**Introducción.**

Hasta este momento todos los ejemplos y ejercicios hemos podido hacerlos en la consola, pero a partir de este momento los ejercicios y ejemplos van a estar relacionados con la interacción o el manejo de elementos visuales en páginas web y por tanto vamos a necesitar tener dichos elementos definidos y cargados en el navegador.

**Paso primero: Plantilla HTML.**

En muchos ejercicios necesitaremos una página HTML. Como punto de partida utilizaremos esta plantilla HTML, que define la estructura de un documento HTML completamente vacío, por tanto escribid este código en un documento HTML en blanco y guardadla en vuestros equipos para usarla como punto de partida en los ejercicios donde se requiera una página HTML.

<!DOCTYPE html>

<html>

<head>

<title>Page Title</title>

<meta charset="utf8" />

</head>

<body>

</body>

</html>

**Paso segundo: Añadiendo contenido**

Añadir en la sección del <body> el contenido HTML necesario para hacer el ejercicio.

<!DOCTYPE html>

<html>

<head>

<title>Page Title</title>

<meta charset="utf8" />

</head>

<body>

<p>Parrafo de prueba</p>

</body>

</html>

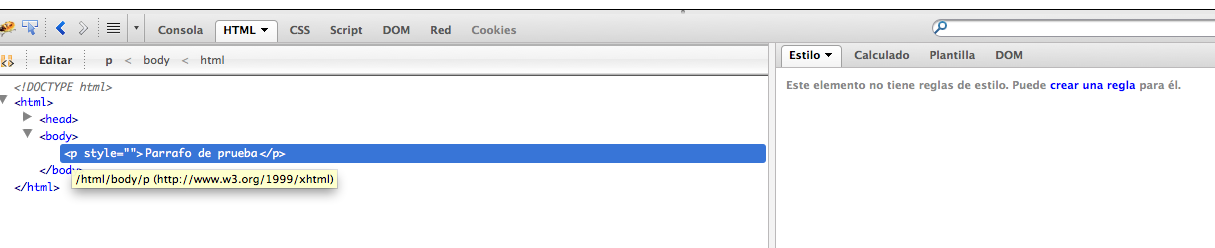
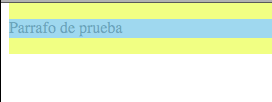
Guardar como un nuevo documento HTML para no modificar la plantilla y mantenerla siempre como punto de partida de nuevos ejercicios y abrir el nuevo documento creado con el navegador Mozilla Firefox.

**Paso tecero: La herramienta Firebug**

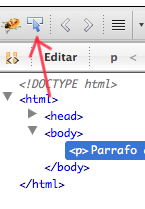
Hasta ahora firebug nos ha servido como entorno de pruebas JavaScript, pero ahora veremos algunas de las múltiples capacidades que tiene para el desarrollo web.

**1. Inspección HTML**

En la pestaña HTML de Firebug podemos ver el código HTML de la página que el navegador está mostrando y colocando el ratón encima de cualquier elemento podemos verlo resaltado en amarillo y azul en el navegador.

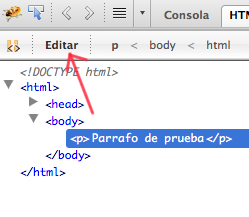


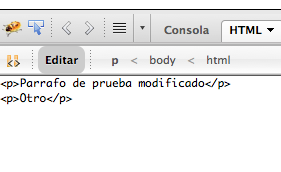
Haciendo click sobre un elemento en la vista de la estructura de Firebug ese elemento quedará seleccionado

Otra forma de inspeccionar los elementos HTML de una página, que será muy útil cuando trabajemos con páginas cuya estructura sea algo más complicada es poder hacer click sobre un elemento de la página en la ventana del navegador, y Firebug nos lo mostrará seleccionado en la vista de su estructura. Para ello seleccionar el botón correspondiente situado en la barra de herramientas de firebug.

En este momento firebug pasa a modo ‘selección con el ratón’ y si colocamos el ratón encima de cualquier elemento de la página veremos un recuadro de selección, y a la vez, en la vista de la estructura HTML de firebug también veremos seleccionado el elemento correspondiente. Si en este modo hacemos click sobre algún elemento, el elemento queda seleccionado en firebug y se desactiva el modo ‘selección con el ratón’. En modo ‘selección con el ratón’, cualquier click se pasará directamente a firebug y no tendrá efecto. Es decir, si queremos inspeccionar un enlace y hacemos click sobre el enlace, el elemento <a> correspondiente se seleccionará en firebug y no se navegará a la página enlazada.

* 1. **Edición HTML**

Firebug nos permite editar el contenido HTML del documento y visualizar los cambios inmediatamente. Para ello, seleccionamos el elemento del documento que queramos modificar y hacemos click en el botón Editar.

A continuación se abrirá un pequeño editor de texto donde podemos modificar el contenido del elemento seleccionado o incluso añadir un nuevo hermano suyo.

Modificar el contenido del párrafo y, por ejemplo, añadir un nuevo párrafo. A medida que vais escribiendo las modificaciones, podréis comprobar como el navegador va reflejando los cambios en tiempo real. Para finalizar la edición basta con pulsar Editar nuevamente y volveremos a la vista con la estructura del contenido de la página que incluye las modificaciones realizadas.

Es importante saber que estos cambios se están realizando sobre una copia temporal que guarda el navegador de nuestro documento HTML, no sobre el archivo HTML que inicialmente abrimos en el navegador. Esto ocurre con ésta y otras modificaciones que Firebug nos permite hacer, y que veremos a continuación, como modificaciones de las reglas CSS, y por tanto, siempre que hagamos algún cambio en Firebug y lo queramos conservar, tenemos que asegurarnos de escribirlo en el archivo original.

**2. Inspección CSS**

Con la pestaña HTML de Firebug activa, al colocar el ratón encima del párrafo vemos que el párrafo se ilumina en la ventana del navegador con dos colores, amarillo y azul. El área coloreada en amarillo representa el margen (propiedad CSS margin) que ese elemento tiene aplicada y el área coloreada de azul representa el área que ocupa nuestro elemento, que será una suma del contenido del elemento más su padding (propiedad CSS padding).

Como en este documento no hemos definido ningún estilo CSS propio, éstos estilos que se le están aplicando son definidor por defecto por el navegador.

**2.1 Edición CSS**

Si seleccionamos un elemento y nos fijamos en la parte derecha de esta vista de Firebug, en la pestaña estilo podemos definir estilos que se aplicarán sobre el elemento que tengamos seleccionado y podremos ver el resultado inmediatamente.

Vamos a añadir una nueva regla CSS para el párrafo de prueba. Para eso hacemos click en el texto resaltado ‘crear una regla’ y escribimos: ‘margin-left: 10px’.

Inmediatamente podemos ver cómo el párrafo se mueve a la derecha 10 px y en la vista de estructura HTML vemos que se ha añadido un atributo ‘style’ con el valor de la nueva propiedad CSS al párrafo.

**3. Ejecución de JavaScript**

En la pestaña Consola de Firebug podremos ejecutar JavaScript que utilice los elementos de la página que estamos visualizando igual que hemos hecho hasta ahora.

Escribir document.getElementsByTagName(‘p’); y ejecutadlo para probar la selección de elementos en un documento que estamos visualizando. En el panel de la salida de la consola veremos el resultado de la expresión como un array que contiene todos los elementos de tipo <p> que hay en la página que se está visualizando en el navegador.

Cada elemento dentro del array es un enlace al elemento correspondiente en la vista de estructura HTML de Firebug. Y si hacemos click en el espacio comprendido entre los corchetes que delimitan el array y los elementos de dentro del array podremos ver todas las propiedades tanto del array como de los objetos contenidos con detalle. Echadle un vistazo para ver las propiedades que tienen. Algunas ya las conocemos, otras no.

**4. JavaScript contenido en el documento HTML**

Como ya todos sabemos, el uso más extendido de JavaScript es incluyéndolo en documentos HTML para mejorar sus características y Firebug es una herramienta muy buena para programadores JavaScript en documentos HTML.

Lo primero que debemos hacer es añadir código JavaScript a nuestro documento HTML, por tanto vamos a modificar nuestro documento para añadirle código JavaScript. Como queremos ser buenos programadores, vamos a separar siempre el código JavaScript del contenido del documento (HTML) y del código de presentación (CSS).

<!DOCTYPE html>

<html>

<head>

<title>Page Title</title>

<meta charset="utf8" />

<script>

var p = document.getElementsByTagName('p');

console.log("script ejecutado");

</script>

</head>

<body>

<p>Parrafo de prueba</p>

</body>

</html>

Guardamos el documento y lo recargamos en el navegador (bien pulsando el botón de recargar página situado en la barra de direcciones, pulsando F5 o pulsando Ctrl + r).

Si abrimos la herramienta Firebug y vamos a la pestaña Consola, podemos ver como la sentencia console.log ha escrito en el área de salida. Con esto tenemos la primera forma de depurar código JavaScript que estemos programando para un documento HTML: imprimiendo valores en la consola.

Vamos a inspeccionar el valor de la variable p imprimiendo su valor en la consola: modificamos la sentencia console.log así: console.log("p contiene: " + p);

La información obtenida nos indica que la variable contiene algo así como un Array, pero no nos da mucha más información.

**4.1 Depurando JavaScript**

Firebug tiene un depurador de JavaScript que podemos usar para depurar nuestros programas. Así podremos ver en detalle qué contiene la variable p.

Para depurar código JavaScript debemos abrir la pestaña Script de Firebug y ahí veremos el código de nuestro documento HTML. Si nos fijamos en la parte izquierda podemos ver los números de línea en dos colores. Un color gris en la mayoría de los números y color negro en otros. Esto nos indica dónde podemos colocar puntos de ruptura para que el navegador se detenga y nos permita depurar el programa. Podemos colocar un punto de ruptura en cualquier lado, pero solo será efectivo si está en una línea cuyo número está en color negro.

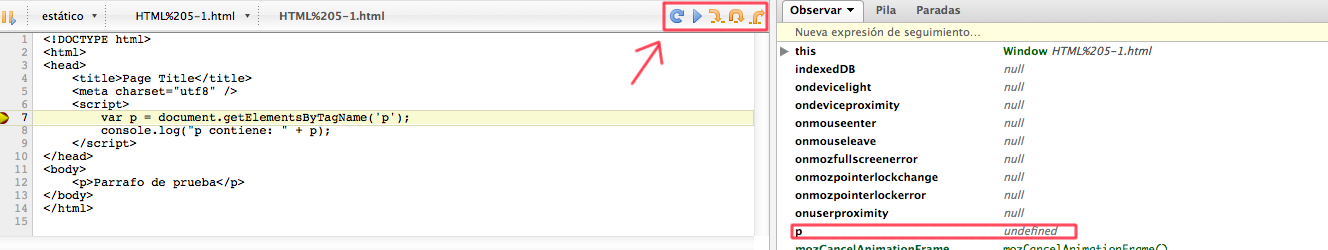
:::::::Desktop:Captura de pantalla 2012-10-22 a la(s) 20.45.20.png

Para colocar un punto de ruptura simplemente hacemos click sobre el número de línea y aparecerá un punto rojo a la izquierda indicando que el punto de ruptura se ha configurado correctamente.

Ahora bastará con recargar la página en el navegador y el Firebug lo detendrá en ese punto para poder depurar.

Mientras estemos depurando la línea que se va a ejecutar aparece resaltada y a la derecha podemos ver todas las variables que están definidas y su valor en este punto.

Para controlar la depuración tenemos 5 botones en la parte superior derecha de la ventana de depuración. Colocando el ratón encima de cada botón se nos mostrará, a los pocos segundos, una descripción de qué hace cada uno.



Para poder inspeccionar el valor de p necesitamos que la sentencia que está resaltada se ejecute, porque, como podemos ver en la parte derecha, la variable p existe pero no tiene valor. ¿Recordáis por qué una variable existe antes de haberse ejecutado la sentencia que la define?

Si pulsamos el botón de Saltar, para pasar a la línea siguiente y poder inspeccionar la variable p, el depurador no hace lo que esperamos, que sería pararse en la línea 8 sino que continúa la ejecución hasta el final. Esto ocurre porque no existen más líneas que haya detectado como ‘depurables’ (marcadas con su número de línea en negro). Por tanto, para poder depurar este programa necesitamos añadir una nueva línea depurable.

Cuando esto ocurra, por ejemplo, añadiremos la declaración de una variable cualquiera:

<!DOCTYPE html>

<html>

<head>

<title>Page Title</title>

<meta charset="utf8" />

<script>

var p = document.getElementsByTagName('p');

var a;

console.log("p contiene: " + p);

</script>

</head>

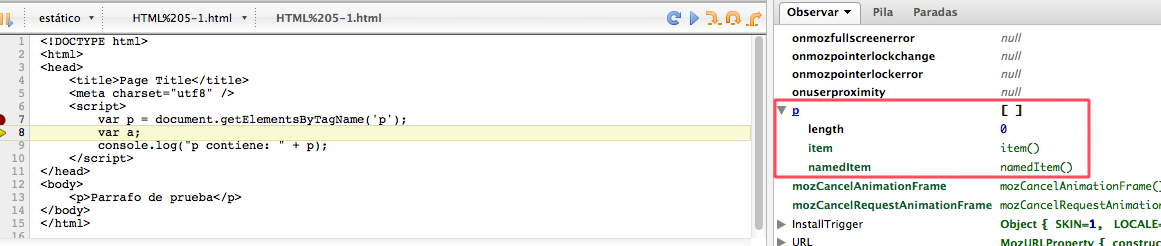
<body>

<p>Parrafo de prueba</p>

</body>

</html>

Con este cambio, Firebug detectará una nueva línea depurable y el depurador no continuará, permitiéndonos inspeccionar la variable p. Salvamos el documento y lo recargamos en el navegador y cuando el depurador se pare, pulsamos Saltar y ahora sí se parará en la siguiente línea y podremos ver el contenido de p.



¿Qué veis raro en este momento? ¡La sentencia document.getElementsByTagName ha devuelto un array vacío y en el body de nuestro documento había un elemento que debería haber incluido!

Esto no es un error. Lo que ocurre es que, como ya hemos dicho, el navegador ejecuta secuencialmente lo que va encontrando en el documento de principio a fin, y el script está por delante del body, por tanto, en este punto, aún no se ha cargado nada del contenido del documento. Podemos comprobar que esto es cierto, porque, con el depurador parado en la línea 8, en la pantalla del navegador no aparece nuestro párrafo.

Para solucionar este inconveniente tenemos dos opciones:

1. Colocar el script al final del body

<!DOCTYPE html>

<html>

<head>

<title>Page Title</title>

<meta charset="utf8" />

</head>

<body>

<p>Parrafo de prueba</p>

<script>

var p = document.getElementsByTagName('p');

var a;

console.log("p contiene: " + p);

</script>

</body>

</html>

1. Utilizar un evento del navegador, que nos indicará cuando el documento está cargado y listo para su manejo por JavaScript. Esta segunda opción es la mejor y, aunque aún no hayamos estudiado los eventos, es la que vamos a utilizar.

<!DOCTYPE html>

<html>

<head>

<title>Page Title</title>

<meta charset="utf8" />

<script>

function loaded() {

var p = document.getElementsByTagName('p');

console.log("p contiene: " + p);

}

</script>

</head>

<body onload="loaded()">

<p>Parrafo de prueba</p>

</body>

</html>

Por el momento es suficiente con saber que el navegador, cuando todo su contenido está cargado y listo para ser manipulado por JavaScript lanza un evento “load” y, debido al atributo que hemos añadido al body, cuando esto ocurra, ejecutará la función loaded(), que hemos definido, por tanto, para todos los ejemplos de aquí en adelante, debemos colocar nuestro código JavaScript dentro de la función loaded.

Además, por encontrarse dentro del cuerpo de una función, Firebug determina la sentencia console.log como ‘depurable’ y nos evitamos tener que definir una variable simplemente para poder parar el depurador donde queremos.

Probad este código y depuradlo con Firebug y probad también a seleccionar el párrafo con los otros métodos de selección de elementos que conocemos:

* document.getElementById
* document.getElementsByName
* document.getElementsByClassName
* document.querySelector
* document.querySelectorAll

Para poder probar el resto de elementos, tendréis que añadirle al párrafo los atributos necesarios.

Fin.