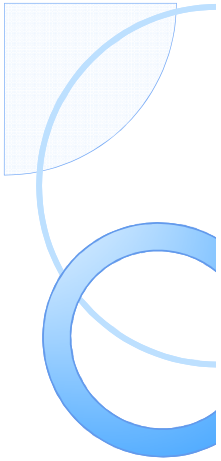
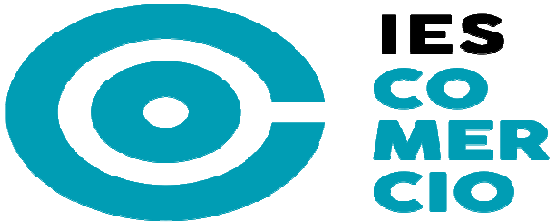




Unidad 7: Utilización del Modelo de Objetos del Documento (DOM-Document Object Model)

Desarrollo Web en Entorno Cliente

Objetivos

- Reconocer el modelo de objetos del documento de una página web, identificando sus objetos, propiedades y métodos.
- Generar y verificar código que acceda a la estructura del documento y crear elementos de la estructura.
- Asociar acciones a los eventos del modelo.
- Identificar las diferencias del modelo en distintos navegadores. Programar aplicaciones para que funcionen en los distintos navegadores.

Contenidos

1. El Modelo de Objetos del Documento (DOM).
2. Programación de eventos.
3. Diferencias en las implementaciones del modelo.
4. Aplicaciones Cross-Browser (multi-cliente).

1.- EL MODELO DE OBJETOS DEL DOCUMENTO (DOM)

- Es una de las innovaciones que más han influido en el desarrollo de paginas web dinámicas y de las aplicaciones web más complejas.
- Es una interfaz de programación de aplicaciones (API) de la plataforma de W3C, que proporciona un conjunto estándar de objetos para representar documentos HTML y XML.
- Permite a los scripts acceder y actualizar dinámicamente su contenido, estructura y estilo de documento.

1.- EL MODELO DE OBJETOS DEL DOCUMENTO (DOM)

- Fue utilizado por primera vez con el navegador Netscape Navigator versión 2.0.
- A esta primera versión de DOM se le denomina *DOM nivel 0* o *modelo básico*.
- El primer navegador de Microsoft que utilizó el DOM nivel 0 fue IE 3.0.
- Debido a las diferencias entre los navegadores, W3C emitió una especificación a finales de 1998 que llamó *DOM nivel 1*.
- En esta especificación ya se consideraba la manipulación de todos los elementos existentes en los archivos HTML y de XML.

1.- EL MODELO DE OBJETOS DEL DOCUMENTO (DOM)

- A finales del año 2000, W3C emitió *DOM nivel 2*, en la cual se incluía el manejo de eventos en el navegador y la interacción con hojas de estilo CSS.
- En 2004 se emitió *DOM nivel 3*, en la cual se utiliza la definición de tipos de documento (DTD) y la validación de documentos.

1.- EL MODELO DE OBJETOS DEL DOCUMENTO (DOM)

- Actualmente DOM se divide en tres partes o niveles según la W3C:
  - Núcleo del DOM (Core DOM): modelo estándar para cualquier documento estructurado. A este nivel se especifican las pautas para definir los objetos y propiedades de cualquier documento estructurado, así como para acceder a ellos.
  - XML DOM: modelo estándar para los documentos XML, define los objetos y propiedades de todos los elementos XML, así como para acceder a ellos.
  - **HTML DOM**: estándar para los documentos HTML. Interfaz de programación estándar para HTML independiente de la plataforma y el lenguaje. Define los objetos y propiedades de todos los elementos HTML, así como para acceder a ellos.

1.- EL MODELO DE OBJETOS DEL DOCUMENTO (DOM)

1.2.-Objetos del DOM HTML, propiedades y métodos

- Mediante el modelo DOM HTML, el navegador web transforma automáticamente todas las páginas web en una estructura de árbol, para que de esta manera se pueda acceder a los elementos de la página.
- Las páginas web son una sucesión de caracteres, por lo que resultaría excesivamente complicado manipular los elementos si no fuera por esta conversión.

1.- EL MODELO DE OBJETOS DEL DOCUMENTO (DOM)

1.2.-Objetos del DOM HTML, propiedades y métodos

Sintaxis para acceder:

```
document.getElementById(objetoID).propiedad | metodo([parametros])
```

Listado de objetos del DOM en HTML				
Document	HTMLElement	Anchor	Area	Base
Body	Button	Evento	Form	Frame/iFrame
Input Hidden	Image	Input Button	Input Checkbox	Input File
Input Text	Link	Meta	Object	Option
Select	Style	Table	Table Cell	Table Row
textarea				

1.- EL MODELO DE OBJETOS DEL DOCUMENTO (DOM)

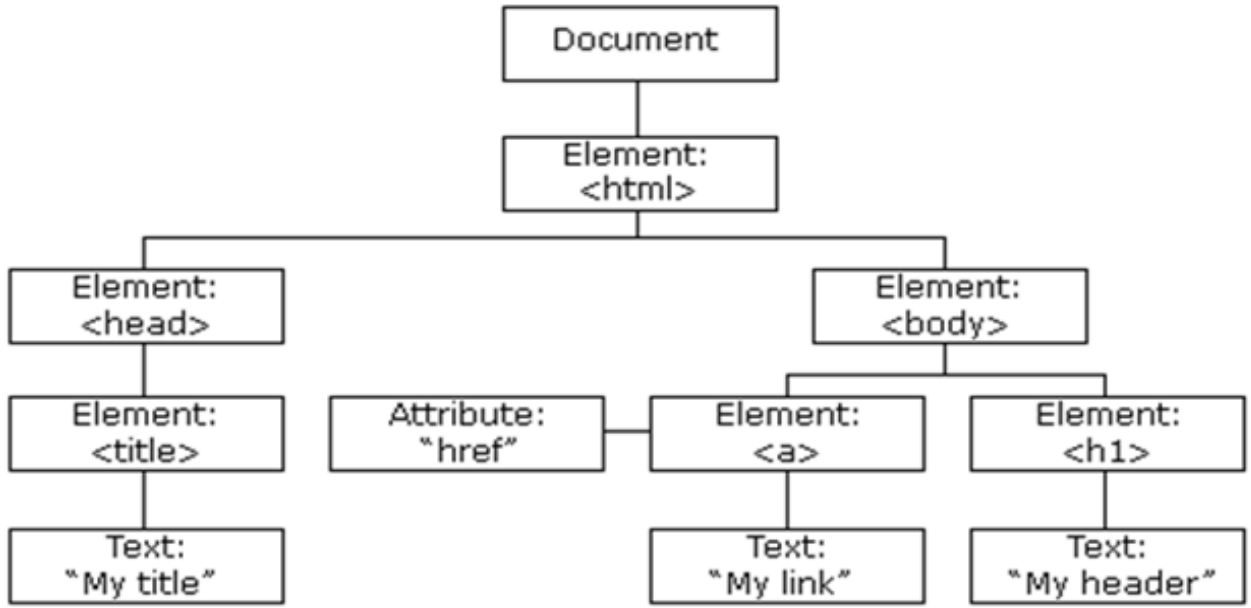
1.3.-Estrucutra del árbol DOM y tipos de nodos

- DOM transforma los documentos HTML en elementos. Estos elementos se llaman **nodos**. Cada nodo es un objeto.
- A su vez, los nodos están interconectados y muestran el contenido de la página web y la relación entre nodos.
- Cuando unimos todos estos nodos de forma jerárquica, obtenemos una estructura de árbol, que se le referencia como **árbol de nodos**.

1.- EL MODELO DE OBJETOS DEL DOCUMENTO (DOM)

- **Estructura del árbol DOM:**
  - Dado el siguiente código HTML:

```
<html>
  <head>
    <title>My tittle</title>
  </head>
  <body>
    <a href="">My link</a>
    <h1>My header>/h1>
  </body>
</html>
```



1.- EL MODELO DE OBJETOS DEL DOCUMENTO (DOM)

Tipos

- **Document**, es el nodo raíz y del que derivan todos los demás nodos del árbol.
- **Element**, representa cada una de las etiquetas XHTML. Es el único nodo que puede contener atributos y el único del que pueden derivar otros nodos.
- **Attr**, con este tipo de nodos representamos los atributos de las etiquetas XHTML, es decir, un nodo por cada atributo=valor.
- **Text**, es el nodo que contiene el texto encerrado por una etiqueta XHTML.
- **Comment**, representa los comentarios incluidos en la página XHTML.
- Los otros tipos de nodos pueden ser: **CdataSection**, **DocumentFragment**, **DocumentType**, **EntityReference**, **Entity**, **Notation** y **ProcessingInstruction**.

1.- EL MODELO DE OBJETOS DEL DOCUMENTO (DOM)

1.4.- Acceso directo a los nodos:

Se puede hacer de dos métodos:

- a través de los nodos padre → partimos del nodo raíz y vamos accediendo a los nodo hijo y as sucesivamente hasta llegar al elemento deseado.
- usando el método de acceso directo → empleamos funciones del DOM, para ir directamente aun nodo sin pasar nodo a nodo.

Para que podamos acceder a los nodos, el árbol debe estar completamente construido.

1.- EL MODELO DE OBJETOS DEL DOCUMENTO (DOM)

- Para realizar la estructura del árbol, existe una serie de reglas:
  - En el árbol de nodos, al nodo superior (document), se le llama raíz.
  - Cada nodo, exceptuando el nodo raíz, tiene un padre.
  - Una hoja es un nodo sin hijos.
  - Los nodos que comparten el mismo padre, son hermanos.

1.- EL MODELO DE OBJETOS DEL DOCUMENTO (DOM)

- Una vez que tenemos creada la estructura de árbol completa, ya disponemos de todos los nodos del árbol DOM.
- El estándar considera un nodo a cada una de las partes del árbol.
- A continuación veremos los tipos de nodos más importantes que pueden existir en un árbol DOM. Los tipos de nodos indicados son específicos del lenguaje XML, no obstante estos pueden ser aplicados a los lenguajes basados en XML como XHTML y HTML.

1.- EL MODELO DE OBJETOS DEL DOCUMENTO (DOM)

ACCESO DIRECTO A LOS NODOS

- Cuando se trabaja con una página web real, existen miles de nodos de todos los tipos en el árbol DOM. Así que no es eficiente acceder a un nodo descendiendo a través de todos los ascendentes.
- DOM añade una serie de métodos para acceder de forma directa a los nodos:
  - `getElementsByTagName()`
  - `getElementsByName()`
  - `getElementById()`
  - `querySelector()`
  - `querySelectorAll()`

ACCESO DIRECTO A LOS NODOS

- **`getElementsByTagName(nombre_etiqueta)`** obtiene todos los elementos de la página cuya etiqueta HTML sea igual que el parámetro que se le pasa. Ejemplo:

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

La función devuelve un array con todos los nodos que cumplen la condición de que su etiqueta coincide con el parámetro proporcionado. En realidad el array es una lista de nodos (`nodeList`), por tanto se debe procesar cada valor del array.

```
var primerParrafo = parrafos[0];

for (var i=0; i<parrafos.length; i++) {
    var párrafo = parrafos [i];
  }
```

La función se puede aplicar de forma recursiva sobre cada uno de los nodos devueltos por la función:

```
var parrafos = document.getElementsByTagName("p");
Var primerParrafo = párrafos[0];
Var enlaces = primerParrafo.getElementsByTagName("a");
```

ACCESO DIRECTO A LOS NODOS

- **`getElementsByName()`** recupera todos los elementos de la página cuyo atributo `name` coincide con el parámetro pasado a la función.

```
var parrafoEspecial = document.getElementsByName("especial");

<p name="prueba">...</p>
<p name="especial">...</p>
<p>...</p>
```

Normalmente el atributo `name` es único para los elementos HTML que lo definen.

En el caso de elementos radiobutton, donde el atributo `name` es común a todos los radiobutton que están relacionados, devuelve un array `nodeList` con los elementos.

ACCESO DIRECTO A LOS NODOS

- **`getElementByID()`** recupera el elemento HTML cuyo id coincide con el parámetro pasado a la función. Como el atributo id debe ser único para cada elemento, la función devuelve únicamente el nodo deseado. Es la función mas utilizada.

```
var cabecera = document.getElementById("cabecera");
```

```
<div id="cabecera">
  <a href="URL" id="logo">...</a>
</div>
```

ACCESO DIRECTO A LOS NODOS

- **`querySelector()`** acepta como parámetro un selector que identifica el elemento (o elementos) a seleccionar. En el caso de esta función, únicamente se devuelve el primer elemento que cumple la condición. Si no existe el elemento, el valor retornado es `null`.

```
var logo = document.querySelector (".enlace");
<div id = "cabecera">
   <a href="/" clas="enlace">…</a>
</div>
<div id = "cuerpo">
   <p>Lorem ipsu <a href="enlace">…</a></p>
</div>
```

- En este caso, a pesar de existir varios elementos de la clase enlace, únicamente es seleccionado el primero de ellos.

ACCESO DIRECTO A LOS NODOS

- **`querySelectorAll()`** acepta como parámetro un selector que identifica el elemento (o elementos) a seleccionar. Esta función devuelve un objeto de tipo `NodeList` con los elementos que coincidan con el selector.

```
var enlaces = document.querySelectorAll (".enlace");
<div id = "cabecera">
   <a href="/" clas="enlace">…</a>
</div>
<div id = "cuerpo">
   <p>Lorem ipsu <a href="enlace">…</a></p>
</div>
```

Para acceder a los elementos almacenado en NodeList, recorremos el objeto como si de un array se tratase.

```
for (var i=0; i<enlaces.length; i++)
{
   var enlaces = enlaces[i];
}
```

ACCESO DIRECTO A LOS NODOS

- **Creación de elementos XHTML simples**

  Consta de cuatro pasos:

  1. Creación de un nodo `Element` que represente al elemento.
  2. Creación de un nodo de tipo `Text` que represente el contenido del elemento.
  3. Añadir el nodo `Text` como nodo de `Element`.
  4. Añadir el nodo `Element` a la pagina, en forma de nodo hijo.

  Si se quiere añadir un párrafo simple al final de la pagina, es necesario incluir el siguiente código:

```
// Crear el nodo element como hijo de la página
var párrafo = document.createElement ("p");
//Crear nodo de tipo Text
var contenido = document.createTextNode ("Hola mundo");
//Añadir el nodo Text como hijo del nodo Element
parrafo.appendChild (contenido);
//Añadir el nodo Element como hijo de la página
document.body.appendChild (parrafo);
```

ACCESO DIRECTO A LOS NODOS

El proceso de creación de nuevos nodos implica la utilización de tres funciones DOM, por lo que puede resultar un poco tedioso:

`createElement(etiqueta)`: crea un nodo de tipo `Element` que representa al elemento XHTML cuya etiqueta se le pasa como parámetro.

`createTextNode(contenido)`: crea un nodo de tipo `Text` que almacena el contenido textual de los elemento XHTML.

`nodoPadre.appenChild(nodoHijo)`: añade un nodo como hijo de otro nodo. Se debe utilizar al menos dos veces con los nodos habituales, uno para añadir el nodo `Text` como hijo del nodo `Element` y a continuación para añadir el nodo `Element` como hijo de algún nodo de la página.

ACCESO DIRECTO A LOS NODOS

- **Inserción de elementos XHTML simples**

Se puede hacer de dos formas:

1.- Mediante el método **`appendChild()`**, lo cual lo inserta como elemento hijo del de referencia, pero al final de los elementos del padre.

```
padre = document.getElemntById ("lateral");
padre.appendChild(nuevo);
```

2.- Se basa en el método **`insertBefore()`**.Inserta el elemento nuevo delante de la etiqueta que hayamos seleccionado

```
var referencia = document.getElementById("insertar");
var padre = referencia.parentNode;
padre.insertBefore (nuevo, referencia);
```

ACCESO DIRECTO A LOS NODOS

- **Eliminación de elementos XHTML simples**

  Se hace con la función **`removeChild()`**, esta función requiere como parámetro el nodo que se va a eliminar.

  La función debe ser ejecutada desde el nodo padre.

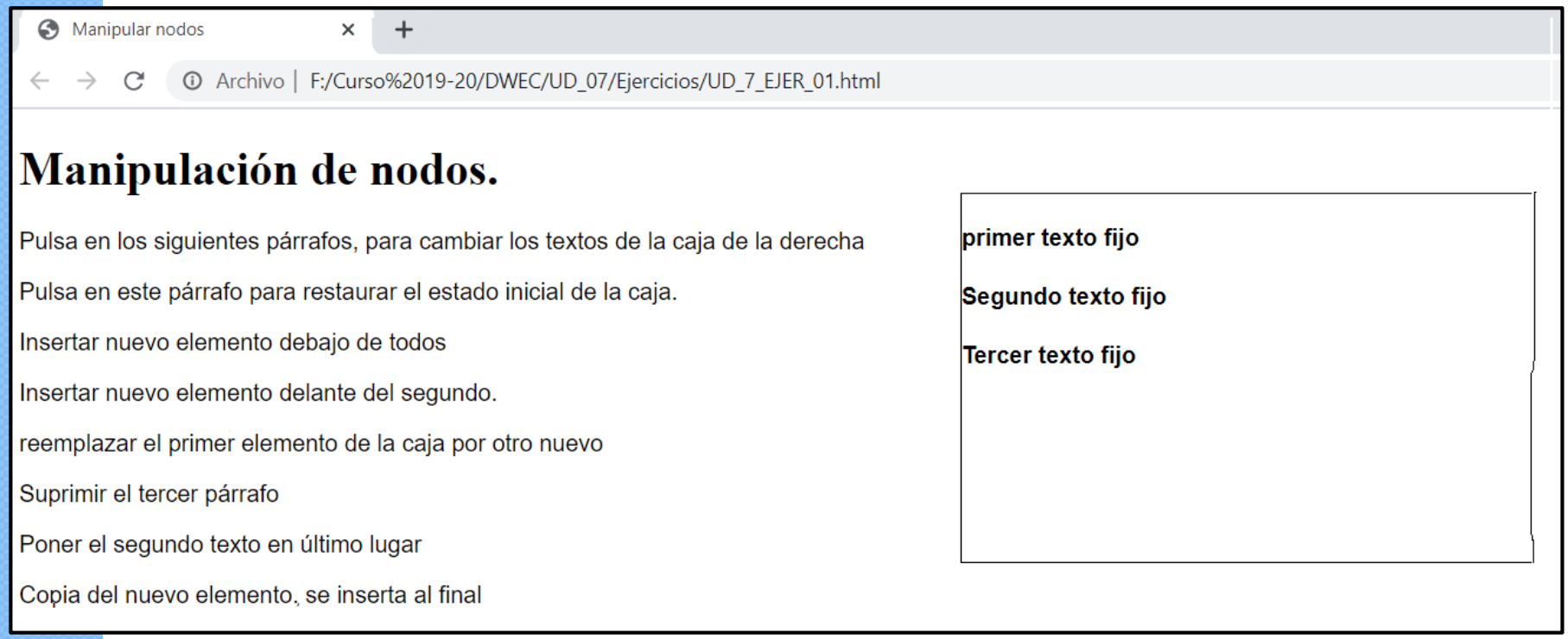
  Para acceder al nodo padre se hace mediante la propiedad **`nodoHijo.parentNode.`**

  Cuando se elimina un nodo, también se eliminan automáticamente todos los nodos hijos que tenga, por lo que no es necesario borrar manualmente cada nodo hijo.

Ejercicio_01 Manipular nodos

Ejercicio_01 Manipular nodos

Con todo lo visto hasta ahora de manipulación de nodos, crea una pagina web como la siguiente:



ACCESO DIRECTO A LOS ATRIBUTOS XHTML

Mediante DOM también es posible acceder de forma sencilla a los atributos XHTML y a todas las propiedades CSS de cualquier elemento de la pagina.

Los atributos XHTML de los elementos se transforman automáticamente en propiedades de los nodos.

```
var enlace = document.getElementById("enlace");
console.log(enlace.href)
<a id="enlace" href=http://www.google.es>Enlace</a>
```

También podemos leer y escribir los atributos mediante los métodos **`getAttribute("nombreAtrib")`**, para lectura; y **`setAttribute("nombreAtrib","valorAtrib")`** para escritura.

ACCESO DIRECTO A LOS ATRIBUTOS XHTML

Ejemplo:

La lectura del atributo:

```
elemento = document.getElementsByTagName("a")[0];
direccion = elemento.getAttribute("href");
```

La escritura del atributo:

```
elemento = document.getElementsByTagName("a")[0];
elemento.setAttribute("href", http://google.es);
```

Mediante la escritura no solo se puede cambiar un valor de un atributo, sino que se pueden añadir nuevos atributos a las etiquetas.

ACCESO DIRECTO A LOS ATRIBUTOS XHTML

Acceso a las propiedades CSS:

Las propiedades CSS requieren un paso extra. Se debe utilizar el atributo **`.style`** y después el nombre de la propiedad.

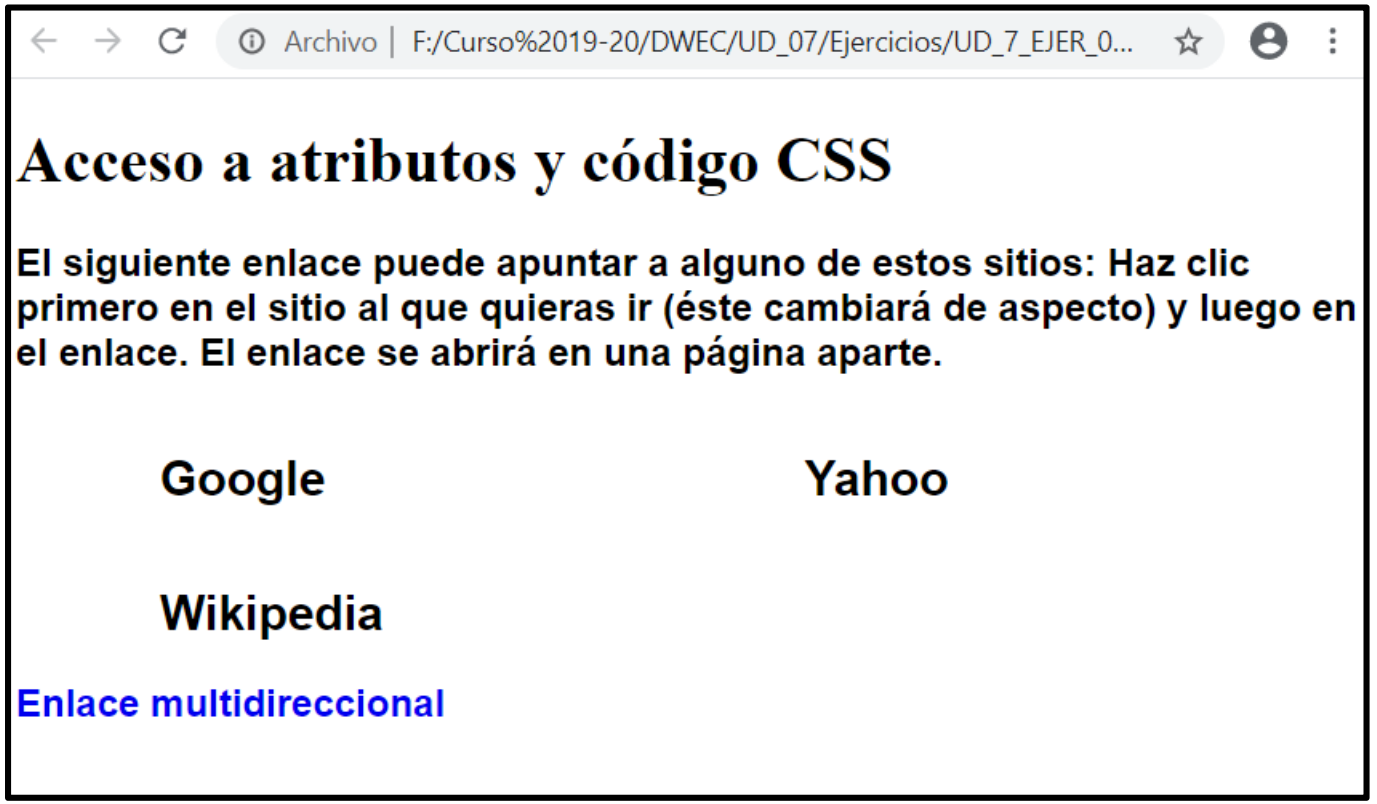
```
var imagen = document.getElementById("parrafo");
console.log(parrafo.style.fontWeight);
<p id="párrafo" style="font-weight: bold;"> …</p>
```

La transformación del nombre de propiedades CSS compuesta se hace eliminando los guiones intermedios y escribiendo la siguiente letra en mayúscula.

Ejercicio_02 Acceso a atributos y propiedades nodos

Ejercicio_02 Acceso a atributos y propiedades nodos

Cuando hacemos clic sobre un texto lo cambia de color (rojo) y lo pone en negrita. Cuando pulsamos Enlace multinivel, va a la página del texto seleccionado.



Propiedades y métodos del los objetos nodo

- La interfaz Node:
  - Para poder manipular la información de los nodos, JavaScript crea un objeto llamado `Node`.
  - En este objeto se definen las propiedades y los métodos para procesar y manipular los documentos.
  - Este objeto define una serie de constantes que identifican los tipos de nodo.
  - En la siguiente tabla vemos la relación entre las constantes y el número que se asigna a cada uno de los tipos de nodo:

- Constantes del objeto `Node`:

Tipo de nodo=Valor	
`Node.ELEMENT_NODE = 1`	`Node.PROCESSING_INSTRUCTION_NODE = 7`
`Node.ATTRIBUTE_NODE = 2`	`Node.COMMENT_NODE = 8`
`Node.TEXT_NODE = 3`	`Node.DOCUMENT_NODE = 9`
`Node.CDATA_SECTION_NODE = 4`	`Node.DOCUMENT_TYPE_NODE = 10`
`Node.ENTITY_REFERENCE_NODE = 5`	`Node.DOCUMENT_FRAGMENT_NODE = 11`
`Node.ENTITY_NODE = 6`	`Node.NOTATION_NODE = 12`

Además de estas constantes, el objeto **`Node`** proporciona también una serie de propiedades y métodos para poder acceder a la jerarquía de elementos del árbol.

- Propiedades de `Node` :

Propiedad	Valor	Descripción
`nodeName`	String	Nombre del nodo (no está definido para algunos tipos de nodos).
`nodeValue`	String	Valor del nodo (no está definido para algunos tipos de nodos).
`nodeType`	Integer	Constante que representa a cada tipo.
`parentNode`	Node	Referencia al padre del nodo hijo (siguiente contendor más externo).
`childNodes`	NodeList	Lista de todos los nodos hijos del nodo actual.
`firstChild`	Node	Referencia al primer nodo de la lista childNode .
`lastChild`	Node	Referencia al último nodo hijo.
`previousSibling`	Node	Referencia al hermano siguiente según su orden en el código fuente.
`nextSibling`	Node	Referencia al nodo hermano anterior o null si este nodo es el primer hermano.
`attributes`	nodeMap	Lo empleamos con nodos de tipo element. Contiene objetos de tipo Attr que definen todos los atributos del elemento.
`ownerDocument`	Document	Referencia del documento al que pertenece.
`namespaceURI`	String	URI a la definición de namespace.
`Prefix`	String	Predijo del namesapace.
`localName`	String	Aplicable a los nodos afectados en el namespace.

- Métodos de `Node` :

Métodos	Valor	Descripción
appendChild(nodo)	Node	Añade un nuevo nodo al final de la lista childNode.
removeChild(nodo)	Node	Elimina un nodo de la lista childNodes.
replaceChild(nuevoNodo, anteriorNodo)	Node	Reemplaza el nodo anteriorNodo por el nodo nuevoNodo.
insertBefore(nuevoNodo, anteriorNodo)	Node	Inserta el nodo nuevoNodo antes de la posición del nodo anteriorNodo dentro de la lista childNodes.
cloneNode(deep)	Node	Realiza una copia del nodo actual (opcionalmente con todos sus hijos)
hasChildNodes()	Boolean	Determina si el nodo actual tiene o no hijos (valor boolean)
isSupported(feature, versión)		Determina cuando el nodo soporta una característica especial.

Con estos métodos podemos acceder a los diferentes nodos del árbol y también a la jerarquía. Así como crear, modificar y eliminar nodos.

- **Acceso a los tipos de nodo**

Los objetos del modelo se diferencian por su tipo. La forma de acceder es a través de la propiedad **`nodeType`**.

Ejemplo:

```
Obj_tipo_document = document.nodeType;     //Devuelve 9

Obj_tipo_document = document.documentElement.nodeType; //Devuelve 1
```

En caso de tener un navegador que no tenga las constantes definidas por defecto, lo deberíamos hacer de forma explicita. Esto ocurre en la versión 7 de Internet Explorer.

Propiedad innerHTML

Esta propiedad nos permite acceder al código HTML comprendido entre las etiquetas de apertura y cierre que definen un nodo, para modificarlas.

El potencial de esta propiedad es el de poder modificar el código HTML en si, no solo el texto.

Dentro del código que escribamos podemos poner otras etiquetas, de forma que queden anidadas a la de referencia.

Sintaxis:

```
nodoAlqueAccedemos.innerHTML = "nuevo código HTML en el nodo";
```

Ejercicio_03 propiedad innerHTML

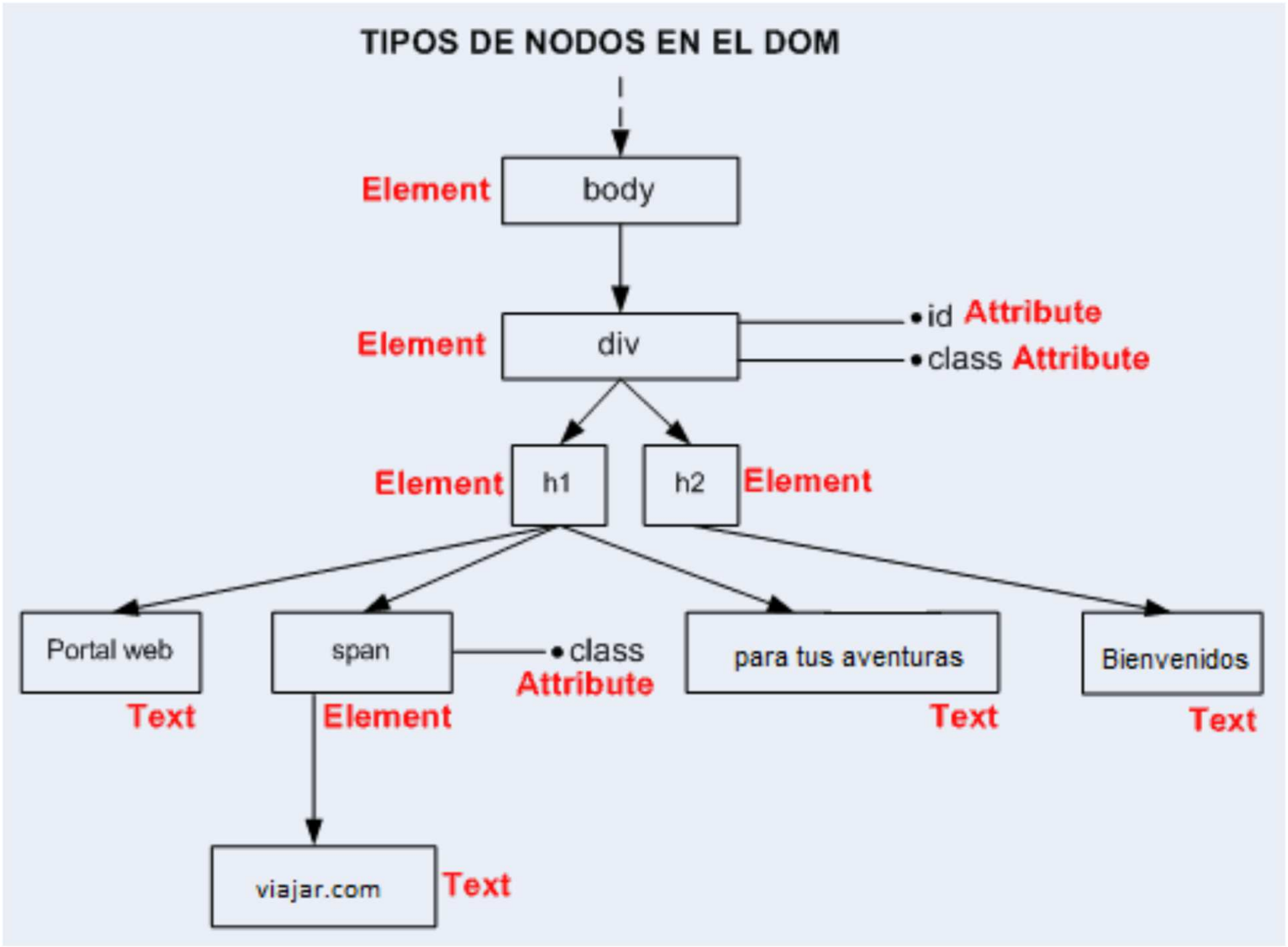
Ejercicio_03 propiedad innerHTML

Crea un documento HTML con un **`div`** central (el cuadrado principal) que contenga a su vez cuatro **`divs`** (los cuadros secundarios). Debajo del cuadrado principal debe aparecer un botón con el texto "**AVANZAR**". Inicialmente los cuadros tendrán el fondo blanco y un signo **?** De gran tamaño. A pulsaren el botón por primera vez, el cuadro superior izquierdo debe aparecer con el fondo purpura, texto en blanco y contener el texto "**La**". Al pulsar de nuevo el botón, el cuadro superior derecho debe aparecer con fondo blanco y texto en negro y tener como texto "**Vida**". Al pulsar de nuevo el botón el cuadrado inferior izquierdo debe aparecer con fondo blanco y texto negro y tener como texto "**Es**". Al pulsar de nuevo el botón el cuadro inferior derecho debe aparecer con fondo amarillo y texto negro y tener como texto "**Bella**". Si se pulsa nuevamente el botón AVANZAR debe mostrarse un mensaje que indique "**No es posible avanzar mas. Nos hemos quedado sin cuadrados**"

EJEMPLO:

- Dibuja la estructura del árbol DOM del siguiente fragmento de código:

```
<body>
  <div id="cabecera" class="brillante">
    <h1>Portal web<span class="destacado">
       viajar.com</span>para tus aventuras</h1>
    <h2>Bienvenidos</h2>
  </div>
</body>
```


TIPOS DE NODOS EN EL DOM
Element
body
Element
div
•id Attribute
•class Attribute
Element
h1
h2
Element
Portal web
Text
span
•class
Attribute
Element
viajar.com
Text
para tus aventuras
Text
Bienvenidos
Text

Hay algunas etiquetas a las que se puede acceder de forma directa, desde el objeto **document**.

Acceso a los enlaces de una página

El objeto **`document.link`** nos da acceso a todos los enlaces de la página. El resultado es un array que contiene todos los enlaces en el orden que aparecen. Si queremos ver el texto que aparece, usaos la propiedad innerHTML.

Ejemplo: `<a href="http://www.google.es">Enlace a Buscador</a>`

```
enlace = document.links[0].href;
```

La variable `enlace` tendrá como valor `http://www.google.es`

```
texto = document.links[0].innerHTML;
```

La variable `texto` tendrá como valor “Enlace a Buscador”

Si queremos cambiar la URL haremos:

```
document.links[0].href = “http://www.youtube.com”;
Document.link[0].innerHTML = “YouTube”
```

Acceso a imágenes

Se hace de forma parecida a como se accede a los enlaces. El objeto **`document.images`** nos da acceso a un array que contiene todas las imágenes de la página.

Para ver o guardar la imágenes en el navegador utilizamos el atributo **`src`**.

Ejemplo:
```
<a href="http://www.google.es">Enlace a Buscador</a>

imagen = document.images[0].sr;
```

La variable `imagen` guarda la ruta de acceso a la imagen.

Podemos cambiar la imagen inicial, por otra cambiando su ruta:

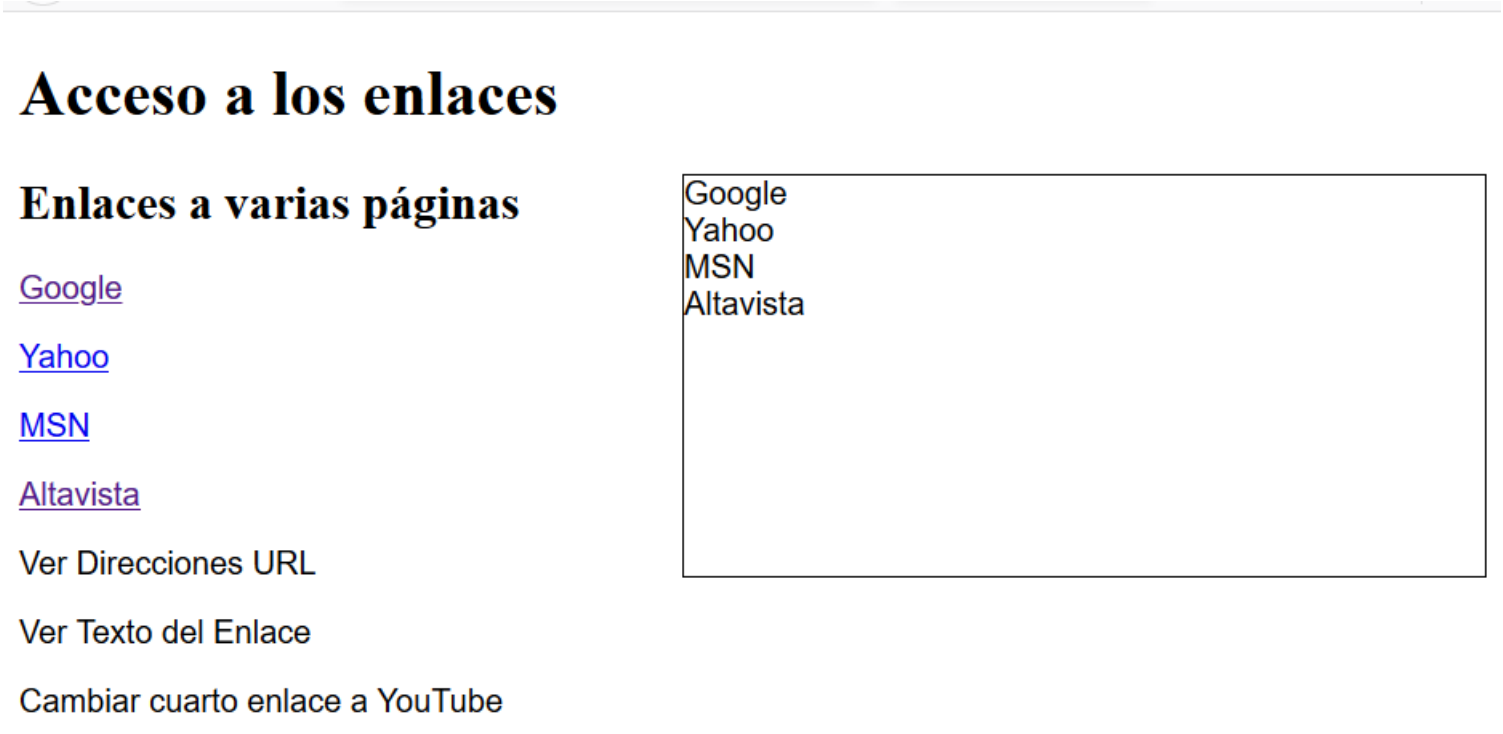
```
document.images[0].src = "objetos/imagen2.jpg";
```

Ejercicio_04 acceso a enlaces

Ejercicio_04 acceso a enlaces

Crear un documento HTML que a la derecha tenga un `div` donde se mostrará la información sobre las direcciones URL, los textos de los enlaces, cuando se haga clic en el texto correspondiente.

A la izquierda tendremos los enlaces a diferentes URL, cuando hagamos clic nos abrirá la pagina correspondiente. Y en la ultima opción, tendremos cuando se haga clic sobre ella, cambiar el enlace de Altavista a youtube.



Trabajar con ventanas

Ventanas de alerta:

Son aquellas en las que sale un mensaje alertando al usuario sobre algún aspecto de la pagina. estas ventanas son **`alert()`** y **`prompt()`**, pero también tenemos **`confirm()`**, la cual nos muestro la información que le pasamos entre paréntesis y dos botones: **Aceptar** y **Cancelar**.

Historial de páginas:

Mediante JavaScript también podemos acceder al historial de paginas visitadas anteriormente, igual que con los botones adelante y atrás del navegador. Para ello utilizaremos el método **`history.go()`**, donde entre paréntesis pondremos el numero de paginas que queremos avanzar o retroceder.

```
<p onclick = "history.go(-1)">Ir  a la pagina anterior </p>
```

Trabajar con ventanas

Ventanas emergentes:

El método **`open()`** nos permite abrir ventanas, pero a veces queremos que estas ventas tengan unas características especiales y además podemos decidir cuando se deben abrir, al cargar una pagina, cuando se produzca un evento, etc. Para ello utilizaremos el método **`open()`**, al cual le pasaremos tres argumentos: UR, nombre y propiedades.

```
Open("URL", "nombre", "propiedades")
```

El argumento propiedades es opcional,, si no se pone, se abrirá como un enlace normal y si se ponen se pondrán las propiedades entre comillas y separadas por comas.

```
<span
onclick="open(`http://www.google.es`,´Google´,´toolbars=yes,
width=650, height=250,top=100,left=200´)">Abrir Goolge</span>
```

Además de las propiedades que nos permiten modificar las barras, el tamaño, etc. de las ventanas, hay más propiedades que nos permiten ver o modificar algunas partes de las ventanas. Algunas de estas propiedades dependen de toros objetos independientes a **`window`**, como pueden ser **`document`**, **`screen`**, **`navigator`**.

Un ejemplo de estas propiedades son:

`document.bgColor`: acceso o modificar color de fondo de la página.

`document.fgColor`: acceso o modificar color del texto de la página.

`location`: acceder a la URL.

`document.title`: acceso o modificar el titulo de la página.

`documnet.lastModified`: acceso a la fecha de la ultima modificación.

`navigator.userAgent`: acceso a la información de l navegador.

`screen.width:`. acceso al ancho de resolución de la pantalla.

`screen.height`: acceso al alto de resolución de la pantalla.

Ejercicio_05 acceso a enlaces

Crear una página en la que vamos a utilizar las propiedades y los efectos que pueden producir:

Propiedades de la página. Entre guiones va escrita la propiedad usada. Pulsa para:

Color de fondo amarillo - document.bgColor -

Volver a color de fondo blanco - document.bgColor -

Color de texto azul - document.fgColor -

Volver a color de texto negro - document.fgColor -

Ver el dominio de la página - location -

Ver el título de la página - document.title -

Cambiar el título de la página - document.title -

Fecha de la última modificación - document.lastModified

Navegador - navigator.userAgent -

Resolución -screen.width - screen.height -

2.- PROGRAMACIÓN DE EVENTOS

- Los eventos se utilizan para relacionar la interacción del usuario con las acciones de DOM vistas hasta ahora.
- Una condición para que se genere la estructura de árbol es que la página se cargue completamente.
- Utilizaremos el evento `onload` el cuál ocurre cuando la página se ha cargado por completo.

```
<html>
  <head>
    <title>Titulo DOM</title>
  </head>
  <body onload="alert("Pagina cargada completamente");" >
    <p>Primer párrafo</p>
  </body>
</html>
```

2.- PROGRAMACIÓN DE EVENTOS

```
<html>
 <head><title>Titulo DOM</title>
  <script type="text/javascript">
    window.onload = function(){
      alert('Página cargada completamente');
       //avisamos que la pagina esta cargada
       console.log(document.getElementById("p1"
       ).childNodes[0].nodeValue);
       //accedemos mediante DOM a sus nodos
       }
  </script>
 </head>
 <body><p id="p1">Primer parrafo</p></body>
</html>
```

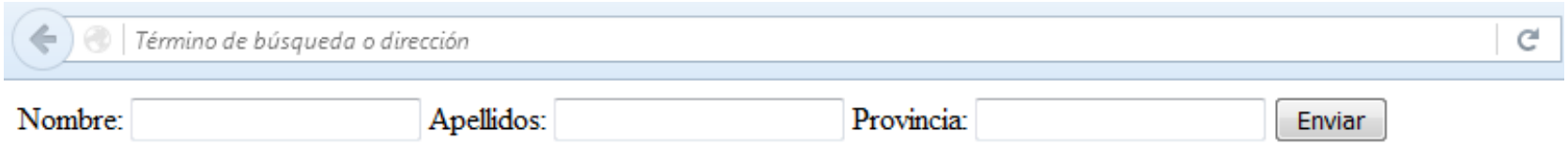
2.- PROGRAMACIÓN DE EVENTOS

- **Actuar sobre el DOM al desencadenarse eventos:**

```
<html>
  <head><title>Titulo DOM</title>
    <script type="text/javascript">
     function ratonEncima(){
      document.getElementsByTagName("div")
          [0].childNodes[0].nodeValue="EL RATON ESTA
ENCIMA";
     }
     function ratonFuera(){
      document.getElementsByTagName("div")
          [0].childNodes[0].nodeValue="NO ESTA EL RATON
ENCIMA";
     }
    </script>
  </head>
  <body><div onmouseover="ratonEncima();"
onmouseout="ratonFuera();"> VALOR POR DEFECTO </div>
  </body>
</html>
```

Actividad 4

Mueve el foco de un campo de texto a otro, dentro de un formulario, al pulsar la tecla ENTER dentro de cada campo. Para poder implementar el desplazamiento entre nodos nos ayudaremos de la propiedad `nextSibling`:



3.- DIFERENCIAS EN LAS IMPLEMENTACIONES DEL MODELO

- Una de las principales dificultades a la hora de utilizar DOM es que no todos los navegadores hacen una misma interpretación.
- W3C hace unas recomendaciones y crea unos estándares, que los navegadores tienen que ir adoptando.
- La guerra entre navegadores a la hora de generar sus propios estándares, ha generado muchos problemas a los programadores de páginas web.
- Todos los navegadores utilizan JavaScript como lenguaje de programación en el entorno cliente, pero los objetos y eventos no se comportan de la misma forma.

3.- DIFERENCIAS EN LAS IMPLEMENTACIONES DEL MODELO

- **Adaptaciones de código para diferentes navegadores:**
  - Pueden ocasionar problemas de interpretación.
  - Complican la actualización futura de la página.
  - Generan la necesidad de modificar el código implementado.
  - W3C para estandarizar la Web creo el modelo DOM único.
  - Internet Explorer ha añadido su propia extensión de DOM, con lo cual genera problemas de interoperabilidad entre los navegadores web.

3.- DIFERENCIAS EN LAS IMPLEMENTACIONES DEL MODELO

- Adaptaciones que se pueden realizar para mejorar la compatibilidad:
  - El navegador Internet Explorer 7, no soporta las constantes predefinidas por lo que si vamos a programar una aplicación web deberemos crear de forma explícita las constantes predefinidas.

```
alert(Node.DOCUMENT_NODE); // Devolvería 9
alert(Node.ELEMENT_NODE); // Devolvería 1
alert(Node.ATTRIBUTE_NODE); // Devolvería 2
```

3.- DIFERENCIAS EN LAS IMPLEMENTACIONES DEL MODELO

- Crear de forma explícita las constantes predefinidas:

```
if(typeof Node == "undefined")
 { var Node = {
      ELEMENT_NODE: 1,
      ATTRIBUTE_NODE: 2,
      TEXT_NODE: 3,
      CDATA_SECTION_NODE: 4,
      ENTITY_REFERENCE_NODE: 5,
      ENTITY_NODE: 6,
      PROCESSING_INSTRUCTION_NODE: 7,
      COMMENT_NODE: 8,
      DOCUMENT_NODE: 9,
      DOCUMENT_TYPE_NODE: 10,
      DOCUMENT_FRAGMENT_NODE: 11,
      NOTATION_NODE: 12
     };
  }
```

4.- SOLUCIÓN CROSS BROWSER

- Para solucionar el problema del desarrollo de aplicaciones web en diferentes navegadores nace *cross-browser.*
- *Cross-browser* tiene la intención de visualizar una página o aplicación web exactamente igual en todos los navegadores.
- Para conseguir este objetivo surgen utilidades que permiten unificar los eventos y sus propiedades.
- Las soluciones cross-browser buscan que las incompatibilidades o diferencias entre los navegadores no se aprecien por el cliente y la pagina web funcione en cualquier navegador sin producir fallos.

4.- SOLUCIÓN CROSS BROWSER

- Para el uso de las utilidades *cross-browser* es necesario implementar funciones que habitualmente vienen definidas en librerías.
- Es necesario comprobar el navegador y tomar una decisión u otra en función de la respuesta.

```
if (navigator.appName.indexOf("Explorer") != -1) {
  // codigo para Internet Explorer
}else{
  // codigo para Firefox, Chrome, Opera, Safari
}
```

4.- SOLUCIÓN CROSS BROWSER

Ejemplo:

```
if (navigator.appName.indexOf("Explorer") != -1)
 {
  // Es un navegador IEInternet Explorer


    this.parentNode.childNodes[i]=this.parentNode.childNodes[i].c
    lassName.replace(/\dseleccionado\b/,``);
  }
   else
    this.parentNode.childNodes[i].classList.remove("seleccionado"
    );
```

En JavaScript podemos ejecutar boques de código dependiendo de una condición determinada. Esto lo vamos a aprovechar con cross-browser para comprobar el tipo de navegador que estemos utilizando y en función de eso ejecutar el código compatible con dicho navegador.

FIN