

```
<script type="text/javascript">
if (sessionStorage.contarpagina)
  {
  sessionStorage.contarpagina=Number(sessionStorage.contarpagina) +1;
  }
else
  {
  sessionStorage.contarpagina=1;
  }
document.write("Se ha visitado "+sessionStorage.contarpagina+" veces(s) durante esta sesión.");
</script>
```

HTML5 Application Cache

- **Application Cache**
  - una aplicación web se almacena en caché, y es accesible sin tener conexión a Internet.

Ventajas:

- **Navegación offline** - los usuarios pueden utilizar la aplicación cuando están fuera de línea.
- **Velocidad** – los recursos en caché se cargan más rápido.
- **Reducción de la carga del servidor** - el navegador sólo descargará actualizados los recursos cambiados desde el servidor

HTML5 Application Cache

Ejemplo que muestra un documento HTML con un manifiesto de caché (para navegar sin conexión):

```
<!DOCTYPE HTML>
<html manifest="demo.appcache">

<body>
    El contenido del documento.....
</body>

</html>
```

HTML5 Application Cache

- Cada página con el atributo **manifest** especificado se almacena en caché cuando el usuario lo visita.
- Si no se especifica el atributo **manifest**, la página no se almacenan en caché (a menos que se especifique la página directamente en el archivo de manifiesto).
- La extensión de archivo recomendada para los archivos de manifiesto son: "**.appcache**"
- Nota: Un archivo de manifiesto debe ser servido con el tipo de medio correcto, que es "**text /cache-manifest**". Debe estar configurado en el servidor web.

HTML5 Application Cache

- El archivo de **Manifest**
  - El archivo de manifiesto es un archivo de texto simple, que le dice al navegador qué almacena en caché (y lo que nunca es cacheable).

El archivo de manifiesto cuenta con tres secciones:

- **CACHE MANIFEST**
  - *Archivos enumerados bajo esta cabecera que se almacenan en caché después de que se descarguen por primera vez*
- **NETWORK**
  - *Archivos aparecen bajo esta cabecera deben requerir una conexión con el servidor, y nunca se almacena en caché.*
- **FALLBACK**
  - *Archivos enumerados bajo esta cabecera especifica páginas de retorno si una página es inaccesible*

HTML5 Application Cache

CACHE MANIFEST

```
CACHE MANIFEST
/estilos.css
/casa.jpg
/main.js
```

El archivo **manifest** tres recursos: un archivo CSS, una imagen JPG, y un archivo JavaScript. Cuando se carga el archivo de manifiesto, el navegador descargará los tres archivos en el directorio raíz del sitio web. Entonces, siempre que el usuario no esté conectado a Internet, los recursos seguirán estando disponibles.

HTML5 Application Cache

NETWORK

```
NETWORK:
login.php
```

```
NETWORK:
*
```

Un asterisco se puede utilizar para indicar que todos los demás recursos / archivos requieren una conexión a Internet.

HTML5 Application Cache

FALLBACK

FALLBACK:
/html/ /offline.html

La sección **FALLBACK** especifica que "offline.html" debe ser servido en lugar de todos los archivos en el directorio /html/, en caso de una conexión a Internet no se puede establecer

Nota: La primera URI es el recurso, el segundo es el retorno (Fallback)

HTML5 Application Cache

- Consideraciones de **Application Cache**
  - Tener cuidado con lo que almacena en caché.
  - Una vez que un archivo se almacena en caché, el navegador seguirá mostrando la versión en caché, incluso si cambia el archivo en el servidor. Para garantizar que el navegador actualiza el almacenamiento en caché, es necesario cambiar el archivo de manifiesto.
  - Nota: Los navegadores pueden tener diferentes límites de tamaño para los datos almacenados en caché (algunos navegadores tienen un límite de 5 MB por sitio).

HTML5 Web Workers

- ¿Qué es un **Web Worker**?
- Con la ejecución de scripts en una página HTML, la página deja de responder hasta que se termine la ejecución del script.

- Un **Web Worker** es un JavaScript que se ejecuta en segundo plano, de forma independiente de otras secuencias de comandos, sin afectar el rendimiento de la página.

- Se puede seguir haciendo otras tareas: clicks, seleccionar cosas, etc, mientras que el web workers se ejecuta en segundo plano.

HTML5 Web Workers

Antes de utilizar se debe comprobar la compatibilidad con los navegadores.

```
if(typeof(Worker) !== "undefined") {
    // Si! Soporta Web worker!
    // código.....
} else {
    // Lo siento el navegandor no soporta web worker}
```

- Creamos un archivo Web Worker
  - Creamos un Web Worker en un JavaScript externo.
  - Un script que cuenta. El script se almacena en el archivo "demo_workers.js":

```
var i = 0;
function timedCount() {
    i = i + 1;
    postMessage(i);
    setTimeout("timedCount()",500);
}
timedCount();
```

- El método **postMessage ()** es importante - que se utiliza para enviar un mensaje de vuelta a la página HTML.
- Nota: Los web workers normalmente no se utilizan para este tipo de scripts simples, pero sí para tareas de uso más intensivo de CPU.

HTML5 Web Workers

Crear un objeto Web Worker.

```
if(typeof(w) == "undefined") {
    w = new Worker("demo_workers.js");
}
```

- Entonces podremos enviar y recibir mensajes desde el Web Worker definido en **w**.

- Añadir evento "**onmessage**" escuchar al Web Worker .

```
w.onmessage = function(event){
    document.getElementById("result").innerHTML = event.data;
};
```

- Cuando el web worker postee un mensaje, se ejecuta el código dentro del detector de eventos. Los datos del web worker se almacenan en event.data.

HTML5 Web Workers

Terminar un Web Worker

Cuando se crea un objeto Web Worker, continuará escuchando los mensajes (incluso después de la secuencia de comandos externo se termina) hasta que se termina.

Para terminar un Web Worker, y liberar los recursos del navegador / computadora, utilice el método terminate():

```
w.terminate();
```

- Reutilizar el Web Worker
  - Si se establece la variable de Web Worker indefinida (undefined), después de que se ha terminado, se puede reutilizar el código:

```
w = undefined;
```

- Web Workers y el DOM
  - Dado que los Web Workers están en archivos externos, no tienen acceso a los siguientes objetos JavaScript:
    - El Objeto window
    - El Objeto document
    - El Objeto parent