
JS
HTML
CSS


HTML
5

Table of Contents

Presentación

Este curso es **gratuito** y lo puedes realizar **incluso si no sabes nada de programación**, sólo hace falta tener ganas de aprender. A pesar de esto, también puede serte útil si ya sabes HTML pero quieres perfeccionar o refrescar tus conocimientos sobre HTML5, CSS3 o JavaScript.

A lo largo del mismo te voy a:

- Enseñar los elementos más comunes de HTML5, CSS3 y JavaScript, osea los que usaremos el día día.
- Mostrar dónde podrás resolver tus dudas cuando tengas problemas.
- Explicar cómo trabajar con algunas de las herramientas que usan los profesionales del mundo real como son Github o las herramientas para desarrolladores de Google Chrome.

Es **importante** saber que en el mundo de la programación hay muchas formas de resolver un mismo problema, y que todas ellas pueden ser igualmente válidas. Dicho esto, quiero aclarar que **el objetivo del curso no es** aprender todas y cada una de las formas de resolver un problema (esto se va aprendiendo con años de práctica), y es por ello que **no entraré en profundidad en todos y cada uno de los elementos y características** de cada lenguaje.

Las definiciones estarán simplificadas, esto está hecho a conciencia ya que como decía antes, no he querido asumir que tengas ningún conocimiento previo sobre desarrollo web. Por eso, para facilitarte la comprensión y evitar distraerte del objetivo del curso, te explicaré conceptos en muchas ocasiones que no serán 100% precisos pero que sí correctos.

En muchas ocasiones encontrarás **enlaces a la Wikipedia**, he elegido hacerlo así por varias razones:

1. Usa un lenguaje bastante coloquial
2. Incluye enlaces a las palabras más complejas
3. Son definiciones consensuadas en las discusiones

A pesar de esto incluiré enlaces al W3C con las definiciones formales, aunque puede que estas sean más difíciles de entender.

Así que **no tomes todas las definiciones al pie de la letra**, tómalas como definiciones lo suficiente buenas como para ayudarte a entender el contexto. De todos modos, siempre que se dé este caso te añadiré un enlace a un recurso con más información o añadiré un superíndice con aclaraciones al final de la página.

El objetivo del curso es dotarte de una buena base que te permita sentirte lo suficientemente seguro[1] para afrontar cualquier proyecto y así seguir aprendiendo. Para ello nos centraremos solamente en los conocimientos y herramientas más importantes de un desarrollador front-end[2].

Formato

Este es el material escrito del Curso de HTML5, CSS3 y JS desde cero, **si quieres puedes estudiarte este curso por tu cuenta, o inscribirte al formato MOOC a través de la web** para que te avise en las próximas ediciones que organice.

Estas ediciones que se celebran periódicamente consisten en ver una serie de vídeo-tutoriales y hacer unos ejercicios que te enviaré semanalmente. La carga de trabajo pretende ser menor a 3 horas semanales e incluyen una sesión online (en grupo) de 1h a la semana para resolver dudas.

El formato MOOC tendrá una **duración aproximada de 6 semanas**.

Snippets interactivos

A lo largo de todo el libro utilizaré ejemplos de código (*snippets*) interactivos para facilitar la compresión y la interacción con el código HTML, CSS y JavaScript. Estos snippets están organizados por lección y número de snippet, por lo que a lo largo de las lecciones haré referencia a ellos y así, en caso de que algo no te haya quedado claro, puedas consultarlos antes de continuar con la lección.

Además de como apoyo al libro, tienen un segundo propósito: servirte de "libreta" en el futuro para tener una lista de ejemplos para tus propios desarrollos.

Dudas, ideas, sugerencias, y contribuciones

Además de resolver dudas en la sesiones online semanales, podrás realizar cualquier pregunta, o aportar cualquier idea o sugerencia en cualquier momento a través del apartado de issues del proyecto en Github.

Recuerda, que no te de vergüenza preguntar cualquier duda que te surja, por simple que te parezca, si te ha surgido a ti, ten por seguro que le surgirá a más personas. **Así me ayudarás también saber qué aclaraciones añadir o qué mejoras puedo hacer al contenido del curso** para facilitarles el aprendizaje a otras personas que vengan detrás.

Además si consigues terminar el curso **sin ninguna duda**, el día que estés haciendo un proyecto y algo no te funcione sabrás mucho mejor por dónde empezar a buscar los errores.

Y si te sientes cómodo usando Github, puedes hacer un pull request al repositorio resolviendo cualquier error que encuentres.

Autor(es)

Mi nombre es Raúl Jiménez Ortega, he creado este curso basándome principalmente en mis años de experiencia haciendo webs (llevo desde 1999) y dando formación (desde 2009), pero además he revisado las mejores fuentes que conozco para seleccionar lo mejor de cada curso/tutorial que he encontrado en estos años en Internet.

Aunque inicialmente me he lanzado sólo a crear este curso, mi objetivo es animar a que otros profesionales aporten su granito de arena para mejorar el contenido y mantener este curso actualizado durante mucho tiempo, **es por eso que he decidido liberar este libro licenciándolo como Creative Commons (CC-BY-NC-SA 4.0 International)**.

En el siguiente apartado veremos las diferentes maneras de contribuir a mejorar el curso. Por supuesto, las personas que contribuyan tendrán su reconocimiento no sólo en Github sino también dentro de los contenidos del libro.

Motivación - ¿Por qué?

¿Por qué otro curso de HTML/HTML5?, pues las razones que me han llevado a lanzarme a esta aventura son:

1. **Compartir los trucos y consejos**: a lo largo de estos años he ido aprendiendo muchos trucos y muchas lecciones sobre cómo ser más productivo y me gustaría compartirlos contigo.
2. **Asentar una buena base**: después de revisar muchos cursos (algunos de ellos muy buenos), tengo la sensación de que ninguno explica todos los conceptos necesarios para crear una buena base que te permita sentirte cómodo a la hora de continuar creciendo por tu cuenta.
3. **Creo que los tiempos han cambiado**: y creo que hay muchísimas personas con mucho potencial a las que saber HTML, CSS y JavaScript les "daría alas" y les abrirían muchas puertas profesionales.
4. **Disfruto compartiendo lo que sé y ayudando a los demás**: siempre quise montar una empresa para crear algún producto que mejorase aunque fuese un poco la vida de miles/millones de personas, pero después de 6 años me he dado cuenta de que

también es posible cambiar la vida de muchas personas enseñándoles lo que sé.

Todo esto ha sido más que suficiente para que decida a dedicar **muchas horas** (y cariño) de mi tiempo libre para ir creando poco a poco los contenido de este curso.

Aclaraciones:
1. A lo largo de todo el libro usaré el masculino por facilitar la lectura evitando los: "seguro/a", o los "segur@", espero que nadie se ofenda ;-P.
2. Un programador front-end no es más que un programador que domina HTML, CSS y JavaScript.

Contribuciones

Cualquier persona puede contribuir de diferentes formas:

1. Ayudando a mejorar los contenidos del libro o los ejercicios.
2. Colaborando en mejorar la página web.
3. Ayudando con las aplicaciones móviles.

Para facilitar la comunicación entre todas las personas que quieran contribuir se ha creado un canal en Gitter. A continuación explicamos cómo contribuir en cada uno de los apartados mencionados anteriormente.

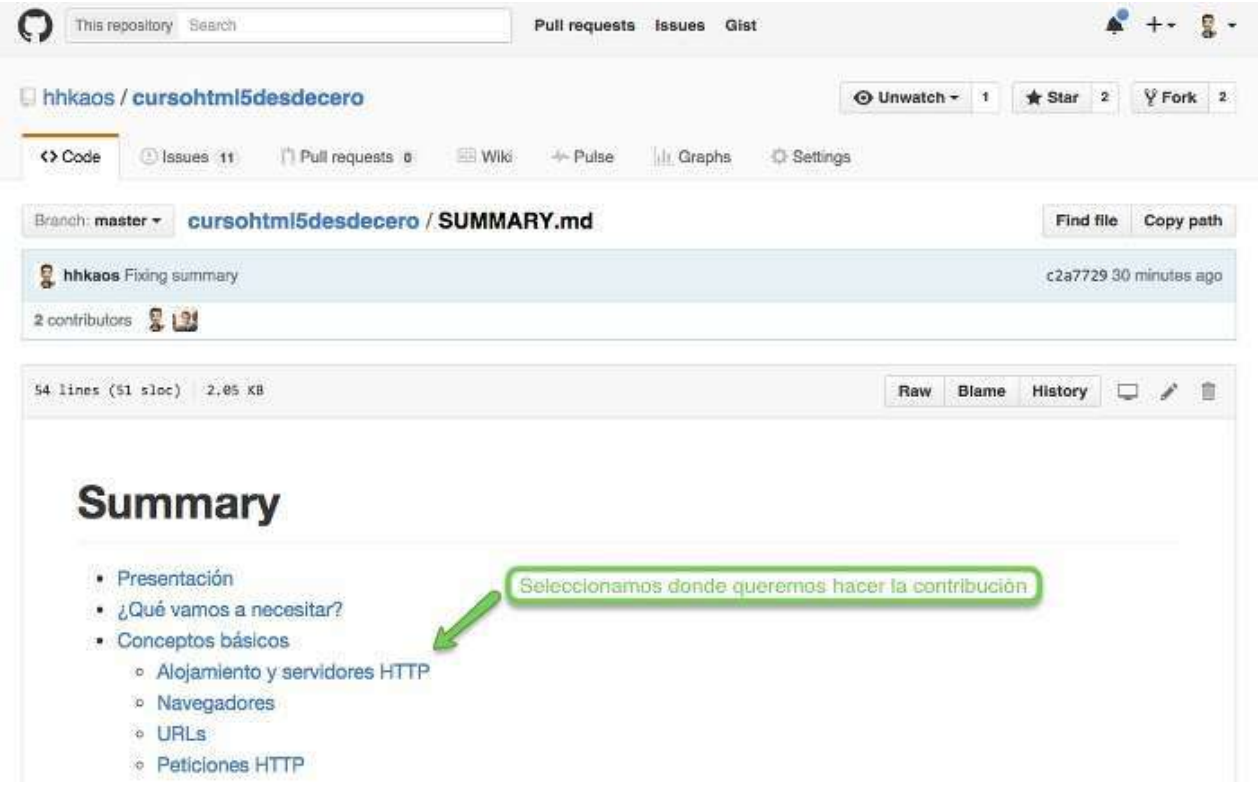
Contenidos del curso

Libro

Estado de las tareas: Waffle | Github

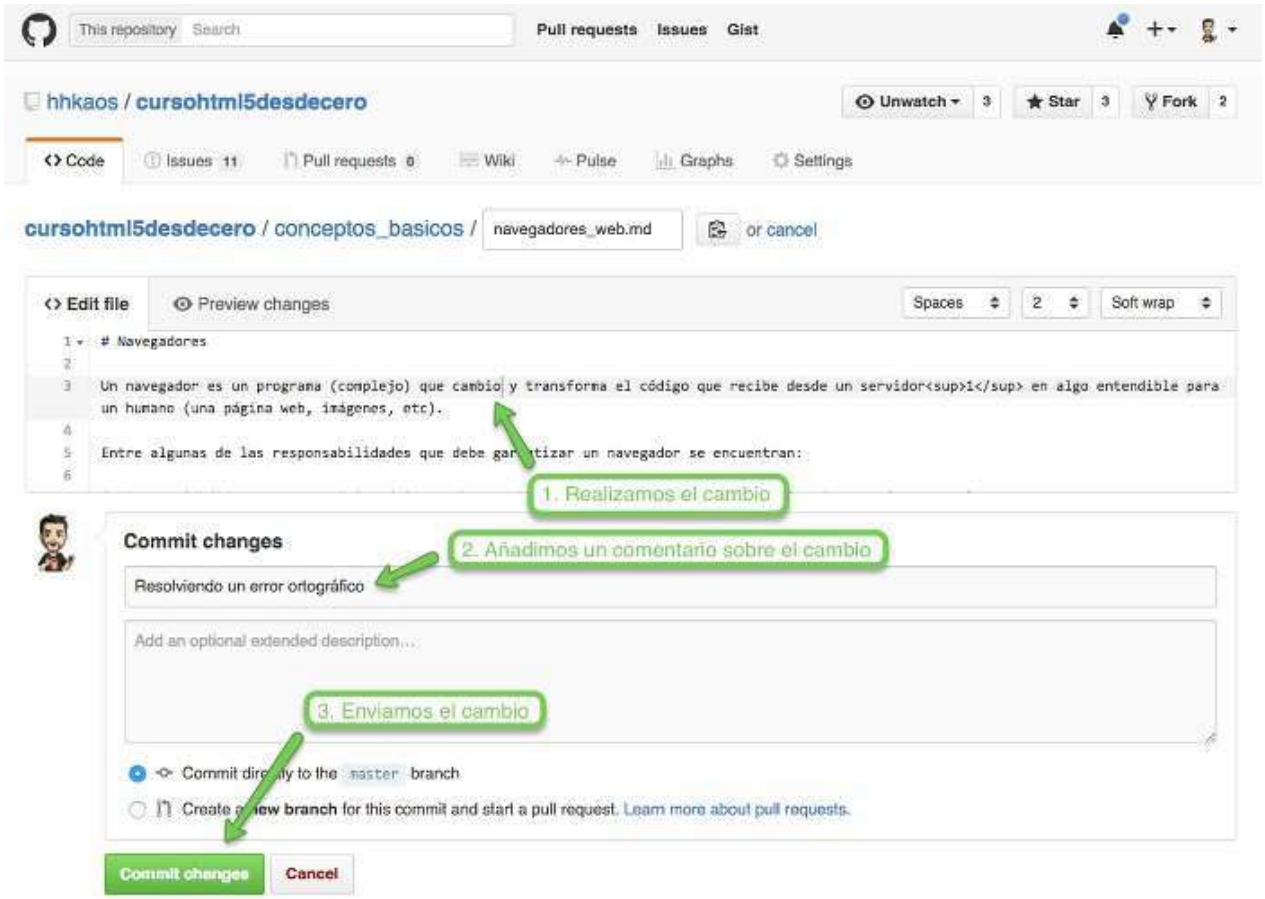
Cualquier persona puede ayudar a introducir mejoras, corregir errores tipográficos, gramaticales, etc.

Los pasos son muy sencillos, primero accedemos a la página con el índice del libro en Github y seguimos los siguientes pasos:



Para "poder editar" una página (aclaración: *en realidad lo que estarás haciendo es enviarme una propuesta de mejora que tendré que revisar y aceptar*), **hace falta tener una cuenta y estar identificado en Github**.

Por último, añadir que los contenidos están escritos en el lenguaje de marcas *Markdown* que veremos en la próxima lección.



Ejercicios

Estado de las tareas: Waffle | Github

Por definir.

Página web

Estado de las tareas: Waffle | Github

Por definir.

Aplicaciones móviles

Estado de las tareas: Waffle | Github

Por definir.

Markdown

Esta lección la he creado para las personas que estén pensando en contribuir con los contenido del curso y no saben **Markdown**

Markdown es un lenguaje de marcas (como HTML) bastante sencillo, aquí te dejo un tutorial con ejemplos por si te hiciese falta.

WIP (simplicar el tutorial aquí)

¿Qué vamos a necesitar?

Un **navegador y un editor de código** (o editor de texto), en este curso usaremos Google Chrome porque incluye ambas herramientas (navegador y editor), y porque además **considero que es más didáctico** y te ayudará a asimilar mejor los conceptos.

Es habitual encontrar profesionales que usen otros programas como: Sublime Text, Atom.io, Brackets, WebStorm, IntelliJ, ... pero nosotros de momento no usaremos ninguno de ellos.

Nota: Es posible que hayas oído hablar de Dreamweaver, Frontpage, Aptana, etc. Personalmente no te recomiendo ninguno de ellos para aprender porque incluyen botones que introducen código, cosa que muchas veces nos incita a introducir código innecesario o código que no sabemos qué significa, y por tanto que no sabremos arreglar en caso de que contenga algún fallo.

El tercero no lo recomiendo porque además de ser un programa que consume bastantes recursos, considero que como decimos habitualmente en España: *es matar moscas a cañonazos*.

Conceptos básicos

En esta primera lección vamos a ver qué son y cómo funcionan:

- Los servidores web y HTTP
- Los navegadores
- Las URLs
- Peticiones HTTP

Por tanto empezaremos por aprender los **conceptos fundamentales** a la hora de entender el funcionamiento de una página web para adquirir una base que nos permita entender de dónde pueden venir los errores que cometamos en el futuro.

Es importante conocer e interiorizar bien estos conceptos ya que los usaremos y nos los encontraremos continuamente tanto en este curso como en cualquier otro recurso de Internet.

Si crees que ya dominas estos conceptos puedes probar a hacer el ejercicio tipo test de esta lección. Si sacas un 100% de aciertos puedes pasar a la siguiente lección, sino te recomiendo que no te la saltes.

Alojamiento y servidores HTTP

Para este curso nos vale con entender las siguientes definiciones (informales):

- **Servidor web HTTP** (o simplemente: *servidor web* o *servidor HTTP*): es un programa que se encarga principalmente de las comunicaciones con el navegador[1]. Envia y recibe mensajes y archivos.
- Un **alojamiento web** (*web hosting* o *hosting*): es un ordenador[2] conectado a Internet[3] (normalmente 24 horas, 7días a la semana) en el que hay instalado entre otros programas, un servidor HTTP y al que podemos solicitarle recursos. Comúnmente también se le llama: **servidor**.

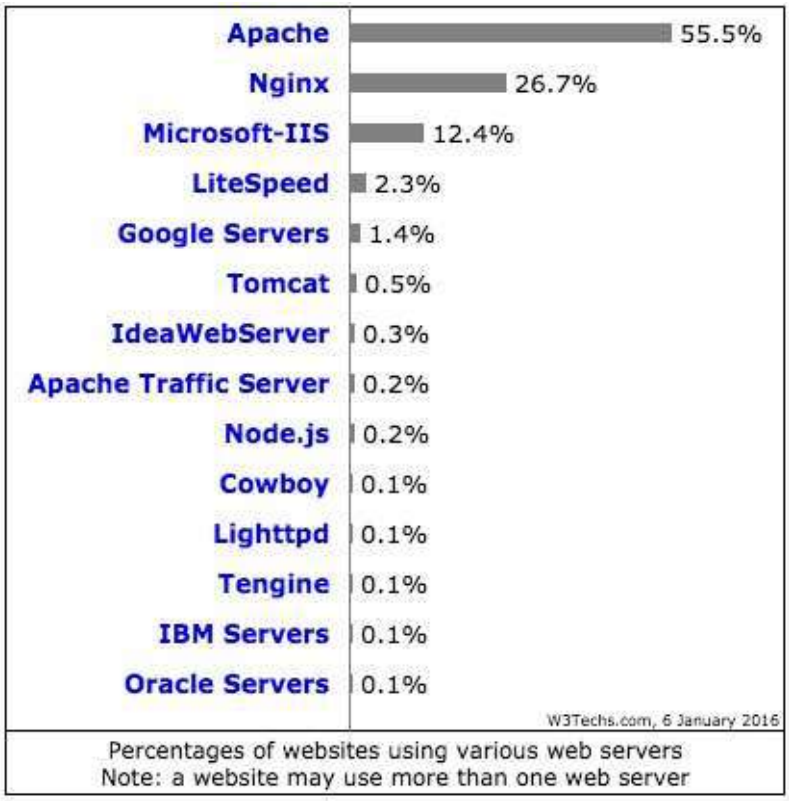
En este curso usarás tu máquina como servidor mientras estés haciendo pruebas, y posteriomente usarás el hosting gratuito y el servidor HTTP que ofrece Github para entregar los ejercicios.

En el apartado *Peticiones HTTP* veremos en detalle cómo se comunica un servidor que tenga un servidor HTTP instalado con nuestro navegador.

Algunos de los servidores webs y empresas que ofrecen alojamiento web:

- Alojamiento web: 1and1, AWS, Linode, etc. (ver más)
- Servidores web: Apache, Nginx y IIS, etc. (ver más)

A continuación tienes una gráfica que muestra el porcentaje de los servidores web más usados:



Fuente: w3techs - 6 de Enero de 2016

Aclaraciones:

1. No siempre tiene que ser con un navegador, puede ser con otro tipo de software.
2. Hay muchos tipos de hosting, aunque a nosotros nos vale con esta definición
3. También podría estar conectado a una intranet (o red local)

Navegadores

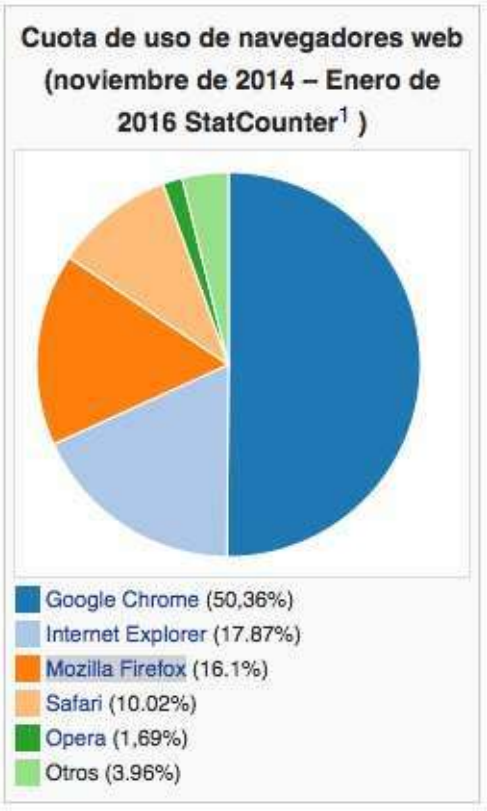
Un navegador es un programa (complejo) que entiende y transforma el código que recibe desde un servidor[1] en algo entendible para un humano (una página web, imágenes, etc).

Entre algunas de las responsabilidades que debe garantizar un navegador se encuentran:

- **Integridad**: para transmitir páginas web se utiliza una forma de comunicación llamada HTTP (o protocolo HTTP). Este protocolo es el lenguaje común entre el servidor y el navegador web.
- **Seguridad**: dado que los navegadores reciben código escrito normalmente por otras personas, el navegador implementa algunas reglas de seguridad. Estas reglas se definen a diferentes niveles: durante la conexión entre las máquinas y el envío (HTTPS), al ejecutarse en el navegador (CORS), etc.
- **Optimización**: al mismo tiempo que reciben la información, los navegadores integran mecanismos para acelerar la carga y mejorar la experiencia del usuario, por ejemplo acelerar el tiempo de carga utilizando una memoria de almacenamiento temporal (memoria caché), o comprimiendo los mensajes durante las comunicaciones.

Aunque existen multitud de navegadores, nosotros usaremos Google Chrome durante todo el curso.

Aquí te dejo además una gráfica que muestra la cuota de uso de los navegadores más populares:



Fuente: StatCounter

Aclaraciones:

1. Normalmente a través de un servidor web HTTP

URLs

Para acceder a un fichero/recurso a través de un navegador usamos la URL (Uniform Resource Locator). Cualquier URL sigue el siguiente formato:

```
scheme:[//[user:password@]host[:port]][/]path[?query][#fragment]
```

Aclaración: En informática se suelen utilizan los corchetes **[ ]** para indicar que lo que se encuentra contenido entre ellos es opcional.

Vamos a hacer un pequeño repaso a cada una de las partes de la URL:

- **scheme (obligatorio)**: este suele ser "http" o "https", aunque podría ser también: file, ftp, mailto, data, skype, etc.
- **user:password@**: usuario y contraseña (p.e. hhkaos:mipass@). Esto por ejemplo se utiliza en otro tipo de conexiones FTP o SSH que requieren identificación. También se puede usar para conectar a sistemas que usen .htpasswd, podremos identificar que una página está protegida con este sistema cuando al acceder nos aparece una ventana parecida a esta:

- **host**: nombre de dominio (p.e. rauljimenez.info), subdominio[1] o dirección IP del servidor web (p.e. 79.82.123.1)
- **port**: puerto[2] (si no se especifica ninguno se usa el 80 por defecto)
- **path (obligatorio)**: ruta al fichero (p.e. blog/index.php)
- **query**: parámetros/variables (p.e. preview=true)
- **fragment**: punto de anclaje[3] (p.e. #introducción)

Así una URL válida podría ser:

```
https://hhkaos.gitbooks.io/cursohtml5desdecero/content/chapter1.html#primeros-pasos-co
n-html5
```

Donde:

- scheme = https
- user:password@ = (vacío)
- host = hhkaos.gitbooks.io
- port = 80
- path = cursohtml5desdecero/content/chapter1.html
- query = (vacío)
- fragment = #primeros-pasos-con-html5

Otro ejemplo de URL válida podría ser: `file:///Users/hhkaos/index.html`

En este caso le estamos indicando al navegador que acceda a un fichero que se encuentra en nuestro disco duro.

Aclaraciones:
1. Un subdominio es un prefijo que le ponemos al dominio, por ejemplo: mail.google.com, y se puede configurar para que apunte a distintas IPs o "carpetas de nuestro disco duro". Así por ejemplo www sería también un subdominio
2. Los puertos son números enteros que nos permiten especificar a qué "*puerta*" o "*canal*" de una máquina nos queremos conectar. Por ejemplo en la web se suele usar el 80 para los servidores HTTP (aunque Skype también lo usa), el 21 para el FTP, el 22 para el SFTP, etc
3. Un punto de anclaje nos permite introducir una URL al abrirla hace scroll hasta un punto de la página determinado (ejemplo).

Peticiones HTTP

Cuando nuestro navegador quiere acceder a una página web a través de HTTP (*scheme* = http), lo que hace es comunicarse con un servidor HTTP. Para ello descompone la URL en diferentes partes que le permite conocer la dirección de la máquina (host) y la ruta (*path*) del recurso que al que quiere acceder (o al que le va a enviar información), y envía un mensaje al servidor, lo que formalmente se conoce como una *petición*.

Algunas de las acciones que provocan que un navegador realice una petición HTTP son: escribir una URL en la barra de direcciones, pulsar un enlace, refrescar una pestaña o enviar un formulario.

Existen varios tipos de peticiones, aunque nosotros en este curso trabajaremos con dos tipos:

- **GET**: para solicitar información.
- **POST**: para enviar información.

Los mensajes de respuesta del servidor pueden ser de muchos tipos, aunque nosotros nos encontraremos normalmente tres, que significan:

- **200**: que se ha encontrado correctamente el fichero/recurso.
- **403**: que no tenemos permiso para acceder al fichero/recurso.
- **404**: que el fichero/recurso que le hemos solicitado no se ha podido encontrar en el disco duro (puede ser porque no esté o porque la ruta es incorrecta).

El siguiente gráfico muestra un esquema simplificado que nos permite hacernos una idea sobre cómo funciona la comunicación entre ambos:

Ejercicio

En esta primera lección sólo quiero que hagas un ejercicio tipo test para ver si has entendido todos los conceptos. Lo que se pregunta en el test es lo que realmente me importa que recuerdes.

Ejercicio tipo test de autoevaluación - Lección 1

Ah! por cierto, puedes repetirlo tantas veces como quieras.

Si hay algo que no te haya quedado claro, recuerda que puedes preguntar cualquier duda en los issues del proyecto en Github.

Recursos

Aunque **no es imprescindible para entender y aprender a crear webs con HTML5, CSS3 y JavaScript**, los siguientes conceptos sí pueden serte útiles si quieres aprender a alojar un sitio web en un servidor tuyo propio.

Dominios y subdominios

En este curso no aprenderemos a comprar y configurar un dominio ya que *no* se ha considerado importante para los objetivos planteados (hay muchos tutoriales online para aprender a hacerlo).

Dicho esto creo que es interesante conocer qué es un dominio y qué es un subdominio:

- **Dominios**: podemos hacer una analogía entre los dominios y los *Accesos directos* del sistema operativo. Para identificar unívocamente una carpeta en nuestro disco duro necesitamos saber la ruta (*p.e: C:\Archivos de programa\Google Chrome\bin\chrome.exe*), pues para idenfiticar nuestros servidores en Internet necesitamos saber su dirección IP (p.e: 213.242.93.227). Pero como recordar todos estos dígitos es muy complejo, hace años se inventaron los dominios (p.e: google.com, rauljimenez.info, etc.) que no son más que nombres que apuntan a estas direcciones (algo así como un "*acceso directo*") y que son más fáciles de recordar para un humano. Nota: los dominios se pueden alquilar por Internet y los precios normalmente varían desde los pocos € o $ hasta cientos de € o $.
- **Subdominos**: un subdominio es un prefijo que le ponemos al dominio, por ejemplo: mail.google.com, y se puede configurar para que apunte a distintas IPs o "carpetas de nuestro disco duro". Así por ejemplo www sería también un subdominio.

Aunque el ICANN es la organización que gestiona los dominios a nivel mundial, existen multitud de empresas autorizadas para vender dominios.

Vídeo en inglés: qué son los dominios y los DNS

Servidores

Como hemos mencionado anteriormente un servidor o hosting no es más que "un ordenador" donde guardamos los ficheros y el resto de información que contienen nuestro sitio web.

Recordatorio: **En este curso usaremos nuestra máquina si ningún servidor HTTP ya que no nos hará falta de momento**.

Estos ordenadores no tienen por qué tener nada en especial, nuestra máquina puede hacer de servidor web, pero dado que (entre otras muchas cosas), una web se espera que funcione 24 horas, 7 días a la semana y que nuestra máquina la tenemos que apagar de vez en cuando, normalmente se alquila parte de una máquina (o una máquina completa) a una empresa que se dedica exclusivamente a esto: empresas de hosting o alojamiento web.

Se puede acceder y gestionar estas máquinas que alquilamos en remoto de diferentes maneras: usando clientes FTPs/SFTPs, paneles de control web, conexiones/clientes SSH, etc.

TCP

Ignora este apartado si no has estudiado nada relacionado con informática o telecomunicaciones. Simplemente quería hacer referencia que tanto los protocolos HTTP, como FTP, SSH, IMAP, DNS, POP, SMTP, etc. están implementados sobre TCP (+ìnfo).

Primeros pasos con HTML5

En esta segunda lección veremos:

- Una introducción a HTML5
- Qué son y cómo funcionan las etiquetas
- Qué es y cómo se hace la anidación de etiquetas
- La estructura básica de una página
- Algunas etiquetas básicas

El objetivo es empezar a familiarizarnos con HTML5 y prepararnos para la siguiente lección en la que configuraremos nuestro Google Chrome para poder empezar a escribir código.

Si crees que ya dominas esta materia puedes probar a hacer el ejercicio tipo test de esta lección. Si sacas un 100% de aciertos puedes pasar a la siguiente, sino te recomiendo que no te la saltes.

Introducción a HTML5

HTML (Hypertext Markup Language) es un lenguaje de marcado (*que no es lo mismo que un lenguaje de programación*) que sirve para definir la estructura y la semántica de nuestra página web (luego veremos que significa esto).

HTML fue creado y es mantenido por una organización sin ánimo de lucro llamada W3C. El W3C es un consorcio formado por más de 400 empresas (entre ellas las que desarrollan los principales navegadores como Google, Microsoft, Mozilla, Apple...), etc.

Desde el consorcio trabajan continuamente en definir cómo debe evolucionar este lenguaje y otros estándares que conforman *la web*. Posteriormente los fabricantes de navegadores preparan los mismos intentando conseguir que un código funcione igual en todos los navegadores. Aunque desafortunadamente no siempre es así, cada vez es una realidad más cercana.

Por tanto, a lo largo de los años las versiones de HTML han evolucionado: HTML 2.0 (1995), HTML 4.0 (1997), XHTML (2000), **HTML5 (2014)**, etc. con el objetivo de adaptarse a los nuevos tiempos y así dar soporte a nuevas necesidades (estandarización de los sistemas de audio, vídeo, etc).

Snippets interactivos

Vamos a ver una breve introducción al funcionamiento de la interfaz:

1. Permite navegar entre las diferentes lecciones
2. Permite navegar entre los diferentes snippets
3. Ejemplo de código HTML (con la sintaxis resaltada)
4. Resultado del código (3) en embebido en la página
5. Nos permite abrir la previsualización a pantalla completa
6. Nos permite usar el editor web Codepen.io, un editor bastante popular para experimentar con el código.
7. Enlace a los **issues** de Github donde podremos publicar cualquier duda o problema sobre los ejemplos.

Puedes acceder a esta interfaz a través de la siguiente URL:
http://libro.cursohtml5desdecero.com/snippets/html/

Etiquetas

En el último estándar de HTML (HTML5) existen más de 100 elementos que nos permitirán crear etiquetas. Como comentaba al inicio del curso no los veremos todos, de hecho no es habitual encontrar a nadie que los conozca todos, ni siquiera los que llevamos tantos años haciendo webs, lo importante es saber dónde buscarlos y saber cómo usarlos.

Por esto vamos a empezar por entender qué aspecto tienen. Lo primero es saber que las etiquetas sólo pueden ser de dos tipos:

1) **Las que tienen una apertura y un cierre** como en el siguiente caso:

```
<elemento atributo="valor">Contenido</elemento>
```

Por ejemplo:

```
<a href="http://www.google.com">Google</a>
```

En este caso decimos que: "*tenemos un elemento **a** donde el valor del atributo **href** es **http://www.google.com**, y que su contenido es **Google***". No hace falta que te preocupes aún por el significado, luego haremos incapié en esto.

Si nos fijamos las etiquetas **siempre** están contenidas entre los símbolos < >, y el cierre sólo incluye el nombre del elemento precedido de una barra lateral "/" (p.e. </elemento>).

2) Por otro lado están los **elementos auto-contenidos** (los que no se cierran explícitamente):

```
<elemento atributo="valor">
```

Por ejemplo:

```
<img src="html-css-js.png" alt="Logotipos de HTML5, CSS3 y JavaScript">
```

Es importante destacar que el *atributo* y el *valor* son opcionales.

Vamos a ver algunos ejemplos de nombres de etiquetas:

- **elemento**: *html*, *head*, *meta*, *title*, *body*, *img*...
- **atributo**: *charset*, *src*, *alt*, ...

- **valor**: *UTF-8*, "*url*" (la URL a un recurso), "texto", ...

A mi siempre me gusta tener una "chuleta" (o cheatsheet) a mano que resuma todos los elementos y algunos de los atributos que aceptan. Pero por ahora no hace falta que te distraigas con esto, lo importante es que entiendas el formato.

Anidación de etiquetas

También es importante saber que unas etiquetas pueden contener a otras (una o varias), o como se suele decir "que se pueden anidar".

Por ejemplo:

```
<head>
    <title>Título de la página</title>
    <meta name="author" content="Raúl Jiménez Ortega - @hhkaos">
</head>
```

En este caso vemos que la etiqueta **head** tiene como contenido a otra etiqueta **title**. En este caso se dice que:

- La etiqueta *head* es el **padre** de la etiqueta *title* y *meta*.
- Y que la etiqueta *title* y *meta* son **hijas** de la etiqueta *head*.
- La etiqueta *title* y *meta* son **hermanas**.

Si nos fijamos, además, la etiqueta anidada (*title*) está en una nueva línea y con un nivel de indentación/sangrado mayor. Esto es así por una convención mundial a la que se ha llegado para que los programadores escribamos código de una manera similar, tanto para hacernos más fácil y comprensible el código cuando este crezca, como para cuando tengamos que compartirlo con otros programadores.

Orden de apertura y cierre

Cuando anidamos etiquetas unas dentro de otras es muy importante cerrarlas *en orden*. Esto quiere decir que la primera etiqueta en cerrarse tiene que ser la última que se abrió, así por ejemplo este ejemplo sería **incorrecto**:

```
<p>El orden es <strong>muy importante</p></strong>
```

La forma **correcta** de hacerlo sería:

```
<p>El orden es <strong>muy importante</strong></p>
```

Recordemos que hay etiquetas que no es necesario cerrarlas por lo que esto sería **correcto**:

```
<p>
    El orden es <strong>muy importante</strong>.<br>
    Así introducíamos un salto de línea.
</p>
```

Otros aspectos

Me gustaría puntualizar otros dos detalles:

1. No todas las etiquetas son anidables entre sí; por ejemplo: una etiqueta **body** no puede contener una etiqueta **head**. Pero no te preocupes de momento por esto, en otra lección veremos cómo podemos saber qué etiquetas son anidables entre sí.
2. Es importante indentar bien el código porque en muchos casos nos encontraremos muchos niveles de anidación, etiquetas que contienen etiquetas que a su vez contienen etiquetas, etc. ya que **no existe un límite máximo de anidamiento**.

Estructura básica de una página

Primero me gustaría hacer una pequeña aclaración sobre terminología que voy a usar, diferenciaremos:

- **Página web**: como una página individual dentro de un sitio web (p.e: rauljimenez.info/contacto)
- **Sitio web (o web)**: como el conjunto de todas las páginas en las que podemos navegar dentro de un mismo dominio (p.e: rauljimenez.info es un sitio web que incluye: rauljimenez.info/experiencia, rauljimenez.info/contacto, etc).

Dicho esto, toda *página web* que hagamos en HTML5 debe tener una estructura inicial similar a la siguiente:

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

A continuación explicamos la función que cumple cada etiqueta:

- `<!DOCTYPE html>` : indicar al navegador que el código HTML en el que está escrita la página es en la versión 5, osea que es HTML5. +info
- `<html lang="es">... </html>` : indica la raíz del documento y **todas** las etiquetas deben estar incluidas dentro. Además se especifica el idioma en el que está escrita, *es* = Español (+idiomas).
- `<head> ... </head>` : se usa para envolver otras etiquetas que ofrecen información principalmente a: el navegador, a los buscadores y a otras páginas (como pueden ser redes sociales, etc). La información especificada dentro del *head* no se muestra *dentro*[1] de la página web que ve el usuario.
- `<meta charset="UTF-8">` : indica al navegador qué tipo de caracteres contiene la página. Con el atributo charset indicamos cuál de todos los juegos de caractares disponibles usamos. Con el valor UTF-8 podremos crear contenido en la mayoría de los sistemas de escritura: español, inglés, francés, etc.
- `<title> ... </title>` : indica el título de nuetra página. Este se muestra en: la pestaña del navegador, el enlace que indexan los buscadores, etc.
- `<body> ... </body>` : contiene todo el contenido visible por el usuario *dentro* de nuestra

página.

Observa que la etiqueta *html* contiene dos hijas: *head* y *body*, esto ya no es obligatorio en HTML5 ya que se puede omitir las etiquetas **html**, **body** y **head**, pero por convención es recomendable usarlas.

> Aquí puedes ver el **ejemplo interactivo**: Lección 1 - Snippet 1

Aclaraciones:

1. Cuando digo **dentro** me refiero al contenido de la página, lo que no incluye la pestaña del navegador ni la barra de direcciones.

Etiquetas básicas

Para terminar esta lección vamos a aprender el significado de ocho de las etiquetas que con más frecuencia tendremos que usar cuando creemos páginas web:

- `<p></p>` : representa un párrafo (+info).
- `<br>` : representa un salto de línea (+ìnfo).
- `<h1></h1>` : esta etiqueta se utiliza para representar el encabezado de una página, como si fuera el índice de un libro. Puede variar desde 1 hasta 6 para diferenciar subniveles (+info).
- `<ul></ul>` : representa una lista de elementos, donde el orden de los elementos no es importante - esto quiere decir que el cambio del orden no modifica el significado. (+ìnfo).
- `<ol></ol>` : representa una lista de elementos, donde el orden de los elementos sí es importante - esto quiere decir que el cambio del orden modifica el significado. (+ìnfo).
- `<li></li>` : representa un elemento de la lista y su padre siempre tiene que ser una etiqueta *ol* o *ul*. (+ìnfo).
- `<strong></strong>` : representa algo muy importante, serio (para avisos o precauciones) o urgente (que debe ser leído antes). (+ìnfo).
- `<em></em>` : sirve para enfatizar en el contenido. (+ìnfo).
- `<!-- -->` : se utiliza para añadir comentarios dentro del código que el usuario no podrá ver. Por ejemplo para añadir notas de tareas pendientes, aclaraciones que nos ayuden a nosotros o a otras personas a entender el código, etc. (+ìnfo).

> Puedes consultar los ejemplos en la lección 2 - Snippet 1-5).

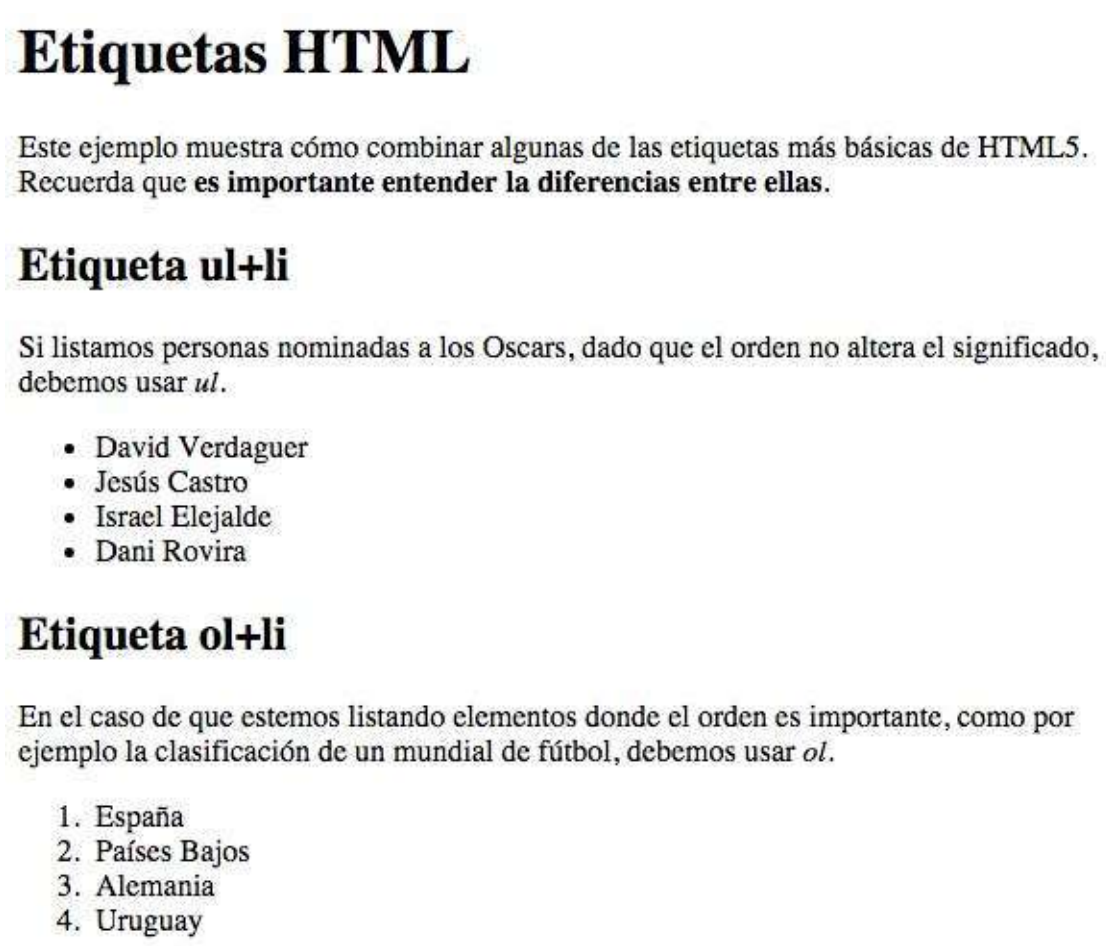
Truco: Para que recuerdes mejor qué significa cada elemento, las etiquetas piensa en los acrónimos en inglés:

- h1 = **h**eading**1**; h2 = **h**eading **2**; ...
- p = **p**aragraph
- br = **b**reak **l**ine
- ul = **u**nordered **l**ist
- ol = **o**rdered **l**ist
- li = **l**ist **i**tem
- em = **em**phasis

El siguiente ejemplo muestra una página web que combina todas ellas:

```
<!DOCTYPE html>
<!-- TODO: añadir la etiqueta lang -->
<html>
<head>
    <meta charset="UTF-8">
    <title>Ejemplo con etiquetas básicas</title>
</head>
<body>
    <h1>Etiquetas HTML</h1>
    <p>
        Este ejemplo muestra cómo combinar algunas de las etiquetas más básicas de HTM
L5. <br>
        Recuerda que <strong>es importante entender la diferencias entre ellas</strong>
.
    </p>

    <h2>Etiqueta ul+li</h2>
    <p>
        Si listamos personas nominadas a los Oscars, dado que el orden no altera el si
gnificado, debemos usar <em>ul</em>.
    </p>
    <ul>
        <li>David Verdaguer</li>
        <li>Jesús Castro</li>
        <li>Israel Elejalde</li>
        <li>Dani Rovira</li>
    </ul>

    <h2>Etiqueta ol+li</h2>
    <p>
        En el caso de que estemos listando elementos donde el orden es importante, com
o por ejemplo la clasificación de un mundial de fútbol, debemos usar <em>ol</em>.
    </p>
    <ol>
        <li>España</li>
        <li>Países Bajos</li>
        <li>Alemania</li>
        <li>Uruguay</li>
    </ol>
</body>
</html>
```

Esto generaría una página como la siguiente:



Etiquetas HTML

Este ejemplo muestra cómo combinar algunas de las etiquetas más básicas de HTML5.
Recuerda que **es importante entender la diferencias entre ellas.**

Etiqueta ul+li

Si listamos personas nominadas a los Oscars, dado que el orden no altera el significado, debemos usar *ul*.

- David Verdaguer
- Jesús Castro
- Israel Elejalde
- Dani Rovira

Etiqueta ol+li

En el caso de que estemos listando elementos donde el orden es importante, como por ejemplo la clasificación de un mundial de fútbol, debemos usar *ol*.

1. España
2. Países Bajos
3. Alemania
4. Uruguay

Si quieres puedes ver el ejemplo en vivo aquí: Lección 1 - Snippet 2

Es importante destacar que aunque el navegador le añade un estilo (CSS) por defecto a las etiquetas, por ejemplo:

- `h1` y `h2` una fuente mayor y negrita
- `strong` en negrita
- `ul` y `ol` con un margen a la izquierda y un punto o número respectivamente
- `em` en cursiva

Esto no es responsabilidad del HTML, esto lo podremos personalizar en un futuro con CSS. Así que insisto, recuerda que HTML **sólo sirve para dotar de estructura y semántica al contenido**.

Este valor semántico es **muy importante** entre otras cosas para por ejemplo:

- Que los buscadores (que no son más que programas automatizados) puedan "entender" el contenido de nuestra página y así poder detectar de qué estamos hablando y qué es importante.
- Para que otras herramientas como por ejemplo los navegadores para invidentes (p.e.

WebbIE) que lo que hacen es leer el contenido al usuario u otros navegadores basados en texto.

Ejercicio tipo test

Ejercicio tipo test de autoevaluación - Lección 2

Recuerda que puedes repetirlo tantas veces como quieras.

Dudas

Si hay algo que no te haya quedado claro puedes preguntar cualquier duda en los issues del proyecto en Github.

Recursos

Aquí te dejo dos tipos de recursos. Los avanzados míralos sólo si ya tenías un conocimiento previo de programación web (XHTML, HTML4, etc) o... si no le tienes miedo a nada ;D:

- Básicos:
  - Chuleta de etiquetas HTML5
  - Artículo: ¿Qué puede ocurrir si realizamos mal la anidación de etiquetas?
- Avanzados:
  - Organización sin ánimo de lucro responsable de gestionar los dominios a nivel mundial: ICANN - Wikipedia
  - The Web Hypertext Application Technology Working Group

Chrome DevTools

En esta lección vamos a empezar a trabajar con las herramientas de Google Chrome para desarrolladores (Chrome DevTools)[1].

Como en las lecciones anteriores, si crees que ya dominas esta materia puedes probar a hacer el ejercicio tipo test de esta lección. Si sacas un 100% de aciertos puedes pasar a la siguiente, sino te recomiendo que no te la saltes.

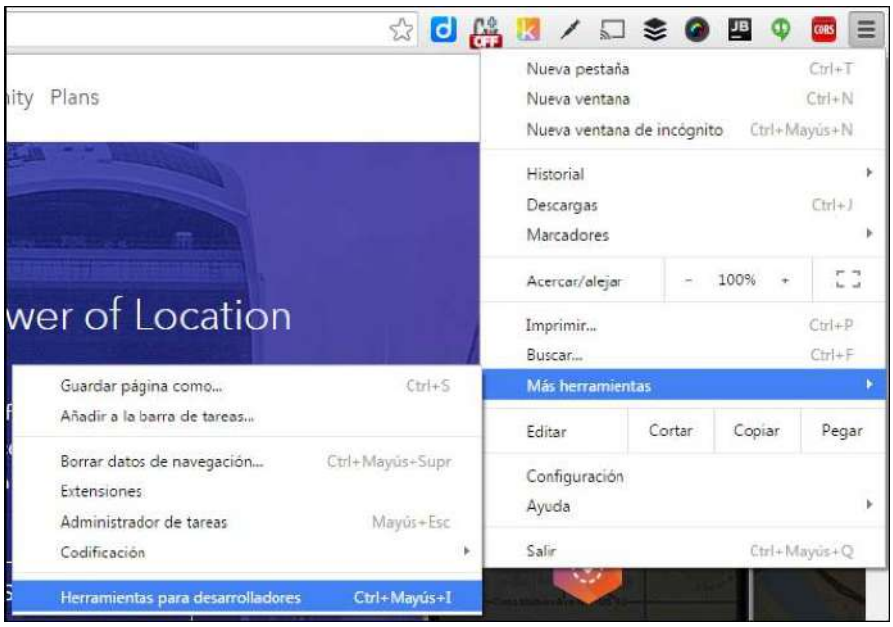
Pestañas

De momento sólo vamos a ver 3 grupos de herramientas que se encuentran organizadas por pestañas:

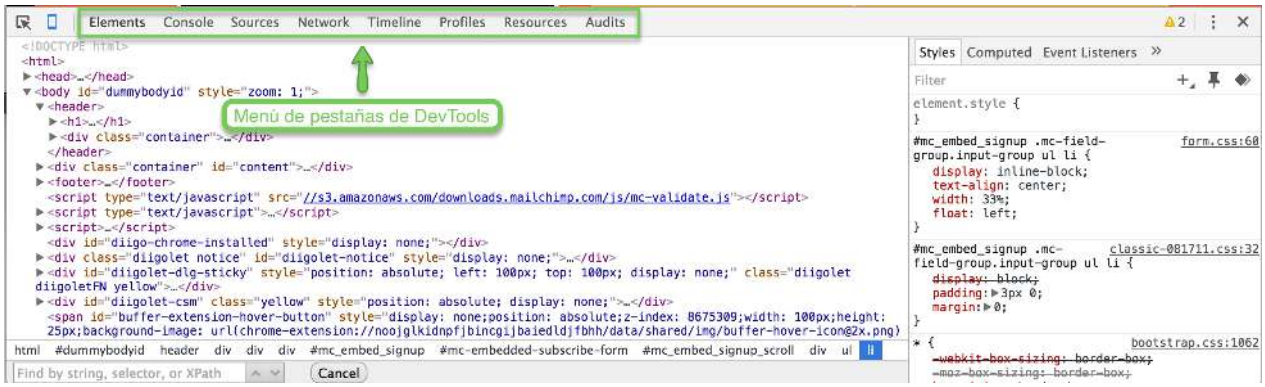
- **Red** (*Network*): esta pestaña nos permite ver los *recursos* que recupera nuestro navegador usando peticiones HTTP mientras cargamos y usamos la página.
- **Elementos** (*Elements*): nos permite ver y modificar el código[2] que representa la página que estamos viendo.
- **Fuentes** (*Sources*): nos permite navegar por todos los ficheros (HTML, CSS y JavaScript) que utiliza la página que estamos viendo.

La barra de herramientas la podemos abrir en cualquier página que estemos viendo. Para abrir esta barra podemos hacerlo mediante un atajo de teclado o mediante el menú de herramientas del Chrome:

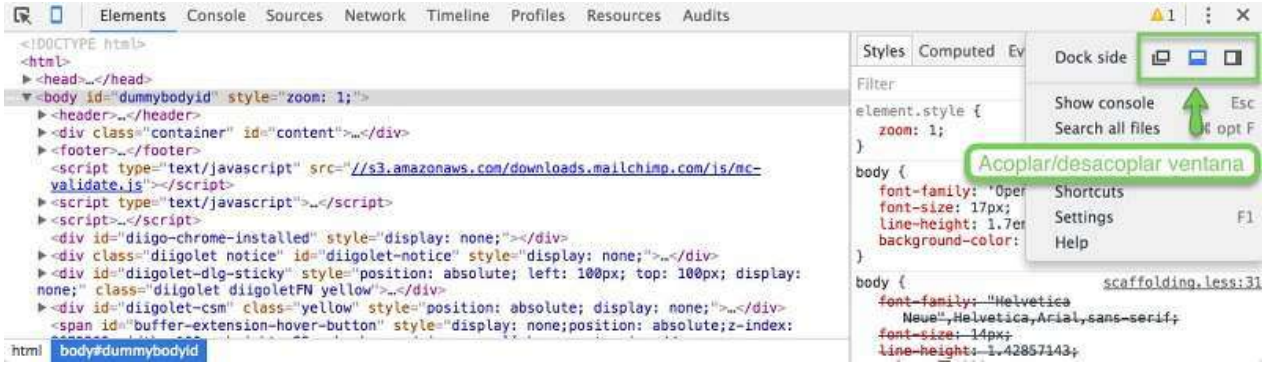
- **Atajo de teclado** (recomendado):
  - En Windows pulsando: F2 o Control + Shift + I
  - En Mac pulsando: Cmd + Opt + I
- **Pulsando el botón de menú**: "Botón de menú" -> "Más herramientas" -> "Herramientas para desarrolladores".
  Como podemos ver en la siguiente imagen.



Una vez hecho esto nos aparecerá la barra de herramientas:



La barra podemos ajustarla a la derecha, abajo o desacoplarla en una nueva ventana como vemos a continuación:



Vayamos ahora analizando las pestañas.

Aclaraciones:

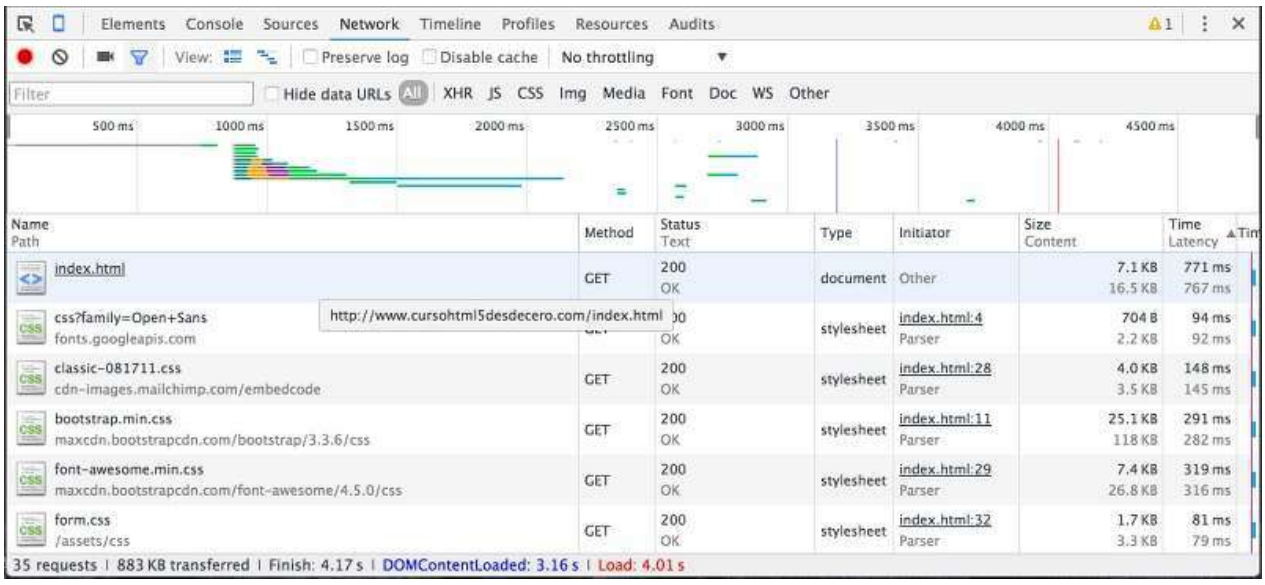
1. Puedes ampliar toda la información que veremos en este capítulo en la página de documentación para desarrolladores de Google Chrome, en el curso Discover DevTools o

para los más avanzados en el curso Website Performance Optimization
2. El del DOM (que luego veremos que es) y el CSS, aunque de momento no nos entretendremos en esta parte.

Pestaña network

Como decíamos, esta pestaña nos permite ver los recursos que solicita el navegador de un servidor usando peticiones HTTP. También nos permite ver los detalles de las mismas: tipo del mensaje (GET/POST), código de respuesta (200, 404, ...), etc[1].

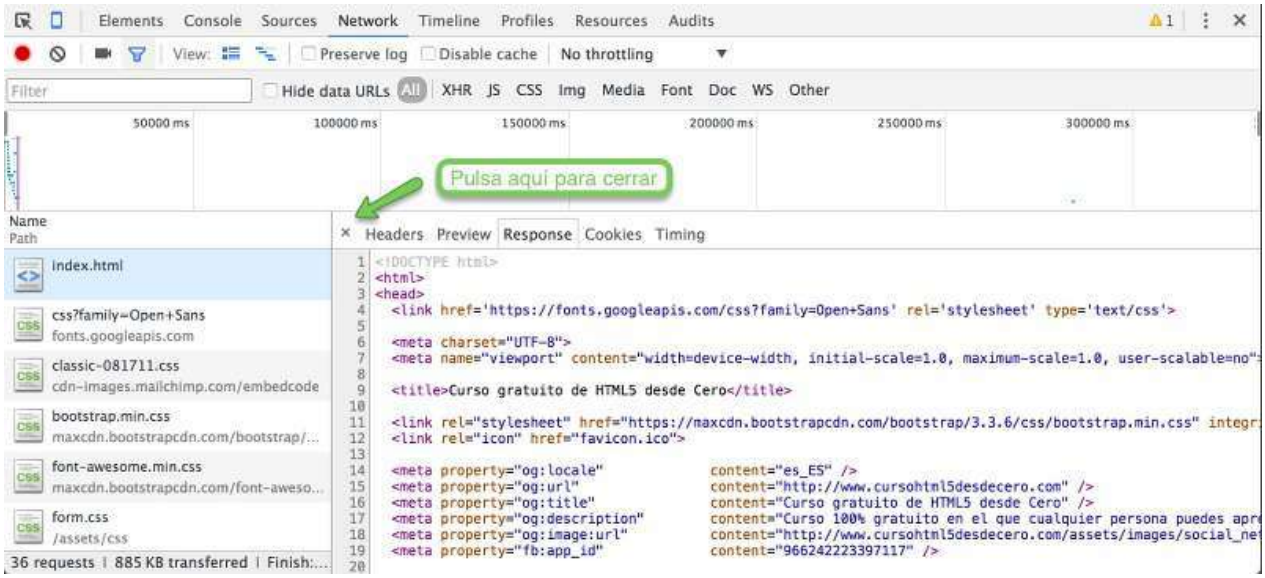
Si abres las DevTools después de haber cargado la página, la pestaña "Network" te saldrá vacía, si es así, refresca la página y verás como te aparece algo parecido[2] a esto:



Cada fila representa una petición HTTP, y si te fijas, dejando el cursor encima de una petición te mostrará la URL completa del recurso y si pulsamos el botón derecho tendremos varias opciones, entre ellas la de abrir el recurso en una nueva ventana, eliminar todos los ficheros de la memoria caché, etc.

> **Nota:**
> Podemos vaciar la memoria caché[3] de dos formas distintas, una es pulsando el icono de refrescar con el botón derecho y luego "Empty Cache and Hard Reload", y otra es pulsando el botón derecho sobre cualquiera de las peticiones HTTP y pulsar "Clear browser cache"

Si haces clic en cualquiera de las peticiones podrás ver los contenidos del recurso y algunos detalles que no nos preocuparán en este curso.



Para cerrar el detalle de la petición puedes pulsar en el aspa.

Además de filtrar las peticiones también puedes reordenarlas pulsando en el título de cada campo: **Name**, **Method**, **Status**, etc.

Veamos ahora **algunas**[4] de las cosas que podemos hacer en esta pestaña. Si te fijas, las opciones en esta imagen no coinciden exactamente con las de la imagen anterior (y posiblemente tampoco con las tuyas), esto se debe a que este "pantallazo" es de una versión anterior del navegador (no importa), veamos que significan:

- **Preserve records on navigation**: por defecto aparece el botón en rojo, esto significa que cada vez que cambiemos de página se eliminarán las peticiones y se añadirán las nuevas. En cambio, si lo desactivamos se dejarán de registrar nuevas peticiones.
- **Preserve log**: si marcas esta opción, el efecto será justo el contrario, nunca se

borrarán las peticiones HTTP, ni cambiando de página ni de dominio (se irán añadiendo una tras otra).
- **Clear records**: este botón nos permite limpiar la información de las peticiones.
- **Filter**: nos permite filtrar las peticiones, se buscarán *URLs* que contengan el texto introducido.
- **Hide/show filter buttons**: para ocultar/mostrar los botones para filtar.
- **Filter buttons**: estos botones nos permite ver sólo las peticiones HTTP que ha recuperado un tipo de recurso. *De momento* no lo usaremos mucho.
- **Summary view**: podemos ver cuántas peticiones HTTP se han necesitado para cargar la página (requests), cuando ocupa la suma de todos los recursos recuperados (XXX transferred), el tiempo exacto que ha tardado en descargar los recursos (ms = milisegundos), y en la siguiente lección veremos qué es el DOM y qué significa el DOMContentLoaded.
- **No throttling** (se ve en la imagen anterior): esto permite simular que tu conexión a Intenet es más lenta[5]. Lo usaremos más adelante cuando queramos ver si nuestra página carga lo suficientemente rápido usando un dispositivo conectado por 3G.

Por último y antes de pasar a la siguiente lección, te animo a que dediques un par de minutos a jugar con esta pestaña, y si te surge alguna duda... no olvides preguntarla en los issues de Github.

Aclaraciones:
1. Es la página de desarrolladores de Chrome puedes consultar que significa cada columna, aunque *no es trivial y no te recomiendo que lo hagas si estás empezando*.
2. Las peticiones serán distintas en cada página, por lo tanto con casi total seguridad tus peticiones serán distintas a las de la imagen.
3. Todos los navegadores incluyen una memoria caché temporal para optimizar el tiempo de carga de la página, de este modo el navegador puede reducir el número de peticiones HTTP (más info).
4. Si ya tienes experiencia con Chrome DevTools y quieres, encontrarás dónde ampliar conocimientos en la sección recursos de esta lección.
5. El throttling no funciona cuando estamos cargando un fichero sin utilizar un servidor web.

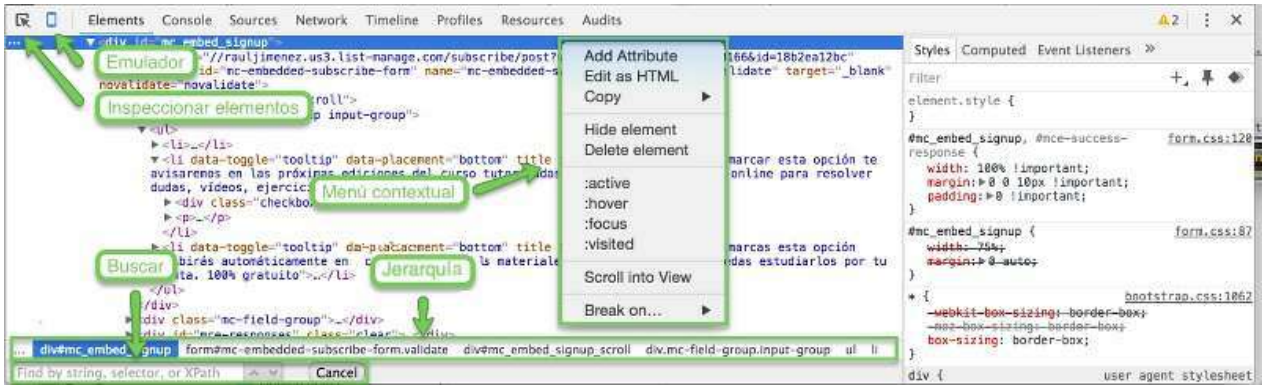
Pestaña elements

La pestaña "**Elements**" representa lo que llamaremos el "**DOM**"[1] (*Document Object Model*), que no es más que lo que representa la página que estamos viendo.

El DOM lo construye el navegador a partir del código HTML que recibe tras hacer la petición HTTP inicial. Además, el navegador intenta arreglar cualquier error que se encuentre en el código, por ejemplo: si se nos olvida cerrar una etiqueta, si anidamos etiquetas que no son anidables, etc. Por ese motivo y porque el DOM se puede modificar[2], este se parece pero no suele ser exactamente igual al código HTML recibido en la petición HTTP.

Además de poder ver el DOM, podemos editarlo, buscar texto en él y hasta reordenar las etiquetas arrastrándolas con el ratón.

En la siguiente imagen vemos las diferentes partes de esta pestaña:



- **Emulador**: esta opción nos permitirá simular que estamos usando un móvil o tablet y hacer *throttling* (simular otro tipo de conexión) al igual que hemos visto en la pestaña "**Network**".
- **Inspeccionar elementos**: activando esta opción podrás hacer clic sobre cualquier parte de la página y el inspector señalará en el DOM el código que representa el elemento seleccionado.
- **Menú contextual**: pulsando el botón derecho sobre el código veremos varias opciones:
  - **Add attribute**: permite añadir un atributo, por ejemplo: *charset="UTF-8*" (atajo de teclado: *Enter*).
  - **Edit as HTML**: nos permite añadir, editar o quitar cualquier cosa (atajo de teclado: F2 tanto en Windows como en Mac)
  - **Copy**: nos permite copiar el código (*outerHTML*), el selector (ya veremos lo que significa cuando veamos CSS), el XPath[3] (es un lenguaje que nos permite buscar y seleccionar elementos teniendo en cuenta la estructura jerárquica del código), cotar y copiar el elemento.
  - **Ocultar elemento**: cambia la visibilidad a "no visible" usando CSS.

    - **Delete element**: permite eliminar el elemento (atajo de teclado: *Suprimir* o *Borrar*).
    - **:active, :hover, :focus, :visited**: nos permite cambiar el estado del elemento (esto lo usaremos en el apartado de CSS)
    - **Scroll into View...**: en caso de ser necesario se hace scroll hasta que se muestre el elemento.
    - **Break on...**: nos permite establecer puntos de parada indicando que se debe detener la ejecución de cualquier código JavaScript si:
        - Se modifica alguno de sus hijos.
        - Se modifica algún atributo.
        - O si se elimina el código.
- **Buscar**: Nos permite buscar cualquier palabra dentro del DOM (atajo de teclado: *Ctrl* + *F* en Windows ó *Cmd* + *F* en Mac).
- **Jerarquía**: nos muestra todos los ancestros del elemento y nos permite seleccionarlos.

Los cambios que hagas sobre esta pestaña no se guardarán ya que no estamos modificando el fichero de código[4] sino el DOM (y ya hemos visto la diferencia), por tanto al refrescar la página todos los cambios desaparecerán.

Para mejorar tu productividad te recomiendo que de vez en cuando consultes los atajos de teclado del panel Elements, como por ejemplo:

Windows/Linux	Mac	Función
Ctrl + Z	Cmd+Z	Deshacer los cambios realizados
Ctrl + Shift + C	Cmd + Shift + C	Abrir DevTools con la herramienta para "**Inspeccionar elementos**" activada por defecto
←, →, ↑, ↓	←, →, ↑, ↓	Navegar por los elementos del DOM

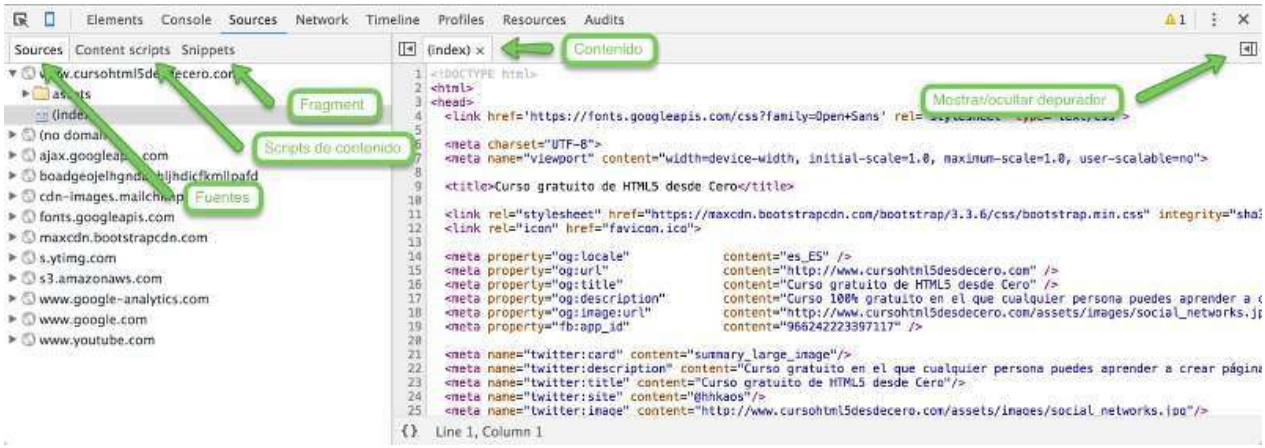
El panel que sale a la derecha es el del código CSS que se le ha aplicado al elemento seleccionado, pero esto ya lo veremos más adelante.

Aclaraciones:
1. Puedes encontrar la definición formal del DOM en la página del W3C.
2. Usando JavaScript, o mediante una extensión del navegador por ejemplo
3. Puedes encontrar la definición formal de XPATH en la página del W3C
4. A veces escucharás "código fuente" en lugar de código, son sinónimos.

Pestaña sources

La pestaña "**Sources**" nos muestra las fuentes de contenido que se han utilizado para construir la página. Desde esta pestaña podemos escribir y modificar ficheros que estén vinculados a nuestro disco duro, pero veremos cómo hacer esto en la siguiente lección.

Empecemos por describir los distintos paneles:



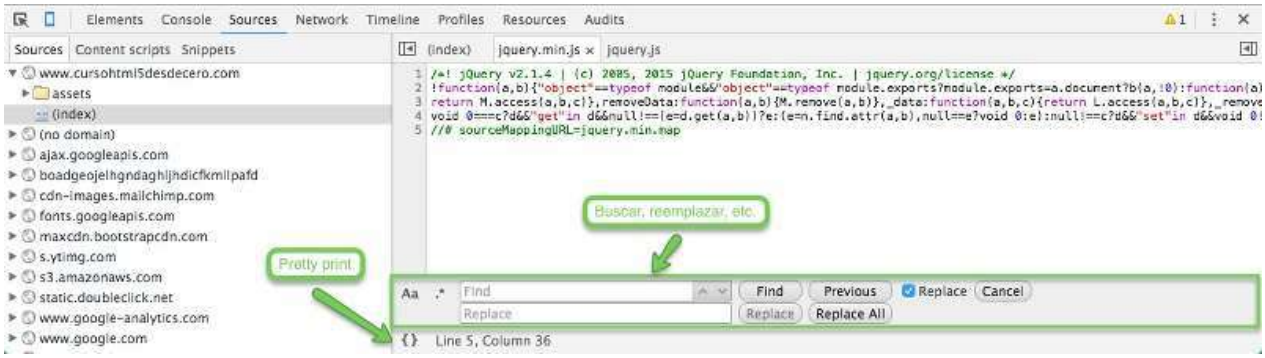
- **Sources**: aquí encontraremos por cada dominio desde el cual nuestro navegador haya obtenido recursos[1] (HTML, CSS o JavaScript) una jerarquía de ficheros. Haciendo clic en cualquiera de ellos se abrirá el código en un panel derecho.
- **Content scripts**: aquí se encuentran ficheros JavaScript simples (scripts) implementados desde las extensiones de Google Chrome.
- **Snippets**: esta pestaña nos permite almacenar *pequeños trozos de código*[2] *reutilizables* (snippets) escritos en JavaScript que podremos ejecutar o reutilizar (valga la redundancia) en cualquier página.
- **Depurador**: este panel nos ayudará a hacer un seguimiento paso a paso de la ejecución de nuestro código JavaScript para encontrar errores, veremos como usarlo en los capítulos de JavaScript.

Al igual que en la lección anterior, te recomiendo que guardes en un lugar seguro los atajos de teclado del panel Sources y de vez en cuando los revises para aumentar tu productividad.

Panel de contenido

El panel de contenido nos ofrece un editor de código que dispone adicionalmente de otros atajos de teclado.

Es importante saber que: a diferencia de los cambios del DOM, en la pestaña "**Elements**" para poder ver los cambios reflejados en la página que estamos viendo es necesario **guardar los cambios y refrescar la página**.



En cuanto a los atajos me gustaría destacar cinco que usaremos con **mucha frecuencia**:

Windows/Linux	Mac	Función
Ctrl + F	Cmd + F	Buscar (y adicionalmente reemplazar) texto dentro de un fichero
Ctrl + S	Cmd + S	Guardar un fichero
Ctrl + R, F5	Cmd + R	Refrescar la página
Ctrl + P	Cmd + P	Buscar ficheros por nombre
Ctrl + P + :**num**	Cmd + P + :**num**	Acceder directamente a un número de línea

Por último, la opción "**Prety print**" veremos que es especialmente útil cuando estemos depurando bibliotecas JavaScript minificadas (*comprimidas*), aunque de momento no te preocupes por esto.

Recursos y aclaraciones:
1. Normalmente mediante peticiones HTTP aunque puede que también mediante las extensiones de Chrome.
2. En este repositorio de Github podrás encontrar una colección de snippets.

Configuración

Ahora vamos a ver cómo podemos configurar Google Chrome para poder modificar ficheros que se encuentren en nuestro disco duro.

Para hacer esto usaremos los "**Workspaces**", estos nos permitirán realizar cambios persistentes en nuestro código sin tener que ejecutar otro editor de código.

Para ello vamos a seguir los siguientes pasos:

1. Creamos una carpeta (de prueba) en nuestro disco duro (por ejemplo "*html*").
2. Creamos un fichero vacío dentro de la carpeta llamado: "*index.html*"[1]
3. Abrimos el fichero con Google Chrome
4. Abrimos las DevTools y nos vamos a la pestaña "**Sources**"
5. Hacemos clic en el panel izquierdo sobre la ruta del directorio y seleccionamos "**Add Folder to Workspace**":

6. Y por último pulsamos "**Allow**" para autorizar a DevTools a realizar cambios persistentes en el disco duro:

> **Nota**:
> Si nos equivocamos al añadir un directorio al Workspace podremos eliminarlo simplemente pulsando con el botón derecho sobre el directorio y seleccionando la opción "**Remove Folder from Workspace**".

Ahora tenemos que "mapear" (vincular) el recurso que ha obtenido el navegador con el fichero del disco duro que queremos modificar (osea, con él mismo), para ello:

1. Hacemos clic con el botón derecho sobre el nombre del fichero (*index.html* que cuelga de "file://")

2. Elegimos la opción "**Map to File System Resource**":

3. Seleccionamos el fichero dentro del espacio de trabajo (Workspace).
4. Y refrescamos la página.

Y ya estamos listos para empezar a programar usando las Chrome DevTools.

También puedes consultar las limitaciones de los "**Workspaces**", pero no te preocupes por ellas ya que no nos afectarán en este curso; por ejemplo, los cambios en la pestaña "**Elements**" no serán persistentes (lógico ya que lo que estamos cambiando es el *DOM*, que como vimos anteriormente es dinámico, osea que va cambiando).

Gestión de ficheros

Una vez hecho todo esto podemos añadir y eliminar ficheros directamente desde DevTools:

1. Añadir ficheros: pulsando con el botón derecho sobre la carpeta y seleccionando "**New File**".
2. Eliminar ficheros: pulsando con el botón derecho y seleccionando "**Delete**"

Sin embargo no podemos crear y eliminar directorios, esto lo tendremos que hacer directamente desde la carpeta que hemos creado en nuestro disco duro.

Aclaraciones:
1. A pesar de que no vamos a usar aún un servidor web, lo llamaremos así para ir acostumbrándonos a este nombre. Por defecto los servidores web cuando reciben la petición de un recurso y no se indica explícitamente el nombre del recurso, busca en la carpeta un fichero con nombre "index.html", y lo devuelve en caso de encontrarlo.

Ejercicio

Ejercicio tipo test de autoevaluación - Lección 3

Una vez más, recuerda que puedes repetirlo tantas veces como quieras.

El ejercicio práctico es muy simple, tan sólo consiste en añadir el código que hemos visto en el capítulo "**HTML5: Primeros pasos > Etiquetas básicas**" (lo puedes encontrar al final de esta lección) al fichero *index.html* que hemos creado.

Luego quiero que te familiarices un poco con las pestañas que hemos visto e intentes:

- Usar los atajos de teclado en la pestaña "**Sources**".
- Buscar la petición HTTP que se envía al cargar la página en "**Network**".
- Editar el código en la pestaña "**Elements**":
  - Modifica el contenido y el HTML de la etiqueta **h1**
  - Elimina una etiqueta
  - Reordena otra etiqueta
  - Haz una búsqueda y encuentra por ejemplo: **Rovira**
  - Haz clic en la jerarquía para acceder al elemento "**ul**".
  - Ejecuta el emulador y activa la opción de **Apple iPhone 5**.

```
<!DOCTYPE html>
<!-- TODO: añadir la etiqueta lang -->
<html>
<head>
    <meta charset="UTF-8">
    <title>Ejemplo con etiquetas básicas</title>
</head>
<body>
    <h1>Etiquetas HTML</h1>
    <p>
        Este ejemplo muestra cómo combinar algunas de las etiquetas más básicas de HTM
L5. <br>
        Recuerda que <strong>es importante entender la diferencias entre ellas</strong>
.
    </p>

    <h2>Etiqueta ul+li</h2>
    <p>
        Si listamos personas nominadas a los Oscars, dado que el orden no altera el si
gnificado, debemos usar <em>ul</em>.
    </p>
    <ul>
        <li>David Verdaguer</li>
        <li>Jesús Castro</li>
        <li>Israel Elejalde</li>
        <li>Dani Rovira</li>
    </ul>

    <h2>Etiqueta ol+li</h2>
    <p>
        En el caso de que estemos listando elementos donde el orden es importante, com
o por ejemplo la clasificación de un mundial de fútbol, debemos usar <em>ol</em>.
    </p>
    <ol>
        <li>España</li>
        <li>Países Bajos</li>
        <li>Alemania</li>
        <li>Uruguay</li>
    </ol>
</body>
</html>
```

Dudas

Si hay algo que no te haya quedado claro puedes preguntar cualquier duda en los issues del proyecto en Github.

Recursos

Definiciones que hemos visto:

- Definición de DOM por el W3C
- Definición de XPath por el W3C

Recursos sobre Chrome y las DevTools:

- Documentación para desarrolladores de Google Chrome
- Chrome DevTools Overview
- Atajo de teclado de Chrome DevTools
- Chuleta de atajos de teclado en Chrome
- Funcionamiento del cacheado
- Configuración del Workspace en Chrome DevTools
- Limitaciones del Workspace de Chrome DevTools
- Content scripts en Chrome DevTools
- Snippets en Chrome DevTools
- Colección de snippets para Chrome DevTools

Si es tu primera vez con DevTools ***no te lo recomiendo***, pero si quieres, puedes encontrar más información sobre cómo sacarle más provecho a Google Chrome en los siguientes cursos:

- Discover DevTools - CodeSchool.com
- Website Performance Optimization - Udacity.com
- Browser rendering optimization - Udacity.com: aprende cómo funciona internamente Google Chrome y cómo optimizar el layout (**conocimientos en CSS requestidos**).

HTML5: Mi Curriculum Vitae

En este capítulo vamos a aprender todo lo necesario para crear el código HTML de una página con nuestro CV, y en el siguiente aprenderemos cómo podemos subir nuestro curriculum a un repositorio de Github y a ponerlo accesible para cualquier persona usando gh-pages.

Por tanto al terminar este apartado tendremos que haber creado una página HTML similar a esta:

- Experiencia profesional
- Habilidades y conocimientos
- Educación
- Perfiles sociales
- Contactar

Raúl Jiménez Ortega

Me apasionan temas como son: los negocios en Internet, la analítica web, la optimización web (A/B Testing, UX, ...), el SEO, los móviles y el SoLoMo, el emprendimiento y las startups, las redes sociales aplicadas a los negocios, las tecnologías web, etc.

Pero creo que es mejor que me conozcas en mi vídeo de presentación.

Experiencia profesional

Empresa / Organización	Cargo	Localización	Desde-hasta
Esri España	Head of Developers and Startups	Madrid	Marzo 2014 - Actualidad
Upplication	Product manager y desarrollador frontend	Madrid	Noviembre 2013 - Marzo 2015
Universidad Europea de Madrid	Profesor del MOOC: Innovación y emprendimiento	Madrid	Noviembre 2013 - Marzo 2013
Ploqq	Product manager y frontend developer	Madrid	Diciembre 2012 - Julio 2013
Google Developers Group	Coordinador nacional	Granada	Septiembre 2012 - Enero 2013
Ingenio Marketing	Socio y asesor	Granada	Mayo 2012 - Diciembre 2012
Pidecurso.com	Fundador y CEO	Granada	Noviembre 2011 - Diciembre 2012
GeoRedmindMe	Fundador y CEO	Granada	Octubre 2010 - Marzo 2012
Asociación de Webmasters de Granada	Presidente	Granada	Marzo 2009 - Marzo 2011

Habilidades y conocimientos

- **Desarrollo web:** HTML5, CSS3, JavaScript, Python+Django, LESS & SASS, PHP, SEO, Node.js, MySQL, AngularJS, jQuery, Bootstrap, Dojo, responsive design, WordPress, usabilidad, CoffeeScript, etc.
- **Analítica web/Métricas:** Google Analytics, Mixpanel, segment.io, etc.
- **Gestión:** SCRUM, gestión de comunidades y organización de eventos, gestión de equipos, etc.
- **Emprendimiento:** startups, Lean Startup, etc.
- **Aplicaciones móviles:** Phonegap
- **Marketing:** email marketing, redes sociales, gamificación, nurture marketing, lead scoring, etc.
- **Sistemas de información geográfica (GIS):** ArcGIS API's.

Educación

Título	Entidad	Promoción
Master of Business Administration (MBA)	EOI Escuela de Negocios	2012
Curso: creación de empresas	EOI Escuela de Negocios	2009
Psicología de la publicidad, marketing y consumo	Facultad de Psicología, Universidad de Granada	2009
Master en informática (Erasmus)	Universiteit van Amsterdam	2008
Master en informática (Erasmus)	Universitetet i Bergen, Noruega	2006
Ingeniería Superior en Informática	ETSIIT, Universidad de Granada	2002

Perfiles sociales

También puedes encontrarme en:

- Raulfimenez.info
- Github
- Twitter
- LinkedIn
- Youtube
- Slides.com
- Dribbble
- Slideshare
- Google Plus
- Pinterest
- Xing
- Skype
- Facebook
- CodeSchool
- Google

Contactar

Para:
Resuelve este CAPTCHA para saber mi email: h...@gmail.com
Temática: Asunto: Cuerpo: Enviar
Cualquier persona puede desarrollar una página como esta a partir de los conocimientos adquiridos en el curso HTML5 desde cero.
W3C HTML5 W3C CSS

Así que vamos a empezar por ver los elementos HTML que nos faltan por aprender para poder llegar a hacerla.

Etiquetas - Parte 2

Ahora vamos a ver las etiquetas básicas para trabajar con: enlaces o hipervínculos, imágenes, tablas, formularios, separadores y otras consideraciones.

Esta vez tampoco veremos todos los atributos posibles aunque añadiré enlaces a la documentación:

Enlaces o hipervínculos

Una de las características más destacables de HTML es la posibilidad de enlazar unas páginas con otras, para hacer esto utilizamos el elemento "**a**" con el atributo "**href**" (Hypertext Reference). Por ejemplo:

```
<a href="http://www.cursohtml5desdecero.com">Curso de HTML5 desde cero</a>
```

Hay 3 tipos de enlaces:

- **Absoluto**: es un enlace que incluye todas las partes de una URL como vimos en el capítulo 1:
  `scheme:[//[user:password@]host[:port]][/]path[?query][#fragment]`
- **Relativo**: hace referencia a un recurso que se encuentra en una posición relativa a nuestra URL, así podemos establecer rutas relativas, por ejemplo:

  ```
  <a href="img/imagen1.jpg">enlace a una imagen</a>
  ```

  Donde indicamos que si por ejemplo la URL actual es http://www.cursohtml5desdecero.com/leccion1, la imagen se encuentra en http://www.cursohtml5desdecero.com/leccion1/img/imagen1.jpg, y si queremos hacer referencia a un recurso que se encuentra en un nivel superior del "**path**" lo hacemos usando "**../**", por ejemplo:

  ```
  <a href="../img/imagen1.jpg">enlace a una imagen</a>
  ```

  Que hará referencia a la siguiente URL: http://www.cursohtml5desdecero.com/img/imagen1.jpg (eliminamos **leccion1**).
- **Ancla (o *anchor*)**: a diferencia de los dos anteriores, este enlace se utilizar para indicar un elemento dentro de la misma página que estamos viendo. Para ello tenemos que explicar un nuevo tipo de atributo que tienen todos los elementos en HTML, el **id**

(*unique identifier*), como su nombre indica es un identificador único y por tanto no podemos ponerle a dos elementos HTML el mismo **id**. Luego para añadir un hipervínculo a este elemento sólo tenemos que establecer el atributo **href** del enlace al **id** del elemento precedido de una almohadilla (#), por ejemplo:

```
<a href="#leccion1">Lección 3</a>
...
<!-- aquí vendría todo el contenido -->
<h2 id="leccion1">Lección 3</h2>
```

Esto estamos acostumbrado a verlo en la Wikipedia, por ejemplo: https://en.wikipedia.org/wiki/Hyperlink#Hyperlinks_in_HTML

> **Ver**: Lección 2 - Snippet 6

Imágenes

Para mostrar una imagen en una página tenemos dos formas de hacerlo, una es usando el elemento **img** y otras es mediante CSS (que veremos en el capítulo correspondiente).

Esta etiqueta sólo requiere de dos atributos obligatorios que son: **src** (de *source*) y el otro es **alt** (de *alternative*), por ejemplo:

```
<img src="http://www.cursohtml5desdecero.com/assets/images/html-css-js.png" alt="Logo de HTML5">
```

Como podemos deducir del código anterior, el atributo **src** lo usaremos para indicar la URL (absoluta o relativa) a la imagen, y **alt** como el texto (alternativo) que mostrará el navegador en caso de no encontrar la imagen[1].

> **Ver**: Lección 2 - Snippet 7

Tablas

Podemos crear tablas en HTML usando el elemento **table**[2]. Para ello como mínimo tendremos que indicar las filas y las columnas usando los elementos **tr** (*table row*) y **td** (*table data*) respectivamente, así por ejemplo:

```
<table>
    <tr>
        <td>Fila 1, columna 1</td>
        <td>Fila 1, columna 2</td>
    </tr>
    <tr>
        <td>Fila 2, columna 1</td>
        <td>Fila 2, columna 2</td>
    </tr>
</table>
```

Que daría un resultado como el siguiente:

Fila 1, columna 1	Fila 1, columna 2
Fila 2, columna 1	Fila 2, columna 2

Como podemos comprobar las columnas (**td**) siempre van contenidas dentro de las filas (**tr**). En caso de querer agrupar celdas de una misma fila o columna lo haremos así:

- Misma fila: la usaremos el atributo **colspan** (*column span* = número celdas a abarcar)
- Agrupar dos celdas de una misma columna usaremos el atributo **rowspan** (*row span* = número de celdas a abarcar)

Por ejemplo:

```
<table>
    <tr>
        <td>Fila 1, columna 1</td>
        <td colspan="3"> Fila 1, columnas 2, 3 y 4</td>
    </tr>
    <tr>
        <td rowspan="2">Fila 2, columna 1 + Fila 3, columna 1</td>
        <td>Fila 2 columna 2</td>
        <td>Fila 2 columna 3</td>
        <td>Fila 2 columna 4</td>
    </tr>
    <tr>
        <td>Fila 3 columna 2</td>
        <td>Fila 3 columna 3</td>
        <td>Fila 3 columna 4</td>
    </tr>
</table>
```

Quedando algo como esto:

<table>
<tr><td>Fila 1, columna 1</td><td colspan="3">Fila 1, columnas 2, 3 y 4</td></tr>
<tr><td rowspan="2">Fila 2, columna 1 + Fila 3, columna 1</td><td>Fila 2 columna 2</td><td>Fila 2 columna 3</td><td>Fila 2 columna 4</td></tr>
<tr><td>Fila 3 columna 2</td><td>Fila 3 columna 3</td><td>Fila 3 columna 4</td></tr>
</table>

Como podemos ver el atributo afecta a las celdas de las siguientes columnas/filas y el valor indica cuántas celdas debe abarcar.

> **Ver**: Lección 2 - Snippet 8

Formularios

Vamos a hablar muy brevemente de los formularios, algunos de los elementos y de sus propiedades:

- **form**: será el elemento padre que anide todos los elementos HTML que representarán los campos de nuestro formulario, incluído el botón de enviar.
  - **action**: indica la URL a la que se enviará la petición HTTP con toda la información del formulario
  - **method**: indica si la petición HTTP será *GET* o *POST*
- **input**: permite introducir diferentes *tipos* de campo de formulario en base al valor del atributo **type**. En función del valor indicado en **type** dispondremos de unos atributos u otros (en total hay 30, pero no todos aplican a todos los casos):
  - **type** (obligatorio): este valor puede tener muchos valores: *text*, *email*, *checkbox*, *color*, *date*, *file*, *hidden*, etc. en función del tipo de campo que queramos, los nombres son bastante auto-explicativos.
  - **id**: este atributo es obligatorio si en el elemento **label** tiene un atributo **for**, en tal caso deberá contener un identificador único[3] en la página.
  - **name**: este atributo es opcional y representa el nombre asignado al campo cuando se envíe la petición HTTP.
  - **value**: este valor es opcional pero representa el valor que se asignará al campo cuando se envíe la petición HTTP.
- **select**: nos permite crear una lista desplegable de opciones, cada opción estará contenida como hija dentro de un elemento **option**.
  - **id**: igual que el elemento **input**
  - **name**: igual que el campo **input**
- **option**: nos sirve para "encapsular" cada opción de la lista.
  - **value**: igual que el atributo **value** del campo **input**.
- **textarea**: representa un campo que nos permite introducir textos con saltos de línea incluidos, normalmente se usa cuando hay que introducir: descripciones, biografías,

etc.
    - **id**: igual que el elemento **input** y **select**.
    - **name**: igual que el campo **input** y **select**.
- **label**: se usa para especificar la etiqueta (o nombre) del campo del formulario.
    - **for**: tiene que tener el mismo valor que el atributo **id** del campo (input, select o textarea) al que hace referencia la etiqueta.
- **button**: representa un botón y el texto del botón está representado por su contenido.
    - **type**: define el comportamiento del botón cuando está activado y puede contener tres valores: *submit*, *reset*, *button*

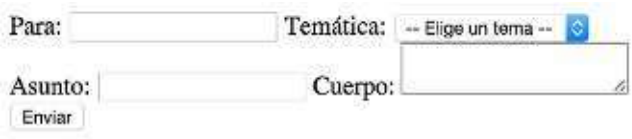
Existen muchos otros atributos que no veremos dado que no les daremos uso en este curso ya que para sacarle el máximo provecho sería necesario tener conocimientos en programación.

Por último añadir que el elemento **input** no requiere una etiqueta de cierre (o lo que es lo mismo, que está autocontenido).

Aquí tenemos un ejemplo de formulario:

```
<form action="miScript.php" method="GET">
    <label for="to">Para:</label>
    <input id="to" type="email">

    <label for="topic">Temática: </label>
    <select id="topic" name="topic">
      <option>-- Elige un tema --</option>
      <option value="proposal">Propuesta</option>
      <option value="report">Reporte</option>
      <option value="other">Otro</option>
    </select>

    <label for="subject">Asunto: </label>
    <input id="subject" name="subject" type="text">

    <label for="body">Cuerpo:</label>
    <textarea id="body" name="body"></textarea>

    <button type="submit">Enviar</button>
</form>
```

Que nos dará como resultado algo así:



Como ves los estilos por defecto serán muy poco atractivos, pero no te preocupes, ya aprenderemos a solucionar esto usando CSS.

Por último comentar que *en muchos de los elementos*[4] podemos añadir (opcionalmente) otros atributos como:

- **required** a un elemento para que el navegador se encargue de validar que este campo está relleno
- **readonly** si queremos que sea de sólo lectura
- **placeholder** si queremos que aparezca un texto de ayuda para rellenar el campo
- **value** para introducir un valor por defecto en el campo

Por ejemplo:

```
<label for="to">Para:</label>
<input id="to" type="email" placeholder="tu@correo.com" required>

<label for="subject">Asunto: </label>
<input id="subject" type="text" value="Formulario de contacto" readonly>
```

> **Ver**: Lección 2 - Snippet 9

Separadores

Existe un elemento que nos permite añadir un separador (una línea horizontal), este elemento es **hr**.

> **Ver**: Lección 2 - Snippet 10

Otras consideraciones

Para terminar este capítulo hay una última cosa que me gustaría comentar:

- En HTML se ignoran todos los espacios a partir del primero, por lo tanto nunca podremos (*ni se debe*) alinear usando espacios.
- Las entidades HTML (*HTML entities*) se usan para pintar palabras, caracteres o símbolos reservados o que puede que no tengas en tu teclado como por ejemplo: <, >, ©, &, €, etc.
  Existen 1446 entidades reservadas que puedes consultar en la página del W3C. (**ver**: Lección 2 - Snippet 11)

Para representar la entidades HTML se usa el siguiente formato:

```
&código_de_la_entidad;
```

Veamos un ejemplo para entender mejor por qué existen las entidades HTML y cómo se usan.

Imaginemos que estamos escribiendo un libro como este y necesitamos hablar sobre la etiqueta `<hr>`:

```
<p>La etiqueta <hr> se utiliza para ...</p>
```

En este caso cuando el navegador esté intepretando el código HTML encontrará "**<hr>**" y en lugar de mostrar la cadena de texto (que es lo que queremos) nos mostrará un separador horizontal.

Para evitar este problema usaríamos el siguiente cógido HTML:

```
<p>La etiqueta &lt;hr&gt; se utiliza para ...</p>
```

En este caso hemos modificado el símbolo "menor que" (**L**ower **T**han) por la entidad HTML **lt** y "mayor que" (**G**reater **T**han) por **gt**, así el navegador podrá representarlo sin ningún problema.

En alguna ocasión puede que navegues por una página con una codificación (encoding) que no soporte los acentos agudos (á, é, í, ó, ú), en ese caso usarán las entidades HTML (`&aacute;, &eacute;, &iacute;, &oacute;, &Uacute;`) para representarlos. Por cierto: "acute" en inglés significa "agudo".

Aclaraciones:

1. Puede que no se encuentre la imagen porque alguien la borre del servidor o porque nos equivoquemos al introducir la URL.
2. Las tablas sólo deben usarse para mostrar datos que tengan sentido en una tabla y nunca para maquetar.
3. Con esto nos referimos a un nombre (o cadena de texto) que no contenga ningún otro elemento, por ejemplo no puede haber dos elementos con **id="email"**.
4. En la documentación del W3C podemos ver qué atributos admite cada elemento: input, textarea, select, etc.

Anidación - Parte 2

En el primer capítulo de HTML vimos que las etiquetas se pueden anidar, pero comentamos que no todas las etiquetas son anidables entre sí, por ejemplo **esto sería incorrecto**:

```
<body>
    <head></head>
</body>
```

Ya que una etiqueta **body** no puede contener a otra **head**, pero... ¿cómo podemos saber qué etiquetas son anidables?.

Con la práctica aprenderá algunas anidaciones que están prohibidas y desarrollarás una capacidad de razonar algunas anidaciones obvias, pero al principio podrás servirte de tres recursos principalmente:

- La pestaña **Elements** del navegador. Como decíamos el navegador es un programa bastante complejo, y entre otras cosas que se encarga es de construir el DOM. Si durante el proceso de construcción del DOM el navegador se encuentra una anidación incorrecta intentará resolverla. Por eso si inspeccionamos el DOM de nuestra página, podremos ver si el propio navegador ha encontrado etiquetas mal anidadas y nos ha modificado el código.
- El validador de código del W3C que veremos cómo usar en el siguiente apartado.
- Pero nuestro principal recurso debe ser la especificación de HTML5 del W3C, ahora veremos cómo usarla.

Usar la especificación del W3C

En la descripción de todo elemento HTML nos encontraremos un apartado que se llama **Content model** que especifica qué tipo de etiquetas se pueden anidar, por ejemplo: **Metadata content**, **Flow content**, **Sectioning content**, **Heading content**, **Phrasing content**, ...

Por ejemplo el elemento "**li**" que usábamos para especificar un elemento de una lista (List Item) indica que su "Content model" es "Flow content", si hacemos clic en el enlace veráss que esto significa que el elemento soporta casi todos los elementos: a, area, article, b, blockquote, br, div, em, footer, form, h1, h2, h3, h4, h5, h6, header, hr, i, iframe, img, input, etc.

Sin embargo si vas a la especificación del elemento "**p**" verás que su modelo es de "Phrasing content", que admite muchas etiquetas pero por ejemplo no admite: "ul", "ol", "hr", etc.

Esta es la mejor forma de saber qué etiquetas son anidables y cuales no.

Validación

Que el código se muestre en nuestro navegador web como queríamos no implica necesariamente que lo hayamos escrito bien. En muchas ocasiones el navegador es capaz de detectar errores humanos y corregirlos de manera automática para que el usuario vea bien la página, pero esto no es siempre así. Si queremos asegurarnos de que nuestra página está correctamente escrita podemos usar el Validador de HTML del W3C, que además en caso de encontrar errores nos dará pistas sobre cómo resolverlos.

Si abres el enlace verás que tienes 3 formas de validar código:

1. **Validate URI**: que te permite introducir la URL de una página para comprobar su código. Como nosotros aún no hemos alojado nuestra página en ningún servidor web no usaremos esta opción (aún).
2. **Validate by File Upload**: que nos permite subir un fichero HTML
3. **Validate by Direct Input**: que nos permite introducir el código dentro de un elemento **textarea**.

Usaremos las opciones 2 y 3 hasta que en la siguiente lección aprendamos cómo alojar nuestra página en un servidor web accesible desde Internet.

Aunque el naegador sea capaz de solucionar algunos de nuestros errores, es importante crear código válido porque:

- Aunque el navegador sea capaz de resolver un problema, no todos los navegadores son iguales, y por tanto puede que algunos no lo resuelvan o la solución que aplique provoque un efecto no deseado.
- Hace tu página más accesible como veremos a continuación.

Accesibilidad

Vamos a ver este apartado muy por encima, pero no quería dejarlo completamente de lado.

Existe una iniciativa que tiene como objetivo hacer la web más accesible, especialmente para personas cualquier tipo de discapacidad: visual (completa o parcial), auditiva, cognitiva, etc. Esta iniciativa se llama: **Web Accessibility Initiative - Accessible Rich Internet Applications (WAI-ARIA)** y están creado un estándar que actualmente se encuentra en la versión 1.1.

Cualquiera puede aplicar algunas prácticas de accesibilidad con poco esfuerzo, como por ejemplo usar los landmarks, que no son más que atributos que añadimos a las etiquetas para indicar las partes más relevantes de nuestra página como: el menú de navegación, el área de contenido principal o contenido complementario[1].

Si hablas inglés te recomiendo ver esta charla de Leonie Watson, una mujer ciega explicando la importancia de la accesibilidad.

Por último, si estos argumento no son suficientes, me gustaría añadir que haciendo una página accesible hacemos que esta esté mejor posicionada por los buscadores, dado que a las arañas[2] de los motores de búsqueda a fin de cuentas tienen una forma de navegar[3] por nuestra página igual que las personas con problemas de accesibilidad.

Aclaraciones:
1. Pequeño vídeo explicativo en inglés sobre cómo usar los landmarks.
2. Las "arañas" (spiders), bots o web crawlers, son es el nombre convencional que le damos a los programas que se dedican a "rastrear" por Internet y que usan entre otros los grandes buscadores para buscar nuevo contenido y cualificarlo para después devolverlo en sus resultados de búsqueda.
3. Sitio web explicando cómo funciona Googlebot (la araña de Google).

Convenciones

Antes de terminar me gustaría explicarte algunas de las principales convenciones o buenas prácticas que deberemos de tener en cuenta a la hora de escribir código HTML:

- Los nombres de los elementos HTML y sus atributos se deben escribir en minúsculas

```
<!-- MAL -->
<p>
  <IMG SRC="html5.gif" ALT="Logo HTML5">
</p>
<!-- BIEN -->
<p>
  <img src="html5.gif" alt="Logo HTML5">
</p>
```

- Los valores de los atributos en HTML deben ir entre comillas dobles:

```
<!-- MAL -->
<img src='html5.gif' alt='Logo HTML5'>
<!-- BIEN -->
<img src="html5.gif" alt="Logo HTML5">
```

- La indentación se debe hacer con 2 espacios (prácticamente todos los editores de código permite configurar este valor).

```
<p>
  <img src="html5.gif" alt="Logo HTML5">
</p>
```

- No introducir espacios antes o después del signo "igual":

```
<!-- MAL -->
<img src = "html5.gif" alt = "Logo HTML5">
<!-- BIEN -->
<img src="html5.gif" alt="Logo HTML5">
```

- Usar UTF-8 como encoding.
- No cerrar elementos autocontenidos, por ejemplo usa `<br>` en lugar de `<br/>`
- Evita el uso de estilos en línea (atributo `style` lo veremos en el siguiente apartado)
- Evita el uso de entidades HTML siempre que sea posible (salvo por ejemplo para `<` y `&`)

```
<!-- MAL -->
<h1>P&áacute;gina sobre &lt; HTML5 &amp; CSS3</h1>
<!-- BIEN -->
<h1>Página sobre &lt; HTML5 &amp; CSS3</h1>
```

- Especifica el atributo **lang** en el elemento **html**:

```
<html lang="es">
```

- Especifica siempre el atributo **for** cuando añadas un elemento **label**

```
<label for="field-email">email</label>
<input type="email" id="field-email" name="email" value="" />
```

- Internet Explore soporta el uso de una etiqueta de compatibilidad **meta** indicando cómo tratar el código, usar siempre que se pueda:

```
<meta http-equiv="X-UA-Compatible" content="IE=Edge">
```

Esta recopilación ha sido extraída de algunas guías de estilo.

- HTML(5) Style Guide and Coding Conventions
- HTML coding standards - CKAN
- Google HTML/CSS Style Guide
- Code Guide by @mdo

Otras convenciones

Nombres de ficheros

Es recomendable seguir las siguientes convenciones:

- Establecer los nombres de los ficheros en minúsculas, Windows no hace distinción entre mayúsculas y minúsculas pero otros sistemas sí, y esto puede provocar que una ruta funcione en un sistema operativo pero no en otro. Por ejemplo si tenemos un fichero llamado **Logo_HTML5.jpg** y una página que hace referencia a él con `<img src="logo_html5.jpg" alt="Logo HTML5">`. Esto funcionará en Windows pero en un sistema basado en Unix (Linux o Mac) no funcionará.
- Dale un nombre que represente lo que contiene, esto no sólo por usabilidad sino por posicionamiento (SEO[1])
- En lugar de espacio usa un guión "-".
- Nunca uses acentos ni caracteres especiales: ñ, ",", etc.

Extensiones de ficheros

Es recomendable que cada tipo de fichero tenga una extensión:

- HTML: "**.html**"
- JPEG: "**.jpg**"
- GIF: "**.gif**"
- PNG: "**.png**"

Estructura de directorios

Conforme crezca tu proyecto agradecerás tener una estructura lógica que te ayude a organizar todos los ficheros. Basándome en esta respuesta en Stackoverflow[2] te recomiendo seguir esta estructura[3]:

```
resources/
    css/
        main.css
    images/
        logo-html5.jpg
    js/
vendors/
    jquery/
        jquery.min.js
index.html
```

Donde:

- **resources**: es un directorio que incluye los **elementos que tú has creado** y que contiene tantos directorios como tipos de recursos (css -> estilos, images -> imágenes, js -> JavaScript).
- **vendors**: para almacenar recursos creados por terceros
- Y en el fichero raíz colocar los ficheros HTML que necesites.

Aclaraciones:
1. SEO es el acrónimo de Search Engine Optimization, o lo que viene a ser lo mismo: Optimización en Motores de Búsqueda. Por ejemplo, estableciendo correctamente los nombres de nuestra imágenes, es más probable que aparezcamos en buenas posiciones en Google Images.
2. Stackoverflow es uno de los sitios de referencia donde podrás encontrar muchas dudas de programación, lo que lo hace realmente interesante es el sistema de valoraciones que permite discernir las "buenas" de las "malas" respuestas.

3. No todos los proyectos se deben organizar igual, en muchas ocasiones dependerá de las tecnologías que estés usando, pero para este curso esta estructura será suficientemente buena.

Errores frecuentes

Este es un listado de alguno de los errores más frecuentes en HTML:

- No poner el encoding UTF-8 hará que algunos caracteres se muestren de manera extraña
- Poner dos identificadores iguales (suele pasar al copiar y pegar código). Esto nos dará problemas de validación y al intentar acceder al elemento usando JavaScript
- Introducir & en las URLs; en su lugar se debe usar &
- Anidamiento incorrecto, ya sea por no cerrar tags en el orden correcto como por anidar elementos de bloque en elementos en linea, por ejemplo: `<a href="#"><h2>Título</h2></a>`
- Utilizar los elementos `<b>` o `<i>` para darle estilo
- Usar múltiples
  consecutivamente en lugar de usar estilos

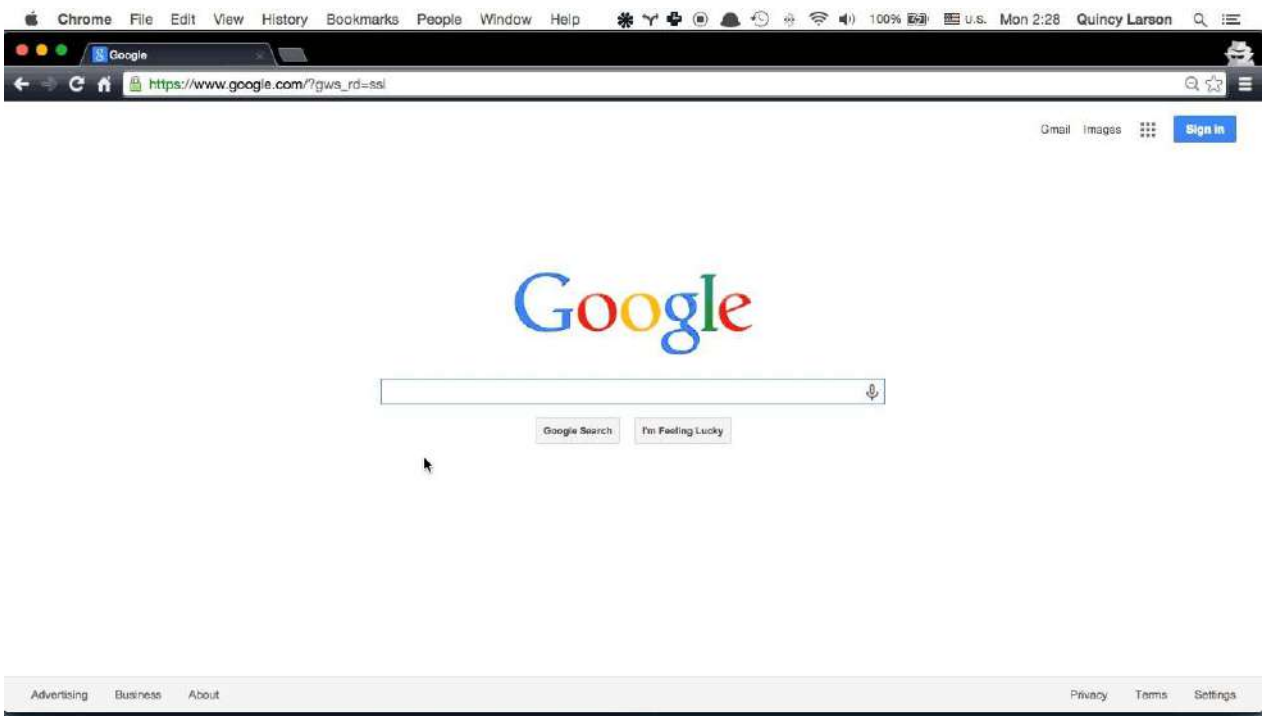
Estos son sólo algunos errores frecuentes, pero en ningún libro, manual, tutorial o curso encontrarás todos los errores que te pueden suceder, por eso es importante que aprendas a buscar las soluciones a los problemas que te vayan surgiendo, mis consejos:

- Lee **atentamente** el error
- Usa Google para buscar tu error (a ser posible busca en inglés)
- Intenta reducir la frase a las palabras clave como el lenguaje, el número de error, ...

Cuando encuentres alguna página donde parezca haber una respuesta, fíjate en:

- Que la fecha de la respuesta sea relativamente nueva (no más de 2 años)
- Busca en la documentación oficial
- Si estás en Stackoverflow revisa el número de valoraciones positivas de la respuesta

Fíjate en la siguiente animación:



Aquí se sigue el proceso recomendado salvo que se ha seleccionado una respuesta que tiene una sola valoración.

Cómo pedir ayuda

Lo más normal es que cualquier error que te ocurra cuando estés empezando ya lo haya preguntado otra persona, insiste en la búsqueda y si después de un buen rato (~30min) no encuentras el error pregunta en cualquier foro y comenta lo que te ocurre (en inglés), pero unos consejos:

- Especifica que ya has buscado antes el error
- Explica cómo lo has buscado (*por si te pueden dar algún consejo para mejorar tu forma de buscar*)
- Y finalmente haz la pregunta dando el máximo número de detalles

Es importante que cuando preguntes algo el resto vea que te has molestado en investigar previamente y también que te esfuerzas en explicar lo que te pasa, sino es probable que alguien te de una mala respuesta porque piense que no te has molestado en solucionar el problema por ti mismo.

Ejercicio

Recursos

- Can I Use?
- Mozilla CDN

Github

En esta lección vamos a aprender a usar Github, una de esas herramientas que no conoces hasta que alguien te habla maravillas de ella, y **cuando la entiendes se convierte en una herramienta sin la que no puedes vivir**.

Para que te hagas una idea rápida, Git es un software de control de versiones, lo que significa que nos va a ayudar guardar "una foto" (versión) de cómo está nuestro código en un momento dado y si después de hacer varios cambios nos arrepentimos, o vemos que algo se ha roto y no sabemos solucionarlo, poder ver gráficamente qué lineas y qué ficheros han cambiado para ayudarnos a encontrar el error, y si fuese necesario deshacer los cambios hasta aquel momento.

Por tanto con Git se acabó el tener múltiples copias de una carpeta "por seguridad", es una forma mucho más cómoda de evolucionar el código.

Github es una compañía americana que ha creado una web donde podremos mantener una copia del código que estemos gestionando con Git y que además nos ayuda a hacer más sencillo colaborar y compartir este código.

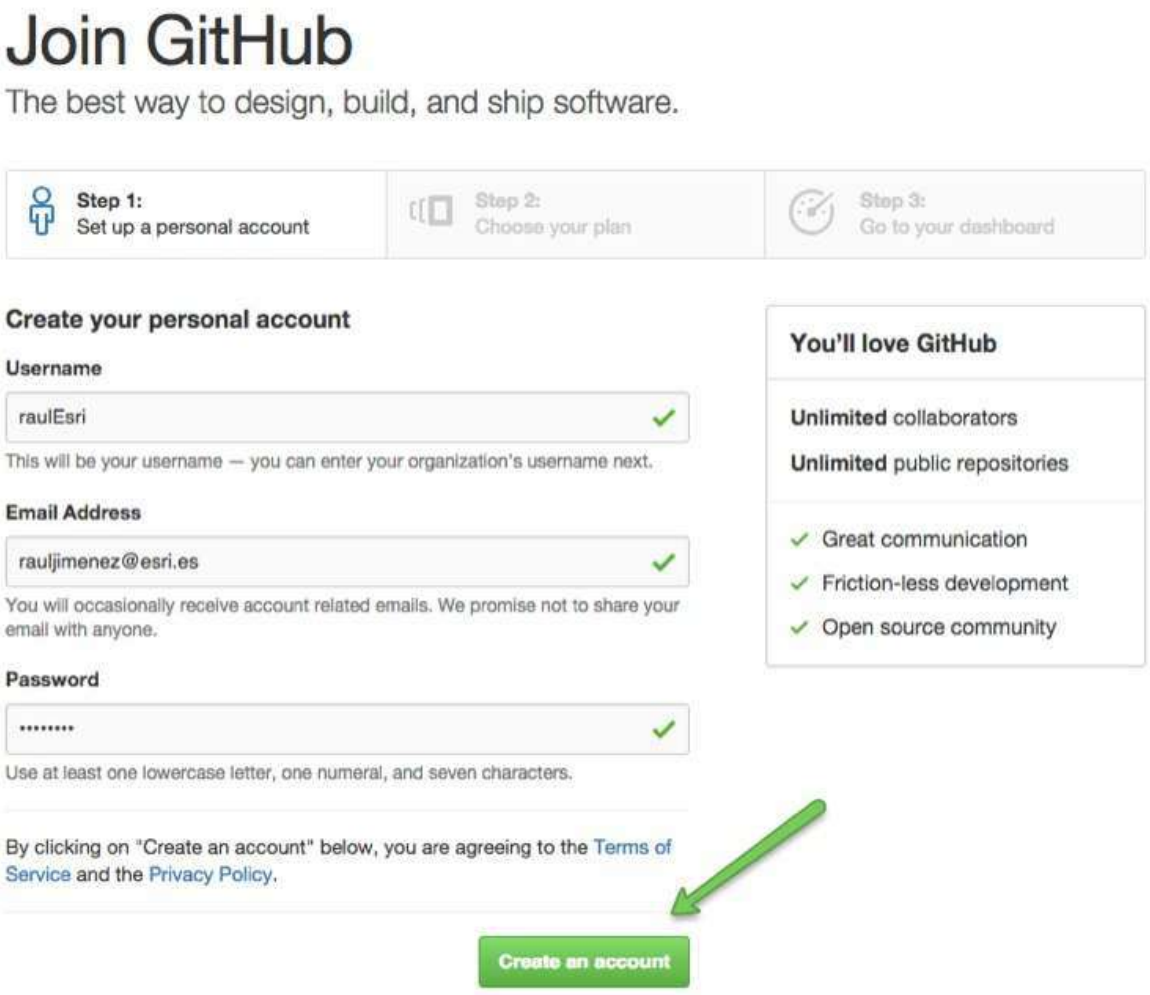
Vamos a empezar a crear una cuenta y te iré explicando los conceptos que te van a hacer falta entender para este curso sobre la marcha.

Aclaraciones:
1. GitHub, Inc. desarrolla una plataforma web que tiene viene el mismo nombre y que permite trabajar colaborativamente respositorios en Git. La compañía fue fundada en 2008 y ya se ha convertido en el estandar mundial de facto para proyectos de software libre, desvancando a otras como Google Code o Sourceforge. En este tiempo ha recibido 350 millones de dolares de inversión.

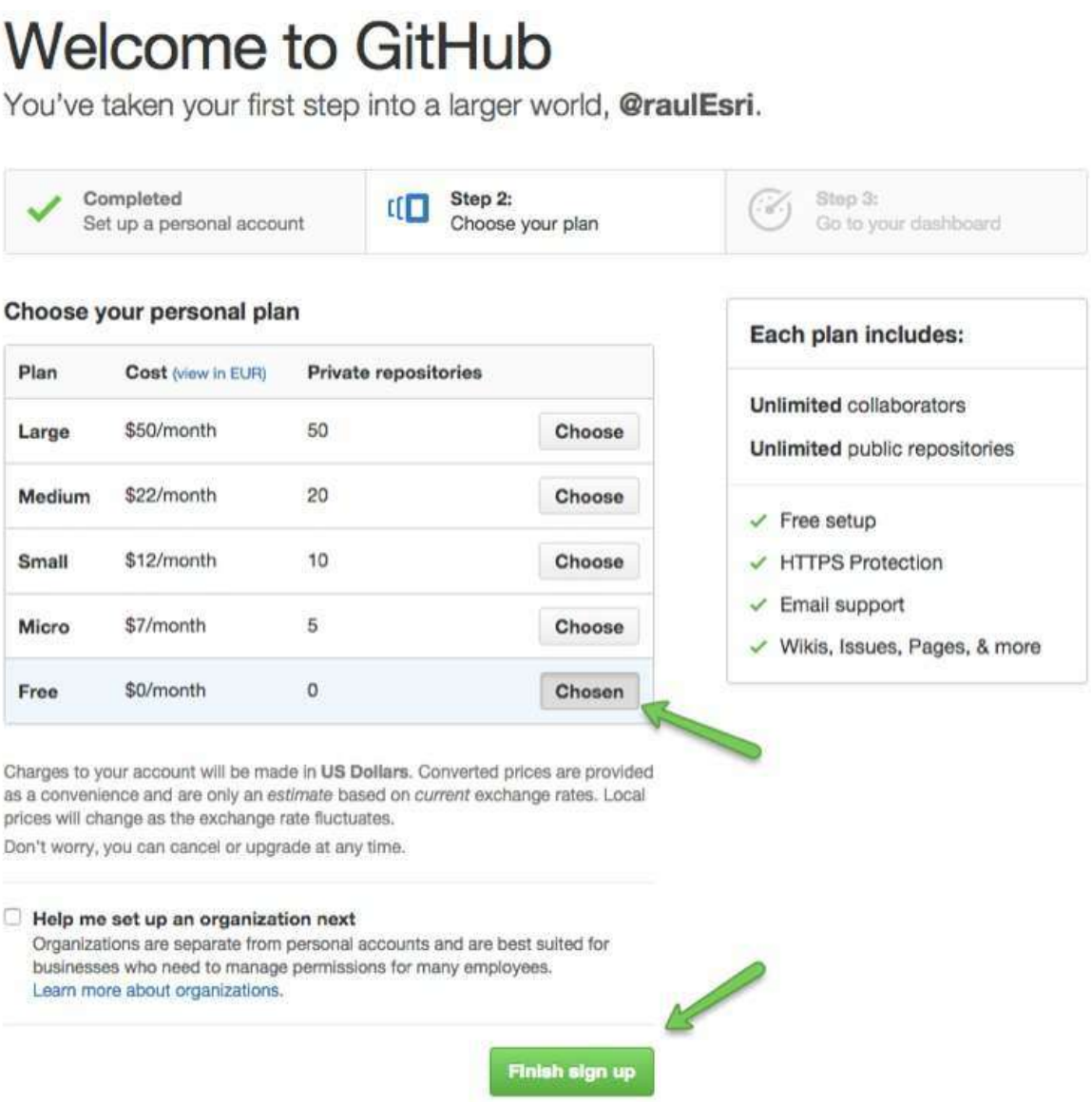
Configurar nuestra cuenta

Vamos a empezar por crear una cuenta en Github.

Primero vamos a la sección "Signup" y completamos nuestros datos:



> **Nota**: El username definirá la URL de nuestra cuenta: http://github.com/**username**

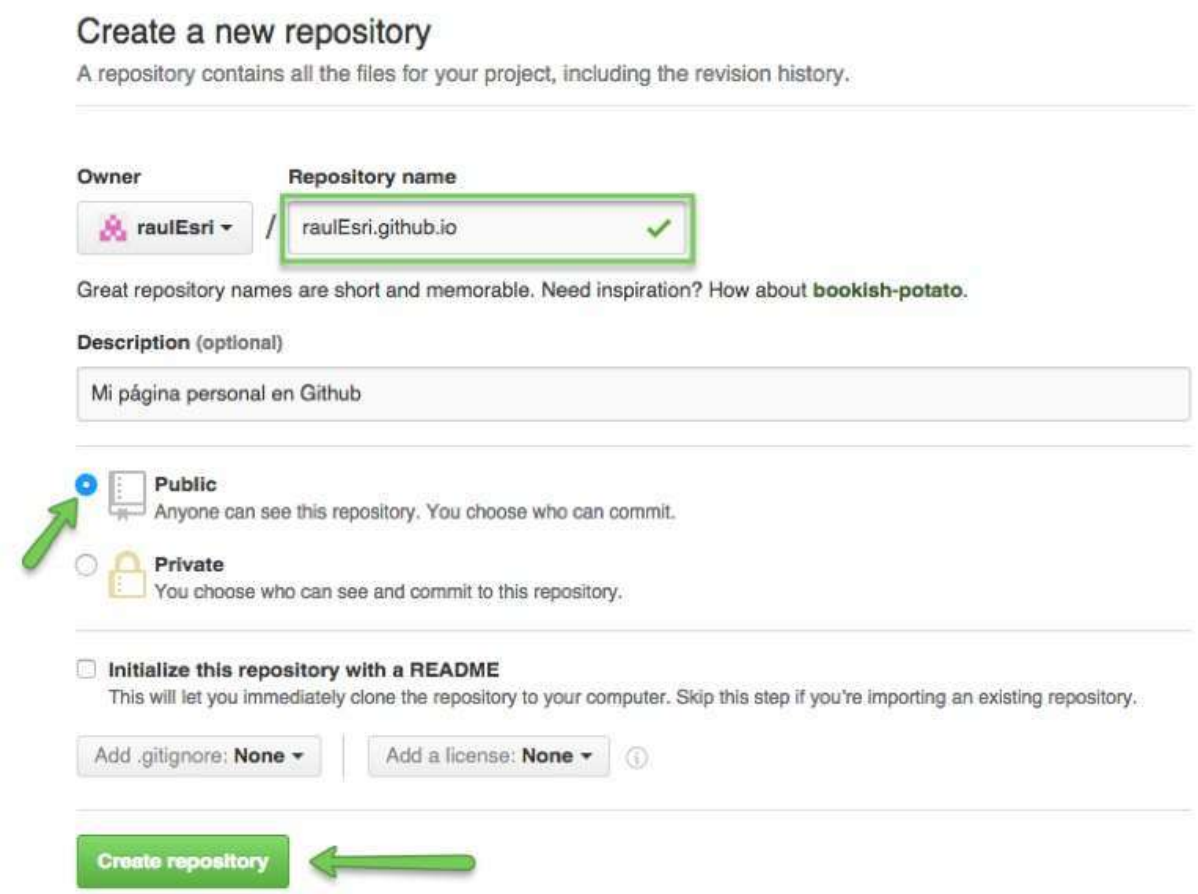


A continuación elegimos el plan gratuito:

Ahora vamos a crear un nuevo repositorio. Crearemos **un repositorio para cada proyecto** que vayamos a crear, para que nos hagamos una idea tendremos un repositorio por cada carpeta raíz que tengamos en nuestros disco duro.

A este repositorio le daremos un nombre que Github tratará de manera especial[1], el nombre debe ser: **username**.github.io, en mi caso raulEsri.github.io.

Luego puedes ponerle una descripción del contenido del proyecto, por ejemplo: *Mi página personal en Github*. Y seleccionamos que será un proyecto público (el código será accesible por cualquier persona).



Y ya está, ¡enhorabuena!, ya tienes una cuenta en Github. Cualquier persona podrá ver todos tus repositorios públicos en: http://github.com/**Username**, en mi caso mi repositorio sería: ~~http://github.com/raulEsri~~, mi repositorio real es http://github.com/hhkaos (el otro lo he creado simplemente para hacer este tutorial).

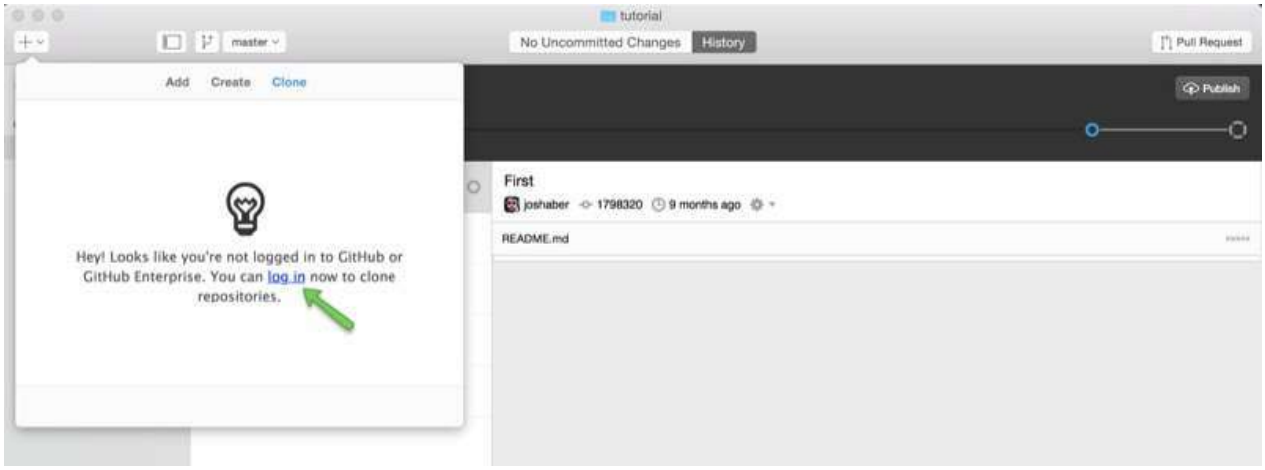
Aclaraciones:
1. El código que tengamos en este repositorio nos los servirá usando un servidor HTTP a través del dominio http://**username**.github.io, por ejemplo en mi caso el repositorio de web cuenta personal es http://hhkaos.github.io y el código está en https://github.com/hhkaos/hhkaos.github.io.

Enviar y recibir cambios

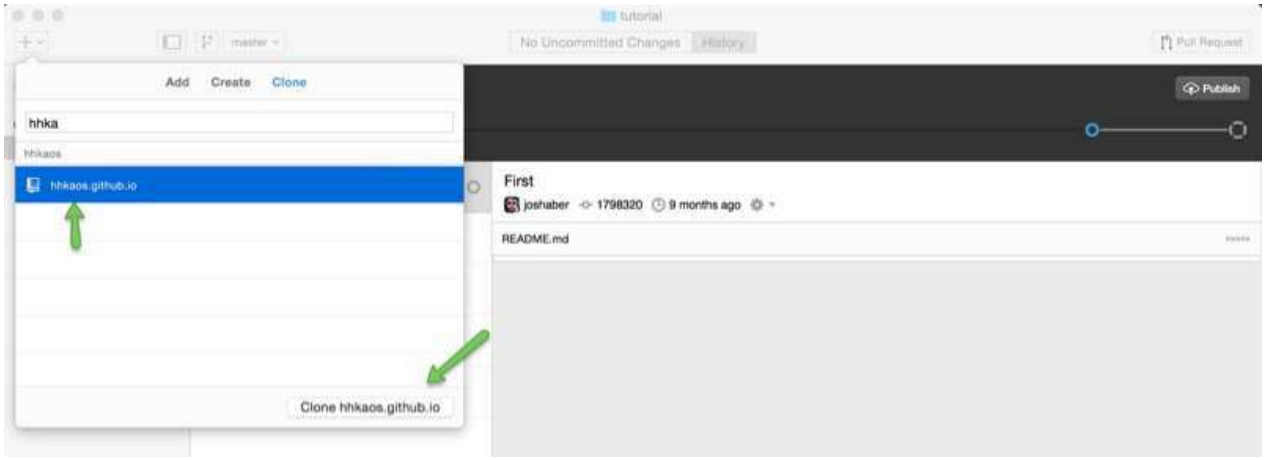
Instalar y configurar Github Desktop

Vamos a empezar por descargar, instalar y configurar Github Desktop que es una herramienta gráfica[1] que nos va a ayudar a gestionar las versiones.

Una vez descargado e instalado lo abrimos y nos identificamos con nuestra cuenta.



Una vez hecho volvemos a la opción de clonar y seleccionamos el repositorio que acabamos de crear.



Esto nos pedirá que seleccionamos una carpeta en nuestro disco duro donde se va a clonar el repositorio (se creará una carpeta nueva en nuestro disco duro lista para controlar los cambios). La seleccionamos y le damos a "OK".

Una vez clonada el repositorio vamos a ver una forma básica de usarlo.

Enviar y recibir cambios

Pendiente

Ver el histórico de cambios

Pendiente

Aclaraciones:
1. Mi recomendación es que hagas el curso gratuito Try Git de CodeSchool para aprender a usar Git desde la línea de comandos cuando te hayas acostumbrado a usar Git (**yo nunca uso ninguna herramienta gráfica**).

Funcionalidades

Issues, readme, etc.

Publicar una web en Github

Colaborar con un proyecto

fork, pull request, issues

WIP (Work In Progress)

Esta sección aún está pendiente de escribir

CSS: Primero pasos

CSS es el acrónimo de Cascading Style Sheet; este también un lenguaje de marcado que nos permite aplicar estilos a nuestros elementos HTML.

En este capítulo vamos a aprender:

- Qué es CSS y cómo lo podemos usar para aplicar estilos
- Qué estilos podemos aplicar a cada elemento HTML
- Cómo combinar HTML y CSS
- Cómo funciona la herencia de estilos

Introducción a CSS

Por si aún no lo sabes, me gustaría empezar comentándote que a los profesionales que diseñan páginas usando HTML y CSS se les suele llamar **maquetadores web**. Si a un maquetador web le añadimos conocimientos en programación con JavaScript ya podemos hablar de un **front-end developer**.

Antes de empezar a hablar de código me gustaría explicarte dos aspectos muy importantes que venimos sufriendo históricamente los que nos dedicamos a esto y sitiéndolo mucho, tú tampoco te vas a escapar.

Inconsistencia de estilos

No sé si te has dado cuenta, pero aunque tú aún no hayas escrito aún ninguna línea de CSS, la página que has creado ya contenía algunos estilos, los estilos por defecto que incluye el navegador.

Es **muy importante que siempre tengas en cuenta que cada navegador incluye estilos propios por defecto** que aplica a los distintos elementos HTML, por ejemplo:

- Los elementos `<p>` tienen estilos para que se produzca un salto entre el párrafo y los elementos anterior y posterior. (*pero la distancia puede variar entre navegadores*
- El elemento `<strong>` para que se muestre en negrita.
- Los elementos `<li>` para que aparezca un punto a la izquierda y una ligera indentación. (*pero el margen puede variar*)
- Los encabezados `<h1>`, `<h2>`, etc para que se muestren de un tamaño mayor que el del resto (*pero los tamaños pueden variar*).
- Etc

Esto sucede en todos los navegadores, el problema como vemos es que no todos definen los estilos exactamente de la misma manera y si no tienes esto en cuenta desde el principio lo vas a sufrir en el futuro.

Aunque lo lógico sería que todos los navegadores se pusiesen de acuerdo en definir unos estilos por defectos comunes, ya hemos asumido que esto no va a pasar nunca, por este motivo quiero presentarte Reset.css, una hoja de estilos comúnmente usada para

uniformizar los estilos en todos los navegadores, lo que nos ayudará a que el resultado final después de aplicar nuestros estilos sean el mismo independientemente del navegador desde la que la abramos.

Soporte a estándares

Por otro lado quería comentarte que vamos a empezar centrándonos en las propiedades de CSS3 que vienen heredadas de la versión 2.1.

¿Por qué?, la versión resumida es: qué por evitarte quebraderos de cabeza iniciales. La versión extendida la he dejado como un "Anexo - Navegadores y estándares soportados".

Mi primer CSS

Hay múltiples formas de añadir CSS a nuestra página, una forma es utilizando el elemento `<style>` dentro de nuesto HTML, por ejemplo:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Mi primer CSS</title>
    <!-- Aquí definimos los estilo CSS para esta página -->
    <style>
      h1{
        color: red;
      }
    </style>
    <!-- Fin del CSS -->
</head>
<body>
    <h1>Encabezado 1</h1>
</body>
</html>
```

De este modo estamos le indicamos al navegador que queremos que nos coloree el texto de todos los elementos **h1** en color rojo, para ello usamos la propiedad "**color**" y establecemos su valor a "**red**".

Los estilos aplicados a un elemento en CSS siempre tienen que estar envueltos entre llaves ("**{}**"). Por cada propiedad definida (en este caso `color`) tenemos que terminar la línea con punto y coma.

Por tanto los estilos se definen de la siguiente manera:

```
nombre-de-la-etiqueta-html {
    propiedad-css: valor-de-la-propiedad;
    propiedad-css-2: valor-de-la-propiedad-2;
}
```

Una buena práctica es tabular las propiedades de un elemento (igual que se hace con el código HTML).

Siempre que se use el elemento "**style**" debe estar anidado dentro del elemento "**head**", aunque si lo ponemos dentro del "**body**" lo más probable es que funcione bien, pero no sería válido según el W3C y por tanto no pasaría el validador.

Snippets interactivos

Puedes acceder a los snippets interactivos de CSS a través de la siguiente URL:

http://libro.cursohtml5desdecero.com/snippets/css/

Aclaraciones:

1.siendo los que más soportan (ordenados de mayor a menor): Chrome, Firefox, Edge y Safari.

Propiedades

Hemos visto una forma de aplicar estilos a nuestro HTML, en el siguiente capítulo veremos otras formas, pero de momento nos vamos a quedar aquí y vamos a ver en detalle algunas de las propiedades más usadas en CSS.

Modificar el color

La propiedad **color** se puede usar en cualquier elemento, aunque principalmente se usa para modificar el color del texto, esta no es su única función[1].

Existen múltiples **formas de especificar el color**, aquí veremos tres, mediante su:

- **Valor hexadecimal**: #faf o #ffaaff
- **Valor RGB (Red, Green, Blue)**: rgb(255, 160, 255) o rgb(100%, 62.5%, 100%)
- **Valor RGBA (RGB + Alpha)**: rgba(255, 160, 255, 1) or rgba(100%, 62.5%, 100%, 1) s El valor Alpha tiene que estar comprendido entre [0-1] y hace referencia a la transparencia del elemento, siendo 1 = opaco y 0 = transparente.

Ahora vamos a ver tres formas equivalentes de representar el color rojo:

```
h1{
    color: rgb(255, 0, 0);
}
```

```
h1{
    color: #F00;
}
```

```
h1{
    color: rgba(255, 0, 0, 1);
}
```

> **Ejemplo interactivo**: Lección 1 - Snippet 1.

En RGB si indica en cada uno de los valores la cantidad de Rojo (**R**ed), Verde (**G**reen) y Azul (**B**lue) que tiene que combinar. Este valor puede ser un porcentaje o un valor comprendido entre 0 y 255, siendo 255 equivalente a 100%.

Por otro lado comentar que el valor hexadecimal "**FF**" equivale a "**255**". Existen múltiples conversores de decimal a hexadecimal y viceversa, la principal ventaja de usar este formato es que ocupa menos caracteres y por tanto menos espacio en disco (lo cual es óptimo para mejorar los tiempos de carga de un fichero[2]). Por último, cuando un valor hexadecimal contiene 3 caracteres significa que cada uno de ellos se repite dos veces, ej: #faf es igual que #ffaaff.

Cuando lleguemos al apartado de "**Estilos con DevTools**" veremos una manera sencilla de usar un círculo cromático para conocer/elegir el color de un elemento.

Modificar la fuente

Para modificar las propiedades de la fuente (font) tenemos distintas propiedades:

- **font-size**: nos permite especificar el tamaño de la fuente
- **font-style**: nos permite darle estilo a la fuente (p.e.: `normal` o `italic`)
- **font-family**: establece una lista de fuentes (`Arial` , `Helvetica` , `sans-serif` ;)
- **font-weight**: nos permite especificar el ancho de la fuente (`bold` , `400` , `600` , ...)
- **font**: atajo para establecer varias propiedades

> **Ejemplo interactivo**: Lección 1 - Snippet 2.

font-size

En cuanto al tamaño hay varias formas de especificarlo, pero vamos a ver cómo especificar el tamaño usando unidades de longitud, por ejemplo:

- **px**: representan un tamaño absoluto.
- **em**: representan un valor relativo respecto al elemento DOM padre.
- **rem** (root em): representan un valor relativo respecto al elemento `body` .

Un elemento puede tener múltiples estilos como vemos a continuación:

```
h1{
    color: #F00;
    font-size: 16px;
}
```

Al igual que con los colores, veremos cómo jugar con estos valores usando Chrome DevTools.

font-style

Los valores que puede contener son: `normal | italic | oblique | inherit`, "**inherit**" significa que tome el valor del elemento padre, y "**oblique**" e "**italic**" indican que la fuente se muestre el cursiva:

```
h1{
    color: #F00;
    font-size: 16px;
    font-style: italic;
}
```

font-family

Esta propiedad indica la fuente que queremos usar, por defecto[3] sólo se pueden especificar fuentes que tenga el usuario instaladas en su sistema operativo, ¿pero cómo sabemos cuales son?. Bueno, existen algunas fuentes que se considera seguras y que por tanto podemos contar con que estarán disponibles en casi cualquier sistema operativo.

Además de esto podemos introducir una lista de fuentes separadas por comas, de este modo en caso de que no se encuentre la primera fuente especificada se intentará con las posteriores.

Aquí podemos ver un ejemplo:

```
body{
    font-family: "Times New Roman", Times, serif;
}
```

Si nos fijamos también veremos que aquellas fuentes que tengan nombres compuestos (con espacios) deben ir encerradas entre comillas dobles.

En la siguiente lección veremos una forma sencilla de utilizar fuentes que no estén instaladas en el sistema operativo usando CSS3.

font-weight

Esta propiedad hace referencia al grosor de la fuente y puede tomar múltiples valores: ```normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 | inherit````

De aquí merece la pena mencionar:

- Que "**normal**" es sinónimo de "**400**" y "**bold**" de "**700**", este último representaría una negrita
- No todos los valores están disponibles para todas las fuentes, de hecho salvo que se

usen fuentes personalizadas es difícil encontrar fuentes con anchos distintos a 400 o 700.

Un ejemplo:

```
body{
    font-family: "Times New Roman", Times, serif;
    font-weight: 700;
}
```

font

Como comentábamos esta es una forma de agrupar múltiples valores en una sola propiedad, no es trivial de recordar y por tanto no te recomiendo que la uses mientras estás empezando porque puede darte problemas.

Para que te suene si te lo encuentras en el código de alguien tiene este aspecto:

```
h1{
   font: bold 1.2em Helvetica, Arial, sans-serif;
}
```

Modificar el texto

Para modificar el texto (text) tenemos otras propiedades:

- **text-align**: para alinear el texto (left, right, center, justify)
- **text-decoration**: permite añadir un subrayado, tachar una palabra, etc (underline, line-throught, ...)
- **text-transform**: permite transformar en mayúsculas, minúculas, etc (uppercase, lowercase, capitalize, ...)
- **line-height**: permite ajustar el interlineado usando unidades como vimos antes (px, em, rem, ...).

Puedes ver el detalle de cada una de las propiedades haciendo clic sobre el enlace en cada una.

Por simplificar aquí te mostramos algunos ejemplos:

```
h1{
    text-align: center;
    text-decoration: underline;
    text-transform: uppercase;
}
```

```
.p{
    line-height: 1.5em;
}
```

Otras etiquetas habituales

No es el objetivo de este curso ver todas las propiedades CSS ya que son muchas y la mejor forma de aprenderlas es con la práctica, pero antes de terminar esta lección me gustaría nombrarte algunas más:

- **background** y **background-color**: que como su nombre indica nos permite cambiar el fondo de un elemento (ver más).
- **list-style**, **list-style-image** y **list-type**: nos permite modificar la imagen que precede a los elementos de una lista, etc (ver más).

Ejemplos:

```
body{ background-color: #efefef; }
h1{ background: url("fondo-encabezado.jpg") no-repeat center; }

li{ list-style: none; }
li{ list-style: square outsite; }
li{ list-style-image: url("punto.jpg"); }
li{ list-style-type: upper-roman; }
```

> **Ejemplo interactivo**:
>
> - Lección 1 - Snippet 4: propiedad "**background**".
> - Lección 1 - Snippet 5: propiedad "**list**"

Luego veremos cómo usar DevTools para editar estos estilos "en caliente", aprender los valores que admiten las propiedades y en definitiva cómo experimentar de manera rápida y sencilla.

> **Nota**: en ocasiones cuando a elemento sólo se le define una propiedad te encontrarás que algunas personas lo escriben en una sola línea.

Chuleta CSS

Ya hemos visto unos pocos estilos, conforme avancemos seguramente te cueste recordarlos así que al igual que con HTML aquí te dejo una chuleta de CSS3 con sus propiedades, y no te asustes, no creo que haya nadie que se las conozca todas, a lo largo de este curso te explicaré sólo aquellas que necesitarás con frecuencia y te enseñaré como puedes seguir luego descubriendo más por tu cuenta.

Aclaraciones:

1. En este artículo se puede leer cómo la propiedad color no sólo afecta al texto
2. Una hoja de estilos de un proyecto mediano puede tener de cientos a miles de líneas, en estos casos reducir el número de caracteres puede ayudar a reducir el tamaño del fichero en bastantes KB lo que hace que la página cargue un poco más rápido.
3. En CSS 2.1 no se podían usar fuentes personalizadas, esta es una nueva características incorporada en CSS3.

Formas de añadir CSS

Ahora vamos a repasar las tres formas que existen de añadir estilos a nuestro HTML y una muy breve presentación a los selectores.

Estilos en línea

Voy a explicarte esta forma de aplicar estilos, aunque debes evitarla siempre que te sea posible ya que es una mala práctica estilar tu página de esta manera.

A cualquier etiqueta HTML puedes añadirle la propiedad "**style"** y dentro de ella añadir tantos estilos como desees separados por punto y coma, por ejemplo:

```
<h1 style="color:red; font-size:2em">Título de la página</h1>
```

Esta práctica está totalmente des-recomendada ya que a la larga complica el mantenmiento de los estilos de tu página incitándote a escribir más código del necesario como veremos a continuación.

Estilos internos

Este es el ejemplo que hemos visto hasta ahora, añadiendo la etiqueta "**<style>"** dentro del "**<head>"** de nuestra página HTML.

Estilos externos

Y por último la forma recomendada (**siempre**) de aplicar estilos: creando un fichero con extensión "**.css**" e indicarle al navegador que cargue dicho dichero, para ello usaremos un elemento auto-contenido llamado **<link>** del siguiente modo:

```
<link rel="stylesheet" href="resources/css/main.css">
```

> **Nota**: Una página puede incluir varias etiquetas **link**, o lo que es lo mismo: *múltiples hojas de estilos*. En caso de que una regla[1] esté duplicada siempre prevalecerá la que se cargue en último lugar[2]. Cuando veamos la herencia veremos qué significa esto.

Selectores de etiquetas

Ya que hemos visto todas las formas de cargar estilos CSS en nuestras páginas y cómo aplicar estilos a etiquetas HTML usando el nombre de la etiqueta, vamos a ver otras dos formas de seleccionar etiquetas HTML para aplicarles estilos:

- Asignando un identificador (**único**) al elemento: para ello añadiremos la propiedad **id="valor"** a la etiqueta.
- Asignando una o varias clases al elemento: en este caso usaremos la propiedad **class="valor1 valor2 ..."**.

Veamos un ejemplo:

```
<!-- index.html -->
<h1 id="experiencia">Experiencia <strong class="destacado">profesional<strong></h1>

<p class="destacado">
    A lo largo de los últimos 16 años blah blah blah...
</p>

<style>
    #experiencia{
        font-size: large;
    }
    .destacado{
        color: blue;
    }
    strong{
        font-weight: normal;
    }
</style>
```

En este caso se ha añadido un identificador a la etiqueta "**h1**" y la misma clase ("**destacado**") a dos etiquetas: "**strong**" y "**p**". Dados los estilos definidos el resultado será el siguiente:

- La frase "**Experiencia profesional**" se le aplicará un tamaño mayor (`large`)
- La palabra profesional aparecerá con un ancho de fuente " `normal` "
- Y todo salvo la palabra "**Experiencia**" aparecerá en color azul.

> **Nota importante**: los identificadores son únicos por cada página HTML. Por tanto dentro de un mismo fichero "**.html**" no podemos asignar el mismo valor a dos "**id**" o nos encontraremos con problemas e inconsistencias.

Fuentes personalizadas

Una novedad de CSS3 frente a las versiones anteriores es que se permite el uso de fuentes personalizadas. El repositorio de fuentes más popular es Google Fonts que ofrece un amplio número de ellas de uso libre.

Para para poder usarlas tendremos que:

1. Añadir la hoja de estilos en nuestro HTML usando la etiqueta "**<link>**".
2. Añadir la propiedad **font-family** en los elementos que queramos aplicar la fuente en nuestro CSS.

Por ejemplo:

```
<!-- index.html -->
<link href='https://fonts.googleapis.com/css?family=Open+Sans:400,400italic,600italic,
700' rel='stylesheet' type='text/css'>
```

```
/* main.css */
body{
    font-family: 'Open Sans', sans-serif, arial;
}
```

> **Nota**: al añadir múltiples nombres de fuente separados por coma lo que estamos indicándole al navegador es que si tuviese problemas para cargar la primera fuente lo intente con la segunda, y si tuviese problemas con la segunda lo intentase con la tercera, y así tantas veces como queramos. Por ejemplo: el problema podría deberse a que el navegador no soporte CSS3 y fuentes personalizadas o por ejemplo porque el fichero que contiene la fuente y que tiene que descargar el navegador no estuviese disponible.

Aclaraciones:

1. Una regla no es más que la forma de específicar el elemento HTML a los que se les debe aplicar un estilo definido.
2. El navegador cargará y leerás los ficheros de manera secuencial, esto significa que lee empezando por la primera línea de un fichero y termina por la última, por tanto para facilitarnos la compresión podemos imaginarnos que cuando se carga un fichero con una etiqueta (**link** o **script**), esta etiqueta es reemplazada por el contenido del fichero al que haga referencia dicha etiqueta.

Selectores y herencia

La palabra CSS viene de *Cascading StyleSheets*, esto quiere decir: Hojas de Estilo en Cascada. La palabra **cascada** hace referencia a una propiedad **muy importante** de las hojas de estilo, y es que los estilos aplicados a un elemento padre son heredados por su hijo.

Por ejemplo:

```
<ul style="color: red">
    <li>Inicio</li>
    <li style="color: blue">Experiencia</li>
</ul>
```

En este caso el color de la fuente "Inicio" aparecerá en rojo y la de "Experiencia" en azul.

> **Nota**: te recuerdo que **no está recomendado** aplicar los estilos usando atributos. En este caso lo he hecho así porque creo que queda más clara la explicación.a

Del mismo modo y como ya adelantábamos al principio del capítulo, si un estilo se define dos veces, el último definido será el que prevalecerá, por ejemplo si en nuestro fichero escribimos:

```
/* main.css */
p {
    color: orange;
    font-size: 24px;
}
p {
    color: green;
}
```

En este caso el color de todas las etiquetas **<p>** será verde.

Ocurre lo mismo si la misma regla está definida en dos hojas CSS distintas que hayan sido cargadas usando la etiqueta **<link>**, en este caso prevalence el estilo definido en la última hoja cargada.

Si por error definiésemos la misma propiedad dos veces en un elemento también prevalecerá la segunda, por ejemplo:

```
/* main.css */
strong {
    background: orange;
    background: yellow;
}
```

En este caso el fondo de la etiqueta prevalecerá en amarillo como se puede ver aquí: http://libro.cursohtml5desdecero.com/css/?lesson=2&snippet=1.

Predominancia del estilo más específico

Ya hemos visto que podemos aplicar los estilos de tres formas, estas formas son de menos a más específicas:

- Por nombre de etiqueta
- Por clase (**class**)
- Por identificador (**id**)

Si asignamos estilos a un elemento de diferentes formas siempre predominará el más específico, esto quiere decir por ejemplo que si definimos lo siguiente:

```
<head>
    <style>
        #food {
            color: green;
        }
        p {
            color: orange;
        }
    </style>
</head>
<body>
    <p id="food">Mi fruta favorita es el mango.</p>
    <p>Mi cereal favorito es el trigo</p>
</body>
```

La frase "**Mi fruta favorita es el mango**" aparecerá en color verde, dado que un identificador es más específico que el nombre de la etiqueta. Sin embargo "**Mi cereal favorito es el trigo**" aparecerá en naranja, porque el estilo más específico para esa etiqueta es el de la etiqueta "**p**".

> **Nota**: la especificidad se calcula de una manera más compleja como se puede ver en este tutorial, pero para este curso he preferido simplificarlo un poco. Luego nos ayudaremos con DevTools para ayudar a experimentar y entender mejor cómo se aplican los niveles de especifidad.

Combinar selectores

Hasta el momento hemos visto cómo utilizar un selector para especificar un elemento, pero podemos combinar **cualquiera** estos selectores siguiendo las siguientes reglas:

- Si escribimos los selectores separados por un espacio estamos haciendo referencia al etiquetas anidadas dentro de otras.
- Si añadimos los selectores sin separar por un espacio estamos haciendo referencia a un mismo elemento de una manera más específica.
- Para entender cómo se aplican los selectores debes leerlos de derecha a izquierda
- El último selector antes del caracter "**{**" será al que se le aplique el estilo.

De este modo podemos seleccionar todos los elemento HTML que contienen una clase predefinida, etc. En este ejemplo vamos a ver cómo combinar nombres de etiquetas HTML con clases:

```
<head>
    <style>
        h1 span{
            color: red;
        }
        p.big{
            font-size: 2rem;
        }
        p.big span{
            font-weight: bold;
        }
    </style>
</head>
<body>
    <h1>Mi <span>Curriculum Vitae</span></h1>
    <p class="big"><span>Nombre</span>: Raúl Jiménez Ortega</p>
    <p class="big"><span>Lugar de nacimiento</span>: Málaga</p>
    <p>Fecha de nacimiento: 11/03/1984</p>
</body>
```

Aquí estamos indicando:

- Que el color del texto de las etiquetas **span** que sean descendientes (no necesariamente hijos directos) de las etiquetas **h1** aparezcan en rojo.

- Que los párrafos con que contengan la clase "**big**" tengan un tamaño de fuente de "**2rem**".
- Que el ancho de la fuente de las etiquetas **span** que sean descendientes de las etiquetas **p** con la clase **big** sea negrita.

Esto al principio puede parecer un poco lioso, pero al final verás que aprenderás a usarlo por sentido común.

> **Notas**:
>
> 1. Del mismo modo podríamos hacemos combinaciones usando identificadores, aunque dado que un identificador es único para un elemento y es el más específico no debería ser necesario usarlo nunca.
> 2. Al combinar los selectores la especifidad cambia como podemos leer en el tutorial antes mencionado.

Múltiples clases

Es habitual utilizar varias clases en un mismo elemento[1], por ejemplo:

```
<a class="btn btn-primary">Entrar</a>
```

A este elemento se le aplicarán los estilos de la clase "**.btn**" y la clase "**.btn-primary**".

Si se diese caso de que ambas clases especifican una misma propiedad, por ejemplo **color** predominará la de la última indicada, en este caso la de la clase "**.btn-primary**".

Otros selectores

Por último añadir que se puede aplicar el mismo estilo a varios selectores o conjunto de selectores separándolos por una coma, por ejemplo

```
.bold,
strong,
p.title{
    font-weight: bold;
}
```

Esto significa que tanto a las etiquetas con la clase "**bold**" como las "****" como las "**<p>**" que contengan la clase "**title**" se les aplicará el estilo "**font-weight: bold;**".

También se puede usar el caracter "**>**" para especificar un hijo directo de un elemento, y otros pseudo-elementos, pero no entraremos en estos detalles en este curso.

Aclaraciones:

1. Esto lo veremos frecuentemente si usamos herramientas como "Bootstrap" u otros frameworks CSS.

Estilos con Chrome DevTools

Las modificaciones del CSS en elements ...

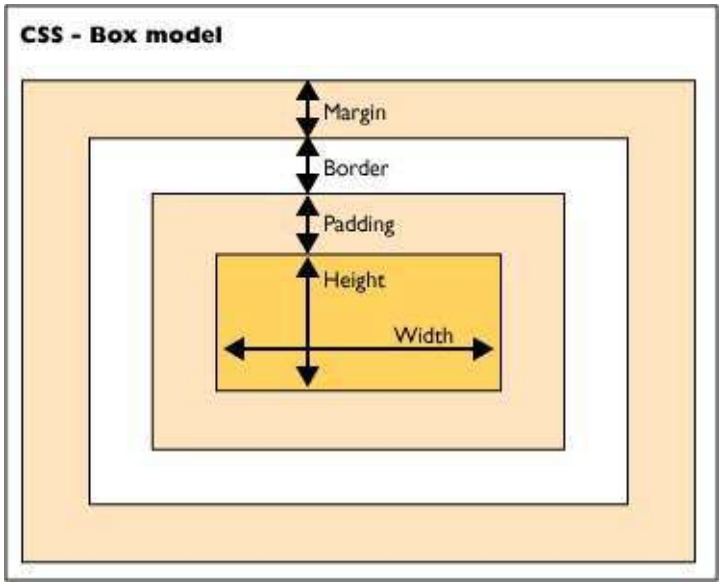
Recursos

- Can I Use?
- http://browsershots.org/

Anexo: Navegadores y estándares soportados

Tanto CSS3 como HTML5 son estándares creados por el W3C que recordemos que incluyen muchos elementos, propiedades, etc. Por tanto el trabajo que implica adaptar cada uno de los navegadores para comprendan estos estándares es costoso y conlleva tiempo, es por eso que el soporte en los mismos evoluciona progresivamente. Además cada empresa u organización responsable de dicha adaptación prioriza (bajo su propio criterio) en qué orden soportará cada característica. Al final esto conlleva a que el soporte de las nuevas características de los estándares a día de hoy (17 Sept. 2016), dos años después de su publicación no estén completamente soportados en ningún navegador, pero ni siguiera estén las mismas características en cada navegador[1].Si te preguntase, ¿de cuántos navegadores crees que te deberías preocupar? (los más importantes). Supongo que te respuesta sería: tres o cuatro (Chrome, Internet Explorer, Firefox y Safari) o algo similar, ¿no?.Bueno, ojalá fuese tan fácil, al final cada uno de estos navegadores no sólo dispone de versiones para distintos sistemas operativos (Windows, Mac, Linux, Android, iOS, Windows Phone, etc), sino que además los usuarios no siempre usan la última versión de cada navegador. Y el problema añadido que supone esto, es que para nuestra desgracia, un mismo navegador (por ejemplo Chrome), no necesariamente tiene que soporta exáctamente igual las características en Windows que en Android.Pero no todo son malas noticias, para nuestra fortuna existe un proyecto llamado: CanIUse.com, que concretamente nos va a resolver nuestras dudas. Por ejemplo: ¿tendré problemas si uso el elemento HTML5 `video` en el código de mi página? Si entras en la página de CanIUse comprobarás que dicho elemento no está soportado por Internet Explorer 8 ni por Opera mini o que la propiedad CSS3 `background-attachment` no está soportada por el buscador de Android ni por Opera mini.Y esto... ¿eso qué quiere decir?, ¿puedes o no?. Para tomar esta decisión yo te recomiendo que te informes todo lo posible de qué tipo de personas visitarán tu web para intentar averiguar qué sistemas operativos, navegadores, etc usan y en base a eso decidas. Por ejemplo no es lo mismo un blog de productos Apple (probablemente tendrás un tráfico mayor que la media de usuarios que usen Safari), que una página que vayas a promocionar mucho en redes sociales (tendrás mucho tráfico móvil), etc.Si estás actualizando una página antigua puedes usar Google Analytics o cualquier otra herramienta alternativa para obtener la información de tu tráfico actual.Por este motivo es por lo que empezaremos aprendiendo propiedades CSS de la versión 2.1, porque actualmente se puede considerar completamente soportado por el 99% de los navegadores.

CSS: Modelo de caja



Elementos HTML

Existen dos elementos que utilizaremos para envolver el contenido pero que no tienen ningún *valor semántico*.

Propiedades - Parte 2

# #

- **opacity**: establece la transparencia de un elemento

Borde (Border)

- border-width:
- border-style:
- border-color:
- border:

Ver todos

Modificar el fondo de un elemento

Fondo (Background)

- **background-color**: nos permite especificar el color (igual que vimos antes) de fondo de un elemento.
- **background-image**: permite especificar una URL de una imagen que queremos que aparezca de fondo (por ej: url('imagen.jpg'))
- **background-repeat**: por defecto si establecemos una imagen de fondo se repite indefinidamente, pero esto podemos cambiarlo (repeat-x, repeat-y, no-repeat, ...)
- **background-position**: nos permite cambiar la posición de la imagen de fondo (left, right, center, ...)
- **background**: es un atajo igual que la propiedad **font**.

Ver todos

Ejemplos

```
body{
    background-color: #efefef;
}
```

```
h1{
    background: url("fondo-encabezado.jpg") no-repeat center;
}
```

Modificar una lista

Lista (List)

- **list-style-image**: permite especificar una imagen para cada que preceda a cada elemento de la lista.
- **list-style-type**: si no es establece la propiedad *list-style-image* especficar el formato que precede a un elemento de la lista
- **list-style**: es un atajo igual que *font* o *background*

Ver todos

Ejemplos

```
li{
    list-style: none;
}
```

```
li{
    list-style: square outsite;
}
```

```
li{
    list-style-image: url("punto.jpg");
}
```

```
li{
    list-style-type: upper-roman;
}
```

Posicionar el contenido

Propiedad position

- static
- relative
- absolute
- fixed

Propiedad z-index

Esta propiedad establece el orden en el que aparecen las cajas en el eje Z (profundidad) y se establece como un número entero.

Ejemplo:

```
div{
    z-index: 999;
}
```

CSS: Refinando el diseño

Tipografías

tamaños %, em, ...

JS: Primeros pasos

JavaScript (JS) es un lenguaje de programación, un lenguaje con su propio vocabulario, sintaxis, semántica, expresiones, errores, etc.

JavaScript nos permite darle vida a la web, hacerla más dinámica e interactiva, y por tanto mostrar algo más que información de manera estática.

¿Qué se puede hacer con JavaScript?:

- Operaciones matemáticas, lógicas, etc.
- Controlar el flujo del programa
- Validar formularios
- Cargar contenidos mediante peticiones HTTP
- Modificar el DOM
- Acceder a información como la versión del navegador, tamaño de la ventana, sistema operativo, localización, etc.
- Etc.

Mi primer script

Este es el script más simple que podemos hacer:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Mi primer script</title>
</head>

<body>
  <script>
    document.writeln('Hola Mundo!');
  </script>
</body>
</html>
```

Vamos a ver cómo interpretar este script. Lo que estamos haciendo es:

1. Llamar al **método** `writeln` [1] que escribe en el DOM lo que recibe como parámetro seguido de un salto de línea, en este caso `Hola Mundo!` seguido de un salto de línea (`\n`).

2. Este método está definido en el `document`[2] y que representa al DOM y que tiene otras funciones para acceder a elementos del DOM, etc.[3]

Para evitar errores que pueden pasar desapercibidos en JavaScript (por si flexibilidad) te recomiendo que introduzcas siempre la expresión `'use strict';` al principio de tus scripts. El modo estricto significa entre otras cosas que hay que declarar todas las variables y objetos[4].

Así quedaría:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Mi primer script</title>
</head>

<body>
  <script>
    'use strict';
    document.writeln('Hola Mundo!');
  </script>
</body>
</html>
```

Sintaxis

Algunas de las características de JavaScript son:

1. Es sensible a mayúsculas y minúsculas (o lo que es lo mismo, es *case-sensitive*), por tanto:

   ```
   var variable;
   ```

   no es equivalente a

   ```
   var Variable;
   ```

2. No es obligatorio (pero sí recomendado) declarar las variables
3. No se define el *tipo* de las variables
4. No es necesario (pero sí recomendado) terminar cada expresión con el carácter de punto y coma (;)
5. Se pueden incluir comentarios en una línea usando `//` y en múltiples líneas usando `/* */`.

Aclaraciones:
1. Más información sobre el método writeln
2. Más información sobre la interfaz document.
3. Y el documento está definido como parte del objeto **`window`** que representa a la ventana del navegador donde está cargado el DOM y donde se almacena mucha más información. Añadir la palabra `window` es opcional.

</small>

- Y otras tantas restricciones más.

Variables

Las variables en los lenguajes de programación se asemejan a las variables utilizadas en matemáticas, se utilizan para almacenar y hacer referencia a valores, gracias a ellas podemos darle vida a la web.

Para declarar/definir una variable utilizaremos la palabra clave `var` seguida del nombre de la variable y un punto y coma (`;`), por ejemplo:

```
var counter;
```

En este caso hemos declarado una variable con el nombre `counter` pero no se le ha asignado ningún valor.

> **Consejos**:
>
> - Aunque no es obligatorio, acaba siempre las sentencias con punto y coma (por convención).
> - Escribe siempre el código en inglés (se considera más profesional).

El nombre de una variable debe cumplir las siguientes normas:

- El primer carácter **no** puede ser un número.
- Sólo puede estar formado por letras, números y los símbolos: dólar (`$`) y guión bajo (`_`).

Por tanto, las siguientes variables estarían bien definidas:

```
var $num1;
var _$name;
var $$$otherNumber;
var $_a__$4;
```

> **Consejo**: elige nombres de variables que sean representativos del valor que almacenan para facilitar la comprensión del código.

Por ejemplo:

```
var counter = 0;
var name = "Raul";

// En lugar de:
var aux = 0;
var tmp = "Raul";
```

Palabras reservadas

Antes de continuar me gustaría comentarte que existen palabras reservadas que tienen un significado en el lenguaje y que no podremos usar como nombres de variables: `abstract, boolean, break, byte, case, catch, char, class, const, continue, debugger, default, delete, do, double, else, enum, export, extends, false, final, finally, float, for, function, goto, if, implements, import, in, instanceof, int, interface, long, native, new, null, package, private, protected, public, return, short, static, super, switch, synchronized, this, throw, throws, transient, true, try, typeof, var, volatile, void, while, with` .

Tipos de variables

En todos los lenguajes de programación existen distintos tipos de variables, en JavaScript tendremos:

```
// Númericas (integer & floats)
// ----------------------------
var counter = 16;   // variable tipo entero
var price = 19.99;  // variable tipo decimal
```

Que nos permiten almacenar números enteros y con decimales para realizar operaciones.

```
// Cadenas de texto (strings)
// --------------------------
var msg = 'Bienvenido a nuestro sitio web';
var txt = 'Una frase con "comillas dobles" dentro';
var txt = 'Una frase con \'comillas simples\' dentro';
```

Que nos permiten trabajar con cadenas de texto. Para ello tenemos con encerrar la cadena entre comillas simples o dobles, pero normalmente se recomienda hacerlo con comillas simples. En caso de querer introducir una comilla simple dentro de una cadena podemos hacerlo incluyendo el carácter contra-barra (\) justo delante, para evitar que se cierre la cadena.

```
// Colecciones (arrays)
// ----------------------------
// Definiendo los días de la semana en cadenas de texto
var day1 = 'Lunes', day2 = 'Martes', ... , 'Domingo';

// Definición equivalente en un Array
var days = ['Lunes', 'Martes', 'Miércoles', 'Jueves', 'Viernes', 'Sábado', 'Domingo'];

// days[0] = 'Lunes'
// days[1] = 'Martes'
// ...
// days[6] = 'Domingo'
```

Los `Arrays` o colecciones nos permiten añadir varios valores dentro de un elemento.

```
// Booleanos (boolean)
// ----------------------------
var valid = false;
var prime = true;
```

Los booleanos se utilizan para almacenar valores lógicos: `true` o `false`.

Funciones

Existen múltiples funciones para trabajar con números: Para los números hay una función muy útil:

```
var n = 231.8273;
n.toFixed(2); // 231.82
```

cadenas de texto, por ejemplo:

```
var hello = 'Hola ';
var world = 'Mundo!';

// Para contar el número de caracteres
console.log(hello.length); // 5

// Para concatenar cadenas
console.log(hello + ' ' + world); // Hola Mundo!
console.log(hello.concat(' ' + world));  // Hola Mundo!

// Para busacar subcadenas en una cadena
var pos = hello.indexOf('a'); // pos = 3
var pos = hello.indexOf('b'); // pos = -1
```

Y otros métodos: `lastIndexOf` , `substring` , `split` , etc.

Al igual para trabajar con Arrays:

```
var fruits = ['banana', 'melon, 'orange'];

// Para contar
var n = fruits.length; // n = 3

//Para añadir elementos
fruits.push('apple', 'peach'); // fruits = ['banana', 'melon, 'orange', 'apple', 'peac
h']
```

`contact` , `join` , `pop` , `shift` , Y otras como: `unshift` , `reverse` .

Operadores

Los operadores nos vas a servir para modificar y comprobar el valor de las variables, vamos a ver diferentes tipos de operadores:

- Matemáticos
- Lógicos
- Relacionales

Operadores matemáticos

Los operadores matemáticos nos van a permitir realizar operaciones matemáticas sobre las variables, veamos algunos ejemplos:

```
// Asignación (=)
var pi = 3.1416;
```

Nos permite darle un valor a una variable.

> **Consejo**: Añade siempre un espacio antes y otro después de cualquier operador (`=` , `<` , ..).

```
// Incremento (++) y decremento (--)
var x = 1, y = 4;
x++; // x = 2
y--; // y = 3
```

Nos permite incrementar o decrementar en una unidad el valor de una variable.

```
// Suma (+) y resta (-)
var x = 2, y = 3, z;
z = x + y // z = 5;
z = x - y // z = -1;
```

```
// División (/) y multiplicación (*)
var x = 4, y = 2, z;
z = x / y // z = 2;
z = x * y // z = 8;
```

```
// Abreviaciones
var x = 5;
x += 3;  // x = x + 3 => 8
x -= 1;  // x = x - 1 => 4
x *= 2;  // x = x * 2 => 10
x /= 5;  // x = x / 5 => 1
```

// Módulo (%) numero1 %= 4; // numero1 = numero1 % 4 = 1

Operadores lógicos

Los operadores lógicos son imprescindibles para realizar aplicaciones complejas, ya que se utilizan para tomar decisiones sobre las instrucciones que debería ejecutar el programa en función de ciertas condiciones.

El resultado de cualquier operación que utilice operadores lógicos siempre es un valor lógico o booleano.

```
var visible = true;
!visible;  // Devuelve "false" y no "true"
```

x	!x
true	false
false	true

Operación AND (&&)

La operación lógica AND obtiene su resultado combinando dos valores booleanos. El operador se indica mediante el símbolo && y su resultado solamente es `true` si los dos operandos son `true`:

x	y	x && y
true	true	true
true	false	false
false	true	false
false	false	false

```
var x = true;
var y = false;
result = x && y; // result = false

x = true;
y = true;
result = x && y; // result = true
```

Operación OR (||)

La operación lógica OR también combina dos valores booleanos. El operador se indica mediante el símbolo || y su resultado es `true` si alguno de los dos operandos es `true` :

x	y	x | | y
true	true	true
true	false	true
false	true	true
false	false	false

```
var x = true;
var y = false;
result = x || y; // result = true

x = false;
y = false;
result = x || y; // result = false
```

Operadores relacionales

Los operadores relacionales definidos por JavaScript son los mismos que los matemáticos:

- Mayor que: `>`
- Menor que: `<`
- Mayor o igual: `>=`
- Menor o igual: `<=`
- Igual que: `==`
- Distinto de: `!=`

Aunque también existe el operador `===` que quiere decir **exáctamente igual**, teniendo en cuenta no sólo el valor de la variable sino también el tipo, por ejemplo:

```
0 == ""         // true
0 === ""        // false

0 == false      // true
0 === false     // false

2 == '2'        // true
2 === '2'       // false
```

Vamos a ver en la siguiente lección que estos operadores son imprescindibles a la hora de controlar el flujo de un programa.

El resultado de todos estos operadores siempre es un valor booleano:

```
var even = 2;
var odd = 5;
result = even > odd; // result = false
result = even < odd; // result = true

a = 5;
b = 5;
result = a >= b; // result = true
result = a <= b; // result = true
result = a == b; // result = true
result = a != b; // result = false
```

Se debe tener especial cuidado con el operador de igualdad (==), ya que es el origen de la mayoría de errores de programación, incluso para los usuarios que ya tienen cierta experiencia desarrollando scripts. El operador == se utiliza para comparar el valor de dos variables, por lo que es muy diferente del operador =, que se utiliza para asignar un valor a una variable:

```
// El operador "=" asigna valores
var x = 5;
y = x = 3;  // y = 3 y x = 3

// El operador "==" compara variables
var x = 5;
y = x == 3; // x = 5 y y = false

/*
   Los operadores relacionales también se pueden
   utilizar con variables de tipo cadena de texto:
*/
var txt1 = "hola";
var txt2 = "hola";
var txt3 = "adios";

result = txt1 == txt3; // result = false
result = txt1 != txt2; // result = false
result = txt3 >= txt3; // result = false
```

Cuando se utilizan cadenas de texto, los operadores "mayor que" (`>`) y "menor que" (`<`) siguen un razonamiento no intuitivo: se compara letra a letra comenzando desde la izquierda hasta que se encuentre una diferencia entre las dos cadenas de texto. Para determinar si una letra es mayor o menor que otra, las mayúsculas se consideran menores que las minúsculas y las primeras letras del alfabeto son menores que las últimas (a es menor que b, b es menor que c, A es menor que a, etc.)

Ejercicios

1) Instalar Sublime Text

Instalar Sublime Text y en `preferences->settings-user` comprobar que están estas tres líneas (sino añadirlas):

```
{
    "indent": 2,
    "tab_size": 2,
    "translate_tabs_to_spaces": true
}
```

2) Operaciones simples

Realiza un script que realice lo siguiente:

- Almacenar en una variable el resultado de sumar 1 y 2
- Almacenar en una variable el resultado de dividir 6 entre 2
- Almacenar en una variable el precio de un artículo de 20€ aplicándole el 21% de IVA .
- Definir una variable con el valor 4 y utilizar el operador (`++`) para incrementar en uno su valor.
- Definir una variable que almacene la concatenación de dos cadenas de texto.
- Definir una variable `price` con el valor `19.99` y aplicar la abreviación `/=` para dividirlo entre `1.21` para obtener el precio sin IVA.
- Asignar a dos variables valores booleanos y hacer al menos una operación combinando un operador lógico: **AND** (`&&`) o **OR** (`||`)
- Realizar 4 expresiones que utilicen operadores relacionales (`<` , `==` , `!=` y `===`) y almacenen los valores en tres variables distintas.

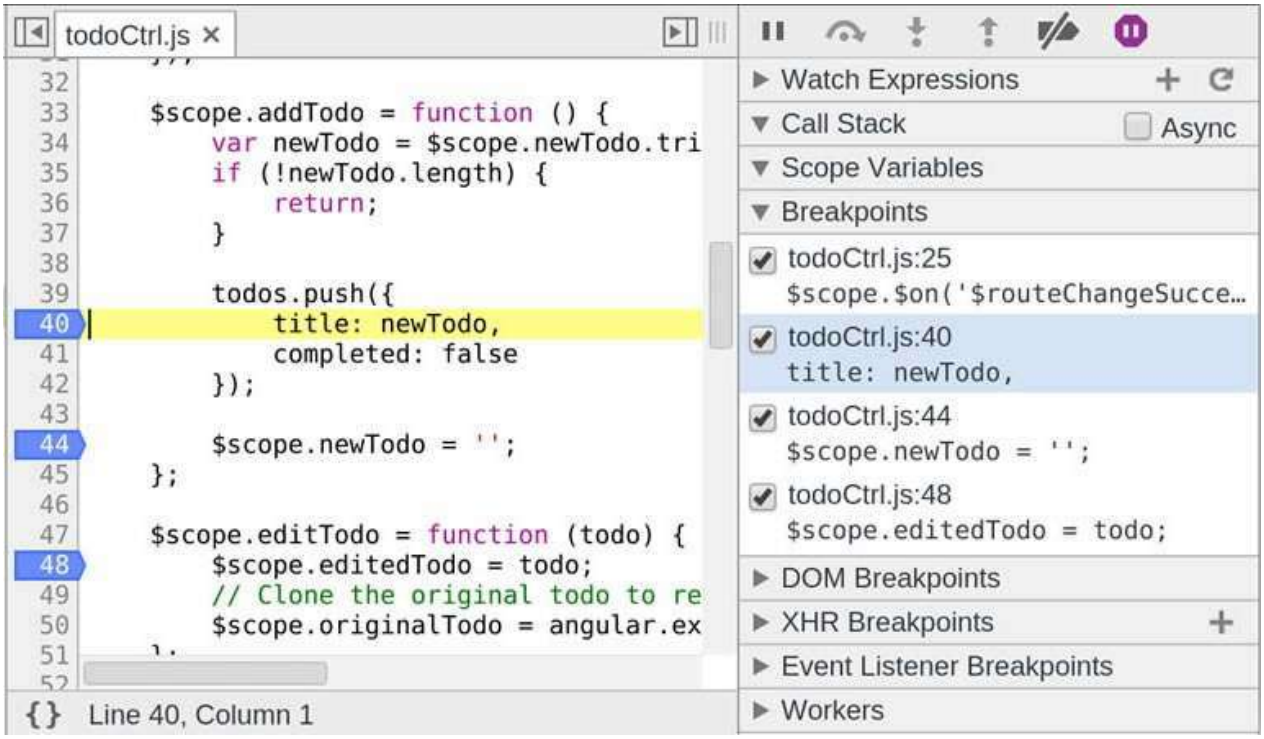
Finalmente imprime **todos valores** en la consola del navegador usando `console.log(nombre_de_la_variable)` , por ejemplo:

```
var result = 1 + 2;
console.log(result);
```

> **Nota**: Como la mayoría de los lenguajes, JavaScript se ejecuta secuencialmente (de arriba a abajo), por lo que el orden de las instrucciones importa.

3) Puntos de parada

Utiliza la pestaña sources y haz clic en alguna línea donde haya una expresión para establecer un punto de parada (se debe marcar en azul) y recarga la página:



Juega con el inspector.

Opcional: Instalar W3CValidators

Recuerda que en las lecciones de HTML utilizábamos el validador online del W3C para comprobar que nuestro código era correcto. Si lo prefieres también puedes usar la extensión W3CValidators de Sublime Text para hacerlo desde el propio editor.

Dudas

Si hay algo que no te haya quedado claro puedes preguntar cualquier duda en los issues del proyecto en Github.

Si tienes problemas o dudas con tu código súbelo a Github, abre un issue en un proyecto con la duda/problema y envíame un correo a info@cursohtml5desdecero.com.

Recursos

- Playlist de Youtube con introducción a SublimeText

Otras aclaraciones

Objeto window

Otras funciones comunes definidas en el objeto `window` son:

- `alert()` que abre una ventaja con un mensaje el navegador del usuario, aquí puedes verlo en funcionamiento.
- `console` que implementa funciones para imprimir mensajes en la consola de error (`console.error()`), etc

Otras características de JS

Existen más características, como que si la ejecución de un script dura demasiado tiempo (por un error, por ejemplo de programación) el navegador puede informarle al usuario de que hay un script que está consumiendo demasiados recursos y darle la posibilidad de detener su ejecución.

Orden de definición de las variables

Definirlas en la parte superior del script

Estructuras de control

Estructura `if`

```
var printMsg = true;

if(printMsg) {
  console.log('Hola Mundo');
}

if(printMsg == true) {
  console.log('Hola Mundo');
}
```

Un ejemplo usando un comparador lógico:

```
var printMsg = false;

if(!printMsg) {
  console.log('Me imprimo');
}

var isFirstMsg = true;

if(!printMsg && isFirstMsg) {
  console.log('Mi primer mensaje');
}
```

Un **error típico** es introducir una asignación (`=`) en lugar de una comparación (`==`)

```
// Error - Se asigna el valor 'false' a la variable
if(printMsg = false) {
  ...
}
```

Estructura `if` ... `else`

```
var age = 18;

if(age >= 18) {
  console.log('Eres mayor de edad');
}
else {
  console.log('Eres menor de edad');
}
```

Estructura `if` ... `else if` ... `else`

```
if(age < 18) {
  console.log('Eres menor de edad');
}
else if(age < 30) {
  console.log('Aún eres joven');
}
else {
  console.log('La sabiduría la da la experiencia');
}
```

Estructura `for`

```
for(initialization; condition; increment) {
  ...
}
```

```
var i;
var days = ['Lunes', 'Martes', 'Miércoles', 'Jueves', 'Viernes', 'Sábado', 'Domingo'];

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

Objetos y funciones

Iterar sobre objetos callbacks

ámbitos

```
// Objetos (objects)
// ----------------------------
var obj = {
    name: 'Raul',
    last_name: 'Jimenez Ortega',
    age: 31
};

// obj.name = 'Raul'
// obj.last_name = 'Jimenez Ortega'
// obj.age = 31
```

Los objetos nos permiten definir estructuras de datos con distintos tipos de valores, ya verás que esto te será muy útil en el futuro.

Peticiones AJAX

Google Spreadsheets CORS http://www.html5rocks.com/en/tutorials/cors/#toc-adding-cors-support-to-the-server https://www.youtube.com/watch?v=3l13qGLTgNw

Expresiones regulares

Aplicaciones web offline

http://www.html5rocks.com/en/tutorials/appcache/beginner/

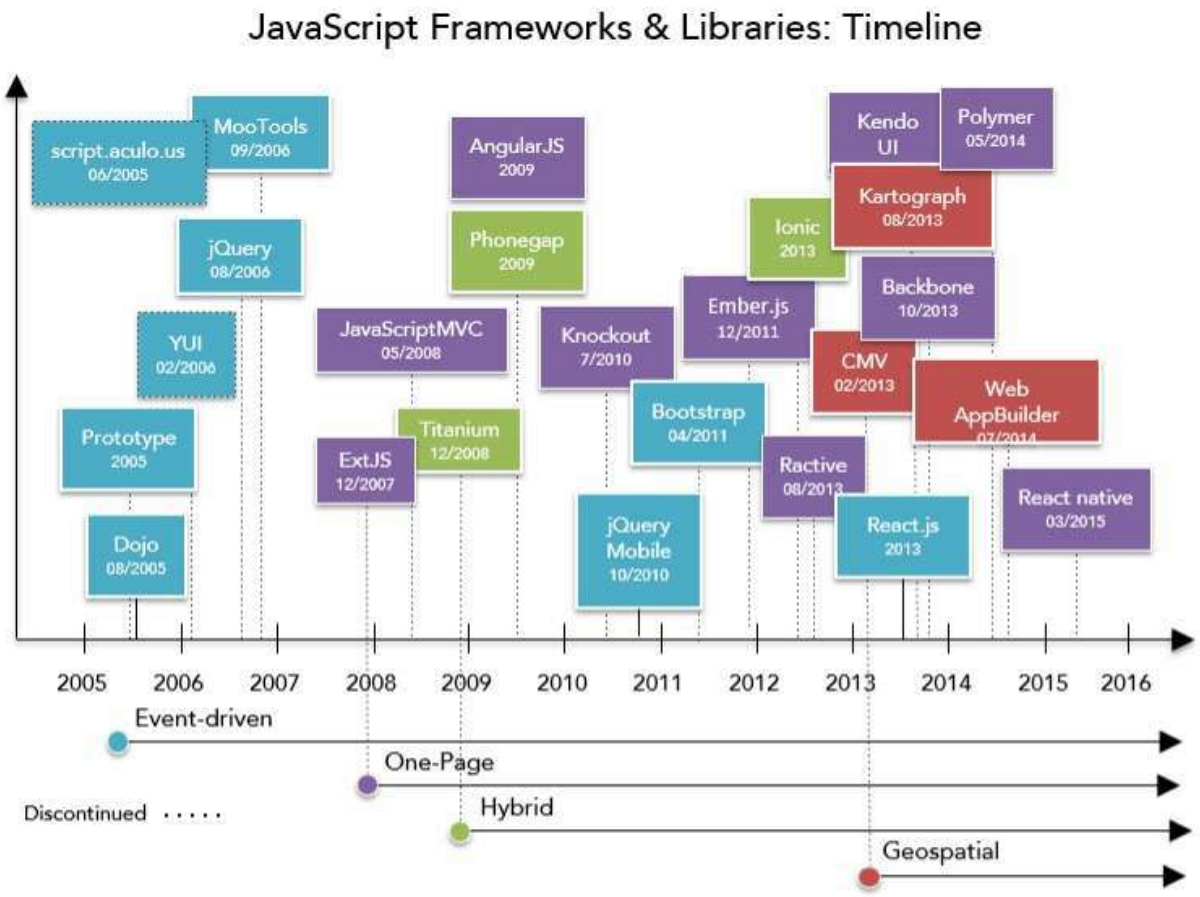
Bibliotecas de terceros

Gráfica

jQuery

Dojo

http://download.dojotoolkit.org/release-1.6.0/cheat.html



Ejercicios

Instalar JSHint

https://github.com/victorporof/Sublime-JSHint