Diseño y elaboración de páginas Web

HTML5, CSS3 y JavaScript

Francisco Cortés Arredondo
Eduardo Ochoa Hernández
Sergio Rogelio Tinoco Martínez
Luis Gerardo Zavala Figueroa
Filho Enrique Borjas García

SEP | conalep Michoacán

Semestre 6

conalep
Michoacán

PRESENTA:

Diseño y elaboración de páginas Web

HTML5, CSS3 y JavaScript

Autores

Francisco Cortés Arredondo
Eduardo Ochoa Hernández
Sergio Rogelio Tinoco Martínez
Luis Gerardo Zavala Figueroa
Filho Enrique Borjas García

Coordinadores:
Lic. José Azahir Gutiérrez Hernández
Ing. Eduardo Ochoa Hernández
Lic. Filho Enrique Borjas García

Título original de la obra:

Registro: **CONALEP-LIB-Web-01A**

Programa: Profesor escritor, objetos de aprendizaje. Desarrollo de la competencia de la producción de información literaria y lectura.

Arredondo Cortés F..; *et al.* (2014) **Diseño y elaboración de páginas Web.** México: CONALEP/CIE

xiv, 238 p.; carta

Registro: **CONALEP-LIB-WEB-01A** Documentos en línea

Editores:
Ing. Eduardo Ochoa Hernández
Lic. Filho Enrique Borjas García

Editorial: CONALEP Michoacán
Col. Chapultepec norte, Gral. Nicolás Bravo No. 144, Morelia, Michoacán.
http://www.cie.umich.mx/conalepweb2013/

Registro: **CONALEP-LIB-ESP-01A**
ISBN: En trámite
Impreso en_____
Impreso en México –Printed in Mexico

DIRECTORIO

Lic. Fausto Vallejo Figueroa
Gobernador Constitucional del Estado de Michoacán

Lic. J. Jesús Sierra Arias
Secretario de Educación

Mtro. Álvaro Estrada Maldonado
Subsecretario de Educación Media Superior y Superior

Ing. Fernando Castillo Ávila
Director de Educación Media Superior

M.A. Candita Victoria Gil Jiménez
Directora General del Sistema CONALEP

M.C. Víctor Manuel Lagunas Ramírez
Titular de la Oficina de Servicios Federales en Apoyo a la Educación en Michoacán

Dr. Salvador Jara Guerrero
Rector de la Universidad Michoacana de San Nicolás de Hidalgo

Lic. José Arturo Villaseñor Gómez
Director General del CONALEP Michoacán

Lic. José Azahir Gutiérrez Hernández
Director Académico

L.E. Rogelio René Hernández Téllez
Director de Planeación, Programación y Presupuesto

Lic. Faradeh Velasco Rauda
Directora de Promoción y Vinculación

Ing. Mónica Leticia Zamudio Godínez
Directora de Informática

Lic. Víctor Manuel Gómez Delgado
Director de Servicios Administrativos

Ing. Genaro González Sánchez
Secretario General del SUTACONALEPMICH

Tec. Juan Pineda Calderón
Secretario General del SUTCONALEP

Prefacio

Estimado estudiante:

Las palabras que sombrean estas páginas, no son simple ciencia dentro del diálogo como depósitos de datos e información, ni son cuestión de vocabulario o listado de definiciones, son la experiencia generosa de la comunidad CONALEP Michoacán, esa realidad oculta pero necesaria que respaldó las tareas de investigación y composición literaria del discurso que integra este libro. Nos referimos a los profesores, administrativos y sindicatos que hoy convergen en el umbral de la existencia para apoyar a un grupo de profesores escritores que han creado en el sereno libre, arquitecturas de conocimientos como un viaje de aprendizaje que exigirá del estudiante, lo mejor de sí mismo ante la presencia luminosa del texto, ese que pretende enseñarle a caminar con la frente en alto.

Las ideas asociadas en este libro, equivalen a la imaginación lograda en el acto de escribir desde otros textos, al decodificarlas el estudiante, se le exige más vocabulario para enriquecer su habla y hacer ver a sus ojos más allá de la estrechez de la información que inunda a la sociedad moderna. El libro no presenta la superficie de la existencia como cruda observación, procura que su dificultad incite a perforar la realidad hasta reflexiones que renueven los modos inciertos de dar significado al mundo. La ciencia, la literatura y la tecnología no las percibimos como mundos incomunicables, los valores son explícitos caminos que las vinculan entorno al currículo del técnico bachiller. Tienen estos textos organización de premisas, técnicas, justificaciones, normas, criterios y como Usted se dará cuenta, también mostrará nuestros límites para seguir haciendo puentes entre las incesantes creaciones de nuevas fronteras de la investigación científica y técnica. Se pretende que estos libros sean contenido y no un libro de prácticas escolares, sean la herramienta de complementación para enriquecer los discursos de la enseñanza-aprendizaje.

Los profesores de CONALEP enfrentan a diario las carencias visibles de medios tecnológicos, materiales y documentales, sería fácil usar las palabras para señalar hasta el cansancio nuestras apremias, pero se ha decidido mejor producir libros como testimonios vivos y luminosos que renueven el rol social de la academia colegiada sensible a la condición social, susceptible de ir perfeccionándose con la acumulación de esta experiencia literaria, para servir de mejor manera al enriquecimiento de las competencias necesarias para realizar el sueño de éxito de tantos jóvenes Michoacanos.

Lic. José Arturo Villaseñor Gómez
Director General del CONALEP Michoacán

Mensaje a la comunidad académica

Se vive una época difícil para los libros, no son momentos de lectura tampoco, sin embargo, esto no debe ser un motivo para no producir literatura para educar mentes creativas, nuestra generación considera importante que sea la literatura la que emocione a realizar los grandes sueños de nuestros jóvenes. Si el libro escrito por mexicanos para mexicanos se perdiera, la conciencia de nuestra sociedad quedaría en orfandad, congelada de sentido y seriamos habitantes de un mundo siempre detrás de mascaras en rituales de simulación. Este argumento presente en *El laberinto de la soledad*, Octavio Paz nos dice al hombre moderno, seamos universales sin dejar de ser diferentes. El efecto de producir literatura en una sociedad, es como dice Steven Pinker, es "sacar los ángeles que llevamos dentro", es lo mejor de nuestro ser como oferta de conocimiento y valores al servicio de nuestra sociedad. El efecto Malinche que prefiere libros de traducción, ocasiona la perdida de identidad y crear la cultura de producir experiencias de conocimiento como herencia educativa para nuestros estudiantes. Los Aztecas en su derrota se sintieron abandonados por sus dioses, en el presente si cancelásemos que la voz de nuestro profesores que hablaran desde libros a las nuevas generaciones, es algo equivalente, al negar que las ideas en su crecimiento son una respuesta sociolingüística económica y democrática de identidad de una nación. Michoacán es un proyecto histórico de libertad, sin ignorar la realidad del malestar de la cultura actual, CONALEP desarrolla un programa académico para impulsar su capacidad y compromiso social para generar las ideas curriculares para enriquecer la sensibilidad y la imaginación científica, técnica y humanista de su comunidad.

Producir literatura curricular en CONALEP Michoacán, es asumir su personalidad moral, como dueña de una conciencia movida para una educación con la pasión de razones y expresiones culturales con la competencia para transformar positivamente la realidad. Aun cuando esta realidad adversa, de actitudes de autoridades renuentes a transformar la realidad educativa y de presiones económicas, hoy entregamos esta literatura escrita con profesores de CONALEP, máxime reconocimiento, porque trabajaron de manera altruista durante extenuantes y largas jornadas de trabajo; la comunidad CONALEP y la sociedad Michoacana toda, reciban esta obra como testimonio de la grandeza de este histórico Michoacán.

Lic. José Azahir Gutiérrez Hernández
Director Académico

Enfoque por competencias

> "Leer no es una virtud; pero leer bien es un arte, un arte que solo el lector nato puede adquirir. El don de la lectura no es ninguna excepción a la regla de que todos los dones naturales necesitan cultivarse mediante la práctica y la disciplina; pero si la aptitud innata no existe, la formación será inútil. El error del lector mecánico es creer que las intenciones pueden sustituir a la aptitud".
>
> Warthon

El texto digital comunica mensajes apoyados en la producción de texto y las telecomunicaciones, pero además, su cobertura y visibilidad no dependen de la tecnología Web, sino de los códigos culturales que hacen referencia a una lectura local o internacional; el contexto del contenido definido por los términos que los concretan por mucho responden a una comunicación social o interpersonal. Tenga en cuenta que una comunicación personal puede llegar a ser de masas, sin embargo, el texto digital Web más que por cuestiones de compatibilidad de plataformas, encuentra sus límites de poder de comunicación, en la originalidad literaria de sus mensajes (de acuerdo al algoritmo Google Pingüino), es decir, el posicionamiento de una página Web dependerá más de su contenido que de su código artificial de soporte.

Teniendo esto en cuenta, el diseñador Web crea un estilo pensado en la audiencia y en contenidos estructurados en función de los objetivos de comunicación. A partir de este criterio, el texto digital en su capa de lenguaje artificial está formado por HTML5, CSS3 y JavaScript; es este ámbito en el que nos enfocaremos en este libro; la competencia en el manejo de estos lenguajes artificiales, pasa por los imperativos del conocimiento de las funciones técnicas, conceptuales y teóricas en la codificación del diseño de contenidos para la Web, con el fin de que el técnico bachiller adquiera una sólida visión de este tipo de tareas.

En toda obra literaria se afirma una realidad independiente de la lengua y del estilo: la escritura considerada como la relación que establece el escritor con la sociedad, el lenguaje literario transformado por su destino social. Esta tercera dimensión de la forma tiene una historia que sigue paso a paso el desgarramiento de la conciencia: de la escritura transparente de los clásicos a la cada vez más perturbadora del siglo XIX, para llegar a la escritura neutra de nuestros días.

Roland Barthes, *El grado cero de la escritura*

Palabra escrita bajo luz

En un mundo cada día con más canales de comunicación, la palabra escrita camina por los muros que denuncian el drama catastrófico sobre el medio ambiente y sobre el control de la vida humana; el combustible de esta desesperanza produce apatía profunda por tener contacto con el mundo de la literatura, esta distorsión moral parece reflejarse entre los que no quieren sentir responsabilidad ni pensar, dejando a otros su indiferencia al ser prisioneros de ligeras razones y tirria justificada en la empresa de sobrevivir.

Especular en un mundo sin libros es exponer al mundo a la ausencia de pensamiento, creatividad y esperanza. Los libros, dedicados a ser arrebatados por el lector, están expuestos a ser poseídos por las bibliotecas vacías y que con el tiempo opacan sus páginas y empolvan la cubierta de lo que alguna vez fue un objeto de inspiración. Resulta difícil transcribir este instante de un peligroso espacio donde ya muy pocas palabras sobreviven dentro de la reflexión y las pocas sobrevivientes han abandonado la unión del sentido de vivir y el sentido del pensar científico. Escribir pasa de ser un placer repentino a ser una necesidad inminente, es el puente entre lo conocido y lo inexplorado. Es un reto de hoy en día inmiscuirse en lo que una vez fue lo cercano y dejar de lado la novedad tecnológica para poder a través de las barreras que nos ciegan abrir fronteras literarias. Es un proyecto que conspira a favor de la libertad creativa, de la felicidad lúcida cargada de libros embajadores de nuevas realidades.

Entre un mar de razones dentro del libro escolar en crisis, se percibe la ausencia de esa narrativa del cuerpo del texto, misma que alimenta al lector de una experiencia de conocimiento, su ausencia, es más un mal glosario, de un mal armado viaje literario científico o de ficción. En esos viajes de libros en crisis, nos cansamos de mirar espacios vacíos de talento, emociones y sensibilidad para responder a un entorno adverso; son muchas veces un triunfalismo de autoevaluación y una falsa puerta de una real competencia para actuar en la realidad. Uno no solo vive, escucha la voz interior de un libro, uno es fundado en el manejo del lenguaje que explica, crea, aplica o expande los límites del horizonte de nuestro imaginario actuante en lo real. No vivimos leyendo texto, sino leyendo el paisaje de una realidad, el libro toma la voz del progreso en una siempre reconstrucción lingüística del sujeto que explica, transforma y comunica desde los desafíos de su generación.

La información cruda que tanto rellena los libros grises, oscuros y papel pintado; requiere ser dotada de conceptos que permitan alimentar al sujeto que toma decisiones, que explora con paso lento, que mira por dentro del lenguaje y aplica la información que cobra sentido en la siempre expansión de las ideas.

Escribir un libro es siempre reconstruir un discurso, sus lectores en este discurso son el puente a un texto profundo que demanda esfuerzo en la reconstrucción de los procesos de razonamiento y el entretejido del discurso que involucra información de fondo, esas fuentes que justifican su análisis y poseen significado privilegiado para la comprensión de una realidad.

El lector puede hacer uso del libro con su propia experiencia y con su autoayuda, al precisar términos y conceptos para prolongar su horizonte de interpretación, el libro se hace cargo de

la memoria de un plan de estudios, es un discurso de diferentes capas de argumentos, tras este texto se anuncia un orden de experiencia propuesto para su aprendizaje. El libro está conformado para jóvenes con memoria sin dolor para nuevas palabras, aborda el olvido como una deficiencia de interactividad entre el discurso argumentativo y los referentes conceptuales. Esto es el reto en la producción de los libros CONALEP. La propuesta es una reconstrucción de una semántica más profunda, como el principal reto del estudiante técnico bachiller del siglo XXI.

Libro,…
todos te miran,
nosotros te vemos bajo la piel.

SUMARIO

Segunda parte
CSS3

Tercera parte
JavaScript

Debido a las restricciones de extensión del presente libro, las pantallas (imágenes gráficas) que expresan el código no se incluyeron, sin embargo, es posible con la ayuda de un editor como Dreamweaver y un navegador de Internet al cortar y pegar el código obtener este importante recurso. Para ello CONALEP colocara en línea las versiones digitales de este contenido (http://www.cie.umich.mx/conalepweb2013/libros.html)

Capítulo I: HTML5

POEMA DE AMOR BASADO EN HTML5

♥ ♥ ♥

Cuando veo tu ***<figure>***
En mi ***<section>*** irrumpir
Mi ***<body>*** se vuelve loco
Y no hay ***<form>*** de resistir.
***
No seas ***<border>*** conmigo
No me ***<float right>*** ni *left*
Quiero ***<nth-childs>*** contigo
Y a tu ***<aside>*** envejecer
***
Tuyo el ***<article>*** que escribo
Tuyo el ***<canvas>*** que pinté
Y si tu ***<queries>***, yo mismo
desde el ***<header>*** al ***<footer>***

Por Dani B Good.

1.1. Revolución digital

Cuando las personas comenzaron a escribir línea tras línea, en un flujo de instrucciones estructuradas y datos, nace el software como un conjunto de tareas lógicas matemáticas, estas órdenes bajan del lenguaje humano hasta el lenguaje máquina. El lenguaje máquina está compuesto de ceros y unos, hoy no se programa a este nivel, dado que una sola falla de un cero por uno, podría ocasionar un mal funcionamiento del software. Llevaría algunos cientos de años escribir un sistema operativo moderno como Windows 8 a nivel de lenguaje de máquina. Hoy es más generalizado el empleo de la programación de alto nivel de abstracción, es decir, se usa en la ingeniería de sistemas un lenguaje muy próximo al lenguaje natural humano.

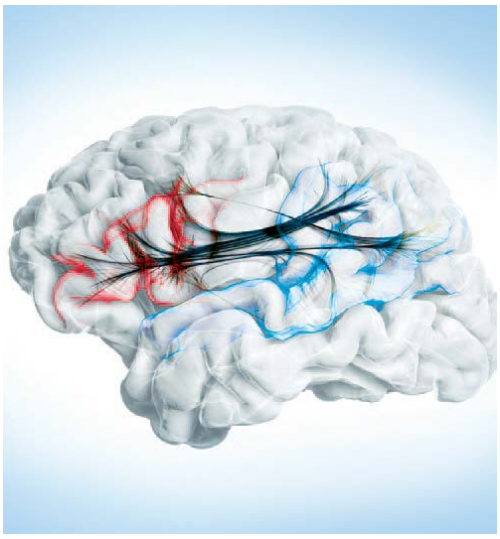


Fig. 1 La nueva era de las neurociencias.[1]

Muy recientemente, las nuevas investigaciones científicas exploran los circuitos de todo nuestro cerebro, se crean modelos de mapas funcionales y se infieren teorías sobre la cartografía de la conectividad **sináptica**[2]. El que hallamos tropezado con la aparición evolutiva de nuestro lenguaje, es sin duda el mayor de los logros biológicos, son facultades intelectuales, procesos complejos y fascinantes recursos cognitivos que permiten alcanzar importantes niveles de alta estructuración de la razón[3]. El lenguaje es concebido como mecanismo de enlace entre los sistemas de la percepción y los sistema responsables del pensamiento, llamado sistemas cognitivos, formados por un lexicón y un sistema de cálculo con operaciones cíclicas de naturaleza gramatical[4]. Esta habilidad genéticamente codificada

para que desarrollemos el lenguaje, depende de acuerdo a los científicos, del procesamiento de la información; sugieren que el aprendizaje es estadístico por inducción de la experiencia del individuo, esto quiere decir que la organización cognitiva del cerebro humano, le permite detectar regularidades gramaticales en el flujo de la información[5]; como resultado podemos procesar el lenguaje (comunicarnos) y con nuestros cerebros, además, como producto de la intensa interlocución con la otredad, alcanzamos el poder de desarrollar lenguajes artificiales[6]. Los niños adquieren el lenguaje para estructurar conocimiento en interminables ciclos de procesamiento e información, caracterizados por la búsqueda de sentido mediante selección de alternativas, comprobación y corrección reconstruida para cada nueva respuesta similar[7].

El gen FOXP2 sugiere que nuestra habla es un recurso lingüístico genético compartido por los cerebros humanos[8], es decir, compartimos los sistemas de interpretación que ayudan a las neuronas espejo a recrear en nuestra mente lo que comunica el lenguaje y crear de esta manera la cultura[9].

El desarrollo del lenguaje humano, alcanzó en este punto la capacidad de crear en forma artificial algunas de las funciones de nuestro cerebro, al revelar parte de sus secretos, nace la revolución digital. Esta revolución implicó la era de la micro y nano electrónica; la era de la información en red (Internet) y la era de la socialización en red encabezada por el texto digital[10].

Hay lenguajes naturales y artificiales, los primeros son producto de una gramática universal genéticamente innata, los segundos, son un lenguaje en el sentido de que solo posee sintaxis y semántica explícita, tienen cancelada la vía pragmática (es decir, la capacidad de producir la cultura). El lenguaje artificial es más un conjunto de reglas semánticas de traducción explícita al lenguaje natural. El lenguaje artificial es de base matemática formal, con derivados en aplicaciones tecnológicas, conocidos como lenguajes informáticos, tales como, ensamblador, orientado a objetos, secuenciales, de marcas o etiquetas, entre otros; donde la declaración de variables, una familia de instrucciones que son estructuradas con propósitos específicos, los distinguen.

Los lenguajes artificiales de representación simbólica, nacen con éxito por la invención de James Cooke Brown (1960), y los llamó *Loglan*, tenían la capacidad de gestión de almacenamiento, recuperación de información y traducción entre comunicaciones hombre-máquina. Un sucesor importante de este, fue el *Lojban*, que representó un gran salto a la familia moderna de lenguajes de programación, para ello se superaron grados de abstracción:

- Código simbólico de bajo nivel, registros, interrupciones (Ensamblador)
- Subrutinas, funciones (Fortran)
- Anidamiento y subprogramas, clases (Pascal)
- Encapsulados (ADA)
- Métodos, módulos (C++, Orientado a objetos)
- Aplicaciones unidad (Orientada a componentes)

Pascal fue concebido con fines académicos y luego encontró fuertes aplicaciones profesionales que le permitieron alcanzar un mayor número de usuarios; con una estructura compleja, fue precursor de estructuras encapsuladas (agrupadas) en una unidad sintáctica, para representar datos y operaciones, incorporó un mecanismo sintáctico *class* para describir la estructura de datos y operaciones, en forma posterior aparecen *cluster, form, módulos*, dentro de C++, VB, Java, C#.

C++ fue uno de los más útiles en los años 90', nace como un lenguaje orientado a objetos, que responde al paradigma orientado a eventos; sin embargo, al ganar terreno el desarrollo de Internet y en la primera década de los años 2000', Java de Sun Microsystems encuentra un lugar importante gracias a que es portable y de movilidad multiplataforma. Ahora la tendencia actual es la programación orientada a componentes (POC), modelo que intenta hacer un mercado de cajas negras que encapsulan funciones ya hechas y probadas, que harán la programación más rápida y robusta, como ejemplos tenemos a .NET (Microsoft), JavaBeans (SUN) y Orbix (OMG).[11]

1.2 Texto digital

Es una evolución que parte de los conceptos de escritura de página, texto y soporte de lectura, nos referimos a los más antiguos. Intentaremos mostrar que el texto digital es el resultado de la convergencia de tres conceptos: lenguajes artificiales, hipertexto y los dispositivos Internet de soporte para producción, distribución, comercialización, promoción y registro de literatura. Dentro de los primeros soportes tecnológicos del texto, se encuentran los rollos o códices, con una unidad de escritura en un flujo continuo; el siguiente gran avance aparece con la idea de página celulosa, que dio origen a un compendio de un texto estático, llamado libro (unidad textual), distribuido en páginas, ofreciendo muchas posibilidades creativas para la producción de manuscritos. En un video de un monje llamado Ansgar, de los más vistos en YouTube en 2010[12], se ejemplifica lo que este salto tecnológico representó para la humanidad, el nacimiento del libro. Poder contar con una lectura en secuencias de páginas, copiar texto y al cerrarlo permanece almacenada la información, el libro frente al códice ofrece más posibilidades literarias, sin embargo, aún no estamos seguros de posibilidades literarias en la página electrónica; así como el poema Blanco de Octavio Paz significó un avance en las posibilidades creativas al modelar al poema como si fueran partituras, las posibilidades topológicas de las páginas electrónicas amplían este horizonte para una poesía híper dimensional. Esto hará que niños en el futuro al encontrar un libro de papel, lo consideren extraño, al ser un medio estático, voluminoso y difícil de almacenar respecto de pantallas ultra delgadas que contienen la posibilidad de almacenar miles de libros, referenciarlos de manera cruzada y acceder a ellos de manera inmediata. Es necesario reflexionar lo sucedido en la literatura con el soporte llamado libro (nacido en Inglaterra en 1604), hasta el surgimiento Google Book, i-book, Amazon, i-tunes, bubok.com.mx entre otros; podremos entonces apreciar las diferentes maneras en que limitan o amplían la posibilidad literaria, seguramente el códice, al parecer recibió críticas duras al igual que ahora el libro electrónico, tenga presente que el formato de soporte no es culpable de que no logremos una buena prosa, poesía o argumentación.

El placer de leer un libro con olor a papel o el leer en tabletas digitales, es el gran paso de Gutenberg a Steve Jobs. La revolución de la imprenta a la revolución digital es una revolución de la democratización del conocimiento, los libreros que adquirían grandes

volúmenes de libros para su exhibición y venta, han casi desaparecido, han reducido el espacio de sus negocios físicos, y ampliado sus sistemas de mercadeo por Internet. El libro es un medio para hacer y comunicar conocimiento, en la historia este objeto ha sufrido de la clandestinidad de su tráfico para evadir la censura; la comercialización para hacer rentable la industria editorial y obtener beneficios para el autor. Visto más profundo, un libro es un objeto tecnológicamente muy complejo que expone textos, los cuales su calidad literaria no está en función del medio de soporte para la lectura y su comercialización, sin embargo, el duro golpe a la actividad de la industria editorial ortodoxa nacida en el siglo XVI, es ya inevitable su desaparición tal como la conocimos, esto ocurrirá en pocos años.

La escritura a lo largo de la historia en sociedades no abiertas, ha experimentado que se le viera como una actividad subversiva, que difunde, fractura y limita los privilegios de pocos que gozan de sus ventajas. Es una tecnología que amplía nuestra memoria histórica, que nos permite reflexionar de manera más profunda, que acrecienta nuestro potencial educativo, que revalora la imagen social del docente y que provoca que nuestra habla participe en el concierto internacional. Además, nuestra habla trasciende a otras generaciones al margen del tiempo. Del papiro aprendimos que conservar el texto era muy limitado en el tiempo, del libro de Gutenberg aprendimos que el conocimiento escrito podría ser accesible a todos, ser el medio por excelencia para articular pensamientos complejos; del texto digital apenas comenzamos a ver todas sus posibilidades, el nacimiento de un nuevo humanismo y literatura digital.

Aprendimos de la palabra escrita frente a la oral, que el intercambio, composición y crítica de las ideas marca diferencias entre las sociedades orales y las sociedades industriales avanzadas de base escrita; la oralidad inclusive lee el discurso escrito como una forma controlada de mayor rigor, elegancia y contenido (oratoria). Los políticos hacen uso de este recurso de oratoria continuamente, los docentes lo han olvidado por error, al confiar en diapositivas y en narrativas improvisadas. Sin duda, la escritura es una actividad recursiva, una y otra vez podemos regresar al texto escrito antes de concluir que este está listo para su publicación, además, esta recursividad permite crear modelos textuales más complejos con características más estéticas, con discursos más extensos para hacer ensayos que replanteen, por ejemplo, nuestra concepción de la sociedad.

El actual contexto digital, no solo replantea los modos de crear literatura, de comunicarla, de preservarla, de comercializarla, …, sino, además, mezclar la oralidad y el texto escrito con tal naturaleza, que nos replantean el espacio literario para nuevos modelos de conocimiento (cine, libro, tesis, ensayo, poesía,…). Con Internet, el texto digital incorpora la fusión de lenguajes naturales y artificiales para presentar las obras literarias. Con la red el texto digital hace del hipervínculo una nueva posibilidad de lectura, de contenido y de información. Ahora, el texto digital está lleno de hipervínculos que hacen referencias a texto completo, correo electrónico, video, animaciones, simulaciones de cálculo; la red no solo supone una nueva forma de llegar a los contenidos, sino la posibilidad de construir un hipertexto moldeado, construido y modificado por la civilización toda. Larry Page y Sergey Brin dieron origen a Google en 1998 y abrieron la puerta a una biblioteca digital con ***Google books***; Steve Jobs con su iPad de Apple en 2010 revolucionó la lectura del libro, su almacenaje, comercialización y su auto publicación con el ***i-books Author***. Pero para que estuviera completa esta revolución faltaría la socialización digital en una escala de más de 600 millones de usuarios, esta la haría Mark Zuckerberg y su ***FaceBook*** para renovar la manera en que interactuamos socialmente los humanos. Sin dejar de lado un desarrollo importante para el texto digital como es el ***procesador de texto***, se pasa del TXT ASCII a Microsoft Word 1997, Bill Gates da un paso decisivo para virtualizar el escritorio de trabajo del escritor, consolidando el formato .DOC, versión anterior al moderno XML ***.DOCX***; un lugar importante en la historia del texto digital lo hace Adobe en 1991 y su formato de documento portable ***PDF***, que se convirtió en norma ISO en 2008, la ISO 32000-1:2008. Pero aún faltaba una pieza clave de la revolución del texto digital, hito que nos permite entender el relanzamiento con éxito del libro electrónico de multiplataforma, el estándar CSS3 del Consorcio World Wide Web (W3C) que en el año 2012 permite que muchos lectores de libros digitales se vuelvan compatibles, gracias al nuevo formato de la industria editorial ***EPUB***, que es un XHTML+CSS2 y que permite además, mantener vigente el PDF. ***Amazon*** de Jeff Bezos, hace realidad el éxito de la librería digital, ya dispone de millones de libros en línea.

Ahora la literatura la podemos imaginar como un ***hipertexto de cobertura global Internet***, Ted Nelson es el responsable de que este texto digital sea una red de texto dinámica. Nada se pierde en la red, porque todo está relacionado con sus términos y semántica. Las diversas

secuencias de lectura del hipertexto no debemos solo verlas como una lectura no secuencial, sino realmente una lectura de hipertexto.

Faltan dos ingredientes clave de esta revolución del texto digital, la enciclopedia ***Wikipedia*** lanzada en 2001 y YouTube en 2005 como repositorio de video bajo demanda basado en Adobe Flash y recientemente en el estándar HTML5.

El texto digital, es un formato por capas de información, entre las que están lenguajes artificiales como Java, HTML, XHTML, CSS, H.264, entre otros, que involucran a tabletas de lectura, Internet, bibliotecas – librerías digitales, y la nueva auto publicación de literatura. Ya en 2012 el 13% de la producción editorial se comercializaba en formato digital, a precios realmente más bajos respecto de la versión analógica (de papel), como se puede constatar en LIBRANDA (http://www.libranda.com) o GANDHI (http://digital.gandhi.com.mx) o planeta libro (http://www.planetadelibros.com).

Herramientas avanzadas contra la piratería como Google Pingüino (http://www.youtube.com/watch?v=9QBcUQ_iNQM) y Grammarly (http://www.grammarly.com) filtran el texto digital para dar prioridad a la originalidad literaria como forma de Ranking para la Web. En la aplicación Grammarly vaciamos hasta este texto y nos arrojó la siguiente evaluación, en la que destaca la originalidad, estilo y vocabulario. Gramaticalmente nos ayuda a perfeccionar nuestra escritura, ortografía y es una garantía de que respetamos la propiedad intelectual.

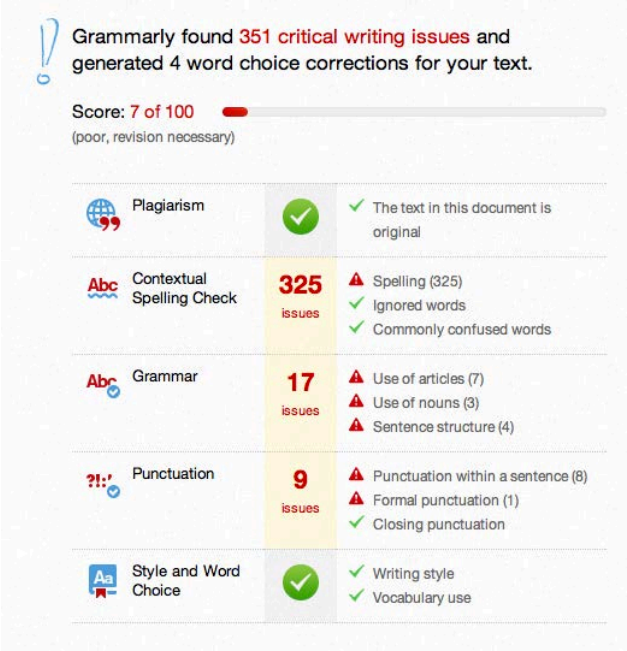


Fig. 2 Evaluador automático gramatical, de originalidad y estilo

La autoedición como la de i-book Author o Bubok, plantea a la educación una nueva posibilidad de libertad de cátedra, en un nuevo y complejo modelo de libro electrónico, donde los roles de autor, lector y texto incrementan sus posibilidades de interacción. Esta revolución del texto digital se fraguó con los jubilados de la era de la guerra fría, universitarios formados para inventar el futuro y centros de investigación con deseos de universalizar el conocimiento.

La página evoluciona, ahora la digital nos permite maquetar, guardar en archivos, escoger el tipo de letra, insertar números de página, notas al pie, cabeceras, referencias, gráficos, configurar su tamaño, insertar video y mucho más, parece lejos la página que inventó la imprenta de Gutenberg como **unidad de lectura**, aquella finita y rugosa de papel celuloso, ahora la página electrónica es una innovación del texto digital, una especie de fusión de capas de lenguaje natural y artificial. El lenguaje artificial detrás del texto natural es el que da su forma dinámica clásica a las páginas Web. Los escritores no necesariamente deben conocer estos lenguajes artificiales, generalmente existen asistentes de diseño que les ayudan a crear sus obras en formato digital. Entre los nuevos cambios a los que rápidamente nos adaptamos están:

- Cortar y pegar secciones de texto digital.
- Digitalización de textos analógicos.
- Creación de libros digitales, en formatos como ePub que permiten maquetar, publicar, teniendo como unidad de lectura la página.
- El hipertexto HTML, XHTML o HTML5 donde el creador multiplica la capacidad de las páginas al introducir bases de datos, animaciones,…

Como fases de diseño

- Producimos el cuerpo de textos (contenido) .DOCX
- Creamos la arquitectura de nuestras páginas (maquetación).
- Seleccionamos los modos en que se dará acceso al contenido.
- Colocamos el contenido en las maquetas.
- Editamos.
- Auto publicamos.

Herramientas de autoedición

https://kdp.amazon.com/self-publishing/signin (KDP)

http://www.lulu.com/publish/?cid=es_tab_publish (LULU)

http://www.apple.com/mx/ibooks-author/ (ibook Author)

http://www.bubok.com.mx (Bubok)

https://mejorando.la/guia-html5/ (aprender HTML5)

Trámite en línea del ISBN

https://www.myidentifiers.com

La tecnología no es un conjunto de objetos, es el desarrollo cultural de la investigación que crea un acervo internacional de conocimientos técnicos acumulados, cuya vigencia no depende del tiempo, sino de la eficacia de sus efectos en nuestro mundo, al punto que su discurso domina ampliamente todos los ámbitos de la vida del hombre, a tal grado, que reinventa la propia manera de socializar, de crear, de evaluar, de hacer democracia, de regular y promover la economía global.[13]

La primera Word Wide Web (Web) fue una caótica red de documentos asociados entre sí (Hipertexto), esto cambiaría por la brillante idea de construir un motor de búsqueda que crearía índices democráticos y confiables de textos en red, permitiera su recuperación

relevante para los usuarios, cuyo lema de la empresa responsable de este salto espectacular es “No seas malo”. Estamos hablando del proyecto Google. Es entonces que el motor de navegación para el ciberespacio de hipertexto, ha obligado a repensar nuestra propia sociedad, llevándonos a debatir intensamente sobre nuevos problemas como el respeto a los derechos de autor y la privacidad, entre muchos otros problemas; sin pretender dar una visión ingenua sobre el potencial de la Web, nuestra generación está obligada a dar una guerra cibernética ética y crítica sobre este recurso que llegó para quedarse en nuestra civilización[14].

Resulta estimulante el propio camino y los valores democráticos del conocimiento que se buscó con la creación de la Web, para sugerir que es importante, por favor no vea al diseño Web como una simple cuestión técnica, es decir, estamos ante un desafío de comunicación y poder[15].

1.3. Lo virtual: el texto digital Web

Lo virtual, es común referirnos al lenguaje y sus productos, piezas virtuosas accesibles, producibles y comunicadas en el ciberespacio producido por computadoras en red, protocolos de Internet y lenguajes HTML, CSS, Javascript, PHP y SQL. Nos referimos a lo virtual, al potencial que contiene razón y poesía para alcanzar a ser parte de nuestra realidad. Debemos superar la falsa idea de que virtual es una falsificación que no alcanza a poseer el potencial de originalidad; virtual es más que la imagen del espejo, destacan los atributos de telepresencia, hipertexto, mediación de multicanales de comunicación, códigos de lenguaje natural sobre lenguajes artificiales que presentan una nueva realidad. La oferta virtual es un paisaje no anclado al tiempo y espacio de la economía materialista, es un producto moral, intelectual y tecnológico que avanza como cibercultura. A los que ya nacieron en la era digital se les suele llamar nativos y a los que no, migrantes digitales; la virtualidad está con nosotros desde los comienzos de la matemática, la poesía, el desarrollo de software y la pintura.

El texto y el concepto de página, evolucionan con las tecnologías del hipertexto, la programación multiplataforma interpretable, los lenguajes de asistencia gráfica al contenido

Web, entre muchos recursos técnicos de nueva generación. Hemos alcanzado tal desarrollo que la propia Web ahora mismo nadie cuestiona que es una referencia virtual que modifica la realidad social con el potencial del texto multicanal hipertextual. Es decir, prosa, audio, video, estilo, ya sean interactivos o estáticos, han potenciado la manera de ver conceptos clásicos como libro, página, correo, conferencia, música, cine, … a tal grado que la socialización humana se ha vuelto horizontal, es decir, sin intermediarios, lo público y lo privado interactúan a nivel global[16].

Es ingenuo modelar una página Web solo como un proyecto de estructura de contenido, comunicación, estilo gráfico y tecnologías a emplear. El potencial de una página Web pasa por los criterios internacionales:

Accesibilidad: facilidad para navegar en las estructuras de información Web, preferentemente en contenido abierto.
Visibilidad: el lugar en la lista en que es desplegada la referencia hipertextual dentro de un navegador de Internet, en función de palabras clave y relaciones semánticas de las mismas. Es decir, es el ranking Web dentro de un navegador.
Impacto: el tráfico de consulta, acceso y descarga de una página Web.
Respeto a la propiedad intelectual: hace transparente los más humildes datos de referencia de originalidad: autor, lugar y fecha, editorial, título, volumen, número, ISBN, DOI, ISSN… para todo documento: fotografías, texto, música, video, …

> "La Ley de Propiedad Intelectual establece el principio general de que la reproducción de una obra protegida, ya sea en todo o en partes, requiere la autorización previa de su autor. Por lo tanto, en principio no es posible reproducir libremente ninguna parte de una obra"…"Nota: No es cierto que no haya ninguna disposición legal que determine ese porcentaje. Lo que determina la Ley es 0%"[17].

Google en su nuevo algoritmo de abril de 2012 de su motor de búsqueda, llamado **Google Pingüino**, busca que esta tecnología pueda dar más visibilidad a los cuerpos de texto de documentos estrictamente originales. Este algoritmo mejora la identificación de contenidos Web que usan **prácticas literarias reales**, no la mera introducción de palabras clave que

engañan a los motores de navegación anteriores como el modelo algorítmico llamado **panda**. El algoritmo Google Pingüino que entra en operación en abril de 2012, intenta[18]:

> **Penalizar la falta de calidad**: la ausencia de coherencia en el contenido, y trata de penalizar las prácticas de enlazar vínculos artificiales al contenido expresado en la página Web en cuestión. Lo mejor, es escribir lo que los lectores quieren leer y colocar hipervínculos de referencia real como fundamento del acto de habla del contenido.

Una página Web, es producto de una buena narrativa del discurso que emplea, no seamos ingenuos que un buen diseño gráfico está por encima de los criterios literarios reales evaluables por los motores de Internet más modernos. Si bien el curso al que responde este libro, tiene el énfasis en HTML5, CSS3 y Javascript en el entorno del diseño de páginas Web, bien vendría empatarlo con el contenido del curso ***Comunicación en los ámbitos escolar y profesional*** del programa de estudios CONALEP; la razón es que los nuevos navegadores penalizan el cortar y pegar, promueven la producción de dispositivos literarios originales, las habilidades y competencias para argumentar, producir discurso, prosa, metáfora, proposiciones y otros muchos aspectos del texto literario científico, técnico, periodístico y de ficción, son esenciales para el diseño Web moderno.

1.4. Diseño básico de una página Web

El diseño básico de una página Web pasa por contestar las siguientes preguntas:

¿Cuál es el objetivo del sitio Web? Se refiere a documentar el concepto del tipo de contenido y los propósitos buscados a nivel de la audiencia, confiabilidad, impacto, accesibilidad y originalidad.

¿Cuál es el perfil de la audiencia potencial (perfil de usuario)? Escolaridad, profesión, idioma, habilidades cibernéticas, y marco ético-cultural del potencial usuario.

¿Recurso de interactividad? Correo, chat, libro electrónico, videoconferencia, red social, telefonía, video bajo demanda, entre muchos de los nuevos modelos electrónicos.

¿Cuáles son las Tecnologías de la Información y la Comunicación (TIC) necesarias para nuestro proyecto?

Garantizar compatibilidad entre plataformas tecnológicas de desarrollo: recurso cliente/servidor y manejo adecuado del pago de licencias para sortear que nuestra página no pierda accesibilidad, visibilidad e impacto.

Tipos de discursos: son las unidades de comunicación implícitas en las secciones, apartados y capitulados del documento HTML; que están en función de claridad, rigor, belleza, extensión, prosa, estilo y calidad gramatical.

Respetar las normas de diseño y no caer en algún delito informático: normatividad gráfica, literaria, ética, técnica y políticas de la empresa en materia de comunicación, todo bajo el respeto al marco legal vigente local y global.

1.5. El hipertexto

Estamos hablando de un texto que es la red de muchos textos, formado por enlaces electrónicos sobre una red. Podríamos decir que el hipertexto está en función de la convergencia de la teoría literaria, la tecnología de hipermedios de comunicación y la producción de nuevas tipologías textuales con el perfil de ser unidades de sentido autónomas, tales como[19] blogs, correo electrónico, mensajería instantánea, muros, …

Los diferentes soportes de la tecnología del hipertexto, permiten la movilidad omnipresente de los usuarios de la información Web, a través de teléfonos inteligentes, tabletas y computadoras portátiles, anteojos de navegación y reloj pulsera Internet. Pero sin importar el mediador entre contenido hipertextual y el lector, debemos reconocer que la escritura, la producción de sentido en audio, video, fotografía, … dependen del talento más allá de la informática envuelta para la producción de páginas Web.

El hipertexto lo podemos reconocer por su principal característica, es un texto digital, como ya lo expresamos, está formado por dos capas, una de texto natural y otra de texto artificial. Es escribir y leer el nuevo soporte del libro electrónico, y todas las otras tipologías de texto consolidadas en la historia epistemológica del hombre (ensayo, poema, artículo, resumen, síntesis,…)[20].

También es necesario advertir que hay innovaciones literarias que no son equivalentes al texto clásico. Es una nueva retórica en la que el flujo de la lectura lo determina el usuario. El diseñador del documento HTML deberá contar con cierta experiencia en la producción de contenido original, de lo contrario es evidente que un mal libro en papel también lo será en su formato de texto digital. Los enlaces lógicos responden a terminologías especializadas, a funcionales textuales como citas, referencias, títulos, subtítulos, secciones, índices, notas, recursos de suplemento, … esta ciencia no es informática, responde a la epistemología (la ciencia de producir, validar y estructurar el saber).

El texto digital no es la mera traducción del texto natural al texto HTML, no es la combinación fortuita de recursos informáticos, es la capacidad creativa de aplicar dentro de las normas producto de la investigación científica, las nuevas prácticas de escritura y lectura en la Web[21]. Fomentar la nueva cultura letrada en la segunda era digital es también responsabilidad de los diseñadores Web[22].

El hipertexto es la versión virtual de la nueva era digital del texto[23], evolución que influye en todas las formas de hacer sociedad[24]: democracia, arte, milicia, economía, …

1.6. Hitos de la Web

Entre los primeros hitos de la historia que cambiaron el concepto de red de cómputo, destaca la respuesta al problema entre ordenadores no compatibles que frustraban la era naciente de Internet (1970), este hito se llamó Lenguaje de Marcado Estándar Generalizado por sus siglas en inglés SGML, este fue el primer lenguaje de hipertexto con el potencial de intercambio de documentos multiplataforma en red. Es un lenguaje interpretable precursor del HTML, es un lenguaje de etiquetas de instrucción con palabras clave de ángulo, por

ejemplo "<h1", para señalar cabeceras, títulos, texto, hipervínculos, pies de página, etc., este rudimentario lenguaje de etiquetas sentó la piedra angular de la moderna Web HTML5, CSS y Javascript. IBM impulsó a partir de este lenguaje SGML el primer protocolo HTML interpretable por navegadores/editores en terminales de usuarios de Internet, permitiendo que estos escribieran sus propios documentos HTML (HyperText Markup Language). HTML es conocido simplemente como lenguaje de marcas, que estructura texto digital vinculado por hiperenlaces a través de la Web. Las etiquetas HTML son palabras clave (nombre de etiquetas) rodeadas por corchetes angulares como <html>, organizadas en pares como y , expresan etiquetas de apertura y cierre. Un elemento HTML está formado por las etiquetas de apertura y cierre, además del contenido entre ellas. En voz de los protagonistas de esta historia, Robert Cailliau, Nicola Pellow y el inventor de la Web Tim Berners-Lee[25] : "la Web ayudaría a la gente a mandar y recibir información entre ellos, sin importar el sistema operativo o los formatos usados por los ordenadores, solo tenían que enviar la URI usando el HTTP o el HTML". Donde la URI es el Identificador Uniforme de Recursos[26], similar al URL, Localizador de Recursos Uniforme que es una cadena de asignación de una dirección única para los recursos disponibles en Internet, como libros, video, música, texto,…

Un URL es por ejemplo: http://www.cie.umich.mx/conalepWeb2013/, y un URI sería: http://www.cie.umich.mx/conalepWeb2013/.

Hitos de la Web

1965: **Ted Nelson acuña el termino Hypertext.**

1967: Se hace operativo el hipertexto.

1969: Dos ordenadores comparten un mensaje.

1971: El primer e-mail es enviado por Ray Tomlinson a un programador de Arpanet.

1974: Arpanet ya era utilizada por empresas civiles.

1981: KRON crea el primer periódico on-line.

1984: Jon Postel propone y crea el primer dominio Web.

1988: Se crea el IRC (mensajería de texto en tiempo real o chat.

1989: Nace la World Wide Web, para interconectar documentos de hipertexto por navegador **Web 1.0.**

1991: Las primeras páginas Web.

1991: Tim Berners-Lee crea y usa por primera vez el URL Web 1.0.

1992: Nace la videoconferencia IP desarrollada por Cornell.

1993: **Se difunde el primer navegador masivo Mosaic 1.0.**

1994: Se impulsa la creación de estándares de Internet por el consorcio W3C.

1994: El estándar URL es incorporado en URI.

1994: Se populariza el navegador Netscape y apararecen Excite, Infoseek, Altavista.

1994: Nace la primera tienda online Pizza Hut.

1995: Nace Yahoo!.

1996: El W3C separa el estilo gráfico de textos, cajas, tablas,…del contenido Web de una página.

1996: **Wium Lie y Bert Bos escriben CSS nivel 1.**

1996: Aparece Amazon y eBay.

1996: Nacen Google y Hotmail.

1996: **Nace JAVASCRIP 1.0, lenguaje de programación para ser interpretado en páginas Web**.

1997: Se crea el protocolo móvil de red inalámbricas.

1997: Se da a conocer el XML (Lenguaje Extensible de Marcado) base del HTML.

1998: **CSS nivel 2.**

1998: PayPal es fundada para el comercio electrónico.

1998: Google se vuelve el buscador dominante de la Web.

2000: Se vivió una crisis conocida como burbuja punto com.

2001: Aparece Wikipedia, el Ipod y **CSS nivel 3.**

2003: iTunes Store y la comercialización de música, libros, películas en la Web.

2004: Las redes sociales MySpace y FaceBook se hacen masivas.

2005: Nace el video bajo demanda YouTube y con el la Web 2.0.

2006: La Web se hace móvil a una escala grande con iPhone.

2006: Aparece el trabajo en la Nube o Cloud Computing.

Entre XHTML ver 1.0 (2000) y la última versión de HTML 4.01 (1999) son pocas las diferencias, se considera XHTML como la versión actual de HTML.

- Los elementos XHTML deben estar correctamente anidados.
- Los documentos XHTML deben ser gramaticalmente correctos.
- Los nombres de etiqueta deben estar en minúsculas.
- Todos los elementos XHTML deben estar cerrados (llevar la etiqueta de cierre).

Nota: Mientras no se indique lo contrario, los ejemplos, presentes en este texto son del tipo de documento: **XHTML 1.0 transitional y HTML5.**

HTML5 no es una nueva versión del HTML 4.01, es una tecnología más adecuada para la era de la Internet móvil, computación en la nube y sistemas de bases de datos de nueva generación. HTML5 es la combinación mejorada de CSS, HTML y Javascript.

1.7. CSS

Las hojas de estilo en cascada, por su siglas en inglés CSS, describen los formatos y aspectos gráficos de los documentos HTML. Un documento adjuntado (hoja de estilo externa) o el mismo texto de lenguaje de marcas (estilo en línea) se complementa con CSS para producir la visión gráfica en las pantallas modernas de las páginas Web. Las hojas de estilo están compuestas por declaraciones de formato y un selector, las declaraciones se forman de propiedad y valor:

Selector es el elemento HTML sobre el que se aplica un estilo.

Propiedad son los atributos a modificar.

Valor son los datos de asignación a la propiedad.

Los selectores pueden aparecer agrupados o individuales, la primera opción exige el uso de comas.

```
selector [, selector2, ...] [:pseudo-clase][::pseudo-elemento] {
 propiedad: valor;
 [propiedad2: valor2;
 ...]
}
/* comentarios */
```

Ejemplo de estructura básica HTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">                    <html
xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
      <title>Documento sin título</title>
      <link rel=”stylesheet” href=”misestilos.css type="text/css”>
      </head>
        <body style=”background-color: #ff0000;”>

      </body>
```

```
</html>
```

La hoja de estilo interna del documento HTML se encuentra dentro de la marca <head> con la etiqueta <style>. Sin embargo, el trabajo de diseño depende de los navegadores empleados, estos se pueden describir en términos de sus componentes para procesar el HTML y mostrar el formato indicado por las CSS. Los motores WebKit, Trident, Gecko, Presto, entre otros, son los responsables de soportar CSS, estos están presentes en Firefox, Google Chrome, Internet Explorer, Safari y Opera[27]. Los motores reconocen todos los selectores, pseudo-clases y propiedades, el soporte moderno de casi todos los navegadores permite observar de la misma manera las páginas, salvo de pequeñas excepciones. Este libro trabajará con componentes de la norma CSS 3 del W3C[28].

Los archivos en los que se presenta el contenido HTML tienen como extensión, por ejemplo index.**html**, donde un servidor Web direccionará las peticiones de nombre de dominio real (**IP**) a través del servidor **DNS**. El protocolo **IP** direcciona al servidor Web, una vez recibida la solicitud de contenido HTML, un **servidor Web** (Apache o IIS) gestiona la página en función del subdominio (carpeta) y nombre del archivo HTML. La URL es la dirección completa que apunta en Internet, podrían ser imágenes, video, correo o en este caso páginas Web. **http** es el protocolo para indicar en la URL que se desea apuntar a una página Web, Apache o IIS son los sistemas informáticos que administran seguridad, acceso y respuesta a tráfico, a este modelo se le llama **cliente/servidor**, además, por todos los continentes están instalados servidores DNS para servir de diccionarios de direcciones, estos traducen las direcciones **IP** solicitadas por los clientes.

Ejemplo de estructura básica de los documentos HTML con estilos definidos sin utilizar CSS

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">                    <html
xmlns="http://www.w3.org/1999/xhtml">
<head>
       <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
       <title>Documento sin título</title>
       </head>
         <body>
```

```
<h1> <font color="red" face="Arial" size="5">Titular del documento</font></h1>
<h2> <font color="gray" face="Arial" size="5">Estructura global HTML</font></h2>
<p>semánticos para estructurar una página Web, incorporan: </p>
      </body>
       </html>
```

1.7.1. Modos de emplear CSS

Nos basaremos a partir de un modelo básico de documento HTML:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">                    <html
xmlns="http://www.w3.org/1999/xhtml">
 <head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
 <title>Documento sin título</title> </head>
<body>
<h1><font color="red" face="Arial" size="5">Titular del documento</font></h1>
<h2><font color="gray" face="Arial" size="5">Estructura global HTML</font></h2>
<p><font color="blue" face="Verdana" size="2"> </font> Los documentos HTML
incorporan elementos </p>
<p>semánticos para estructurar una página Web, incorporan: </p>
<p>&lt;!DOCTYPE&gt;</p>
<p>&lt;html&gt;</p>
<p>&lt;head&gt;</p>
<p>&lt;body&gt;</p>
<p>&lt;meta&gt;</p>
<p>&lt;title&gt;</p>
<p>&lt;link&gt;</p>
/* comentarios */
</body>
 </html>
```

Caso 1) Mediante el atributo **style** aplicado directamente sobre los segmentos del contenido a afectar dentro del cuerpo **<body>.**

Tiene la desventaja de mezclar contenido con los estilos.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"> <html
xmlns="http://www.w3.org/1999/xhtml">
 <head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
 <title>Documento sin título</title> </head>
<body  style=”background-color:  font-family:  'Comic  Sans  MS',  cursive;  color:
#0C0; #fffff;”>
<h1><font color="red" face="Arial" size="5">Titular del documento</font></h1>
<h2><font color="gray" face="Arial" size="5">Estructura global HTML</font></h2>
<p><font  color="blue"  face="Verdana"  size="2">  </font>  Los  documentos  HTML
incorporan elementos </p>
<p>semánticos para estructurar una página Web, incorporan: </p>
<p>&lt;!DOCTYPE&gt;</p>
<p>&lt;html&gt;</p>
<p>&lt;head&gt;</p>
<p>&lt;body&gt;</p>
<p>&lt;meta&gt;</p>
<p>&lt;title&gt;</p>
<p>&lt;link&gt;</p>
/* comentarios */
</body>
 </html>
```

Caso 2) Mediante etiquetas <style> se introducen los cambios de estilo.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"> <html
xmlns="http://www.w3.org/1999/xhtml">
 <head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
 <title>Documento sin título</title> </head>
<body>
<h1><font color="red" face="Arial" size="5">Titular del documento</font></h1>
```

```
<h2><font color="gray" face="Arial" size="5">Estructura global HTML</font></h2>
<p><font color="blue" face="Verdana" size="2"> </font> Los documentos HTML incorporan elementos </p>
<p>semánticos para estructurar una página Web, incorporan: </p>
<p>&lt;!DOCTYPE&gt;</p>
<p style="font-size: 16px; font-family: 'Times New Roman', Times, serif; color: #F90;">&lt;html&gt;</p>
<p style="font-size: 16px; font-family: 'Times New Roman', Times, serif; color: #F90;">&lt;head&gt;</p>
<p>&lt;body&gt;</p>
<p>&lt;meta&gt;</p>
<p>&lt;title&gt;</p>
<p>&lt;link&gt;</p>
/* comentarios */
</body>
</html>
```

Caso 3) Mediante un archivo CSS externo se vincula con la etiqueta <link>, la cual estará en la sección de cabecera <head>.

Tiene la ventaja que se pueden crear todos los archivos CSS que sean necesarios y cada página HTML puede enlazar tantos archivos CSS como necesite. Además, modificando solamente la hoja de estilo, se puede dar una apariencia totalmente diferente al documento HTML.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"> <html
xmlns="http://www.w3.org/1999/xhtml">
 <head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
 <title>Documento sin título</title> </head>
  <link rel="stylesheet" href="misestilos.css" type="text/css">
<body>
<h1><font color="red" face="Arial" size="5">Titular del documento</font></h1>
<h2><font color="gray" face="Arial" size="5">Estructura global HTML</font></h2>
```

```
<p><font color="blue" face="Verdana" size="2"> </font> Los documentos HTML
incorporan elementos </p>
<p>semánticos para estructurar una página Web, incorporan: </p>
<p>&lt;!DOCTYPE&gt;</p>
<p>&lt;html&gt;</p>
<p>&lt;head&gt;</p>
<p>&lt;body&gt;</p>
<p>&lt;meta&gt;</p>
<p>&lt;title&gt;</p>
<p>&lt;link&gt;</p>
/* comentarios */
</body>
 </html>
```

Implementación del caso 3)

1) Se crea un archivo de texto, se agrega el código CSS, por ejemplo:

```
p { color: black; font-family: Verdana; }
```

2) Se guarda el archivo con la extensión: misestilos.css

3) Se enlaza el archivo dentro de una página Web.

```
<link rel="stylesheet" href="misestilos.css type="text/css">
```

Aunque generalmente se usa la etiqueta **<link>** para enlazar los archivos CSS, también es posible hacerlo con la etiqueta **<style>:**

```
<style type="text/css" media="screen">
  @import '/css/misestilos.css';
</style>
```

Dentro de un archivo CSS externo, la primera regla es **@charset**, especifica la codificación de caracteres, generalmente se prefiere utilizar el UTF-8[29]. **@import** permite importar reglas desde otras hojas de estilo.

Caso 4) CSS permite separar los contenidos de la página y la información sobre su aspecto.

Tiene la ventaja de utilizar una zona donde se incluyen todos los aspectos del diseño CSS, se define este caso para un número pequeño de estilos.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"> <html
xmlns="http://www.w3.org/1999/xhtml">
<style type="text/css">
.Miestilo1 { font-family: "Lucida Sans Unicode", "Lucida Grande", sans-serif; }
 </style>

 <head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
 <title>Documento sin título</title> </head>

<body>
<h1><font color="red" face="Arial" size="5">Titular del documento</font></h1>

<h2><font color="gray" face="Arial" size="5">Estructura global HTML</font></h2>
<p><font color="blue" face="Verdana" size="2"> </font> Los documentos HTML
incorporan elementos </p>
<p>semánticos para estructurar una página Web, incorporan: </p>
<p class="Miestilo1">&lt;!DOCTYPE&gt;</p>
<p>&lt;html&gt;</p>
<p>&lt;head&gt;</p>
<p>&lt;body&gt;</p>
<p>&lt;meta&gt;</p>
<p>&lt;title&gt;</p>
<p>&lt;link&gt;</p>
/* comentarios */
</body>
 </html>
```

1.8. HTML5

Tengamos presente que la organización del documento HTML5 básica (2007) W3C está formada por la marca del tipo de documento (<!DOCTYPE>), la raíz (<html>), declara la cabecera del documento HTML (<head>) y enseguida se declara el cuerpo, es la parte visible para el usuario del documento (<body>); además, se incluye dentro de la cabecera el tipo de caracter (<meta>) y se especifica el título del documento (<title>). Un elemento distintivo de este tipo de documentos es la etiqueta que invoca archivos externos, tales como los CSS (<link>).

En el nuevo HTML5, no se requiere especificar el tipo de estilo que estamos usando, solo se incorporan los atributos **rel** (se especifica la relación entre el documento y el archivo **href**) **href; rel** tiene el valor de atributo "**stylesheet**" para indicar al navegador que el archivo **misestilos.css** es un archivo CSS. El archivo .**CSS** contiene un conjunto de reglas sobre la apariencia de nuestra página Web, atributos de texto, tablas, hipervínculos, imágenes, etc., aunque hay otras maneras de declarar estos estilos como lo veremos enseguida.[30]

1.8.1.Estructura global HTML5

Elementos, atributos y valores de atributos en HTML5 se definen (por esta especificación) por tener ciertos significados (semántica). Por ejemplo, el elemento **ol** representa una lista ordenada, y el atributo de **lang** representa el idioma del contenido.

Estas definiciones permiten a los procesadores de HTML5, como navegadores Web o motores de búsqueda, presentar y utilizar documentos y aplicaciones en una amplia variedad de contextos que el autor no podría haber considerado.

Como ejemplo simple, considere una página Web escrita por un autor que solo considera los navegadores Web de escritorio de ordenador. Debido a que HTML5 transmite significado, en lugar de la presentación, la misma página puede ser utilizada también por un pequeño navegador en un teléfono móvil, sin ningún cambio en la página. En lugar de las partidas que están en letras grandes como en el escritorio, por ejemplo, el navegador del

teléfono móvil puede utilizar el mismo tamaño de texto para toda la página, pero con los títulos en negrita.

Pero va más allá que simples diferencias en el tamaño de la pantalla: la misma página, igualmente podría ser utilizada por un usuario ciego utilizando un navegador basado en torno a la síntesis de voz, que en lugar de mostrar la página en una pantalla, lee la página al usuario, por ejemplo, mediante el uso de auriculares. En lugar de texto grande para los encabezados, el navegador del habla podría utilizar un volumen diferente o una voz más lenta.

Eso no es todo, dado que los navegadores saben qué partes de la página son los encabezados, se puede crear un esquema del documento que el usuario puede utilizar para desplazarse rápidamente por el documento, con las teclas de "saltar al siguiente encabezado" o "saltar al encabezado anterior". Estas características son especialmente comunes en los navegadores de voz, donde los usuarios de otro modo encontrarían la navegación de una página bastante difícil.

Incluso, más allá de los navegadores, el software puede hacer uso de esta información. Los motores de búsqueda pueden utilizar las cabeceras con mayor eficacia al indizar una página, o para proporcionar enlaces rápidos a las subsecciones de la página de sus resultados. Las herramientas pueden utilizar las cabeceras para crear una tabla de contenido. Este ejemplo se ha centrado en las cabeceras, pero el mismo principio se aplica a toda la semántica en HTML5.

Aunque los autores no sepan utilizar elementos, atributos o valores de atributos para fines distintos a la finalidad prevista en su semántica, no impide en muchos casos, que el software de procesamiento lo haga correctamente.

Por ejemplo, el siguiente ejemplo es disconforme al estándar HTML5, a pesar de ser sintácticamente correcto:

```
<!DOCTYPE HTML>
<html lang="es">
 <head> <title> Ejemplo1 </title> </head>
 <body>
```

```
  <table>
   <tr> <td> Mi texto favorito del 2013. </td> </tr>
   <tr>
    <td>
     —<a
href="http://www.cie.umich.mx/conalepWeb2013/Matematica.html"><cite>Matemáticas,
lenguaje artificial de tautologías</cite></a>,
     en CONALEP 2013
    </td>
   </tr>
  </table>
 </body>
</html>
```

... debido a que los datos que se colocan en las celdas no son claramente los datos tabulares (y el elemento de cita fue mal utilizado). Esto haría que el software que se ha basado en esta semántica fallara: por ejemplo, un navegador de voz que permite a un usuario ciego navegar por las tablas en el documento, informará de la cita anterior como una tabla, confundirá al usuario. De manera similar, una herramienta que extrae autores de obras de las páginas, que lee: "Matemáticas, lenguaje artificial de tautologías", lo extrae como autor de una obra, a pesar de que en realidad es el título de una obra y no un autor.

Una versión corregida de este documento puede ser:

```
<!DOCTYPE HTML>
<html lang="es">
 <head> <title>Ejemplo1</title> </head>
 <body>
  <blockquote>
   <p> Mi texto favorito del 2013. </p>
  </blockquote>
  <p>
   —<a href="http://www.cie.umich.mx/conalepWeb2013/Matematica.html">Matemáticas,
lenguaje artificial de tautologías</a>,
   en CONALEP 2013
  </p>
 </body>
</html>
```

En el siguiente fragmento de documento, la intención de representar la cabeza titular de un sitio corporativo, es igualmente disconforme porque la segunda línea no está destinada a ser un título de un apartado, sino simplemente una subcabecera o de subtítulo (subordinado en dirección a la misma sección) .

```
<!DOCTYPE HTML>
<html lang="es">
 <head> <title>Ejemplo1</title> </head>
```

```
 <body>

  <h1>CIE Compañía </h1>
  <h2>A la vanguardia en el diseño de contenidos desde 2000</h2>


 </body>
</html>
```

El elemento **hgroup** se emplea para este tipo de situaciones:

```
<!DOCTYPE HTML>
<html lang="es">
 <head> <title>Ejemplo1</title> </head>
 <body>

 <hgroup>
  <h1>CIE Compañia</h1>
  <h2>A la vanguardia en el diseño de contenidos desde 2000</h2>
 </hgroup>

 </body>
</html>
```

Los autores no deben utilizar elementos, atributos o valores de atributos que no están permitidos por esta especificación u otras especificaciones aplicables, ya que esto hace que sea mucho más difícil dominar el lenguaje que se extiende en su desarrollo en el futuro HTML5.

1.8.2. Elementos y atributos HTML5

!DOCTYPE

Es la primera línea del documento HTML5, indica al navegador que leerá marcas tipo HTML5.

```
<!DOCTYPE html>
```

html

El elemento **html** representa la raíz de un documento HTML. Los autores pueden especificar el atributo **lang** en el elemento raíz **html**, dando el idioma al documento. Esto ayuda a herramientas de síntesis de voz para determinar qué pronunciaciones usar, herramientas de traducción para determinar qué reglas debe usar, y así sucesivamente. Esto se refiere a una DTD (Declaración de Tipo de Documento), que implica cuestiones diferentes a lo que el texto dice.

El elemento HTML en el siguiente ejemplo declara que el lenguaje del documento es el español. El **html** es la palabra clave de etiquetas HTML5, **lang** es el atributo y "**es**" es el valor, el cierre será por la etiqueta clave **</html>.**

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Escribir</title>
</head>
<body>
<h1>Escribir es resistir</h1>
<p>Las personas tienen la capacidad de modificar el significado de los mensajes
que reciben en función de su cultura, añaden códigos de analogía apoyados en sus
referentes de realidad; escribir ideas, es un laboratorio de procesos de
representación con una complejidad importante para producir sentido en el lector.
La mente del lector actúa en el flujo de información de un texto, de manera
activa, generando inferencias que se derivan de su experiencia previa.</p>
</body>
</html>
```

head

El elemento de cabecera representa una colección de metadatos para el documento. La colección de metadatos en un elemento de cabecera puede ser grande o pequeña.

He aquí un ejemplo de un documento con modelo pequeño:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Es un documento de cabecera corto</title>
 </head>

<body>

</body>
</html>
```

He aquí un ejemplo largo:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <base href="http://www.example.com/">
  <title>Apliacando el modelo largo de cabecera</title>
  <link rel="stylesheet" href="default.css">
  <link rel="stylesheet alternate" href="big.css" title="Big Text">
  <script src="support.js"></script>
  <meta name="Aplicación CONALEP" content="Proceso de escritura">
 </head>

<body>

</body>
</html>
```

title

El elemento **title** es un elemento secundario necesario en la mayoría de situaciones, pero cuando un protocolo de nivel superior proporciona la información del título, por ejemplo, en la línea de asunto de un e-mail cuando se utiliza HTML como formato de edición de e-mail, el elemento **title** puede ser omitido.

El elemento **<head>** se hereda de los antiguos lenguajes de etiquetas, su función es proveer de información al documento, tales como estilos, archivos externos, códigos javascript, íconos, o incluso imágenes para generar en pantalla. Esta información no es visible al usuario.

body

El elemento **body** representa el contenido principal del documento, tal como funcionaba en los antiguos lenguajes HTML. Exponemos en seguida una visión de este elemento, apoyándonos en lenguaje JavaScript, este último se amplicará su comprensión en el apartado tres del presente libro.

En documentos formales, solo hay un elemento de cuerpo. El atributo **document.body IDL** proporciona secuencias de comandos con fácil acceso al elemento del cuerpo del documento.

El elemento **body** expone contenido controlado por eventos que atribuye una serie de eventos al objeto **Window**. También refleja sus atributos **IDL** al controlador de eventos.

Los controladores de eventos **onblur**, **onerror**, **onfocus**, **onload**, y **onscroll** del objeto **Window**, expuestos en el elemento **body**, son ocultos por los controladores de eventos genéricos que tienen los mismos nombres en otros elementos HTML.

Así, por ejemplo, un evento de **error** surge enviado por un hijo del elemento **body** de un documento, el cual en primer lugar activa el controlador de eventos del atributo **onerror** contenido en ese elemento, y luego el del elemento **html** raíz y, solo entonces, se activará el controlador del evento **onerror** del elemento **body**. Un detector de eventos regularmente fijado al elemento **body** mediante **addEventListener ()**, se puede ejecutar cuando el evento surge a través del **body,** y no cuando llega al objeto **Window**.

Esta página actualiza un indicador que muestra si el usuario está en línea:

```
<!DOCTYPE HTML>
<html>
 <head>
  <title>¿En línea o fuera de línea?</title>
  <script>
   function update(online) {
     document.getElementById('status').textContent =
       online ? 'en linea' : 'fuera de linea';
   }
  </script>
 </head>
 <body ononline="update(true)"
       onoffline="update(false)"
       onload="update(navigator.onLine)">
  <p>Usted esta:  <span id="status">(Desconocido)</span></p>
 </body>
</html>
```

meta

El elemento **meta** representa varios tipos de metadatos que no se pueden expresar con **title**, **base**, **style**, **link** y elementos **script**.

El elemento **meta** puede representar metadatos de nivel de documento con el atributo **name**, directivas **pragma** con el atributo **http-equiv**, y la declaración de codificación de caracteres del archivo cuando un documento HTML es guardado en forma de cadena (por ejemplo, para la transmisión a través de la red o de almacenamiento en disco), con el atributo **charset**.

Exactamente uno de los **name**, **http-equiv**, **charset** o atributos **itemprop** debe especificarse. Si se especifica **name, http-equiv** o **itemprop**, entonces también se debe especificar el atributo **content**. De lo contrario, debe ser omitido.

El atributo **charset** especifica la codificación de caracteres que utiliza el documento. Esta es una declaración de codificación de caracteres. Si el atributo está presente en un documento XML, su valor debe ser una coincidencia de mayúsculas y minúsculas ASCII de la cadena "**UTF-8**" (y por lo tanto, el documento se ve obligado a usar **UTF-8** como codificación).

El atributo de **charset** en el elemento **meta** no tiene ningún efecto en los documentos XML, y solo se permite con el fin de facilitar la migración hacia HTML5 desde XHTML. No debe haber más de un elemento **meta** con un atributo **charset** por documento.

El atributo **content** da el valor de los metadatos del documento, o la directiva **pragma** cuando se utiliza el elemento para esos fines. Los valores permitidos dependen del contexto exacto, tal como se describe en las siguientes secciones de esta especificación. Los metadatos del documento se expresan en términos de pares de nombre-valor; el atributo **name** en el elemento **meta** proporciona el nombre y el atributo contenido en el mismo elemento da el valor. Si un elemento **meta** no tiene atributo de contenido, la parte del valor del par nombre-valor de metadatos es una cadena.

name

Nombres estándar de metadatos. Esta especificación define algunos nombres para el atributo **name** del elemento meta. Los nombres se distinguen entre mayúsculas y minúsculas.

application-name

El valor debe ser una cadena de forma libre corta, con el nombre de la aplicación Web que la página representa. Si la página no es una aplicación Web, no se debe utilizar el nombre de metadatos **application-name**. No debe haber más de un elemento meta con su atributo **name** establecido en el valor **application-name** por documento.

author

El valor debe ser una cadena de forma libre con el nombre de uno de los autores de la página.

descripcion

El valor debe ser una cadena de formato libre que describe la página. El valor debe ser adecuado para su uso en un directorio de páginas, por ejemplo, en un motor de búsqueda. No debe haber más de un elemento **meta** con su atributo **name** establecido en la descripción del valor por documento.

generator

El valor debe ser una cadena de formato libre que identifica a uno de los paquetes de software que se utilizó para generar el documento. Este valor no debe ser utilizado en las páginas cuyo margen de beneficio no es generada por software, por ejemplo, páginas cuyas marcas fueron escritas por un usuario en un editor de texto.

Aquí está lo que una herramienta llamada "Espectrostexto" podría incluir en su salida, en el elemento **head** de la página, para identificarse como la herramienta que se utiliza para generar la página:

<meta name=generator content=" Espectrostexto 1.0">

keywords

El valor debe ser un conjunto de **tokens** (elementos o palabras a las que se le da una autorización) separados por comas, cada uno de los cuales es una palabra clave correspondiente a la página.

Esta página sobre tipos de letra en las autopistas utiliza un elemento meta para especificar algunas palabras clave que los usuarios pueden utilizar para buscar la página:

```
<!DOCTYPE HTML>
<html>

   <head>
  <title>Tipos de letra</title>
  <meta name="keywords" content="british,type face,font,fonts,highway,highways">
 </head>

 <body>

 </body>
</html>
```

Muchos motores de búsqueda no tienen en cuenta estas palabras clave, ya que esta característica se ha utilizado históricamente de maneras poco fiable e, incluso, erróneamente como una forma de resultados de los motores de búsqueda de **spam,** de forma que no es útil para los usuarios.

Directiva **Pragma**

Cuando se especifica el atributo **http-equiv** en un elemento **meta**, el elemento es una directiva **pragma**.(ver http://www.w3c.es)

El atributo **http-equiv** es un atributo enumerado. En la siguiente tabla se muestran las palabras clave definidas para este atributo:

Estado	Palabra clave
Encoding declaration	content-type

Default style	default-style
Refresh	refresh

http-equiv="content-type"

El estado de declaración de codificación es una forma alternativa de establecer el atributo charset: es una declaración de codificación de caracteres.

http-equiv="default-style"

Este pragma establece el nombre del conjunto de hojas de estilo alternativo predeterminado.

http-equiv="refresh"

Este pragma actúa como cronómetro de redirección. En una secuencia de páginas se podría utilizar como una presentación automática, por lo que cada actualización de la página a la página siguiente en la secuencia, aparecería con el uso de marcas como las siguientes:

```
<meta http-equiv="Refresh" content="50; URL=page2.html">
<meta charset="utf-8">
```

En HTML5, para declarar que la codificación de caracteres **UTF-8**, el autor podría incluir la siguiente marca en la parte superior del documento (en el elemento de **head**).

style

El elemento **style** permite a los autores insertar la información de estilo en sus documentos. El elemento **style** es una de varias entradas al modelo de procesamiento de estilo CSS. Este elemento no representa contenido para el usuario. El atributo **type** proporciona el lenguaje de estilo. Si el atributo está presente, su valor debe ser un tipo válido MIME que designa un lenguaje de estilo. No se debe especificar el parámetro **charset**. El valor por defecto para el atributo de **type**, que se utiliza cuando el atributo no está presente, es "**text / css**".

El atributo **media** dice qué estilos se aplican a **media**. El valor debe ser una consulta de **media** válida.

El valor por defecto, si se omite el atributo **media**, es "**all**", lo que significa que los estilos predeterminados se aplican a todos los **media**. El atributo **scoped** es un atributo booleano. Si

está presente indica que los estilos están destinados solo para el subárbol con raíz en el elemento padre del elemento de estilo, en lugar de todo el documento.

Si el atributo **scoped** está presente y el elemento tiene un elemento primario a continuación, el elemento de estilo debe ser el primer nodo de contenido de flujo en su elemento padre que no sea entre elementos espacio en blanco, y el modelo de contenido del elemento padre no debe tener un componente transparente.

Esto implica que solo un elemento de estilo **scoped** se permite a la vez, y que tales elementos no pueden ser de los hijos, por ejemplo, los elementos **a** o **ins**, incluso cuando aquellos se usan como recipientes de contenido dinámico.

El atributo **title** en los elementos de estilo define los conjuntos de hojas de estilo alternativas. Si el elemento **style** no tiene atributo **title**, entonces no tiene título, el atributo **title** anterior no se aplica al elemento de estilo.

El atributo **title** en los elementos **style**, al igual que el atributo **title** en los elementos **link**, se diferencia del atributo **title** global en que un bloque **style** sin un título no hereda el título del elemento padre: simplemente no tiene título.

El siguiente documento tiene su estilo de énfasis para dotar texto de color rojo brillante en lugar de texto subrayado, dejando los títulos de las obras y las palabras latinas en sus cursivas por defecto. Se muestra cómo el uso de los elementos apropiados permite un nuevo y fácil estilo de documentos.

```
<!DOCTYPE html>
<html lang="en-US">
 <head>
  <title>Mi libro favorito</title>
  <style>
   body { color: black; background: white; }
   em { font-style: normal; color: red; }
  </style>
 </head>
 <body>
  <p>Mi libro <em>favorito</em>  es el de un  <em>valiente soñador</em> llamado
  <cite>Hamlet</cite>. El libro es de  <i lang="la">William Shakespeare</i> un
clásico de la literatura.</p>
 </body>
```

```
</html>
```

link

El elemento **link** permite a los autores vincular su documento a otros recursos.

El destino del **link** está dado por el atributo **href**, que debe estar presente y debe contener una dirección **URL** no vacía y válida.

Un elemento **link** debe tener ya sea un atributo **rel** o un atributo **itemprop**, pero no ambos. Si se utiliza el atributo **rel**, el elemento se limita al elemento **head**. Cuando se usa con el atributo **itemprop**, el elemento puede ser utilizado tanto en el elemento **head** como en el **body** de la página, sujeto a las limitaciones del modelo de microdatos.

Los tipos de enlace **link** (de relaciones) vienen dados por el valor del atributo **rel**, que si está presente, debe tener un valor que es un conjunto de símbolos separados por espacio. Las palabras clave permitidas y sus significados se definen en una sección posterior.

Dos categorías **link** se pueden crear con el elemento de enlace: enlaces a recursos externos e hipervínculos. Un ***link*** *tipo sección* definen un tipo de enlace en particular, es un recurso externo o un hipervínculo. Un elemento de enlace puede crear varios enlaces (de los cuales algunos pueden ser enlaces a recursos externos y algunos podrían ser hipervínculos); exactamente cuáles y cuántos enlaces se crean depende de las palabras clave que figuran en el atributo **rel**. Los agentes de usuario deben procesar los elementos de enlace, enalce por enlace, no por elemento.

Cada enlace creado por un elemento de enlace se maneja por separado. Por ejemplo, si hay dos elementos de enlace con **rel = "stysheet**", cada uno de ellos cuenta como un recurso externo separado, y cada uno se ve afectado por sus propios atributos de forma independiente. Del mismo modo, si un solo elemento **link** tiene un atributo **rel** con el valor **`next stylesheet`**, se crea tanto un hipervínculo (para la próxima palabra clave) y un enlace de recursos externos (por la palabra clave **`stylesheet`**), y que están afectados por otros atributos (como **media** o **title**) de manera diferente.

Por ejemplo, el siguiente elemento de enlace crea dos hipervínculos (a la misma página):

<link rel="autor license" href="/about">

El atributo **download**, si está presente, indica que el autor propone el hipervínculo para ser utilizado para la descarga de un recurso. El atributo puede tener un valor, el valor, en su caso, especifica el nombre de archivo predeterminado que el autor recomienda para su uso en el etiquetado de los recursos en un sistema de archivos local. No hay restricciones sobre los valores permitidos.

La siguiente tabla resume los tipos de vínculos que están definidos por esta especificación.

Tipo Link	**Efecto sobre...**		**Breve descripción**
	`link`	`a` y `area`	
`alternate`	Hyperlink	Hyperlink	Proporciona representaciones alternativas del documento actual.
`author`	Hyperlink	Hyperlink	Proporciona un enlace al autor del documento o artículo actual.
`bookmark`	*no se permite*	Hyperlink	Da el enlace de la sección antepasada más cercana.
`help`	Hyperlink	Hyperlink	Proporciona un vínculo a la ayuda sensible al contexto.
`icon`	External Resource	*no se permite*	Importa un icono para representar el documento actual.
`license`	Hyperlink	Hyperlink	Indica que el contenido principal del documento actual está cubierto por la licencia de derechos de autor que se describe en el documento de referencia.
`next`	Hyperlink	Hyperlink	Indica que el documento actual es una parte de una serie, y que el documento siguiente en la serie es el documento de referencia.
`nofollow`	*no se permite*	Annotation	Indica que el autor o editor original del documento actual no respalda el documento de referencia.
`noreferrer`	*no se permite*	Annotation	Requiere que el agente de usuario no envíe un Referer (sic) en encabezado HTTP si el usuario sigue el enlace.
`prefetch`	External Resource	External Resource	Especifica que el recurso de destino debe estar preventivamente en caché.
`prev`	Hyperlink	Hyperlink	Indica que el documento actual es una parte de una serie, y que el documento anterior de la serie es el documento de referencia.
`search`	Hyperlink	Hyperlink	Proporciona un vínculo a un recurso que se puede utilizar para buscar en el documento actual y sus páginas relacionadas.
`stylesheet`	External	*no se permite*	Importa una hoja de estilo.

	Resource		
tag	*no se permite*	Hyperlink	Da una etiqueta (identificado por la dirección dada) que se aplica al documento actual.

Tenga presente que para adjuntar nuestro archivo de estilos CSS solo requerimos de **rel** y **href**. El atributo **rel** refiere a la relación entre el documento y el archivo que se incorpora con **href**. Si **rel** tiene el valor **stylesheet** le decimos al navegador que el archivo misestilos.css es un archivo CSS requerido para presentar en pantalla nuestro documento.

```
<!DOCTYPE html>
<html lang="es">
 <head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">
  <meta name="keyword" content="HTML5, CSS3, Javascript">
  <title> Título del documento </title>
  <link rel="stylesheet" href="misestilos.css">
 </head>
 <body>
   <p><label> Tópico: <input name=topic> <a href="conalep/topic.html"
rel="help">(ayuda)</a></label></p>

 </body>
</html>
```

En HTML5 no es necesario utilizar el especificador **type**, a diferencia del XHTML basta con **rel** y **href.**

base

El elemento **base** permite a los autores especificar la dirección URL base de los documentos para los efectos de resolver las direcciones URL relativas, y el nombre del contexto de navegación para efectos de seguir hiperenlaces. El elemento no representa ningún contenido más allá de esta información.

No debe haber más de un elemento **base** por documento. Un elemento **base** debe tener ya sea un atributo **href**, un atributo **target**, o ambos.

El atributo **href** contenido, si se especifica, debe contener una URL válida. Un elemento **base**, si se tiene un atributo **href**, debe llegar antes de que los otros elementos en el árbol que tienen atributos definidos tengan las direcciones URL, excepto el elemento **html** (su atributo no se ve afectado por elementos **base**).

El atributo **target**, si se especifica, debe contener un nombre válido de navegación contextual o palabra clave, que especifica qué contexto de navegación se va a usar como el valor predeterminado cuando el hipervínculo encausa la navegación del documento.

Un elemento **base,** si tiene un atributo **target**, debe contener los elementos en el árbol que representan los hipervínculos.

En este ejemplo, un elemento **base** se utiliza para establecer la URL base del documento:

```
<!DOCTYPE html>
<html lang="es">
 <head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">
  <meta name="keyword" content="HTML5, CSS3, Javascript">
  <title>Este es un ejemplo del elemento base</title>
  <link rel="stylesheet" href="misestilos.css">

  <base href="http://www.cie.umich.mx/conalepWeb2013/">

 </head>
  <body>
  <p>Visite los <a href="SPAI.html">archivos</a>.</p>

  </body>
</html>
```

script

El elemento **script** permite a los autores incluir scripts dinámicos y bloques de datos en sus documentos. El elemento no representa contenido para el usuario.

Cuando se usa para incluir **scripts** dinámicos, las secuencias de comandos o bien pueden ser incorporadas en línea o se pueden importar desde un archivo externo usando el atributo **src**. Si el lenguaje no es el descrito por "**text / javascript**", el atributo **type** debe estar presente,

como se describe a continuación. Sea cual sea el lenguaje que se utilice, el contenido del elemento **script** debe cumplir con los requisitos de la especificación de ese lenguaje.

Cuando se usa para incluir bloques de datos (en lugar de secuencias de comandos), los datos deben estar incrustados en línea. El formato de los datos debe ser dado usando el atributo **type**, el atributo **src** no se debe especificar, y el contenido del elemento **script** debe cumplir con los requisitos definidos para el formato empleado.

El atributo **type** da el lenguaje del **script** o el formato de los datos. Si el atributo está presente, su valor debe ser un tipo **MIME** válido. No se debe especificar el parámetro **charset**. El valor por defecto, que se utiliza cuando el atributo no está presente, es "**text / javascript**".

El atributo **src**, si se especifica, da la dirección del recurso de **script** externo para su uso. El valor del atributo debe ser una URL no vacía válida, que identifica un recurso de **script** del tipo dado por el tipo de atributo, si el atributo está presente, "**text / javascript**". Un recurso **script** de un tipo dado, identifica un lenguaje de programación y el recurso se ajusta a los requisitos de la especificación del lenguaje.

El atributo **charset** da la codificación de caracteres del recurso de **script** externo. El atributo no se debe especificar si el atributo **src** no está presente. Si se establece el atributo, su valor debe ser un nombre de codificación de caracteres válidos, debe ser una coincidencia de mayúsculas y minúsculas ASCII para el nombre preferido para esa codificación **MIME**, y debe coincidir con la codificación dada en el parámetro **charset** de los metadatos de tipo de contenido del archivo externo, en su caso. Los atributos **async** y **defer attributes** son booleanos que indican cómo se debe ejecutar la secuencia de comandos. Los atributos **defer** y **async** no se deben especificar si el atributo **src** no está presente.

Existen tres modos posibles que se pueden seleccionar usando estos atributos. Si el atributo asíncrono **async** está presente, entonces la secuencia de comandos se ejecuta de forma asíncrona, tan pronto como esté disponible. Si el atributo asíncrono **async** no está presente, pero el atributo aplazar **defer** está presente, entonces la secuencia de comandos se ejecuta

cuando la página ha terminado el análisis. Si ninguno está presente, entonces la secuencia de comandos se recupera y ejecuta inmediatamente, antes de que el agente de usuario continúe analizando la página.

Los detalles exactos de procesamiento para estos atributos son, sobre todo por razones históricas, algo no trivial, que implica un número de aspectos de HTML. Los requisitos de implementación, por lo tanto fueron dirigidos por los requerimeintos de memoria. Los algoritmos siguientes (en esta sección) describen el núcleo de este proceso, pero estos algoritmos de referencia, hacen referencia a las reglas de análisis de las secuencia de comandos, a las etiquetas de cierre de HTML, al contenido externo, a XML, a las reglas del método document.write (), manejo de secuencias de comandos, etc.

El atributo **defer** puede especificarse incluso si se especifica el atributo **async**.

En este ejemplo, se utilizan dos elementos de **script**. Se incrusta un **script** externo, y el otro que incluye algunos datos.

```
<script src="juego.js"></script>
<script type="text/x-juego-mapa">
........U.........e
o............A....e
.....A.....AAA....e
.A..AAA...AAAAA...e
</script>
```

Los datos en este caso pueden ser utilizados por la secuencia de comandos para generar el mapa de un videojuego. Los datos no tienen que ser utilizados de esa manera, sin embargo, tal vez los datos del mapa son en realidad incrustados en otras partes del etiquetado de la página, y el bloque de datos aquí solo se utiliza en el motor de búsqueda del sitio para ayudar a los usuarios a buscar en particular características en sus mapas del juego.

A continuación se enumeran las cadenas de tipo **MIME** que el agente debe reconocer, y los lenguajes a los que se refiere:

```
"application/ecmascript"
"application/javascript"
"application/x-ecmascript"
"application/x-javascript"
"text/ecmascript"
"text/javascript"
"text/javascript1.0"
```

```
"text/javascript1.1"
"text/javascript1.2"
"text/javascript1.3"
"text/javascript1.4"
"text/javascript1.5"
"text/jscript"
"text/livescript"
"text/x-ecmascript"
"text/x-javascript"
JavaScript.
"text/javascript;e4x=1"
JavaScript with ECMAScript for XML.
```

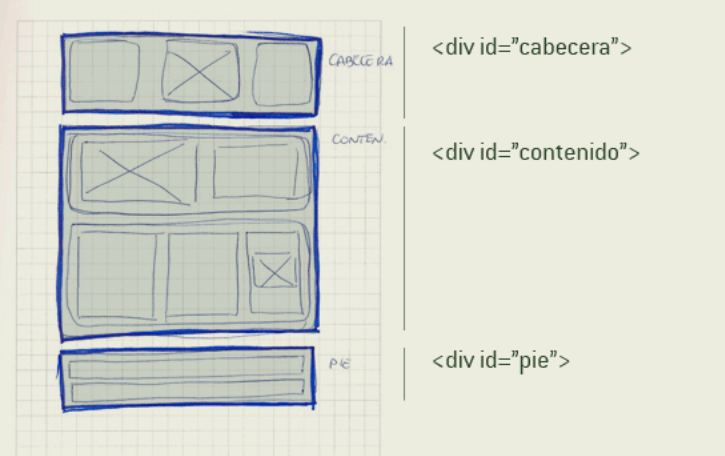
```
<script src="cool-effects.js">
 // crea una nueva instancia:
 //    var e = new Effect();
 // Comienza el efecto usando play, stop:
 //    e.play();
 //    e.stop();
</script>
```

Se profundizará más en la sección JavaScript

1.8.3. Estructura del cuerpo HTML5

Para organizar la maqueta que alojará el contenido que mostrará la página Web, en principio se usó el elemento tabla **<table>,** conforme evolucionó la Web se incorporó un elemento más avanzado y ahora mismo domina el diseño de páginas HTML, se trata del elemento , especifica una división del cuerpo, como la antigua celda de una tabla, además incluye qué clase de división es, el propósito y lo que contendrá.

Con la aparición de tabletas y teléfonos inteligentes este elemento se hace indispensable para adaptar la página Web al formato de cada pantalla, responde al modelo de diseño de maquetación, es decir, a la organización de mapa de contenido de la página.



Por ejemplo maquetamos tres columnas con diseño fijo y Hojas de Estilo en Cascada (CSS).

div

El elemento **div** no tiene ningún significado especial. Representa a sus hijos. Se puede utilizar con los atributos class, **lang** y **title** para marcar semántica común a un grupo de elementos consecutivos.

Se anima a los autores a ver el elemento **div** como un elemento de último recurso. El uso de los elementos más apropiados en lugar del elemento **div** conduce a una mejor accesibilidad para los lectores y más fácil mantenimiento para los autores.

Por ejemplo, una entrada de blog estaría marcada por el uso del artículo, un capítulo con la sección, ayudas a la navegación de una página y un grupo de controles de formulario usando **fieldset**.

Por otro lado, los elementos **div** pueden ser útiles para los propósitos estilísticos o para envolver varios párrafos dentro de una sección que son todos anotados en una manera similar. En el siguiente ejemplo, vemos elementos **div** utilizados como una forma de cambiar el lenguaje de dos párrafos a la vez, en lugar de ajustar el idioma de los dos elementos de párrafo separado:

```
<!DOCTYPE html>
<html lang="es">
 <head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">
  <meta name="keyword" content="HTML5, CSS3, Javascript">
  <title>Este es un ejemplo del elemento div</title>
  <link rel="stylesheet" href="misestilos.css">
  <title>Online or offline?</title>


 </head>
 <body>

<article lang="en-US">
 <h1>Su de dos lenguajes </h1>
 <p>Today digital competition in the production of scientific information is a
must for the beginner in the knowledge society. The knowledge economy is to
evaluate criteria for competitive funding processes and respond to accreditation
systems of higher education quality according to the institutional competence in
scientific and technical production.</p>
 <div lang="es">
  <p>Hoy en día la competencia digital en la producción de información científica
es una necesidad para quien se inicia en la sociedad del conocimiento. La
economía del conocimiento tiene como criterio evaluar los procesos para fondos
concursables y responder a los sistemas de acreditación  de la calidad de la
educación superior en función de las competencias institucionales en la
producción científica y técnica.</p>
  <p>(...) no hay conocimiento sino en función de una cierta reorganización del
conocimiento ...</p>
 </div>
 <p>Invitamos a leer!</p>
</article>

 </body>
</html>
```

En el siguiente ejemplo, vemos elementos **div** utilizados como una forma de crear la maqueta de la página Web.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Documento sin título</title>

</head>

<style type="text/css">
      BODY {
   font: 8pt Verdana, Geneva, Arial, Helvetica, sans-serif;
   margin: 10 0 10 0px;
   text-align: center;
   background-color: #ebebeb;
}
#contenedor{
   text-align: left;
   margin: auto;
}
#cabecera{
   background-color: #d0d0ff;
   color: #333300;
   font-size:12pt;
   font-weight: bold;
   padding: 3 3 3 10px;
}
#cuerpo{
   margin: 10 0 10 0px;
}
#lateral{
   width: 160px;
   background-color: #999999;
   float:left;
}
#lateral ul{
   margin : 0 0 0 0px;
   padding: 0 0 0 0px;
   list-style: none;
}
#lateral li{
   background-color: #ffffcc;
   margin: 2 2 2 2px;
   padding: 2 2 2 2px;
   font-weight: bold;
}
#lateral a{
   color: #3333cc;
   text-decoration: none;
}
#principal{
   margin-left: 170px;
   background-color: #ffffff;
   padding: 4 4 4 4px;
   margin-right: 130px;
}

#otrolado{
      width: 120px;
      float: right;
```

```
}

#pie{
   background-color: #cccccc;
   padding: 3 10 3 10px;
   text-align:right;

   clear: both;
}
     </style>


<body>

 <div id="contenedor">
    <div id="cabecera">
       Cabecera 01
    </div>
    <div id="cuerpo">
       <div id="lateral">
          <ul>
             <li><a href="#">Enlace 1</a>
             <li><a href="#">Vínculo 2</a>
             <li><a href="#">Otro enlace</a>
             <li><a href="#">Link casa</a>
             <li><a href="#">Más enlaces</a>
             <li><a href="#">Otro último</a>
          </ul>
       </div>
       <div id="otrolado">
         <img src="banerlateral.png" width="120" height="600" alt="">
       5</div>
       <div id="principal">
           Diseño de empresa CONALEP

       </div>
    </div>
    <div id="pie">
       © 2013 DesarrolloWebCONALEP
    </div>
</div>

</body>
</html>
```

body

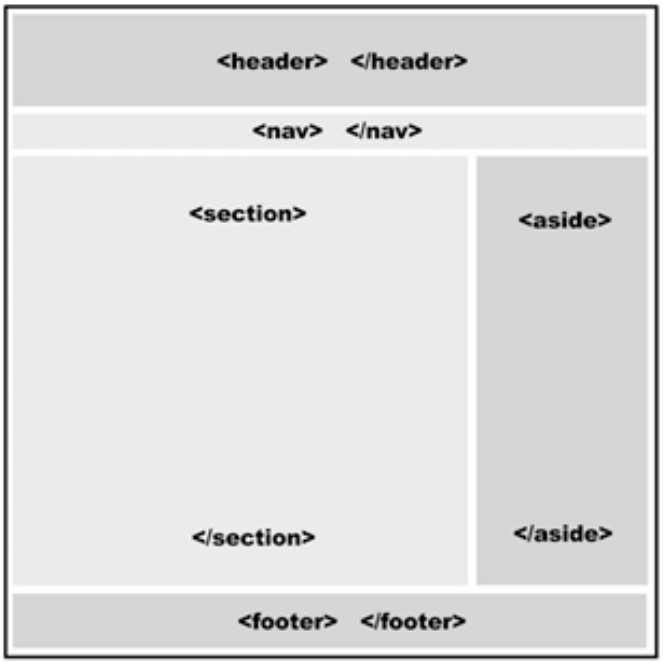
El elemento **body** representa el contenido principal del documento. En documentos solo hay un elemento **body**.

Esta página se actualiza con un indicador que muestra si el usuario está en línea:

```
<!DOCTYPE html>
<html lang="es">
 <head>
  <title>Mi libro favorito</title>
```

```
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">
  <meta name="keyword" content="HTML5, CSS3, Javascript">
  <title>Este es un ejemplo del elemento base</title>
  <link rel="stylesheet" href="misestilos.css">
  <title>Online or offline?</title>
  <script>
   function update(online) {
     document.getElementById('status').textContent =
       online ? 'Online' : 'Offline';
   }
  </script>
 </head>
 <body ononline="update(true)"
       onoffline="update(false)"
       onload="update(navigator.onLine)">
  <p>Usted esta: <span id="status">(estado no definido)</span></p>

 </body>
</html>
```

En este modelo se presenta la página en el paradigma F[31], Cabecera, Barra de Navegación, secciones de Información principal, Barra Lateral y Pie o Barra Institucional. A partir de este momento le diremos al navegador los fines de cada sección.



HTML5 incluye etiquetas para especificar cada sección de contenido de acuerdo al modelo de la figura anterior.

header

El elemento **header** representa un grupo de ayudas de introducción o de navegación, está dentro de las etiquetas **body**. **Body** es el comienzo del cuerpo del documento visible para el usuario, esta cabecera de documento visible contiene títulos, subtítulos, logos y entrada de búsqueda de contenido. Un elemento **header** está destinado a contener, por lo general, la partida de la sección (un elemento **h1-h6** o un elemento **hgroup**), pero esto no es necesario. El elemento **header** también se puede utilizar para envolver tablas, secciones de contenidos, un formulario de búsqueda, o cualquier logotipo correspondiente.

```
<!DOCTYPE html>
<html lang="es">
 <head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">
  <meta name="keyword" content="HTML5, CSS3, Javascript">
  <title>Este es un ejemplo del elemento base</title>
  <link rel="stylesheet" href="misestilos.css">
  <title>Online or offline?</title>
```

```
 </head>
 <body>
<header>
 <p>bienvenidos a CONALEP ...</p>
 <h1>comencemos a revelar el conocimiento tecnológico!</h1>
</header>


 </body>
</html>
```

El siguiente código muestra cómo puede ser utilizado el elemento **header** para marcar un pliego de condiciones:

```
<!DOCTYPE html>
<html lang="es">
 <head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">
  <meta name="keyword" content="HTML5, CSS3, Javascript">
  <title>Este es un ejemplo del elemento header</title>
  <link rel="stylesheet" href="misestilos.css">


 </head>
 <body>
<header>
 <hgroup>
  <h1>Escalado grafico vectorial (SVG) 1.2</h1>
  <h2>W3C t 27 October 2004</h2>
 </hgroup>
 <dl>
  <dt>Las versiones:</dt>
  <dd><a href="http://www.w3.org/TR/2004/WD-SVG12-
20041027/">http://www.w3.org/TR/2004/WD-SVG12-20041027/</a></dd>
  <dt>Previous version:</dt>
  <dd><a href="http://www.w3.org/TR/2004/WD-SVG12-
20040510/">http://www.w3.org/TR/2004/WD-SVG12-20040510/</a></dd>
  <dt>La siguiente versión SVG 1.2:</dt>
  <dd><a href="http://www.w3.org/TR/SVG12/">http://www.w3.org/TR/SVG12/</a></dd>
  <dt>Recomendaciones:</dt>
  <dd><a href="http://www.w3.org/TR/SVG/">http://www.w3.org/TR/SVG/</a></dd>
  <dt>Editor:</dt>
  <dd>Dean Jackson, W3C, <a href="mailto:dean@w3.org">dean@w3.org</a></dd>
  <dt>Autores:</dt>
  <dd>See <a href="#authors">Author List</a></dd>
 </dl>
 <p class="copyright"><a href="http://www.w3.org/Consortium/Legal/ipr-notic"/p>
</header>


 </body>
</html>
```

En el siguiente ejemplo la página tiene un encabezamiento propuesto por el elemento **h1**, y dos subsecciones cuyos encabezamientos están dados por los elementos **h2**. El contenido,

después de que el elemento **header** sigue siendo parte del último inciso, se inició en el elemento **header**, ya que el elemento de cabecera no participa en el algoritmo de esquema.

```
<!DOCTYPE html>
<html lang="es">
 <head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">
  <meta name="keyword" content="HTML5, CSS3, Javascript">
  <title>Este es un ejemplo del elemento header</title>
  <link rel="stylesheet" href="misestilos.css">

 </head>
 <body>
<body>
 <header>
  <h1>La poesía en el CONALEP</h1>
  <nav>
   <ul>
    <li><a href="/games">El placer de lo humano</a>
    <li><a href="/forum">Foro de la metáfora </a>
    <li><a href="/download">Descarga de PDF</a>
   </ul>
  </nav>
  <h2>Poemas modernos</h2> <!-- esto comienza un segundo párrafo ->
   <- Esto es parte de la subsección titulada "Poemas modernos" -->
  <p>Para la poesía hoy tendrá que actualizar su espíritu</p>
  <h2>Poetas del hoy</h2> <!-- esto comienza un tercer párrafo -->
 </header>
 <p>Tiene tres nuevo poemas:</p>
 <!-- esto sigue siendo parte de la subsección titulada "Poetas del hoy" -->

 </body>
</html>
```

nav

El elemento generador de la barra de navegación está dentro de las etiquetas **body**. El elemento **nav** representa una sección de una página que enlaza a otras páginas o a partes dentro de la página: una sección con enlaces de navegación.

No todos los grupos de enlaces en una página necesitan estar en un elemento de navegación, el elemento está destinado principalmente a las secciones que constan de grandes bloques de navegación. En particular, es común que los pies de página tengan una lista de enlaces a diferentes páginas de un sitio, como las condiciones del servicio, la página principal, y una página de derechos de autor. El elemento de pie de página, por sí solo, es

suficiente para tales casos, mientras que un elemento de navegación por lo general es innecesario.

Los agentes de usuario (como los lectores de pantalla) pueden beneficiarse de la información de navegación disponible de inmediato, pueden utilizar este elemento como una forma de determinar el contenido de la página.

En el siguiente ejemplo, la página tiene varios lugares donde los enlaces están presentes, pero solo uno de esos lugares se considera una sección de navegación.

```
<!DOCTYPE html>
<html lang="es">
 <head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">
  <meta name="keyword" content="HTML5, CSS3, Javascript">
  <title>Este es un ejemplo del elemento header</title>
  <link rel="stylesheet" href="misestilos.css">


 </head>
 <body>
<body>
 <body itemscope itemtype="http://schema.org/Blog">
 <header>
  <h1>CONALEP en el vértigo del cambio tecnológico!</h1>
  <p><a href="news.html">Noticias</a> -
     <a href="blog.html">Blog</a> -
     <a href="forums.html">Foros</a></p>
  <p>Última modificación: <span itemprop="dateModified">2013-07-21</span></p>
  <nav>
   <h1>Navegación</h1>
   <ul>
    <li><a href="articles.html">Índice de objetos de aprendizaje</a></li>
    <li><a href="today.html">Cosas que tienen que despertar para hoy</a></li>
    <li><a href="successes.html">Relación de eventos</a></li>
   </ul>
  </nav>
 </header>
 <div>
  <article itemprop="blogPosts" itemscope
itemtype="http://schema.org/BlogPosting">
   <header>
    <h1 itemprop="headline">En la frontera de la tecnología</h1>
   </header>
   <div itemprop="articleBody">
    <p>El conocimiento técnico es acumulativo</p>
    ...más contenido en función de la eficacia observada en la realidad...
   </div>
   <footer>
    <p>La personalidad necesaria para el texto técnico: <time
itemprop="datePublished" datetime="2009-10-10">carácter,
```

```
perseverancia.</time></p>
   </footer>
  </article>
  ...terminología especializada...
 </div>
 <footer>
  <p>Copyright ©
   <span itemprop="copyrightYear">2013</span>
   <span itemprop="copyrightHolder">CONALEP Michoacán</span>
  </p>
  <p><a href="about.html">Acerca </a> -
     <a href="policy.html">Política de privacidad</a> -
     <a href="contact.html">Contacto</a></p>
 </footer>


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

Observe los elementos **div,** se utilizan para envolver todo el contenido de la página que no sea el encabezado y pie de página, y todo el contenido de la entrada del blog que no sea el encabezado y pie de página. También puede ver las anotaciones de microdatos en el ejemplo anterior, que utiliza el vocabulario schema.org para proporcionar la fecha de publicación y otros metadatos acerca de la entrada del blog. http://schema.org ofrece una colección de esquemas, es decir, etiquetas HTML, que los Webmasters pueden utilizar para el marcado de sus páginas de manera que sean reconocidos por los principales proveedores de búsqueda en Internet. Los motores de búsqueda como Bing, Google, Yahoo! y Yandex dependen de este marcado para mejorar la visualización de los resultados de búsqueda, lo que facilita a los usuarios encontrar las páginas Web adecuadas.

Muchos sitios se generan a partir de datos estructurados, que a menudo se almacenan en bases de datos. Cuando estos datos tienen el formato en HTML, esto hace que se vuelva muy difícil de recuperar en su forma estructurada original. Muchas aplicaciones, especialmente los motores de búsqueda, se pueden beneficiar en gran medida del acceso directo a estos datos estructurados. En la página de marcado permite a los motores de búsqueda entender la información de las páginas Web y proporcionar resultados de búsqueda más ricos con el fin de hacer más fácil para los usuarios encontrar la información relevante en la Web. El marcado también puede permitir nuevas herramientas y aplicaciones que hacen uso de la estructura de datos.

Un vocabulario de marcado hace que sea más fácil para los Webmasters decidir sobre un esquema de marcado y obtener el máximo beneficio de sus esfuerzos. Así, en el espíritu de http://www.sitemaps.org/es/, los motores de búsqueda se han unido para ofrecer una colección compartida de los esquemas que los Webmasters pueden utilizar para tal empresa.

En el siguiente ejemplo, hay dos elementos de navegación, uno para navegación principal alrededor del sitio, y otro para la navegación secundaria alrededor de la propia página.

```
<!DOCTYPE html>
<html lang="es">
 <head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">
  <meta name="keyword" content="HTML5, CSS3, Javascript">
  <title>Este es un ejemplo del elemento header</title>
  <link rel="stylesheet" href="misestilos.css">



 </head>
<body>
 <h1>Centro  Wiki CONALEP</h1>
 <nav>
  <ul>
   <li><a href="/">Inicio</a></li>
   <li><a href="/events">Actualidad</a></li>
   ...más...
  </ul>
 </nav>
 <article>
  <header>
   <h1>Demos in Exampland</h1>
   <p>Escritos por E.O.H. et al.</p>
  </header>
  <nav>
   <ul>
    <li><a href="#public">Información pública</a></li>
    <li><a href="#destroy">Información interna</a></li>
    ...más...
   </ul>
  </nav>
  <div>
   <section id="public">
    <h1>Información pública</h1>
    <p>...más...</p>
   </section>
   <section id="destroy">
    <h1>Información interna</h1>
    <p>...más...</p>
   </section>
   ...más...
  </div>
  <footer>
   <p><a href="?edit">Editorial Conalep</a> | <a href="?delete">Borrar</a> | <a
```

```
href="?Rename">Renombrar</a></p>
  </footer>
 </article>
 <footer>
  <p><small>© copyright 2013 Conalep</small></p>
 </footer>
</body>
</html>
```

Un elemento de navegación no tiene que contener una lista, puede contener otros tipos de contenido. Una alternativa es que en este bloque de navegación, por ejemplo, los enlaces se proporcionen en prosa:

```
<!DOCTYPE html>
<html lang="es">
 <head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">
  <meta name="keyword" content="HTML5, CSS3, Javascript">
  <title>Este es un ejemplo del elemento header</title>
  <link rel="stylesheet" href="misestilos.css">

 </head>
<body>
 <h1>Centro  Wiki CONALEP</h1>
 <nav>
 <h1>Navegación</h1>
 <p>Usted está en página de inicio. El página le estructura el contenido <a
href="/blog">
 blog</a>,  <a
 href="/school"> institucional</a> y comentarios públicos. El Colegio Nacional de
Educación Profesional Técnica (CONALEP) es un organismo público descentralizado
del Estado, que forma profesionales técnicos para el mercado laboral y
profesionales técnicos bachilleres para continuar sus estudios de educación
superior.<a href="/school/thesis">Filosofía</a>.</p>
 <p>La  educación como reforma moral se entenderá como justicia social y
compromiso institucional.<a
 href="http://www.cie.umich.mx/conalepWeb2013/">"CONALEP"</a>.a educación
tecnológica es el mecanismo en el que se amplían los horizontes de la realidad
con ayuda de técnicas y plataformas tecnológicas, donde principios técnicos
encuentran una explicación y una real aplicación para la intervención eficaz del
técnico bachiller en la transformación de la realidad.<a
 href="http://www.cie.umich.mx/conalepWeb2013/SPAI.html">SPAI™</a>.</p>
 <p>Atreveos a conocer, la lluvia de palabras que alimentan los ríos de
literatura, pueden al hundirse en la tierra sedienta restablecer la creatividad,
como la ciencia de hacer conciencia. <a href="/about">Contacto</a>. Colegio de
Educación Profesional
Técnica del Estado de Michoacán,
Gral. Nicolás Bravo # 144 Col. Chapultepec Norte  Morelia, Michoacán, México.</p>
</nav>
</body>
</html>
```

section

El elemento **section** representa una sección genérica de un documento o aplicación. Una sección, en este contexto, es una agrupación temática de los contenidos, por lo general con una partida.

Ejemplos **section** serían capítulos, las diversas páginas con pestañas en un cuadro de diálogo con fichas, o las secciones numeradas de una tesis. La página de inicio de un sitio Web se puede dividir en secciones para una introducción, noticias e información de contacto.

Los autores pueden usar el elemento **article** en lugar del elemento **section** cuando tenga sentido señalar el contenido del elemento.

El elemento **section** no es un elemento contenedor genérico. Cuando se necesita un elemento solo para fines de estilo o como una conveniencia para **scripting**, se anima a los autores a utilizar en su lugar el elemento **div.** Una regla general es que el elemento **section** es apropiado solo si el contenido del elemento se enumera explícitamente en el esquema del documento.

El siguiente ejemplo es un artículo (parte de una página Web más grande) acerca de cómo vivir la literatura, el cual contiene dos secciones cortas.

```
<!DOCTYPE html>
<html lang="es">
 <head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">
  <meta name="keyword" content="HTML5, CSS3, Javascript">
  <title>Este es un ejemplo del elemento header</title>
  <link rel="stylesheet" href="misestilos.css">


 </head>
<body>
 <h1>Centro  Wiki CONALEP</h1>
 <article>
 <hgroup>
  <h1>Literartura</h1>
  <h2>Placer, de lo humano!</h2>
 </hgroup>
```

```
 <p>Leer literatura de ficción, se gana tolerancia social.</p>
 <section>
  <h1>Lenguaje literario</h1>
  <p>Formas creativas para contar la más perfecta ficción.</p>
 </section>
 <section>
  <h1>William Shakespeare</h1>
  <p> El viaje a nuestros infiernos, para renovar nuestro sentido de vida.</p>
 </section>
</article>
</body>
</html>
```

Observe cómo el uso de la etiqueta **section** significa que el autor puede utilizar los elementos **h1**, sin tener que preocuparse acerca de si una sección en particular se encuentra en el nivel superior, el segundo nivel, tercer nivel, y así sucesivamente.

El siguiente ejemplo es un programa de graduación con dos secciones, una lista de personas que se gradúan, y otra para la descripción de la ceremonia. El marcado en este ejemplo dispone de un estilo poco común, a veces se utiliza para reducir al mínimo la cantidad de espacio en blanco entre elementos. El código HTML5 siguiente radicaliza su formato y se libera de la compatibilidad XHTML.

Además, en este ejemplo puede observarse una forma más simple para declarar el juego de etiquetas de HTML5.

```
<!DOCTYPE Html>
<Html
 ><Head
   ><Title
     >Ceremonia de graduación 2013</Title
   ></Head
 ><Body
   ><H1
     >Graduación</H1
   ><Section
     ><H1
       >Ceremonia</H1
     ><P
       >Apertura</P
     ><P
       >Discurso inaugural</P
     ><P
       >Discurso del catedrático emérito</P
     ><P
       >Presentación del diplomado</P
     ><P
       >Cierre de ceremonia</P
   ></Section
   ><Section
     ><H1
```

```
        >Graduados</H1
      ><Ul
        ><Li
          >Carlos Pita</Li
        ><Li
          >Juan Domínguez</Li
        ><Li
          >Francisco Mentiras</Li
        ><Li
          >Tiburcio Jodínez</Li
        ><Li
          >Ruth Muchas Palabras</Li
        ><Li
          >Peña Nieto García</Li
      ></Ul
    ></Section
  ></Body
></Html>
```

En este ejemplo, dado un libro, el autor ha marcado algunas secciones como capítulos y algunas en forma de apéndices, y utiliza CSS para el estilo de los encabezados en estas dos clases de sección diferente. Todo el libro está envuelto en un elemento **article** como parte de un documento.

```
<!DOCTYPE Html>
<Html
 ><Head
 <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">
  <meta name="keyword" content="HTML5, CSS3, Javascript">

  <link rel="stylesheet" href="misestilos.css">

   <article class="book">
 <style>
  section { border: double medium; margin: 2em; }
  section.chapter h1 { font: 2em Roboto, Helvetica Neue, sans-serif; }
  section.appendix h1 { font: small-caps 2em Roboto, Helvetica Neue, sans-serif;
}
 </style>

 <body>
 <header>
  <hgroup>
   <h1>Español: eBook </h1>
   <h2>Contenido</h2>
  </hgroup>
  <p><small>Publicado por CONALEP Michoacán.</small></p>
 </header>
 <section class="chapter">
  <h1>Capítulo I: lenguaje</h1>
  <p>Lenguaje, genética y cultura.</p>
  <p>Paradigmas del texto</p>
 </section>
 <section class="chapter">
```

```
  <h1>Capítulo II: el habla</h1>
  <p>Producir significado.</p>
 </section>
 <section class="chapter">
  <h1>Capítulo III: el discurso</h1>
  <p>Intensiones de comunicación</p>
  <p>Tipos de discursos.</p>
 </section>
 <section class="appendix">
  <h1>Apendice A: Ejemplos</h1>
  <p>Reportes de lectura.</p>
 </section>
 <section class="appendix">
  <h1>Appendice B: razones y argumentos</h1>
  <p>Proposiciones<em>can</em> premisas, conlsusiones y partículas
discurcivas</p>
 </section>
</article>

   </body>

</Html>
```

aside

El elemento **aside** (a un lado), representa una sección de una página que consiste de contenido, que es tangencialmente relacionado con el contenido alrededor del elemento, y que podría ser considerado por separado a partir de ese contenido. Estas secciones se representan a menudo como barras laterales en la tipografía impresa.

El elemento puede ser utilizado para efectos tipográficos como citas de tracción o barras laterales, para la publicidad, por grupos de elementos de navegación, y por otro contenido que se considera separado del contenido principal de la página.

```
<!DOCTYPE Html>
<Html
 ><Head
 <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">
  <meta name="keyword" content="HTML5, CSS3, Javascript">

  <link rel="stylesheet" href="misestilos.css">

   <article class="book">


<body>


<aside>
    <div class="block">
```

```
            <h3>Título</h3>
            <ul>
              <li>Opciones</li>
            </ul>
    </div>
</aside>

  </body>

</Html>
```

footer

El elemento **footer** representa un pie de página. El pie de página suele contener información acerca de su sección, como quién lo escribió, enlaces a documentos relacionados, los datos de autor, y similares. Cuando el elemento **footer** contiene secciones enteras, constituyen apéndices, índices, colofones largos, contratos de licencia detallados, y otro tipo de contenido.

Información del autor o editor, una sección de contacto, de dirección; podría ser adecuado tanto para un encabezado o un pie de página. El propósito principal de estos elementos es simplemente ayudar al autor a escribir marcado explicito de fácil acceso para mantener y modificar estilo; no está destinado a imponer estructuras específicas en autores.

Aquí hay una página con dos **footer**, uno en la parte superior y otro en la parte inferior, con el mismo contenido (es mejor usar header en la parte superior):

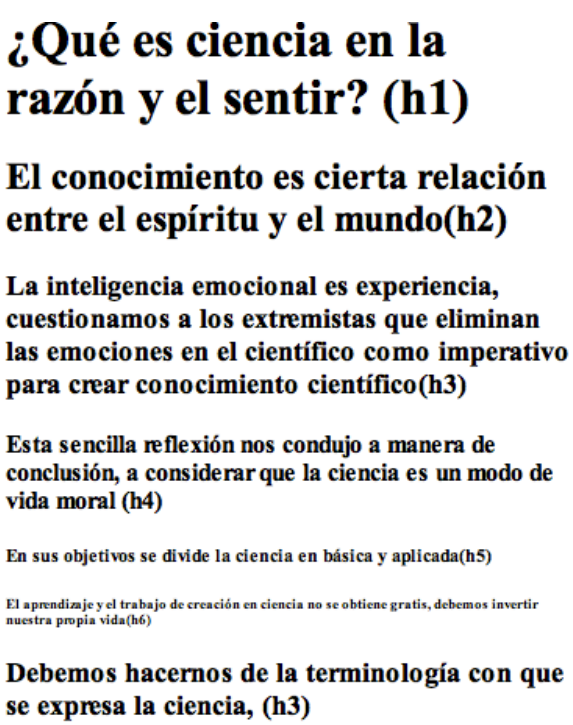
```
<!DOCTYPE Html>
<Html
 ><Head
 <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">
  <meta name="keyword" content="HTML5, CSS3, Javascript">

  <link rel="stylesheet" href="misestilos.css">

   <article class="book">


<body>

<footer><a href="../">volver al índice...</a></footer>
 <hgroup>
  <h1>Exordio</h1>
  <h2>verda y vida</h2>
 </hgroup>
```

```
 <p>Aunque a veces abran la idea muda de la matemática para intentar hacer ver
que esta es el silencio más útil a la ciencia;  razones puras donde no hay tiempo
ni vejez para el hombre. Su notación muda, se hace el oficio del científico y el
ingeniero para transformar la realidad material del mundo. La poesía sin prisa,
sin pausa,  conquista a la derecha del cero la esperanza y a la izquierda del
cero la voz de la denuncia y reposa en el cero la indeterminación del espíritu
humano. La poesía no tiene una sola realidad, sino que afirma que el monopolio de
la esperanza a la luz de la ciencia enferma a la conciencia  que da dignidad al
hombre. Después de muchos logros espectaculares de la ciencia y la tecnología, el
hombre, polvo de estrellas  heredero de fuego, yace frente a sí en calles de
niebla y corrupción, el invierno funerario de los valores augura la pobreza más
cruda. Las dificultades para decir la verdad no se comparan con las dificultades
para vivir la vida.</p>
 <footer><a href="../">volver al índice...</a></footer>

 <footer> <!-- site wide footer -->
 <nav>
  <p><a href="/credits.html">Creditos</a> —
     <a href="/tos.html">Término del servicio</a> —
     <a href="/index.html">Índice</a></p>
 </nav>
 <p>Copyright © 2013 CONALEP</p>
</footer>

  </body>

</html>
```

h1-h6 hgroup

Los elementos **h1-h6** y el elementos **hgroup** son aplicados para partidas de texto.

De manera anidada a los elemento **article, header y hgroup** se colocan los elementos **h**, **h1-h6**, para declarar un título; **h#** se emplea para la cabecera de cada parte del contenido del documento, de **h1,h2,h3,h4,h5 y h6** modifican el tamaño del texto para indicar el orden jerárquico de títulos y subtítulos dentro de un documento, por ejemplo:




```
<!DOCTYPE Html>
<html
 ><head
 <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">
  <meta name="keyword" content="HTML5, CSS3, Javascript">

  <link rel="stylesheet" href="misestilos.css">

   <article class="book">

<body>

<h1>¿Qué es ciencia en la razón y el sentir? (h1)</h1>
<h2>El conocimiento es cierta relación entre el espíritu y el mundo(h2)</h2>
<h3>La inteligencia emocional  es  experiencia, cuestionamos a los extremistas
que eliminan las emociones en el científico como imperativo para crear
conocimiento científico(h3)</h3>
<h4>Esta sencilla reflexión nos condujo a manera de conclusión, a considerar que
la ciencia es un modo de vida moral (h4)</h4>
<h5> En sus objetivos se divide la ciencia en básica y aplicada (h5)</h5>
<h6>El aprendizaje y el trabajo de creación en ciencia no se obtiene gratis,
debemos invertir nuestra propia vida(h6)</h6>
<h3>Debemos hacernos de la terminología con que se expresa la ciencia, (h3)</h3>
</body>

</html>
```

Con el fin de aprovechar las etiquetas **h** en varias partes del contenido de la página se agrupan **article, header y hgroup** :

```
<!DOCTYPE Html>
<html
 ><head
 <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">
  <meta name="keyword" content="HTML5, CSS3, Javascript">

  <link rel="stylesheet" href="misestilos.css">

   <article class="book">


<body>

 <article>
     <header>
          <hgroup>
<h1>¿Qué es ciencia en la razón y el sentir? (h1)</h1>
<h2>El conocimiento es cierta relación entre el espíritu y el mundo(h2)</h2>
<h3>La inteligencia emocional  (h3)</h3>
<h4>Considerar que la ciencia es un modo de vida moral (h4)</h4>
<h5>En sus objetivos se divide la ciencia en básica y aplicada(h5)</h5>
<h6>El aprendizaje y el trabajo de creación en ciencia no se obtiene gratis,
debemos invertir nuestra propia vida(h6)</h6>
<h3>Debemos hacernos de la terminología con que se expresa la ciencia, (h3)</h3>
          </hgroup>
     </header>
</article>

          <article>
     <header>
          <hgroup>
<h1>Los sistemas conceptuales de la ciencia son instrumentos en dos dimensiones
lingüísticas: sintaxis y semántica. (h1)</h1>
<h2>La lectura en ciencia se da en tres niveles, lingüístico -términos y frases-,
nivel conceptual -conceptos y proposiciones- y nivel óntico -hechos, cosas
propiedades-. (h2)</h2>
<h3>El concepto como función metodológica divide, ordena y sistematiza el trabajo
de generación del conocimiento científico  (h3)</h3>
<h4>La terminología en función del enriquecimiento natural de la extensión e
intensión conceptual, es un imperativo para la reducción de la ambigüedad
conceptual dentro de los textos científicos a nivel de lectura y escritura
(h4)</h4>
<h5>Encontrar problemas originales, el sentido de la ciencia (h5)</h5>
<h6>El trasfondo científico, implica una moda intelectual, o tomar más riesgos
para plantear un desafío de paradigma.(h6)</h6>
<h3>La ciencia en su quehacer sustantivo justifica la generación de
problemas(h3)</h3>
          </hgroup>
     </header>
</article>

</body>

</html>
```

El elemento **hgroup** nos ayuda cuando tenemos títulos y subtítulos con etiquetas **h** y deseamos volver a emplear su jerarquía en secciones diferentes del documento.

article

El elemento **article** representa una composición autónoma en un documento, página, aplicación o sitio, y que es, en principio, independiente, distribuible o reutilizable, por ejemplo, en la **sindicación Web**. Esto podría ser un mensaje en el foro, un artículo de una revista o de un periódico, un blog, un comentario enviado por el usuario, de un widget interactivo o gadget, o cualquier otro elemento independiente del contenido.

Cuando se anidan los elementos **article**, los elementos **article** interiores representan artículos que son en principio parte del artículo exterior. Por ejemplo, una entrada de blog en un sitio que acepta que el usuario presente observaciones en una caja de texto, esto podría representar los comentarios del artículo, que serán anidados dentro del elemento **article** para la entrada de blog. Cuando se utiliza específicamente con el contenido en sindicación Web, este elemento de trabajo es similar en efecto al elemento de entrada en Atom.

El vocabulario microdatos schema.org se puede utilizar para proporcionar la fecha de publicación de un elemento del artículo, utilizando uno de los subtipos CreativeWork.

Este ejemplo muestra una entrada de blog usando el elemento artículo, con algunas anotaciones schema.org, la misma entrada del blog y algunos de los comentarios:

```
<!DOCTYPE html>
<html
 ><head
 <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">
  <meta name="keyword" content="HTML5, CSS3, Javascript">

  <link rel="stylesheet" href="misestilos.css">

   <article class="book">


<body>
```

```
<article itemscope itemtype="http://schema.org/BlogPosting">
 <header>
  <h1 itemprop="headline">Escribir el ensayo</h1>
  <p><time itemprop="datePublished" datetime="2009-10-09">Hace 3 días </time></p>
  <link itemprop="url" href="?comments=0">
 </header>
 <p>Apoyándonos en  Octavio Paz, diremos que las verdaderas ideas o ensayos, no
son las que se le ocurren al ensayista –estudiante o profesor- antes de escribir
el ensayo, sino las que después, con su voluntad, se desprenden naturalmente de
la obra</p>
 <p>...</p>
 <section>
  <h1>Comentarios</h1>
  <article itemprop="comment" itemscope itemtype="http://schema.org/UserComments"
id="c1">
   <link itemprop="url" href="#c1">
   <footer>
    <p>Comenta: <span itemprop="creator" itemscope
itemtype="http://schema.org/Person">
     <span itemprop="name">Miguel Hidalgo</span>
    </span></p>
    <p><time itemprop="commentTime" datetime="2009-10-10">Hace 15
minutos</time></p>
   </footer>
   <p>El fondo brota de la forma y no a la inversa!</p>
  </article>
  <article itemprop="comment" itemscope itemtype="http://schema.org/UserComments"
id="c2">
   <link itemprop="url" href="#c2">
   <footer>
    <p>Comenta: <span itemprop="creator" itemscope
itemtype="http://schema.org/Person">
     <span itemprop="name">George Oswell</span>
    </span></p>
    <p><time itemprop="commentTime" datetime="2009-10-10">Hace 3
minutos</time></p>
   </footer>
   <p>La armazón de argumentos en su forma es el fondo y cada forma crea su
idea.</p>
  </article>
 </section>
</article>


</body>

</html>
```

time

El elemento **time** permite declarar un texto que es interpretado por máquinas y humanos, con semántica de hora y fecha. El elemento **time** compone un tiempo específico sin información de zona horaria, que consiste en una hora, un minuto, un segundo, y una fracción de un segundo.

Dos dígitos ASCII, representan horas, en el rango $0 \leq 23$ horas

Un caracter dos puntos (:) para separar horas: minutos: segundos

Dos dígitos ASCII, representan minutos, en el rango $0 \leq 59$ minutos

figure

El elemento **figure** hace referencia como una sola unidad de la corriente principal del documento. El elemento se puede utilizar para anotar ilustraciones, diagramas, fotos, listas de códigos, etc, que se conocen a partir del contenido principal del documento, sin afectar el flujo del documento, que se alejó de ese contenido principal, por ejemplo, a un lado de la página, a páginas dedicadas, o a un apéndice.

El elemento secundario **figcaption**, en su caso representa el título de los contenidos del elemento figura. Si no hay ningún elemento **figcaption** hijo, entonces no hay subtítulo.

En este ejemplo se muestra el elemento de figura para marcar una lista de código.

```
<!DOCTYPE Html>
<Html
 ><Head
 <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">
  <meta name="keyword" content="HTML5, CSS3, Javascript">


   <title>El primer script</title>

<script type="text/javascript">
  alert("Hola Mundo!");
</script>


<body>
<p>Esta página contiene el primer script</p>
<p>El <a href="#14">ejemplo 1</a> un primer script.</p>
<figure id="14">
 <figcaption>Código 1. un bloque de código dentro de una página
XHTML</figcaption>
 <pre><code>interface PrimaryCore {
 boolean verifyDataLine();
 void sendData(in sequence&lt;byte> data);
 void initSelfDestruct();
}</code></pre>
```

```
</figure>
<p>Emplea a UTF-8.</p>


</body>
```

mark

El elemento **mark** representa una extensión del texto en un documento, es para fines de referencia debido a que resalta su relevancia en el contexto. Cuando se utiliza en una cita u otro bloque de texto que se refiere a la prosa, indica un punto culminante que no estaba presente originalmente, sino que se ha añadido para atraer la atención del lector a una parte del texto, que podría no haber sido considerada importante por el autor original cuando el bloque de prosa ha sido escrito, pero que es ahora objeto de escrutinio inesperado. Cuando se utiliza en la prosa principal de un documento, indica una parte del documento que se ha destacado por su posible importancia para la actividad actual del usuario.

Este ejemplo muestra cómo puede ser usado el elemento **mark** para llamar la atención sobre una parte específica de una cita:

```
<!DOCTYPE Html>
<Html
 ><Head
 <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">
  <meta name="keyword" content="HTML5, CSS3, Javascript">


<body>

<p lang="es"> Los que producen una idea escrita, se les considera seres con
talento o simplemente genios:</p>
<blockquote lang="es">
 <p>Las personas tienen la capacidad de modificar el significado de los mensajes
que reciben en función de su cultura
 <mark>añaden códigos de analogía apoyados en sus referentes de realidad;</mark>
escribir ideas, es un laboratorio de procesos de representación con una
complejidad importante para producir sentido en el lector.</p>
</blockquote>
<p lang="en-US">Geniuses.  <em>Adjective that gives reference to achievements
that are original first and second, are exceptional.</em> Individuals can
according to their research see with their own eyes, speak with your own voice
and feel with their own hearts.</p>


</body>
```

lang

El atributo **lang** (en ningún espacio de nombres) especifica el idioma principal de los contenidos del elemento texto. Su valor debe ser una etiqueta BCP 47 de idioma válido[32], o la cadena vacía. Al establecer el atributo en la cadena vacía indica que el idioma principal es desconocido. En la siguiente tabla se expresan más atributos presentes en HTML5.

Atributo	Descripción	Ejemplo de uso
id	El atributo "id" asigna un identificador al elemento asociado. Este identificador debe ser único en el documento y puede ser usado para referirse a ese elemento.	Este es el primer párrafo nombrado parrafo1. Para cambiar dinámicamente las propiedades del mismo usa este identificador.
class	El atributo "class" asigna un nombre de clase (o una lista de nombres de clase separados por espacios) al elemento contenedor. Es usado con hojas de estilos e indica al navegador la clase a la cual el elemento está asociado. Una clase provee atributos visuales para los elementos.	Este artículo está basado en el libro "Viento en los árboles" por Jhon L. Brooks Este artículo está basado en el libro "Viento en los árboles" por Jhon L. Brooks... y es más importante que el anterior.
style	Define un estilo visual para el elemento. Es una mejor práctica definir atributos en hojas de estilos externas agrupándolos en clases. Los atributos en el parámetro "style" deben preservar este orden "nombre : valor" y ser separados por un punto y coma. Si estás escribiendo código XHTML te recomendamos no utilizar este atributo y probar con las clases de hojas de estilo (con el atributo "class").	Este es un párrafo con un estilo definido Y este es otro texto sin estilo.
title	Indica un título para el elemento. Usado para dar una descripción acerca del elemento que es usualmente mostrado como un "tool tip" cuando el usuario pone el puntero del mouse sobre el elemento.	Código HTML
lang	Especifica el lenguaje del contenido de un elemento. El valor predeterminado es "desconocido". Al escribir código XHTML la sintaxis "xml:lang" representa una alternativa preferida en XHTML 1.0 y un reemplazo en XHTML 1.1 (por ejemplo, xml:lang="en").	This is a paragraph in english. Este es un párrafo en español.
dir	Especifica la dirección de texto del contenido y atributos del elemento, así como la direccionalidad de las tablas. Tiene dos valores posibles que son insensibles a	<q lang="he" dir="rtl">...Una cita en Hebreo...</q>

	mayúsculas/minúsculas: RTL: Derecha a izquierda, LTR: Izquierda a derecha.	

span, p

El elemento **span** no significa nada por sí mismo, pero puede ser útil cuando se utiliza junto con los atributos globales, por ejemplo, **class**, **lang**, o **dir**. En este ejemplo, un fragmento de código se caracteriza por el uso de elementos **span** y atributos **class** de modo que sus palabras clave e identificadores pueden ser un código de colores empleando CSS.

Cuando el texto es corto (o por muy largo una línea de texto) se emplea **span**, cuando se trata de un párrafo se emplea **<p>.**

```
<!DOCTYPE html>
<html lang="en-US">
 <head>
  <title>Mi libro favorito</title>
  <style>
   body { color: black; background: white; }
   em { font-style: normal; color: red; }
   blu { font-style: normal; color: blue; }


  </style>


 </head>
 <body>


  <pre><code class="lang-c"><span class="keyword"><em>for</em></span></span>
(<span class="em">j</span> = 0; <span class="em">j</span> &lt; 256; <span
class="ident">j</span>++) <blu>{</blu>
  <span class="ident">i_t3</span> = (<span class="ident">i_t3</span> & 0x1ffff) |
(<span class="ident">j</span> &lt;&lt; 17);
  <span class="ident">i_t6</span> = (((((((<span class="redwine">i_t3</span> >>
3) ^ <span class="ident">i_t3</span>) >> 1) ^ <span class="redwine">i_t3</span>)
>> 8) ^ <span class="ident">i_t3</span>) >> 5) & 0xff;
  <span class="keyword">if</span> (<span class="ident">i_t6</span> == <span
class="ident">i_t1</span>)
    <span class="keyword"><em>break</em></span>;
<blu>}</blu></code></pre>

 </body>
</html>
```

small

El elemento **small** representa comentarios diversos en letra pequeña. La letra pequeña ofrece típicamente renuncias, advertencias, restricciones legales o derechos de autor. La letra pequeña también se utiliza a veces para la atribución, o para satisfacer los requisitos de licencia.

No debe utilizarse para tramos largos de texto, tales como párrafos, listas, o secciones de texto.

```
<!DOCTYPE html>
<html lang="en-US">
 <head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">
  <meta name="keyword" content="HTML5, CSS3, Javascript">

 </head>
 <body>


  <dl>
 <dt>El papel relevante que tiene la escritura científica como medio de
aprendizaje focaliza esta propuesta, para que Usted en un sentido autodidacta
comprometido con el valor epistémico de honradez con la originalidad, pueda
formular preguntas de investigación profundas en el mismo acto de
redacción<dd>Nota: <small>Tome en cuenta que redactar un texto es pensar en el
nivel superior del intelecto humano</small>

</dl>

 </body>
</html>
```

dl, dt, dd

El elemento **dl** representa una lista de asociación, consta de cero o más grupos de nombre-valor (una lista de descripción). Cada grupo debe estar formado de uno o más nombres de elementos (**dt**), seguido de uno o más valores de los elementos (**dd**). Dentro de un solo elemento **dl**, no debe haber más de un elemento **dt** para cada nombre.

Grupos de nombres valor pueden ser términos y definiciones, temas y valores de metadatos, preguntas y respuestas, o cualquier otro grupo de datos de nombre-valor.

El orden de la lista de grupos de los nombres y los valores dentro de cada grupo, pueden ser significativos.

En el siguiente ejemplo, una entrada **dl** está vinculada a valores **dt** y **dd**:

```
<!DOCTYPE html>
<html lang="en-US">
 <head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">
  <meta name="keyword" content="HTML5, CSS3, Javascript">

 </head>
 <body>

<dl>
 <dt> Autores
 <dd> Francisco Mentiras
 <dd> Huerta Sínico
 <dt> Editor
 <dd> Juan Pérez
</dl>

<dl>
 <dt lang="es"> <dfn>poesía</dfn> </dt>
 <dt lang="es"> <dfn>poema</dfn> </dt>
 <dd> La experiencia de vivir la creación poetica da como producto al poema
</dd>
</dl>

</dl>

 </body>
</html>
```

cite

El elemento **cite** representa el título de la obra (por ejemplo, un libro, un artículo, un ensayo, un poema, una partitura, una canción, un guión, una película, un programa de televisión, un juego, una escultura, una pintura, un teatro producción, una obra de teatro, una ópera, un musical, una exposición, un caso legal, etc.) Esto puede ser un trabajo que se cita o se hace referencia en detalle (es decir, una cita), o simplemente puede ser un trabajo que se menciona de pasada.

El siguiente ejemplo muestra un uso típico del elemento **cite**:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">
  <meta name="keyword" content="HTML5, CSS3, Javascript">


 </head>
 <body>

<p>Mi libro favorito <cite>El sentido de un final</cite> de
Julian Barnes.</p>
<p> </p>

</dl>

 </body>
</html>
```

ISO-8859-1

ISO-8859-1 es el juego de caracteres predeterminado en la mayoría de los navegadores. Los primeros 128 caracteres de la norma ISO-8859-1 es el juego de caracteres ASCII originales (los números 0-9, el alfabeto Inglés en mayúsculas y minúsculas y algunos caracteres especiales). La parte superior de la norma ISO-8859-1 (códigos 160-255), contiene los caracteres utilizados en los países de Europa Occidental y algunos caracteres especiales utilizados. Se utilizan para implementar caracteres reservados o para expresar caracteres que no se pueden introducir fácilmente con el teclado.

Caracteres reservados para HTML

Algunos caracteres son reservados en HTML y XHTML. Por ejemplo, no se puede utilizar el mayor o menor que, estos signos en el texto se podrían confundir con marcas al momento de ser interpretados por los navegadores.

Por tal motivo, los HTML y XHTML deben ser compatibles con los cinco caracteres especiales que se enumeran en la tabla siguiente:

Carácter	Número de entidad	Nombre de entidad	Descripción
"	&#34;	&quot;	Comillas
'	&#39;	&apos;	Apóstrofe
&	&#38;	&amp;	Signo &
<	&#60;	&lt;	Menor que
>	&#62;	&gt;	Mayor que

Note: Los nombre de entidad son el caso sensitivo a HTML

ISO 8859-1 Símbolos

Carácter	Número de entidad	Nombre de entidad	Descripción
			Espacio de no separación
¡	&#161;	&iexcl;	Signo de exclamación invertido
¢	&#162;	&cent;	Centavo
£	&#163;	&pound;	Libra
¤	&#164;	&curren;	Moneda
¥	&#165;	&yen;	yen
¦	&#166;	&brvbar;	Barra vertical rota
§	&#167;	&sect;	Sección
¨	&#168;	&uml;	Diéresis separación
©	&#169;	&copy;	Derechos de autor
ª	&#170;	&ordf;	Indicador ordinal femenino
«	&#171;	&laquo;	Comillas angular (izquierda)
¬	&#172;	&not;	Negación
--	&#173;	&shy;	Guión suave
®	&#174;	&reg;	Marca comercial registrada
¯	&#175;	&macr;	Macro separación
°	&#176;	&deg;	Grado
±	&#177;	&plusmn;	Más o menos
²	&#178;	&sup2;	Superíndice 2
³	&#179;	&sup3;	Superíndice 3
´	&#180;	&acute;	Separación aguda
µ	&#181;	&micro;	Micro
¶	&#182;	&para;	Párrafo
·	&#183;	&middot;	Punto medio
¸	&#184;	&cedil;	Cedilla separación
¹	&#185;	&sup1;	Superíndice 1
º	&#186;	&ordm;	Indicador ordinal masculino
»	&#187;	&raquo;	Comilla angular (derecha)
¼	&#188;	&frac14;	Fracción 1/4
½	&#189;	&frac12;	Fracción 1/2
¾	&#190;	&frac34;	Fracción 3/4
¿	&#191;	&iquest;	Signo de interrogación invertido
×	&#215;	&times;	Multiplicación

÷	÷	÷	División

ISO 8859-1 Caracteres

Carácter	Número de entidad	Nombre de entidad
À	À	À
Á	Á	Á
Â	Â	Â
Ã	Ã	Ã
Ä	Ä	Ä
Å	Å	Å
Æ	Æ	Æ
Ç	Ç	Ç
È	È	È
É	É	É
Ê	Ê	Ê
Ë	Ë	Ë
Ì	Ì	Ì
Í	Í	Í
Î	Î	Î
Ï	Ï	Ï
Ð	Ð	Ð
Ñ	Ñ	Ñ
Ò	Ò	Ò
Ó	Ó	Ó
Ô	Ô	Ô
Õ	Õ	Õ
Ö	Ö	Ö
Ø	Ø	Ø
Ù	Ù	Ù
Ú	Ú	Ú
Û	Û	Û
Ü	Ü	Ü
Ý	Ý	Ý
Þ	Þ	Þ
ß	ß	ß
à	à	à
á	á	á
â	â	â
ã	ã	ã
ä	ä	ä
å	å	å
æ	æ	æ
ç	ç	ç
è	è	è
é	é	é
ê	ê	ê
ë	ë	ë
ì	ì	ì

í	í	í
î	î	î
ï	ï	ï
ð	ð	ð
ñ	ñ	ñ
ò	ò	ò
ó	ó	ó
ô	ô	ô
õ	õ	õ
ö	ö	ö
ø	ø	ø
ù	ù	ù
ú	ú	ú
û	û	û
ü	ü	ü
ý	ý	ý
þ	þ	þ
ÿ	ÿ	ÿ

address

El elemento de dirección **address** representa la información de contacto de su artículo ancestro más cercano o elemento **body**. Si ese es el elemento **body**, la información de contacto se aplica al documento en su conjunto.

Por ejemplo, una página en el sitio Web del W3C relacionada con HTML podría incluir la siguiente información de contacto:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">
  <meta name="keyword" content="HTML5, CSS3, Javascript">


 </head>
 <body>



<address>
 <A href="http://www.cie.umich.mx/conalepWeb2013/">CONALEP</A>,
 <A href="http://www.cie.umich.mx/conalepWeb2013/">Michoacán</A>,
 contacto institucional <A href="AnclaD">Colegio de Educación Profesional
Técnica del Estado de Michoacán,
```

```
Gral. Nicolás Bravo # 144 Col. Chapultepec Norte  Morelia, Michoacán, México.
C.P. 58260
Teléfono:   (443) 113-6100 al 113-6102
FAX:    +443 324-6018
Email: direcmich@prodigy.net.mx</A>
</address>


</dl>

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

a, href

Si el elemento <**a>** tiene un atributo **href**, entonces representa un hipervínculo (un ancla de hipertexto) marcado por su contenido. Podemos crear un enlace dentro del mismo documento:

a) Se inserta la marca <**a name**="conalep"> en el punto del documento al que quieres que se llegue (conalep es un identificador del punto en el que el navegador deberá visualizar la página. Obviamente puedes sustituirlo con otros términos). Esto se llama crear anclas.
b) En el enlace desde el que se quiere llegar al punto del nuevo documento, se inserta la siguiente sintaxis, donde # indica que se trata de un enlace interno al documento:
Portal Web CONALEP
c) Si quisiéramos llegar a un punto concreto de un documento externo, la sintaxis correcta sería:
Portal Web CONALEP
donde "nombre_archivo.htm" es el nombre del documento al que hay que llegar, y "conalep" es el punto preciso de dicho documento.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">
  <meta name="keyword" content="HTML5, CSS3, Javascript">

 </head>
 <body>
```

```
<A HREF="http://www.cie.umich.mx/conalepWeb2013/">Visita Web CONALEP</A>

</dl>

 </body>
</html>
```

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8"
  <meta name="description" content="ejemplo adjuntar CSS"
  <meta name="keyword" content="HTML5, CSS3, Javascript"

 </head>
 <body>

<aside class="advertising">
 <h1>WEB</h1>
 <a href="http://www.cie.umich.mx/conalepWeb2013/">
  <section>
   <h1>CONALEP</h1>
   <p>El Colegio Nacional de Educación Profesional Técnica (CONALEP)</p>

  </section>
 </a>
 <p><a href="http://www.cie.umich.mx/conalepWeb2013/"> </a></p>
 <p><a id="ancla1"></a>México  (ancla)</p>

</body>
</html>
```

hr

El elemento **hr** se puede utilizar para crear un salto de un documento, en un punto donde puede haber un cambio de pensamiento o significado, pero en el que no necesariamente puede ser conveniente introducir una subpartida.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>
<p> Las palabras cruzando el puente.</p>
<hr/>
<div id="footer">&copy; Todos los contenidos  copyright 2013. incluso comentarios
html.</div>
 </body>
```

```
</html>
```

br

El propósito del elemento **br** es muy simple: se crea un salto de línea dentro de un bloque de texto, dejando sin relleno a los márgenes entre los dos bloques de texto creados por el salto de línea.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>


<p>Nos prometimos silencio,<br/>
    hablamos sin palabras,<br/>
    por si el solitario disfraz de la madrugada,<br/>
    nos ponía la tediosa tarea de amarnos,<br/>
    en aquella calle vacía.</p>

 </body>
</html>
```

```
@charset "UTF-8";
/* CSS CSS archivo misestilos.css */

body {
      text-align:
      justify
}
```

blockquote

El elemento **blockquote** es un mecanismo para el marcado de un bloque de texto de la cita de una persona o de otro documento o fuente. Pueden ser solo unas pocas líneas, o puede contener varios párrafos.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">
```

```
<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>


<blockquote>
  <p>Las personas tienen la capacidad de modificar el significado de los mensajes
que reciben en función de su cultura, añaden códigos de analogía apoyados en sus
referentes de realidad; escribir ideas, es un laboratorio de procesos de
representación con una complejidad importante para producir sentido en el lector.
La mente del lector actúa en el flujo de información de un texto, de manera
activa, generando inferencias que se derivan de su experiencia previa.</p>
</blockquote>

 </body>
</html>
```

```
@charset "UTF-8";
/* CSS CSS archivo misestilos.css */

body {
     text-align:
     justify
}
```

li

El elemento **li** define un elemento de la lista individual y solo puede aparecer dentro de un puñado de elementos relacionados con la lista antes detallada. Cada elemento de la lista está representado por una bala como viñeta (para las listas no ordenadas, definido por el elemento **ul**) o un número o letra (en el caso de las listas ordenadas, definido por el elemento **ol**).

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>


<ul>
  <li>Ensayo</li>
  <li>Tesis</li>
  <li>Artículo</li>
  <li>Poster</li>
  <li>Apatente</li>
  <li>Revisión</li>
</ul>
```

```
<ol>
  <li>Ensayo</li>
  <li>Tesis</li>
  <li>Artículo</li>
  <li>Poster</li>
  <li>Apatente</li>
  <li>Revisión</li>
</ol>

 </body>
</html>
```

menu

El elemento **menu** fue pensado originalmente para ser utilizado para mostrar una lista de opciones de menú, y es casi idéntico en su propósito al elemento **dir** (excepto que, a diferencia de **dir**, el elemento **menu** está destinado a varias columnas de visualización).

```
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">

<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>

<menu>
  <li>Ensayo</li>
  <li>Tesis</li>
  <li>Artículo</li>
  <li>Poster</li>
  <li>Apatente</li>
  <li>Revisión</li>
</menu>

<menu label="Edit">
<button type="button" onclick="edit_cut()">Cortar</button>
<button type="button" onclick="edit_copy()">Copiar</button>
<button type="button" onclick="edit_paste()">Pegar</button>
</menu>

 </body>
</html>
```

em

El elemento **em** se utiliza para enfatizar el contenido del texto, y se muestra en cursiva en todos los navegadores actuales. Proporciona significado semántico sobre el texto que contiene, efectivamente diciendo, "este texto es un poco más importante que el otro texto

aquí". Esto diferencia **em** del elemento **i**, que se limita a establecer la fuente a cursiva. En la mayoría de los casos, **em** es el elemento preferido de usar.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>


<p>Hay cenizas <em>desiertas</em>, mares pronunciano tu nombre,  <em>Luz</em>,
futuro carente de recuerdos de soledad.</p>


 </body>
</html>
```

var

El elemento **var** se utiliza para indicar que el texto es una variable y no debe ser tomada literalmente. Es un marcador de posición donde los contenidos se deben reemplazar con su propio valor.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>


<p>Donde x es la varable
<kbd>f(x) <var>x</var>.  Ec. (1)</kbd>.</p>

 </body>
</html>
```

code

El propósito del elemento **code** es la identificación de código de computadora, por ejemplo, un fragmento de código HTML o XML, o algún otro código legible por la máquina, que sea un lenguaje de servidor como Javascript del lado del cliente. La mayoría de los navegadores representan el contenido de código en una fuente de ancho fijo, como Courier, pero este estilo se puede sustituir empleando CSS.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>


<p>La siguiente trama es código de computadora <code>Javascript</code></p>
<p>  <code>
  // Write to a heading:
  document.getElementById("myH1").innerHTML="Welcome to my Homepage";
  // Write to a paragraph:
  document.getElementById("myP").innerHTML="This is my first paragraph.";
  </code>.</p>

 </body>
</html>
```

pre

El elemento **pre** se utiliza para conservar espacios en blanco y retornos de carro importantes en el marcado de origen. De forma predeterminada, solo el primer espacio es respetado; espacios posteriores, a menos que se especifique el uso de esta etiqueta no son respetados por los navegadores. El elemento **pre** se usa más frecuentemente en combinación con el elemento **code,** para proteger los ejemplos de código en los que la presencia de espacios y/o retornos de carro puede hacer una diferencia importante en cuanto a si el código va a funcionar o no, cuando se copia y se pega en otro lugar.

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

```
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>

<pre><code>
for (i=1;i&lt;array.length;i++)
 {
 str += "&lt;li&gt;Value of array " + i + " is: " +array[i] + "&lt;/li&gt;";
 }
document.write(str);
document.write ("&lt;/ul&gt;");
</code></pre>

 </body>
</html>
```

strong

El elemento **strong** se utiliza para enfatizar una frase de contenido de texto. Por lo general hace que el texto figure en negrita.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>

<p>Palabras cruzando el puente,
    <strong>el retorno de pensar escribiendo<strong>, escribir es resistir...</p>

 </body>
</html>
```

q

El elemento **q** es el hermano del elemento **blockquote**. Cuando **blockquote** crea un bloque distinto del texto citado, el elemento **q** se utiliza para las citas en línea. Se pretende que el navegador debe insertar las comillas necesarias, al estilo de lo que debería depender de la

lengua del documento o de la sección del documento, en lugar de que el autor añada comillas.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>

<p>Hay caminos <q>poblándose de palabras</q>, tiempos de inmortales recuerdos
apareció la luna.</p>

 </body>
</html>
```

s, strike

La **s** es idéntica en el propósito que el elemento **strike**, el cual marca el texto referido.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>


<p><strike>Hay caminos.</strike> poblándose de palabras, tiempos de inmortales
recuerdos apareció la luna.</p>

<p><s>Hay caminos.</s> poblándose de palabras, tiempos de inmortales recuerdos
apareció la luna.</p>

 </body>
</html>
```

dfn

El elemento **dfn** se utiliza para identificar la instancia de definición de un término (más probablemente una palabra específica de la industria, o la jerga de algún tipo de terminología especializada). Algunos, pero no todos, los navegadores hacen que el contenido del elemento **dfn sea** en cursiva.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>


<p>El concepto de <dfn>apoptosis</dfn>, a revolucionado los esfuerzos por hacer
mortales a las células cancerígenas ⋮</p>
 </body>
</html
```

abbr

El elemento **abbr** se utiliza para proporcionar una alternativa de una frase abreviada. La intención es que en los navegadores y otras tecnologías de asistencia se interprete esta información y se presente al usuario en un formato adecuado cuando se le solicite.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>

<p>Quiero moverme como pasto que deja la lluvia, <abbr title="se nos mueve la
vida">supervivencia</abbr>
    caminar en soles de ligeras notas, de escena desperada por el sentido de la
vida</p>

 </body>
</html>
```

samp

El propósito del elemento **samp** es para identificar una muestra de caracteres que forman la salida o resultado de algún proceso.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>


<p>Quiero moverme como pasto que deja la lluvia,<samp>supervivencia</samp>
caminar en soles de ligeras notas, de escena desperada por el sentido de la vida
<samp>experiencia</samp> y <samp>emociones,</samp> formas de conocimiento</p>


 </body>


</html>
```

kbd

El elemento **kbd** se utiliza para identificar las pulsaciones de teclado que el usuario necesita para hacer algo con el teclado. Estas pulsaciones podrían formar una cadena de texto, o pueden reflejar pulsaciones de teclas individuales que el usuario debe realizar.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>

<p>Emplee las teclas <kbd>CTRL</kbd>, <kbd>ALT</kbd>, y <kbd>DELETE</kbd>,
    en las tareas de edición de su video</p>

 </body>
```

```
</html>
```

sub

El elemento **sub** se utiliza para definir el texto subíndice que aparece a media altura del texto por debajo de la línea base, y se utiliza con mayor frecuencia en las fórmulas matemáticas o químicas.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>

<p>El agua. Su fórmula química  H<sub>2</sub>O. Obsérvese
    H<sub>2</sub>...</p>

 </body>

</html>
```

sup

El elemento **sup** se utiliza para definir el texto superíndice que aparece a media altura del texto, por encima de la línea base. Se utiliza con mayor frecuencia en las fórmulas matemáticas, o para los fines de indicar referencias al pie.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>

<p>Si tuviera que decir dónde estaban mis auténticas esperanzas, diría allí donde
era subyugado por experiencias que no podía explicar<sup>1</sup> y hasta ahora
```

```
nadie, ni siquiera yo mismo he conseguido liberarme de algo explicándomelo hasta
el fin.<sup>2</sup>
    </p>

 </body>

</html>
```

i

Elemento de estilo **i** (abreviatura de cursiva, itálicas en inglés) del texto que contiene en cursiva, pero no ofrece ningún significado semántico sobre el texto contenido.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>

<p><i>Si tuviera que decir dónde estaban mis auténticas esperanzas</i>, diría
allí donde era subyugado por experiencias que no podía explicar<sup>1</sup> y
hasta ahora nadie, ni siquiera yo mismo he conseguido liberarme de algo
explicándomelo hasta el fin.<sup>2</sup>
    </p>


 </body>

</html>
```

b

El elemento **b** (abreviatura de negrita, bold en inglés) simplemente aplica estilo al texto que encierra en un tipo de letra negrita; suponiendo que no está ya presente un tipo de letra negrita, pero no ofrece ningún significado semántico sobre el texto contenido.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>
```

```
<p><b>Si tuviera que decir dónde estaban mis auténticas esperanzas</b>, diría
allí donde era subyugado por experiencias que no podía explicar<sup>1</sup> y
hasta ahora nadie, ni siquiera yo mismo he conseguido liberarme de algo
explicándomelo hasta el fin.<sup>2</sup>
    </p>

 </body>

</html>
```

u

El elemento **u** (abreviatura de subrayado, underlined en inglés), simplemente aplica al texto contenido en su interior una línea de subrayado, pero no ofrece ningún significado semántico sobre el texto contenido.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>

<p><u>Si tuviera que decir dónde estaban mis auténticas esperanzas</u>, diría
allí donde era subyugado por experiencias que no podía explicar<sup>1</sup> y
hasta ahora nadie, ni siquiera yo mismo he conseguido liberarme de algo
explicándomelo hasta el fin.<sup>2</sup>
    </p>


 </body>

</html>
```

ruby

El elemento **ruby** proporciona un mecanismo para anotar caracteres de idiomas asiáticos (japonés, chino, coreano, etc). Por lo general, estas anotaciones aparecen en una tipografía más pequeña por encima del texto normal.

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

```
   <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>

<p>
 <ruby>
  <rb>家辺 勝文</rb>
  <p>enlace</p>
 </ruby>
</p>


 </body>

</html>
```

plaintex

El propósito del elemento **plaintex** es ignorar el código HTML para mostrarlo en el navegador.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>

<plaintext>

<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>

<plaintext>


</plaintext>

</html>
</plaintext>
```

```
</html>
```

noscript

El elemento **noscript** tiene un solo objetivo: ofrecer contenidos a las que acceden a la página Web con un navegador que no es compatible con secuencias de comandos del lado del cliente.

img

El elemento **img** proporciona un medio para incrustar una imagen en el documento.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>

<p>Comer manzana hace la vida rica <img
    src="manzana.jpg" alt="fruta">,
    sabor y olor símbolo de la naturaleza.</p>
</noscript>

</html>
```

fieldset

El **fieldset** es una herramienta útil para organizar y agrupar elementos relacionados en un formulario, y se ha utilizado durante mucho tiempo en las aplicaciones de escritorio. Tiene el efecto de crear un cuadro alrededor de los objetos agrupados y mostrar una descripción a la derecha de cada elemento.

```
<link rel="stylesheet"  href="misestilos.css" >
 </head>
```

```
 <body>

<form>
  <fieldset>
      <legend>Amistad</legend>
      <input type="radio" name="radFriendship" value="Not_Applicable"
          id="radFriendNot_Applicable"/>
      <label for="radFriendNot_Applicable">No aplica</label>
      <input type="radio" name="radFriendship" value="acquaintance"
          id="radFriendaquaintence"/>
      <label for="radFriendaquaintence">Amistad</label>
      <input type="radio" name="radFriendship" value="friend"
          id="radFriendfriend"/>
     <label for="radFriendfriend">Amigo</label>
  </fieldset>
</form>

</html>
```

form

El elemento **form** es el contenedor que define los puntos de inicio y fin de una forma que un visitante del sitio puede llenar. Los campos que se encuentran entre el <form> apertura y cierre </ form> estarán asociados con esta forma. Es posible tener varios elementos de formulario en una página.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>

<form action="form-to-email.php" method="post">
  <div>
    <label for="txtname">Nombre:</label>
    <input type="text" name="txtname" id="txtname"/>
  </div>
  <div>
    <label for="txtcontacttel">Contacto Tel:</label>
    <input type="text" name="txtcontacttel" id="txtcontacttel"/>
  </div>
  <div>
    <input type="submit" name="cmdSubmit" id="cmdSubmit"
        value="Enviar información"/>
  </div>
</form>

</html>
```

label

El elemento **label** es invisible para los usuarios. Por defecto, la aplicación de una etiqueta **label** en un texto descriptivo de un control de formulario, no hace ninguna diferencia en su aspecto visual. Sin embargo, hay otros beneficios que se pueden obtener al vincular explícitamente el texto al control de formulario con la etiqueta.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>

 <input type="radio" name="radFriendship" value="Not_Applicable" id="b"/>
 <label for="b">No aplica</label>

</html>
```

input

Describir el elemento **input** simplemente no es sencillo, ya que hay mucha variación en la forma de una entrada y los atributos que utiliza o requiere, dependiendo del tipo de atributo especificado. Pero cualquiera que sea el tipo, la característica que es común a todos los elementos de entrada es que permiten a los usuarios introducir datos.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>

<form>
  <label for="firstname">Primer Nombre</label>
  <input type="text" name="firstname" id="firstname"/>
```

```
</form>

</html>
```

legend

legend se utiliza para proporcionar el texto del título de controles de formulario agrupados y textos contenidos en un conjunto de campos (fieldset). Por defecto, aparece el texto a la izquierda, sobre el contorno en caja que el **fieldset** crea.

```
<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>

<form>
  <fieldset>
      <legend>Amistad</legend>
      <input type="radio" name="radFriendship" value="Not_Applicable"
          id="radFriendNot_Applicable"/>
      <label for="radFriendNot_Applicable">No aplica</label>
      <input type="radio" name="radFriendship" value="acquaintance"
          id="radFriendaquaintence"/>
      <label for="radFriendaquaintence">Amistad</label>
      <input type="radio" name="radFriendship" value="friend"
          id="radFriendfriend"/>
     <label for="radFriendfriend">Amigo</label>
  </fieldset>
⋮
</form>

</html>
```

button

El botón de control de formulario es un intento de mejorar el botón "submit" (input type = "submit"), que solo puede contener una línea de texto (como valor del atributo), y establece posibilidades de estilo CSS.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
```

```
 </head>
 <body>

<form>
  <button value="proceed">Este es un <em>proceso</em>
  de búsqueda</button>
</form>

</html>
```

select

El control de formulario creado con una etiqueta **select**, es un contenedor de una serie de elementos opcionales que deben mostrarse en el navegador como un menú desplegable.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>

<form>
  <label for="favoritefood">Tarea favorita</label>
  <select name="favoritefood" id="favoritefood">
    <option>leer</option>
    <option>escribir</option>
    <option>hablar</option>
    ⋮
  </select>
</form>

</html>
```

textarea

Es un área de texto similar a la de entrada de texto, pero permite a la persona que está rellenando el formulario introducir varias líneas de información, en lugar de una sola línea, por lo que es mejor para la entrada de texto de forma libre.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">
```

```
<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>

<form>
  <label for="perfectday">Describa sus tareas de este día:</label>
  <textarea id="perfectday"></textarea>
  <input type="submit" value="enviar"/>
</form>

</html>
```

option

Cada etiqueta **option** de selección representa una opción que el usuario puede elegir. El texto contenido entre las etiquetas **<option>** apertura y cierre </ option> se muestran al usuario, mientras que el valor del atributo contiene los datos reales que se envían cuando se envía el formulario. El atributo seleccionado es útil para preseleccionar una opción de la lista al cargar la página.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>

<form>
  <label for="favoritefood">Tarea favorita</label>
  <select name="favoritefood" id="favoritefood">
    <option>leer</option>
    <option>escribir</option>
    <option>hablar</option>
    ⋮
  </select>
</form>

</html>
```

table

El elemento **table** se utiliza para presentar los datos en forma de rejilla (en filas y columnas), con los encabezados apropiados para identificar los datos que figuran en cada columna y fila.

En su forma más básica, una tabla se construye a partir de **tr** (renglón), y el elemento **td** (columna), además, el elemento **th** es utilizado para el marcado de los encabezamientos de columna o fila.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>

<table>
  <tr>
    <th>Producción de texto</th>
    <th>Borrador</th>
  </tr>
  <tr>
    <td>Ensayo</td>
    <td>de 20%</td>
  </tr>
  <tr>
    <td>Cantidad</td>
    <td>de 16%</td>
  </tr>
</table>

</html>
```

video

Una de las mejoras de HTML5 es procesar el video, para especificar en un documento un elemento de video se marca con <video> y se usan etiquetas de apertura y cierre, y solo unos pocos parámetros del control. Se recomienda usar un formato MP4 con H.264 y audio AAC.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


 <bgsound src="11 Tired.wav"/>

<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>

<video width="640" height="340" controls poster="manzana.jpg">
  <source src="escribir.mp4" type="video/mp4">
  <source src="escribir.ogg" type="video/ogg">
Your browser does not support the video tag.
</video>
 </body>
</html>
```

Los atributos de video **width** (ancho) y **height** (altura) dimensionan el video como se mostrará en navegador, **src** especifica el archivo origen a cargar y desplegar. OGG es reconocido por Opera, Safari y Firefox; MP4 por Safari, Internet Explorer y Google Chrome.

Antes de HTML5 no era posible reproducir video sobre una página Web, sin la ayuda de software complementario instalado en el navegador (plug-in), tal como Flash, QuickTime, RealPlayer, etc.

El elemento **<video>** puede utilizar cualquiera de los siguientes atributos específicos:

Atributo	Valor	Descripción
width	pixeles	Define el ancho del reproductor de video.
height	pixeles	Define el alto del reproductor de video.
controls	controls	Establece si los controles de reproducción deben de ser mostrados.
autoplay	autoplay	Define si el video debe de comenzar su reproducción tan pronto esté listo.
loop	loop	Especifica que el video debe de reproducirse de forma cíclica.
muted	muted	Establece que el audio debe de estar en silencio.
preload	auto metadata none	Especifica si el video debe de ser cargado al momento de carga de la página y de qué forma debe de hacerlo.
poster	URL	Especifica la imagen que debe de mostrarse antes de que el video comience su reproducción.

audio

Para insertar sonido dentro de una Web, se emplea en HTML5 el elemento **audio.** Este elemento encuentra su aplicación en podcasts y radio, sus atributo pueden ser:

<source>: Carga el archivo para diferentes tipos de audio.

autoplay: automáticamente comienza correr el audio.

preload: none el audio no debe ser cacheado; **metadata** indica solo obtener información sobre el medio; **auto** carga de inmediato el archivo.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>
<audio src="11 Tired.mp3"  controls>
<pre class="prettyprint lang-html"><code><audio controls>
<source src="11 Tired.ogg" type="audio/ogg" />
<source src="11 Tired.mp3" type="audio/mpeg" />
<object type="application/x-shockwave-flash" data="player.swf?soundFile=11
Tired.mp3.mp3">
<param name="movie" value="player.swf?soundFile=archivo.mp3" />
<a href="11 Tired.mp3">Descarga el archivo de audio</a>
</object>
</audio>

</body>
</html>
```

Sumario

A estas alturas de su estudio apoyado en el presente texto, Usted debe haber descubierto que HTML5 es un nuevo enfoque organizado para responder a la era de los dispositivos Internet móviles, conserva la filosofía de producir texto digital altamente estructurado; la familia básica de marcas HTML5 se ha prensentado con definiciones y breves ejemplos para familiarizarse con sus aplicaciones y de esta manera con el lenguaje de etiquetas reconocido y promovido en sus estándares por el W3C. Se hace evidente además, la necesidad de otros recursos para que los documentos HTML5, tales como multimedia, algoritmos programables y diseño gráfico presente contenidos atractivos.

En seguida abordaremos los recursos de las hoja de estilo CSS, mismos que permiten construir la imagen de diseño gráfico y tipológico para los documentos Web. Dentro de los ejemplos seguiremos agregando más etiquetas de la familia del lenguaje HTML5.

URL`s

http://www.w3schools.com/html/default.asp

http://thinkwasabi.com/2009/05/editores-texto-mac/

https://developer.mozilla.org/es/docs/Gu%C3%ADa_de_referencia_de_CSS

http://www.w3schools.com/html/html_intro.asp

http://librosWeb.es/css/capitulo_1/breve_historia_de_css.html

http://www.mclibre.org/consultar/amaya/otros/otros_historia.html#Cascading

http://www.w3.org/2005/11/Translations/Lists/ListLang-es.html

http://htmledit.squarefree.com

http://manual-xhtml.blogspot.mx

http://www.whatwg.org/specs/Web-apps/current-work/

http://html.conclase.net/w3c/html401-es/struct/dirlang.html#adef-lang

http://www.materialesdelengua.org/LITERATURA/index_literatura.htm

http://reference.sitepoint.com/html/elements-text-formatting

http://simon.html5.org/html-elements

Referencias

[1] Pastrana Erika (2013) Focus on mapping. Nature Methods 10(6): 481. Consulta: 24 de agosto de 2013, de http://neuroscience.mssm.edu/files/MappingTheBrain.pdf

[2] L Morgan & W Lichtman Jeff (2013) Why not connectomics?. Nature Methods 10 (6):494-500. Consulta: 24 de agosto de 2013, de http://www.nature.com/nmeth/journal/v10/n6/full/nmeth.2480.html

[3] Bloomfield, Leonard. 1933. Language. New York: Henry Holt.

[4] Balari, S., Benítez-Burraco, A., Camps, M., Longa, V. M. & Lorenzo, G. (2010), "La importancia de ser moderno. Problemas de método e ideología en el debate sobre la cognición y la conducta de los Neandertales". Ludus Vitalis. Revista de Filosofía de las Ciencias de la Vida XVIII/34, pp. 143-170. Consulta: 24 de agosto de 2013, de http://www.ludusvitalis.org/indice/actual.html

[5] Balari, S. & Lorenzo, G. (2009), "Computational phenotypes: Where the theory of computation meets Evo-Devo", Biolinguistics 3(1), pp. 2-60. Consulta: 24 de agosto de 2013, de http://www.unioviedo.es/biolang/fundamento/publicaciones.php

[6] Sporns Olaf (2013) Making sense of brain network data. Nature Methods 10(6): 491-493. Consulta: 24 de agosto de 2013, de http://www.nature.com/nmeth/journal/v10/n6/full/nmeth.2485.html

[7] Helmstaedter Moritz (2013) Cellular-resolution connectomics: challenges of dense neural circuit reconstruction. Nature Methods 10(6): 501-507. Consulta: 24 de agosto de 2013, de http://www.nature.com/nmeth/journal/v10/n6/full/nmeth.2476.html

[8] Benítez Burraco A (2009) Genes y lenguaje. Barcelona: Reverté

[9] García, G. Emilio (2008) De las neuronas espejo a la teoría de la mente. R. De psicología y educación. 1(3): 69-90. Consulta: 24 de agosto de 2013, de http://eprints.ucm.es/9972/1/Revista_Psicologia_y_Educacion.pdf

[10] Torres Ariel (2009) bit viaje al interior de la revolución digital. México: Atlantida

[11] Owe Olaf et al. (2004) From Object-Orientation to formal methods. Berlin: Springer

[12] http://www.youtube.com/watch?v=N5mLK4V5P30

[13] Aronowitz Stanley et al. (1998) Tecnociencia y cibercultura. Barcelona: Paidós

[14] Vaidhyanathan Siva (2010)La Googlización de todo. México: Oceano

[15] Castells Manuel (2012) Comunicación y poder. México: Siglo XXI

[16]Marie-laure Ryan (2004) La narración como realidad virtual. Barcelona: Paidós

[17] Centro Español de Derechos Reprográficos (CEDRO). Consulta 24 de agosto de 2013, de http://cedro.org/nosotros/funciones/internacional

[18] Polly Wakefield (2012) GOOGLE PENGUIN ALGORITHM REVIEW 2012. Relevance Web Marketing Blog. Consulta 4 de junio de 2012, de http://www.relevanceWeb.com/blog/item/google-penguin-algorithm-review-2012?category_id=20

[19] George P. Landow (2006) Hipertexto 3.0. Barcelona: Paidós

[20] George P. Landow (1997) Teoría del Hipertexto. Barcelona: Paidós

[21] Fundación del español urgente. http://www.fundeu.es http://crosspaper.e-mutation.com/books/5/914/crosspaper.html

[22] Eloy Martos et al. (2009) Prácticas de lectura y escritura. Brasil: Editora Universitaria. Consulta 24 de agosto de 2013, de http://universidadeslectoras.org/docs/practicas_lectura_y_escritura.pdf

[23] Philippe Quéau (1995) Lo virtual: virtudes y vértigo. Barcelona: Paidós

[24] Pierre lévy (1999) ¿Qué es lo virtual?. Barcelona: Paidós

[25] Tim Berners-Lee (2002) Tejiendo la Red. Madrid: Siglo XXI. Consulta: 15 de agosto de 2013, de http://books.google.es/books/about/Tejiendo_la_red.html?hl=es&id=QRe-iutQQmQC
[26] Uniform Resource Identifier (URI): Generic Syntax. Consulta: 15 de agosto de 2013, de http://tools.ietf.org/html/rfc3986
[27] Comparison of layout engines (2013) Consulta: 19 de agosto de 2013, de http://en.wikipedia.org/wiki/Comparison_of_layout_engines_(Cascading_Style_Sheets)#Explanation_of_the_tables
[28] CSS Specifications(2013) Consulta: 19 de agosto de 2013, de http://www.w3.org/Style/CSS/current-work
[29] UTF-8, a transformation format of ISO 10646. Consulta: 19 de agosto de 2013, de http://tools.ietf.org/html/rfc3629
[30] Ramos Amrtín Alicia (2011) Aplicaciones Web. Madrid: Paraninfo. Consulta: 19 de agosto de 2013, de http://books.google.es/books?id=LXs3YlMoeNgC&pg=PA55&dq=CSS+sintaxis&hl=es&sa=X&ei=mBYSUrdMxMLYBdeRgYAE&ved=0CFIQ6AEwBg#v=onepage&q=CSS%20sintaxis&f=false
[31] http://www.nngroup.com/articles/f-shaped-pattern-reading-Web-content/
[32] Tags for Identifying Languages http://www.rfc-editor.org/rfc/bcp/bcp47.txt

Capítulo II: CSS3

2.1. CSS

Las hojas de estilo en cascada (Cascading Style Sheets, por sus siglas en inglés CSS), aplican criterios gráficos a las páginas Web, se apoyan en palabras de lenguaje natural, en inglés, para dotar al lenguaje de **etiquetas o marcas**, de selectores, propiedades y atributos; su estructura se basó en el modelo de reglas construidas, definidas por uno o más selectores y una partición de estilos que aplican a los elementos HTML que apuntan los selectores dentro del documento de hipertexto[1]. Cada partición es un conjunto de instrucciones de estilo con la forma:

Selector { propiedad: valor; }

El W3C publica el CSS nivel 1 en 1996[2], y es hasta 1998 que se publica la segunda versión oficial por este mismo organismo internacional, el CSS nivel 2 y la última versión oficial la 2.1 aparece en 2009; es importante advertirle apreciable lector, que a pesar de estar 100% operativo el nivel 3 CSS, aún no existe norma oficial, se trabaja con el borrador CSS3 del W3C[3].

En CSS Nivel 2 se define el estilo en un documento único de varios capítulos. Sin embargo, el Grupo de Trabajo del WC3 decidió adoptar un enfoque modular, donde a cada módulo le asigna una parte de las definiciones CSS, en lugar de definir una única especificación monolítica. Esto rompe la especificación en segmentos de código manejables y permite complejizarse de manera incremental a los recursos CSS, hasta alcanzar gran calidad en el diseño de imagen gráfica en las páginas Web.

Dado que los diferentes módulos CSS 3 se encuentran en diferentes niveles de estabilidad, el W3C decidió publicar este perfil para definir el alcance actual y el estado de Cascading Style Sheets a partir de finales de 2010[4]. Este perfil incluye solamente las especificaciones que considera la experiencia como estables y para los que tenemos suficiente experiencia en la ejecución sobre las nuevas tecnologías móviles, estamos seguros de que la estabilidad será irreversible, además, mantiene el modelo de objetos de documento (Document Object Model DOM).

Tenga en cuenta también que, si bien no prevemos cambios significativos en las especificaciones que formarán la norma oficial CSS 3, su inclusión no quiere decir que no estén en aplicación. Ahora mismo al escribir este libro (agosto de 2013) CSS 3 es tan popular que los principales navegadores implementan el uso de selectores, pseudo-clases y muchas propiedades; tales navegadores son Google Chrome, Internet Explorer, FireFox, Safari, y Opera. Estos navegares manejan simultáneamente CSS 2.1 y el nuevo modelo CSS 3.

En el capítulo anterior se introducen brevemente los casos para aplicar especificaciones de estilo a un documento HTML, recordando solo dos casos:

1) CSS se escribe dentro del documento, con el elemento **<style>** dentro del elemento **<head>**, este caso encuentra funcionalidad con situaciones requeridas de pocos estilos aplicados y cuando solo estarán en una sola página.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">

  <style type="text/css">
  p { color: blue; font-family: optima; }
</style>
 </head>
 <body>

<p>Un párrafo de texto.</p>

 </body>
</html>
```

2) CSS se escribe fuera del documento, con el elemento **<link>,** dentro del elemento **<head>**, se enlaza un archivo **.CSS** externo; este caso encuentra funcionalidad en las situaciones en las que requieren aplicar muchos estilos y cuando varias páginas del sitio responderán a un mismo enfoque gráfico. Se abre un archivo de texto, se introduce la especificación:

```
p { color: blue; font-family: optima; }
```

y se guarda como misestilos.css. Dentro del elemento **<head>** se enlaza al archivo que debe estar dentro de la carpeta raíz del documento HTML.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">

<link rel="stylesheet" type="text/css" href="/css/estilos.css" media="screen" />
 </head>
 <body>

<p>Un párrafo de texto.</p>

 </body>
</html>
```

```
@charset "UTF-8";
/* CSS archivo misestilos.css */
 p { color: blue; font-family: "Courier New", Courier, monospace; }
```

Este método hace **referencia directa** a la palabra clave del elemento (se conoce como **método selector de elemento**), para el caso anterior, la palabra clave **p**, referencia a cada etiqueta **<p>** del documento, este método aplica para cada elemento **HTML**, pero cuando anhelemos un estilo distinto sobre cada elemento **<p>**, este modelo de referencia no es el más adecuado.

Referenciado con el atributo id, con este recurso asignamos un nombre para identificar una etiqueta única para aplicar un estilo particular (**selector de atributo**). Dentro de nuestro archivo .CSS escribimos con el símbolo # al frente del elemento a identificar.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">

<link rel="stylesheet"  href="misestilos.css" />
 </head>
 <body>


<p id="estilo1">Un párrafo de texto.</p>

<p >Un párrafo de texto.</p>

 </body>
```

```
</html>
```

```
@charset "UTF-8";
/* CSS CSS archivo misestilos.css */
 #estilo1 { color: blue; font-family: "Courier New", Courier, monospace; }
```

De este modo será especificado solo el estilo a los elementos que contengan el identificador **estilo1**. Una forma alternativa es usar el modo de referencia con el atributo **class (selector clase)**, se declara la regla con un punto antes del nombre, este modo es más flexible a cualquier elemento que se quiera afectar. Por ejemplo el archivo CSS del siguiente ejemplo:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">

<link rel="stylesheet"  href="misestilos.css" />
 </head>
 <body>



<p class="estilo1">Un párrafo de texto.</p>

<p >Un párrafo de texto.</p>

 </body>
</html>
```

```
@charset "UTF-8";
/* CSS CSS archivo misestilos.css */
 .estilo1 { color: blue; font-family: "Courier New", Courier, monospace; }
```

Cuando se desea ser más específico, por ejemplo, aplicar el estilo a todos aquellos elementos de algún tipo, se usará el atributo **class**, con selector de la siguiente manera:

p.estilo1{…}	todos los elementos p
p .estilo1{…}	todos los elementos p que contengan solo estilo1
p, .estilo1{…}	la suma de los dos casos anteriores

Cuando se desea aplicar varios efectos a un mismo elemento, class incluirá cada efecto separado por un espacio en blanco.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">
  <meta name="keyword" content="HTML5, CSS3, Javascript">
```

```
<link rel="stylesheet"  href="misestilos.css" />
 </head>
 <body>


<p class="estilo1 estilo2 estilo3">Un párrafo de texto.</p>

<p >Un párrafo de texto.</p>

 </body>
</html>
```

```
@charset "UTF-8";
/* CSS CSS archivo misestilos.css */
 .estilo1  { color: blue; font-family: "Courier New", Courier, monospace; }
 .estilo2  { font-size: 30px; }
 .estilo3    { font-weight: bold; }
```

Tenga presente que un selector es un patrón que aplica a elementos del documento HTML, {…} entre estas llaves se declara separada por (;) cada propiedad: valor. El archivo de hoja de estilo comienza con las reglas arroba (@):[5]

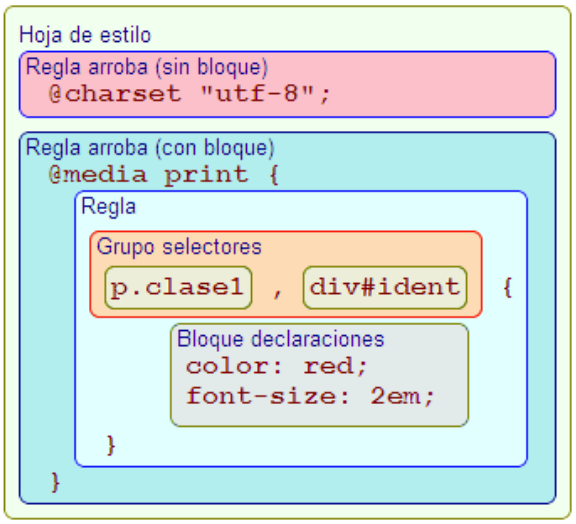
@import adjunta otras hojas de estilo.

@charset define la codificación de símbolos en hoja de estilo

@media especifica los medios de contenido como all, speech, screen, print, ...

@page define características de caja de página

Al conjunto de un bloque de selectores con un grupo de declaraciones que comparten, se le suele llamar regla de estilo.



Tenga presente que CSS 3 puede referenciar además de los atributos **id class**, cualquier otro atributo, por ejemplo el caso de **name**,

p[name]={Font-size: 20px}

Este código solo referencia los elementos p que tienen el atributo **name**.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">

<link rel="stylesheet"  href="misestilos.css" />
 </head>
 <body>



<p class="estilo1 estilo2 estilo3">Un párrafo de texto.</p>

<p name="casa">Un párrafo de texto con atributo name.</p>


<p >Un párrafo de texto.</p>

 </body>
</html>
```

```
@charset "UTF-8";
/* CSS CSS archivo misestilos.css */
 .estilo1  { color: blue; font-family: "Courier New", Courier, monospace; }
 .estilo2  { font-size: 30px; }
 .estilo3       { font-weight: bold; }

 p[name="casa"] {font-size:18px; color:#6F0}
```

Además de los selectores anteriores del CSS 2.1, HTML5 incorpora los nuevos selectores: pseudo-clases y pseudo-elementos. Por supuesto que los selectores anteriores son los básicos y en muchos casos los necesarios para una buena programación del enfoque gráfico, sin embargo, cuando tenemos hijos anidados y referenciados a un elemento HTML, podemos aprovechar esto para afectar a alguno de ellos sin necesidad de conocer sus atributos o el valor de estos.[6]

Selector	Descripción
*	Selector universal, son todos los elementos del CSS.
E	E representa cualquier elemento del tipo E (span, p, …).
E F	Todos los elementos F que sean descendientes de E.
E > F	Todos los elementos F que sean hijos de E.
E:first-child	De esta forma podemos seleccionar el primer elemento de tipo E.
E:link , E:visited	Selecciona los elementos E que sean enlaces y no hayan sido visitados (:link) y los sí visitados (:visited).
E:active , E:hover , E:focus	Selecciona los elementos de tipo E , en sus correspondientes acciones.
E:lang(c)	Cogemos los elementos del tipo E que estén en el idioma (humano) especificado en (c).
E + F	Se trata de cualquier elemento F inmediatamente despues del elemento del tipo E.
E[foo]	Elementos del tipo E con el atributo foo.
E[foo="ejemplo"]	Elementos del tipo E con el atributo foo que igual a “ejemplo”
E[foo~="ejemplo"]	Elementos del tipo E con el atributo foo contenga “ejemplo”. Se pueden añadir varias palabras separadas por espacios. (~ =ALT + 0126).
E[lang|="es"]	Similar al anterior, pero se referirá a todos los elemento E tal que su atributo lang comience por “es”. Por ejemplo: “es_ES”, “es_CA”,…
E[foo$="ejemplo"]	Elementos del tipo E en el que el atributo foo termine con “ejemplo”.
DIV.ejemplo	Todos los elementos DIV que sean de la clase ejemplo.

2.2. Pseudo clases

pseudo clase nth-child

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
```

```
 <body>
 <table cellpadding="0" cellspacing="0" border="1"
  summary="Esta tabla expresa las horas de lectura sobre una lectura de novela">
  <caption>Días de lectura</caption>
  <thead id="foo">
    <th id="días">Días</th>
    <th id="horas">Horas</th>
  </thead>
  <tbody>
    <tr>
      <th id="wh" headers="días">Lunes</th>
      <td headers="wh horas">1.9 </td>
    </tr>
    <tr>
      <th id="pt" headers="días">Martes</th>
      <td headers="pt horas">2</td>
    </tr>
    <tr>
      <th id="jp" headers="días">Miércoles</th>
      <td headers="jp horas">3.2</td>
    </tr>
    <tr>
      <th id="tb" headers="actor">Jueves</th>
      <td headers="tb horas">2.4</td>
    </tr>
    <tr>
      <th id="pd" headers="días">Viernes</th>
      <td headers="pd horas">1.5</td>
    </tr>
    <tr>
      <th id="cb" headers="días">Sabado</th>
      <td headers="cb horas">1.9</td>
    </tr>
    <tr>
      <th id="sm" headers="días">Domingo</th>
      <td headers="sm horas">4</td>
    </tr>
  </tbody>
</table>

 </body>
</html>
```

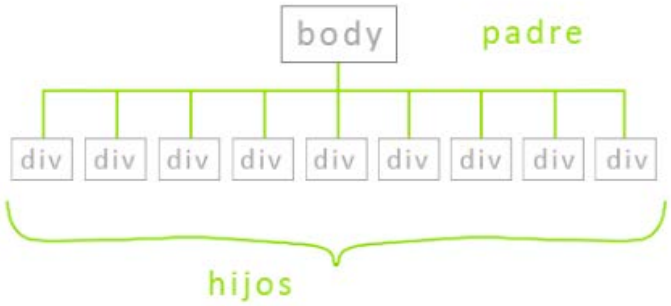
```
@charset "UTF-8";
/* CSS CSS archivo misestilos.css */
/* here's pseudoclass-nthchild in action */

tr:nth-child(2n+1)
{
  background:#ffc;
  font-style:italic;
}

tr
{
  background:#FF0;
  font-style:normal;
}
```

Esta pseudo-clase afecta a los elementos relacionados sobre la base de sus posiciones dentro de la lista de un elemento primario. La pseudo-clase acepta un argumento, **n**, que

puede ser una palabra clave, un número, o un número de expresión de la forma **an + b**. Para obtener más información, consulte Descripción: nth-child expresiones pseudo-clase. Además, puede usar las palabras claves **odd** y **even** como argumento para referirse a elementos impares y pares respectivamente.

Ejemplos:

```
p:nth-child(1) {
  ⋮ declarations
}


tr:nth-child(odd) {
  ⋮ declarations
}
```



Aplicando el ejemplo para párrafos:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">



<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>
 <div id="lucas">
 <p class="linea1"> Texto de línea uno</p>
 <p class="linea2"> Texto de línea dos</p>
 <p class="linea3"> Texto de línea tres</p>
 <p class="linea4"> Texto de línea cuatro</p>
    </div>
 </body>
</html>
```

```
@charset "UTF-8";
/* CSS CSS archivo misestilos.css */
/* here's pseudoclass-nthchild in action */

p:nth-child(2)
{
```

```
  background:#ffc;
  font-style:italic;
}
```

pseudo clase nth-last-child

Las pseudo clases se emplean para apuntar a elementos HTML sin requerir emplear identificadores o clases. La pseudo clase **nth-last-child()** hace referencia al último hijo, en función de las posiciones de la lista de elementos agrupados por un elemento HTML. La referencia apunta a **n-1**, donde n es el número total de la lista.

Ejemplos:

Este modelo hace referencia a los últimos 4 elementos de la lista

```
li:nth-last-child(-n+4) {
  ⋮ declarations
}
```

Este modelo hace referencia al último elemento secundario respecto del elemento padre:

```
p:nth-last-child(1) {
  ⋮ declarations
}
```

pseudo clase nth-of-type

En este caso, la pseudo clase **nth-of-type** hace referencia a elementos que preceden a un valor **n**; acepta valores numéricos, las palabras clave **odd, even** y fórmulas numéricas. Por ejemplo:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">
```

```
<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>

 <section>
 <p>Texto de línea uno</p>
 <p>Texto de línea dos</p>  <!--este elemento es referido-->
 <p>Texto de línea tres</p>
 <p>Texto de línea cuatro</p>
 </section>

 </body>
</html>
```

```
@charset "UTF-8";
/* CSS CSS archivo misestilos.css */


p:nth-of-type(2)
{
  color:#F00;

}
```

pseudo clase nth-last-of-type

En este caso, la pseudo clase **nth-last-of-type** hace referencia a elementos que preceden al último elemento; acepta un valor **n**, numérico, las palabras clave **odd y even** y fórmulas numéricas. Por ejemplo:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>

 <div>
      <span>Microrrelato!</span>
      <span>narración de extensión breve:(</span>
      <em>Texto preciso y sintetizado. </em>
      <span>Estructura planteamiento, nudo y desenlace!</span>
      <strike> Se inicia, se desarrolló y se resulve a un mismo tiempo </strike>
      <span>  Con espacios mínimos es conducido a la sorpresa...</span>
    </div>

 </body>
</html>
```

```
@charset "UTF-8";
/* CSS CSS archivo misestilos.css */
```

```
span:nth-last-of-type(2) {
      background-color: lime;
    }
```

pseudo clase last-of-type

En este caso, la pseudo clase **last-of-type** hace referencia al último elemento hermano de un tipo dado en la lista de elementos hijo, de su elemento padre. Ejemplificamos esta pseudo clase para el elemento **strong** dentro de un elemento **p**.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>

 <p>
  <em>La bofetada sonó en el hueco de sus manos:(</em>
  <strong>Pocos se habían dado cuenta de lo sucedido :(</strong>
  <strong>Nos miramos sorprendidos, pero el silencio aturdió el momento</strong>
</p>

 </body>
</html>
```

```
@charset "UTF-8";
/* CSS CSS archivo misestilos.css */


p strong:last-of-type {
  color: lime;
}
```

pseudo clase only-child

En este caso, la pseudo clase **only-child** se aplica cuando un elemento padre contiene un solo hijo, y hace referencia a ese elemento hijo único. Ejemplificamos esta pseudo clase con un elemento **em** dentro de un elemento **p.**

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>

 <p>
  <em>La bofetada sonó en el hueco de sus manos</em>
</p>


    <p>
  <em>La bofetada sonó en el hueco de sus manos</em>
<strong>Pocos se habían dado cuenta de lo sucedido :(</strong>
  <strong>Nos miramos sorprendidos, pero el silencio aturdió el momento</strong>

</p>

 </body>
</html>
```

```
@charset "UTF-8";
/* CSS CSS archivo misestilos.css */


p em:only-child {
  color: lime;
}
```

pseudo clase only-type

En este caso, la pseudo clase **only-type** hace referencia al único elemento de cierto tipo.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>
 <div id=”ContenedorPadre”>
<form>
<label for=”nombre”>Nombre</label>
<input type=”text” id=”nombre” required/>
<label for=”apellido”>Apellido</label>
<input type=”text” id=”apellido” required/><br/>
<label for=”email”>E-mail</label>
<input type=”email” id=”email” required/><br/>
<label for=”direccion”>Direcci&oacute;n</label>
<input type=”text” id=”direccion”/><br/>
<label for=”nacionalidad”>Tel&eacute;fono</label>
<input type=”text” id=”telefono”/><br/>
<label for=”vivo”>Vivo en</label>
<input type=”text” id=”vivo” readonly value=”México”/><br/>
<label for=”extra”>Ciudad</label>
<input type=”text” id=”extra” disabled/><br/>
<textarea id=”textos” required></textarea>
</form>
</div>

 </body>
</html>
```

```
@charset "UTF-8";
/* CSS CSS archivo misestilos.css */


form:only-of-type

{
      color:#F3F;
        }
```

pseudo clase root

Esta pseudo-clase coincide con un elemento que es el elemento raíz del documento. En los documentos HTML, el selector siempre será el elemento HTML. Es decir, representa un elemento que es la raíz del documento. En el siguiente ejemplo al elemento **<big>** se le asignan estilos con la pseudo clase :root big {…}, con el selector big {…}, y el selector de clase big.root{…}; observe cómo se sobrescriben los colores en función de la pseudo clase.[7]

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

```
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>
<div class="ejemplo-linea">
    <big>Premio 1</big>
    <big class="linroot">Premio 2</big>
    <big class="linroot" id="idroot">Premio 3</big>
    <big class="linroot" id="idroot" style="color: maroon">Premio 4</big>
</div>

<!-- ESTILO -->

<style>
    :root big {
        color: red;
        background-color: lime;
        border: solid 1px;
        font-size: 2em;
        margin: 0.2em;
        padding: 0.1em;
    }
    big {
        color: magenta;
        background-color: beige!importante;
    }
    big.linroot {
        color: blue;
        background-color: aqua;
    }
    big#idroot {
        color: orange;
        background-color: green;
    }
</style>

 </body>
</html>
```

pseudo clase empty

Esta pseudo-clase selecciona elementos que no tienen hijos. Los nodos de elementos y nodos de texto no vacíos se consideran hijos; nodos vacíos de texto, comentarios e instrucciones de procesamiento no cuentan como hijos. Un nodo de texto se considera vacío si tiene una longitud de datos cero, de modo que, por ejemplo, un nodo de texto con un único espacio no está vacío.

```
<!DOCTYPE html>
<html lang="es">
<head>
```

```
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
 </head>
 <body>
<div class="ejemplo-linea" id="vacios">
    <p title="Un &lt;p&gt; vacío"></p>
    <input type="text" value=" &lt;entrada&gt;" />
</div>

<!-- ESTILO -->
<style>
    div#vacios *:empty {
        border: blue solid 1px;
        padding: 0.5em;
    }
</style>

 </body>
</html>
```

pseudo clase target

Esta pseudo-clase coincide con un elemento que es el objetivo de un identificador de fragmento en URI del documento. El identificador de fragmento en una URI tiene un caracter # seguido de un identificador de nombre que coincide con el valor de un atributo **id** de un elemento en el documento.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">



 </head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>pseudo clase target &ndash; CSS Lightbox</title>
<style type="text/css">
div.lightbox {
  display: none;
  position: fixed;
  left: 0;
  top: 0;
  width: 100%;
  height: 100%;
}

div.lightbox:target {
  display: table;
}
```

```
div.lightbox figure {
  display: table-cell;
  margin: 0;
  padding: 0;
  width: 100%;
  height: 100%;
  vertical-align: middle;
}

div.lightbox figure figcaption {
  display: block;
  margin: auto;
  padding: 8px;
  background-color:#FFC;
  height: 250px;
  position: relative;
  overflow: auto;
  border: 1px #000000 solid;
  border-radius: 10px;
  text-align: justify;
  font-size: 14px;
}

div.lightbox figure .closemsg {
  display: block;
  margin: auto;
  height: 0;
  overflow: visible;
  text-align: right;
  z-index: 5001;
  cursor: default;
}

div.lightbox figure .closemsg, div.lightbox figure figcaption {
  width: 300px;
}

.closemsg::after {
  content: "\00D7";
  display: inline-block;
  position: relative;
  right: -20px;
  top: -10px;
  z-index: 5002;
  color:#FFC;
  border: 1px #ffffff solid;
  border-radius: 10px;
  width: 20px;
  height: 20px;
  line-height: 18px;
  text-align: center;
  margin: 0;
  background-color: #000000;
  font-weight: bold;
  cursor: pointer;
}

.closemsg::before {
  content: "";
  display: block;
  position: fixed;
  left: 0;
```

```
  top: 0;
  width: 100%;
  height: 100%;
  background-color: #000000;
  opacity: 0.85;
}
</style>

</head>

<body>

<h1>Microrrelatos</h1>

<p>Recomendaciones de narrativa corta&hellip;</p>

<p>[ <a href="#example1">Ejemplo #1</a> | <a href="#example2">Ejemplo #2</a>
]</p>

<p>Mundo palabras&hellip (http://www.mundopalabras.es);</p>

<div class="lightbox" id="example1">
  <figure>
    <a href="#" class="closemsg"></a>
    <figcaption>Bárbaros perfeccionistas: Las tropas de asalto llegaron a la
biblioteca decididos a desembarazarse de los libros más dañinos para el sistema
impuesto por el gobierno. Pero no lo harían, como otras veces en el pasado, de
una forma arbitraria y sin criterio. Esta vez llevaban consigo a más de un
centenar de especialistas que cribarían los ejemplares, leyendo las obras de los
autores menos prestigiosos, para que no se convirtieran, con el tiempo, en
biblias revolucionarias. En alguno de aquellos millones de ejemplares habría
alguno que querría pasar desapercibido a los ojos de los censores. Pero esta vez
no se libraría de las llamas purificadoras.<br />
    Etiam varius adipiscing mi eget imperdiet. Nulla quis vestibulum leo. Integer
molestie massa ut massa commodo in blandit purus aliquam. Mauris sit amet posuere
massa. Ut a eleifend augue. Proin sodales mauris nec tellus pharetra
dictum.</figcaption>
  </figure>
</div>

<div class="lightbox" id="example2">
  <figure>
    <a href="#" class="closemsg"></a>
    <figcaption>Cantante: Se filtró la noticia de que iba a sacar un nuevo disco,
mucho más impactante, en estilo, que el anterior. Alguien del grupo o de la
productora, aseguró, se estaba sacando un sueldo extra con esa exclusiva. Por
ello, y sin que sirviera de precedente, obligó a su manager a convocar una rueda
de prensa, y con el tiempo que tenía para prepararla -casi un mes- sería, con
toda seguridad, multitudinaria. Y sería entonces cuando impactaría al mundo,
declarando que era otra, y no ella, quien cantaba en todos los temas, y que así
había sido desde el comienzo ....</figcaption>
  </figure>
</div>

</body>
</html>
```

pseudo clase enabled

Esta pseudo-clase afecta a elementos de interfaz de usuario que están habilitados. Un elemento se activa cuando se puede obtener el foco, esto por lo general significa que el elemento se puede seleccionar, hacer clic en, o que acepte la entrada de texto.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">



 <link href="misestilos.css" rel="stylesheet" type="text/css">
</head>



<body>

<form action="url_of_form">
      <label for="FirstField">Campo 1 (enabled):</label> <input type="text"
id="FirstField" value="Nombre"><br />
      <label for="SecondField">Campo 2 (disabled):</label> <input type="text"
id="SecondField" value="Profesión" disabled="disabled"><br />
      <input type="submit" value="Capturar" />

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

```
@charset "UTF-8";
/* CSS CSS archivo misestilos.css */

input:enabled {
      color: #22AA22;
    }
    input:disabled {
      color: #D9D9D9;
    }
```

pseudo clase disabled

La pseudo-clase **:disabled** hace referencia a cualquier elemento desactivado. Un elemento se desactiva por ejemplo, seleccionado, haciendo clic, al aceptar la introducción de texto o aceptar el foco. El elemento también tiene un estado habilitado, en el que se puede activar o aceptar el foco.

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

```
  <meta name="description" content="ejemplo adjuntar CSS">


 <link href="misestilos.css" rel="stylesheet" type="text/css">
</head>


<body>

<form action="#">
  <fieldset>
    <legend>Ensayos favoritos</legend>
    <input type="text" name="Ensayo1" disabled>
    <input type="text" name="Ensayo2" disabled>
    <input type="text" name="Ensayo3" disabled>
    <input type="text" name="Ensayo4" disabled>
    <input type="text" name="Ensayo5" disabled>
    <input type="text" name="Ensayo6" disabled>
  </fieldset>
  <fieldset>
    <legend>Novelas favoritas</legend>
    <label>
      <input type="checkbox" name="capturar novela" value="true" checked>
      Desea reportar sus novelas favoritas
    </label>
    <input type="text" name="novela1" disabled>
    <input type="text" name="novela2" disabled>
    <input type="text" name="novela3" disabled>
    <input type="text" name="novela4" disabled>
    <input type="text" name="novela5" disabled>
    <input type="text" name="novela6" disabled>
  </fieldset>
</form>
</body>
```

```
@charset "UTF-8";
/* CSS CSS archivo misestilos.css */

input[type="text"]:disabled { background:#FFC; }
```

pseudo clase not

La pseudo-clase **:not()** permite declarar una excepción. Los estilos aplicados serán referidos, excepto a aquellos incluidos en la referencia entre paréntesis:

: not (.parrafo1){color: blue}

Esta pseudo-clase es también conocida como la negación pseudo-clase. El argumento que se necesita puede ser cualquier selector simple, pero no puede contener o bien la negación pseudo-clase o un pseudo-elemento.

pseudo clase checked

La pseudo clase **:checked** hace referencia a los checkbox y radio que están seleccionados.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


 <link href="misestilos.css" rel="stylesheet" type="text/css">
</head>

<style type="text/css">
#expand-btn {
    margin: 0 3px;
    display: inline-block;
    font: 12px / 13px "Lucida Grande", sans-serif;
    text-shadow: rgba(255, 255, 255, 0.4) 0 1px;
    padding: 3px 6px;
    border: 1px solid rgba(0, 0, 0, 0.6);
    background-color: #969696;
    cursor: default;
    -moz-border-radius: 3px;
    -webkit-border-radius: 3px;
    border-radius: 3px;
    -moz-box-shadow: rgba(255, 255, 255, 0.4) 0 1px, inset 0 20px 20px -10px
white;
    -webkit-box-shadow: rgba(255, 255, 255, 0.4) 0 1px, inset 0 20px 20px -10px
white;
    box-shadow: rgba(255, 255, 255, 0.4) 0 1px, inset 0 20px 20px -10px white;
}

#isexpanded:checked ~ #expand-btn, #isexpanded:checked ~ * #expand-btn {
    background: #B5B5B5;
    -moz-box-shadow: inset rgba(0, 0, 0, 0.4) 0 -5px 12px, inset rgba(0, 0, 0, 1)
0 1px 3px, rgba(255, 255, 255, 0.4) 0 1px;
    -webkit-box-shadow: inset rgba(0, 0, 0, 0.4) 0 -5px 12px, inset rgba(0, 0, 0,
1) 0 1px 3px, rgba(255, 255, 255, 0.4) 0 1px;
    box-shadow: inset rgba(0, 0, 0, 0.4) 0 -5px 12px, inset rgba(0, 0, 0, 1) 0
1px 3px, rgba(255, 255, 255, 0.4) 0 1px;
}

#isexpanded, .expandable {
    display: none;
}

#isexpanded:checked ~ * tr.expandable {
    display: table-row;
    background: #cccccc;

}

#isexpanded:checked ~ p.expandable, #isexpanded:checked ~ * p.expandable {
    display: block;
    background: #cccccc;
}
</style>

<body>
<input type="checkbox" id="isexpanded" />
```

```
<h1>Relación de libros</h1>
<table>
    <thead>
        <tr><th>Librero #1</th><th>Librero #2</th><th>Librero #3</th></tr>
    </thead>
    <tbody>
        <tr
class="expandable"><td>[Libro]</td><td>[Libro]</td><td>[Libro]</td></tr>
        <tr><td>[Libro]</td><td>[Libro]</td><td>[Libro]</td></tr>
        <tr><td>[Libro]</td><td>[Libro]</td><td>[Libro]</td></tr>
        <tr
class="expandable"><td>[Libro]</td><td>[Libro]</td><td>[Libro]</td></tr>
        <tr
class="expandable"><td>[Libro]</td><td>[Libro]</td><td>[Libro]</td></tr>
    </tbody>
</table>

<p>[Captura ]</p>
<p><label for="isexpanded" id="expand-btn">captura de registro</label></p>
<p class="expandable">[another sample text]</p>
<p>gracias</p>
</body>
</html>
```

2.3.Selectores con operadores >,+ y ~

El selector **>** refiere que el elemento a ser referido por la regla es el elemento de la derecha cuando tiene a la izquierda el elemento padre.

```
div > p.texto {
color :#fffff
}
```

El selector **+** referencia al elemento de la derecha cuando es inmediatamente precedido por el de la izquierda. Ambos elementos deben compartir el mismo padre:

```
 p.texto + p {
color :#fffff
}
```

El selector ~es similar al anterior solo que el elemento referido no necesita estar precedido de inmediato por el elemento de la izquierda.

2.4.Pseudo elementos :before :after

Primero que nada tenemos que aclarar que "**:before**" y "**:after**" no son los únicos pseudo elementos que existen, la lista completa abarca a **":first-letter**", **":first-line**", **":before**", "**:after**" y **":selection**", pero como en esta ocasión nos estamos enfocando únicamente a los dos ya mencionados, cada vez que nos refiramos con el término "pseudo elemento" nos enfocaremos a ":**after**" y "**:before**".

La función de un pseudo elemento es sencilla, básicamente es hacer lo que su nombre indica, crear un elemento falso para insertarlo antes o después del contenido del elemento al que se ha orientado. La codificación de los pseudo elementos es bastante sencilla de realizar, basta con recordar dos sencillos aspectos para crearlos de manera correcta.[8]

Primero debemos crear un selector que indique a qué elemento le vamos a crear pseudo elementos, es decir tenemos que definir en qué parte del documento se añadirán los elementos falsos que utilizaremos. Por ejemplo, si quisiéramos crear un pseudo elemento que se encuentre antes de un **div,** identificado como "ícono", entonces la sintaxis de mi selector quedaría así "#ícono:before".

El segundo aspecto a considerar es la propiedad "content", toda declaración de pseudo elemento debe tener dicha propiedad CSS si quiere que su creación surta efecto, ya que sin ella no podremos insertar nada en los pseudo elementos que se crearon. En pocas palabras, "content", como su nombre lo indica, es la encargada de portar el nuevo contenido.

Se puede presentar el caso en que no se necesite contenido, en estas ocasiones puede utilizar la propiedad "content" vacía, pero por ningún motivo la debe de quitar. Este caso se puede presentar cuando utilizamos cajas vacías a las que únicamente agregamos estilos.

Una vez analizados estos dos aspectos, podemos definir que nuestra sintaxis básica quedará de esta manera:

```
#ícono:before {
```

```
   content: "Hola";
}

#ícono:after {
   content: "libro";
}
```

En CSS 2, los pseudo-elementos son precedidos de un solo carácter de dos puntos. Como pseudo-clases también estaban siguiendo la misma convención, eran indistinguibles. Para solucionar esto, en CSS 2.1 y CSS 3 cambió la convención de pseudo-elementos. Ahora, un pseudo-elemento va precedido de dos caracteres de dos puntos y una pseudo-clase todavía se prefija con un solo dos puntos.

2.5. Modelo de cajas

Si bien el maquetado de una página Web comenzó con el elemento **table**, es decir, se dividía la distribución del contenido en estructuras de celdas de tabla, se presentaban problemas con el manejo del tipo de contenido y la herencia de sus atributos. Por ello, surge un modelo de cajas para la distribución del contenido en diferentes tipos de pantallas.

Para lograr este desafío, el modelo de caja CSS requiere la comprensión de su concepto, y cómo se relaciona con la forma en que se determinan las dimensiones finales de un elemento; será esencial para la comprensión de cómo se coloca un elemento en una página web. El modelo de caja se aplica a los elementos a nivel de bloque, principalmente ** .**

En CSS 2.1, los elementos a nivel de bloque solo pueden ser rectangulares, CSS3 los incorpora de la misma manera. Calculamos las dimensiones generales de un elemento a nivel de bloque, teniendo en cuenta la altura y el ancho del área de contenido, así como los márgenes, rellenos y bordes que se aplican al elemento.

Podemos definir el ancho y altos del contenido de un elemento, al declarar su anchura y altura. Si no se aplican las declaraciones, el valor por defecto para las propiedades width y height es auto.

Para los elementos estáticos (no ubicados) y elementos en posición relativa, en la propiedad de ancho tiene el valor **auto**. El ancho calculado será el ancho del bloque de contención menos los márgenes horizontales, bordes, el relleno y barras de desplazamiento. Es decir, va a ser lo que sobra cuando los márgenes horizontales, bordes, el relleno y barras de desplazamiento (si existen) se han deducido a partir del ancho del bloque de contención.

El bloque de contención es el rectángulo de referencia, cuya posición y dimensiones se utilizan para los cálculos relativos de las posiciones y dimensiones de los elementos descendientes. Aunque los elementos se posicionan con respecto a su bloque de contención, no están confinados por él, y pueden desbordarse. En la mayoría de los casos, las cajas generadas actúan para las cajas descendientes. Los detalles completos de los bloques que contienen se tratan en esta sección.

Para elementos flotantes o con posición absoluta (incluyendo elementos para los cuales se establece la posición fija), una anchura automática hará que la caja generada se reduzca a las dimensiones intrínsecas de su contenido. El juego de cajas es modelado en 3D, en él podemos reconocer su estructura, misma que será modificada por elementos CSS.

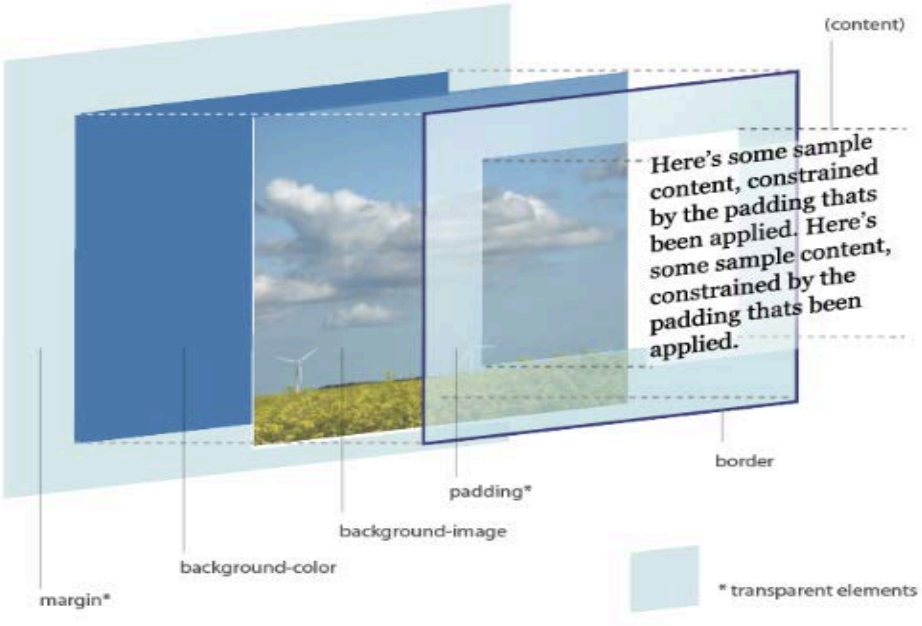


Fig. 1. Modelo de cajas 3D CSS[9].

No importa cómo se coloca el área de contenido, su valor de altura será igual a la altura del contenido si no hay valores que hayan sido declarados para la altura. Una caja está formada por, y en muchas ocasiones sus dimensiones dependerán del contenido HTML: Párrafos e imágenes (**Content**); el espacio libre entre contenido y bordes se llama relleno (**padding**); el **border** es el perímetro de líneas que encierra al relleno y al contenido; se usan imágenes de fondo por detrás del contenido y relleno (**background image**), para ver la imagen deben los elementos superiores contenido y relleno ser transparentes; detrás de **background** image se encuentra el color de fondo (**background color**), y por último es opcional contar con el margen de separación entre cajas dado (**margin)**.

Por lo tanto, para determinar el espacio total necesario para colocar un elemento caja en la página, debemos sumar las dimensiones del área de contenido, relleno, bordes, márgenes e imágenes y color de fondo. Por supuesto, un elemento puede no tener relleno, borde y margen, en cuyo caso sus dimensiones serán dictadas exclusivamente por su contenido. Si una caja tiene imagen y color de fondo, la imagen tiene prioridad para ser mostrada.

Si un elemento contiene solo elementos flotantes o de posición absoluta, que no tendrán ningún contenido en absoluto, su altura será igual a cero.

El modelo de caja se demuestra mejor con un pequeño ejemplo. El cálculo que utilizaremos para determinar el espacio total necesario para dar cabida a un elemento de la página (ignorando el colapso de los márgenes por el momento), será el siguiente:

Ancho total = margen izquierdo + borde izquierdo + relleno izquierdo + ancho +
relleno derecho + borde derecho + margen derecho

Altura total = margen superior + borde superior + relleno superior + altura +
+ margen inferior + borde inferior + margen inferior

en inglés:

Total width = left margin + left border + left padding + width +
right padding + right border + right margin

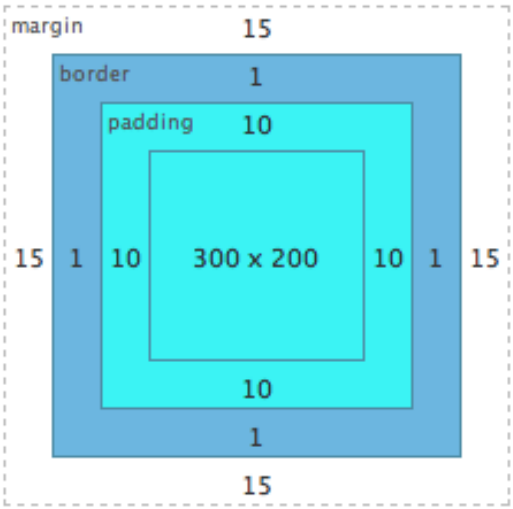
Total height = top margin + top border + top padding + height +
bottom padding + bottom border + bottom margin

Aquí está nuestro ejemplo CSS, un conjunto de reglas que contiene las declaraciones de todas las propiedades del cuadro de un elemento que tiene la clase "box" :

```
.box {
  width: 300px;
  height: 200px;
  padding: 10px;
  border: 1px solid #000;
  margin: 15px;
}
```

El tamaño total del elemento anterior se calcula de la siguiente manera:

```
Total width = 15 + 1 + 10 + 300 + 10 + 1 + 15 = 352px

Total height = 15 + 1 + 10 + 200 + 10 + 1 + 15 = 252px
```



Un punto importante a destacar es que un elemento que tiene su conjunto de anchura del 100 % (es decir, 100 % de la anchura del contenido de su elemento padre) no tiene ningún

margen, el relleno o bordes aplicados, ya que esto lo haría demasiado grande para el espacio que se encuentra en la caja. Esto a menudo es pasado por alto por los diseñadores y pueden perturbar gravemente el diseño de una página, provocando desbordamiento o el empuje de los elementos más allá de lo que debería ser.

La solución, en la mayoría de los casos, es evitar la adición de un valor al ancho de la propiedad (que no sea automático), y aplicar los márgenes, rellenos y bordes solamente. La propiedad **width** de un elemento estático por defecto en automático, e incluso con relleno , bordes y márgenes añadidos; asume la anchura del contenido disponible.

Por supuesto, este enfoque puede no ser factible en algunos casos, como en los casos en que el elemento no es un elemento estático, y requiere la definición de un valor específico anchura (como en el caso de un elemento flotante que no se expande automáticamente). En estos casos, usted tiene dos opciones:

Si el espacio disponible es de ancho fijo, solo tiene que añadir el valor de cada componente en conjunto para asegurarse que coincida con la anchura disponible. Por ejemplo, si el espacio disponible es de 500 píxeles de ancho, y necesita tener 20 píxeles de relleno, simplemente ajusta el ancho de 460px y el relleno a 20 píxeles para ese elemento (20 + 460 + 20 = 500). Esta solución supone que los valores de longitud especificados para las propiedades del cuadro del elemento utilizan la misma unidad de medida, ya que no será capaz de sumar una mezcla de unidades (200px + 10 %, por ejemplo , no tiene sentido en este contexto) .

Cuando el espacio de contenido disponible tiene una anchura desconocida, como en el caso de un esquema, el método no puede ser utilizado como sumas de porcentajes y píxeles. En este caso, la solución es declarar una anchura de 100 % para el elemento en cuestión y aplicar el relleno, el borde y valores de los márgenes a un elemento anidado en su lugar. Ese elemento anidado no tiene declaración de ancho, y puede mostrar el relleno necesario, los bordes y los márgenes sin invadir el elemento principal.

El bloque de contenido CSS comprende las tareas de trazado. Los elementos de caja se colocan dentro de un contexto de formato, que, por defecto, es proporcionado por la caja generada por un elemento primario.

Cuando especificamos las posiciones o dimensiones de los elementos de caja, lo que estamos haciendo con respecto a lo que se conoce como el bloque de contención, es un concepto muy importante en el diseño CSS.

El bloque de contenido para el elemento raíz se llama el bloque de contenido inicial, y tiene las mismas dimensiones que el área de visualización de los medios continuos (como la pantalla) y el área de la página para medios paginados (como **print**). El medio de salida continuo de pantalla gráfica que muestra el documento Web es una ventana rectangular y, el medio de paginado, es la vista en dimensiones para impresión, que incluyen márgenes de página.

El bloque de contención para cualquier otro elemento de caja se determina por el valor de la propiedad de posición para ese elemento. Si el valor de la propiedad position es estático (por defecto) o relativo, el bloque está formado por el borde de la caja de contenido del elemento ancestro más cercano, cuya indicación del valor de la propiedad es uno de:

- **block**
- **inline-block**
- **list-item**
- **run-in**
- **tabla**
- **table-cell**

Si el valor de la propiedad de posición es absoluta, el bloque que contiene es el más cercano antepasado posicionado, es decir, el antepasado más cercano cuya propiedad de posición tiene uno de los valores absolutos, fijo o relativo. El bloque de contención es formado por el borde de relleno ancestro.

Si el valor de la propiedad de posición es fija, el bloque de contenido es el del cuadro de la página (para medios paginados) o la ventana (para papel continuo).

Tenga en cuenta que aunque las posiciones y dimensiones se dan con respecto al bloque que contiene, una caja descendiente no está limitada por su bloque de contención; puede desbordarse.

Vamos a explorar exactamente cuáles son las consecuencias del colapso de los márgenes y cómo afectarán a los elementos de la página.

La especificación W3C define el colapso de los márgenes de la siguiente manera[10]:

"En esta descripción, la expresión colapso de los márgenes significa que los márgenes adyacentes de dos o más cajas se combinan para formar un solo margen ".

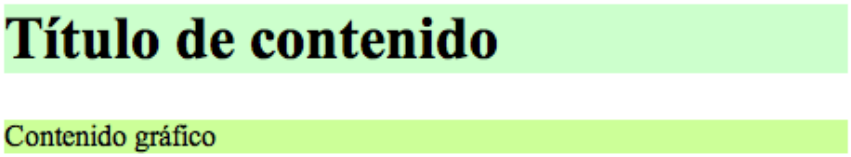
En términos simples, esta definición indica que cuando los márgenes verticales de dos elementos están en contacto, solo el margen del elemento con el valor de mayor margen será respetado, mientras que el margen del elemento con el valor de menor margen se derrumbó a cero. En el caso de que uno de los elementos tenga un margen negativo, se añaden los valores de los márgenes en conjunto para determinar el valor final. Si ambos son negativos, se utiliza el mayor valor negativo. Esta definición se aplica a los elementos adyacentes y los elementos anidados.

Si, por ejemplo, uno de los elementos tiene un margen inferior de 25px y el elemento inmediatamente debajo de él tiene un margen superior de 20 píxeles, solo se aplicará el margen inferior de 25px, y los elementos se mantendrán a una distancia de 25px entre sí. No serán 45px (25+20) de separación, como era de esperar.

Este comportamiento se demuestra mejor con un pequeño ejemplo. Considere el siguiente código separado por 25px:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


 <link href="misestilos.css" rel="stylesheet" type="text/css">
</head>


<body>

<h1>Título de contenido</h1>
<div>
  <p>Contenido gráfico</p>
</div>

</body>
</html>
```

```
@charset "UTF-8";
/* CSS CSS archivo misestilos.css */

h1 {
  margin: 0 0 25px 0;
  background: #cfc;
}
p {
  margin: 20px 0 0 0;
  background: #cf9;
}
```

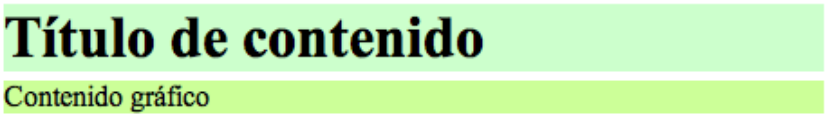


Como podrá ver en la figura, la diferencia entre los elementos es solo 25px, y el margen más pequeño se ha derrumbado a cero. Si en el ejemplo anterior los elementos tenían márgenes iguales (por ejemplo, 20 píxeles cada uno), la distancia entre ellos sería solo 20px.

No existe una situación que provoque una ligera desviación del comportamiento de los márgenes: si uno de los elementos tiene un margen superior o inferior negativo, se añadirán los márgenes positivos y negativos juntos para alcanzar el margen final. He aquí un ejemplo de hoja de estilo que demuestra el concepto:

```
@charset "UTF-8";
/* CSS CSS archivo misestilos.css */
```

```
h1 {
  margin: 0 0 25px 0;
  background: #cfc;
}
p {
  margin: -20px 0 0 0;
  background: #cf9;
}
```



El margen inferior del elemento h1 es un número positivo (25px), y el margen superior del elemento **p** es un número negativo (-20px). En esta situación, se añaden los dos números para calcular el margen final: 25px + (-20px) = 5px.

Si el resultado de este cálculo es un número negativo, este valor tendrá el efecto de un elemento de superposición. Se podría decir que el margen negativo tira del elemento en la dirección opuesta a la de un margen positivo. Vea margen para más detalles sobre los márgenes negativos.

En seguida estudiamos las propiedades que nos permiten controlar la altura y la anchura de las cajas.

height (propiedad de dimensiones CSS)

Esta propiedad establece el alto de un bloque de contenido. Esta altura no incluye el relleno, bordes o márgenes.

`height: { length | percentage | auto | inherit } ;`

Si el bloque de contenido requiere más espacio vertical del que se produjo por la altura que se asignó, su comportamiento se define por la propiedad de desbordamiento (overflow).

Modelo: esta regla de estilo asigna una altura fija de 100 píxeles a los párrafos en el elemento con ID "texto1":

```
#texto p {
  height: 100px;
}
```

La propiedad tiene una longitud px, pt, em, etc., en porcentaje o la palabra clave **auto**. Valores de longitud negativos son ilegales.

Los valores de porcentaje se refieren a la altura de bloque que contiene el elemento. Si la altura del bloque de contenido no se especifica explícitamente (es decir, que depende de la altura del contenido), y este elemento no está posicionado absolutamente, el valor porcentual se trata como automático. Un valor de porcentaje también se considera auto en celdas, filas de tabla, y grupos de filas.

El valor **auto** permite que el navegador calcule la altura de contenido de forma automática, en base a otros factores. Para los elementos con posición absoluta, por ejemplo, la altura del contenido se puede calcular sobre la base de los valores de propiedad superiores e inferiores, o de los márgenes superior e inferior, bordes y relleno aplicado al elemento. Si no hay restricciones son impuestas por otras propiedades, al elemento se le permite asumir su altura del contenido "natural", sobre la base de la altura de los contenidos que contiene.

min-height (propiedad CSS dimensiones)

Esta propiedad establece el alto mínimo de un bloque de contenido. Esta altura mínima no incluye relleno, bordes o márgenes.

```
min-height: { length | percentage | inherit } ;
```

La altura de un elemento nunca será menor que la altura mínima especificada, pero se dejan crecer normalmente si el contenido excede el conjunto de altura mínima.

min-height se utiliza por lo general para asegurar que un elemento tiene al menos una altura mínima, incluso si hay contenido presente. Se usa comúnmente en conjunción con un máximo de altura para producir un rango de altura para el elemento en cuestión.

```
#texto {
  min-height: 60px;
  height: 150px;
}
#texto p {
```

```
  min-height: 100px;
}
```

La propiedad tiene una longitud px, pt, em, etc., en porcentaje o la palabra clave **auto**. Valores de longitud negativos son ilegales.

Los valores de porcentaje se refieren a la altura del bloque que contiene el elemento. Si la altura del bloque de contenido no se especifica explícitamente (es decir, que depende de la altura del contenido), y este elemento no está posicionado absolutamente, el valor porcentual se trata como cero.

```
#texto p {
  max-height: 100px;

}
```

max-height (propiedad de dimensiones CSS)

Esta propiedad establece el alto máximo de un bloque de contenido. Esta altura máxima no incluye relleno, bordes o márgenes.

```
max-height: { length | percentage | none | inherit } ;
```

Un elemento que ha definido su propiedad **max-height** nunca será más alto que el valor especificado, incluso si la propiedad height se encuentra establecida a algo más grande. Existe una excepción a esta regla, sin embargo, si el mínimo de altura se especifica con un valor que es mayor que el máximo de altura, la altura del contenedor será el valor más grande, que, en este caso, significa que el valor mínimo de altura será de hecho el que está aplicado.

```
max-height: { length | percentage | none | inherit } ;
```

max-height se utiliza generalmente en conjunción con un mínimo de altura para producir un rango de altura para el elemento del que se trate.

width (propiedad de dimensiones CSS)

Esta propiedad establece el ancho de un bloque de contenido. Esta anchura no incluye relleno, bordes o márgenes.

```
width: { length | percentage | auto | inherit } ;
```

Si el contenido de un bloque requiere más espacio horizontal que la anchura que se asigna, su comportamiento se define por la propiedad de desbordamiento.

```
#texto p {
  width: 100px;

}
```

min-width (propiedad de dimensiones CSS)

Esta propiedad establece el ancho mínimo de un bloque de contenido. Esta anchura mínima no incluye relleno, bordes, márgenes.

```
width: { length | percentage | auto | inherit } ;
```

Un elemento al que se aplica **min-width** nunca será más estrecho que la anchura mínima especificada, pero se deja crecer normalmente si su contenido supera el mínimo de anchura establecido.

min-width se utiliza a menudo en conjunción con un máximo de ancho para producir un rango de anchura para el elemento de que se trate.

```
#texto p {
  min-width: 100px;
}
```

max-width (propiedad de dimensiones CSS)

Esta propiedad establece el ancho máximo de un bloque de contenido. Esta anchura máxima no incluye relleno, bordes, márgenes.

Un elemento al que se aplica un máximo de ancho nunca será mayor que el valor especificado, incluso si la propiedad de ancho se ajusta para que sea más amplia. Existe una excepción a esta regla, sin embargo, si el mínimo de ancho se especifica con un valor mayor que el de **max-width**, la anchura del contenedor será el valor más grande, que en este caso significa que el valor mínimo de ancho será el que se aplica al elemento.

max-width se utiliza a menudo en conjunción con un mínimo de ancho para producir un rango de ancho para el elemento de que se trate.

```
#texto p {
  max-width: 400px;
  min-width: 100px;
}
```

margin-top (propiedad de márgenes CSS)

Esta propiedad define la distancia vertical desde el borde superior de la frontera de un elemento, hasta el borde de su bloque de contenido, o el elemento que es verticalmente adyacente por encima de ella. Su efecto también depende de otros factores, tales como la presencia de colapso de los márgenes en los elementos adyacentes verticales.

`margin-top: { `*length* | *percentage* | `auto | inherit } ;`

Si el elemento por encima del elemento en cuestión es flotante, o de posición absoluta, el margen superior pasará a través del elemento flotante, porque flotadores y elementos absolutos se eliminan del flujo. El margen solo se verá afectado por elementos estáticos (o

elementos para los cuales se establece posición relativa, y que no tienen coordenadas), en el flujo normal del documento, lo que incluye el propio bloque de contenido.

```
#texto p {
  margin-top: 20px;
}
```

margin-right (propiedad de márgenes CSS)

Esta propiedad define la distancia horizontal desde el borde derecho del elemento de que se trate hasta el borde de su bloque de contenido, o el elemento que es horizontalmente adyacente al mismo.

```
margin-right: { length | percentage | auto | inherit } ;
```

Si el elemento a la derecha del elemento en cuestión es flotante, o posición absoluta, el margen pasará a través de él, porque flotadores y elementos absolutos se eliminan del flujo. El margen solo se verá afectado por elementos estáticos (o elementos para los cuales se establece posición relativa, y que no tienen coordenadas), en el flujo normal del documento, lo que incluye el propio bloque de contenido.

```
#texto p {
  margin-right: 20px;
}
```

margin-bottom (propiedad de márgenes CSS)

Esta propiedad define la distancia vertical desde el borde inferior del fondo de la frontera del elemento en cuestión a la orilla de su bloque de contenido, o el elemento que hay debajo verticalmente adyacente. Su efecto es también dependiente de otros factores, tales como la presencia de colapso de los márgenes en los elementos adyacentes verticalmente.

```
margin-bottom: { length | percentage | auto | inherit } ;
#texto p {
```

```
  margin-bottom: 20px;
}
```

margin-left (propiedad de márgenes CSS)

Esta propiedad define la distancia horizontal desde el borde izquierdo del elemento de que se trate, hasta el borde de su bloque de contención, o el elemento que es horizontalmente adyacente al mismo.

```
margin-left: { length | percentage | auto | inherit } ;
```

```
#texto p {
  margin-left: 20px;
}
```

margin (propiedad de márgenes CSS)

En el flujo normal de un documento, los márgenes se utilizan generalmente para controlar el espacio en blanco horizontal y vertical alrededor de los elementos. Usted puede pensar en que un margen tiene el efecto de empujar el elemento lejos de otros elementos de la página.

```
margin: { { length | percentage | auto }  1 a 4 valores
 | inherit } ;
```

padding-top (propiedad de relleno CSS)

La propiedad **padding-top** establece el relleno de un elemento por el lado superior con el valor especificado. El relleno es el área que se encuentra entre las fronteras de un elemento y su contenido. Cualquier imagen de fondo o color de fondo que se aplica al elemento se extenderá a través del área de relleno.

```
padding-top: { length | percentage | inherit } ;
```

Cuando se utiliza relleno vertical (padding-top y el relleno de fondo padding-bottom) puede causar la superposición de los elementos.

```
#texto p {
  padding-top: 2em;
}
```

padding-right (propiedad de relleno CSS)

La propiedad **padding-right** ajusta el relleno para el lado derecho de un elemento, utilizando el valor especificado.

```
padding-right: { length | percentage | inherit } ;
```

```
#texto p {
  padding-right: 2em;
}
```

padding-bottom (propiedad de relleno CSS)

La propiedad **padding-bottom** establece el relleno en el lado inferior de un elemento, con el valor especificado.

```
padding-bottom: { length | percentage | inherit } ;
```

```
#texto p {
  padding-bottom: 2em;
}
```

padding-left (propiedad de relleno CSS)

La propiedad padding-left ajusta el relleno para el lado izquierdo de un elemento utilizando el valor especificado.

```
padding-left: { length | percentage | inherit } ;
```

```
#texto p {
  padding-left: 2em;
```

```
}
```

padding (propiedad de relleno CSS)

La propiedad de relleno **padding** es una forma abreviada de establecer el relleno de los cuatro lados de un elemento, con el valor o los valores especificados. Cada lado puede tener su propio valor.

```
padding: { { length | percentage } 1 to 4 values | inherit } ;
```

```
#texto p {
  padding: 2px 4px 6px 8px;
}
```

border (propiedad de bordes CSS)

Propiedades fronterizas que permiten al diseñador controlar el borde de una caja -la zona entre el relleno y los márgenes-.

Los bordes son propiedades de esquema que permiten controlar el contorno de una caja. El contorno se dibuja generalmente fuera de la zona fronteriza, pero no ocupa ningún espacio como lo hacen las fronteras.

border-top-color

Establece el color del borde superior de un elemento

```
#texto p {
  border-top-color: #000;
  border-top-style: solid;
}
```

border-top-style

Establece el estilo del borde superior de un elemento

```
#texto p {
  border-top-style: solid;
}
```

border-top-width

Establece el ancho del borde superior de un elemento

```
#texto p {
  border-top-width: 2px;
  border-top-style: solid;
}
```

border-top

Establece el ancho, color y tamaño para el borde superior de un elemento

```
#texto p {
  border-top: 2px solid red;
}
```

border-right-color

Establece el color del borde derecho de un elemento

```
#texto p {
  border-right-color: blue;
  border-right-style: solid;
}
```

boder-right-style

Establece el estilo del borde derecho de un elemento

```
#texto p {
  border-right-style: solid;
}
```

border-right-width

Establece el ancho del borde derecho de un elemento

```
#texto p {
  border-right-width: 2px;
  border-right-style: solid;
}
```

border-right

Establece el ancho, color y estilo del borde derecho de un elemento

```
#texto p {
  border-right: 2px solid red;
}
```

border-bottom-color

establece el color del borde inferior de un elemento

```
#texto p {
  border-bottom-color: #000;
  border-bottom-style: solid;
```

```
}
```

border-bottom-style

Establece el estilo del borde inferior de un elemento

```
#texto p {
  border-bottom-style: solid;
}
```

border-bottom-width

Establece el ancho del borde inferior de un elemento

```
#texto p {
  border-bottom-width: 2px;
  border-bottom-style: solid;
}
```

border-bottom

Establece el ancho, color y tamaño en el borde inferior de un elemento

```
#texto p {
  border-bottom: 2px solid blue;
}
```

border-left-color

Establece el color del borde izquierdo de un elemento

```
#texto p {
  border-left-color: green;
  border-left-style: solid;
}
```

border-left-style

Establece el estilo del borde izquierdo de un elemento

```
#texto p {
  border-left-style: solid;
}
```

border-left-width

Establece el ancho del borde izquierdo de un elemento

```
#texto p {
  border-left-width: 2px;
  border-left-style: solid;
}
```

border-left

Establece el ancho, color y tamaño del borde izquierdo de un elemento

```
#texto p {
  border-left: 2px solid green;
}
```

border-color

Establece el color del borde de los cuatro bordes de un elemento

```
#texto p {
  border-color: red blue green;
  border-style: solid;
}
```

border-style

Establece el estilo de los cuatro bordes de un elemento

```
#texto p {
  border-style: solid dotted dashed;
}
```

border-width

Establece el ancho de los cuatro bordes de un elemento

```
#texto p {
  border-width: 2px 4px;
  border-style: solid;
}
```

border

Establece el ancho, color y tamaño para los cuatro bordes de un elemento

```
#texto p {
  border: 2px solid red;
}
```

outline-color

Establece el color de un marco

```
#texto a:focus {
  outline-color: red;
  outline-style: solid;
}
```

outline- style

Establece el estilo de un marco

```
#texto a:focus {
  outline-style: solid;
}
```

outline-width

Establece el ancho de un marco

```
#texto a:focus {
  outline-width: 2px;
  outline-style: solid;
}
```

outline

Propiedad abreviada que establece el marco de un elemento

```
#texto a:focus {
  outline: 2px solid red;
}
```

En HTML5 las propiedades CSS3 permiten llevar un paso más allá el diseño, el siguiente código crea una ventana cuadrada:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">

</head>

<title> Margenes especiales</title>
 <link href="misestilos.css" rel="stylesheet" type="text/css">
<body>
<header id="principal">
<span id="titulo"> estilos web ventana cuadrada</span>
</header>


</body>
</html>
```

```
@charset "UTF-8";
/* CSS CSS archivo misestilos.css */

body {
	text-align:center
}
#principal{
	display:block;
	width:500px;
	margin:50px auto;
	padding:15px;
	text-align:center;
	border: 1px #FFC;
	background:#6F9;


}
#titulo{
	font:bold 36px Verdana, Geneva, sans-serif;
```

```
}
```

estilos web ventana cuadrada

border-radius (propiedad de esquinas redondeadas CSS)

CSS3 cada vez más, incorpora nuevas alternativas de diseño, por ejemplo la propiedad **border-radius**, disponible para Safari y Firefox. Este elemento opera con dos valores, el primero afecta las esquinas en su radio horizontal y el segundo las esquinas en su radio vertical

```
-webkit-border-radius: 15px 12px;  /* Safari  */
-moz-border-radius: 15px 12px;     /* Firefox */
```

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">

</head>

<title> Margenes especiales</title>
 <link href="misestilos.css" rel="stylesheet" type="text/css">
<body>
<header id="principal">
<span id="titulo"> estilos web radius</span>
</header>

</body>
</html>
```

```
@charset "UTF-8";
/* CSS CSS archivo misestilos.css */

body {
       text-align:center
}
#principal{
       display:block;
       width:500px;
       margin:50px auto;
       padding:15px;
       text-align:center;
       border: 1px #FFC;
```

```
        background:#6F9;

        -moz-border-radius: 22px;
        -webkit-border-radius: 22px;
       border-radius: 20xp /8xp;
}
#titulo{
        font:bold 36px Verdana, Geneva, sans-serif;
}
```

estilos web radius

box-shadow (propiedad de ventanas sombreadas CSS)

Ahora introducimos el efecto de sombras para las propiedades de las cajas con la propiedad **box-shadow**, especificada por tres valores, el primero es generado por la función **rgb();** los siguientes dos valores expresados en **px** establecen el desplazamiento de la sombra.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">

</head>

<title> Margenes especiales</title>
 <link href="misestilos.css" rel="stylesheet" type="text/css">
<body>
<header id="principal">
<span id="titulo"> estilos web radius shadow</span>
</header>

</body>
</html>
```

```
@charset "UTF-8";
/* CSS CSS archivo misestilos.css */

body {
      text-align:center
}
#principal{
      display:block;
      width:500px;
      margin:50px auto;
      padding:15px;
      text-align:center;
      border: 1px #FFC;
      background:#6F9;

      -moz-border-radius: 22px;
      -webkit-border-radius: 22px;
          border-radius: 20xp /8xp;
```

```
      -moz-box-shadow:rgb(150,150,150) 5px 5px;
      -webkit-box-shadow: rgb(150,150,150) 5px 5px;
      box-shadow: rgb(150,150,150) 5px 5px;
}
#titulo{
      font:bold 36px Verdana, Geneva, sans-serif;
}
```

Podemos emplear la palabra clave inset para referir una sombra de profundidad superior, pruebe el siguiente código:

```
-moz-box-shadow:rgb(150,150,150) 5px 5px inset;
-webkit-box-shadow: rgb(150,150,150) 5px 5px inset;
 box-shadow: rgb(150,150,150) 5px 5px inset;
```

text-shadow (propiedad CSS que produce sombras sobre el texto)

La propiedad **text-shadow** es aplicada a cajas para producir un efecto de sombra en el texto. Se declara el color de sombra con **rgb()** y las distancias de profundidad de la sombra. Por favor experimente con el siguiente código. [11]

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">

</head>

<title> Margenes especiales</title>
 <link href="misestilos.css" rel="stylesheet" type="text/css">
<body>
<header id="principal">
<span id="titulo"> estilos web radius, sombra de texto y ventana</span>
</header>

</body>
</html>
```

```
@charset "UTF-8";
/* CSS CSS archivo misestilos.css */

body {
      text-align:center
}
#principal{
      display:block;
```

```
      width:500px;
      margin:50px auto;
      padding:15px;
      text-align:center;
      border: 1px #FFC;
      background:#6F9;

      -moz-border-radius: 22px;
      -webkit-border-radius: 22px;
       border-radius: 20xp /8xp;



      -moz-box-shadow:rgb(150,150,150) 5px 5px 10px;
      -webkit-box-shadow: rgb(150,150,150) 5px 5px 10px;
      box-shadow: rgb(150,150,150) 5px 5px 10px;
}
#titulo{
      font:bold 36px Verdana, Geneva, sans-serif;
      text-shadow:  rgb(0,0,150) 3px 3px 5px;
}
```

@font-face (propiedad CSS que introduce un tipo particular de letra)

En un principio la Web tenía el problema de solo emplear los tipos de letra que estaban presentes del lado del cliente, la propiedad @font-face permite adjuntar el archivo de las fuentes específicas necesarias para mostrar el documento HTML tal como lo diseñamos.

Visite los sitios http://www.creamundo.com, http://www.moorstation.org/typoasis/ o http://www.dafont.com/es/ para descargar tipos de letra en archivos .ttf. Para el siguiente ejemplo descargamos el archivo de tipo de letra Arabella.ttf, el cual debe estar en el mismo dominio que nuestro documento HTML.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">

</head>

<title> Márgenes especiales</title>
 <link href="misestilos.css" rel="stylesheet" type="text/css">
<body>
<header id="principal">
<span id="título"> estilos web tipo de letra</span>
</header>
```

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

```
@charset "UTF-8";
/* CSS CSS archivo misestilos.css */

body {
      text-align:center
}
#principal{
      display:block;
      width:500px;
      margin:50px auto;
      padding:15px;
      text-align:center;
      border: 1px #FFC;
      background:#6F9;

      -moz-border-radius: 22px;
      -webkit-border-radius: 22px;
       border-radius: 20xp /8xp;

      -moz-box-shadow:rgb(150,150,150) 5px 5px 10px;
      -webkit-box-shadow: rgb(150,150,150) 5px 5px 10px;
      box-shadow: rgb(150,150,150) 5px 5px 10px;
}
#titulo {
      font:bold 36px MiFuente, verdana, sans-serif;
      text-shadow:  rgb(0,0,150) 3px 3px 5px;
}
@font-face {
      font-family: 'MiFuente';
      src: url('Arabella.ttf');
}
```

`linear-gradient` (propiedad CSS que aplica efecto de gradiente al color de fondo)

El gradiente de color es aplicado a los fondos de las cajas, la función **linear-gradient()** tiene tres valores, el primero es la posición, en seguida color inicial y el color final. Las variantes podrían ser estas:

linear-gradient(15deg, #C93, #C30); /* degradado a 15 grados*/

linear-gradient(top, #C93, #C30); /* degradado desde la parte superior*

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">
```

```
</head>

<title> Margenes especiales</title>
 <link href="misestilos.css" rel="stylesheet" type="text/css">
<body>
<header id="principal">
<span id="titulo"> estilos web fondos color en gradiente</span>
</header>

</body>
</html>
```

```
@charset "UTF-8";
/* CSS CSS archivo misestilos.css */

body {
      text-align:center
}
#principal{
      display:block;
      width:500px;
      margin:50px auto;
      padding:15px;
      text-align:center;
      border: 1px #FFC;
      background:#FC3;


      -moz-border-radius: 22px;
      -webkit-border-radius: 22px;
       border-radius: 20xp /8xp;



      -moz-box-shadow:rgb(150,150,150) 5px 5px 10px;
      -webkit-box-shadow: rgb(150,150,150) 5px 5px 10px;
      box-shadow: rgb(150,150,150) 5px 5px 10px;

      background: -webkit-linear-gradient(15deg, #C93, #C30);
      background: -moz-linear-gradient(15deg, #C93, #C30);
}

#titulo {
      font:bold 36px MiFuente, verdana, sans-serif;
      text-shadow:  rgb(0,0,150) 3px 3px 5px;
}
@font-face {
      font-family: 'MiFuente';
      src: url('Arabella.ttf');
}
```

rgba (función CSS que aplica color y transparencias)

Esta función agrega un valor de transparencia aportando mayor calidad gráfica. En este ejemplo el valor de trasparencia es de 0.5.

```
<!DOCTYPE html>
```

```
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">

</head>

<title> Margenes especiales</title>
 <link href="misestilos.css" rel="stylesheet" type="text/css">
<body>
<header id="principal">
<span id="titulo"> estilos web  sombreado con transparencia</span>
</header>

</body>
</html>
```

```
@charset "UTF-8";
/* CSS CSS archivo misestilos.css */

body {
      text-align:center
}
#principal{
      display:block;
      width:500px;
      margin:50px auto;
      padding:15px;
      text-align:center;
      border: 1px #FFC;
      background:#FC3;


      -moz-border-radius: 22px;
      -webkit-border-radius: 22px;
       border-radius: 20xp /8xp;



      -moz-box-shadow:rgb(150,150,150) 5px 5px 10px;
      -webkit-box-shadow: rgb(150,150,150) 5px 5px 10px;
      box-shadow: rgb(150,150,150) 5px 5px 10px;

      background: -webkit-linear-gradient(top, #C93, #C30);
      background: -moz-linear-gradient(top, #C93, #C30);
}

#titulo {
      font:bold 36px MiFuente, verdana, sans-serif;
      text-shadow:  rgba(0,0,0,0.5) 3px 3px 5px;
}
@font-face {
      font-family: 'MiFuente';
      src: url('Arabella.ttf');
}
```

Hsla (función CSS que aplica color y opacidad)

La función **hsla()** recibe cuatro valores, el primero es el color de tiente en una rueda de 360 grados; el segundo es el valor de saturación entre 0 y 100%, el tercer valor es la luminosidad entre 0 y 100%; y el último valor representa la opacidad.

```
@charset "UTF-8";
/* CSS CSS archivo misestilos.css */

body {
      text-align:center
}
#principal{
      display:block;
      width:500px;
      margin:50px auto;
      padding:15px;
      text-align:center;
      border: 1px #FFC;
      background:#FC3;


      -moz-border-radius: 22px;
      -webkit-border-radius: 22px;
       border-radius: 20xp /8xp;



      -moz-box-shadow:rgb(150,150,150) 5px 5px 10px;
      -webkit-box-shadow: rgb(150,150,150) 5px 5px 10px;
      box-shadow: rgb(150,150,150) 5px 5px 10px;

      background: -webkit-linear-gradient(top, #C93, #C30);
      background: -moz-linear-gradient(top, #C93, #C30);
}

#titulo {
      font:bold 36px MiFuente, verdana, sans-serif;
      text-shadow:  rgba(0,0,0,0.5) 3px 3px 5px;
      color: hsla(120,100%, 20%,0.1);
}
@font-face {
      font-family: 'MiFuente';
      src: url('Arabella.ttf');
}
```

outline (propiedad CSS que aplica un segundo borde)

Con la propiedad **outline** agregamos un segundo borde a nuestra caja, de tamaño en **px**, y con desplazamiento **outline-offset** en **px**.

```
@charset "UTF-8";
/* CSS CSS archivo misestilos.css */

body {
	text-align:center
}
#principal{
	display:block;
	width:500px;
	margin:50px auto;
	padding:15px;
	text-align:center;
	border: 1px #FFC;
	background:#FC3;


	-moz-border-radius: 22px;
	-webkit-border-radius: 22px;
	 border-radius: 20xp /8xp;



	-moz-box-shadow:rgb(150,150,150) 5px 5px 10px;
	-webkit-box-shadow: rgb(150,150,150) 5px 5px 10px;
	box-shadow: rgb(150,150,150) 5px 5px 10px;

	background: -webkit-linear-gradient(top, #C93, #C30);
	background: -moz-linear-gradient(top, #C93, #C30);

	outline:1px dashed #FF0000;
	outline-offset: 15px;
}

#titulo {
	font:bold 36px MiFuente, verdana, sans-serif;
	text-shadow:  rgba(0,0,0,0.5) 3px 3px 5px;
	color: hsla(120,10%, 90%,0.1);
}
@font-face {
	font-family: 'MiFuente';
	src: url('Arabella.ttf');
}
```

border-image (propiedad CSS que aplica una imagen como borde)

La propiedad **border-image** emplea una imagen como patrón para construir el borde de un elemento.

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

```
  <meta name="description" content="ejemplo adjuntar CSS">

</head>

<title> Márgenes especiales</title>
 <link href="misestilos.css" rel="stylesheet" type="text/css">
<body>
<header id="principal">
<span id="título"> CONALEP</span>
</header>

</body>
</html>
```

```
@charset "UTF-8";
/* CSS CSS archivo misestilos.css */

body {
      text-align:center
}
#principal{
      display:block;
      width:300px;
      margin:50px auto;
      padding:15px;
      text-align:center;
      border: 1px #FFC;
      background:#FC3;


      border: 29px;
      -moz-border-image: url("manzana.jpg") 29 stretch;
      -webkit-border-image: url("manzana.jpg") 29 stretch;
      border-image: url("manzana.jpg") 29 stretch;
}

#titulo {
      font:bold 36px MiFuente, verdana, sans-serif;
      text-shadow:  rgba(0,0,0,0.5) 3px 3px 5px;
      color: hsla(120,10%, 90%,0.4);
}
@font-face {
      font-family: 'MiFuente';
      src: url('Arabella.ttf');
}
```

La cosa más importante que usted puede hacer para mejorar el rendimiento de su sitio Web es optimizar las imágenes. Históricamente, la cuestión de las funciones del tamaño de los archivos de imagen es una decisión de ingeniería, por lo que el peso de la página es extremadamente importante para el tiempo de respuesta global.

La optimización de imágenes comienza con una decisión cualitativa sobre el número de colores, resolución, o la precisión requerida para una imagen dada. Estos cambios implican pérdida general de calidad. La imagen puede tener menos colores, o en el caso del formato

JPEG, codificación menos detallada. Aunque el 60% a 70% de calidad es el estándar aceptado para JPEG, algunas imágenes o contextos pueden requerir más o menos calidad. Por ejemplo, las imágenes brillantes de las celebridades pueden requerir un tamaño de archivo más grande que tablas generadas automáticamente o miniaturas pequeñas. Estas decisiones son decisiones creativas y deben ser realizadas por el diseñador, utilizando herramientas tales como la función **Guardar** para la Web, en Photoshop.[12] Los formatos de imágenes más comunes para la web son GIF, JPEG y PNG.

2.6. Funciones CSS

Transformaciones: rotate (función CSS que rota el elemento html)

La función rotate modifica la orientación de nuestra caja en ciertos grados de inclinación.

```
@charset "UTF-8";
/* CSS CSS archivo misestilos.css */

body {
      text-align:center
}
#principal{
      display:block;
      width:300px;
      margin:50px auto;
      padding:15px;
      text-align:center;
      border: 1px #FFC;
      background:#FC3;


      border: 29px;
      -moz-border-image: url("manzana.jpg") 29 stretch;
      -webkit-border-image: url("manzana.jpg") 29 stretch;
      border-image: url("manzana.jpg") 29 stretch;


transform: rotate(30deg);
-ms-transform: rotate(30deg); /* IE 9 */
-webkit-transform: rotate(30deg); /* Safari and Chrome */

}

#titulo {
      font:bold 36px MiFuente, verdana, sans-serif;
      text-shadow:  rgba(0,0,0,0.5) 3px 3px 5px;
      color: hsla(120,10%, 90%,0.4);
}
@font-face {
      font-family: 'MiFuente';
      src: url('Arabella.ttf');
```

```
}
```

Transformaciones: skew (función CSS que cambia la simetría del elemento html)

```
@charset "UTF-8";
/* CSS CSS archivo misestilos.css */

body {
      text-align:center
}
#principal{
      display:block;
      width:300px;
      margin:50px auto;
      padding:15px;
      text-align:center;
      border: 1px #FFC;
      background:#FC3;


      border: 29px;
      -moz-border-image: url("manzana.jpg") 29 stretch;
      -webkit-border-image: url("manzana.jpg") 29 stretch;
      border-image: url("manzana.jpg") 29 stretch;


transform: skew(30deg,20deg);
-ms-transform: skew(30deg,20deg); /* IE 9 */
-webkit-transform: skew(30deg,20deg); /* Safari and Chrome */

}

#titulo {
      font:bold 36px MiFuente, verdana, sans-serif;
      text-shadow:  rgba(0,0,0,0.5) 3px 3px 5px;
      color: hsla(120,10%, 90%,0.4);
}
@font-face {
      font-family: 'MiFuente';
      src: url('Arabella.ttf');
}
```

Transformaciones: translate (función CSS que cambia la posición del elemento html en la pantalla)

```
@charset "UTF-8";
/* CSS CSS archivo misestilos.css */

body {
      text-align:center
}
```

```
#principal{
      display:block;
      width:300px;
      margin:50px auto;
      padding:15px;
      text-align:center;
      border: 1px #FFC;
      background:#FC3;


      border: 29px;
      -moz-border-image: url("manzana.jpg") 29 stretch;
      -webkit-border-image: url("manzana.jpg") 29 stretch;
      border-image: url("manzana.jpg") 29 stretch;


transform: translate(50px,100px);
-ms-transform: translate(50px,100px); /* IE 9 */
-webkit-transform: translate(50px,100px); /* Safari and Chrome */

}

#titulo {
      font:bold 36px MiFuente, verdana, sans-serif;
      text-shadow:  rgba(0,0,0,0.5) 3px 3px 5px;
      color: hsla(120,10%, 90%,0.4);
}
@font-face {
      font-family: 'MiFuente';
      src: url('Arabella.ttf');
}
```

Hemos mencionado que para un buen diseño cuando se trata de un sitio Web de gran número de páginas, lo correcto es agregar las funciones de estilos CSS3 en su totalidad desde un archivo adjunto, esto nos simplificará en mucho las modificaciones necesarias para estar actualizado el diseño del conjunto de nuestra Web. Por ejemplo, ingrese a la URL http://www.csszengarden.com/?cssfile=/214/214.css&page=2, y observe cómo el contenido es invariante y el diseño cambia radicalmente.

Hay muchos diseñadores que al construir un diseño de un sitio web de tres columnas estándar, eligen un par de colores para él. No se molestan llevando su diseño más lejos, o ajustando los detalles. Tal vez no hay tiempo ni dinero en el presupuesto del proyecto para hacer esto, o tal vez ha tomado el axioma un tanto literalmente. No todos los sitios web tienen que ser bellos, pero cada sitio web lo puede ser. CSS ha dado a los diseñadores web una gran cantidad de control sobre la apariencia de un sitio, pero creemos que el verdadero problema es que muchas personas simplemente no están seguras por dónde empezar

cuando se trata de la personalización estetica. Tiene que ver con ese proceso de llevar su diseño un paso más allá con la ayuda de texturas[13].

Otro factor importante, es por ejemplo trabajar para un cliente en una actualización de la página web de su empresa, tome en cuenta el contexto del color institucional de su imagen, este criterio es fundamental para el manejo de color y texturas que integrará con CSS a sus documentos HTML5[14].

Otra parte esencial es agregar dinamismo por programación, eso se logra empleando Javascript, por ejemplo, cual es el tema del capítulo siguiente donde se explorán sus aspectos básicos.

URL`S

http://reference.sitepoint.com/css/pseudoclass-nthlastchild

http://www.wextensible.com/temas/xhtml-css/css-descripciones.html#selector

http://untitled.es/category/manual-css3-2/

http://librosweb.es/css_avanzado/capitulo_3/pseudo_clases.html

http://bigbangdiseñoweb.com/font-css/

http://reference.sitepoint.com/css/demos

http://bigbangdiseñoweb.com/propiedad-display-modelo-de-cajas/

http://librosweb.es/css/capitulo_2/selectores_basicos.html

https://developer.mozilla.org/es/docs/CSS

http://bigbangdiseñoweb.com/html5/

https://developer.mozilla.org/en-US/docs/Web/CSS/::first-letter

http://serescritor.com

http://www.minkbooks.com

http://librosweb.es/javascript/

http://simon.html5.org/html-elements

http://www.w3c.es/Divulgacion/GuiasReferencia/XHTML1/

http://reference.sitepoint.com/css/moz-border-radius

http://librosweb.es/css_avanzado/capitulo_1/esquinas_redondeadas.html

Referencias

[1] W3C CSS2.1. Syntax and basic data types. Consulta: 26 de agosto de 2013, de http://www.w3.org/TR/CSS21/syndata.html#q10
[2] W3C (2011) Cascading Style Sheets (CSS) Snapshot 2010. Consulta: 26 de agosto de 2013, de http://www.w3.org/TR/CSS/#css1
[3]W3C (2001) Introducción to CSS3. http://www.w3.org/TR/2001/WD-css3-roadmap-20010523/
[4]W3C (2011) Selector level 3. Consulta: 26 de agosto de 2013, de http://www.w3.org/TR/2011/REC-css3-selectors-20110929/
[5] Clark Richard & Studholme Oli (2012) Beginning HTML5 and CSS3. New York: Springer
[6] Power David (2012) Beginning CSS3. Apress
[7] Paganotti Sandro (2013) Designning next generation web projects with CSS3. Mumbai: Packt
[8] Studholme Oli and Clark (2012) Beginning HTM5 and CSS3. New York: Springer
[9] http://www.hicksdesign.co.uk/boxmodel/
[10] http://www.w3.org/TR/CSS21/box.html#collapsing-margins
[11] Niederst Robbins Jennifer (2012) Learning web design. O'REILLY
[12] Souder Steve (2009) Even faster Web sites. O'Reilly
[13] Jason Beaird (2010) The principles of beautiful web design. Kelly Steele
[14] Hogan Brian (2009) Web design for developers. Pragmatic bookshelf

Capítulo III: Javascript

3.1. JavaScript (JS)

El estudio de los diversos aspectos de la historia de la informática es más que un ejercicio intelectual, muestra cómo llegamos nosotros a nuestra condición actual, indica enfoques eficaces, así como los errores del pasado, y proporciona la perspectiva y visión de futuro, y en un sentido más seguro, el cómo llegar allí.

Los usuarios, por error, piensan en el diseño Web de una manera muy simplista, cuando en realidad intervienen consideraciones de comunicación relevantes en cuanto a los mensajes, menús, hojas de estilo, programación y bases de datos[1]. Usted tiene una meta, tiene cierta desiderata que se combina en una función de utilidad no lineal, para ello debemos tener una visión de las limitaciones que presupone la forma contextual, y una de ellas es clave, el poder de comunicación con hipervínculos. La Web no siempre es tecnología, pero siempre hay una restricción, el poder de expresar significado dentro de una interacción entre contenido y soporte. Nos basamos en un árbol de diseño de las decisiones posibles para una página Web, el cual hace que para esas diversas decisiones, el trabajo de su diseño sea hacia abajo del árbol, por un proceso de capas desde el lenguaje natural, pasando por capas de estilo y programación[2].

En la anatomía de una página Web, aunque estamos familiarizados con su utilidad, a veces no vemos que en realidad es un montaje de archivos diferentes: documento **.html**; hojas de estilos **.css**; gráficos jpg, gif, png; vídeo mp4; audio mp3; programación JavaScript **.js**; entre los principales. El documento HTML presenta el contenido en lenguaje natural, son elementos de texto como títulos, encabezados, párrafos, texto resaltado, enlaces, etiquetas de menús, datos de autoría., entre muchos otros. Es importante destacar que hay varias versiones de HTML, las más utilizadas son HTML 4.01 y HTML5, sin embargo en ellas hay cierto enfoque sobre lo pretendido como concepto Web[3].

Con el desarrollo de Internet en la década de los 90's, surge la necesidad de páginas Web dinámicas que apliquen interactividad para múltiples propósitos; en este contexto, Brendan Eich de la empresa *Netscape Communications Corporation* inventa un lenguaje de programación interpretado y ejecutado por navegadores (Livescript), es decir, sin necesidad

de programación del lado del servidor. Ahora la versión moderna de este lenguaje es conocida como JavaScript (JS), nos referimos a esa filosofía orientada a objetos y basada en el paradigma imperativo. Imperativo refiere al modo en que un programa está en función de un estado que cambia usando variables y sentencias; alcanza gran aceptación en los programadores debido a que incorpora una programación basada en prototipos (objetos que heredan propiedades de otros objetos)[4]. En **JS**, un objeto es un conjunto de propiedades, en otras palabras, pares de indicadores y valores asociados, todo objeto JavaScript tiene oculta una propiedad que se llama prototipo[5]. Gracias a la ECMA (European Computer Manufacturers Association), organismo que estandariza el ECMAScript, se logra que sea portable y multiplataforma en el empleo de rutinas JS; además de las funciones elementales, en JS se agregan las interfaces de programación de aplicaciones por su siglas en inglés **API**. Las API acceden a los objetos presentes en un documento HTML, esto se logra gracias al DOM (Document Object Model), modelo que hace posible estructurar y jerarquizar los objetos de una página Web para aplicar HTML y CSS dinámicos[6]. Las API son interfaces de las librerías de los navegadores que complejizan y hacen más fácil la programación para los documentos HTML. Al igual que CSS, se puede declarar JS dentro del HTML o en un archivo adjunto; nosotros lo haremos de ambas maneras en HTML5, aunque se recomienda ampliamente que sea un archivo adjunto.

3.2. Las entrañas de JavaScript (JS)

Toda programación en algún lenguaje artificial, requiere **interpretación**, sin ella su trabajo resultará frustrante. **JS** incluye un juego de funciones, escritura suelta, objetos dinámicos y una semántica con notación literal para expresar los objetos; además de una compleja programación de variables globales. El acto de procesar el código se expresa como correr o ejecutar el programa, normalmente las instrucciones o lo que se denomina código se procesan a partir de la línea de arriba hacia abajo y las instrucciones están dadas en inglés natural; sin embargo, hay ocasiones en que es posible cambiar el flujo de ejecución o incluso saltar sobre alguna sección de código en particular[7].

Las funciones **JS** son objetos de primera clase del ámbito léxico. **JS** es el primer lenguaje **lambda** de corriente principal[8]. El cálculo lambda[9] es el marco teórico matemático para las **funciones computables**, escrito por Alonzo Church y Stephen Kleene en 1930. Una función computable refiere al poder expresivo de los lenguajes de programación, es decir, nos dice que una función puede calcularse mediante algún algoritmo, sin importar su forma de calcularla. Una función computable en su interior está formada por **funciones iniciales** que por su simplificada forma son computables; al combinarse estas funcione iniciales dan origen a **funciones originales**, y si un lenguaje de programación abarca todas estas funciones originales, decimos que el lenguaje posee todo el poder de cómputo posible[10]. El **cálculo lambda** es el lenguaje más sencillo y universal de programación que puede ser definido por algoritmos calculados por máquinas de Turing[11]. La funciones JS son funciones lambda anónimas que permiten asignar funciones a variables y hacer referencias a ellas empleando las variables, además, se pueden pasar funciones como parámetros a otras funciones y crear funciones como resultado de otras funciones computables. Esto hace que JS sea un lenguaje donde todo en él, es un objeto.

Las expresiones lambda, se basan en un concepto matemático de ya casi 80 años: **el cálculo lambda**. Para comprender el cálculo lambda sigamos el siguiente razonamiento:

> Sea la función identidad $f(x) = x$, que toma un único argumento, **x**, e inmediatamente devuelve **x**. Además, la función suma $f(x,y) = x + y$, toma dos argumentos, **x** e **y**, y devuelve la suma de ambos: **x + y**.
> Usando estas dos funciones como ejemplo, es posible hacer algunas observaciones útiles acerca de varias ideas fundamentales del cálculo lambda: la primera observación es que las funciones no necesitan ser explícitamente nombradas. Esto es, la función **f(x,y) = x + y** puede ser reescrita como una función anónima: **(x,y) → x + y** (que se lee: «el par de **x** e **y** se opera a **x + y**»). Del mismo modo, **f(x) = x** puede ser reescrita de forma anónima como **x → x**, que se lee: «el argumento **x** se opera a sí mismo».
>
> La segunda observación es que el nombre que se asigne a los argumentos de la función, es generalmente irrelevante. Esto es, **x → x** e **y → y** expresan la misma

función: **la función identidad**. Del mismo modo, **x,y → x + y** y **u,v → u + v** expresan la misma función: la función suma de dos números.

Una tercera observación es que toda función que requiere dos argumentos, como por ejemplo la función suma, puede ser reescrita como una función que acepta un único argumento, pero que devuelve otra función, la cual a su vez acepta un único argumento. Por ejemplo, **x,y → x + y** puede ser reescrita como **x → (y → x + y)**. Esta transformación se conoce como **currificación**, y puede generalizarse para funciones que aceptan cualquier número de argumentos. Esto puede parecer oscuro, pero se entiende mejor mediante un ejemplo. Considérese la función suma no currificada: **x,y → x + y,** al tomar a los números 4 y 6 como argumentos, se obtiene: 4 + 6 lo cual es igual a 10. Considérese ahora la versión currificada de la función: x → (y → x + y), si se toma al número 4 como argumento **x**, se obtiene la función: **y → 4 + y**, tomando luego al número 6 como argumento **y**, se obtiene: 4 + 6 lo cual es igual a 10. De modo que la versión currificada devuelve el mismo resultado que la versión no currificada. En el cálculo lambda, todas las expresiones representan funciones anónimas de un solo argumento[12].

En el cálculo lambda, las funciones están definidas por expresiones lambda, que dicen qué se hace con su argumento. Tome en cuenta que una función es un conjunto de instrucciones agrupadas y reutilizables.

JS es un lenguaje de programación interpretado, bajo el estándar ECMA-262[13] y desarrollado por Sun Microsystems. En el fondo, JS tiene más en común con **Lisp** que con **Java**. **Lisp** es el primer lenguaje multiparadigma basado en el cálculo lambda. Esto hace que JavaScript sea un lenguaje muy poderoso. Sin embargo, Java y JS no están relacionados por el hecho de poseer propósitos diferentes y distinta semántica. JS es de hecho para programación del lado del cliente.

La moda en la mayoría de los lenguajes de programación hoy en día exige la tipificación estricta. La teoría es que la tipificación estricta permite a un compilador detectar un gran número de errores en tiempo de compilación. El tipado fuerte implica que el lenguaje de

programación impone restricciones a la combinación del los tipos de datos, para garantizar que se eviten errores de escritura y con ello, se logre un comportamiento eficiente en tiempo de ejecución.

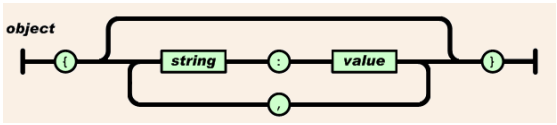
Por contrario, JavaScript es un lenguaje débilmente **tipado**, por lo que los compiladores de JavaScript son incapaces de detectar errores de tipo. Esto puede ser alarmante para las personas que vienen a JavaScript desde lenguajes fuertemente tipados como C++ o Visual Basic o Java. JavaScript tiene una muy poderosa notación literal de objetos. Los objetos pueden ser creados simplemente enumerando sus componentes. Esta notación se llama JSON, y se basa en dos estructuras[14]:

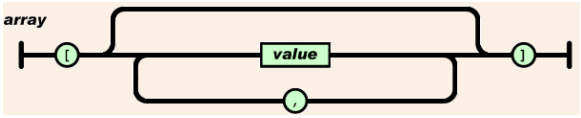
- Una colección de pares nombre-valor, son objetos que comienza con llave izquierda { y terminan con llave derecha }, cada nombre es seguido por dos puntos (:) y los pares nombre-valor están separados por comas (,).
- Una lista ordenada de valores, se trata de estructuras de datos universales, como vectores, arreglos, listas o secuencias.

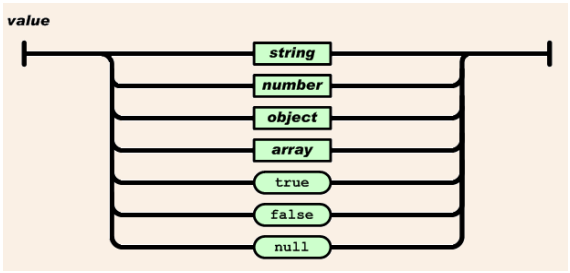
JavaScript Object Notation (JSON pronunciado "yeison") es un formato de intercambio de datos ligero. Se basa en el objeto de notación literal, esta es una de las mejores partes de JavaScript. Puede ser utilizado para el intercambio de datos entre los programas escritos en otro lenguaje de programación moderna. Se trata de un formato de texto, por lo que es legible por humanos y máquinas. Es fácil de aplicar y fácil de utilizar. Hay una gran cantidad de material sobre JSON en http://www.json.org/json-es.html .

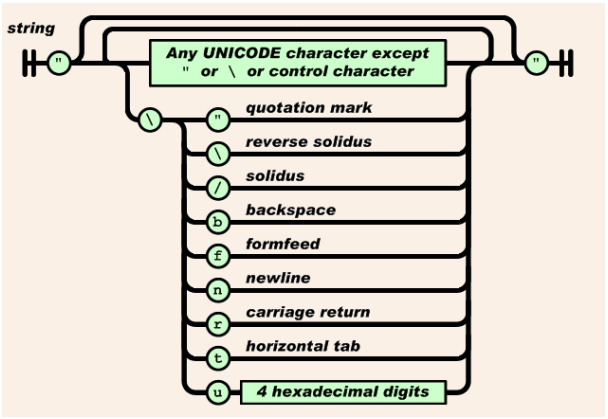
JSON tiene seis tipos de valores: objetos, matrices, cadenas, números, booleanos (verdadero y falso), y el valor especial nulo. Los espacios en blanco (espacios, tabuladores, retornos de carro y caracteres de nueva línea) se puede insertar antes o después de cualquier valor. Esto puede hacer que los textos JSON sean más fáciles de leer para las personas. Los espacios en blanco se pueden omitir para reducir los costos de transmisión o almacenamiento.

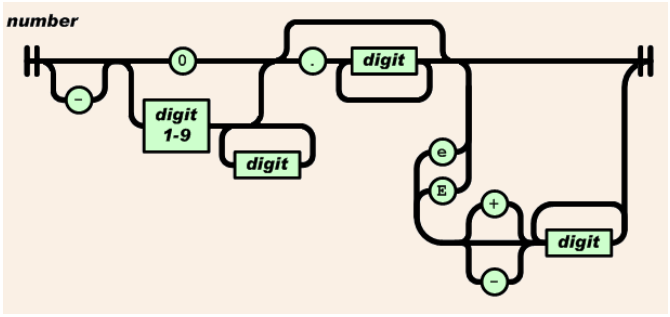
Los siguientes diagramas representan la gramática JSON para los diferentes elementos que componen a JS; para saber cómo se escribe correctamente, solo hay que seguir de izquierda a derecha por algún camino, uniendo los elementos descritos con círculos y recuadros.











Un aspecto controvertido en JavaScript es la herencia de prototipos. JS tiene un sistema de objetos de clase libre en el que los objetos heredan las propiedades directamente de otros objetos. Esto es muy poderoso, pero no es familiar para los programadores de formación

clásica, algunos de los que no se tomarán el tiempo para explorar las entrañas de JS podrían sentirse frustrados. Pero si Usted aprende a trabajar con la naturaleza prototípica de JS, será un placer su programación. Considere que en JS las variables globales son fundamentales.

El estándar que define JavaScript (también conocido como JScript) es la edición 5.1 del estándar ECMAScript. El lenguaje de programación, disponible en http://www.ecma-international.org/publications/files/ecma-st/ECMA-262.pdf, es un subconjunto propio de ECMAScript; es un lenguaje de muchos contrastes, por un lado, la Web se ha convertido en una importante plataforma para el desarrollo de aplicaciones y JavaScript es el único idioma que se encuentra en todos los navegadores, JavaScript es realmente bueno, es ligero y expresivo.

Tome en cuenta además que, si dispone de un navegador Web y un editor de texto, ya tendrá todo lo necesario para ejecutar programas JS. Una vez creado un archivo HTML, en el mismo directorio agregaremos un archivo de programa JS para ser invocado dentro de nuestro documento.

Los navegadores de Internet pueden bloquear a JavaScript, para ello HTML incluye la etiqueta **<noscript>,** para alertar al usuario sobre este hecho, el siguiente ejemplo muestra su aplicación:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">
<link rel="stylesheet"  href="misestilos.css" >
<script type="text/javascript" src="funciones.js"></script>

<noscript>
  <p>Nuestro libro favorito</p>
  <p>Se encuentra inactivo JavaScript.
Por favor vuelve a activarlo.</p>
</noscript>

</head>
 <body>
```

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

Existen tres formas de insertar JS en nuestros documentos HTML, pero igual que para CSS3 lo recomendable es usar la inclusión de archivos adjuntos para HTML5.

Modo en línea

Este modo se apoya en los manejadores para eventos onClick, onMouseOver u onMouseOut que, una vez activados, ejecutan código JS contenido en ellos.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
<script type = "text/javascript">
</script>

<noscript>
  <p>Nuestro libro favorito</p>
  <p>Se encuentra inactivo JavaScript.
Por favor vuelve a activarlo.</p>
</noscript>

 </head>
 <body>

<p onClick="alert('hizo click')">Hacer click</p>

</body>
</html>
```

Modo incrustado

El código JavaScript se debe insertar entre las etiquetas <script>...</script>, esto nos organiza unidades de compilación en HMTL5, no es necesario type = "text/javascript. Este recurso provoca que el documento sea más repetitivo y difícil de reutilizar.

Método externo

Para reducir los tiempos de descarga, incrementar nuestra productividad y reusar código, es necesario tener como criterio de diseño, adjuntar archivos externos .js, con ayuda del atributo src.

<script src="codigoprog.js">

Ya hemos dicho que, es JavaScript un lenguaje interpretado en lugar de un lenguaje compilado. **¿Qué se entiende por los términos interpretado y compilado?**

Cuando se necesita algo para interpretar el código JavaScript y convertirlo en algo que se entienda por la máquina, se trata de un lenguaje interpretado. Los ordenadores entienden solo código de máquina, que es esencialmente una cadena de números binarios (es decir, una cadena de ceros y unos). La función del navegador es ser un programa especial llamado un intérprete, que convierte el código JavaScript a código máquina en el equipo que atiende. Por ejemplo, es un poco como tener un traductor que dice en inglés lo que dice un documento en español. Un punto importante a destacar es que la conversión de JavaScript sucede en el momento en que el código se ejecuta, tiene que ser repetido cada vez que esto sucede. JavaScript no es el único lenguaje interpretado, hay otros, incluyendo VBScript.

El lenguaje compilado alternativo, es uno en el que el código del programa se convierte al código máquina antes de que en realidad esté en ejecución, y esta conversión se tiene que completar y solo hacer una vez. El programador utiliza un compilador para convertir el código que escribió, en código máquina, y el código máquina se ejecuta por parte del usuario del programa. Los lenguajes compilados incluyen por ejemplo a Visual Basic y C++. A menos que cambie el documento, usted puede utilizarlo sin retraducción.

Tal vez este es un buen punto para disipar un mito generalizado: JavaScript no es la versión de secuencia de comandos script del lenguaje Java. De hecho, a pesar de que comparten el mismo nombre, eso es prácticamente todo lo que los vincula. Particularmente la buena noticia es que JavaScript es mucho, mucho más fácil de aprender y usar que Java.

Usted verá en este libro el código JavaScript que se ejecuta dentro de una página Web por parte de un navegador. Ya hemos dicho que todo lo que necesita para crear estas páginas Web es un editor de texto - por ejemplo, el bloc de notas de Windows y un navegador Web, como Firefox o Safari, con el que se puedan ver las páginas. Estos navegadores vienen equipados con intérpretes de JavaScript.

De hecho, el lenguaje JavaScript llegó a estar disponible en el navegador Web Netscape Navigator 2. Inicialmente, se llamaba LiveScript. Sin embargo, debido a que la tecnología Java era dinámica en el tiempo, Netscape decidió que JavaScript sonaba más emocionante. Cuando JavaScript despegó, Microsoft decidió añadir a su propia marca JavaScript, llamado JScript para Internet Explorer. Desde entonces, Netscape, Microsoft y otros han lanzado versiones mejoradas y las han incluido en sus últimas versiones de navegadores. A pesar de que estas diferentes marcas y versiones de JavaScript tienen mucho en común, no hay suficientes diferencias como para causar problemas si se tiene cuidado. Inicialmente, Usted va a crear un código que va a trabajar con la mayoría de los navegadores, ya sea Firefox , Internet Explorer o Safari.

¿Por qué a veces denominamos JavaScript a ECMAScript?. La European Computer Manufacturers Association (ECMA) es una organización privada que desarrolla estándares de los sistemas de información y comunicación. Una de las normas que controlan es JavaScript, lo que ellos llaman ECMAScript. Controlan varios aspectos del lenguaje y ayudan a asegurar que las diferentes versiones de JavaScript sean compatibles. Sin embargo, mientras que el ECMA establece normas para el lenguaje real, también especifica cómo se utiliza en determinados ordenadores.

La organización que establece los estándares para las páginas Web es el Consorcio World Wide Web (W3C). Ellos no solo establecen estándares para HTML, XHTML, y XML, sino también el cómo interactúan con JavaScript las páginas Web dentro de un navegador Web.

En principio, Usted está estudiando aquí lo esencial de JavaScript antes que las cosas más avanzadas. La mayoría de las páginas Web que contengan código JavaScript en este libro

pueden ser almacenadas en su disco duro y se cargan directamente en su navegador desde el disco duro.

Básicamente del lado del servidor Web, el trabajo es tener muchas páginas Web en el disco duro para ser accesibles. Cuando un navegador, por lo general en un equipo diferente, solicita una página Web que figura en ese servidor Web, el servidor Web lo carga de su propio disco duro y luego pasa la página Web de nuevo a la computadora solicitante, a través de un servicio de comunicaciones especiales llamado Hypertext Transfer Protocol (http, Protocolo de transferencia de hipertexto). El equipo que ejecuta el navegador Web que hace la solicitud, se conoce como cliente.

Piense en la relación cliente/servidor como un poco la de un cliente en una relación comercial. El cliente entra en una tienda y le dice: "Dame una de esas". El tendero sirve al cliente al alcanzar el artículo solicitado y ponerlo al alcance del cliente. En una situación en la Web, al equipo cliente que ejecuta el navegador Web (una tableta, teléfono o PC), el servidor Web le proporciona la página solicitada, como lo hace el dueño de la tienda.

3.3. Gramática JS

Para ilustrar la gramática JS vamos a representarla como en líneas atrás con ayuda de diagramas JSON (JavaScript Object Notation). Las reglas para la interpretación de estos diagramas son simples:

- Se comienza en el borde izquierdo y se siguen las pistas hasta el borde derecho.
- A medida que avanza, se encontrará con literales en óvalos y las reglas o descripciones en rectángulos.
- Cualquier secuencia que se puede hacer siguiendo las pistas es permitida en JS.
- Cualquier secuencia que no se puede hacer siguiendo las pistas no es permitida.
- Los diagramas JSON con una barra en cada extremo permiten espacios en blanco que se insertan entre cualquier par de símbolos.

El modelo de la unidad de compilación dentro de un documento HTML5, su modelo es:

```
<script ...>
```

```
 Código JavaScript
</script>
```

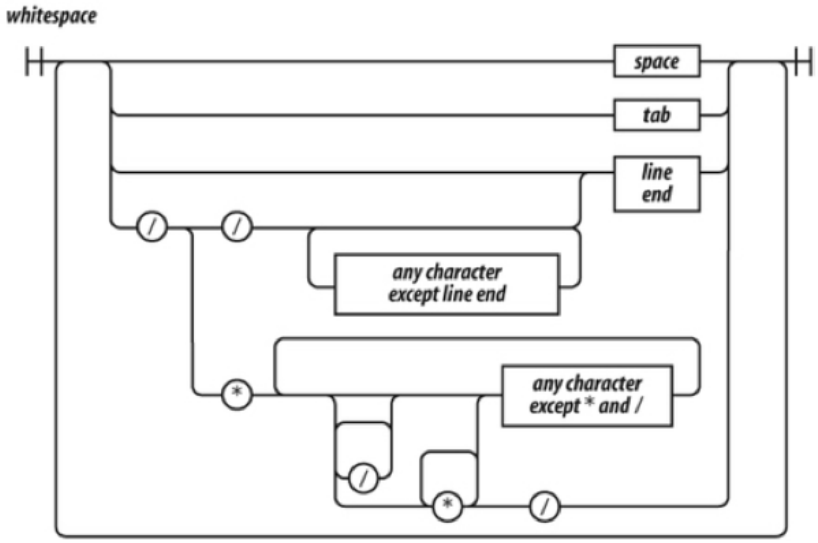
Los espacios en blanco

El espacio en blanco puede tomar la forma de caracteres de formato o comentarios. El espacio en blanco es normalmente insignificante, pero a veces es necesario usar espacios en blanco para separar las secuencias de caracteres que, de otro modo, pueden combinarse en un solo símbolo. Por ejemplo, en:

```
var tiempo = fecha;
```

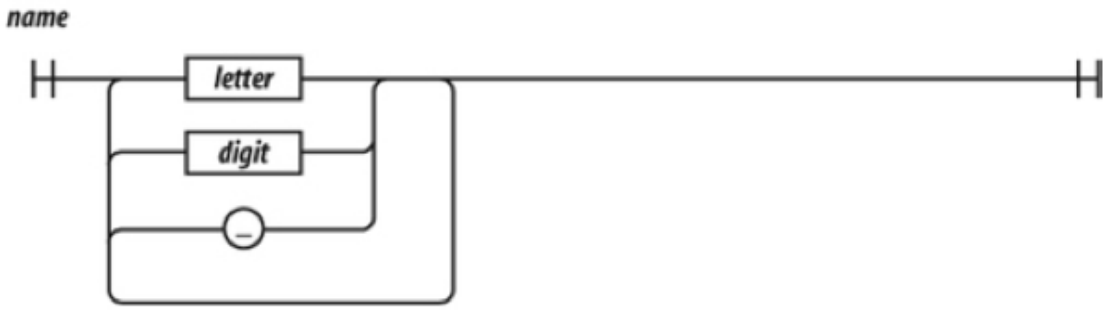
El espacio entre **var** y **tiempo** no puede ser eliminado, pero los otros espacios se pueden quitar. JS ofrece dos tipos de comentarios, comentarios de bloques formados con / *…* / y los comentarios de fin de línea que comienzan con //... Los comentarios deben ser usados libremente para mejorar la legibilidad de los programas.

```
<!doctype html>
<html>
<head>
<meta charset="UTF-8">


<script type="text/javascript">
/* La etiqueta de código JS, son los elementos HTML5 script */
document.write("Texto desplegado por JS");

//document.write escribe el texto en nuestro html
</script>
<title>Programa Javascript</title>
</head>

<body>
<p> nuestro primer programa </p>

</body>
</html>
```



JavaScript no obliga a terminar cada sentencia con el caracter de punto y coma (;), pero es recomendable hacerlo por disciplina. En este ejemplo, **document.write()** abre una ventana para mostrar un mensaje de texto.

Nombre



Un nombre es una letra seguida opcionalmente por una o más letras, dígitos o guiones de subrayado. Un nombre no puede ser una de estas palabras reservadas:

```
break; case; catch; continue; default; delete; do; else;
finally; for; function; if; in; instanceof; new; return;
switch; this; throw; try; typeof; var; void; while; with.

abstract; boolean; byte; char; class; const; debugger;
double; enum; export; extends; final; float; goto;
```

```
implements; import; int; interface; long; native; package;
private; protected; public; short; static; super;
synchronized; throws; transient; volatile; const; export.
```

No se permite el uso de una palabra reservada como el nombre de una propiedad de objeto en un objeto literal. Los nombres se utilizan para las declaraciones, variables, parámetros, nombres de propiedades, operadores y etiquetas.

Números

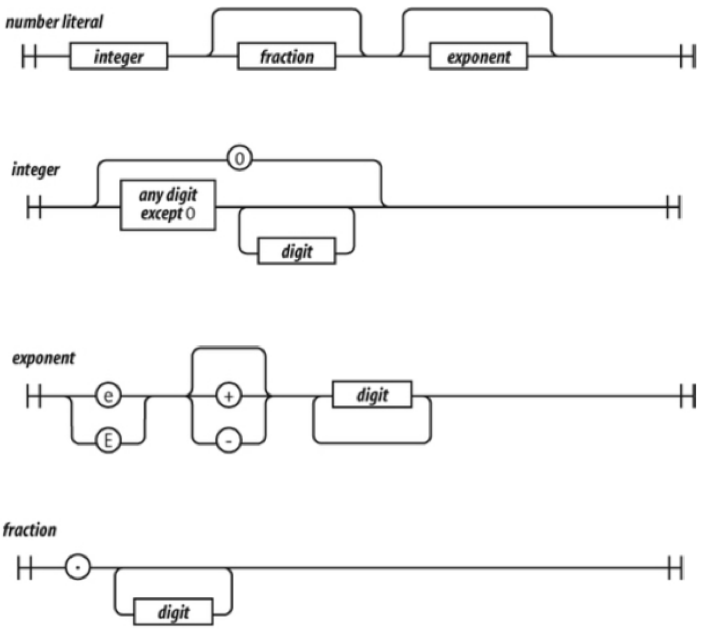
Una de las características más fundamentales de un lenguaje de programación es el conjunto de tipos de datos que soporta. Estos son el tipo de valores que se pueden representar y manipular en un lenguaje de programación.

JavaScript permite trabajar con tres tipos de datos primitivos:

- Los números, por ejemplo: 41, 69,7, etc.
- Las cadenas de texto, por ejemplo: “Palabras cruzando el puente”, etc.
- Boolean, por ejemplo, verdadero o falso (true o false)

JavaScript también define dos tipos de datos triviales, nulo e indefinido, cada uno de los cuales define un valor único. Además de estos tipos de datos primitivos, JavaScript soporta un tipo de datos compuesto conocido como **objeto**. Se verá con más detalle más adelante.

Nota: todos los números en JavaScript se representan como valores de punto flotante. JavaScript representa números con el formato de punto flotante de 64 bits definido por el estándar IEEE 754.

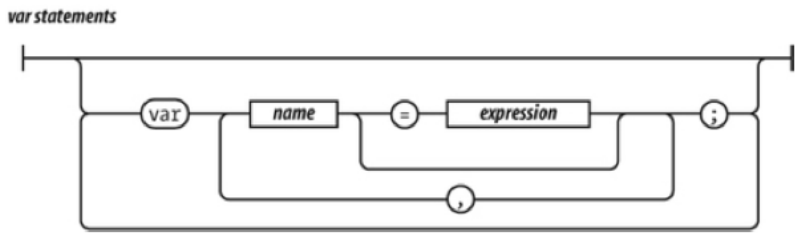


Variables

Al igual que muchos otros lenguajes de programación, JavaScript tiene variables. Las variables pueden ser consideradas como contenedores con nombre. Usted puede colocar datos en estos contenedores y luego referirse a los datos simplemente nombrando el contenedor.

Antes de utilizar una variable en un programa JavaScript, debe declararla. Las variables se declaran con la palabra clave **var,** de la siguiente manera:

```
<script type="text/javascript">
<!--
var Nacionalidad, estado, ciudad;
var nombre;
//-->
</script>
```



Una unidad de compilación contiene un conjunto de instrucciones ejecutables. En los navegadores Web, cada etiqueta **<script>** entrega una unidad de compilación que se ejecuta inmediatamente. A falta de un enlazador, JavaScript lanza a todas las unidades compiladas juntas un espacio de nombres global y común, es decir, variables globales; por otro lado, cuando se utiliza dentro de una función, la sentencia **var** define variables privadas de la función.

Almacenar un valor en una variable se denomina inicialización de la variable. Usted puede hacer la inicialización de variables en el momento de creación de variables o en un punto en el tiempo cuando se necesita esa variable, de la siguiente manera:

```
<script type="text/javascript">
<!--
var nombre = "CONALEP";
var dinero;
dinero= 20000.07;
//-->
</script>
```

Nota: se debe usar la palabra clave **var** solo para la declaración o inicialización de la vida de cualquier nombre de variable en un documento, no debe declarar la misma variable dos veces.

Una variable de JavaScript puede contener un valor de cualquier tipo de datos. A diferencia de muchos otros lenguajes, Usted no tiene que decirle a JavaScript, en la declaración de variables, el tipo de valor de la variable que se mantendrá. El tipo de valor de una variable puede cambiar durante la ejecución de un programa y JavaScript se encarga de ello automáticamente.

En matemáticas, se emplean las variables para generalizar, mediante el uso de ecuaciones y fórmulas, programas que generan un campo de soluciones aritméticas útiles. El modelo para una asignación de valor a una variable es:

```
variablex=5
/*inicialización de variable identificada como     variablex*/
```

```
variabley=9
resultado = variablex - variabley
/*resultado es creado por JS, al no estar declarado, como variable
global*/
```

JavaScript tiene ámbito de las variables, es decir, el alcance de una variable es la región del programa en el que se define. Las variables de JavaScript solo pueden tener dos ámbitos:

Variables Globales: una variable global tiene un alcance total al documento, que significa que se define en todas partes en su código JavaScript.

Variables locales: una variable local será visible solo dentro de una función que se define. Los parámetros de la función son siempre locales a esa función.

Dentro del cuerpo de una función, una variable local tiene prioridad sobre una variable global con el mismo nombre. Si se declara una variable local o parámetro de la función con el mismo nombre que una variable global, se oculta con eficacia la variable global. Usted no debe usar palabras reservadas **JS** como nombres de variable; y no deben comenzar con un número (0-9). Deben empezar con una letra o el caracter de subrayado. Por ejemplo, 123test es un nombre de variable no válido, pero _123test es válido. JavaScript diferencia para los nombres de variables entre mayúsculas y minúsculas.

Nota: el tipo de variable es dado por el tipo de valor asignado, por ejemplo un número.

```
var iva = 15;   //variable tipo entero
var iva = 8.9; //variable tipo decimal
var nombre = "Juan Sin Miedo"; /* variable tipo string o cadena
emplea comillas simples o dobles. Se pueden emplear \n (avance de
línea), \t (tabulador), \" (comilla) o \\ (barra invertida) */
var grupo1 = ["texto","palabra","frase","párrafo","documento"];
/*variable arreglo, colección de variables, grupo1 almacena los
valores y son accesibles por su posición de índice que comienza en
cero*/
```

```
var literatura = grupo1[0];
// la variable recupera el valor en el índice cero del arreglo
grupo1*/
var finado = false; //variable tipo booleana
```

El operador = es un símbolo de asignación de valor , el operador == compara si los dos miembros izquierdo y derecho del operador son iguales.

3.4. Funciones

Mientras que las variables almacenan la información, las funciones procesan esa información. Todas las funciones tienen la forma **functionName().** A veces hay algo entre paréntesis y, a veces, no lo hay. Ya has visto una función**, document.write()**, que es de las muchas funciones integradas a JavaScript la cual escribe lo que se encuentra entre los paréntesis en la página Web. Antes de la inmersión en las funciones de datos, vamos a hablar de dos funciones interesantes, solo para adentrarnos en el cómo se trabaja con funciones.

alert()

Una función muy útil es **alert(),** que devuelve una cadena de texto en un pequeño anuncio dentro de una caja (también llamado un cuadro de alerta).

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">

<link rel="stylesheet"  href="misestilos.css" >
<script type="text/javascript" src="funciones.js"></script>

 </head>
 <body>
```

```
<script>
alert("Describe la función alert()!"); // JS  ejecuta la función alert()
// show me -->
</script>

<noscript>
  <p>Nuestro libro favorito</p>
  <p>Se encuentra inactivo JavaScript.
Por favor vuelve a activarlo.</p>
</noscript>
 </body>
</html>
```

La función **alert()** es útil para notificar al usuario sobre cómo solucionar problemas cuando JavaScript no está funcionando correctamente, además puede devolver códigos de error.

prompt()

Otra función incorporada que le será muy útil es **prompt(),** que solicita a sus usuarios información y luego establece una variable igual al tipo del dato recibido.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
<script type = "text/javascript">

var the_name = prompt("Cuál es su nombre?", "ingrese aquí ");
// show me -->
</script>

<noscript>
  <p>Nuestro libro favorito</p>
  <p>Se encuentra inactivo JavaScript.
```

```
Por favor vuelve a activarlo.</p>
</noscript>

 </head>
 <body>

<h1>Estimado
<script type = "text/javascript">
<!-- hide me from older browsers
document.write(the_name);
// show me -->
</script>
,</h1>
Gracias por registrar su nombre.

</body>
</html>
```

Las palabras dentro de los paréntesis de funciones se llaman parámetros, la función **document.write ()** requiere un parámetro: una cadena a escribir en su página Web. El prompt() toma dos parámetros: una cadena que describe la caja y una cadena dentro de la caja. Los parámetros son el único aspecto de una función que se puede controlar, son el medio para proporcionar a la función la información que necesita para hacer su trabajo. Con una función **prompt()**, por ejemplo, no se puede cambiar el color de la caja, el número de botones que tiene, o cualquier otra cosa, es un cuadro de diálogo predefinido. Solo puede cambiar los parámetros que están permitidos específicamente, de requerir algo distinto tendrá que construir sus propias funciones.

Cuando requiera escribir la fecha en su página, pueden solicitar a **JS** que tome la hora local de la computadora del usuario.

New Date()

```
var ahora = new Date();
```

La primera parte de esta línea, **var ahora=**, debería resultar familiar. Se establece la variable

ahora a algún valor. La segunda parte, **new Date ()**, crea un objeto, los objetos son almacenas de datos que pueden guardar múltiples piezas de información, tales como el momento particular en el tiempo. Por ejemplo, en JavaScript necesita un objeto para describir las 15:30 horas del viernes 7 de enero de 2012, en Morelia. Esto se debe a que requiere muchos bits de información diferentes: el tiempo, el día, el mes, la fecha, y el año, así como una representación (en relación con el meridiano de Greenwich) de la hora local del usuario. Como se puede imaginar, trabajar con un objeto es un poco más complicado que trabajar con un número o una cadena.

Dado que las fechas son tan ricas en información, JavaScript tiene un objeto constructor de la fecha con sus detalles. Cuando desee la fecha actual y la hora del usuario, se utiliza **new Date ()** para decirle JavaScript que cree un objeto **Date** con toda la información correcta. Ahora que JavaScript ha creado su objeto **Dato**, vamos a extraer la información del mismo utilizando una función de fecha de JavaScript. Para extraer el año actual, utilice la función **getYea**r:

```
var ahora = new Date();
var Elaño = ahora.getYear();
```

En el código anterior, la variable ahora es un objeto **Date**, y la función **getYear()** es un método del objeto **Date**. Los métodos son simplemente funciones que están incorporadas en los objetos. Por ejemplo, **getYear ()** se integra en el objeto **Date** y obtiene el año del objeto. Dado que la función es parte del objeto Date, recibe el nombre de método. Para utilizar **getYear()** para obtener el año de la fecha almacenada en la variable, ahora Usted escribiría:

```
ahora.getYear();
```

Otros métodos comunes al objeto Date() son:

getDate()	El día del mes como un número entero de 1 a 31
getDay ()	El día de la semana como un número entero, donde 0 es el domingo y el 1 es el lunes
getHour ()	La hora como un número entero entre 0 y 23
getMinutes ()	Los minutos como un número entero entre 0 y 59

getMonth ()	El mes como un número entero entre 0 y 11, donde 0 es enero y 11 es diciembre
getSeconds ()	El segundo como un número entero entre 0 y 59
getTime ()	La hora actual en milisegundos, donde 0 es 1 de enero de 1970, 00:00:00
getYear ()	El año, pero este formato se diferencia de un navegador a otro.

Ahora vamos a poner todo esto junto, escribiendo código para la fecha y la hora. Para empezar el día, el mes y el año, utilizando los métodos getDate (), getMonth (), y getYear (). Para obtener la hora y en minutos, utilizamos getHours () y getMinutes ().

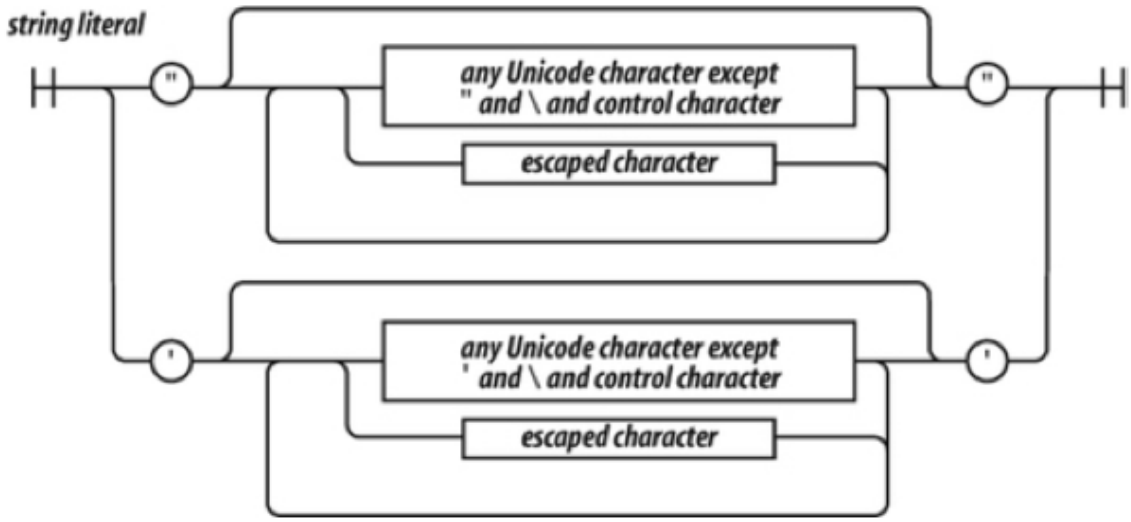
La figura 2-12 muestra el código completo para escribir la fecha y la hora (sin segundos) a una página Web.

```
<html>
<head><title>Mi libro favorito de javascript</title>
<script type = "text/javascript">

var date = new Date();
// obtener los datos de salida del objeto Date
//
var month = date.getMonth();
var day = date.getDate();
var year = date.getYear();
var hour = date.getHours();
var minutes = date.getMinutes();
month = month + 1; // enero es mes 0
// corregir el error Y2K
//
year = fixY2K(year);
// fijar los minutos, añadiendo un 0 delante si es inferior a 10
//
minutes = fixTime(minutes);
// crea la cadena date
//
var date_string = month + "/" + day + "/" + year;
var time_string = hour + ":" + minutes;
var date_time_string = "Hoy es " + date_string + ". El tiempo es ahora " +
time_string + ".";
```

```
// Esta es la función Y2K, no te preocupes acerca de cómo funciona esto

function fixY2K(number) {
if (number < 1000) {
number = number + 1900;
}
return number;
}
// Esta es la función que corrige la hora - No te preocupes acerca de cómo funciona
esto tampoco.
function fixTime(number) {
if (number < 10) {
number = "0" + number;
}
return number;
}
// mostrar -->
</script>
</head>
<body>
<h1>Bienvenidos a JavaScript!</h1>
<script type = "text/javascript">

document.write(date_time_string);
// mostrar -->
</script>
</body>
</html>
```

3.5. Cadenas

Una cadena literal (**string**) puede ser envuelta en comillas simples o dobles. Puede contener cero o más caracteres. La barra invertida (\) es el caracter de escape. JavaScript se construyó en la época de Unicode, un conjunto de caracteres de 16 bits, por lo que todos los caracteres de JavaScript son de 16 bits de ancho.



3.6. Expresiones

Las expresiones u operandos más simples son un valor literal (como una cadena o un número), una variable, un valor integrado (true, false, null, undefined, NaN o Infinity). Javascript aporta los siguientes tipos de operadores para crear expresiones más complejas:

- Operadores aritméticos
- Operadores de comparación
- Operadores lógicos (o relacionales)
- Operadores de asignación
- Los operadores de bits y el operador ternario.

Operadores aritméticos

Operador	Descripción	Ejemplo
+	Suma dos operandos.	A + B
-	Resta el segundo operando al primero.	A - B
*	Multiplica ambos operandos	A * B
/	Divide numerador entre denominador.	B / A
%	Operador módulo, devuelve	B % A

	resto de la división.	
++	Aumenta un numero entero en uno.	A++
--	Decrementa un entero en uno.	A--

Operadores de comparación

Operador	Descripción	Ejemplo
==	Prueba si el valor de dos expresiones son iguales, si es afirmativa, entonces la condición será verdadera.	(A == B)
!=	Comprueba si el valor de los dos operandos son iguales o no, si los valores no son iguales, la condición será verdadera.	(A != B)
>	Comprueba si el valor del operando de la izquierda es mayor que el valor del operando derecho, si es así, entonces la condición será verdadera.	(A > B)
<	Comprueba si el valor del operando de la izquierda es menor que el valor del operando derecho, si es así, entonces la condición será verdadera.	(A < B)
>=	Comprueba si el valor del operando de la izquierda es mayor o igual al valor del operando derecho, si es así, entonces la condición será verdadera.	(A >= B)
<=	Comprueba si el valor del operando de la izquierda es menor o igual al valor del operando derecho, si es así, entonces la condición será verdadera.	(A <= B)

Operadores lógicos (o relacionales)

Operador	Descripción	Ejemplo
&&	Llamado operador relacional AND. Si ambos operandos son	(A && B)

	no cero, entonces la condición será verdadera.	
||	Llamado operador relacional OR. Si cualquiera de los dos operandos son no cero, entonces la condición será verdadera.	(A || B).
!	Llamado operador relacional NOT. Se utiliza para invertir el estado lógico de su operando. Si la condición es verdadera, entonces el operador lógico hará falsa su salida.	!(A && B)

Los operadores de bits y el operador ternario

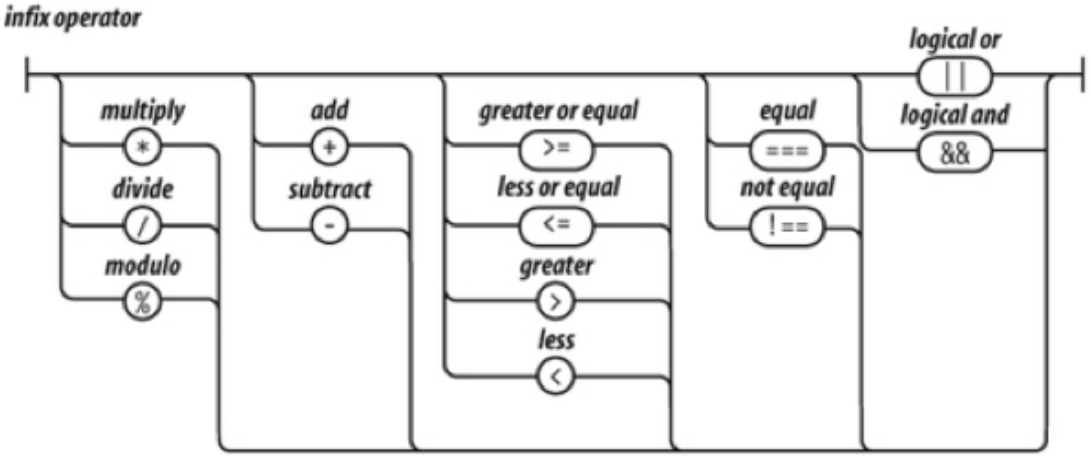
Operador	Descripción	Ejemplo
&	Llamado operador AND. Se realiza una operación booleana AND en cada bit de sus argumentos enteros.	(A & B)
|	Llamado operador OR. Se realiza una operación booleana OR en cada bit de sus argumentos enteros.	(A | B)
^	Llamado operador XOR bit a bit. Realiza una operación OR exclusiva booleana en cada bit de sus argumentos enteros.	(A ^ B)
~	Llamado operador NOT bit a bit. Es un operador unitario y opera mediante la inversión de todos los bits en el operando.	(~B)
<<	Llamado operador bit a bit de corrimiento a la izquierda. Se desplazan todos los bits a la izquierda por el número de posiciones especificado en el segundo operando.	(A << 1)
>>	Llamado desplazamiento a la derecha bit a bit con el operador signo. Se desplazan todos los bits en el primer operando a la derecha, el número de posiciones especificado en el segundo operando. Los bits	(A >> 1)

	introducidos a la izquierda son copia del bit de signo.	
>>>	Llamado desplazamiento a la derecha bit a bit con operador cero. Este operador es igual que el operador >>, salvo que los bits introducidos a la izquierda, son siempre cero.	(A >>> 1)
?:	Expresión de condición. Si la condición es verdadera, entonces el valor de X, de lo contrario el valor Y.	(Condición ?x:y)

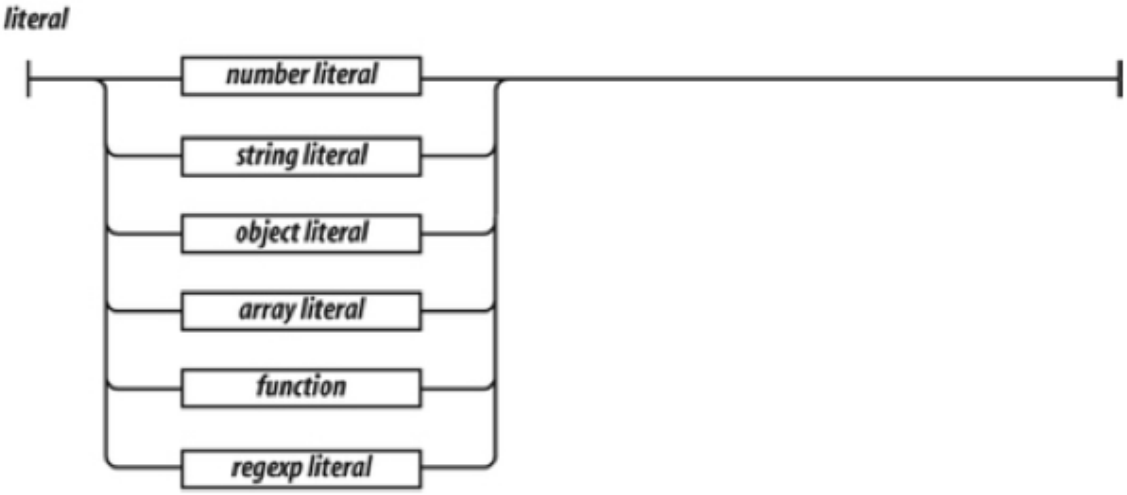
Operadores de asignación

Operador	Descripción	Ejemplo
=	Operador de asignación simple, asigna el valor del operando del lado derecho al operando izquierdo.	C = A + B
+=	Añadir y asignar, se añade el operando derecho al operando izquierdo y se asigna el resultado al operando izquierdo.	C += A es equivalente a C = C + A
-=	Restar y asignar, resta el operando derecho del operando de la izquierda y asigna el resultado al operando izquierdo.	C -= A es equivalente a C = C - A
*=	Multiplicar y operada la asignación, se multiplica el operando derecho por el operando de la izquierda y se asigna el resultado al operando izquierdo.	C *= A es equivalente a C = C * A
/=	Dividir y asignar, se divide el operando izquierdo entre el operando de la derecha y se asigna el resultado al operando izquierdo.	C /= A es equivalente a C = C / A
%=	Módulo y operador de asignación, se calcula el resto de la división del operador izquierdo entre el operador derecho y se asignan el	C %= A es equivalente a C = C % A

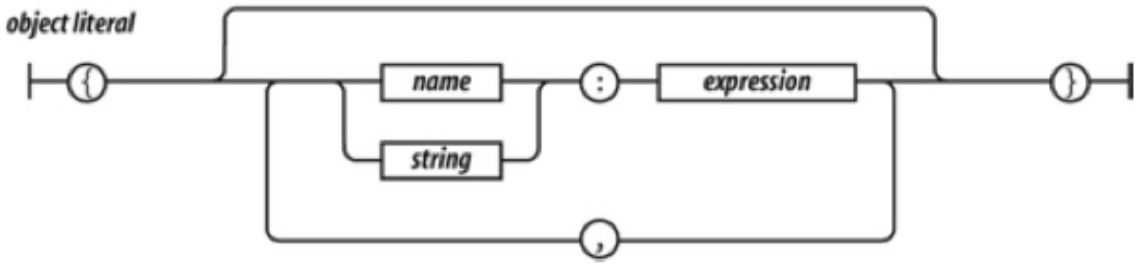
	resultado al operando izquierdo.	



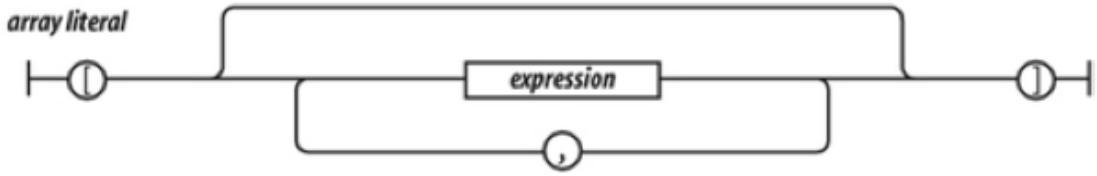
Los objetos literales son una notación conveniente para especificar los objetos nuevos. Los nombres de las propiedades se pueden especificar como nombres o como cadenas. Los nombres se tratan como nombres literales, no como nombres de variables, por lo que los nombres de las propiedades del objeto deben ser conocidos en tiempo de compilación. Los valores de las propiedades son expresiones.



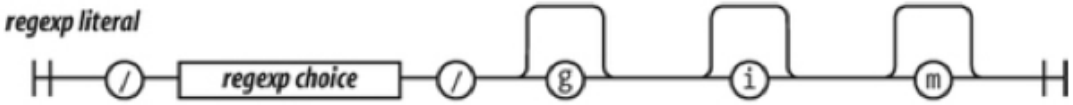
Expresiones de objetos



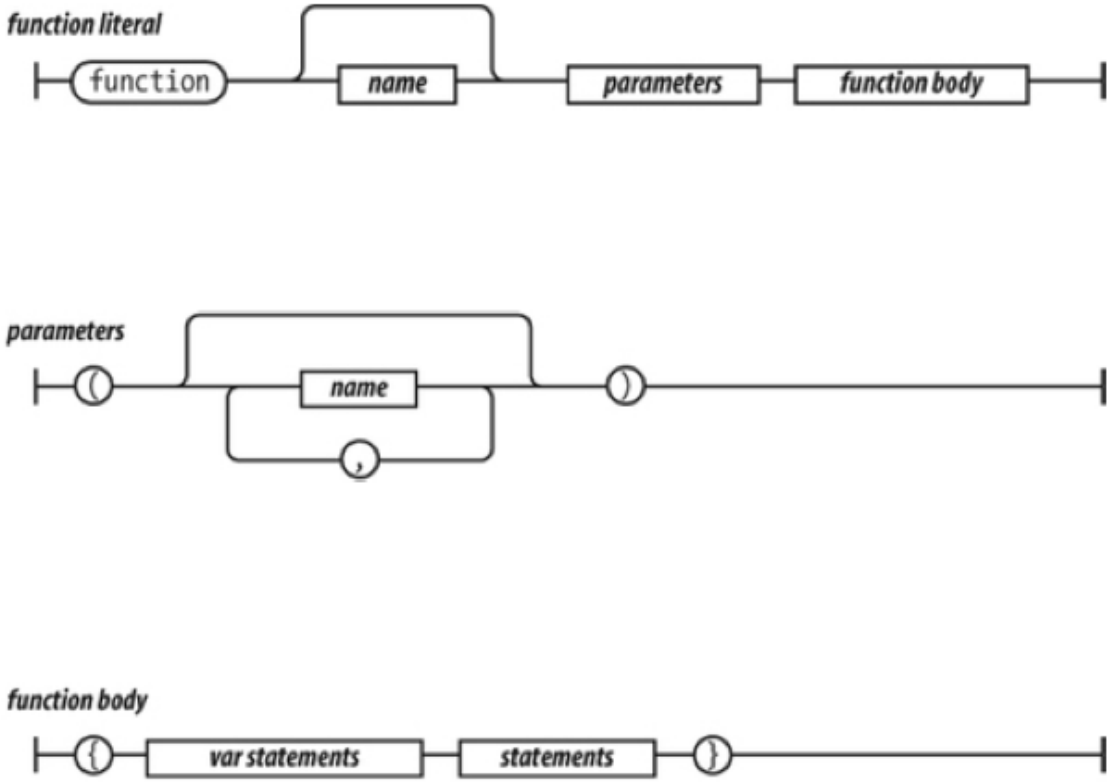
Expresiones de arreglos



Expresiones regulares



Expresiones de funciones



Una función literal define un valor de la función. Puede tener un nombre opcional que puede usar para llamarse a sí misma de forma recursiva. Se puede especificar una lista de parámetros que actuarán como variables inicializadas por los argumentos de invocación. El cuerpo de la función incluye definiciones y declaraciones de variables. En esta sección del texto abarcamos la gramática de JS, en seguida estudiaremos los objetos y en la sección siguiente las funciones a más profundidad.

3.7. Objetos

Los tipos simples de JavaScript son números, cadenas, booleanos (true y false), null, e indefinido. **Todos los demás valores son objetos**. Los números, cadenas y booleanos son similares a los objeto en que tienen métodos, pero son inmutables. Los objetos en JavaScript son colecciones mutables con claves. En JavaScript, las matrices son objetos, las funciones son objetos, las expresiones regulares son objetos, y, por supuesto, los objetos son objetos.

Un objeto es un contenedor de propiedades, donde una propiedad tiene un nombre y un valor. El nombre de una propiedad puede ser cualquier cadena, incluyendo la cadena vacía. El valor de una propiedad puede ser cualquier valor de JavaScript, a excepción del valor indefinido. Los objetos en JavaScript son de clase libre. No existe ninguna limitación en los nombres de nuevas propiedades o en los valores de las propiedades. Los objetos son útiles para la recopilación y organización de datos. Los objetos pueden contener otros objetos, por lo que pueden representar fácilmente estructuras de árbol o grafos. JavaScript incluye una función de vinculación prototipo que permite que un objeto herede las propiedades de otro. Cuando se usa bien, esto puede reducir el tiempo de inicialización del objeto y el consumo de memoria.

JavaScript es un lenguaje de programación orientada a objetos (POO). Un lenguaje de programación puede ser llamado orientado a objetos si proporciona cuatro capacidades básicas para los desarrolladores:

Encapsulación. La capacidad de almacenar información relacionada, datos o métodos juntos en un objeto.

Agregación. La capacidad de almacenar un objeto dentro de otro objeto.

Herencia. La capacidad de una clase de confiar en otra clase (o el número de clases) para algunas de sus propiedades y métodos.

Polimorfismo. La capacidad de escribir una función o método que funciona en una variedad de maneras diferentes.

Los objetos se componen de atributos. Si un atributo contiene una función, que se considera un método del objeto, el atributo se considera una propiedad.

Propiedades de los objetos: las propiedades de un objeto pueden ser cualquiera de los tres tipos de datos primitivos, o cualquiera de los tipos de datos abstractos, tales como otro objeto. Las propiedades de los objetos suelen ser las variables que se utilizan internamente

en los métodos del objeto, pero también pueden ser variables visibles globalmente que se utilizan en toda la página.

La sintaxis para añadir una propiedad a un objeto es:

```
objectoNombre.objectoPropiedad = ValorPropiedad;
```

Por ejemplo:

```
var str.título = "Mi página Web"
```

Métodos del objeto: los métodos son funciones que permiten al objeto hacer algo o dejar que se haga algo con él. Hay poca diferencia entre una función y un método, excepto que una función es una unidad independiente de los estados y un método está unido a un objeto y se puede hacer referencia a la palabra clave **this**.

Los métodos son útiles para todo, desde el que muestra el contenido del objeto en la pantalla, o por ejemplo, para realizar operaciones matemáticas complejas en un grupo de propiedades locales y parámetros.

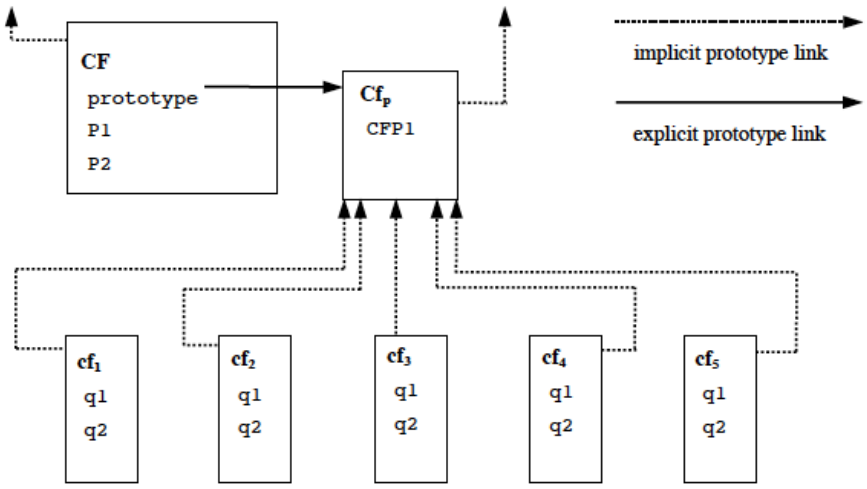
A continuación se presenta un ejemplo simple para mostrar cómo utilizar el método **write ()** del objeto documento para escribir cualquier contenido en pantalla:

```
document.write(" JavaScript es emocionante");
```

JS no utiliza clases tales como JAVA y C++, entre otros. En su lugar, los objetos se pueden crear de varias maneras, incluyendo a través de una notación literal o a través de constructores que crean objetos y luego ejecutan código que inicializa todo o parte de ellos, mediante la asignación de valores iniciales a sus propiedades. Cada constructor es una función que tiene una propiedad denominada prototipo, que se utiliza para implementar la herencia basada en prototipos y propiedades compartidas. Los objetos se crean mediante el uso de constructores de nuevas expresiones.

Cada objeto creado por un constructor tiene una referencia implícita (llamada prototipo del objeto) en el valor de la propiedad prototipo de su constructor. Por otra parte, un prototipo puede tener una referencia implícita no nula a su prototipo, y así sucesivamente, lo que se llama la cadena de prototipo. En otras palabras, primero el objeto mencionado directamente se examina para este tipo de propiedad, y si ese objeto contiene la propiedad con nombre, que es la propiedad a la que se refiere la referencia, se toma el valor de dicha propiedad pero, si ese objeto no contiene la propiedad nombre, el prototipo de ese objeto es examinado enseguida, y así sucesivamente hasta revisar toda la cadena prototipo. De acuerdo con ECMAScript[15].

A diferencia de los lenguajes basado en clases, las propiedades se pueden añadir a los objetos de forma dinámica mediante la asignación de valores a ellos. Es decir, los constructores no están obligados a nombrar o asignar valores a todas o cualquiera de las propiedades del objeto construido. En el diagrama siguiente ECMAScript, se podrá apreciar esto.



Resulta conveniente que todos los objetos que no contienen directamente una propiedad particular, hagan que su prototipo comparta esa propiedad y su valor. CF es un constructor (y también un objeto). Cinco objetos han sido creados mediante el uso de nuevas expresiones: cf_1, cf_2, cf_3, cf_4 y cf_5. Cada uno de estos objetos contiene propiedades denominadas q1 y q2. Las líneas discontinuas representan la relación implícita prototipo, así que, por ejemplo, el prototipo de cf3 es CFp. El constructor, CF, sí tiene dos propiedades,

denominadas P1 y P2, que no son visibles para CFp, cf1, cf2, cf3, cf4 o cf5. La propiedad con nombre CFP1 en CFp es compartida por cf1, cf2, cf3, cf4 y cf5 (pero no por CF), al igual que las propiedades que se encuentran en la cadena de prototipo implícita de la CFp que no se llamen q1, q2 o CFP1. Tenga en cuenta que no existe una relación implícita entre el prototipo de CF y de la CFp.

3.8. Objetos literales

Los objetos literales proporcionan una notación muy conveniente para la creación de nuevos objetos con valores asignados. Un objeto literal es un par de llaves con cero o más pares nombre-valor. Un objeto literal puede aparecer en cualquier lugar:

```
var empty_object = {};
var stooge = {
"first-name": "Jerome",
"last-name": "Howard"
};
```

El nombre de una propiedad puede ser cualquier cadena, incluyendo la cadena vacía. Las comillas en el nombre de una propiedad en un objeto literal son opcionales si el nombre es un nombre permitido por JavaScript y no una palabra reservada. La coma se utiliza para separar los pares. El valor de una propiedad puede ser obtenido de cualquier expresión, incluyendo otro objeto literal.

Los objetos pueden estar anidados:

```
var flight = {
airline: "Oceanic",
number: 815,
departure: {
IATA: "SYD",
time: "2004-09-22 14:55",
city: "Sydney"
},
arrival: {
IATA: "LAX",
time: "2004-09-23 10:42",
city: "Los Angeles"
}
};
```

Recuperación

Los valores se pueden recuperar de un objeto envolviendo una expresión de cadena en un sufijo []. Si la expresión de cadena es una constante, y si se trata de un nombre legal de JavaScript y no una palabra reservada, entonces la notación se puede utilizar en su lugar. Se prefiere esta notación debido a que es más compacta y se lee mejor:

```
stooge["first-name"] // "Joe"
flight.departure.IATA // "SYD"
```

Se produce el valor indefinido si se hace un intento por recuperar un miembro inexistente:

```
stooge["middle-name"] // undefined
flight.status         // undefined
stooge["FIRST-NAME"] // undefined
```

El operador | | se puede utilizar para rellenar los valores predeterminados:

```
var middle = stooge["middle-name"] || "(none)";
var status = flight.status || "unknown";
```

El intento de recuperar los valores indefinidos producirá una excepción TypeError, esto puede evitarse con el operador &&:

```
flight.equipment // undefined
flight.equipment.model // throws "TypeError"
flight.equipment && flight.equipment.model // undefined
```

Actualizar

Un valor en un objeto puede ser actualizado por asignación. Si el nombre de la propiedad ya existe en el objeto, el valor de la propiedad se sustituye:

```
stooge['first-name'] = 'Jerome';
```

Si el objeto no tiene aún ese nombre de propiedad, se aumenta la propiedad al objeto:

```
stooge['middle-name'] = 'Lester';
stooge.nickname = 'Curly';
flight.equipment = {
model: 'Boeing 777'
};
```

```
flight.status = 'overdue';
```

Referenciar

Los objetos se pasan por referencia, nunca se copian:

```
var x = stooge;
x.nickname = 'Curly';
var nick = stooge.nickname;
// nick es 'Curly' porque "x" y stooge
// son referencias para el mismo objeto
var a = {}, b = {}, c = {};
// a, b, y c cada una
// refiere a un objeto vacío diferente
a = b = c = {};
// a, b, y c todas referencian al mismo objeto vacío
```

3.9. Prototipo

Cada objeto está vinculado a un objeto prototipo del que puede heredar propiedades, todos los objetos creados a partir de objetos literales están vinculados a **Object.prototype**, un objeto que viene de serie con JavaScript.

Cuando usted hace un nuevo objeto, puede seleccionar que sea prototipo. El mecanismo que JavaScript ofrece para hacer esto es complicado y complejo, pero puede ser simplificado significativamente. Vamos a añadir un método de creación del objeto función. El método crea un nuevo objeto que utiliza un objeto como su prototipo.

```
if (typeof Object.create !== 'function') {
Object.create = function (o) {
var F = function () {};
F.prototype = o;
return new F();
};
}
var another_stooge = Object.create(stooge);
```

El enlace de prototipo no tiene ningún efecto sobre la actualización. Cuando hacemos cambios a un objeto, el objeto prototipo no es tocado:

```
another_stooge['first-name'] = 'Harry';
another_stooge['middle-name'] = 'Moses';
another_stooge.nickname = 'Moe';
```

El enlace de prototipo se utiliza solo en la recuperación. Si tratamos de recuperar un valor de una propiedad de un objeto, y si el objeto no tiene el nombre de propiedad, JavaScript intenta recuperar el valor de la propiedad del objeto prototipo. Si ese objeto también carece de la propiedad, entonces va a su prototipo, y así sucesivamente hasta que el proceso finalmente toque fondo con **Object.prototype**. Si la propiedad deseada no existe en ninguna parte de la cadena de prototipo, el resultado es el valor undefined. Esto se llama delegación.

La relación prototipo es una relación dinámica. Si añadimos una nueva propiedad a un prototipo, la propiedad será visible inmediatamente en todos los objetos que se encuentran sobre la base de ese prototipo:

```
stooge.profession = 'actor';
another_stooge.profession // 'actor'
```

Reflexión

Es fácil inspeccionar un objeto para determinar qué propiedades tiene al intentar recuperar sus propiedades y examinar los valores obtenidos. El operador **typeof** puede ser muy útil para determinar el tipo de una propiedad:

```
typeof flight.number // 'number'
typeof flight.status // 'string'
typeof flight.arrival // 'object'
typeof flight.manifest // 'undefined'
```

Debemos tener cuidado, ya que cualquier propiedad de la cadena de prototipos puede producir un valor de:

```
typeof flight.toString // 'function'
typeof flight.constructor // 'function'
```

Hay dos enfoques para hacer frente a estas propiedades no deseadas. El primero busca y rechaza los valores de función. El otro enfoque usa el método **hasOwnProperty**, que

devuelve verdadero si el objeto tiene una propiedad particular. El método hasOwnProperty no se fija en la cadena de prototipo:

```
flight.hasOwnProperty('number') // true
flight.hasOwnProperty('constructor') // false
```

Enumeración

Se puede iterar sobre la declaración de todos los nombres de las propiedades de un objeto. La enumeración incluirá todas las propiedades, incluyendo funciones y propiedades prototipo en los que podría no estar interesado en lo que es necesario filtrar los valores. Los filtros más comunes son el método que hace uso de hasOwnProperty y typeOf para excluir funciones.

No hay garantía en la enumeración de los nombres, así se debe contemplar los nombres que aparezcan en cualquier orden. Si desea asegurarse de que las propiedades aparecen en un orden determinado, lo mejor es evitar la declaración en su totalidad y, en su lugar, hacer un arreglo que contenga los nombres de las propiedades en el orden correcto:

```
var i;
var properties = [
'first-name',
'middle-name',
'last-name',
'profession'
];
for (i = 0; i < properties.length; i++) {
document.writeln(properties[i] + ': ' +
another_stooge[properties[i]]);
}
```

Borrar

El operador **delete** se puede utilizar para eliminar una propiedad de un objeto. Se eliminará una propiedad del objeto, si la tiene. No va a tocar cualquiera de los objetos en el enlace de prototipo.

Eliminar una propiedad de un objeto puede permitir que una propiedad del enlace de prototipo se viera:

```
another_stooge.nickname // 'Moe'
// eliminar nickname de another_stooge, revela
```

```
// el nickname del prototipo.
delete another_stooge.nickname;
another_stooge.nickname // 'Curly'
```

Reducción global

JavaScript hace que sea fácil de definir variables globales que pueden contener todos los activos de su aplicación. Por desgracia, las variables globales debilitan la resistencia de los programas y se deben evitar.

Una forma de minimizar el uso de variables globales es crear una única variable global para su aplicación:

```
var MYAPP = {};
```

Esa variable, entonces se convierte en contenedor para su aplicación:

```
MYAPP.stooge = {
"first-name": "Joe",
"last-name": "Howard"
};
MYAPP.flight = {
airline: "Oceanic",
number: 815,
departure: {
IATA: "SYD",
time: "2004-09-22 14:55",
city: "Sydney"
},
arrival: {
IATA: "LAX",
time: "2004-09-23 10:42",
city: "Los Angeles"
}
};
```

Al reducir su huella global a un solo nombre, se reduce significativamente la posibilidad de malas interacciones con otras aplicaciones, widgets o bibliotecas. Su programa también se convierte en más fácil de leer, ya que es obvio que MYAPP.stooge se refiere a una estructura de nivel superior.

3.10. Flujo de control

Cuando deseamos ejecutar uno u otro conjunto de sentencias o comandos, los lenguajes de programación reservan estructuras sujetas a condicionales que evalúan variables de control, entre las más importantes están:

- **if/then/else**
- **while/do..while**
- **switch/case**
- **for/in**
- **label**
- **continue**
- **break**
- **throw**
- **try/catch/fanally**
- **returns**
- **let**
- **do…while**
- **block**
- **export**
- **debugger**

La estructura de sentencias es encerrada en {} y es opcional terminarla con el ";",

{

sentencia;

sentencia;

…

sentencia

};

if … else

Cuando requerimos ejecutar sentencias bajo la condición lógica específica que sea verdadera o, por el contrario, sean ignoradas por ser la condición falsa, recurrimos a la secuencia de control:

```
if (condición){
    sentencia1
} else{
    sentencia2
}
```

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
<script type="text/javascript" src="funciones.js"></script>


<noscript>
  <p>Nuestro libro favorito</p>
  <p>Se encuentra inactivo JavaScript.
Por favor vuelve a activarlo.</p>
</noscript>


 </head>
 <body>
<script type="text/javascript">
//-->
var edad = 15;
if( edad > 18 ){
   document.write("<b>Califica para credencial IFE</b>");
} else{
   document.write("<b>No califica para credencial IFE</b>");
}
//-->
</script>


</body>
</html>
```

En este ejemplo podemos observar que dependiendo del valor de la variable **edad,** se desplegará "califica para credencial IFE", o por lo contrario, **else**, "no califica para credencial IFE". La sintaxis para múltiples sentencias de este tipo de control es:

```
if (condición1){
   sentencia1
else if (condición2)
   sentencia2
else if (condición3)
   sentencia3
...
else
   sentenciaN
}
```

While

Secuancia de control que ejecuta un bloque de sentencias mientras la condición de control evaluada antes de cada ciclo sea como verdadera, de lo contrario termina la ejecución del ciclo.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
<script type="text/javascript" src="funciones.js"></script>


<noscript>
  <p>Nuestro libro favorito</p>
  <p>Se encuentra inactivo JavaScript.
Por favor vuelve a activarlo.</p>
</noscript>


 </head>
 <body>
<script type="text/javascript">
var n = 0;
var x = 0;
while (n < 5) {
  n ++;
  x += n;
}
</script>

</body>
</html>
```

Este ejemplo expresa que si el valor de n es menor que 5, el programa realiza la siguiente pasada del ciclo hasta que n = 5.

do … while

Es igual que el flujo de control anterior, cambia solo en que por lo menos en una ocasión se ejecuta el bloque de sentencias.

```
Var i=0
do {
   i += 1;
   document.write(i);
} while (i < 10);
```

Se ejecuta el despliegue del valor de i, y en seguida cada paso dependerá de que sea menor que 10.

switch …case

Ejecutará las sentencias que correspondan al caso, de no ser así, por default ejecuta un juego de sentencias. Para el siguiente ejemplo cambie el valor de la variable por la letra Z:

```
switch (condición)
{
  case condition 1: statement(s)
                    break;
  case condition 2: statement(s)
                    break;
   ...
  case condition n: statement(s)
                    break;
  default: statement(s)
}
```

```
<!DOCTYPE html>
<html lang="es">
<head>
 <title>Mi libro favorito</title>
 <meta charset="UTF-8">
 <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
<script type="text/javascript" src="funciones.js"></script>


<noscript>
 <p>Nuestro libro favorito</p>
```

```
  <p>Se encuentra inactivo JavaScript.
Por favor vuelve a activarlo.</p>
</noscript>


 </head>
 <body>
<script type="text/javascript">
<!--
var letra='C';
document.write("conmuta al caso C<br />");
switch (letra)
{
  case 'A': document.write("Vida larga<br />");
            break;
  case 'B': document.write("Vida corta<br />");
            break;
  case 'C': document.write("Vida entre literatura <br />");
            break;
  case 'D': document.write(" Vida perdida<br />");
            break;
  case 'F': document.write("Resistir es vivir<br />");
            break;
  default:  document.write("Negar la vida<br />")
}
document.write("Despliga el caso C");
//-->
</script>

</body>
</html>
```

for (inicialización; condición; incremento)

El ciclo **for** es la forma más compacta de ciclo, e incluye las tres partes importantes siguientes:

- La inicialización del ciclo, donde se inicializa el contador a un valor de partida. La instrucción de inicialización se ejecuta antes de que comience el ciclo.

- La declaración de prueba que permite determinar si la condición dada es cierta o no. Si la condición es verdadera, entonces el código dado dentro del ciclo se ejecutará, de lo contrario el ciclo **for** termina.

- Incremento, es la instrucción de iteración en la que puede aumentar o disminuir su contador que contrata el ciclo.

Puede poner las tres partes en una sola línea, separadas por un punto y coma.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
<script type="text/javascript" src="funciones.js"></script>


<noscript>
  <p>Nuestro libro favorito</p>
  <p>Se encuentra inactivo JavaScript.
Por favor vuelve a activarlo.</p>
</noscript>


 </head>
 <body>
<script type="text/javascript">
<!--
var count;
document.write("Comienza el ciclo" + "<br />");
for(count = 0; count < 12; count++){
  document.write("Valor del contador: " + count );
  document.write("<br />");
}
document.write("Salir del bucle!");
//-->
</script>

</body>
</html>
```

for … in

Hay otro ciclo con el apoyo de JavaScript. Se llama ciclo **for ... in**. Este ciclo se utiliza para recorrer las propiedades de un objeto.

Tenga presente el concepto de ciclo, recuerde que un objeto está definido por propiedades. Este código es muy útil.

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


<noscript>
  <p>Nuestro libro favorito</p>
  <p>Se encuentra inactivo JavaScript.
Por favor vuelve a activarlo.</p>
</noscript>


<head>
```

```
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
<script type="text/javascript" src="funciones.js"></script>


 </head>
 <body>
<script type="text/javascript">
<!--
var aProperty;
document.write("Explorar las propiedades de un objeto<br /> ");
for (aProperty in navigator)
{
  document.write(aProperty);
  document.write("<br />");
}
document.write("Salir del ciclo!");
//-->
</script>

</body>
</html>
```

El programa anterior imprime las propiedades del objeto **navigator** del navegador Web.

label

Label nos permite etiquetar una posición para referir a una sentencia, se usa con ayuda de sentencias **break** y **continue**.

```
etiqueta :
    sentencia
```

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
<script type="text/javascript" src="funciones.js"></script>


<noscript>
  <p>Nuestro libro favorito</p>
  <p>Se encuentra inactivo JavaScript.
Por favor vuelve a activarlo.</p>
</noscript>
```

```
 </head>
 <body>

<script type="text/javascript">
<!--
document.write("interacción de ciclo!<br /> ");
externobucle:   // nombre de label
for (var i = 0; i < 5; i++)
{
  document.write("externobucle: " + i + "<br />");
  internobucle:   // nombre de label
  for (var j = 0; j < 5; j++)
  {
     if (j >  3 ) break ;         // salir de bucle interno
     if (i == 2) break internobucle; // haga lo mismo
     if (i == 4) break externobucle; // Salir del bucle externo
     document.write("internobucle: " + j + "  <br />");
   }
}
document.write("Exiting the loop!<br /> ");
//-->
</script>

</body>
</html>
```

continue

La sentencia **continue** indica al intérprete iniciar de inmediato la siguiente iteración del ciclo y saltar el bloque de código restante.

Cuando se encuentra una sentencia **continue**, el flujo del programa se trasladará a la expresión de comprobación del ciclo de inmediato y, si la condición es verdadera, entonces comenzará la próxima iteración, de lo contrario el control sale del ciclo.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
<script type="text/javascript" src="funciones.js"></script>


<noscript>
  <p>Nuestro libro favorito</p>
  <p>Se encuentra inactivo JavaScript.
Por favor vuelve a activarlo.</p>
</noscript>


 </head>
```

```
 <body>

<script type="text/javascript">

<!--
var x = 1;
document.write("Interacción paso de bucle<br /> ");
while (x < 11)
{
  x = x + 1;
  if (x == 6){
     continue;  // habilidad resto del cuerpo del bucle
  }
  document.write( x + "<br />");
}
document.write("Salir de bucle!<br /> ");
//-->
</script>

</body>
```

Funciones

Antes de usar una función, tenemos que definir esa función. La forma más común para definir una función en JavaScript es mediante el uso de la palabra clave **function**, seguido de un nombre de función único, una lista de parámetros (que puede estar vacía), y un bloque de instrucciones rodeado por llaves. La sintaxis básica se muestra aquí:

```
<script type="text/javascript">
<!--
function nombre_función(lista de parámetros)
{
  sentencias
}
//-->
</script>
```

Cuando leemos código JS, generalmente nos damos cuenta que está en forma de segmentos dados por funciones, bloques de código que agrupan expresiones **JS**, estas son invocadas en cualquier zona del código principal. La palabra clave **function** identifica el nombre con el que será llamada la función. En el siguiente ejemplo creamos la función saludo, con una lista de parámetros vacío, misma que es llamada y despliega un cadena de texto.

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

```
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
<script type="text/javascript" src="funciones.js"></script>


<noscript>
  <p>Nuestro libro favorito</p>
  <p>Se encuentra inactivo JavaScript.
Por favor vuelve a activarlo.</p>
</noscript>


<noscript>
  <p>Nuestro libro favorito</p>
  <p>Se encuentra inactivo JavaScript.
Por favor vuelve a activarlo.</p>
</noscript>


 </head>
 <body>

<script type="text/javascript">

<!--
function saludo()
{
   document.write("esta es una función<br /> ");
}
//-->

saludo();
//esta es la llamada a la función-->

</script>

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

Podemos declarar variables locales dentro de las funciones y devolver valores con la instrucción **return**. Las funciones tienen el objeto de estructurar el código JS, de manera que sean reutilizables, prácticas y que permitan asociar los eventos dentro del programa con eventos divididos en funciones.

Por ejemplo, el siguiente programa realiza el llamado de la función **adición y** realiza la suma de los valores dados como parámetros:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">
```

```
<link rel="stylesheet"  href="misestilos.css" >
<script type="text/javascript" src="funciones.js"></script>


 </head>
 <body>

</script>

<script type="text/javascript">
var total=suma(4,-7);
document.write("Adición de x + y:   ", total);

</script>

</body>
</html>
```

```
// JavaScript Document
function suma (x, y){
var sum;
sum=x+y;
return (sum);
}
```

3.11. Funciones predefinidas en JS

Así como JS proporciona objetos predefinidos, también proporciona funciones predefinidas.

eval()

Devuelve el valor de una expresión JS interpretada.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
<script type="text/javascript" src="funciones.js"></script>


<noscript>
  <p>Nuestro libro favorito</p>
  <p>Se encuentra inactivo JavaScript.
Por favor vuelve a activarlo.</p>
</noscript>


 </head>
 <body>

<script type="text/javascript">
```

```
var total;

total= eval("1+3+5+7");
document.write("Adición de x + y +...:   ", total);
</script>

</body>
</html>
```

atob()

Convierte el texto codificado en base64 a binario; y la función btoa() hace lo inverso.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi libro favorito</title>
  <meta charset="UTF-8">
  <meta name="description" content="ejemplo adjuntar CSS">


<link rel="stylesheet"  href="misestilos.css" >
<script type="text/javascript" src="funciones.js"></script>


 </head>
 <body>

<script type="text/javascript">


string1="Hola CONALEP"
document.write('Cadena a codificar: '+string1 + "<br />")

encodedData = btoa(string1)
document.write('Cadena cófidicada : '+encodedData + "<br />")

decodedData = atob(encodedData)
document.write('Decodificación con atob()  : ' +decodedData + "<br />")
</script>

<noscript>
  <p>Nuestro libro favorito</p>
  <p>Se encuentra inactivo JavaScript.
Por favor vuelve a activarlo.</p>
</noscript>
 </body>
</html>
```

isNaN()

Identifica si el valor no es un número.

number()

Convierte una cadena en un valor de punto flotante.

parseFloat()

Convierte una cadena en un número flotante.

3.12. Eventos

JS usa un modelo orientado a eventos, para ello cuenta con elementos como botón, enlace, imagen, que responden al clic del ratón, al presionar una tecla, etc. Son acciones del usuario que desencadenan código JS, dotando de interactividad al documento HTML. Los eventos permiten responder al usuario con lógica de programación, a estos elementos se les llama manipuladores o manejadores de eventos, por ejemplo:

onClik: responde a eventos de clic del ratón.
onAbort: el evento detiene la carga de páginas o por ejemplo archivos de imagen.
onChange: evento se activa cuando un elemento de formulario cambia.
onClik: se activa cuando el ratón está sobre un botón o link.
onDragDrop: se activa al final de arrastrar un algo dentro de nuestro documento.
onError: cuando se rompe una carga de archivo.
onFocus: cuando el foco de un elemento de la página es alcanzado.
onKeyDown: cuando se presiona alguna tecla.
onKeyPress: cuando se deja presionada una tecla se activa el evento.
onKeyUp: cuando el usuario deja de presionar una tecla.
onLoad: cuando se termina de cargar una página o imagen se activa este evento.
onMouseDown: al hacer clic sobre un elemento se activa este evento.
onMouseOut: cuando el puntero del ratón sale de una área se activa este evento.
onMouseOver: cuando el puntero del ratón entra en una área ocupada por un elemento del documento.
onMouseUp: cuando se suelta el botón del ratón.
onMove: cuando se mueve la ventana del navegador.
onPresize: cuando se redimensiona la ventana del navegador.
onPreset: responde al reset de un formulario.
onBlur: cuando se mueve el puntero fuera de un objeto de formulario.

3.13. jQuery

En el momento de escribir este texto, jQuery es, por un amplio margen, la biblioteca disponible más popular JavaScript, se utiliza en más del 50% de todos los mejores sitios Web. Fue escrita por John Resig y lanzada en 2006. jQuery, como la mayoría de bibliotecas de JavaScript, es de código abierto y gratuito para el público. jQuery es una biblioteca en el verdadero sentido de la palabra. Es una colección de funciones predefinidas y métodos cargados para sus documentos a través de un script externo que puede accederse y utilizarse como mejor le parezca. jQuery es una biblioteca muy grande, debido a su facilidad de uso, y ha bajado la barrera de entrada para un diseñador, para comenzar la codificación con el lenguaje Javascript. El equipo de jQuery ha hecho un gran trabajo al crear una capa sobre JavaScript, haciendo que el lenguaje sea accesible a cualquier persona que necesite codificar con él. Para bien o para mal, no es raro que la lista de habilidades necesarias par un diseñador Web incluya JQuery, sin omitir JavaScript. jQuery es tan inmensamente popular que muchas personas no lo ven como JavaScript real. Pero, de nuevo, no se olvide todavía es solo JavaScript.

La biblioteca jQuery contiene casi todo lo que se necesita para construir cualquier página Web, desde un pequeño sitio de marketing hasta una aplicación Web robusta. Hay un sinnúmero de libros, artículos y publicaciones escritas sobre todos los aspectos de jQuery, incluyendo toda la documentación de la API de jQuery.com.

La biblioteca jQuery se puede descargar desde jquery.com. Puede obtener una versión de producción de la biblioteca, que ha sido comprimida (sin espacios en blanco), o una versión de desarrollo, que es más grande, pero mantiene todos los espacios en blanco y comentarios apropiados. La versión de desarrollo de jQuery puede ser una gran fuente de aprendizaje, por estar construida por algunas de las mejores mentes de JavaScript en la industria.

Glosario

Tipo

Un tipo es un conjunto de valores de datos .

Valor primitivo

Un valor primitivo es un miembro de uno de los tipos **Undefined, Null , Boolean , Number o String.** Un valor primitivo es un dato que se representa directamente en el nivel más bajo de la implementación del lenguaje .

Objeto

Un objeto es un miembro del tipo **objeto** . Es una colección no ordenada de propiedades cada una de las cuales contiene un valor primitivo, un objeto o una función. A una función almacenada en una propiedad de un objeto se le llama método.

Constructor

Un constructor es un objeto **Function** que se crea y se inicializa. Cada constructor tiene asociado un objeto prototipo que se utiliza para implementar la herencia y las propiedades compartidas.

Prototipo

Un prototipo es un objeto utilizado para implementar la estructura, el estado y el comportamiento de la herencia en JS. Cuando un constructor crea un objeto, ese objeto implícitamente hace referencia al constructor que está asociado, con el fin de resolver las referencias de la propiedad. El prototipo asociado al constructor puede ser referenciado por el **constructor.prototype,** expresión de programa y propiedades que añade un objeto de prototipo a través de la herencia, para todos los objetos que comparten el prototipo.

Objeto Nativo

Un objeto nativo es cualquier objeto suministrado por una aplicación ECMAScript independiente del ambiente del ordenador. Los objetos nativos estándar están definidos en esta especificación. Algunos objetos nativos son intrínsecos; otros pueden ser construidos durante el curso de la ejecución de un programa de ECMAScript.

Objeto de host

Un objeto de **host** es cualquier objeto proporcionado por el entorno de acogida para completar el entorno de ejecución de ECMAScript. Cualquier objeto que no es nativo es un objeto de host.

Valor indefinido

El valor definido es un valor primitivo que se utiliza cuando a una variable no se le ha asignado un valor.

Undefined Type

El tipo no definido tiene exactamente un valor , llamado indefinido.

Valor Null

El valor nulo es un valor primitivo que representa la referencia nula, vacía o no existente.

Null type

El tipo Null tiene exactamente un valor, llamado nulo.

Valor booleano

Un valor booleano es un miembro del tipo Boolean y es uno de dos valores únicos, verdadero o falso.

Type Boolean

El tipo Boolean representa una entidad lógica, tiene exactamente dos valores únicos. Uno se llama verdadero y el otro se llama falso.

Objeto Boolean

Un objeto Boolean es un miembro del tipo objeto y es una instancia del objeto Boolean incorporado. Es decir, un objeto Boolean se crea mediante el constructor Boolean en una nueva expresión, suministrando un valor booleano como argumento. El objeto resultante tiene una propiedad implícita (sin nombre) que es el valor booleano. El objeto Boolean se puede convertir en un valor booleano.

Valor String

Un valor de cadena es un miembro del tipo string, es una secuencia finita de ceros o valores enteros sin signo de 16bits .

Tipo String

El tipo String es el conjunto de todos los valores de cadena.

Objeto String

Un objeto String es un miembro del tipo objeto y es una instancia del objeto String incorporado . Es decir, un Objeto String se crea utilizando el constructor String en una nueva expresión, suministrando una cadena como argumento. El objeto resultante tiene una propiedad implícita (sin nombre) que es la cadena. Un objeto String se puede convertir a un valor de cadena, llamando al constructor String como una función.

Valor Number

Un valor numérico es miembro del tipo Number y es una representación directa de un número.

Tipo Number

El tipo **Number** es un conjunto de valores que representa números. Representan el conjunto de valores en formato IEEE 754 de 64 bits de doble precisión, incluyendo los valores espciales " Not- a-Number " (NaN), infinito positivo e infinito negativo .

Objeto Number

Un objeto Number es un miembro del tipo objeto y es una instancia del objeto **Numbe**r.

Es decir, un objeto Number que se crea mediante el constructor **Number** en una nueva expresión, el suministro de un número como argumento. El objeto resultante tiene una propiedad implícita (sin nombre) que es el número. Un objeto Number puede ser convertido a un valor numérico mediante una llamada al constructor número **Number**.

Infinity

El valor primitivo infinito representa el valor positivo infinito. Este valor es un miembro del tipo Number.

NaN

El valor NaN representa el conjunto de valores del estándar IEEE-754 " Not- a- Number " . Este valor es un miembro del tipo Number.

URL'S

http://eleventyone.done.hu/OReilly.JavaScript.The.Good.Parts.May.2008.pdf

http://www.tutorialspoint.com/javascript/javascript_objects.htm

https://developer.mozilla.org/es/docs/Referencia_de_JavaScript_1.5

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

http://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html#use-case

http://www.quirksmode.org/js/intro.html

https://developer.mozilla.org/es/learn/javascript

http://www.javascriptkit.com/jsref/arithmetic_operators.shtml

http://msdn.microsoft.com/es-es/library/ie/b9w25k6f(v=vs.94).aspx

https://developer.mozilla.org/es/learn/javascript

http://www.tutorialspoint.com/javascript/javascript_quick_guide.htm

http://www.d.umn.edu/itss/training/online/Webdesign/javascript.html

http://msdn.microsoft.com/en-us/library/ie/d1et7k7c(v=vs.94).aspx

http://www.visibone.com/products/ebk10-11_850.jpg

http://javascript-reference.info

http://librosWeb.es/javascript/

http://www.fundacionjosepons.com/estudios/multimedia/1117035859Javascript.pdf

http://www.desarrolloWeb.com/manuales/20/

http://www.manualdejavascript.com

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide

http://becas2012.fundayacucho.gob.ve/becas2012/javascript.pdf

http://www.Webestilo.com/javascript/

http://www.w3schools.com/js/

http://www.ohio.edu/technology/training/upload/java-script-reference-guide.pdf

http://www.etnassoft.com/biblioteca/

http://www.etnassoft.com/biblioteca/css3-y-javascript-avanzado/

http://jquery.com

http://www.etnassoft.com/biblioteca/javascript-in-ten-minutes/

http://www.pixelovers.com/5-mejores-libros-aprender-javascript-1265958

http://www.linguee.es/espanol-ingles/traduccion/el+sue%F1o+de+una+noche+de+verano.html

http://html5labs.interoperabilitybridges.com

http://www.bookofjavascript.com

Familia de código

http://www.java2s.com/Code/JavaScriptReference/CatalogJavaScriptReference.htm

Referencias

[1] Bill Scott & Theresa Neil (2009) Designing Web interfaces. California: O'Reilly Media
[2] Bergin Thomas J. & Gibson Richard G. (1996) History of programming lenguages II. New York: ACM
[3] Jennifer Niederst R. (2012) Learning Web design. California: O'Reilly
[4] Quigley Ellie (2010) JavaScript. EEUU: Prentice Hall. Consulta: 5 de Septiembre de 2013, de http://books.google.es/books?id=zyMUnspbsekC&printsec=frontcover&dq=javascript&hl=es&sa=X&ei=NNcoUqHwIa-C2gWrq4GwCg&ved=0CGcQ6AEwBzgK#v=onepage&q=javascript&f=false
[5] Gutierrez Emmanuel (2009) JavaScript conceptos básicos y avanzados. Barcelona: ENI. Consulta: 5 de Septiembre de 2013, de http://books.google.es/books?id=gsxVpvEC4iUC&printsec=frontcover&dq=javascript&hl=es&sa=X&ei=d5coUo60BqLD2wXnxYHIDA&ved=0CGkQ6AEwCA#v=onepage&q=API&f=false
[6] W3C (2005) Document Object Model (DOM). W3C. Consulta: 5 de Septiembre de 2013, de http://www.w3.org/DOM/
[7] Paul Wilton & Jeremy McPeak (2010) Beginning JavaScript. Canada: Wiley
[8]Eric Lévénez (2013) Computer languages history. Consulta: 5 de Septiembre de 2013, de http://www.levenez.com/lang/lang_a4.pdf http://www.levenez.com/lang/
[9] Rojas R. (1998) A Tutorial Introduction to the Lambda Calculus. Autoeducación. http://www.etnassoft.com/biblioteca/a-tutorial-introduction-to-the-lambda-calculus/
[10] J. Glenn Brookshear (1993) Teoría de la computación. Addison-Wesley. Consulta: 5 de Septiembre de 2013, de http://www.dc.fi.udc.es/~grana/TALF/computabilidad.pdf
[11] EpistemoWikia (2012) Funciones computables. Hiperenciclopédica 6(4). Consulta: 5 de Septiembre de 2013, de http://campusvirtual.unex.es/cala/epistemowikia/index.php?title=Funciones_computables
[12] Ariel Ferreira Szpiniak (1998) Programación avanzada. Universidad Nacional de Rio http://books.google.es/books?id=fBtqpGE_5PYC&pg=PA112&lpg=PA112&dq=currificación+de+funciones&source=bl&ots=CiQxPtwrqR&sig=9VwTx9NHyS42sDXe0nau1YszfEs&hl=es&sa=X&ei=TvwrUtemG8GG2wWf2oCQAg&ved=0CFkQ6AEwCDgK#v=onepage&q=currificación%20de%20funciones&f=false
[13]ECMA (2011) Standard ECMA-262. ECMA Consulta: 5 de Septiembre de 2013, de http://www.ecma-international.org/publications/standards/Ecma-262.htm
[14] Introducción a JSON. Consulta: 5 de Septiembre de 2013, de http://www.json.org/json-es.html
[15]ECMA (1999) Standard ECMA-262. ECMA Consulta: 5 de Septiembre de 2013, de http://www.ecma-international.org/publications/files/ECMA-ST-ARCH/ECMA-262,%203rd%20edition,%20December%201999.pdf