**U1: INTRODUCCIÓN A LAS APLICACIONES WEB Y HTML5**

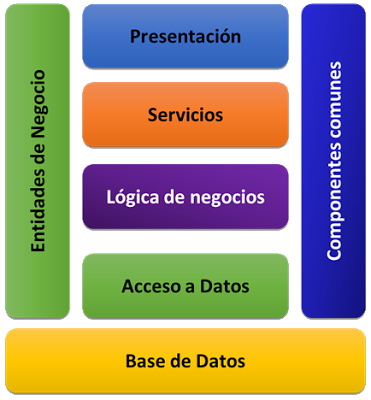
GIT PUTOOOOOOOOOOOOOOOOOO

ARQUITECTURA DE N-CAPAS DE UNA APLICACIÓN WEB



La **arquitectura de N-capas** es probablemente uno de los modelos más utilizados en programación. Se utiliza tan a menudo porque es escalable, extensible, segura, fácil de mantener en el tiempo, reutilizable y se puede trabajar por diferentes programadores según su especialidad sin interferirse el trabajo.

Para el desarrollo de una arquitectura en capas robustas y perdurables en el tiempo, se deben tener en cuenta los siguientes **aspectos**:

* **Las capas de una aplicación deben ser independientes entre sí**. De esta manera podemos reemplazar cualquiera de las capas por una más actualizada en cualquier momento.
* **Se debe mantener el principio de responsabilidad modular**, esto quiere decir que cada capa es responsable de un tema o característica.
* **Principio de caja negra**. Cada componente es independiente de otros y solo interactúa en sus entradas y salidas.
* **Don´t repeat yourself (DRY)**. Cada funcionalidad debe estar una sola vez en el Sistema.
* **Establecer normas en la codificación del desarrollo**. Esto asegura la consistencia de código y el mantenimiento del mismo.

**VENTAJAS**

Siendo la programación en capas una arquitectura que tiene como objetico principal separar la capa de diseño de la lógica, existen otros valores adicionales que aportan a proyectos grandes entre otros como:

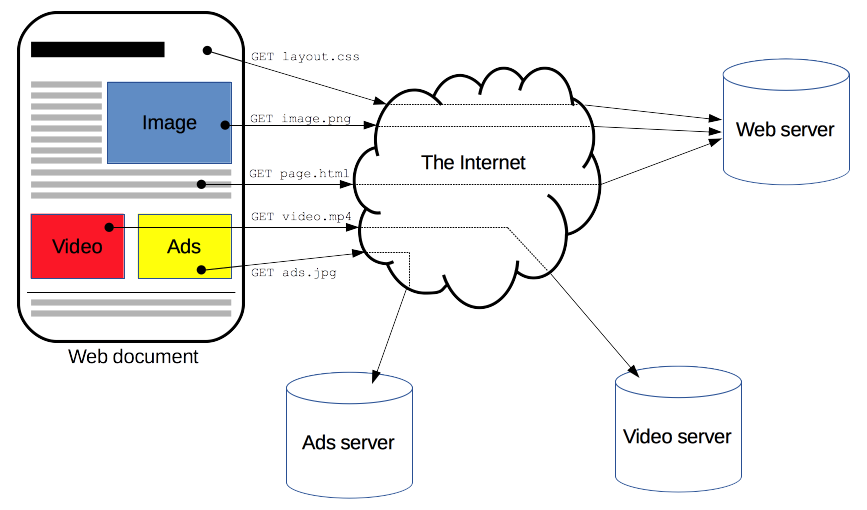
* Facilita la creación de código estructurado.
* Facilita el trabajo en equipo, permitiendo que varios programadores trabajen paralelamente en una misma funcionalidad sin interferirse mutuamente.
* Permite separar algunas de las capas de código en diferentes servidores para balanceo de cargas, buses de datos empresariales, etc.

¿QUÉ ES UNA APLICACIÓN WEB SPA?

Dentro del desarrollo de aplicaciones web hay una tendencia importante a las denominadas **SPA**, Single Page Apps.

Uno de los **principales objetivos** es conseguir una mejora importante en la experiencia de usuario: la mejor de los tiempos de espera o latencia entre vistas (miliar a las aplicaciones nativas).

Un SPA se carga completo durante el cargado de la página inicial y luego las regiones se reemplazan o se actualizan con los fragmentos de las nuevas páginas según petición del servidor. Para evitar descargas excesivas de características inutilizadas. Un SPA descarga progresivamente las características cuando se necesiten, pueden ser fragmentos de las paginas o módulos completos de la pantalla.



De esta forma existe una **analogía entre los "estados" de un SPA y las "páginas" de un sitio web tradicional**. Como la navegación de estados en la misma página es análogo a la navegación de las páginas, en teoría, cualquier página de sitio web podría convertirse a un sitio de página única reemplazando las páginas solamente en las partes donde se generen un cambio.El enfoque de SPA en la página web es similar a Single Document Interface (SDI) que es una técnica para las aplicaciones de escritorio.

**¿QUÉ DIFERENCIA HAY RESPECTO A APLICACIONES WEB CLÁSICAS?**

Habitualmente la lógica de negocio (el código ejecutable) de aplicaciones web se realiza íntegramente en el lado del servidor, y se confía la propia naturaleza del sistema de URLs el mostrar una “vista de aplicación u otra.

Para el navegador, cada URL diferente es completamente independiente del resto. Aunque tenga los mismos estilos y/o plantillas, estos tienen que volver a ser procesados desde cero. Esto, para la gran mayoría de páginas web dinámicas, implica que al cambiar entre vistas se sufrirá el problema de la latencia en la web.

Otra característica negativa es que el estado de la aplicación del cliente es difícil de mantener, teniendo que hacer auténticos malabarismos para poder gestionar una simple trasferencia de información de una vista a otra.

El sistema REST y URLs tiene mucho sentido y aporta muchas otras ventajas desde la misma arquitectura de servicios web, hasta el facilitar el lado humano de compartir un recurso vía un enlace, aunque en ciertos aspectos juegue en nuestra contra.

**ARQUITECTURA BÁSICA**

Un **SPA** es la interfaz de la aplicación web implementada casi íntegramente en el navegador (en JS), auqnue como toda página web tenga una base importante de HTML y CSS.

Todas las vistas de la interfaz de la aplicación están contenidas en la SPA, realizando una única carga inicial y potencialmente solo posponiendo recursos pesados.

Con las SPAs eliminamos por completo uno de los principales cuellos de botella, el **problema de latencia**. Así podemos conseguir que la aplicación web alcance la velocidad de cualquier aplicación nativa en lo que se refiere al tiempo de espera al cambia de vistas.

Podemos ahorrar mucho ancho de banda y tiempo de proceso de cálculo (en servidor y cliente) si gestionamos el estado de la aplicación desde el cliente con un SPA.

**HTML5 Y SPAs**

Se pueden crear SPAs bajo los estándares de HTML4, por tanto, dando cobertura a la totalidad de navegadores utilizados hoy en día que tenga soporte HTML y JS.

Sin embargo, algunas APIs incorporadas en el marco de HTML5 aportan soluciones a ciertos **retos bastantes específicos de las SPAs**:

* **HTML5 History API**: las URLs clásicas eran un problema para las SPAs hasta cierto punto, aunque eso ya no ocurre gracias a esta API, esto a su vez simplificaría en buena medida el problema de “Search Engine Optimisation” ya que con PhtanomJS en tu servidor poder renderizar una URL concreta de tu SPA.
* **HTML5 Application Cache**: si consigues sacarle partido a esta API puedes conseguir que tu aplicación funcione incluso cuando el dispositivo cliente está sin conexión a internet. Toda una killer feature de las SPA.

**HERRAMIENTAS**

Hay muchas herramientas para desarrollar SPA. Por mencionar alguna, AngularJS tiene entre otros muchos objetivos crear aplicaciones SPA y a su vez mantener las ventajas del sistema de URLs.

Como este tipo de aplicaciones son bastante complejas (comparado con el uso previo de Javascript), es mejor realizarlas de forma estructurada a partir de un framework como Angular, siguiendo paradigmas de diseño de software similares a MVC.

HTML

Tim Berners-Lee creó el HTML original en 1989 para solucionar las deficiencias de los métodos existentes para acceder a información en Internet. Desde que se concibió, encontrar su camino en Internet era una tarea difícil. El contenido en Internet era tratado como documentos individuales, sin que hubiese métodos sencillos para navegarlos. En esencia, usted tenía que conocer la dirección del documento que estaba buscando e ingresarla manualmente. Para solucionar este problema, Berners-Lee creó **dos tecnologías**: Hypertext Transfer Protocol (HTTP) y HTML.

**HTTP** es un protocolo de servicio utilizado para entregar contenido. El comienzo de un URL en su navegador Web (suponiendo que el navegador muestre el URL completo) muy probablemente comenzará con http://. Esta parte del URL le dice al navegador qué tipo de protocolo usar cuando esté haciendo la solicitud al servidor Web. Cuando el servidor recibe una solicitud de documento, es probable que ese documento esté escrito o sea convertido a HTML. El documento HTML es lo que se envía de regreso al navegador que hace la solicitud.

**HTML** es un lenguaje de scripting que le dice al navegador Web cómo presentar el contenido. En este contenido puede haber enlaces a otros documentos, proporcionando un método fácil de usar para navegar entre documentos en Internet.

La **combinación de HTTP y HTML** ofrece una navegación rápida y fácil por el contenido en Internet, al permitirle simplemente hacer clic en los enlaces de texto para navegar entre documentos. Después de crear estas dos tecnologías, Berners-Lee continuó y fundó el World Wide Web Consortium (W3C). El **W3C** fue la fuerza guía detrás de las cuatro primeras versiones de HTML.

La **intención original de Internet** era servir documentos de texto simples. Los primeros navegadores todos estaban basados en texto (sin ventanas lujosas — sólo texto en una pantalla). Incluso la adición de imágenes era un gran problema cuando se introdujo al principio. Ahora, las personas hacen de todo, desde enviar mensajes de e-mail hasta ver televisión en Internet. Internet se ha convertido en mucho más que un mecanismo para transportar documentos de texto simples. Con los recursos y usos llegaron nuevos retos y problemas que el lenguaje HTML nunca fue diseñado para manejar.

El **W3C** intentó resolver los problemas del Internet de hoy con el estándar Extensible Hypertext Markup Language (XHTML) 2.0. Sin embargo, este estándar no fue bien recibido y fue abandonado en gran medida. En el 2004, mientras el W3C se estaba enfocando en el estándar XHMTL 2.0, un grupo llamado el Web Hypertext Application Technology Working Group (WHATWG) comenzó a trabajar en el estándar HTML5, que tuvo una acogida más cálida que el estándar XHTML 2.0. El W3C abandonó el estándar XHTML 2.0 y está trabajando ahora con WHATWG en el desarrollo del HTML5.1.

**¿QUÉ ES HTML?**

Es el bloque de construcción más básico de la web.

* Describe y define el contenido de una página web.
* Para describir la apariencia de una página se utilizan hojas de estilo CSS.
* Para agregarle funcionalidad se utiliza el lenguaje JS.
* HyperText se refiere a vínculos o links que conectan una página a otra.
* Utiliza un lenguaje de marcas o “markup” para especificar el texto a mostrar, imágenes, videos y cualquier otro contenido para mostrar en el navegador web.

**¿QUÉ ES UN SERVIDOR WEB?**

* Es un programa que se ejecuta en un equipo servidor que procesa peticiones HTTP generalmente por el puerto 80 y devuelve documentos HTML.
* Posee una estructura de directorios y archivos que constituyen un sitio web
* Los **servidores web más utilizados** son:
  + Internet Information Services (IIS10),
  + Apache, NGINX, GWS
* **Lenguajes más utilizados**:
  + ASP.NET Web/API, MVC, WebForms
  + PHP
  + Java
  + JavaScript/Typescript
  + Python

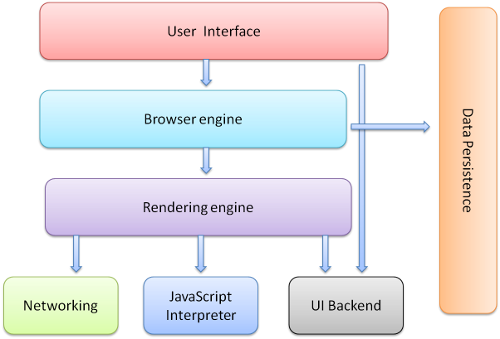
**NAVEGADORES WEB – COMPONENTES**

La **función principal** de un navegador es solicitar al servidor los recursos web que elija el usuario y mostrarlos en una ventana. El recurso suele ser un documento HTML, pero también puede ser un archivo PDF, una imagen o un objeto de otro tipo. El usuario especifica la ubicación del recurso mediante el uso de una URI (siglas de Uniform Resource Identifier, identificador uniforme de recurso).

La **forma en la que el navegador interpreta y muestra los archivos HTML** se determina en las especificaciones de CSS y HTML. Estas especificaciones las establece el consorcio W3C (World Wide Web Consortium), que es la organización de estándares de Internet.

Durante años, los navegadores cumplían solo una parte de las especificaciones y desarrollaban sus propias extensiones. Esto provocó graves problemas de compatibilidad para los creadores de contenido web. En la actualidad, la mayoría de los navegadores cumplen las especificaciones en mayor o menor grado.

Los **componentes principales** de un navegador son:

* **Interfaz de usuario**: incluye la barra de direcciones, el botón de avance/retroceso, el menú de marcadores, etc. (en general, todas las partes visibles del navegador, excepto la ventana principal donde se muestra la página solicitada).
* **Motor de búsqueda**: coordina las acciones entre la interfaz y el motor de renderización.
* **Motor de renderización**: es responsable de mostrar el contenido solicitado. Por ejemplo, si el contenido solicitado es HTML, será el responsable de analizar el código HTML y CSS y de mostrar el contenido analizado en la pantalla.
* **Red**: es responsable de las llamadas de red, como las solicitudes HTTP. Tiene una interfaz independiente de la plataforma y realiza implementaciones en segundo plano para cada plataforma.
* **Servidor de la interfaz**: permite presentar widgets básicos, como ventanas y cuadros combinados. Muestra una interfaz genérica que no es específica de ninguna plataforma. Utiliza métodos de la interfaz de usuario del sistema operativo en segundo plano.
* **Intérprete de JavaScript**: permite analizar y ejecutar el código JavaScript.
* **Almacenamiento de datos**: es una capa de persistencia. El navegador necesita guardar todo tipo de datos en el disco duro (por ejemplo, las cookies). La nueva especificación de HTML (HTML5) define el concepto de "base de datos web", que consiste en una completa (aunque ligera) base de datos del navegador.

El motor de renderización

La responsabilidad del motor de renderización es "renderizar", es decir, mostrar el contenido solicitado en la pantalla del navegador.

De forma predeterminada, el motor de renderización puede mostrar imágenes y documentos HTML y XML. Puede mostrar otros tipos mediante el uso de complementos (o extensiones); por ejemplo, puede mostrar documentos PDF mediante un complemento capaz de leer archivos PDF.

Nuestros navegadores de referencia (Firefox, Chrome y Safari) están basados en dos motores de renderización. Firefox utiliza **Gecko**, un motor de renderización propio de Mozilla. Tanto Safari como Chrome utilizan **WebKit**.

**Trident** es el nombre del motor de renderizado propietario usado por Microsoft Internet Explorer 11 para Windows.

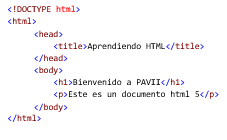
**EdgeHTML** es un motor de renderizado desarrollado por Microsoft para el navegador Microsoft Edge.

**WebKit** es un motor de renderización de código abierto que empezó siendo un motor de la plataforma Linux y que fue modificado posteriormente por Apple para hacerlo compatible con Mac y Windows.

**NUEVAS CARACTERÍSTICAS DE HTML5**

* **Semántica**: Permite describir con mayor precisión cuál es su contenido y definir elementos propios de html como así también atributos.
* **Conectividad**: Permite comunicarse con el servidor de formas innovadoras.
* **Sin conexión y almacenamiento**: Permite a las páginas web almacenar datos localmente en el lado del cliente y operar sin conexión de manera más eficiente.
* **Multimedia**: soporte contenido multimedia de audio y video nativamente.
* **Gráficos y efectos 2D/3D**: Incorpora nuevas características que se ocupan de los gráficos en la web como lo son canvas 2D, WebGL, SVG, etc.
* **Rendimiento e Integración:** Proporciona una mayor optimización de la velocidad y un mejor uso del hardware.
* **Acceso al dispositivo**: Proporciona APIs para el uso de varios componentes internos de entrada y salida de nuestro dispositivo, tales como geolocalización.
* **CSS3:** Gran variedad de opciones para hacer diseños más sofisticados.

**ESTRUCTURA DE UN DOCUMENTO HTML5**

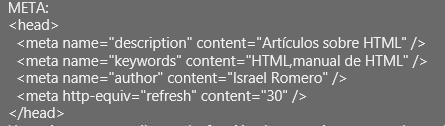
* La declaración **<!DOCTYPE html>** define el tipo de documento.
* El texto entre **<html> y </html>** describe el documento HTML.
* El texto entre **<head></head>** proporciona información sobre el documento.
* El texto entre **<body> y </body>** describe el contenido de la página visible en el navegador.

**PARA QUÉ SIRVE EL TAG <HEAD> Y <BODY>**

El **HEAD** es la parte donde se incluye la información acerca del documento, es una sección de un documento HTML meramente "técnica e informativa", pues la mayoría de esta información no la muestra el navegador al usuario e inclusive pudiéramos dejarla vacía y esto no afectaría al funcionamiento o la forma en que se visualiza la página, y si bien el HEAD de un documento HTML pudiera ir vacío siempre es mejor darles la suficiente importancia a las etiquetas que el HEAD contiene, mucho más aún si nuestro objetivo es publicar nuestro trabajo en la web, pues muchas de las etiquetas del HEAD son importantes para los buscadores y para un buen posicionamiento en los resultados de búsqueda.

* **Head**:
  + Es parte fundamental de la estructura de un documento html.
  + Se usa siempre.
  + No se puede poner dentro de la etiqueta <body>.
  + Se incluye información acerca del documento.
  + Sección meramente técnica e informativa.
  + Dentro de esta sección **se pueden establecer**: Título de la página, Configuraciones de Meta Tags, Estilos CSS, Scripts (javascript y/o enlaces a archivos).
* **Body**:
  + Representa el contenido de un documento HTML.
  + Sólo puede haber uno por documento.

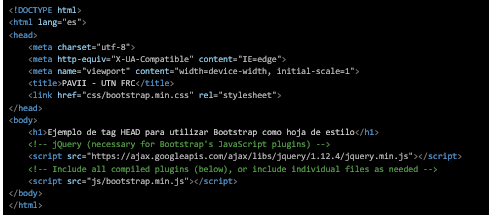
Etiqueta con atributos

La función de esta etiqueta va a depender totalmente de los atributos y valores que contenga, en nuestro ejemplo las primeras 3 tienen los atributos "name" y "content" pero tienen valores diferentes, la **primera** (con valor "description") indica la descripción de la página, la **segunda** indica las palabras clave con la que los buscadores deberían de relacionarla, en el ejemplo se pretende que los buscadores muestren la página en sus resultados cuando alguien busque "HTML" o "manual de HTML", la **tercera** incluye el nombre del autor de la página. Ahora la **cuarta** etiqueta tiene atributos diferentes, en este caso esta etiqueta le dice al navegador que la página se debe actualizar cada 30 segundos.

Ejemplo tag <head> y <body>

**<meta charset="utf-8">** : Especifica el juego de caracteres (codificación) de la página html  
Valores comunes:

* UTF-8 - Codificación de caracteres para Unicode.
* ISO-8859-1 - Codificación de caracteres para el alfabeto latino.
* En teoría, cualquier codificación de caracteres se puede utilizar, pero no todos los navegadores entienden todas las codificaciones.

**<meta http-equiv="X-UA-Compatible" content="IE=edge">** El **atributo content** específica el modo de la página; por ejemplo, para imitar el comportamiento de Windows Internet Explorer 7, especifique IE=EmulateIE7. Del mismo modo, especifique IE=5, IE=7 o IE=8 para seleccionar uno de estos modos de compatibilidad. También puede especificar IE=edge para indicar a Windows Internet Explorer 8 o 10 que use el máximo modo disponible.

El **encabezado X-UA-compatible** no distingue entre mayúsculas y minúsculas; no obstante, debe aparecer en el encabezado de la página web (la sección HEAD) antes que todos los demás elementos, excepto el elemento TITLE y otros elementos META.

**<meta name="viewport" content="width=device-width, initial-scale=1">** La **etiqueta meta** para el viewport fue introducida por Apple en Safari para móviles, originalmente se utilizó para ayudar a los desarrolladores a mejorar la presentación de sus aplicaciones web en un iPhone, iPod Touch o iPad.

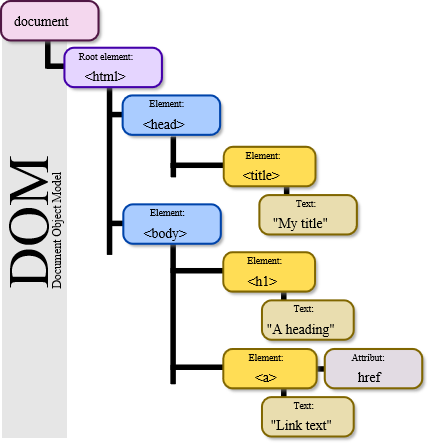
La **etiqueta viewport** nos permite a los que construimos sitios web o web apps, definir el ancho, alto y escala del área usada por el navegador para mostrar contenido.

Al fijar el ancho o alto del viewport, los desarrolladores podemos usar un número fijo de pixeles (ej: 320, 480, etc) o usar dos constantes, **device-width** y **device height** respectivamente.Se considera una buena práctica configurar el viewport con algunas de estas dos constantes, en vez de un ancho o alto fijo.  
 **<link href="css/bootstrap.min.css" rel="stylesheet">** Esta etiqueta permite enlazar un archivo JavaScript u Hoja de estilo (css) externos con el documento actual. El **atributo rel=”stylesheet”** especifica que el archivo **css/bootstrap.min.css** es de hoja de estilo, y está ubicado en el directorio css del árbol de directorios del sitio web.

**<script src="**[**https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js**](https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js)**"></script>** Script javascript que está en otro sitio web. En html5 ya no se usa el atributo type=”text/javascript”. En este caso se posee una referencia a la librería de JQuery que necesita bootstrap.

El **tag <script>** puede estar en el <head> o a finalizar el <body> o en ambos lugares.   
La colocación de secuencias de comandos en la parte inferior del elemento <body> mejora la velocidad de visualización, porque la compilación de secuencias de comandos disminuye la visualización.

DOM DE UN DOCUMENTO HTML

El DOM es una W3C (World Wide Web Consortium) estándar. El DOM define un estándar para acceso a los mismos:

"El modelo de objetos de documento del W3C (DOM) es una interfaz de plataforma y lenguaje neutro que permite a los programas y scripts acceder y actualizar el contenido, la estructura y el estilo de un documento de forma dinámica."

El estándar W3C DOM se separa en 3 **partes diferentes**:

* **Núcleo del DOM** - modelo estándar para todos los tipos de documentos
* **DOM XML** - modelo estándar para documentos XML
* **HTML DOM** - modelo estándar para documentos HTML

El **DOM HTML** es un estándar de objetos de modelo y de programación de interfaces para HTML. Se define:

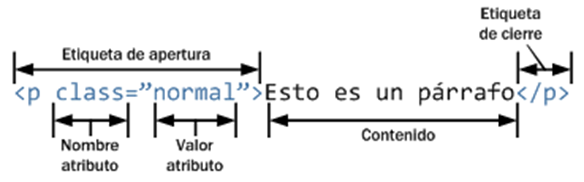
* Los elementos HTML como objetos.
* Las propiedades de todos los elementos HTML.
* Los métodos para acceder a todos los elementos HTML.
* Los eventos de todos los elementos HTML.

El **DOM HTML** es un estándar de cómo obtener, cambiar, añadir o eliminar elementos HTML.

Con el modelo de objetos, JavaScript posee toda la **funcionalidad necesaria para crear HTML dinámico**:

* Puede alterar todos los elementos HTML en la página.
* Puede modificar todos los atributos de HTML en la página.
* Permite cambiar todos los estilos CSS en la página.
* Es posible eliminar elementos y atributos HTML existente.
* Contiene la funcionalidad para agregar nuevos elementos y atributos HTML.
* Permite implementar los eventos de HTML existentes en la página.
* Se puede agregar nuevos manejadores de eventos en la página HTML.

DEFINICIÓN DE ELEMENTO DE UN DOCUMENTO HTML



Los elementos son los componentes fundamentales del HTML. Cuentan con 2 **propiedades básicas**:

* Atributos
* Contenido

En general se conforman con una Etiqueta de Apertura y otra Cierre.

Los **atributos** se colocan dentro la etiqueta de apertura, y el **contenido** se coloca entre la etiqueta de apertura y la de cierre.

Todos los elementos del estándar HTML5 están listados aquí, descritos por su etiqueta de apertura y agrupados por su función.

Esta lista solamente los elementos válidos de HTML5. Solamente aquellos elementos listados aquí son los que deberían ser usados en nuevos sitios Web.

El simbolo **This element was added as part of HTML5** indica que el elemento fue añadido en HTML5. Nótese que otros elementos listados aquí pueden haber sido modificados o extendido en su significado por la especificación HTML5.

**Elemento raíz**

***<! doctype html>*** Define que el documento está bajo el estándar de HTML 5

***<html>*** Representa la raíz de un documento HTML o XHTML. Todos los demás elementos deben ser descendientes de este elemento.

**Metadatos del documento**

**<head>** Representa una colección de metadatos acerca del documento, incluyendo enlaces a, o definiciones de, scripts y hojas de estilo.

***<title>*** Define el título del documento, el cual se muestra en la barra de título del navegador o en las pestañas de página. Solamente puede contener texto cualquier otra etiqueta contenida no será interpretada.

***<base>*** Define la URL base para las URLs relativas en la página.

***<link>*** Usada para enlazar JavaScript y CSS externos con el documento HTML actual.

***<meta>*** Define los metadatos que no pueden ser definidos usando otro elemento HTML.

***<style>*** Etiqueta de estilo usada para escribir CSS en línea.

**Scripting**

***<script>*** Define ya sea un script interno o un enlace hacia un script externo. El lenguaje de programación es JavaScript

***<noscript>*** Define un contenido alternativo a mostrar cuando el navegador no soporta scripting.

**Secciones**

***<body>*** Representa el contenido principal de un documento HTML. Solo hay un elemento <body> en un documento.

***<section>*** (HTML5) Define una sección en un documento.

***<nav>*** (HTML5) Define una sección que solamente contiene enlaces de navegación

***<article>*** (HTML5) Define contenido autónomo que podría existir independientemente del resto del contenido.

***<aside>*** (HTML5) Define algunos contenidos vagamente relacionados con el resto del contenido de la página. Si es removido, el contenido restante seguirá teniendo sentido

***<h1>, <h2>,***

***<h3>, <h4> ,*** Los elemento de cabecera implementan seis niveles de cabeceras de documentos; <h1> es la de mayor y <h6> es la de menor importancia.

***<h5>,<h6>***

Un **elemento** de cabecera describe brevemente el tema de la sección que introduce.

***<header>*** (HTML5) Define la cabecera de una página o sección. Usualmente contiene un logotipo, el título del sitio Web y una tabla de navegación de contenidos.

***<footer>*** (HTML5) Define el pie de una página o sección. Usualmente contiene un mensaje de derechos de autoría, algunos enlaces a información legal o direcciones para dar información de retroalimentación.

***<address>*** Define una sección que contiene información de contacto.

***<main>*** (HTML5) Define el contenido principal o importante en el documento. Solamente existe un elemento <main> en el documento.

**Agrupación de Contenido**

***<p>*** Define una parte que debe mostrarse como un párrafo.

***<hr>*** Representa un quiebre temático entre párrafos de una sección o articulo o cualquier contenido.

***<pre>*** Indica que su contenido esta reformateado y que este formato debe ser preservado.

***<blockquote>*** Representa un contenido citado desde otra fuente.

***<ol>*** Define una lista ordenada de artículos.

***<ul>*** Define una lista de artículos sin orden.

***<li>*** Define un artículo de una lista enumerada.

***<dl>*** Define una lista de definiciones, es decir, una lista de términos y sus definiciones asociadas.

***<dt>*** Representa un término definido por el siguiente <dd>.

***<dd>*** Representa la definición de los términos listados antes que él.

***<figure>*** (HTML5) Representa una figura ilustrada como parte del documento.

***<figcaption>*** (HTML5) Representa la leyenda de una figura.

***<div>*** Representa un contenedor genérico sin ningún significado especial.

**Semántica a nivel de Texto**

**<a>** Representa un hiperenlace, enlazando a otro recurso.

***<em>*** Representa un texto enfatizado, como un acento de intensidad.

***<strong>*** Representa un texto especialmente importante.

***<small>*** Representa un comentario aparte, es decir, textos como un descargo de responsabilidad o una nota de derechos de autoría, que no son esenciales para la comprensión del documento.

***<s>*** Representa contenido que ya no es exacto o relevante.

***<cite>*** Representa el título de una obra.

***<q>*** Representa una cita textual inline.

***<dfn>*** Representa un término cuya definición está contenida en su contenido ancestro más próximo.

***<abbr>*** Representa una abreviación o un acrónimo; la expansión de la abreviatura puede ser representada por el atributo title.

***<data>*** (HTML5) Asocia un equivalente legible por máquina a sus contenidos. (Este elemento está solamente en la versión de la WHATWG del estándar HTML, y no en la versión de la W3C de HTML5).

***<time>*** (HTML5) Representa un valor de fecha y hora; el equivalente legible por máquina puede ser representado en el atributo datetime.

***<code>*** Representa un código de ordenador.

***<var>*** Representa a una variable, es decir, una expresión matemática o contexto de programación, un identificador que represente a una constante, un símbolo que identifica una cantidad física, un parámetro de una función o un marcador de posición en prosa.

***<samp>*** Representa la salida de un programa o un ordenador.

***<kbd>*** Representa la entrada de usuario, por lo general desde un teclado, pero no necesariamente, este puede representar otras formas de entrada de usuario, como comandos de voz transcritos.

***<sub>, <sup>*** Representan un subíndice y un superíndice, respectivamente.

***<i>*** Representa un texto en una voz o estado de ánimo alterno, o por lo menos de diferente calidad, como una designación taxonómica, un término técnico, una frase idiomática, un pensamiento o el nombre de un barco.

**<b>** Representa un texto hacia el cual se llama la atención para propósitos utilitarios. No confiere ninguna importancia adicional y no implica una voz alterna.

***<u>*** Representa una anotación no textual sin-articular, como etiquetar un texto como mal escrito o etiquetar un nombre propio en texto en chino.

***<mark>*** (HTML5) Representa texto resaltado con propósitos de referencia, es decir por su relevancia en otro contexto.

***<ruby>*** (HTML5) Representa contenidos a ser marcados con anotaciones ruby, recorridos cortos de texto presentados junto al texto. Estos son utilizados con regularidad en conjunto a lenguajes de Asia del Este, donde las anotaciones actúan como una guía para la pronunciación, como el furigana japonés.

***<rt>*** (HTML5) Representa el texto de una anotación Ruby.

***<rp>*** (HTML5) Representa los paréntesis alrededor de una anotación ruby, usada para mostrar la anotación de manera alterna por los navegadores que no soporten despliegue estándar para las anotaciones.

***<bdi>*** (HTML5) Representa un texto que debe ser aislado de sus alrededores para el formateado bidireccional del texto. Permite incrustar un fragmento de texto con una direccionalidad diferente o desconocida.

***<bdo>*** Representa la direccionalidad de sus descendientes con el fin de anular de forma explícita al algoritmo bidireccional Unicode.

***<span>*** Representa texto sin un significado específico. Este debe ser usado cuando ningún otro elemento semántico le confiere un significado adecuado, en cuyo caso, provendrá de atributos globales como class, lang, o dir.

***<br>*** Representa un salto de línea.

***<wbr>*** (HTML5) Representa una oportunidad de salto de línea, es decir, un punto sugerido de envoltura donde el texto de múltiples líneas puede ser dividido para mejorar su legibilidad.

**Ediciones**

***<ins>*** Define una adición en el documento.

***<del>***  Define una remoción del documento.

**Contenido incrustado**

***<img>*** Representa una imagen.

***<iframe>*** Representa un contexto anidado de navegación, es decir, un documento HTML embebido.

***<embed>*** (HTML5) Representa un punto de integración para una aplicación o contenido interactivo externo que por lo general no es HTML.

***<object>*** Representa un recurso externo, que será tratado como una imagen, un sub-documento HTML o un recurso externo a ser procesado por un plugin.

***<param>*** Define parámetros para el uso por los plugins invocados por los elementos. ***<object>***

***<video>***  (HTML5) Representa un video, y sus archivos de audio y capciones asociadas, con la interfaz necesaria para reproducirlos.

***<audio>*** (HTML5) Representa un sonido o stream de audio.

***<source>*** (HTML5) Permite a autores especificar recursos multimedia alternativos para los elementos multimedia como <video> o <audio>.

***<track>*** (HTML5) Permite a autores especificar una pista de texto temporizado para elementos multimedia como <video> o <audio>.

***<canvas>*** (HTML5) Representa un área de mapa de bits en el que se pueden utilizar scripts para renderizar gráficos como gráficas, gráficas de juegos o cualquier imagen visual al vuelo.

***<map>*** En conjunto con <area>, define un mapa de imagen.

***<area>*** En conjunto con <map> , define un mapa de imagen.

***<svg>*** (HTML5) Define una imagen vectorial embebida.

***<math>*** (HTML5) Define una fórmula matemática.

**Datos tabulares**

***<table>*** Representa datos con más de una dimensión.

***<caption>*** Representa el título de una tabla.

***<colgroup>*** Representa un conjunto de una o más columnas de una tabla.

***<col>*** Representa una columna de una tabla.

***<tbody>*** Representa el bloque de filas que describen los datos concretos de una tabla.

***<thead>*** Representa el bloque de filas que describen las etiquetas de columna de una tabla.

***<tfoot>*** Representa los bloques de filas que describen los resúmenes de columna de una tabla.

***<tr>*** Representa una fila de celdas en una tabla.

***<td>*** Representa una celda de datos en una tabla.

***<th>*** Representa una celda encabezado en una tabla.

**Formularios**

***<form>*** Representa un formulario, consistiendo de controles que puede ser enviado a un servidor para procesamiento.

***<fieldset>*** Representa un conjunto de controles.

***<legend>*** Representa el título de un <fieldset>.

***<label>***  Representa el título de un control de formulario.

***<input>*** Representa un campo de datos escrito que permite al usuario editar los datos.

***<button>*** Representa un botón.

***<select>*** Representa un control que permite la selección entre un conjunto de opciones.

***<datalist>*** (HTML5) Representa un conjunto de opciones predefinidas para otros controles.

***<optgroup>*** Representa un conjunto de opciones, agrupadas lógicamente.

***<option>*** Representa una opción en un elemento <select>, o una sugerencia de un elemento <datalist>.

***<textarea>*** Representa un control de edición de texto multi-línea.

***<keygen>*** (HTML5) Representa un control de par generador de llaves.

***<output>*** (HTML5) Representa el resultado de un cálculo.

***<progress>*** (HTML5) Representa el progreso de finalización de una tarea.

***<meter>*** (HTML5) Representa la medida escalar (o el valor fraccionario) dentro de un rango conocido.

**Elementos interactivos**

***<details>*** (HTML5) Representa un widget desde el que un usuario puede obtener información o controles adicionales.

***<summary>*** (HTML5) Representa un resumen, título o leyenda para un elemento <details> dado.

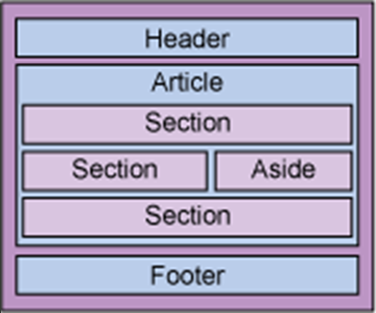
***<command>*** (HTML5) Representa un comando que un usuario puede invocar.

***<menu>*** (HTML5) Representa una lista de comandos.

RECOMENDACIONES PARA ESCRITURA DE HTML

* Uso minúsculas en nombres de elementos
* HTML5 permite mezclar letras mayúsculas y minúsculas en nombres de elementos. Se recomienda utilizar nombres de elementos minúsculas porque:  
  mezclar mayúsculas y minúsculas los nombres es una mala práctica en HTML
* El texto en minúsculas es más limpio y fácil de leer
* Encerrar los valores de los atributos siempre con comilla doble
* Evitar espacios entre el signo igual (=) en la especificación del atributo= “valor” en los elementos de HTML.

ELEMENTOS ESTRUCTURALES DE HTML

La **razón para crear nuevas etiquetas estructurales** es dividir las páginas Web en partes lógicas que describan el tipo de contenido que incluyen. Conceptualmente, piense en la página Web como un documento. Los documentos tienen encabezados, pies de página, capítulos y otras convenciones diferentes que dividen el documento en partes lógicas.

HTML5 responde a este problema proporcionando un conjunto de etiquetas que definen con mayor claridad los bloques principales de contenido que componen un documento HTML. Sin importar el contenido final mostrado por la página Web, la mayoría de páginas Web consisten en combinaciones de variantes de secciones y elementos de página comunes.

La **etiqueta header** tiene por objeto marcar una sección de la página HTML como el encabezado.

La **etiqueta section** tiene por objeto identificar porciones significativas del contenido de la página. Esta etiqueta es de alguna forma análoga a dividir un libro en capítulos.

La **etiqueta article** identifica las secciones principales del contenido dentro de la página Web. Piense en un blog, donde cada publicación de cada individuo constituye una porción significativa de contenido.

La **etiqueta aside** indica que el contenido dentro de ella está relacionado el contenido principal de la página pero que no es parte de ella. En cierta forma es análogo a usar paréntesis para hacer un comentario en un cuerpo de texto (como este). El contenido entre paréntesis proporciona información adicional sobre el elemento que lo contiene.

La **etiqueta footer** marca el contenido dentro del elemento que es el pie de página del documento.

El contenido dentro de la **etiqueta nav** tiene por objeto propósitos de navegación.

HTML LISTAS

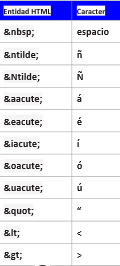
Permiten estructurar mejor el texto y para ordenar la información. Hay dos **tipos de lista:**

* **Listas no ordenadas o listas de viñetas.**

***<ul><li></li></ul>***

* **Listas ordenadas o numeradas.**

**<ol><li></li></ol>**

CARACTERES ESPECIALES DE TEXTO EN HTML

En HTML los caracteres propios de los idiomas diferentes al inglés pueden ser problemáticos.

La correcta visualización depende de la codificación del documento UTF-8, del editor de HTML, y del servidor.

Para evitar problemas de visualización con los caracteres reservados de HTML se debe utilizar el carácter de entidad HTML.

Las entidades son unas estructuras que, mediante el uso de una codificación, nos permiten representar un símbolo.

La estructura de la entidad HTML es un ampersand (&) seguido del código o nombre de la entidad y terminado en un punto y coma.

HTML – IMÁGENES

Se especifican los siguientes **atributos**:

* El archivo a visualizar (src).
* texto alternativo, o para personas con discapacidad visual (alt).
* width, y height.

***<img src=" " alt=" " width="" height="">***

HTML – ENLACES

Define un hiper vínculo a una ubicación en la misma página o cualquier otra página en la Web.

***<a href=" "> <img src=" " alt=" "> </a>***

HTML – TABLAS

* **<table>** indica que se debe renderizar una tabla.
* Cada fila es un tag **<tr>** (table row).
* La cabecera se especifica como **<th>.**
* Cada celda es un tag **<td>** (table data).
* El atributo **colspan** indica que una celda va a ocupar “n” columnas. Por defecto colspan=”1”.

HTML – ELEMENTOS DE LÍNEA Y DE BLOQUE

* **Elementos inline**: Se posiciona horizontalmente en línea con los otros elementos.
  + La altura y el ancho se define en base al contenido que posea.
  + Solo puede contener elementos de tipo inline.
  + No se puede aplicar una anchura y una altura fija por medio de CSS.
  + Solo ocupan el espacio necesario para mostrar sus contenidos
  + Ej.: <a>, <br>, <span>, <img>, <i>, <b>, <big>, <small>, <u>, <s>, <em>, <strong>, <input>, <select>, <textarea>, <label>, <button>
* **Elementos block**: Forma un bloque y se posiciona de forma vertical con un nuevo salto de línea.
  + La anchura es la máxima que puede tomar dentro de su elemento contenedor (padre)
  + La altura cambia en base al contenido que posea.
  + Puede contener otros elementos de tipo inline y block
  + Por medio de CSS se le puede aplicar una anchura y una altura fija.
  + Ej: <p>, <h1>, <h2>, <h3>, <h4>, <h5>, <h6>, <div> <ul>, <ol>, <li>, <table>, <form>

HTML – ETIQUETA <DIV>

* Define un bloque de contenido o sección de la página.
* Puede contener otros elementos tipo block o inline de html.
* Muy utilizado para el maquetado de la página y para aplicar hojas de estilo.
* El atributo style="color:red" especifica el color.

***<div style=”color:red”></div>***

MINIFICAR RECURSOS (HTML, CSS, JS)

La **minificación de recursos** se refiere a la eliminación de bytes innecesarios, como los espacios adicionales, saltos de línea y sangrías. Al minimizar los códigos HTML, CSS y JavaScript, es posible acelerar la descarga, el análisis y el tiempo de ejecución. Además, en CSS y en JavaScript, es posible reducir aún más el tamaño del archivo al cambiar el nombre de las variables, siempre y cuando el código HTML esté actualizado correctamente para garantizar que los selectores sigan funcionando.

En JavaScript y CSS los archivos minificados aparecen con la **extensión .min.js** o **.min.css***.* Existen varios **minificadores online y analizadores** como:

* PageSpeed Insights de google
* cssmin.js
* [www.minifier.org](http://www.minifier.org)
* Closure Compiler

**Recomendaciones**:

* Debes minificar el código HTML, CSS y JavaScript.
* Para minificar HTML, puedes utilizar la [extensión de PageSpeed Insights para Chrome](https://developers.google.com/speed/docs/insights/using_chrome) con la que generar una versión optimizada del código HTML. Analiza tu página HTML y busca la regla "Minificar HTML". Haz clic en "Ver el contenido optimizado" para obtener el código HTML optimizado.
* Para minificar CSS, usa [YUI Compressor](http://developer.yahoo.com/yui/compressor/) y [cssmin.js](http://www.phpied.com/cssmin-js/).
* Para minificar JavaScript, prueba [Closure Compiler](https://developers.google.com/closure/compiler), [JSMin](http://www.crockford.com/javascript/jsmin.html) o [YUI Compressor](http://developer.yahoo.com/yui/compressor/). Se puede crear un proceso de construcción que utilice estas herramientas para minificar y cambiar el nombre de los archivos de desarrollo, y guardarlos en un directorio de producción.

HTML – FORMULARIOS

Es un elemento de html definido por la etiqueta <form> que permite agrupar múltiples elementos de ingreso de datos y botones. **Atributos más utilizados:**

* **method**: indica cómo se envían los valores del formulario. Los métodos permitidos son POST y GET.
  + **POST** Envía los datos de manera oculta.
  + **GET** envía los datos agregándolos a la dirección URL y los separa de la URL base con un signo de interrogación**.**
* **action**: Indica la ubicación a la cual será enviada la información. Puede ser un correo electrónico un script, inclusive, otra página HTML.

Sólo se enviará la información de los elementos de ingreso de datos

***<form method=”” action=””></form>***

Dentro de un formulario se pueden insertar cualquier elemento HTML en como texto, botones, tablas y enlaces, pero los elementos interactivos son los más interesantes.

HTML – ELEMENTOS DE INGRESO DE DATOS

**Full compatible**

* input (previo a html5).
  + Text
  + Password
  + Button
  + Submit
  + Reset.
  + Image.
  + Hidden.
  + Radio.
  + Checkbox.
* Textarea
* Button
* Select

**Parcialmente compatible**

* Input
  + Color
  + Date, datetime, datetime-local, month
  + Email
  + Number
  + Range
  + Search
  + Tel
  + Time
  + url
  + week

HTML – ELEMENTO <INPUT>

Permite ingreso de datos en un cuadro de texto.

***<input type="text" name="Nombre" required maxlength="45" />***

Para el ingreso de contraseña.

***<input type="password" name="Clave" placeholder="Clave" value=""/>***

Para establecer valores no visibles al usuario pero que se envían al servidor.

***<input type="hidden" value="valor oculto" name="id" />***

Permite definir un botón que es una imagen.

***<input type="image" src="img\_submit.gif" alt="Submit">***

Botón para envío de datos comprendidos entre <form> y </form>

***<input type="submit" name="Enviar" value="Enviar datos" />***

Botón para la ejecución de acciones en JS.

***<input type="button" name="alerta" value="Mostrar alerta" onclick="alert('Alerta!!!!')" />***

Otro tipo de botón.

***<button type="button">Click aquí!!!</button>***

Dentro de un **elemento <button>** puedes poner contenido, como texto o imágenes. Esta es la diferencia entre este elemento y los botones creados con el elemento <input>. El type puede ser button,reset,submit

**Atributos**:

**type**:

* **text**: texto claro
* **password**: El ingreso de los caracteres aparece en la interfaz del navegador de manera oculta, pero a través del DOM se puede acceder al valor que contiene

**name**: identificador para trabajar con el DOM

**required**: requerido

**maxlength**: cantidad máxima de caracteres

**placeholder**: El atributo de placeholder especifica una sugerencia que describe el valor esperado de un campo de entrada (un valor de muestra o una breve descripción del formato). La indicación se muestra en el campo de entrada antes de que el usuario introduzca un valor. Funciona con los siguientes tipos de entrada: texto, búsqueda, url, tel, correo electrónico y contraseña.

**value:** valor inicial que contiene el control

**readonly**: solo lectura, no permite edición

**disabled**: deshabilitado, no se permite 3edición, pero el valor se envía con el formulario

**size:** El atributo size especifica el tamaño (en caracteres) del campo de entrada.

**Atributos nuevos en HTML 5**

**pattern:** El atributo pattern especifica una expresión regular que comprueba el valor del elemento <input>.  
El atributo patrón funciona con los siguientes tipos de entrada: texto, búsqueda, url, tel, correo electrónico y contraseña.

HTML - <INPUT> DE BOTONES DE SELECCIÓN Y CHECKBOX

* **Radio**: establecer el mismo identificador para el **atributo nam**e para funcionamiento en conjunto mutuamente excluyente. **Atributo checked**: establece de manera predeterminada la opción seleccionada

***<input type=”radio” name=”” value=”” checked>***

* **Checkbox**: permite seleccionar algún element o ninguno.

***<input type=”checkbox” name=”” value=”” checked>***

HTML - <TEXTAREA>

***<textarea name="" cols="" rows="" placeholder=" "></textarea>***

El **atributo rows** especifica el número visible de líneas en un área de texto.

El **atributo cols** especifica el ancho visible de un área de texto.

HTML – ELEMENTO <SELECT>

El atributo **selected** permite establecer cuál es la opción seleccionada de manera predeterminada.

El atributo **value** es el que valor que va a tomar el elemento <select> cuando se seleccione la opción del cuadro de lista.

***<select name=””> <option value=””></option></select>***

**Otros atributos**:

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Value** | **Descripción** |
| Autofocus | Autofocus | Specifies that the drop-down list should automatically get focus when the page loads. |
| Disable | Disable | Specifies that a drop-down list should be disabled. |
| Form | Form\_id | Defines one or more forms the select field belongs to. |
| Multiple | multiple | Specifies that multiple options can be selected at once |
| Name | Name | Defines a name for the drop-down list |
| Required | required | Specifies that the user is required to select a value before submitting the form |
| Size | number | Defines the number of visible options in a drop-down list |

HTTP – PROTOCOLO (Hypertext Transfer Protocol)

HTTP es el conjunto de reglas que gobiernan el formato y el contenido de una conversación entre un cliente web y un servidor.

**Stateless**: cada transacción entre el cliente y el servidor es independiente, es decir, no guarda ninguna información sobre conexiones anteriores.

Para mantener el estado de la aplicación web se usan las cookies, que permiten instituir la noción de sesión, y también permite rastrear usuarios

HTTP ha sido diseñado como un **protocolo sin estado** (statetless protocol) lo que significa que cada solicitud (request) o respuesta (Response) es una transacción independiente

Las peticiones al servidor se denominan **request** y las respuestas del servidor **responses** .

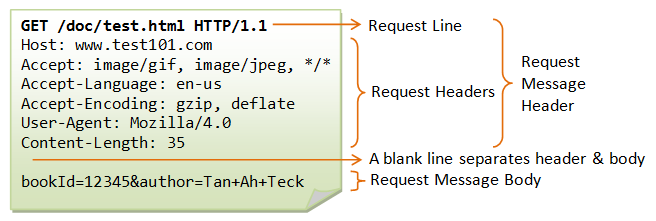
* Especifica las reglas de comunicación entre un cliente web y un servidor web.
* Protocolo orientado a transacciones.
* Sigue el esquema request-response.
* El cliente web se denomina user agent.
* La petición (request) se realiza enviando un mensaje con un determinado formato. Dicho formato especifica la acción.
* La respuesta (response) contiene el documento, archivo, imagen solicitado.

**MENSAJE HTTP**

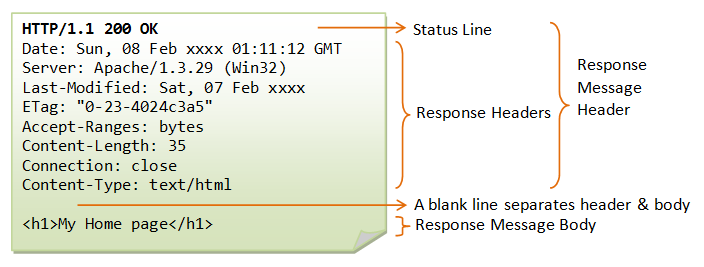
Un mensaje del tipo request y response consta de la siguiente **información**:

1. Una línea de inicio.
2. Opcionalmente una o más cabeceras del mensaje.
3. Opcionalmente el cuerpo del mensaje.

**REQUEST**



**RESPONSE**



**MÉTODOS O VERBOS HTTP**

* **GET**: obtiene el recurso especificado por la URL.
* **POST**: envía o somete datos para que sean procesados por el recurso identificado en la URL.
* **PUT**: sube o actualiza un recurso especificado.
* **DELETE**: Borra el recurso especificado.
* **PATCH:** solicita la actualización parcial de un recurso
* **HEAD**: Obtiene sólo las cabeceras de una petición. Es un verbo similar a GET, sólo que GET además de las cabeceras también obtiene el cuerpo de la respuesta.
* **TRACE**: este método se emplea con fines de diagnóstico ya que solicita al servidor que envíe todos los datos de la solicitud enviada.
* **OPTIONS**: sirve para obtener una lista de los métodos HTTP que soporta el servidor
* **CONNECT**: se utiliza para saber si tenemos acceso a un determinado host.

**CÓDIGOS DE ESTADO HTTP**

El primer dígito de cada código de tres dígitos comienza con uno de los números del 1 al 5. Existen los siguientes **códigos de estado de 100 a 500**:

* Los **códigos en los 100s** son informativos e indican que la solicitud fue recibida y el proceso continúa.
* Los **códigos en los 200s** indican la recepción de la solicitud y su procesamiento exitoso.
* **300s** indican la recepción de una solicitud, pero la necesidad de llevar a cabo un paso más para que la solicitud se ha completado.
* **Los 400s** simbolizan error del cliente en que la solicitud fue hecha pero la página ha perdido su validez.
* **El 500** indican el error del servidor - hubo una solicitud válida desde el cliente, pero servidor no pudo completar la misma.

Códigos 200 y 301

* **Código 200** simboliza la solicitud correcta y es la correcta para la mayoría de los escenarios.
* El **código 301** indica que "se trasladó de forma permanente". El recurso se ha dado una nueva dirección URL que es permanente. referencias futuras deben utilizar una de las URL que fueron devueltos. La redirección 301 debe ser utilizado en cualquier momento una URL debe ser redirigido a una URL diferente.

Código 302

El **código 302** indica que el servidor está respondiendo a la solicitud con una página ubicación diferente. Sin embargo, el solicitante mantiene mediante la dirección original para futuras peticiones.

302 no es favorable, ya que los rastreadores de motores de búsqueda tendrán en cuenta la redirección como un arreglo temporal y no proporcionar a la página con el poder de los enlaces de 301 redirecciones.

Código 404 y 200

El **código 404** indica archivo no encontrado. El servidor no pudo igualar la URL de la solicitud. No hay ninguna indicación del estado de la enfermedad, ya sea permanente o temporal. Esto ocurre cuando el servidor no puede encontrar una solicitud de página que se pongan en venta. Webmasters suelen mostrar texto de error 404 cuando el código de respuesta es de 200. Esto les da a los rastreadores de motores de búsqueda de la indicación de que la página se procesa de forma correcta. A veces, la página web está indexado erróneamente.

Códigos 410 y 404

El **código 410** indica que la página no existe y no se dispone de más en el servidor y también que no hay una dirección de reenvío conocido. Esto es más o menos una situación permanente. Los clientes que son capaces de modificar el vínculo necesario suprimir toda referencia a la URL de solicitud tras la aprobación del usuario. Si el servidor no puede determinar el recurso, se debe utilizar el código de estado 404.

Código 503

El **código de estado 503** indica la incapacidad del servidor de la concesión de la dirección URL solicitada por el mantenimiento o la sobrecarga temporal del servidor. 503 debe ser utilizado en el caso de una interrupción temporal. Se da la indicación de los motores de búsqueda que el sitio es sólo temporal hacia abajo.

CORS: INTERCAMBIO DE RECURSOS DE ORIGEN CRUZADOS

Cuando se abre una página web, cargar datos de servidores ajenos está, en teoría, estrictamente prohibido. Sin embargo, puede haber excepciones: si los administradores de ambas webs han acordado trabajar juntos, no hay por qué evitar el intercambio. En estos casos, el llamado cross-origin resource sharing (CORS) regula la colaboración.

**¿CÓMO FUNCIONA EL CORS?**

La **same-origin policy** (SOP o política de seguridad del mismo origen) prohíbe que se carguen datos de servidores ajenos al acceder a una página web. Todos los datos deben provenir de la misma fuente, es decir, corresponder al mismo servidor. Se trata de una medida de seguridad, ya que JavaScript y CSS podrían cargar, sin que el usuario lo supiese, contenido de otros servidores (y, con este, también contenido malicioso).

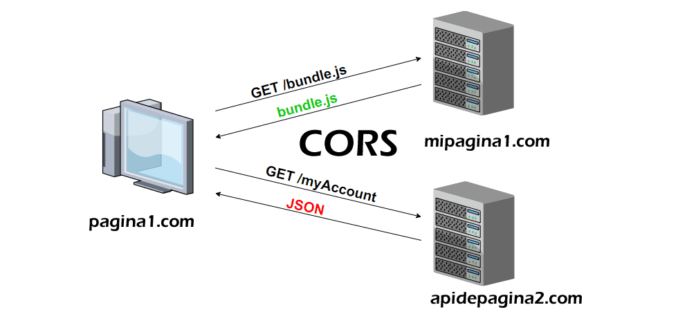
Tales intentos son denominados “**cross-origin requests**”. Si, por el contrario, ambos administradores web saben del intercambio de contenido y lo aprueban, no tiene sentido impedir este proceso. El servidor solicitado (es decir, aquel del que se quiere cargar contenido) puede permitir entonces el acceso mediante **cross-origin resource sharing**, en castellano, intercambio de recursos de origen cruzado.

Este permiso se da, no obstante, únicamente a clientes concretos, es decir, el CORS no es un comodín para realizar cualquier cross-origin request. En lugar de eso, el segundo servidor permite al primero un acceso exclusivo mediante una [**cabecera HTTP**](https://www.ionos.es/digitalguide/hosting/cuestiones-tecnicas/cabecera-http/). En dicha cabecera de la respuesta HTTP está indicado específicamente qué servidores pueden cargar datos y ponerlos a disposición del usuario. El acceso generalizado a todos los clientes se permite únicamente mediante una “wildcard” o certificado comodín. Esta solución, sin embargo, solo es conveniente para servidores cuyo contenido debe estar a disposición del público general, como es el caso, por ejemplo, de las tipografías web.

Si todo sale bien, el usuario no se percatará en absoluto del intercambio entre ambos servidores. Todos los navegadores actuales soportan el CORS, y el envío de solicitudes y respuestas sucede rápidamente al solicitar una página web sin que el usuario lo note.

**ESTRUCTURA DE LA CORS HEADER**

De acuerdo con la política de seguridad del mismo origen, en una conexión entre servidores, los datos referentes al origen se componen de tres elementos: **host, puerto y protocolo**. De este modo, y tomando el ejemplo de la imagen, la directriz prohíbe que ’*https://example.com’* acceda a *’http://example.com’* o a *’https://example.org’*. En el primer caso, **el protocolo no es el mismo** y, en el segundo, los **datos de host no coinciden.**

****

Una petición de origen cruzado es, en teoría, una petición HTTP. Los métodos específicos no suelen dar problemas. GET y HEAD no pueden alterar datos y, por lo tanto, no suelen considerarse como un riesgo para la seguridad. No se puede decir lo mismo de PATCH, PUT o DELETE: con ellos sí se puede llevar a cabo acciones maliciosas, por lo que en estos casos también hay que activar el cross-origin resource sharing, ya que CORS no solo puede contener información sobre el origen permitido, sino también acerca de qué peticiones HTTP están permitidas por la fuente.

Si se trata de métodos HTTP de seguridad, el cliente envía en primer lugar una **solicitud preflight** (preflight request) en la que solo se indica qué método HTTP se piensa transmitir al servidor a continuación y se pregunta si la solicitud es considerada segura. Para ello, se usa la cabecera OPTIONS (OPTIONS header). Una vez se haya recibido una respuesta positiva, ya se puede realizar la solicitud propiamente dicha.

Existen **diferentes cabeceras** o CORS headers y cada una aborda un aspecto distinto. Ya hemos mencionado dos cabeceras importantes para identificar orígenes seguros y métodos permitidos, pero hay más:

* **Access-Control-Allow-Origin**: ¿qué origen está permitido? Especifica desde qué otro host se puede acceder al servidor solicitado. Además de una dirección concreta, en dicha cabecera también se puede incluir una **wildcard** en forma de asterisco. De esta manera, el servidor permitirá cross-origin requests de cualquier origen.
* **Access-Control-Allow-Credentials**: ¿también se aceptan solicitudes cuando el modo de credenciales es incluir (include)?
* **Access-Control-Allow-Headers**: ¿qué cabeceras pueden utilizarse?
* **Access-Control-Allow-Methods**: ¿qué métodos de petición HTTP están permitidos?
* **Access-Control-Expose-Headers**: ¿qué cabeceras pueden mostrarse?
* **Access-Control-Max-Age**: ¿cuándo pierde su validez la solicitud preflight?
* **Access-Control-Request-Headers**: ¿qué header HTTP se indica en la solicitud preflight?
* **Access-Control-Request-Method**: ¿qué método de petición HTTP se indica en la solicitud preflight?
* **Origin**: ¿de qué origen proviene la solicitud?

Si la cabecera de la respuesta no correspondiera con las especificaciones de la solicitud, o si el servidor solicitado no respondiese, la cross-origin request no se podría realizar.

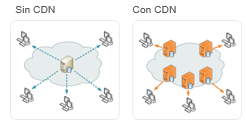
**VENTAJAS E INCOVENIENTES DEL CORS**

El **propósito** del CORS es eludir la medida de seguridad establecida como configuración predeterminada (la política de seguridad del mismo origen). Dicha política es, de hecho, un medio muy eficaz para **bloquear conexiones potencialmente peligrosas**. Internet, sin embargo, se basa a menudo precisamente en este tipo de cross-origin requests, ya que muchas de las conexiones entre hosts sí son deseadas.

Por eso, el CORS ofrece una solución intermedia, permitiendo hacer excepciones a la prohibición en aquellas situaciones en que las solicitudes de origen cruzado son expresamente requeridas. No obstante, se corre el riesgo de que los administradores web se aprovechen de las wildcards por comodidad, haciendo que la protección de la SOP sea en vano. Por eso, es importante utilizar el CORS **solo en casos especiales** y configurarlo de la manera más restrictiva posible.

CONTENT DELIVERY NETWORK (CDN)

Las CDN emergen como la solución al actual **problema que presenta una web centralizada**: lograr bajo tiempo de respuesta y mínima pérdida de información moviendo el contenido de la información más cerca de los usuarios. El **objetivo** es lograr un equilibrio entre los costos en que incurren los proveedores de contenido web y la calidad de servicio para los usuarios finales.



Servidores con copias de datos en varios puntos de la red muy utilizado para librerías javascript.

**Ventajas**:

* Reduce la carga de los servidores.
* Red de tráfico distribuida.
* Reduce la latencia.
* Incrementa el ancho de banda.
* Aumenta el web caching.

**ARQUITECTURA DE UN CDN**

* + **Componente de entrega de contenidos**: se cuenta con un servidor de origen y un conjunto de servidores sustitutos para replicar el contenido.
  + **Componente de enrutamiento de solicitudes**: Usuarios solicitan directamente a los servidores sustitutos. Interactúa con el componente de distribución para mantener y actualizar el contenido.
  + **Componente de distribución de contenido**: Mueve el contenido desde el origen a los servidores sustitutos y asegura consistencia.
  + **Componente de contabilidad**: Mantiene registros de los accesos de los clientes y los registros de uso de los servidores. Ayuda a la presentación de informes de tráfico y facturación basada en el uso.

**SERVICIOS Y CONTENIDOS COMPATIBLES CON CDN**

* **Contenido estático**: Páginas estáticas HTML, imágenes, documentos, parches de software.
* **Distribución de audio y video por internet**: Audio y video en tiempo real. Videos generados por el usuario.
* **Servicio de contenido**: Directorio, e – commerce, servicio de transferencia de archivos.
* **Fuentes de contenido**: Grandes empresas, proveedores de servicios Web, compañías de medios de comunicación, y emisoras de noticias.
* **Clientes**: Medios de comunicación y empresas de publicidad por internet, centros de datos, proveedores de Internet, minoristas de música en línea, operadores móviles y fabricantes de electrónica de consumo.
* **Interacción del usuario**: Celular, smartphone/PDA, notebooks y computadores de escritorio.

**OBJETIVOS DE NEGOCIO**

* Escalabilidad.
* Habilidad para expandirse con el objetivo de manejar nuevos y grandes cantidades de datos. Usuarios y transacciones.
* Requiere capacidad para la entrega de contenido dinámico de aprovisionamiento y de alta calidad, con bajo costo operacional.
* Tendencia futura: los proveedores de contenidos, así como usuarios finales pagarán para obtener contenido de alta calidad.
* Seguridad.
* Protección del contenido contra modificaciones y accesos no autorizados.
* Requiere red física, software, datos y procedimientos de seguridad.
* Tendencia futura: reducir la interrupción del negocio mediante la lucha contra los ataques de negación de servicio y otras actividades maliciosas.
* Fiabilidad, Capacidad de respuesta y rendimiento.
* Disponibilidad de servicios, manejo de posibles interrupciones y experiencia del usuario final.
* Requiere una red tolerante a fallas con balanceo de carga adecuada.
* Tendencia futura: ubicación del contenido distribuido, la coherencia de caché y los mecanismos de enrutamiento

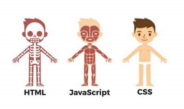
**BENEFICIOS DEL CDN**

* Mayor capacidad de conexión.
* Disminución del tiempo de respuesta de entrega de información al usuario.
* Disminución de los costos asociados a la entrega de contenidos.
* Reducción de la pérdida y demora de paquetes ya que trabajan con nodos cercanos al usuario.
* Disminución de carga de la red.
* Se tiene 100% de disponibilidad de información, incluso ante la caída de uno de los servidores.
* Permite obtener estadísticas de comportamiento de usuarios basado en el registro de páginas visitadas, ubicación geográfica, entre otras.

**U2: DISEÑO DEL FRONTEND CON CSS3 Y BOOTSTRAP**

QUÉ ES CSS

* CSS quiere decir Hojas de Estilo en Cascada.
* Un archivo CSS es típicamente un archivo de texto con extensión .css que contiene una serie de comandos y reglas.
* Estas reglas le dicen a HTML como se debe mostrar. Trabajan en conjunto para ofrecer características estéticas y funcionales a los sitios web.



ADJUNTAR UNA HOJA DE ESTILOS

Hay tres **formas de agregar CSS a un documento HTML**:

1. **Inline.**

*<p style=”color:red”>Algun texto</p>*

1. **Hoja de estilos interna.**

*<style type=”text/css>*

*h1{color:red}*

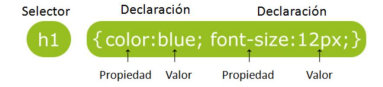
*</style>*

1. **Hoja de estilos externa.**

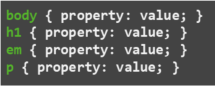
*<link rel=”stylesheet” type=”text/css” media=”all” href=”css/style.css”/>*

ESTRUCTURA DE UNA REGLA CSS

Una regla CSS consiste en un **selector**, siguiendo por una o más declaraciones. Una **declaración** es un par propiedad valor.

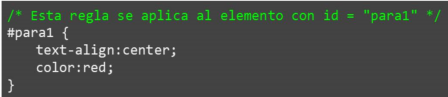


**SELECTOR TAG**

* Corresponde el tag HTML del elemento.
* Se aplica en todo el documento.

**SELECTOR ID**

* El selector id se usa para especificar el estilo para un único elemento.
* El selector id usa el atributo id del elemento HTML y se define con un “#”.



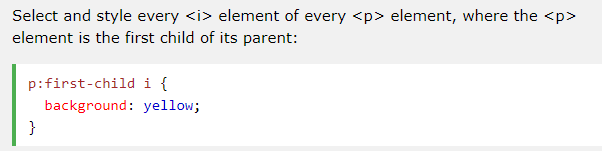
**SELECTOR CLASS**

* El **selector class** se usa para especificar un estilo para un grupo de elementos. A diferencia del selector id, el **selector class** se utiliza para aplicar estilo a múltiples elementos.
* Esto permite tener un conjunto de estilos para muchos elementos de la misma clase.
* El **selector class** utiliza el atributo HTML class y se define con un “.”

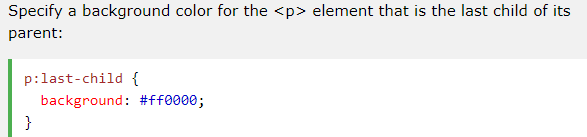


**OTROS SELECTORES: PSEUDO CLASES** (https://www.w3schools.com/cssref/css\_selectors.asp)

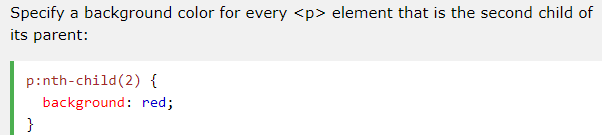
* **:First-child** -> se utiliza para seleccionar el selector especificado, solo si es el primer hijo de su padre.



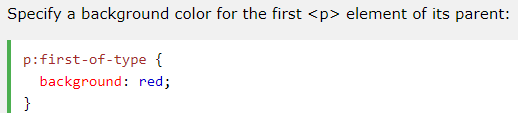
* **:last-child** -> une cada elemento que es el último hijo de su padre.
  + El ultimo hijo es igual a p:nth-last-child(1)



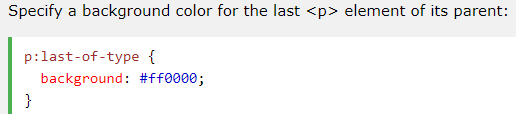
* **:nth-child(n)** -> une cada elemento que es el nth hijo, independientemente del tipo, de su padre. N puede ser un número, una palabra clave o una formula.



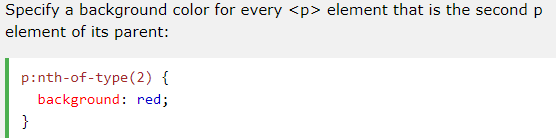
* **:first-of-type** -> une cada elemento que es el primer hijo, de un tipo particular, de su padre.



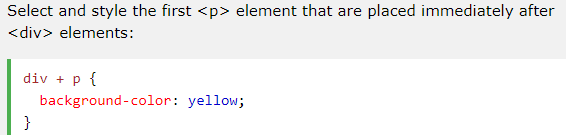
* **:last-of-type** -> une cada elemento que es el último hijo, de un tipo particular, de su padre.
  + Esto es lo mismo que: nth-last-of-type(1).



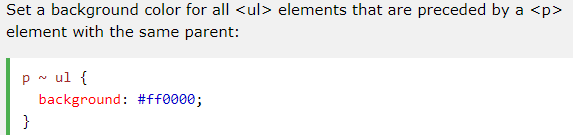
* **:nth-of-type(n)** -> une cada elemento que es el n-ésimo hijo, de un tipo particular, de su padre. N puede ser un número, una palabra clave o una formula.



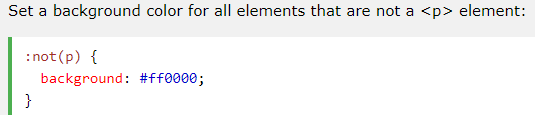
* **Hemana adyacente +** -> se utiliza para seleccionar un elemento que está directamente después de otro elemento especifico.



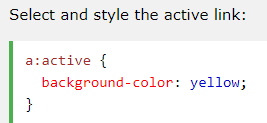
* **Todas las hermanas ~** -> une las apariciones del elemento 2 que están precedidas por elemento 2. Ambos elementos deben tener el mismo padre, pero elemento2 no tiene que estar inmediatamente precedido por elemento1.



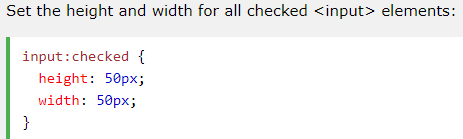
* **:not(selector)** -> une todos los elementos que no son el elemento/selector especificado.



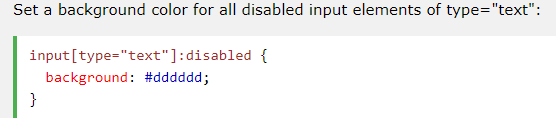
* **:active** -> se utiliza para seleccionar y aplicar estilo al enlace activo. Un enlace se activa cuando se hace clic en él.
  + debe ir **después de :hover** (si está presente) para que sea efectivo.



* **:checked** -> une cada elemento <input> marcado (solo para radio buttons y checkboxes) y el elemento <option>.

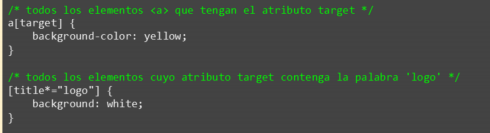


* **:disabled** -> une todos los elementos desactivados ( se usa principalmente en los elementos del formulario).



**OTROS SELECTORES: ATRIBUTO**

Se puede usar como selector la presencia o el valor de un atributo.

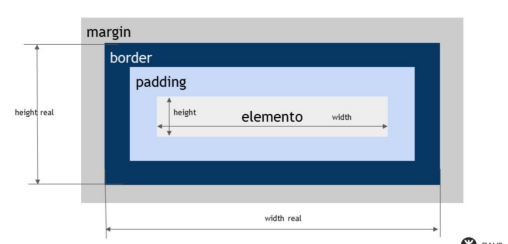


ORDEN DE LA CASCADA

Cuando hay varios estilos definidos para un mismo elemento, se aplica:

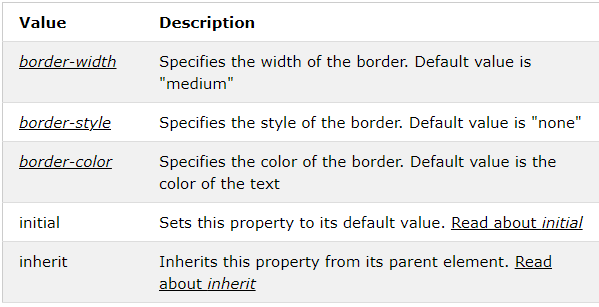
1. Browser default.
2. Style sheet externa.
3. Style sheet interna.
4. Style sheet inline.

CSS BOX MODEL https://www.w3schools.com/cssref/default.asp



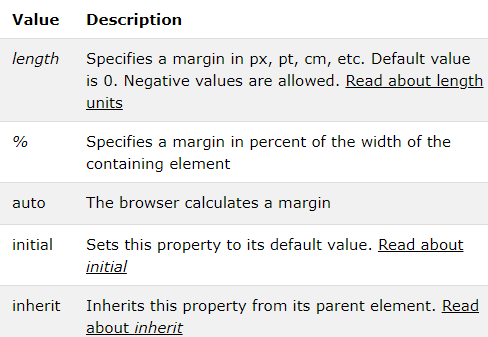
**BORDER**

* Es una propiedad abreviada para:
  + Border-width
  + Border-style
  + Border-color
* Si se omite el color del borde, el color aplicado será el color del texto.
* *border: border-width border-style border-color|initial|inherit;*



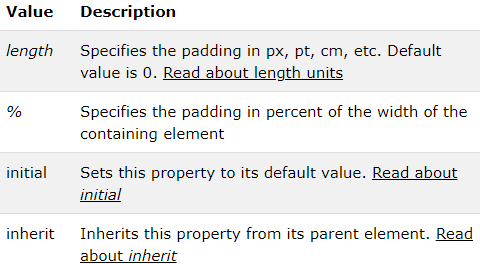
**MARGIN**

* Establece los márgenes de un elemento y es una propiedad abreviada de las siguientes propiedades:
  + Margin-top.
  + Margin-right.
  + Margin-bottom.
  + Margin-left.
* Si la propiedad tiene **4 valores** -> *margin: margin-top margin-left margin-bottom margin-right.*
* Si la propiedad tiene **3 valores** -> *margin: margin-top margin-right/left margin-bottom.*
* Si la propiedad tiene **2 valores** -> margin: margin-top/bottom margin-right/left
* Si la propiedad tiene **un valor** -> margin: todos los márgenes.
* Los valores negativos están permitidos.



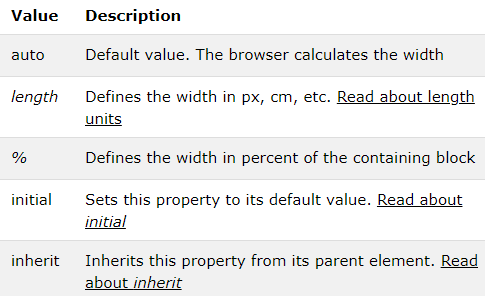
**PADDING**

* Es el espacio entre el contenido y su borde. Es una propiedad abreviada para:
  + Padding-top
  + Padding-rigth
  + Padding-bottom
  + Padding-left
* Crea un espacio extra dentro de un elemento, mientras que margin crea un espacio extra alrededor de un elemento.
* Puede tener entre 1 y cuatro valores (equivalente a margin)
* Los valores negativos no están permitidos.
* *Padding: length|initial|inherit;*



**WIDTH**

* Establece el ancho de un elemento.
* El ancho de un elemento no incluye padding, borde o margen.
* Las propiedades min-width y max-width anulan la propiedad width.
* *Width: auto|value|initial|inherit;*



**HEIGHT**

* Establece la altura de un elemento.
* La altura de un elemento no incluye paading, borde o margen.
* **height:auto** -> el elemento ajustará automáticamente su altura para permitir que su contenido se muestre correctamente.
* Si la altura se establece en un valor numérico, como pixeles, r(em), porcentajes, si el contenido no se ajusta a la altura especificada, se desbordará. La propiedad Overflow define como el contener manejará el contenido desbordado.
* *Height: auto|length|initial|inherit;*

