

Práctica 02

Programación Web dinámica con interpretación en el cliente

Fundamentos de Aplicaciones y Servicios Telemáticos
2º Curso Grado en Ingeniería de Tecnologías de Telecomunicación
Departamento de Ingeniería Telemática (DIT)

Universidad de Sevilla



Ignacio Campos Rivera
Francisco José Fernández Jiménez
José Ángel Gómez Argudo
Francisco Javier Muñoz Calle
Juan Antonio Ternero Muñiz

@2018

ÍNDICE



1	Programación Web dinámica cliente.....	3
1.1	Introducción.....	3
1.2	Objetivo.....	3
1.3	Documentación de apoyo.....	3
2	Ficheros para la práctica.....	3
3	ECMAScript.....	5
3.1	Historia.....	5
3.2	Primer script: variables y estructuras de control.....	5
3.3	Depuración de ECMAScript con el navegador Firefox.....	10
3.4	Objetos.....	13
3.5	Funciones.....	19
3.5.1	Funciones predefinidas: cajas emergentes.....	20
3.6	Eventos.....	21
3.6.1	Manejadores de eventos.....	22
3.7	DOM: Document Object Model.....	25
3.7.1	Introducción a DOM.....	25
3.7.2	Nodos, métodos y propiedades de DOM desde ECMAScript.....	28
3.7.3	El documento del DOM.....	30
3.7.4	Los elementos del DOM.....	32
3.7.5	El documento y los elementos del DOM.....	35
3.7.6	Atributos en DOM.....	36
3.7.7	Lista de nodos y colección del HTML DOM.....	38
3.7.8	Eventos en el DOM HTML.....	40
3.7.9	Navegación a través de los nodos del HTML DOM.....	42
3.8	Validación de Formularios.....	44
3.9	Eventos de temporización.....	47
3.10	Etiqueta noscript.....	49
4	AJAX.....	51

4.1	AJAX (Asynchronous JavaScript And XML).....	51
4.2	XmlHttpRequest y primera aplicación AJAX	52
4.3	JSON	55
5	Aplicación comentarios	57
5.1	Aclaraciones sobre las funciones	58
6	Propuesta de aplicación de la práctica.....	60
6.1	HTML y CSS	60
6.2	XML	62
6.3	ECMAScript.....	62
6.4	AJAX.....	66
7	Anexo (no evaluable)	67
7.1	JQuery	67
7.2	Clases y herencia con ECMAScript	68

1 Programación Web dinámica cliente

1.1 Introducción

En temas anteriores se presentaron las distintas tecnologías de programación web, estática y dinámica. Dentro de la programación web dinámica se distinguía entre programación web dinámica en el lado del cliente y en el lado del servidor. La programación web estática se desarrolló en el tema anterior. En este tema se desarrolla la programación web dinámica en el lado del cliente, y en temas posteriores se desarrollará la programación web dinámica en el lado del servidor.

1.2 Objetivo

El objetivo de esta práctica es introducir al alumno en el uso de la programación web dinámica en el lado del cliente. En concreto, en esta práctica nos centraremos en la sintaxis de ECMAScript, el lenguaje de programación del lado del cliente más utilizado. La verdadera potencia de ECMAScript nace al usarlo de forma conjunta al Modelo de Objetos del Documento, DOM, y de bibliotecas de utilidades (como por ejemplo jQuery). Posteriormente estudiará el uso básico de peticiones asíncronas a través de uso de la tecnología conocida como AJAX.

1.3 Documentación de apoyo

- Standard ECMA-262: ECMA Script Language Specification:
<http://www.ecma-international.org/publications/standards/Ecma-262.htm>
- W3C: Document Object Model (DOM):
<http://www.w3.org/DOM/>
- Tutoriales Librosweb: AJAX:
<http://www.librosweb.es/ajax/>
- W3schools JS Tutorial:
<http://www.w3schools.com/js/>
- Librosweb: JavaScript
<http://www.librosweb.es/javascript/>
- AJAX, Mozilla Developer Network:
<https://developer.mozilla.org/en/AJAX/>
- Javascript:
<https://developer.mozilla.org/en/JavaScript/Reference>
- W3schools: trying HTML, CSS y Javascript online:
http://www.w3schools.com/tags/tryit.asp?filename=tryhtml_doctype

2 Ficheros para la práctica

En este apartado se recogen algunas cuestiones para facilitar el uso del entorno de trabajo y el desarrollo de la práctica.

En las prácticas anteriores se ha introducido el uso de distintos entornos de desarrollo, y a lo largo de esta memoria, a no ser que se indique lo contrario, podrá trabajar con el IDE o editor que prefiera, y visualizar el fichero en el navegador que prefiera.

Por otra parte se aplica todo lo presentado en temas previos sobre despliegue y localización de los archivos (URL y directorios locales correspondientes) para ser servidos por un servidor web.

DESCARGA DE LOS EJEMPLOS DE ESTA PRÁCTICA

Como irá observando, esta práctica se basa en el uso de múltiples ficheros. Para descargarlos realice los siguientes pasos:

1º Acceda a la página Web de la Asignatura en Enseñanza Virtual y vaya a la misma carpeta en la que se encuentra la memoria de esta práctica. En dicha carpeta encontrará el fichero "FAST_t02-ficheros.tar.gz". Descárguelo a su máquina local, dentro del directorio "/home/dit/".

2º Con el usuario "dit", ejecute los siguientes comandos para descomprimir el archivo:

```
cd /home/dit/  
tar xfvz ./FAST_t02-ficheros.tar.gz
```

A partir de ese momento, dispondrá de los distintos ficheros en las carpetas "/home/dit/workspace/AppWeb/WebContent/p02", y podrá acceder a los ficheros usando la URL

`http://localhost:8080/AppWeb/p02/index.html`

Verifique que en la carpeta /home/dit/workspace/AppWeb/WebContent/p02 están los ficheros de esta práctica, organizados en carpetas: ajax, js, jsp y objetos. En cada uno de los apartados de esta práctica, cuando se haga referencia al nombre de un fichero, se supondrá localizado en la carpeta que le corresponda.

Recuerde que /home/dit/workspace/AppWeb/WebContent/ es el directorio de trabajo de Eclipse, pero que el alias asociado a/AppWeb es /home/dit/tomcat/webapps/AppWeb/, que es donde busca el servidor web de tomcat. Lo que ocurre es que cuando se edita un fichero en Eclipse (o se refresca el directorio con F5), Eclipse copia los ficheros modificados al directorio de tomcat (incluso puede que borre ficheros). Compruebe que una vez que ha arrancado Eclipse (habiendo previamente descargado los ficheros de esta práctica y de la anterior) los ficheros se han copiado automáticamente en el directorio correspondiente de tomcat.

3 ECMAScript

3.1 Historia

ECMAScript es un lenguaje de programación interpretado, sus variables pueden cambiar de tipo, maneja objetos, y su cualidad principal es todos los navegadores modernos interpretan el código ECMAScript integrado en las páginas web. Además, para interactuar con una página web se provee al lenguaje ECMAScript de una implementación del Document Object Model (DOM).

ECMAScript tiene sus orígenes en el lenguaje JavaScript, desarrollado originalmente por Netscape (año 1995), y Microsoft desarrolló un lenguaje similar, JScript.

En junio de 1997 fue adoptado como estándar ECMA-262 (ECMA International es un organismo internacional de normalización y soporte técnico) y ya se han publicado múltiples ediciones, conocidas también por el año de publicación (por ejemplo, la 8ª edición, publicada en 2017, es conocida como ECMAScript-8 y también como ECMA-2017). Puede encontrar la última edición en:

<http://www.ecma-international.org/publications/standards/Ecma-262.htm>

También está publicado como norma ISO/IEC 16262. Por ejemplo, ISO/IEC 16262:2011 (que se corresponde con ECMAScript-5, de 2011). Puede consultar la última versión, buscando en:

<http://www.iso.org>

Además, el World Wide Web Consortium (W3C) diseñó el estándar Document Object Model (DOM, o Modelo de Objetos del Documento en español), que incorporan prácticamente todos los navegadores que estén actualizados, y a través del cual se puede acceder a las distintas partes del documento HTML. Esta norma está disponible en:

<http://www.w3.org/DOM/>

3.2 Primer script: variables y estructuras de control

ECMAScript proporciona los siguientes tipos básicos: `Undefined`, `Null`, `Boolean`, `Number`, y `String` (y también `Symbol` partir de ECMAScript-6). Algo que sorprende en ECMAScript es que al definir las variables usando la palabra reservada `var` no se indica el tipo.

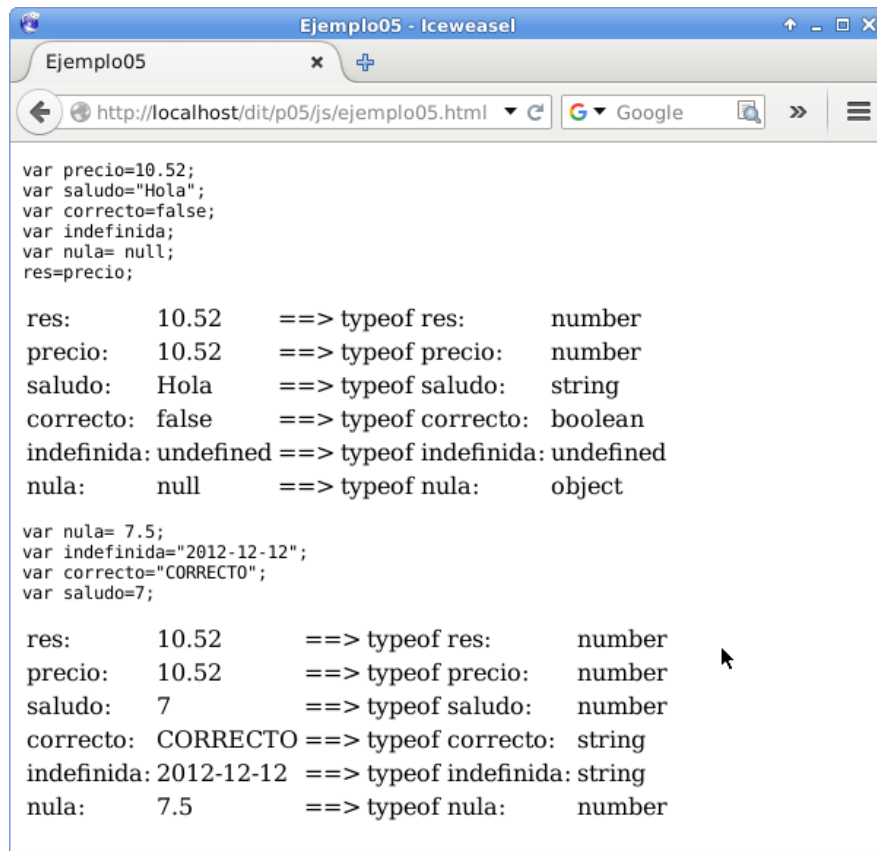
```
var precio;  
var nombre="Alfredo Rivera";  
var error=false;  
var nula=null;
```

El tipo inicial de la variable depende del valor asignado. Si no se le da un valor inicial en el momento de la definición, la variable es del tipo `Undefined`. El tipo de la variable se puede consultar con `typeof`, como verá en un ejemplo posterior. Además ese tipo puede cambiar a lo largo de la interpretación del código, simplemente asignándole a la variable un valor de un tipo distinto.

El siguiente ejemplo lo vamos a desarrollar sobre el archivo `ejemplo05.html`. Observe que se definen varias variables, y se imprimen sus valores, y sus tipos usando `typeof`. Estos además se modifican

durante la ejecución cambiando de tipos. Las variables del ejemplo son de los tipos básicos. Cuando más adelante se vean los objetos se observará mejor que las variables guardan referencias a esos objetos (similar a Java).

El resultado de presentar la página web en un navegador sería:



The screenshot shows a web browser window with the title "Ejemplo05 - Iceweasel". The address bar shows the URL "http://localhost/dit/p05/js/ejemplo05.html". The page content displays two blocks of JavaScript code and their corresponding variable types as determined by the `typeof` operator.

```
var precio=10.52;
var saludo="Hola";
var correcto=false;
var indefinida;
var nula= null;
res=precio;

res:      10.52    ==> typeof res:      number
precio:   10.52    ==> typeof precio:   number
saludo:    Hola    ==> typeof saludo:   string
correcto:  false   ==> typeof correcto: boolean
indefinida: undefined ==> typeof indefinida: undefined
nula:      null    ==> typeof nula:     object

var nula= 7.5;
var indefinida="2012-12-12";
var correcto="CORRECTO";
var saludo=7;

res:      10.52    ==> typeof res:      number
precio:   10.52    ==> typeof precio:   number
saludo:    7        ==> typeof saludo:   number
correcto:  CORRECTO ==> typeof correcto: string
indefinida: 2012-12-12 ==> typeof indefinida: string
nula:      7.5      ==> typeof nula:     number
```

Pruebe los ejemplos, arrancando el servidor web con Eclipse-Tomcat (tal como se explicó en la práctica anterior) y accediendo desde un navegador a:

`http://localhost:8080/AppWeb/p02`

TAREAS I

- 1º Muestre el fichero ejemplo05.html usando un navegador web. Analice su contenido. Acceda a través de http en el directorio p02/js:

<http://localhost:8080/AppWeb/p02/js/ejemplo05.html>

- 2º Observe que las variables sin valor inicial son del tipo undefined, y que, al cambiar el valor almacenado en una variable, cambia de tipo. Note también que la variable `res` se ha declarado sin usar la palabra reservada `var`.
- 3º Asigne, desde el editor de Eclipse, a la variable `saludo` el valor `true` (sin comillas). ¿Qué valor se imprime? ¿Cuál es su tipo?
- 4º Asigne a la variable `saludo` el valor `"true"` (ahora entre comillas). ¿Qué valor se imprime? ¿Cuál es su tipo?
- 5º Añada una fila a la tabla mostrando el valor y el tipo de la variable de nombre `x` (no declarada ni asignada previamente). ¿Qué ocurre? Pulse F12 y seleccione la pestaña Consola. ¿Aparece algún error? (Asegúrese que en filtrar salida tiene marcadas al menos las casillas Errores y Depurar).
- 6º Añada al principio del script (donde está la escritura de la tabla) la declaración de la variable de nombre `x`. ¿Qué ocurre?
- 7º Mueva al final del script la declaración de la variable de nombre `x`. ¿Qué ocurre? Consulte una explicación en:
- https://www.w3schools.com/js/js_hoisting.asp
- 7º Vuelva a poner al principio del script la declaración de la variable de nombre `x`: es una buena práctica y se deben poner SIEMPRE las declaraciones al principio.
- 8º Sustituya la declaración de la variable de nombre `x` por una asignación (`x=3;`) y compruebe si se imprime su tipo y valor. Añada al principio del script (donde está la escritura de la tabla) una línea que contenga (importante las comillas):
- ```
"use strict";
```
- ¿aparece algún error en la consola? Ponga la palabra reservada `var` al principio de la asignación para convertirla en declaración (y asignación). ¿Ha desaparecido el error de la consola? Consulte una explicación en:
- [https://www.w3schools.com/js/js\\_strict.asp](https://www.w3schools.com/js/js_strict.asp)
- 9º Añada al principio del primer script (el que está al principio de body) una línea que contenga `"use strict";` ¿aparece algún error en la consola? Modifique la asignación de la variable `res` para que no haya error. Es una buena práctica y se debe poner SIEMPRE al principio de cada `<script>` la línea `"use strict";`
- 10º Analice el código fuente desde el navegador (botón derecho, “ver código fuente”). ¿Puede ver los comentarios de ECMAScript dentro de HTML en el navegador?



Nota: Si al hacer una modificación en el servidor ésta no se refleja en el navegador puede deberse a varias razones:

- El servidor no ha detectado el cambio. Para corregirlo en Eclipse pulse botón derecho > Refresh sobre AppWeb (o pulse F5)
- El navegador está usando su caché. Para corregirlo y borrar la caché puede:
  - o Recargar página borrando la caché con Ctl-F5 (no siempre es efectivo)
  - o Borrar la caché del navegador en menú > preferencias > privacidad > limpiar historial
  - o Añadir a la URL una ? y un valor que sea distinto cada vez. Por ejemplo:

http://localhost:8080/AppWeb/p02/js/ejemplo05.html?1

http://localhost:8080/AppWeb/p02/js/ejemplo05.html?2

http://localhost:8080/AppWeb/p02/js/ejemplo05.html?3

En el siguiente ejemplo se muestra la diferencia entre los operadores de igualdad (==) e igualdad estricta (===).

#### **ejemplo08.html**

```
<html>
<body>
<script type="text/javascript">
"use strict";
var x=5;
var y="5";
if (x == y)
 document.write('
x que vale ', x, ' de tipo ', typeof x,
 ' es igual(==) a y que vale ', y, ' de tipo ', typeof y);
else
 document.write('
x que vale ', x, ' de tipo ', typeof x,
 ' NO es igual(==) a y que vale ', y, ' de tipo ', typeof y);
document.write('<hr />')
if (x === y)
 document.write('
x que vale ', x, ' de tipo ', typeof x,
 ' es igual(===) a y que vale ', y, ' de tipo ', typeof y);
else
 document.write('
x que vale ', x, ' de tipo ', typeof x,
 ' NO es igual(===) a y que vale ', y, ' de tipo ', typeof y);
</script>
</body>
</html>
```

## TAREAS 2

- 1° Acceda desde el navegador mediante http a ejemplo08 (dentro de p02/js)
- 2° ¿Qué aparece en pantalla? Analice el código fuente desde el navegador (botón derecho, “ver código fuente”).
- 3° Modifique la asignación de la variable y para que ahora valga 5 pero sea numérica.
- 4° ¿Qué aparece en pantalla?

Si vamos a trabajar con tablas o vectores, lo más sencillo es usar el tipo Array (tabla) de ECMAScript. Observe en el ejemplo cómo se define una tabla, y cómo se recorre una tabla con un bucle for. Además observe el uso de la propiedad length y del método join.

Consulte el funcionamiento del bucle for en:

[https://www.w3schools.com/js/js\\_loop\\_for.asp](https://www.w3schools.com/js/js_loop_for.asp)

Consulte la propiedad length en:

[https://www.w3schools.com/js/js\\_strings.asp](https://www.w3schools.com/js/js_strings.asp)

## ejemplo09.html

```
<html>
<body>
<script>
"use strict";
var i;
var meses = ["Enero", "Febrero", "Marzo", "Abril", "Mayo",
 "Junio", "Julio", "Agosto", "Septiembre", "Octubre",
 "Noviembre", "Diciembre"];
for (i=0; i<meses.length; i++)
 document.write("meses[" + i + "]=", meses[i], "
");

document.write("
", meses.join("_"), "

");

for (i=0; i<meses.length; i++)
 document.write("meses[" + i + "]=", meses[i], "
");
</script>
</body>
</html>
```

## TAREAS 3

- 1º Acceda desde el navegador mediante http a `js/ejemplo09.html`
- 2º Analice el código fuente desde el navegador (botón derecho, “ver código fuente”).
- 3º Busque en w3schools el funcionamiento del método `join`  
`http://www.w3schools.com/js/js_array_methods.asp`
- 4º Modifique el código en el servidor para mostrar en el segundo bucle los meses en orden inverso
- 5º Modifique el código para mostrar sólo los meses impares.

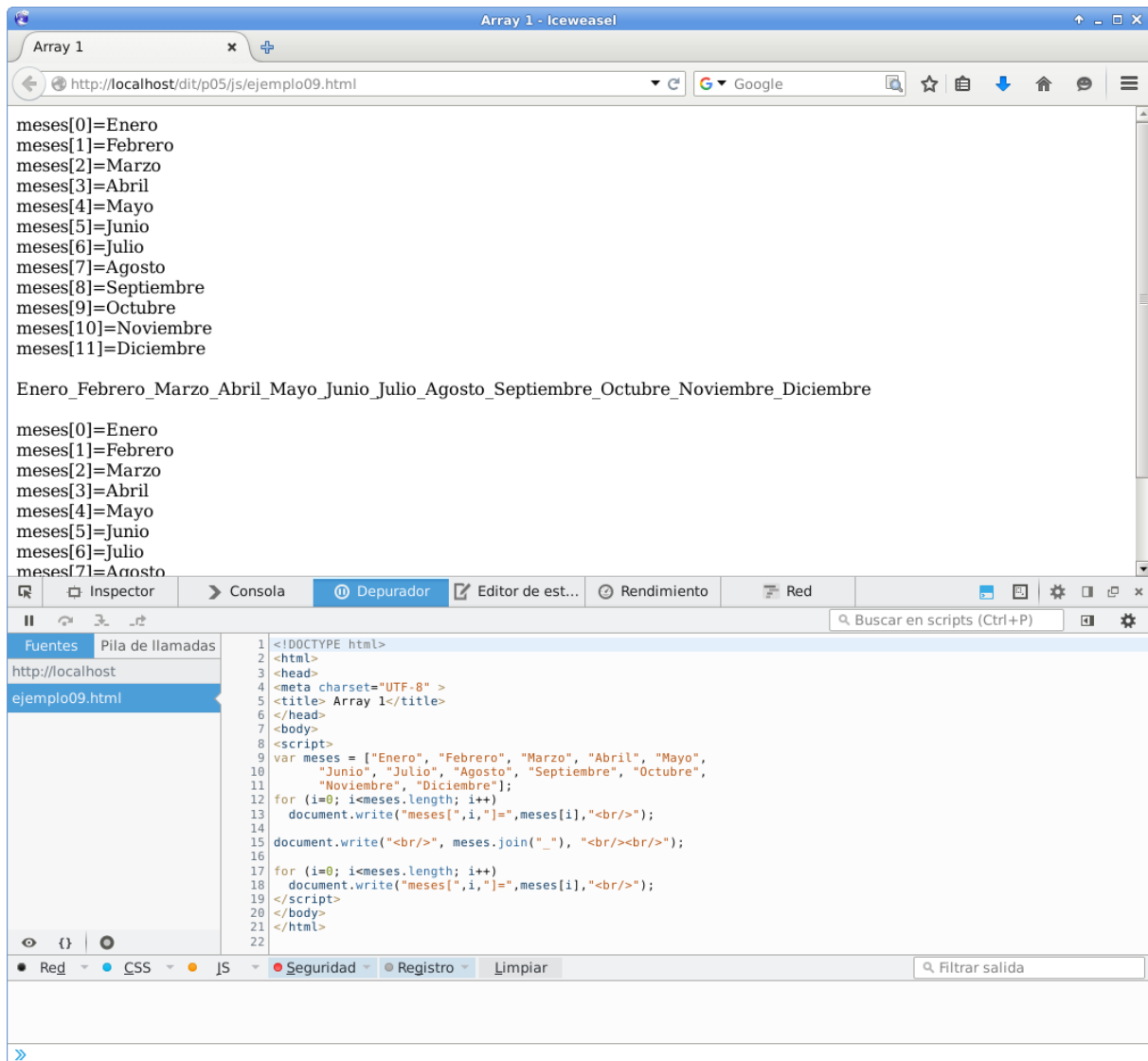
Una tabla (`Array`) es un tipo especial de objeto a cuyos elementos se acceden con números (empezando por el 0). ECMAScript no soporta las tablas asociativas (a los elementos se acceden con nombres), para ello están los objetos (`Object`), que se ven más adelante.

Aprovecharemos este ejemplo para hacer una breve presentación del depurador de Firefox. Esto nos ayudará en la corrección de nuestros ejemplos y ejercicios.

### 3.3 Depuración de ECMAScript con el navegador Firefox

Con los navegadores actuales se puede analizar, editar, monitorizar y depurar el código fuente CSS, HTML y ECMAScript de una página web, basta con pulsar la tecla F12 (o menú “Desarrollador > Depurador” o “botón derecho > Inspeccionar elemento” o Ctl-Shift-S).

Una vez activado el depurador deberá recargar la página, por ejemplo usando la tecla F5 (o ctrl+F5 que además borra la caché). El aspecto del depurador en Firefox es similar al de la figura:



En el panel de script podrá añadir puntos de ruptura para detener la ejecución y reanudarla posteriormente, o comprobar los valores de las variables, por ejemplo colocando el ratón sobre las mismas (o utilizando la pestaña "Observar"). Para fijar un punto de ruptura basta hacer clic con el ratón sobre el número de línea en que desea introducir el punto de ruptura.

Una vez fijados los puntos de ruptura, debe cargar de nuevo la página web para que se ejecute el código. El intérprete detendrá la ejecución en el primer punto de ruptura que encuentre. En ese momento esperará que seleccione alguna de las opciones posibles, que se encuentran representadas por 4 iconos que aparecen en la parte superior del panel de script:



De izquierda a derecha:

1. Continuar hasta siguiente punto de ruptura, F8
2. Saltar (equivalente a next de gdb) , F10
3. Entrar (equivalente a step de gdb), F11
4. Salir, Shif-F11

Puede encontrar información adicional de cómo depurar en:

<https://developer.mozilla.org/en-US/docs/Tools/Debugger>

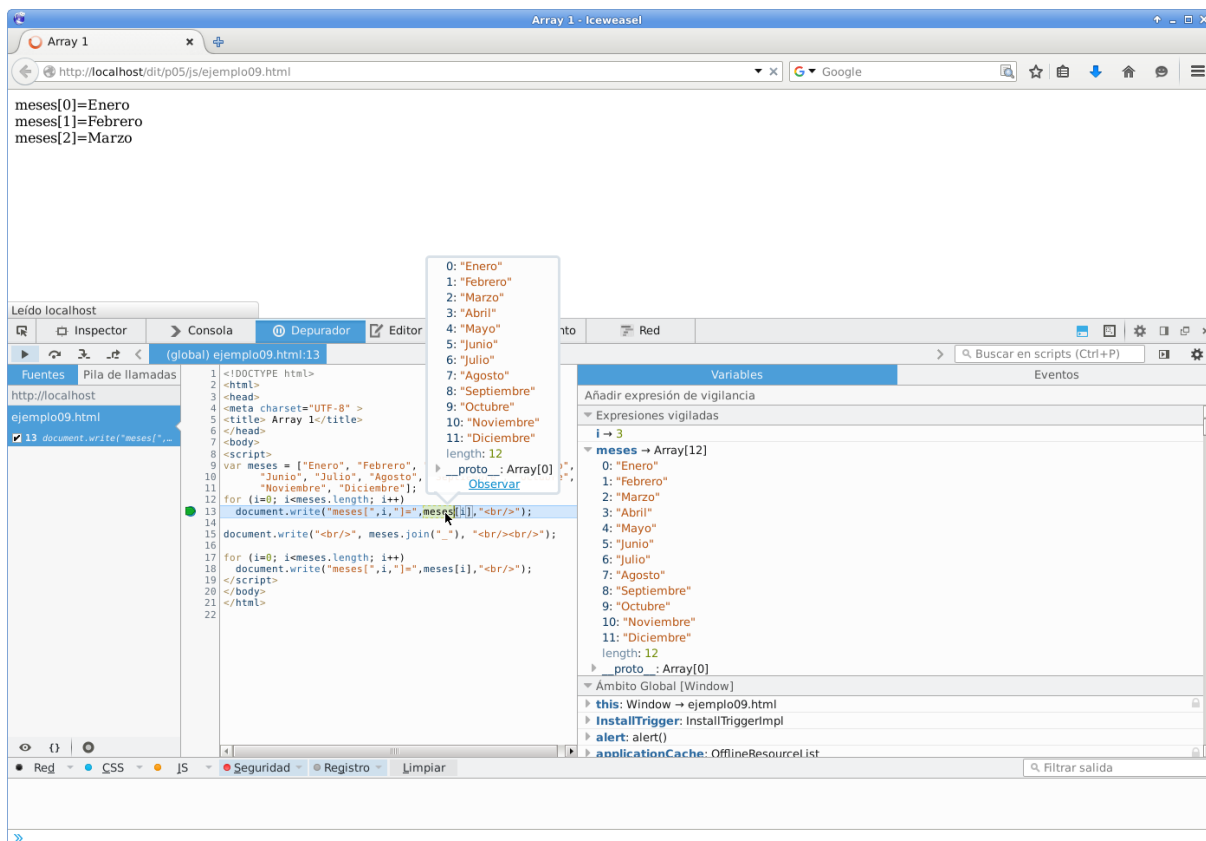
En concreto, el significado de los iconos en:

[https://developer.mozilla.org/en-US/docs/Tools/Debugger/How\\_to/Step\\_through\\_code](https://developer.mozilla.org/en-US/docs/Tools/Debugger/How_to/Step_through_code)

Es posible crear un punto de ruptura condicional: una vez fijado el punto de ruptura, pulse con el botón derecho sobre el punto de ruptura y podrá introducir una condición lógica.

## TAREAS

- 1º Acceda con el navegador mediante http al fichero js/ejemplo09.html
- 2º Active el depurador pulsando F12. Recargue la página.
- 3º Incluya un punto de ruptura en la sentencia document.write del primer bucle.  
Recargue la página para comprobar que se detiene en dicho punto.
- 4º Pruebe las distintas opciones para reanudar la ejecución: Continuar, Entrar, Saltar...  
(tenga en cuenta que no es una función definida por el usuario).
- 5º Modifique el punto de ruptura para incluir la condición `i==5`. Cargue de nuevo la página para comprobar que se detiene tras imprimir algunos de los meses.
- 6º Compruebe el valor de las variables pasando el ratón sobre el nombre de las mismas, y también marcando la opción “observar”.



Ahora vamos a activar la consola, y provocar un error sintáctico en ECMAScript para observar cómo se comporta Firefox:

## TAREAS

- 1º Edite en el servidor el fichero anterior, `js/ejemplo09.html`, y borre el paréntesis que cierra la condición del `for` del primer bucle.
- 2º Inicie Firefox con el fichero `js/ejemplo09.html` y observe que la pantalla se queda en blanco, pero no muestra ningún error.
- 3º Active el depurador pulsando F12.
- 4º Recargue de nuevo la página. Observe en el panel consola la indicación del error sintáctico encontrado.
- 5º Edite el fichero para corregir el error.

## 3.4 Objetos

Para definir un objeto creado por el usuario se puede usar la palabra reservada `new` y el tipo `Object`. Una vez creado el objeto, se le pueden añadir de forma dinámica propiedades y métodos. Observe cómo se crea un objeto (con `new` o en forma abreviada) y cómo se le añade una propiedad y se accede a ella:

**objetos-01.js**

```
"use strict";
var miobjeto = new Object();

// Notación "abreviada"
var tuobjeto = {};

// definiendo la propiedad
miobjeto.id="A001";
tuobjeto.name="Ana";

// accediendo a las propiedades
alert(miobjeto.id+": "+ tuobjeto.name);
...
```

Si además desea añadir métodos al objeto puede hacerlo de dos formas: a) usando una función anónima, o b) definiendo una función que posteriormente se asigna al objeto. Observe las dos formas en el siguiente ejemplo, y cómo se utiliza `this` dentro de los métodos, haciendo referencia al objeto que invoca el método.

**objetos-01.js**

```
...
// creando métodos: a) función no definida previamente,
miobjeto.muestraId = function () {
 alert(this.id);
}
// Invocación del método
miobjeto.muestraId();

// b) asignando una función definida previamente
// Para asignar una función externa al método de un objeto,
// basta indicar el nombre de la función sin paréntesis.

function muestraName() { // no tiene por qué llamarse igual..
 alert(this.name);
}
tuobjeto.muestraName = muestraName; // Sin paréntesis

tuobjeto.name="Pablo";
tuobjeto.muestraName();
```

## TAREAS

- 1° Acceda con el navegador mediante `http` al fichero `objetos/objetos.html`
- 2° Analice el código fuente desde el navegador (botón derecho, “ver código fuente”). Observe la etiqueta `script`. ¿Qué atributo contiene? ¿Cuál es el nombre del fichero que contiene código ECMAScript? ¿Dónde se encuentra el fichero?
- 3° Vuelva a la página de `objetos.html`, active el depurador pulsando F12.
- 4° Recargue de nuevo la página. Observe en el panel Inspector el código HTML. Observe en el panel Depurador el código ECMAScript. ¿Aparece el nombre del fichero?
- 5° Modifique el fichero `js` en el servidor: Defina un objeto de nombre `suobjeto`, con la propiedad `name` de valor “Pepe” y el método `muestraNombre` enlazado a la función `muestraName`. Llame a la función `muestraNombre` de `suobjeto`.
- 6° Vuelva a repetir los pasos iniciales para comprobar el funcionamiento.
- 7° Ponga como comentario la línea que contiene la declaración de `suobjeto`. ¿Qué ocurre? ¿Se observa algún error en la consola, con la casilla JS activada?
- 8° Quite el comentario de la declaración y póngalo en el enlazado a la función. ¿Qué ocurre?
- 9° Quite el comentario del enlazado y cambie la propiedad para que en vez de `name` sea `nam`. ¿Qué ocurre? Establezca un punto de ruptura en la llamada a la función `muestraNombre` de `suobjeto`. Cuando se alcance, entre en la ejecución de la función (pulse el icono Entrar o F11). Antes de ejecutar `alert` intente ver en la pestaña de Variables (ventana a la derecha) el nombre de la propiedad del objeto señalado por `this` cuyo valor es Pepe.

El siguiente paso es definir clases que puedan ser usadas para definir objetos. Pero ECMAScript (versión ECMA-5) no permite crear clases como C++ o Java. A pesar de ello, hay formas de simular las clases usando funciones constructoras y `prototype` (ver Anexo).

En ECMAScript se emula el funcionamiento de los constructores mediante el uso de funciones. El objeto `Function` es utilizado como objeto instanciable en ECMAScript, y el cuerpo de la función es el constructor de nuestro objeto. Una vez que tenemos un constructor, podemos llamarlo con el operador `new`. A estas funciones se les denomina “funciones constructoras”.

En el siguiente ejemplo se define una función “constructora”, `Impresora`, que posteriormente se utiliza con el operador `new` para crear el objeto. Observe que es habitual comenzar el nombre de la función constructora en mayúsculas para distinguirla del resto de las funciones.



## objetos-02.js

```
// Creamos nuestra función constructora
"use strict";
function Impresora (id){
 this.id = id;
 this.muestraId = function(){
 alert(this.id);
 };
};

// Definido el constructor, se crea objeto
// con llamada a new con función constructora
var laser = new Impresora("GH-Laser1345");
// uso del método y la propiedad
laser.muestraId();
alert("Id de la impresora: " + laser.id);

// segundo objeto con la misma clase
var tintaColor = new Impresora("DEF-Color-4500");
tintaColor.muestraId();
```

## TAREAS

- 1º Cree un nuevo fichero dentro del directorio objetos de nombre objetos2.html, similar a objetos.html, pero que ahora incluya el fichero objetos-02.js
- 2º Acceda con el navegador mediante http al fichero objetos/objetos2.html y compruebe su funcionamiento.
- 3º Modifique el código para que la función constructora tenga un segundo parámetro de nombre color y se lo asigne a una nueva propiedad llamada color. ¿Hay que especificar el tipo?
- 4º Modifique el código de la función constructora para añadir un método de nombre esColor asociado a una función sin nombre y sin parámetros que acceda a la propiedad llamada color (creada en el apartado anterior) y que compruebe si es booleana (typeof color == "boolean"). Si la propiedad color es booleana y tiene el valor verdadero, debe aparecer en pantalla el mensaje “es de color” y si no “es b/n”. Si la propiedad color no es booleana debe aparecer "El segundo parámetro del constructor debe ser booleano". Incluya en las llamadas a la función constructora el segundo parámetro, en un caso true y en otro false, y compruebe el funcionamiento. Modifique el código poniendo en las llamadas el segundo parámetro entre comillas. Pruébalo.
- 5º Modifique el código y quite las comillas de boolean en la comprobación del tipo (typeof color == boolean). Pruebe el funcionamiento, ¿funciona ahora? Para intentar encontrar el problema ponga un punto de ruptura en la línea donde está (typeof color == boolean) y añada typeof color en la ventana derecha en Añadir expresión de vigilancia ¿qué valor toma? Añada también la variable color. Ejecute paso a paso y compruebe el valor que va tomando.

Si no necesita usar clases, puede usar los objetos de una forma similar a los tipos básicos o nativos. Al igual que hemos usado la notación abreviada para definir una tabla, se puede usar para definir un objeto, como se observa en el siguiente ejemplo. Observe la sintaxis de la definición: se usan las llaves para definir el objeto, la coma (",") para separar las propiedades, y los dos puntos (":") para separar la propiedad de su valor. Además podemos recorrer las propiedades de un objeto, sin conocer sus identificadores, mediante el bucle `for in`.

**ejemplo10.html**

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" />
<title> Objeto </title>
</head>
<body>
<script type="text/javascript">
"use strict";
var mes = {"nombre": "Enero", "dias": 31, "abr": "ene"};
var i;
document.write("<p>");
for (i in mes)
 document.write("i=" + i + ", mes[i]=" + mes[i], "
 ");
document.write("</p>");
</script>
</body>
</html>
```

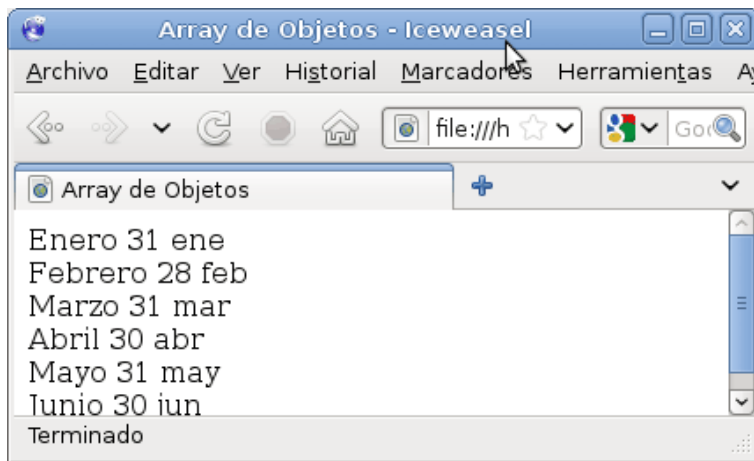
**TAREAS**

- 1º Abra el navegador con el fichero `js/ejemplo10.html`.
- 2º Abra con un editor el fichero anterior. Modifique el código para añadir una propiedad adicional, de nombre `"name_en"` que almacenará el nombre del mes en inglés. Modifique el código para imprimir todas las propiedades del mes.

Vamos a dar un paso más, y definir una tabla de objetos siguiendo la notación abreviada. Se creará una tabla de objetos `mes`, con nombre, número de días y abreviatura del nombre del mes.

**TAREAS**

- 1º A partir de los ficheros `ejemplo09.html` y `ejemplo10.html` cree una variable `tablaMes` de tipo Array que contenga 6 objetos `Mes` (para los 6 primeros meses del año). Cree para ello una función constructora para `Mes` que usará al rellenar la tabla, donde cada mes tendrá como propiedades el nombre, número de días y abreviatura del nombre del mes. Una vez rellena la tabla utilice bucles para mostrar la información: un bucle numérico para recorrer la tabla y un bucle `for in` para recorrer las propiedades de cada elemento. Deberá imprimir una línea para cada mes, y en cada línea todos los datos del mes. El aspecto será similar al de la figura:



En el siguiente ejercicio vamos a usar variables de tipo string, definiéndolas como objetos (o con la notación abreviada). En ambos casos se muestra la cadena, su longitud, y cómo convertirlas en mayúsculas. Finalmente se presenta el uso del método split.

El método split se usa para obtener una tabla de cadenas a partir de una cadena (string). Devuelve como resultado la tabla.

```
cadena.split(separador, límite);
```

El separador, opcional, especifica el carácter que se usará para dividir la cadena. Si se omite, se devuelve toda la cadena. Si el separador es una cadena vacía (""), se divide en caracteres. El límite, opcional, especifica el número de cadenas que se devuelven.

#### ejemplo06.html

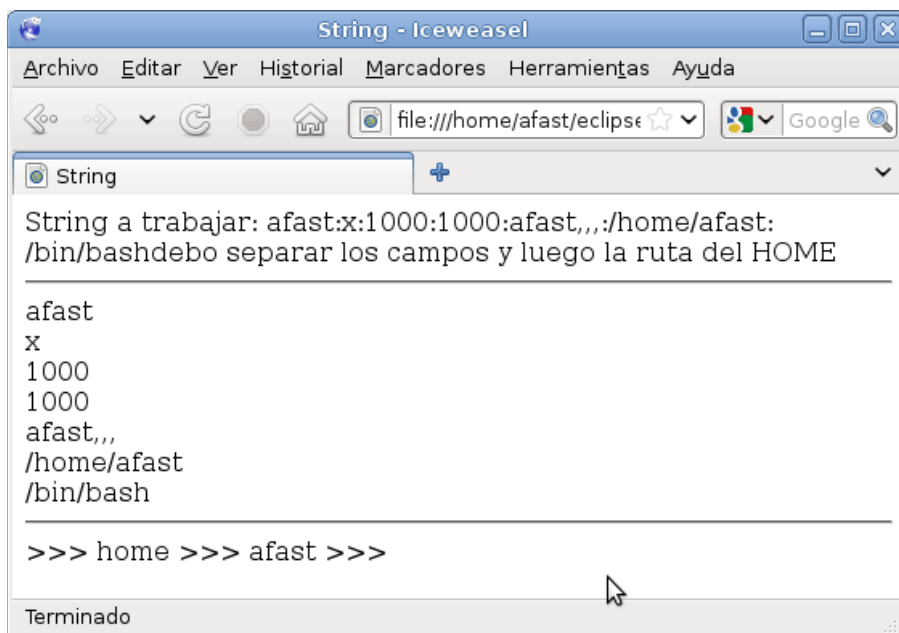
```
...
<body>
<script type="text/javascript">
var saludo="Hola Alumnos de Fast"
document.write("
Saludo: ", saludo,
 "
Propiedad length: ", saludo.length,
 "
Método toUpperCase(): ", saludo.toUpperCase(),
 "
Saludo: ", saludo, "<hr/>");

var cadena = new String("Hola Alumnos de Fast2");
document.write("
Cadena: ", cadena,
 "
Propiedad length: ", cadena.length,
 "
Método toUpperCase(): ", cadena.toUpperCase(),
 "
Cadena: ", cadena, "<hr/>");

// Uso de split
document.write(cadena.split() + "
");
document.write(cadena.split(" ") + "
");
document.write(cadena.split("") + "
");
document.write(cadena.split(" ",3));
</script> </body>...
```

## TAREAS

- 1º Abra el navegador con el fichero js/ejemplo06.html
- 2º Analice el código y el funcionamiento del método split(). Modifique el ejemplo para que el valor devuelto por cadena.split(" ",3) se asigne a una variable, e imprima el contenido de esa variable recorriéndola con un bucle de índice numérico. Imprima después la variable cadena ¿se ha modificado?
- 3º Al final del ejemplo anterior se incluye la definición de una variable de tipo string de nombre user. Modifique el código para que se muestre por pantalla los distintos campos que conforman esa cadena de texto, que están separados por el carácter ":" (usuario, clave, grupo, etc).
- 4º Modifique el fichero anterior para que imprima el nombre de cada uno de los directorios que forman parte del directorio inicial del usuario. El directorio inicial del usuario está definido en la variable user que se le proporciona en el ejemplo usuario (el sexto elemento cuando se usa el separador :). El resultado será similar al de la siguiente figura:



### 3.5 Funciones

Las funciones contienen código que será ejecutado al invocar la función, o al ocurrir un evento asociado a la misma. El código de las funciones puede ser definido en el elemento <head> o <body> de la página HTML. Lo más frecuente es hacerlo en la cabecera, para asegurar que la función está cargada antes de ser invocada desde el <body> o por algún evento.

Si el número de funciones es elevado, o son funciones incluidas en alguna biblioteca de utilidades, lo normal es incluir el fichero externo donde estén definidas:

```
<head>
 <script src="jquery.js"> </script>
```

```
</head>
```

El aspecto de una función será similar a:

```
function nombreFuncion (var1,var2,...,varX)
{ // recuerde de teoría: nombre de la función puede ser opcional
 // código del cuerpo de la función
 return value // recuerde que es opcional
}
```

Los parámetros var1, var2, etc. son los valores que recibe función. Las llaves delimitan la función. La sentencia return especifica el valor que devuelve la función.

Observe en el siguiente ejemplo la función definida en la cabecera (calculaPrecioTotal), y la invocación desde el código ECMAScript incluido en el body. La función recibe dos parámetros para calcular el precio total, y devuelve como resultado dicho precio. Dentro de una función, la palabra reservada var hace que la variable sea local a la función, y no se pueda acceder desde fuera.

#### ejemplo20.html

```
<html>
<head>
<title> Método 1</title>
<script type="text/javascript">
function calculaPrecioTotal(precio, porcentajeImpuestos) {
 var gastosEnvio = 10;
 var precioConImpuestos = (1 + porcentajeImpuestos/100) * precio;
 var precioTotal = precioConImpuestos + gastosEnvio;
 return precioTotal.toFixed(2);
}
</script>
</head>
<body>
<script type="text/javascript">
var precioTotal = calculaPrecioTotal(100,16);
document.write("Precio total de 100 con 16% impuestos y gastos envio: ",
precioTotal);
</script>
</body></html>
```

Además puede apreciar el uso del método toFixed aplicado a la variable precioTotal en la función, para limitar el resultado a dos decimales.

#### TAREAS

- 1º Acceda desde el navegador mediante http al fichero js/ejemplo20.html y observe el resultado.
- 2º Analice el código, y modifique el código en el servidor para que la función reciba un tercer parámetro, los gastos de envío. Deberá modificar también la invocación de la función.

### 3.5.1 Funciones predefinidas: cajas emergentes

Estas funciones de cajas emergentes o” pop boxes” están ya definidas en el lenguaje. Tenemos tres tipos de cajas emergentes:

1. `alert`, de alerta. El usuario simplemente hace clic para continuar. La función no devuelve nada como resultado.
2. `confirm`, de confirmación. El usuario elige una opción entre dos que se le presentan como botones para pulsar. Devuelve `true` (OK) o `false` (Cancel).
3. `prompt`, de solicitud. El usuario introduce un valor que se devuelve como una cadena como resultado de la función, o `null` si el usuario pulsa “Cancelar”.

En el siguiente ejemplo se muestra el uso de las tres posibilidades:

**ejemplo15.html**

```
...<body>
<script type="text/javascript">
var r=confirm("clic en uno de los dos botones");
if (r==true) {
 alert("Pulsó Si");
}
else {
 alert("Pulsó No");
}
document.write("El valor de r es: ", r, "
");
var x;

x=prompt("Nombre ", "");

if (x!=null && x != "")
 document.write("<p>Hola " + x + "</p>");
else
 document.write("<code> Nombre no válido </code>");
</script>
</body></html>
```

## TAREAS

- 1º Acceda desde el navegador mediante http al fichero js/ejemplo15.html
- 2º Modifique el código en el servidor para que la función `prompt` reciba como segundo parámetro el texto “CodigoUser”. Compruebe que al presentar la página en la caja emergente del `prompt` contiene dicho valor, que podemos aceptar o modificar.
- 3º Modifique el código para que distinga entre las opciones a) pulsar “Cancelar”, b) que el usuario introduzca una cadena vacía, c) que el usuario deje el valor por defecto o d) que el usuario introduzca un nuevo nombre.

## 3.6 Eventos

Las aplicaciones web creadas con el lenguaje ECMAScript pueden utilizar el modelo de programación basado en eventos. En este tipo de programación, los scripts esperan que el usuario "haga algo" (que pulse una tecla, que mueva el ratón, que cierre la ventana del navegador). A continuación, el script responde a la acción del usuario, normalmente procesando esa información y generando un resultado.

ECMAScript define numerosos eventos que permiten una interacción completa entre el usuario y las páginas/aplicaciones web. La pulsación de una tecla constituye un evento, así como hacer clic o mover el ratón, seleccionar un elemento de un formulario, redimensionar la ventana del navegador, etc.

ECMAScript permite asignar una función a cada uno de los eventos. De esta forma, cuando se produce cualquier evento, ECMAScript ejecuta su función asociada. Este tipo de funciones se denominan "event handlers" en inglés y suelen traducirse por "manejadores de eventos".

En el modelo básico de eventos, cada elemento o etiqueta HTML define su propia lista de posibles eventos que se le pueden asignar. Un mismo tipo de evento (por ejemplo, pinchar el botón izquierdo del ratón) puede estar definido para varios elementos HTML diferentes. Y un mismo elemento HTML puede tener asociados varios eventos diferentes.

El nombre de cada evento se construye mediante el prefijo `on`, seguido del nombre en inglés de la acción asociada al evento (`click`, `mousemove`, etc). Así, el evento de hacer clic en un elemento con el ratón se denomina `onclick` y el evento asociado a la acción de mover el ratón se denomina `onmousemove`.

La siguiente tabla resume alguno de los eventos más importantes definidos por ECMAScript:

Evento	Descripción
<code>onclick</code>	Pinchar y soltar el ratón
<code>ondblclick</code>	Pinchar dos veces seguidas con el ratón
<code>onfocus</code>	Seleccionar un elemento
<code>onkeypress</code>	Pulsar una tecla
<code>onload</code>	La página se ha cargado completamente
<code>onmousemove</code>	El ratón se está moviendo cuando está sobre el elemento
<code>onmouseover</code>	El ratón "entra" en el elemento (pasa por encima del elemento)
<code>onmouseout</code>	El ratón "sale" del elemento (pasa por encima de otro elemento)
<code>onsubmit</code>	Enviar el formulario
<code>onreset</code>	Inicializar el formulario (borrar todos sus datos)

### 3.6.1 Manejadores de eventos

El método más sencillo y directo de indicar al código ECMAScript que se debe ejecutar cuando se produzca un evento es hacerlo en el elemento HTML, mediante un atributo (de nombre igual al evento) en el que el valor (entre comillas) es el código ECMAScript que se debe ejecutar cuando ocurra el evento. Lo que está entre comillas se convierte en el contenido del cuerpo de una función (sin nombre) que se ejecuta cuando sucede el evento. Es usual que ese código sea la invocación de una función (por

eso hay que incluir los paréntesis después del nombre de la función, para que sea invocada), aunque puede ser un conjunto de instrucciones separadas por punto y coma.

`<p onmouseover="alert('Acabas de pasar el ratón por encima');">` Al pasar por encima de este elemento de tipo párrafo “salta” un mensaje de alerta `</p>`

Hay definidos atributos HTML con el mismo nombre que los eventos que se quieren manejar. En el ejemplo anterior se desea controlar qué ocurre cuando el ratón pasa por encima del elemento, el evento de nombre `onmouseover`. Si se quiere controlar un evento producido en un elemento HTML debe incluir un atributo llamado `onmouseover`. El valor del atributo será las instrucciones ECMAScript que se ejecutan cuando se produce el evento.

Es preferible que, en vez de incluir el código directamente en el atributo, éste se incluya en una función, y desde el atributo simplemente invocar la función.

#### ejemplo30.html

```
...<head>
<script>
function muestraMensaje() {
 alert("Gracias por pinchar.");
}
</script>
</head>
<body>

<input type="button" value="Haz clic para activar evento "
 onclick="muestraMensaje()" />
</body>
</html>
```

#### TAREAS

- 1º Acceda desde el navegador mediante http al fichero `js/ejemplo30.html` y pruebe su comportamiento.
- 2º Pulse F12 y en la pestaña Inspector (Ctrl+Shift+C) despliegue body para que se vea el elemento button. Pulse el icono ev que debe aparecer a la derecha de ese elemento ¿puede ver el evento y la función asociada?
- 2º Modifique el código en el servidor y añada un nuevo párrafo. Modifique el código para que al hacer doble clic sobre ese párrafo se invoque la función `muestraMensaje()`. Compruebe que al hacer clic “simple” sobre ese nuevo párrafo no ocurre nada.

Otra forma de hacer que se ejecute una función cuando se produzca un evento de un elemento HTML sin tener que modificar el propio HTML (como se ha visto hasta ahora con el atributo correspondiente) es usando código ECMAScript: Se busca el elemento mediante ECMAScript, y se le asigna a la propiedad asociada al evento el **nombre de la función** a ejecutar.



## ejemplo31.html

```

...<head>
<script>
function muestraMensaje() {
 alert("Gracias por pinchar.");
}
</script>
</head>
<body>

<input id="b1" type="button" value="Haz clic para activar evento " />
<script>
var x = document.getElementById("b1");
x.onclick = muestraMensaje;
var y=x;
</script>
</body>
</html>

```

Nota: Al hacer las tareas, recuerde las recomendaciones dadas en el apartado 3.2 para que los cambios se reflejen en el navegador. Además, puede que tenga que cerrar las Developer Tools y volverlas a abrir para ver los cambios en la depuración.

## TAREAS

- 1º Acceda desde el navegador mediante http al fichero js/ejemplo31.html y pruebe su comportamiento.
- 2º Use F12 y ponga un punto de ruptura en la asignación de la variable y. Ponga x en “Añadir expresión de vigilancia”. Ponga x.onclick en “Añadir expresión de vigilancia” para ver mejor su valor. ¿Qué valor tiene la propiedad onclick? (por orden alfabético debe estar entre onchange y onclose)
- 3º Modifique el código en el servidor y haga que la asignación sea:
 

```
x.onclick = "muestraMensaje()";
```

 ¿Qué ocurre ahora? ¿Qué valor tiene la propiedad onclick?
- 4º Modifique el código en el servidor y haga que la asignación sea:
 

```
x.onclick = muestraMensaje();
```

 ¿Qué ocurre ahora? ¿Se ejecuta la función antes de hacer click? ¿Qué valor tiene la propiedad onclick?
- 5º Modifique el código en el servidor y defina una nueva función de nombre muestraMensaje2 que muestre por pantalla “Esta es muestraMensaje2”. Añada al final de la función inicial (muestraMensaje) una línea para devolver la nueva función:
 

```
return muestraMensaje2;
```

 Como en el apartado anterior, haga que la asignación sea:
 

```
x.onclick = muestraMensaje();
```

 ¿Qué ocurre ahora? ¿Qué se ejecuta la hacer click? ¿Qué valor tiene la propiedad onclick?

Observe que en este caso original el valor asignado es el nombre de la función, aunque también podría ser la definición de una función sin nombre.

Recuerde que lo que estaba entre comillas asignado al atributo onclick de HTML se convertía en el contenido del cuerpo de una función (sin nombre) que se ejecuta cuando sucede el evento. Así, usar el atributo onclick de HTML de la siguiente forma:

```
<input id="b1" type="button" value="Haz clic para activar evento "
onclick="muestramensaje()" />
```

Es equivalente a usar la propiedad onclick del objeto ECMAScript de la siguiente forma:

```
<script>
var x = document.getElementById("b1");
x.onclick = function() {muestraMensaje()};
</script>
```

Que a su vez es equivalente a:

```
<script>
var x = document.getElementById("b1");
x.onclick = muestraMensaje;
</script>
```

Hay también una tercera forma de añadir código a eventos, mediante el método `addEventListener()`, que se verá en el apartado 3.7.8.

Puede ver un resumen aplicado al evento onclick (que terminará de comprender más adelante) en:

[https://www.w3schools.com/jsref/event\\_onclick.asp](https://www.w3schools.com/jsref/event_onclick.asp)

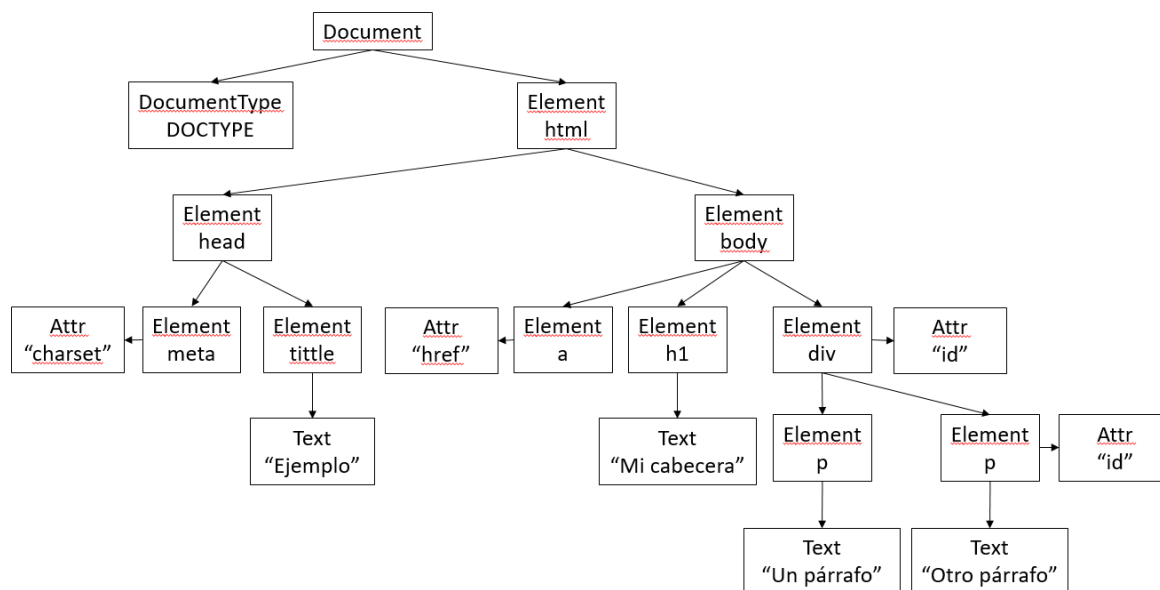
## 3.7 DOM: Document Object Model

### 3.7.1 Introducción a DOM

DOM (Modelo de Objetos de Documento) está normalizado por el W3C (World Wide Web Consortium), y define un estándar para acceso a los documentos:

"El DOM del W3C es una interfaz, de lenguaje neutro, que permite a los programas y scripts acceder y actualizar dinámicamente el contenido, la estructura y el estilo de un documento."

Mediante código ECMAScript que sigue el estándar DOM es posible leer y modificar documentos HTML (y XML). Aunque DOM es genérico, con DOM se modela el documento HTML como un árbol de nodos:



De acuerdo con el estándar, todo en un documento HTML es un nodo (aunque hay 6 tipos básicos de nodos):

tipo	descripción	hijos posibles
Document	Todo el documento es un nodo documento	Element (máximo uno), Comment, DocumentType, ...
Element	Cada elemento HTML es un nodo elemento	Element, Text, Comment, ...
Text	El texto dentro de los elementos HTML son nodos de texto	Sin hijos
Attr	Cada atributo HTML es un nodo de atributo	Text, ...
Comment	Todos los comentarios son nodos comentario	Sin hijos
DocumentType	Asociado a DOCTYPE	Sin hijos
...	...	...

Los nodos en el árbol de nodos tienen una relación jerárquica entre sí, y los términos padre, hijo y hermano se utilizan para describir las relaciones.

- Cada nodo tiene exactamente un padre (*parent*), excepto la raíz (que no tiene padre)
- Un nodo puede tener un cierto número de hijos (*child*), que lo tienen como padre
- Hermanos (*siblings*) son nodos con el mismo padre

En el siguiente ejemplo:

```
<html>
 <head>
 <title>DOM Tutorial</title>
 </head>
 <body>
 <h1>DOM Lesson one</h1>
 <p>Hello world!</p>
 </body>
</html>
```

Las relaciones son:

<html> es el padre de <head> y <body>  
<head> es el primer hijo de <html>  
<body> es el último hijo de <html>

y:

<head> tiene un hijo: <title>  
<title> tiene un hijo (un nodo de texto): "DOM Tutorial"  
<body> tiene dos hijos: <h1> y <p>  
<h1> tiene un hijo: "Lección DOM uno"  
<p> tiene un hijo: "¡Hola, mundo!"  
<h1> y <p> son hermanos

Puede utilizar un visor de nodos en línea en:

<http://software.hixie.ch/utilities/js/live-dom-viewer/>

← → ↻ ⓘ No es seguro | software.hixie.ch/utilities/js/live-dom-viewer/#

## Live DOM Viewer

Markup to test ([permalink](#), [save](#), [upload](#), [download](#), [hide](#)):

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" />
<title> ejemplo 50 eventos</title>
<script type="text/javascript">
function ChangeText() {
 document.getElementById("p1").innerHTML="Texto que sustituye al anterior";
}
</script>
</head>
<body>
<!-- el contenido siguiente puede cambiar -->
<p id="p1">Texto que puede cambiar si hace clic</p>
<input type="button" onclick="ChangeText()" value="Cambia texto" />
</body>
</html>
```

DOM view ([hide](#), [refresh](#)):

```
DOCTYPE: html
HTML
 HEAD
 #text:
 META charset="UTF-8"
 #text:
 TITLE
 #text: ejemplo 50 eventos
 #text:
 SCRIPT type="text/javascript"
 #text: function ChangeText() { document.getElementById("p1").innerHTML="Texto que sustituye al anterior"; }
 #text:
 #text:
 BODY
 #text:
 #comment: el contenido siguiente puede cambiar
 #text:
 P id="p1"
 #text: Texto que puede cambiar si hace clic
 #text:
 INPUT type="button" onclick="ChangeText()" value="Cambia texto"
 #text:
```

### 3.7.2 Nodos, métodos y propiedades de DOM desde ECMAScript

El nodo básico de un documento HTML es document, que es único, representa la página web y a través de él se accede al resto de los nodos. El objeto en ECMAScript para referenciarlo es:

```
document
```

El método más conocido de este nodo es getElementById(), y devuelve el nodo del tipo elemento cuyo atributo id contenga el valor pasado como parámetro. Si no existe devuelve null. Su uso es:

```
document.getElementById()
```

La propiedad más utilizada de un nodo del tipo elemento es innerHTML, y hace referencia al contenido del elemento.

```
innerHTML
```

Observe el siguiente ejemplo. Hay un elemento `<p>` identificado por `id="p1"`, y un botón que invoca a la función `ChangeText`. Ésta obtiene el elemento de `id="p1"` mediante el método `getElementById()` aplicado al objeto `document`, y modifica su valor a través de la propiedad `innerHTML`.

**ejemplo50.html**

```
<html>
<head>
<script type="text/javascript">
function ChangeText() {
 document.getElementById("p1").innerHTML="Texto que sustituye al
anterior";
}
</script>
</head>
<body>

<p id="p1">Texto que puede cambiar si hace clic</p>

<input type="button" onclick="ChangeText()" value="Cambia texto" />
</body></html>
```

**TAREAS**

- 1º Acceda desde el navegador mediante http al fichero `js/ejemplo50.html`.
- 2º Analice el código.
- 3º Modifique el código en el servidor para que, al pulsar el botón, el texto de la función se añada al que ya tenía el párrafo.
- 4º Consulte la norma donde viene especificado el método `getElementById()`:  
<http://www.w3.org/TR/dom/>
- 5º ¿Qué devuelve si no encuentra ningún elemento adecuado? ¿Y si encuentra varios?
- 6º Consulte ese método también en w3schools, dentro de la referencia de ECMAScript (JavaScript), en la lista de métodos y propiedades que implementa el objeto `document` asociado a un documento HTML:  
[http://www.w3schools.com/jsref/dom\\_obj\\_document.asp](http://www.w3schools.com/jsref/dom_obj_document.asp)
- 7º Tarea opcional no evaluable: el método anterior está especificado en la norma mediante el lenguaje WebIDL. Puede consultar este lenguaje en:  
<http://www.w3.org/TR/WebIDL/>

Nota: El navegador lee el fichero completo línea a línea (sea HTML o ECMAScript) en una sola pasada, por lo que el orden de ejecución es fundamental: No puede ejecutarse código ECMAScript que por ejemplo haga alusión a una etiqueta HTML que aún no se haya leído.

**Error por ejecución antes de leer la etiqueta**

```

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" />
</head>
<body>
 <script>
 x = document.getElementById("idx"); // error al ejecutarse
 x.innerHTML="Hola";
 </script>
 <p id="idx">texto</p>
</body></html>

```

**TAREAS**

- 1º Cree un fichero html que contenga el código que da el error y pruébelo accediendo desde el navegador mediante http. Use F12 para ver el valor de x.
- 2º Modifique el fichero en el servidor y ponga el script al final. Pruébalo.
- 3º Vuelva a poner el script al principio, pero haga que las dos líneas de código estén dentro de una función de nombre cambiaTexto. Si no llama a la función ¿se ejecuta?
- 4º Añada al elemento body el atributo onload y enlázelo con la llamada a la función cambiaTexto. Pruébalo ¿Se ejecuta la función? Ponga un punto de ruptura en la primera línea de la función y poniendo la variable x en Añadir expresión de vigilancia. ¿Qué valor tiene x? Avance un paso ¿y ahora qué valor toma x?

**3.7.3 El documento del DOM**

El objeto `document` (del tipo `document`) implementa algunos métodos que no están implementados por los objetos del tipo `element`:

Métodos sólo de <code>document</code>	Descripción
<code>document.getElementById()</code>	devuelve un elemento (sólo uno) por su atributo id
<code>document.getElementsByName()</code>	devuelve elementos (puede ser más de uno, fíjese en la s) por su atributo name
<code>document.createElement()</code>	crea un elemento y lo devuelve
<code>document.createTextNode()</code>	Crea un nodo de texto y lo devuelve
<code>document.write(texto)</code>	escribe en el flujo de salida de HTML

## TAREAS

- 1º Consulte la norma donde viene especificado el método `getElementByName()`.  
Fíjese que este método no viene especificado en la norma de DOM, sino en la de HTML:  
  
<https://www.w3.org/TR/html/>  
<https://www.w3.org/TR/html/dom.html#the-document-object>  
<https://html.spec.whatwg.org/>
- 2º ¿Qué devuelve si no encuentra ningún elemento adecuado? ¿Y si encuentra varios?
- 3º Consulte ese método y otros en w3schools, dentro de la referencia de ECMAScript (JavaScript), en la lista de métodos y propiedades que implementa el objeto `document` asociado a un documento HTML:  
  
[http://www.w3schools.com/jsref/dom\\_obj\\_document.asp](http://www.w3schools.com/jsref/dom_obj_document.asp)
- 4º Consulte ese método y otros en [developer.mozilla.org](https://developer.mozilla.org):  
  
<https://developer.mozilla.org/en-US/docs/Web/API/Document>

El objeto `document` de HTML también implementa unas propiedades que no están implementadas por los objetos del tipo `element`:

Propiedades sólo de <code>document</code>	Descripción
<code>document.anchors</code>	devuelve los elementos <code>&lt;a&gt;</code> que tienen atributo <code>name</code>
<code>document.images</code>	devuelve los elementos <code>&lt;img&gt;</code>
<code>document.forms</code>	devuelve los elementos <code>&lt;form&gt;</code>
<code>document.links</code>	devuelve los elementos <code>&lt;a&gt;</code> y elementos <code>&lt;area&gt;</code> que tienen atributo <code>href</code>
<code>document.domain</code>	devuelve el nombre de dominio del servidor
<code>document.referrer</code>	devuelve la URI del documento que lo ha enlazado
<code>document.URL</code>	devuelve la URL completa del documento
<code>document.body</code>	Referencia al elemento <code>body</code> (para leer o modificar)

## TAREAS

- 1º Consulte la norma donde viene especificado la propiedad `anchors`:  
  
<https://www.w3.org/TR/html/>  
<https://www.w3.org/TR/2014/REC-html5-20141028/obsolete.html#other-elements,-attributes-and-apis>
- 2º Consulte esa propiedad y las otras en w3schools:  
  
[http://www.w3schools.com/jsref/dom\\_obj\\_document.asp](http://www.w3schools.com/jsref/dom_obj_document.asp)



### 3.7.4 Los elementos del DOM

Los objetos del tipo `element` implementan algunos métodos que no son implementados por el objeto `document` (del tipo `document`). Por ejemplo, los relacionados con atributos, ya que el documento HTML no tiene.

Métodos sólo de nodos de tipo <code>element</code>	Descripción
<code>elemento_x.setAttribute( atributo, valor)</code>	añade o cambia el atributo
<code>elemento_x.getAttribute( atributo)</code>	devuelve el valor del atributo especificado
<code>elemento_x.removeChild()</code>	elimina el elemento hijo especificado
<code>elemento_x.appendChild()</code>	añade el elemento especificado al final de los hijos
<code>elemento_x.insertBefore()</code>	inserta el elemento especificado antes de un elemento hijo dado
<code>elemento_x.replaceChild()</code>	reemplaza por uno especificado un elemento hijo dado

#### TAREAS

1º Consulte la definición completa de estos métodos (y otros) en w3schools:

[http://www.w3schools.com/jsref/dom\\_obj\\_all.asp](http://www.w3schools.com/jsref/dom_obj_all.asp)

2º ¿Hay alguna forma de usar el método `insertBefore()` que tenga un funcionamiento similar a `appendChild()`?

También hay algunas propiedades implementadas por los objetos del tipo `element` y no por el objeto `document`:

Propiedades sólo de nodos de tipo <code>element</code>	Descripción
<code>elemento_x.innerHTML=</code>	cambia o devuelve el contenido HTML
<code>elemento_x.textContent=</code>	cambia o devuelve el contenido textual
<code>elemento_x.children</code>	devuelve los elementos hijos (no confundir con <code>childNodes</code> )
<code>elemento_x.nextElementSibling</code>	devuelve el siguiente elemento hermano (no confundir con <code>nextSibling</code> )

<code>elemento_x.previousElementSibling</code>	devuelve el anterior elemento hermano (no confundir con <code>previousSibling</code> )
<code>elemento_x.atributo_y =</code>	cambia o devuelve el atributo
<code>elemento_x.className =</code>	cambia o devuelve el atributo class
<code>elemento_x.style.propiedad_y =</code>	cambia o devuelve el estilo
<code>elemento_x.evento_y =</code>	asocia código de función a un evento sobre un elemento
<code>elemento_x.attributes</code>	devuelve los elementos atributos hijos del elemento
<code>elemento_x.nodeName</code>	devuelve el nombre del nodo (si es un elemento la etiqueta, si es un atributo el nombre)
<code>elemento_x.nodeValue =</code>	cambia o devuelve el valor del nodo (si es un elemento vale null, si es un nodo texto el contenido)

En el siguiente ejemplo se ve el uso de `textContent`:

#### **ejemplo50\_bis.html**

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" >
</html>
<body>

<ul id="lista1">
 ram
 cpu

<p>Pulse el botón para coger el contenido del elemento ul.</p>

<button onclick="cogeLista()">Pruebe</button>

<p id="vacio"></p>

<script>
function cogeLista() {
 var x = document.getElementById("lista1").textContent;
 document.getElementById("vacio").innerHTML = x;
}
</script>

</body>
</html>
```

## TAREAS

- 1º Consulte la diferencia entre la propiedad children y childNodes en w3schools:  
[http://www.w3schools.com/jsref/dom\\_obj\\_all.asp](http://www.w3schools.com/jsref/dom_obj_all.asp)
- 2º Consulte el uso de className en w3schools.
- 3º Para comprobar la diferencia entre innerHTML y textContent modifique ejemplo50\_bis.html y sustituya textContent por innerHTML. Ponga un punto de ruptura en la segunda línea de la función y ponga la variable x en añadir expresión de vigilancia para comprobar su valor.

En el siguiente ejemplo vamos a añadir un nuevo elemento a la página. El mecanismo usual tiene 3 partes:

- crear un nuevo elemento en document
- rellenar el elemento
- añadirlo como hijo de algún elemento de document

## ejemplo51.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" >
<script>
function AnadeElemento () {
// Crear nodo de tipo Element
var parrafo = document.createElement("p");
// Rellenar el nodo de tipo Element
parrafo.innerHTML="Nuevo párrafo tras pulsar botón";
// Añadir el nodo Element como hijo del elemento body
document.body.appendChild(parrafo);
}
</script>
</head>
<body>
<input type="button" onclick="AnadeElemento()" value="Añade párrafo"/>

<p>Párrafo inicial. Al pulsar sobre el botón se añadirán
nuevos párrafos a la página.</p>

</body>
</html>
```

## TAREAS

- 1º Acceda desde el navegador mediante http al fichero js/ejemplo51.html, y pulse el botón que se muestra para ver qué ocurre.
- 2º Abra el fichero con un editor. Analice el código.
- 3º Abra el fichero js/ejemplo51\_bis.html en un navegador, analice el código y pulse el botón que se muestra para ver qué ocurre (uso de `textContent`).
- 4º Abra el fichero js/ejemplo51\_tris.html en un navegador, analice el código y pulse el botón que se muestra para ver qué ocurre (uso de `createTextNode`).
- 5º En el navegador, pulsando F12, observe el documento HTML que se va creando (en los tres casos de ejemplo51).
- 6º Modifique el código para que al pulsar el botón se borre el párrafo que ya existía y se añada otro nuevo.

### 3.7.5 El documento y los elementos del DOM

Hay métodos que se pueden usar tanto en el nodo `document` como en los nodos de tipo `element`.

Métodos	Descripción
<code>document.getElementsByTagName()</code> <code>elemento_x.getElementsByTagName()</code>	devuelve los elementos por su etiqueta (la etiqueta del elemento coincide con el parámetro)
<code>document.getElementsByClassName()</code> <code>elemento_x.getElementsByClassName()</code>	devuelve los elementos por su atributo class
<code>document.querySelector()</code> <code>elemento_x.querySelector()</code>	devuelve el primer elemento que coincide con el selector CSS que se pasa como argumento
<code>document.querySelectorAll()</code> <code>elemento_x.querySelectorAll()</code>	devuelve los elementos que coinciden con el selector CSS que se pasa como argumento
<code>document.addEventListener()</code> <code>elemento_x.addEventListener()</code>	añade un evento y su código asociado (nota: el evento debe especificarse sin el prefijo on. Por ejemplo click en vez de onclick)
<code>document.removeEventListener()</code> <code>elemento_x.removeEventListener()</code>	elimina un evento y su código asociado

Es importante destacar que aunque los métodos que contienen `getElements` (terminado en s) devuelvan en algún caso un solo elemento, lo que devuelven es un array, y en ese caso un array de un único elemento.

## TAREAS

- 1º Cree un fichero html que contenga 2 cabeceras y 4 párrafos. Incluya el atributo id en una de las cabeceras y dos de los párrafos. Añada también un botón que al hacer click sobre él se ejecute la función f1. Incluya el siguiente script:

```
<script>
function f1() {
 var x = document.querySelectorAll("[id]");
 x[0].style.backgroundColor = "red";
 x[1].style.backgroundColor = "yellow";
 x[2].style.backgroundColor = "cyan";
}
</script>
```

- 2º ¿Qué devuelve el método querySelectorAll?

- 3º Modifique la función para recorrer x con un bucle y finalmente mostrar los elementos de x en uno de los párrafos.

### 3.7.6 Atributos en DOM

Mediante DOM, es posible acceder de forma sencilla a todos los atributos HTML de cualquier elemento de la página. Los atributos HTML de los elementos de la página se transforman automáticamente en propiedades de los nodos. Para acceder a su valor, simplemente se indica el nombre del atributo HTML detrás del nombre del nodo.

En el siguiente ejemplo se accede a un elemento con `id="enlace"` y muestra su atributo `href` mediante el nombre del atributo.

#### ejemplo52.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8"/>
<title> ejemplo 52 Atributos</title>
</head>
<body>

<p id="primero" class="elemento_con_enlace"> primer párrafo
 Google
</p>

<a id="enlace" class="elemento_con_enlace"
href="http://www.cnn.com">CNN

<script type="text/javascript">
 var elem_enlace = document.getElementById("enlace");
 alert(elem_enlace.href); // muestra http://www.cnn.com
</script>

</body>
</html>
```

También es posible acceder al valor del atributo y modificarlo con los métodos `setAttribute("atributo", "valor")` y `getAttribute("atributo")`.

## TAREAS

- 1º Acceda desde el navegador mediante http al fichero `js/ejemplo52.html` y pulse el botón que se muestra para ver qué ocurre
- 2º Abra el fichero con un editor. Analice el código.
- 3º Modifique el ejemplo para que muestre el valor del atributo `class` del primer párrafo de la página usando el método `getAttribute()`.

El estilo que se aplica a un elemento puede ser accedido de una forma similar a los atributos, añadiendo entre el nombre del elemento y la propiedad CSS el atributo `style`. El siguiente ejemplo accede a la propiedad `border` y `backgroundColor` de uno de los párrafos de la página. Primero accede al párrafo, localizándolo por `getElementById`:

```
var parrafo = document.getElementById("p");
```

## ejemplo53.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" >
<title> ejemplo 53 DOM</title>
<script>
function CambiaEstilo() {
 var parrafo = document.getElementById("segundo");
 parrafo.style.border="solid thick blue";
 parrafo.style.backgroundColor="yellow";
}
</script>
</head>
<body>
<p id="primero"> primer elemento p
 enlace a Google y US
</p>
<p id="segundo"> segundo elemento p </p>
<input type="button" onclick="CambiaEstilo()" value="Cambia Estilo" />
</body>
</html>
```

## TAREAS

- 1º Acceda desde el navegador mediante http al fichero `js/ejemplo53.html`, y pulse el botón que se muestra para ver qué ocurre.
- 2º Abra el fichero en el servidor. Analice el código. Modifique el ejemplo para que al pulsar el botón el estilo del border sea discontinuo y más fino. Además el margen y relleno de los párrafos debe ser de 10px.

### 3.7.7 Lista de nodos y colección del HTML DOM

Una lista de nodos o colección HTML es lo que devuelven los métodos que pueden devolver más de un elemento. Es parecida a una tabla, pero no es una tabla (no se pueden usar sus métodos, aunque sí la propiedad `length` y acceder mediante índice comenzando en el 0).

Por ejemplo, el método `getElementsByTagName ()` devuelve una lista de nodos. El código siguiente selecciona todos los nodos `<p>` en un documento:

```
var x = document.getElementsByTagName ("p");
```

Los nodos pueden ser accedidos por un número de índice. Para acceder al segundo nodo `<p>` puede escribir:

```
y = x [1];
```

Para recorrer la lista:

```
var i;
for (i = 0; i < x.length; i++) {
 x[i].style.backgroundColor = "red";
}
```

El siguiente ejemplo accede a la propiedad `border` y `backgroundColor` de todos los párrafos de la página. Primero accede a los párrafos, localizándolos por `getElementsByTagName`:

```
var parrafos = document.getElementsByTagName("p");
```

Como `document.getElementsByTagName("p")` devuelve todos los párrafos (una colección), se recorren con un bucle `for` como se observa en el ejemplo:

**ejemplo54.html**

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" >
<title> ejemplo 54 DOM</title>
<script>
function CambiaEstilo() {
 var parrafos = document.getElementsByTagName("p");
 for(var i=0; i<parrafos.length; i++) {
 parrafos[i].style.border="solid thick blue";
 parrafos[i].style.backgroundColor="yellow";
 }
}
</script>
</head>
<body>
<p id="primero"> primer elemento p
 enlace a Google y
 US
</p>
<p id="segundo"> segundo elemento p </p>
<input type="button" onclick="CambiaEstilo()" value="Cambia Estilo" />
</body>
</html>
```

En el siguiente ejemplo se observa el uso de `childNodes`, `nodeName` y `nodeValue`, pero contiene un error.

#### ejemploNode.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" >
</head>
<body>

<p>Pulse el botón para mostrar nodeName y nodeValue de distintas
partes.</p>

<button onclick="muestraProp()">Pruebe a pulsar</button>

<p id="vacio"></p>

<script>
function muestraProp() {
 var x = document.getElementById("miImagen");
 var y = document.getElementsByTagName("button")[0];
 var z = y.childNodes;
 var txt = "y.nodeName=" + y.nodeName + "
";
 txt = txt+ "y.nodeValue=" + y.nodeValue + "
";
 txt = txt+ "z.nodeName=" + z.nodeName + "
";
 txt = txt+ "z.nodeValue=" + z.nodeValue + "
";
 txt = txt+ "x.nodeName=" + x.nodeName + "
";
 txt = txt+ "x.nodeValue=" + x.nodeValue + "
";
 var i;
 for (i = 0; i < x.attributes.length; i++) {
 txt = txt + x.attributes[i].nodeName + " = " +
x.attributes[i].nodeValue + "
";
 }
 document.getElementById("vacio").innerHTML = txt;
}
</script>

</body>
</html>
```

#### TAREAS

- 1º Acceda desde el navegador mediante http al fichero `js/emplonode.html`, y pulse el botón que se muestra para ver qué ocurre. ¿Tiene la variable `z` el valor adecuado?
- 2º Ponga un punto de ruptura en la última línea del script. Ponga las variables `x`, `z` e `y` en Añadir expresión de vigilancia. ¿Qué valor tiene `y`? ¿Es una tabla o similar? ¿Qué valor tiene `z`? ¿Es una tabla o similar? Modifique la asignación de `z` para que se muestre el nombre del nodo y su valor.



### 3.7.8 Eventos en el DOM HTML

Ya se ha indicado una forma simple de cómo asociar eventos a una etiqueta: añadiendo un atributo, por ejemplo `onclick`.

También existe un método que permite asociar eventos de una forma más general:

```
elemento_x.addEventListener(event, function);
```

Por ejemplo:

```
document.getElementById("miBoton").addEventListener("click", miFuncion);

function miFuncion(){ alert("Hola World!"); }
```

Nótese que no se antepone el prefijo “on” al evento.

La ventaja de usar este método es que puede asociar múltiples eventos a un elemento, sin sobreescribirlos.

Si se desea eliminar eventos, existe el método:

```
elemento_x.removeEventListener("mousemove", miFuncion);
```

Nota: hay un tercer parámetro opcional del método `addEventListener` para decidir el orden en que se propagan los eventos anidados. Para más información puede consultar:

[http://www.w3schools.com/js/js\\_htmldom\\_eventlistener.asp](http://www.w3schools.com/js/js_htmldom_eventlistener.asp)

En el siguiente ejemplo se observa cómo a un mismo elemento se le pueden asociar varios eventos con el método `addEventListener()`. Además, se pueden asociar varias funciones al mismo evento.

**ejemploEvent.html**

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" >
</head>
<body>
<p>Este ejemplo usa el método addEventListener() para añadir múltiples
eventos al mismo botón.</p>

<button id="Boton1">Pruebe</button>
<button id="Boton2" onclick="f5()">Pruebe a quitar el alert</button>

<p id="vacio"></p>

<script>
var x = document.getElementById("Boton1");
x.addEventListener("mouseover", f1);
x.addEventListener("click", f2);
x.addEventListener("click", f3);
x.addEventListener("mouseout", f4);

function f1() {
 document.getElementById("vacio").innerHTML += "Evento mouseover
";
}

function f2() {
 document.getElementById("vacio").innerHTML += "Evento click
";
}

function f3() {
 alert("Evento click: Ha pulsado el botón");
}
function f4() {
 document.getElementById("vacio").innerHTML += "Evento mouseout
";
}
function f5() {
 // código para quitar f3 del evento click en botón 1
}
</script>

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

## TAREAS

- 1º Acceda desde el navegador mediante http al fichero js/ejemploEvent.html, y mueva el ratón sobre el botón 1. Pulse el botón para ver qué ocurre.
- 2º Pulse F12 y ponga un punto de ruptura en la primera línea del script. Vaya avanzando paso a paso y observe la pestaña Eventos en la ventana de la derecha.
- 3º Quite el punto de ruptura. Modifique el código de f5() para que haga su función. Recargue la página y pruébelo.
- 4º Pulse F12 y en la pestaña Inspector (Ctrl+Shift+C) despliegue body para que se vean los elementos button. Pulse el icono ev que debe aparecer a la derecha de esos elementos ¿puede ver los eventos y las funciones asociadas?

### 3.7.9 Navegación a través de los nodos del HTML DOM

Un error común en el procesamiento DOM es esperar que un nodo elemento contenga texto. El texto está contenido en un nodo de tipo texto (no es de tipo element).

En el siguiente ejemplo, el nodo de elemento <title> no contiene texto. Contiene un nodo de texto con el valor "DOM Tutorial".

```
<title> DOM Tutorial </title>
```

El valor del nodo de texto se puede acceder por la propiedad innerHTML del nodo, la propiedad textContent, o el nodeValue del primer nodo hijo (el nodo de texto).

El siguiente ejemplo recoge el valor del nodo de un <h1> y lo copia en un elemento <p>:

**ejemplo56.html**

```
<!DOCTYPE html>
<html>
<body>
<h1 id="intro1">Mi primera cabecera</h1>
<h2 id="intro2">Mi segunda cabecera</h2>
<p id="demo1">Hello World!</p>
<p id="demo2">Hello World!</p>
<script>

var miTextol = document.getElementById("intro1").childNodes[0].nodeValue;

document.getElementById("demo1").innerHTML= miTextol;

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

## TAREAS

- 1º Compruebe que en el ejemplo anterior se modifica el contenido del elemento identificado por demo1.
- 2º Añada dos líneas al ejemplo para modificar el contenido del elemento identificado por demo2 (con el contenido del elemento identificado por intro2) pero esta vez use `innerHTML` para crear la variable y `childNodes[0].nodeValue` para modificar el contenido.

Puede utilizar las siguientes propiedades del nodo para navegar entre los nodos (de cualquier tipo) con JavaScript (`childNodes` devuelve los nodos hijos, sean de tipo elemento, texto o comentario, pero NO atributos):

```
parentNode
childNodes
nextSibling
previousSibling
```

En HTML, normalmente sólo interesan los nodos de tipo element, y para ello se usan las propiedades ya vistas en apartados anteriores (`children` devuelve los nodos hijos que sean elementos).

```
children
nextElementSibling
previousElementSibling
```

En el siguiente ejemplo puede observar cómo borrar e insertar algunos nodos que no son hijo directo de body (usando `parentNode`):

**ejemplo57.html**

```
<script>
 "use strict";
 alert("Borramos 1er párrafo del artículo");
 var p = document.querySelector("article p");
 p.parentNode.removeChild(p);

 alert("Añadimos li")
 var li = document.createElement("li");
 li.textContent = "nuevo li añadido desde JS";
 var lista = document.querySelector("article ul");
 lista.appendChild(li);

 alert("Añadimos li el primero")
 var li = document.createElement("li");
 li.textContent = "otro nuevo li añadido desde JS";
 var lista = document.querySelector("article ul");
 // no es lo mismo firstElementChild que firstChild
 // similar a diferencia entre childNodes y Children
 console.log(lista, lista.firstElementChild, li);
 lista.insertBefore(li, lista.firstElementChild);
</script>
```

## TAREAS

- 1º Compruebe el funcionamiento del ejemplo anterior.
- 2º Añada código para invertir los elementos de esa lista.
- 3º Añada el código de una función para invertir los elementos de una lista pasada como parámetro y úsela.

### 3.8 Validación de Formularios

Una de las utilidades de ECMAScript es la comprobación de los campos de un formulario antes de ser enviado al servidor que los procesa. El modelo HTML DOM facilita el acceso a los formularios de la página HTML mediante el objeto `forms` que depende de `document`. También es posible acceder a los campos del formulario a través de los identificadores de las entradas, con el método `document.getElementById("valorid")`.

Por otra parte, para validar las entradas del formulario, debe modificar el elemento `form` para añadirle un atributo que asocie la pulsación del botón de envío con una función que compruebe si los campos del formulario son válidos antes de continuar con el envío del formulario. Esta función devolverá `true` si todo es correcto, o `false` si alguno de los campos no lo es. El atributo a utilizar es `onsubmit`, y su valor será `"return nombre_función()"`. El formulario tendrá un aspecto similar a:

```
<form action="" method="post" id="formulario"
 onsubmit="return validacion()">
Nombre: <input type="text" id="nombre">

Edad: <input type="text" id="edad">

<input type="submit" value="Submit">
</form>
```

La función que valida los datos del formulario irá accediendo a los campos a comprobar de uno en uno, y si hay algún error, devuelve el valor `false`. De esta forma, en cuanto un campo no sea correcto, la función que comprueba devuelve `false`, `onsubmit` devuelve `false`, y el formulario no se envía.

En este primer ejemplo sólo se comprueba que el campo edad sea numérico. Se añade un mensaje de aviso en caso de que no lo sea, el formulario no se envía, y los datos introducidos en los campos permanecen.

## ejemplo60.html

```
<html>
<head>
<title> ejemplo eventos</title>
<script type="text/javascript">
function validacion() {
 valor = document.getElementById("edad").value;
 if (isNaN(valor)) {
 alert('[ERROR] El campo edad debe ser numérico..');
 return false;
 }
 // Si script llega a este punto, todas las condiciones
 // se han cumplido, por lo que se devuelve el valor true
 return true;
}
</script>
</head>
<body>
<form action="" method="post" id="formulario" name="formulario"
onsubmit="return validacion()"> ...
Nombre: <input type="text" id="nombre">

Edad: <input type="text" id="edad">

<input type="submit" value="Submit">...
</form>
</body></html>
```

## TAREAS

- 1º Acceda desde el navegador mediante http al fichero js/ejemplo60.html. Muestre el resultado al enviar el formulario, introduciendo una edad incorrecta (por ejemplo una letra, un número con una letra, etc.). Busque información sobre la función `isNaN()`.
- 2º Modifique el ejemplo para que se compruebe que la edad es mayor que cero.
- 3º Modifique el ejemplo para que se compruebe que el campo nombre no se envíe vacío.
- 4º Use “required” y “number” de HTML5 para realizar las tareas anteriores .

Nota: Actualmente hay un atributo para HTML para evitar que se envíe un campo vacío: `required`, pero se sigue usando una función de validación para comprobar valores que debe introducir una persona (y no un robot). Observe su uso en ejemplo61.html.

## ejemplo61.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" />
<title> ejemplo valida form 2</title>

<script>
function validaForm() {
 var x = document.forms["miForm"]["fc"].value;
 if (x != "azul") {
 alert("Clave incorrecta");
 return false;
 }
}
</script>
</head>
<body>

<form name="miForm" action=""
onsubmit="return validaForm()" method="post">
<label for="c1">Nombre usuario: </label>
<input type="text" id="c1" name="fn" required="required">
<label for="c2">Clave (color del cielo): </label>
<input type="text" id="c2" name="fc">
<input type="submit" value="Enviar">
</form>

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

## TAREAS

- 1º Acceda desde el navegador mediante http al fichero js/ejemplo61.html. Muestre el resultado al enviar el formulario, dejando el primer campo vacío o introduciendo una clave incorrecta.). Busque información sobre la etiqueta label y el atributo for.
- 2º Modifique el ejemplo para que tenga un campo numérico y con un valor mínimo y máximo, usando los atributos min y max

Nota sobre el atributo "name":

Solo se considera válido en DOM para algunos objetos específicos. En esta asignatura se usarán en los relacionados con los formularios:

<button>, <form>, <input>, <select> y <textarea>

Para estos objetos puede establecer, obtener y cambiar el atributo 'name' utilizando object.name, pero para cualquier otro objeto DOM el atributo 'name' es un atributo personalizado y debe crearse utilizando SetAttribute () o agregándolo a la declaración HTML. Una vez creado, puede acceder a él utilizando setAttribute () y getAttribute () o puede hacer referencia a su valor directamente utilizando object.attributes.name.value

Puede consultarlo en:

<https://developer.mozilla.org/en-US/docs/Web/API/Element/name>

### 3.9 Eventos de temporización

ECMAScript permite ejecutar código después de un intervalo de tiempo especificado. Los métodos que se usan para ello son `setTimeout()` y `setInterval()`.

`setTimeout("función",milisegundos)` ejecuta la función después de esperar los milisegundos indicados. Por ejemplo,

```
setTimeout(function(){alert("He esperado 3 segundos")},3000);
```

espera 3 segundos para mostrar la ventana de alerta. Esta función puede ser parte del código asociado a otro evento, por ejemplo la carga de la página, o pulsar un botón, como en este ejemplo:

#### **ejemploTemp01.html**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" >
<script type="text/javascript">
function avisa(seg) {
 setTimeout(function(){alert("Ha esperado " + seg + "
segundos.")},seg*1000);
}
</script>
</head>
<body>
<p>Haga clic sobre el botón y espere...</p>
<button onclick="avisa(2)">Prueba!</button>
</body></html>
```

#### **TAREAS**

- 1º Acceda desde el navegador mediante http al fichero `js/ejemploTemp01.html`.
- 2º Modifique el fichero en el servidor para añadir un contador del número de veces que se llama a `avisa` y que se muestre mientras se está esperando. ¿Es asíncrono `setTimeout`?

El método `setInterval("función",milisegundos)` es similar a la anterior, pero la función se ejecuta después de esperar los milisegundos indicados de forma periódica. Si modificamos el ejemplo anterior, `ejemplote01.html` y cambiamos el nombre de la función `setTimeout` por `setInterval`, el aviso de alerta aparecerá de forma periódica.

#### **TAREAS**

- 1º Modifique el fichero `ejemploTemp01.html` cambiando la función `setTimeout` por `setInterval`.
- 2º Acceda desde el navegador mediante http al fichero y pruebe su comportamiento.



Si desea detener la repetición periódica de la función deberá usar la función `clearInterval()`. Esta función recibe como parámetro el valor que devuelve la función `setInterval()`.

**ejemploTemp03.html**

```
...
<script type="text/javascript">
function avisa(seg) {
 timer = setInterval(function(){alert("Ha esperado " + seg + "
segundos.")},seg*1000);
}
function parar() {
 clearInterval(timer);
}
</script>
</head>
<body>

<p>Haga clic sobre el botón y espere...</p>
<button onclick="avisa(2)">;Arranca!</button>

<p>Si está desesperado...</p>
<button onclick="parar()">;Para!</button>

</body> </html>
```

**TAREAS**

- 1º Acceda desde el navegador mediante http al fichero `js/ejemploTemp03.html` y compruebe su funcionamiento.
- 2º Modifique el fichero `js/ejemplote03.html` para que un único botón inicie o detenga la aparición de las alertas al hacer clic sobre él de forma consecutiva.

Veamos un ejemplo de uso de `setInterval` para presentar un reloj en pantalla. En este caso se inicia el temporizador al cargar la página (`body onload`), y cada 500 milisegundos se llama a la función `muestrahora`, que la muestra en el elemento con `id="txt"`:

## ejemploTemp05.html

```
...
<script type="text/javascript">
function checkTime(i){ // add a zero in front of numbers<10
 if (i<10)
 i="0" + i;
 return i;
}
function muestrahora() {
 var today=new Date();
 var h=today.getHours();
 var m=checkTime(today.getMinutes());
 var s=checkTime(today.getSeconds());
 document.getElementById('txt').innerHTML = h + ":"+m+":"+s;
}
function arrancatimer() {
 muestrahora();
 setInterval(muestrahora,500);
}
</script>
</head>
<body onload="arrancatimer()">
<p> Hora local:

...

```

## TAREAS

- 1º Acceda desde el navegador mediante http al fichero js/ejemploTemp05.html y compruebe su funcionamiento.
- 2º Modifique el fichero anterior para que incluya un botón que inicie o detenga el reloj.

### 3.10 Etiqueta noscript

HTML define la etiqueta `<noscript>` para incluir un mensaje que los navegadores muestran cuando la posibilidad de ejecutar scripts (por ejemplo ECMAScript) se encuentra bloqueada o deshabilitada (o no fueran capaces).

Esto es tan sencillo como incluir la etiqueta `<noscript>` dentro del `<body>`.

## enoscript.html

```
...
<body>
<h2>Bienvenido a Mi Sitio</h2>
<script type="text/javascript">
document.write("<p> Texto que se muestras si ECMAScript está habilitado
</p>");
</script>
<noscript>
 <p style="color:red; background: silver;">La página que estás viendo
requiere para su funcionamiento el uso de ECMAScript.
Si lo has deshabilitado intencionadamente, por favor vuelve a
activarlo.</p>
</noscript>
</body></html>
```

## TAREAS

- 1º Acceda desde el navegador mediante http al fichero js/enoscript.html y compruebe su funcionamiento.
- 2º Modifique la configuración del navegador para deshabilitar ECMAScript. Una forma de hacerlo es en una pestaña del navegador escriba `about:config`, después en el filtro Buscar escriba `javascript` y finalmente modifique `javascript.enabled`. El aspecto de la página antes y después de desactivar ECMAScript se muestra en las siguientes figuras. No olvide posteriormente activar ECMAScript para continuar la práctica.





## 4 AJAX

### 4.1 AJAX (Asynchronous JavaScript And XML)

AJAX, acrónimo de JavaScript asíncrono y XML, es una técnica de desarrollo web para crear aplicaciones interactivas. Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios, mientras se mantiene la comunicación **asíncrona** con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas completamente, lo que significa aumentar la interactividad, velocidad y usabilidad en las aplicaciones.

AJAX es una técnica válida para múltiples plataformas y utilizable en muchos sistemas operativos y navegadores, dado que está basado en estándares abiertos como ECMAScript y Document Object Model (DOM)

AJAX no es una tecnología en sí misma: se trata de varias tecnologías que se unen para colaborar. Las tecnologías que pueden formar parte de una “aplicación” AJAX son:

- HTML y CSS.
- DOM, para la interacción y manipulación dinámica de la presentación.
- XMLHttpRequest, objeto necesario para el intercambio asíncrono de información.
- ECMAScript, para unir todas las demás tecnologías.
- Un formato de datos para el intercambio cliente servidor, que puede ser XML, HTML, JSON, etc.

Las aplicaciones construidas con AJAX eliminan la recarga constante de páginas mediante la creación de un elemento intermedio entre el usuario y el servidor. La nueva capa intermedia de AJAX mejora la respuesta de la aplicación, ya que el usuario nunca se encuentra con una ventana del navegador vacía esperando la respuesta del servidor.

Las peticiones HTTP al servidor ahora se inician por ECMAScript, que se realizan de forma asíncrona mediante AJAX.

La única “novedad” en el uso de AJAX es cómo hacer las invocaciones “asíncronas” al servidor, manteniendo el resto de la página invariable. Para ello se usa el objeto `XmlHttpRequest`, con sus propiedades y métodos.

## 4.2 XmlHttpRequest y primera aplicación AJAX

La aplicación AJAX más sencilla consiste en una adaptación del clásico “Hola Mundo”. En este caso, una aplicación ECMAScript descarga un archivo del servidor y muestra su contenido sin necesidad de recargar la página.

**ejemplo160.html**

```
...
<script>
"use strict";
function loadTextDoc(url) {
 var xmlhttp;
 var txt;
 xmlhttp = new XMLHttpRequest();
 xmlhttp.onreadystatechange = function() {
 if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
 txt = xmlhttp.responseText; // alert(txt);
 document.getElementById('Info').innerHTML = txt;
 }
 }
 xmlhttp.open("GET", url, true);
 xmlhttp.send();
}
</script>
...
```

Por un lado tenemos un bloque div con `id="Info"` en la página html. En este elemento tenemos un botón que al ser pulsado invoca una función, pasándole como parámetro el nombre del fichero a descargar desde el servidor:

```
<div id="Info" onclick="loadTextDoc('HolaMundo.txt')" >
 <button> Carga contenido de Holamundo.txt en este div </button>
</div>
```

El contenido de la función es:

```
function loadTextDoc(url) {
 var xmlhttp; // objeto para petición asíncrona
 var txt;
 xmlhttp = new XMLHttpRequest();// code for IE7+, Firefox, Chrome,...
```

En este primer bloque de la función se crea el objeto que se usará posteriormente para la petición asíncrona. Por diferencias con versiones previas de Internet Explorer se puede encontrar código en la que aparecen dos formas de iniciar dicho objeto. Actualmente ya no es necesario.

A continuación se hace uso del método `onreadystatechange`, responsable de manejar los eventos que se producen. Se invoca cada vez que se produce un cambio en el estado de la petición HTTP. Normalmente es una referencia a una función ECMAScript. En nuestro ejemplo es una función que en el caso de que cambie el estado de la petición lo primero que hace es comprobarlo mediante las propiedades `status` y `readyState`:

- `readyState`: Valor numérico (entero) que almacena el estado de la petición. El valor 4 significa completada (DONE). Puede consultarlo en:

<https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/readyState>

- `status`: El código de estado HTTP devuelto por el servidor (200 para una respuesta correcta, 404 para “No encontrado”, 500 para un error de servidor, etc.).

```
xmlhttp.onreadystatechange = function() {
 if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
 txt = xmlhttp.responseText; // alert(txt);
 document.getElementById('Info').innerHTML = txt;
 }
}
```

Si estas peticiones son correctas, la función invocada, al existir cambio de estado, almacena la respuesta en formato de texto (`xmlhttp.responseText`) en la variable `txt`, que se utiliza para modificar el contenido del elemento `id="Info"` usando los métodos DOM ya conocidos.

Las dos últimas sentencias de la función `loadXMLdoc` son las que hacen la petición al servidor.

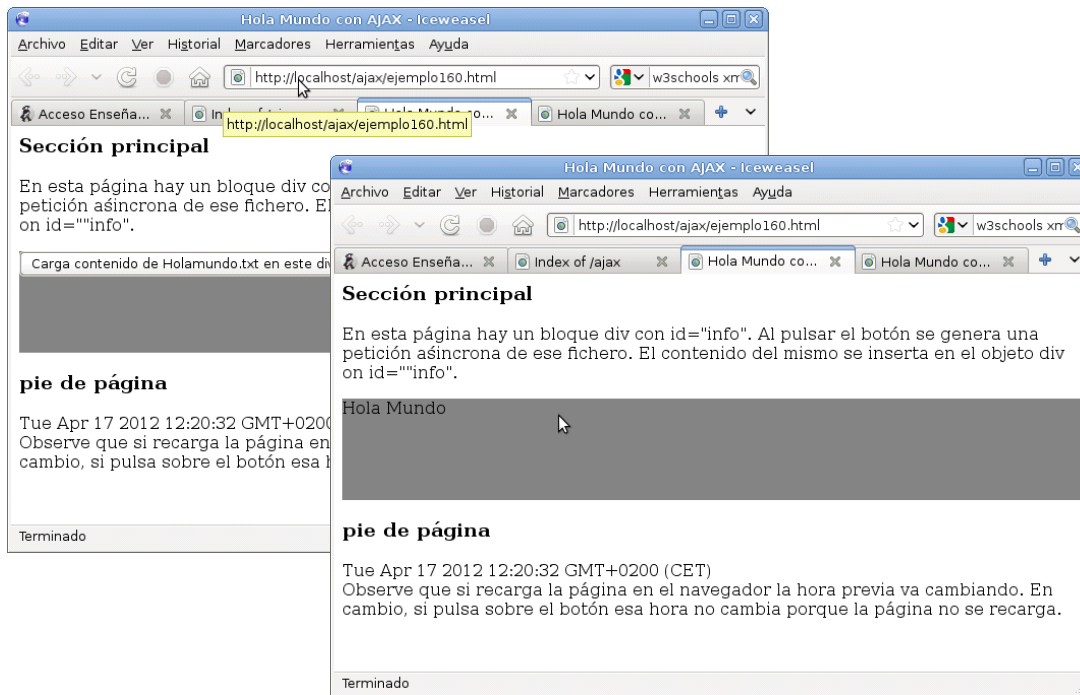
```
xmlhttp.open("GET", url, true);
xmlhttp.send();
```

Las instrucciones anteriores realizan el tipo de petición más sencillo que se puede enviar al servidor. Se trata de una petición de tipo GET simple, que no envía ningún parámetro al servidor. La petición HTTP se crea mediante el método `open()`, en el que se incluye el tipo de petición (GET), la URL solicitada, que es un parámetro de la función, y un tercer parámetro que vale `true` y que indica que la petición es asíncrona.

Una vez creada la petición HTTP, se envía al servidor mediante el método `send()`.

Fíjese que antes de enviar la petición se ha tenido que dejar preparada la función que se va a ejecutar cuando se reciba la respuesta. Esta función es conocida como función de callback y es un paradigma muy utilizado en la programación asíncrona.

Observe el resultado de mostrar la página, y pulsar el botón:



## TAREAS

- 1º Acceda desde el navegador mediante http al fichero `ajax/ejemplo160.html`.
- 2º Analice el código, tanto la parte html como la parte ECMAScript, uso de XMLHttpRequest, propiedades y métodos, etc.
- 3º Observe la fecha que aparece en la parte inferior. Recargue la página (ctrl-F5) para ver cómo cambia dicha fecha. Finalmente pulse el botón para cargar el fichero de forma asíncrona, y ver el resultado, que será similar al de la figura. Observe cómo no cambia la fecha de la parte inferior.
- 4º Use F12 para ver el intercambio de mensajes (pestaña Red). Cuando recarga la página ¿cuántos octetos contiene el cuerpo de la respuesta? Busque la cabecera Content-Length. Cuando pulsa el botón ¿qué contiene la petición? ¿cuántos octetos contiene el cuerpo de la respuesta? Observe el cuerpo de la respuesta en la pestaña Respuesta de la ventana de la derecha.

Si hace algo similar y ahora la respuesta es el contenido de un fichero xml, será necesario recorrer el árbol xml que forma la respuesta (`xmlhttp.responseXML`) con funciones ECMAScript, como por ejemplo `getElementsByTagName()`;

## ejemplo162.html

```
...
x=xmlhttp.responseXML.getElementsByTagName("CD");
for (i=0;i<x.length;i++) {
 txt=txt + "<tr>";
 xx=x[i].getElementsByTagName("TITLE");
 txt=txt + "<td>" + xx[0].firstChild.nodeValue + "</td>";

 xx=x[i].getElementsByTagName("ARTIST");
 ...
}
```

## TAREAS

- 1º Acceda desde el navegador mediante http al fichero ajax/ejemplo162.html.
- 2º Analice el código, tanto parte html como parte ECMAScript, uso de XMLHttpRequest, sus propiedades y métodos, etc.
- 3º Observe las funciones que se usan para localizar elementos y recorrer el árbol xml.
- 4º Use F12 para ver el intercambio de mensajes. Cuando pulsa el botón ¿qué contiene la petición? ¿cuántos octetos contiene el cuerpo de la respuesta? Observe el cuerpo de la respuesta en la pestaña Respuesta de la ventana de la derecha. Sitúe el ratón sobre el icono js que aparece a la izquierda de xhr en el segundo mensaje ¿a qué línea del fichero hace referencia LoadXMLDoc?
- 5º Edite el fichero ejemplo162.html en el servidor y ponga el botón que llama a LoadXMLDoc en otro div que no se sobrescriba al descargar los Cds. Pruébalo. Modifique el fichero XML quitando la información de un CD y compruebe que ya no aparece (puede que tenga que borrar la caché del navegador en menú > preferencias > privacidad > limpiar historial).
- 6º Modifique la creación de la tabla para que aparezca una tercera columna que sea el año (YEAR).

### 4.3 JSON

JSON (JavaScript Object Notation) es un formato de intercambio de datos de texto, independiente del lenguaje, auto-descriptivo, legible, jerárquico y fácil de comprender. Usa la sintaxis de ECMAScript para definir objetos, y existen analizadores y bibliotecas para JSON para la mayoría de los lenguajes de programación. Los programas ECMAScript pueden usar la función JSON.parse() para convertir los datos JSON en objetos ECMAScript nativos. La función eval() está desaconsejada por cuestiones de seguridad:

[https://www.w3schools.com/js/js\\_json\\_eval.asp](https://www.w3schools.com/js/js_json_eval.asp)

Al igual que XML, JSON es texto, y como veremos más tarde, se integra bien con Ajax. Por el contrario, no tiene etiquetas, es más corto, en general más sencillo de leer y escribir, y puede ser “parseado”. Para aplicaciones Ajax, en muchos casos es más rápido y sencillo que XML.

La sintaxis de JSON es sencilla:



- Los datos se especifican en pares nombre: valor.
- El nombre va entrecomillado.
- El valor puede ser: número, cadena (delimitada por comillas), valor lógico (true o false), null, un objeto o una tabla.
- Los datos se separan por comas.
- Las llaves delimitan objetos.
- Los corchetes delimitan tablas.

Un ejemplo de objeto JSON sería:

```
{ "nombre": "John" , "apellido": "Doe", "edad": 32 }
```

Y un ejemplo de tabla JSON como valor dentro de un objeto sería:

```
{ "empleados": [
 { "nombre": "John" , "apellido": "Doe" , "edad": 32 },
 { "nombre": "Anna" , "apellido": "Smith" , "edad": 26 } ,
 { "nombre": "Peter" , "apellido": "Jones" , "edad": 41 }
]
```

La sintaxis de ECMAScript es igual:

```
var obj_es={ "empleados" : [
 { "nombre": "John" , "apellido": "Doe" , "edad": 32 },
 { "nombre": "Anna" , "apellido": "Smith" , "edad": 26 } ,
 { "nombre": "Peter" , "apellido": "Jones" , "edad": 41 }
]};
```

Una vez en el objeto, los datos pueden ser accedidos o modificados:

```
edad= obj_es.empleados[1].edad;
obj_es.empleados[0].nombre= "Luis";
```

En el siguiente ejemplo se observa cómo teniendo los datos en una cadena en formato JSON, con el método `JSON.parse()` se obtiene un objeto ECMAScript.

## ejson03.html

```
...
<body>
<h2>Crear objeto desde cadena JSON </h2>
<p>
Nombre:

Apellido:

Edad:

</p>
<script>
var txt = ' {"empleados":[' +
'{"nombre":"John","apellido":"Doe", "edad": 32 },' +
'{"nombre":"Anna","apellido":"Smith", "edad": 27 },' +
'{"nombre":"Peter","apellido":"Jones", "edad": 41 }]' ;

alert("parse de la cadena JSON y usamos el objeto creado");
obj = JSON.parse(txt);

document.getElementById("nombre").innerHTML=obj.empleados[1].nombre;
document.getElementById("apellido").innerHTML=obj.empleados[1].apellido;
document.getElementById("edad").innerHTML=obj.empleados[1].edad;

</script>

</body>
</html>
```

## TAREAS

- 1º Acceda desde el navegador mediante http al fichero js/ejson03.html.
- 2º Modifique el fichero para que se muestren por pantalla los datos de todos los empleados. Como es obvio, debe usar un bucle.

Aunque en este ejemplo los datos en formato JSON están en una cadena de texto, la utilidad fundamental es cuando los datos en formato JSON vienen en una respuesta obtenida utilizando AJAX. En el apartado 6.4 verá un ejemplo.

## 5 Aplicación comentarios

Se presenta la aplicación “Comentarios” como código de ejemplo, no como ejercicio.

La aplicación utiliza el objeto implícito `application` para ir guardando una serie de mensajes. Los clientes usan Ajax para mostrar instantáneamente los mensajes nuevos que se van añadiendo, utilizando sondeo cada 1 segundo.

## TAREAS

1. Acceda desde el navegador mediante `http` al fichero `jsp/appcomentarios/comentarios.html` y compruebe el funcionamiento.
2. Abra con un editor de texto el fichero `comentarios.js` y compruebe que contiene las funciones (más adelante se dan algunas indicaciones):

```
function utf8_to_b64(str)
function b64_to_utf8(str)
function ajax(peticion, funcionRespuesta, parametro)
function ajaxPost(peticion, datosPost, funcionRespuesta, parametro)
function elem(id)
function enviarComentario()
function resultadoEnvio(xmlHttp, parametro)
function iniciarSondeo()
function sondeo()
function pedirTotal()
function resultadoTotal(xmlHttp, parametro)
function recibirComentarios()
```

Notas sobre Java:

El ejemplo hace uso de los siguientes ficheros Java (y sus correspondientes `.class`):

```
/home/dit/web/WEB-INF/src/fast/AlmacenComentarios.java
/home/dit/web/WEB-INF/src/fast/Comentario.java
```

## 5.1 Aclaraciones sobre las funciones

Las funciones incluyen comentarios que explican su funcionamiento básico.

Las funciones `utf8_to_b64` y viceversa se usan para trabajar en codificación base64.

Las funciones `ajax` y `ajaxPost` realizan las peticiones AJAX GET y POST respectivamente, incluyendo la llamada a la función a ejecutar con la respuesta.

La función `enviarComentario` toma los datos del formulario y los envía con una petición AJAX POST, y el resultado se procesará con la función `resultadoEnvio`

La función `resultadoEnvio` recibe la respuesta y si es null indica envío incorrecto

La función `iniciarSondeo` se pone en marcha al cargarse la página y asocia `enviarComentario` con el evento de pulsar el botón de enviar y también con pulsar `intro` en el campo de texto. También pone en marcha el temporizador periódico para el sondeo.

La función `sondeo` llama a `pedirTotal` para saber cuántos comentarios hay en el servidor y a `recibirComentarios`, por si en el servidor hay más de los que tiene el cliente. Utiliza la variable `sondeoEnCurso` para que no se acumulen peticiones.

La función `pedirTotal` envía una petición AJAX GET, y el resultado se procesará con la función `resultadoTotal`.

La función `resultadoTotal` analiza la respuesta de texto y la convierte a un número en base 10.

La función `recibirComentarios` (llamada por sondeo) comprueba si el servidor tiene más comentarios que en el cliente y en ese caso envía una petición AJAX POST solicitando el siguiente comentario (sólo uno), y el resultado se procesará con la función `resultadoRecibir`.

La función `resultadoRecibir` analiza la respuesta XML, actualiza el contador de comentarios del cliente, crea un nuevo elemento con los datos obtenidos de la respuesta XML y lo inserta al principio del elemento identificado por `comentarios`. También llama otra vez a `recibirComentarios` por si quedan más comentarios por solicitar.

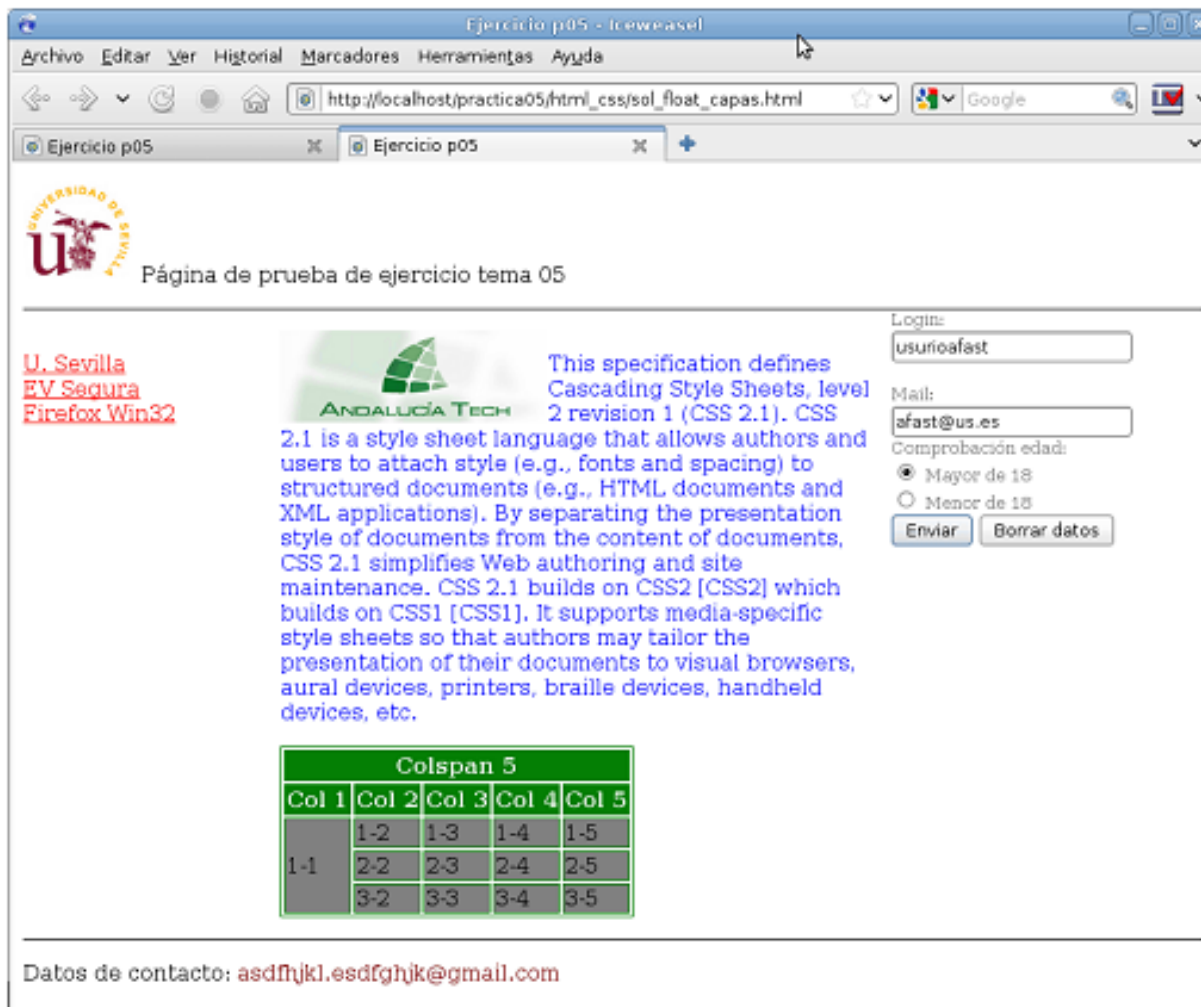
## TAREAS

- 1º Para comprobar el funcionamiento puede utilizar la pestaña “Red” de la consola de depuración del navegador. También puede usar `wireshark`. Si al usar `wireshark` ve las respuestas comprimidas, configure el navegador para que no las acepte: en una pestaña del navegador escriba `about:config`, después en el filtro Buscar escriba `encoding` y finalmente modifique `network.http.accept-encoding` para que contenga la cadena vacía.
- 2º Acceda desde el navegador mediante `http` al fichero `jsp/appcomentarios/getTotal.jsp` y analice la respuesta
- 3º Acceda desde el navegador mediante `http` a `jsp/appcomentarios/recuperar.jsp?id=0` y analice la respuesta.
- 4º ¿Qué modificaciones habría que añadir a `comentarios` para que se pudiera dejar de sondear? Añada un botón (y el código necesario) para que al pulsar sobre él deje de sondear, y si vuelve a pulsar vuelva a sondear. El mensaje del botón debe indicar lo que vaya a suceder al pulsar.

## 6 Propuesta de aplicación de la práctica

### 6.1 HTML y CSS

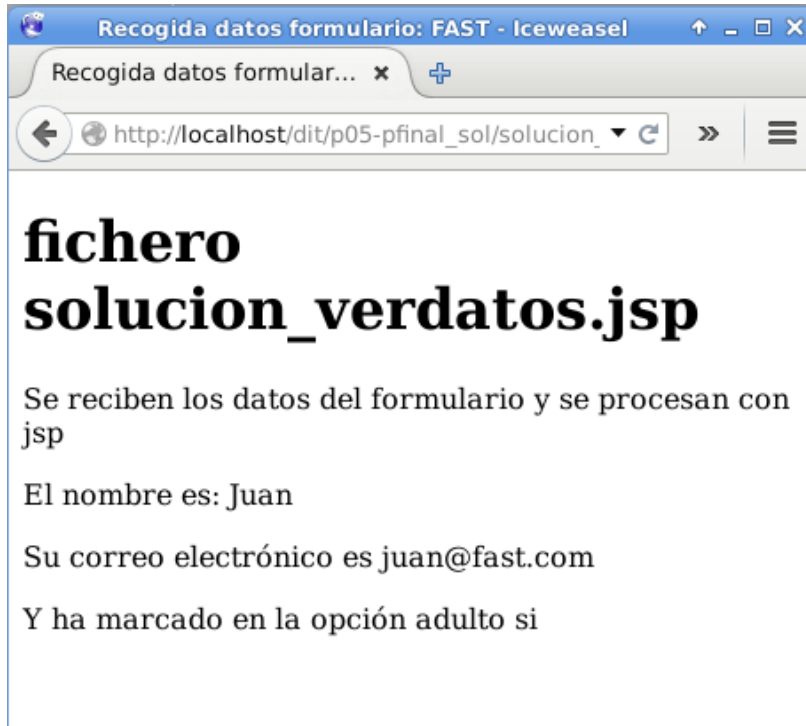
Deberá realizar una página web con una estructura o layout similar al recogido en la siguiente captura: cabecera, menú, zona principal, zona secundaria y pie.



- En la cabecera deberá incluir un logo, y el título de la página.
- En la zona de menú situará enlaces a páginas externas: seguros, no seguros y un enlace mediante protocolo ftp.
- En la zona principal incluirá texto e imágenes (observe que el texto fluye alrededor de la imagen). Al final de esa zona una tabla en la que deberá usar entre otras las etiquetas <th> y los atributos colspan y rowspan. El fondo de la tabla es gris, y el de las dos primeras filas verde con letras en blanco.
- En la zona secundaria un pequeño formulario de alta en un servicio imaginario que solicite un código de usuario, una dirección de correo electrónico, y una entrada de tipo “radio” para seleccionar si es mayor o menor de edad. Los nombres de estos inputs deberán ser “user”, “mail” y “adulto”.

- Para probar el formulario se le proporciona el fichero `solucion_verdatos.jsp`, muy similar al usado en la práctica. Deberá residir en el mismo directorio que su página, y ser enlazado desde el formulario usando el atributo `action`. Aquí se propone un uso básico del mismo para que pueda probar un formulario (para poder procesar en el servidor los datos del formulario recibidos desde el navegador).

El resultado obtenido en el navegador como respuesta resultante de ese fichero `jsp` es similar al mostrado en la siguiente captura:



Deberá usar hojas de estilos, y además deberá tener en cuenta que:

- La lista del menú de enlaces no tenga ninguna viñeta de estilo.
- El color por defecto de los párrafos sea azul.
- No deberán usarse atributo de formato en las etiquetas HTML.

Deberá probar estos ejemplos en su máquina. Tenga en cuenta que para probar la parte del formulario y `jsp` necesitará un servidor web donde desplegar su aplicación (debe arrancar `tomcat`).

Para practicar la inclusión de estilo CSS debe añadir los estilos:

- Con el atributo `style` en una etiqueta HTML
- Directamente en la cabecera, `head`, por ejemplo, un estilo que sólo se vaya a usar en esa página
- El resto en un fichero CSS.
- Nota: Puede crear un tercer fichero HTML que utilice este fichero CSS; para ver la ventaja de definir el estilo en un fichero CSS independiente. Este puede ser incluido por todos los ficheros de su sitio web y mantener un estilo uniforme, que además es muy sencillo e inmediato de modificar posteriormente.

## 6.2 XML

Deberá crear un fichero xml para almacenar la información de un curso online. El elemento raíz del fichero deberá llamarse curso, y tendrá

- un atributo obligatorio de nombre id
- un elemento titulo
- un elemento profesorado, que constará de uno o más elementos profesor, cada uno de ellos compuesto de un elemento nombre y un elemento mail.
  - El elemento profesor tendrá un atributo opcional lang, que si tuviera valor sólo podrá tomar los valores “en” o “es”.
- un elemento unidades, compuesto de uno o más elementos unidad. Cada elemento unidad tendrá
  - un atributo obligatorio titulo
  - un elemento objetivos
  - uno o más elementos fichero

Podrá comprobar si su fichero es correcto probando a validarlo con el fichero curso.dtd que se le proporciona.

Tarea opcional no evaluable: Cree un fichero de esquema curso.xsd que sea equivalente a curso.dtd, y compruebe si su fichero es correcto probando a validarlo con el fichero curso.xsd.

## 6.3 ECMAScript

Aplicación web que estará compuesta de varios ficheros: código HTML, código ECMAScript y hoja de estilo CSS. Todos los ficheros estarán incluidos en un directorio de nombre e1, y en el mismo existirá un subdirectorio css para la hoja de estilo y un subdirectorio js para los ficheros ECMAScript.

(1). Deberá aplicar una hoja de estilos (el fichero se llamará e1-estilo.css. y estará incluido en la carpeta css) a la página e1-index.html que se le proporciona para que tenga el aspecto que aparece en la figura tarea1.png. Tenga en cuenta que puede añadir etiquetas o atributos a la página e1-index.html, pero no puede borrar los ya existentes. La primera columna tendrá una anchura de un 25% y las otras de un 35% cada una.

E1: APELLIDOS NOMBRE - Iceweasel

Archivo Editar Ver Historial Marcadores Herramientas Ayuda

E1: APELLIDOS NOMBRE

Dinero es deuda

Producido por Paul Grignon, presenta la mecánica de creación de dinero por parte de los bancos y cómo se pasó del uso de monedas al uso de billetes, y posteriormente a las cuentas digitales, la supresión del patrón oro, el sistema de reserva fraccionada, etc.

¿Zona oculta?

**Contenido**

1. Historia: los Goldsmith
2. Sistema Monetario
3. Money as Debt II
4. Enlaces externos

**La historia..**


Una ficticia familia de orfebres comienzan a acuñar monedas de oro y darlas a sus vecinos para que las usen. Luego deciden comenzar a dar créditos en billetes. Sus conciudadanos, que creen que está prestando su oro, deciden retirar el que quede. Sin embargo, el oro sigue en el depósito, ya que habían "inventado" el dinero...

**Sistema actual**


Con la derogación del patrón oro, el único límite para la creación de dinero "mediante la promesa de pagarlo" es el dinero preexistente. Dado que el dinero se crea de cero con una deuda igual, el pago de todas las deudas supondría la desaparición de éste. Sin embargo, los intereses hacen que la deuda total sea mayor que el préstamo.. ver también el documental Zeitgeist: Addendum.

**2ª parte**

Money as Debt II: Promises Unleashed continúa la investigación del fraude de la denominada "modern banking".

**Zona de enlaces**

- [Vimeo](#)

Observe que hay texto en la página HTML que NO se muestra. Una de las posibles propiedades que hace que un elemento sea visible o no es la propiedad “display”. Si toma el valor “none” el elemento no se muestra. Si toma el valor “block” o “inline” el elemento se muestra como de bloque o en línea.

(2). Añadir a la aplicación código ECMAScript de forma que al pulsar el botón se solicite al usuario si desea ver o no la zona oculta. Si la respuesta es positiva se muestra, si es negativa se oculta. El código estará en un fichero de nombre e1-script.js, en la carpeta js.

Observe las secuencias en la figura tarea2.png. A la izquierda, el resultado de hacer clic sobre el botón y pulsar “Aceptar”. A la derecha, una vez mostrado el texto oculto, el resultado de hacer clic sobre el botón y pulsar “Cancelar”.





(3). Añadir un nuevo fichero `el-script2.js` (en la carpeta `js`) a la página `html`, al final de la etiqueta “`body`” (para que se ejecute al terminar de cargar la página), que permita que al pulsar sobre cualquier ítem de una lista ordenada se pueda modificar el contenido de dicho elemento, como se observa en las siguientes figuras. Observe la secuencia en `tarea3.png`: clic sobre el texto del primer elemento de la lista, tecleamos un texto, pulsamos “Aceptar” y observe que el texto del primer elemento queda modificado (si recarga la página volverá al texto original).



(4). Permitir que al pasar el ratón sobre cualquier párrafo este cambie su fondo a color verde. Al salir el ratón del párrafo debe volver al color original. Para ello deberá los eventos “`mouseenter`” y “`mouseleave`”. Añadir el código necesario en el fichero `el-script2.js`. Observe la secuencia en

tarea4.png, con el cursor en grande: cursor en primer párrafo, cursor en párrafo de la derecha, en párrafo central y cursor en lista ordenada



(5). Permitir que al hacer clic sobre una imagen se pueda modificar su tamaño, alterando la altura de la imagen. Al solicitar la nueva altura mostrará por defecto el valor actual. Añadir el código necesario en el fichero e1-script2.js. Observe la secuencia en tarea5.png.



NOTAS:

- Puede añadir etiquetas o atributos a la página e1-index.html, pero no puede borrar las etiquetas o atributos ya existentes. El fichero de estilo debe ir en el directorio css y los ficheros de código en el directorio js.

- Todas las figuras de este enunciado están en la carpeta figuras.
- Los colores aplicados son “silver”, “cyan”, “green” y “yellow”. Fuentes: courier para zona oculta, fuente por defecto para el resto. Tamaño de fuente al 80%.
- Valores de la propiedad display que puede utilizar: none | block | inline
- Las capturas están realizadas con resolución 800x600.

## 6.4 AJAX

Se desea hacer un formulario en el que aparezcan valores en una lista desplegable que dependan de valores previamente escogidos, y que los valores que aparecen sean solicitados al servidor.

Se va a utilizar el directorio AJAX dentro de la aplicación práctica. Si ha descargado correctamente los ficheros, puede empezar accediendo con el navegador mediante http al fichero p02-ap/AJAX/ejer\_ajax\_01.html.

Se parte de un formulario básico totalmente estático donde las listas desplegables son fijas y no hay ningún código ECMAScript. Observe el fichero `ejer_ajax_01.html` y compruebe que al enviar el formulario aparecen en pantalla los códigos enviados (que no son iguales que los nombres de la lista desplegable). Puede activar las herramientas de desarrollo (por ej. pulsando F12) y en la pestaña “Red” observar qué método se usa para solicitar la página (pulse recargar página) o para enviar el formulario (botón “Enviar datos”). Este formulario es poco útil ya que permite escoger cualquier combinación de comunidad autónoma, provincia y localidad, aunque no tenga sentido.

Una vez comprendido el funcionamiento del fichero anterior, observe el fichero `ejer_ajax_02.html` y compruebe que en este caso las listas desplegables en el fichero html están vacías (sólo tienen un elemento, el de “Seleccione una opción”) y que cuando se ejecuta el código ECMAScript asociado se rellena la lista desplegable correspondiente a la selección de comunidad autónoma. Una vez seleccionada la comunidad autónoma, entonces aparecen las provincias correspondientes, y una vez seleccionada la provincia aparecen las localidades correspondientes. Intente deducir cómo es el código de `ejer_ajax_02.js` (una solución podría ser almacenando los datos en tablas ECMAScript, y usando funciones de acceso al DOM para añadir opciones a las listas desplegables). Observe el código del fichero anterior y compruebe su funcionamiento con las herramientas de desarrollo, pulsando en la pestaña “Depurador” y poniendo un punto de ruptura en la primera instrucción de la función `rellena_aut` (que es la primera que se ejecuta, después de cargar el cuerpo del documento html) y de la función `ha_cambiado` (que se ejecuta cuando cambia la selección de comunidad autónoma o de provincia). Vaya ejecutando paso a paso para ver cómo se van rellenando las opciones de las listas desplegables (avance entrando en las funciones, puede hacerlo pulsando F11).

Una vez comprendido el funcionamiento del fichero anterior, compruebe que `ejer_ajax_03.html` es igual que el 02, pero que ahora el fichero de script es el 03, y desarrolle el código necesario en `ejer_ajax_03.js` para que ahora los datos correspondientes a cada comunidad autónoma se descarguen en formato JSON, concretamente accediendo a los ficheros `datos_AND.json` y `datos_EXT.json` en el servidor. Si le resulta complicado el desarrollo del código, pruebe a modificar el fichero proporcionado. Para la modificación, simplemente descomente

las líneas que contienen ???, sustituyéndolo por el valor adecuado. Una vez modificado con éxito, compruebe con las herramientas de desarrollo, pulsando en la pestaña “Red”, cómo se descarga el fichero JSON correspondiente cada vez que se cambia la comunidad autónoma, pero no cuando se cambia la provincia. Compruebe también que `datos_provloc` contiene los datos como una tabla (para ello habrá tenido que usar la propiedad `responseText` del objeto `XMLHttpRequest` y además el método `JSON.parse`). Si modifica los ficheros JSON en el servidor, y estas modificaciones no se ven reflejadas en el navegador, puede ser que el navegador esté usando la caché y no esté descargando los ficheros. Para solucionar eso, pruebe a añadir un valor cambiante en la url solicitada, por ejemplo añadiendo una valor que dependa del instante actual (separado por el carácter "?"), para que el navegador no use la caché. Así las peticiones serían algo así como:

```
GET datos_AND.json?123456
```

```
GET datos_AND.json?123457
```

```
GET datos_AND.json?123458
```

Este cambio en la url "obliga" al navegador a volverlo a solicitar.

Una vez comprendido el funcionamiento del fichero anterior, compruebe que `ejer_ajax_04.html` es igual que el 03, pero que ahora el fichero de script es el 04, y modifique el código necesario en `ejer_ajax_04.js` para que ahora los datos correspondientes a cada comunidad autónoma se descarguen en formato XML, concretamente accediendo a los ficheros `datos_AND.xml` y `datos_EXT.xml` en el servidor. Haga las modificaciones de forma similar a como lo ha hecho en el ejercicio anterior. Utilizando las herramientas de desarrollo compruebe que la variable `datos_prov_loc_xml` contiene los datos como un documento XML (para ello habrá tenido que usar la propiedad `responseXML` del objeto `XMLHttpRequest`, pero no habrá hecho falta usar ningún método en la asignación). En este ejercicio se accede al documento XML mediante métodos del DOM, y se crean las tablas necesarias para poder usar las funciones del ejercicio 02, las que añadían opciones a las listas desplegables a partir de tablas bidimensionales.

#### APAGUE EL EQUIPO

Cuando finalice la práctica no olvide apagar el equipo.

## 7 Anexo (no evaluable)

### 7.1 JQuery

jQuery es una de las bibliotecas ECMAScript más extendidas. Su autor original es John Resig, aunque como sucede con todas las librerías de gran uso, actualmente recibe contribuciones de múltiples

programadores. jQuery se ha programado de forma eficiente y su versión compacta apenas ocupa 90 KB.

La documentación de jQuery es muy completa e incluye muchos ejemplos. Además, también existen algunos recursos útiles para aprender su funcionamiento básico como:

<https://www.w3schools.com/jquery/>

La forma de incluir la biblioteca jQuery en una página html desde fichero local para poder ser usada en sus scripts será:

```
<script src="jquery.js"></script>
```

Dada su utilidad, está disponible en muchos servidores, y es posible incluir el fichero con una referencia remota. Por ejemplo, las siguientes referencias incluyen la última versión disponible:

- Google AJAX API CDN

```
src="http://ajax.googleapis.com/ajax/libs/jquery/1/jquery.min.js"
```

- jQuery CDN

```
src="http://code.jquery.com/jquery-latest.min.js"
```

Una vez incluido el fichero jquery.js puede utilizar todos los métodos de la API JQuery.

Es posible también separar el código ECMAScript del código html e incluirlo en varios ficheros externos. En ese caso tendrá en el fichero html referencias a dos ficheros con código ECMAScript:

- la primera para incluir la biblioteca JQuery.

```
<script src="jquery.js"> </script>
```

- la segunda, a un fichero donde está el código implementado por el programador.

```
<script src="codigo_propio.js"> </script>
```

## 7.2 Clases y herencia con ECMAScript

Incluir los métodos como funciones dentro de la propia función constructora es una técnica que funciona correctamente, pero que tiene un inconveniente: cada vez que se instancia un objeto, se definen tantas nuevas funciones como métodos incluya la función constructora (el código de las funciones está repetido). La penalización en el rendimiento y el consumo de recursos de esta técnica puede suponer un inconveniente en las aplicaciones realizadas con ECMAScript. Para evitar esto se usa “prototype”.

ECMAScript es un lenguaje orientado a funciones. En él, todas las funciones tienen una propiedad llamada `prototype`. Esta propiedad será utilizada como ‘modelo’ inicial de todos los objetos que sean creados con esta función cuando sea utilizada como constructor. Inicialmente esta propiedad es un objeto vacío, pero debemos modificarla para aprovechar esta característica del lenguaje.

Normalmente los métodos no varían de un objeto a otro del mismo tipo, por lo que se puede evitar el problema comentado anteriormente añadiendo los métodos al prototipo a partir del cual se crean los

objetos. Para ello sacaremos los métodos de la función constructora, y los añadimos a continuación usando `prototype`:

**objetos03.js**

```
// Creamos nuestra función constructora
// sin los métodos
function Impresora (id){
 this.id=id;
};
// Usamos prototype para los métodos
Impresora.prototype.muestraId = function() {
 alert(this.id);
};
```

**TAREAS**

- 1º Pruebe el código de los ficheros `objetos-01.js`, `objetos-02.js` y `objetos-03.js`. Para ello debe usar un fichero HTML que incluya dichos ficheros.
- 2º A partir de `objetos-03.js`, añada la clase `Impresora` una propiedad llamada `usuarios` (sin `prototype`, dentro de la función constructora), que almacenará en una tabla los usuarios permitidos de esa impresora. Además añada un método `muestraUsuarios`, que muestra los usuarios con `alert`.
- 3º Añadir al apartado anterior el método `nuevoUsuario` (con `prototype`, fuera de la función constructora) que añade un nuevo usuario a los objetos de tipo `Impresora`. Este método recibe como parámetro el nombre del usuario a añadir al final de la propiedad `usuarios` (una forma de hacerlo es asignándolo al elemento cuyo índice coincide con la longitud).
- 4º Asigne nuevos usuarios distintos a cada impresora y compruebe su buen funcionamiento.
- 5º Cambie la propiedad `usuarios`, y en vez añadirla en la función constructora, añádala con `prototype`, fuera de la función constructora.
- 6º Asigne nuevos usuarios distintos a cada impresora y compruebe cómo es el funcionamiento ahora. Conclusión: usar `prototype` para los métodos pero no para las propiedades.

Si además quiere aplicar herencia, nuevamente no está disponible, y hay que “emularla”. Puede encontrar información de cómo hacerlo en la siguiente dirección:

[https://developer.mozilla.org/es/docs/Web/JavaScript/Introducci%C3%B3n\\_a\\_JavaScript\\_orientado\\_a\\_objetos](https://developer.mozilla.org/es/docs/Web/JavaScript/Introducci%C3%B3n_a_JavaScript_orientado_a_objetos)

Aunque queda fuera del alcance de esta asignatura, puede ver un ejemplo en el fichero `objetos-11.js`.

En la versión 6 de ECMAScript se ha introducido una nueva sintaxis y la palabra reservada `class` (y `extends`), pero la naturaleza de las clases no ha cambiado (aunque la nueva sintaxis es más intuitiva).