**¿Qué es CSS?**

Para empezar, es crucial entender que una página web es, en su esencia, un documento de texto.

* Este documento utiliza código HTML para crear el contenido y la estructura de la página.
* Además, se emplea código CSS para aplicar estilos visuales, como colores, disposición y otras características, al contenido.

En resumen, HTML y CSS son como un “idioma” que los navegadores web (como Chrome o Firefox) entienden. Nuestro objetivo como desarrolladores web es aprender este lenguaje para indicar al navegador como se debe mostrar el contenido.

**¿Qué significa CSS?**

CSS (Cascading Style Sheets):

CSS significa “Hojas de Estilo en Cascada”. Su concepto es simple pero potente: aplicar estilos (colores, formas, márgenes, etc) a uno o varios documentos (generalmente HTML) de forma automática y masiva.

Se le llama “en cascada” porque el código CSS se lee y aplica de arriba hacia abajo (siguiendo un patrón de herencia). Si hay estilos contradictorios, se resuelven siguiendo unas reglas específicas.

Ventaja Clave:

Normalmente, un sitio web tiene múltiples documentos HTML pero un único archivo CSS. Al vincular todos los HTML a ese mismo CSS, cualquier cambio en el archivo CSS se refleja automáticamente en todas las páginas HTML asociadas, facilitando la gestión del diseño.

**¿Por qué se usa CSS?**

La idea central de CSS es la separación de presentación y contenido. Esto significa que, como programadores, debemos mantener el código HTML y el código CSS claramente divididos.

¿Por qué Separar?

* Facilita la modificación y el mantenimiento del código a largo plazo.

Roles de los Documentos:

* Documentos HTML (Contenido): Solo deben incluir la información y los datos a trasmitir.
* Documentos CSS (Presentación): Contendrán únicamente los aspectos relacionados con el estilo (diseño, colores, formas, etc).

Ventajas de esta Separación:

Al unificar todo lo visual en un solo documento CSS, obtenemos varias ventajas:

* Modificaciones Centralizadas: Los cambios visuales se realizan en un solo lugar y se aplican a todo el sitio web.
* Menos Duplicación: Se reduce la repetición de estilos, haciendo el código más organizado.
* Carga Más Rápida: La información a trasmitir es menor, lo que acelera la descarga de las páginas.
* Adaptabilidad: Facilita la creación de versiones diferentes para distintos dispositivos (Tablets, Smartphones, etc).

Analogía Representativa:

HTML – The Skeleton

CSS – The Skin

JavaScript – The Brain

Selector h1 ({Porperty color: Value blue; font-size:12px;) Declaration cada una clave-valor.

**Formas de enlazar CSS:**

Hay tres formas principales de aplicar estilos CSS a un documento HTML, desde la más recomendada hasta la menos común:

1. **Archivo CSS Externo (¡Recomendada!):**

* **Cómo:** Usando la etiqueta <link rel=”stylesheet” href=”ruta/archivo.css” /> dentro de la sección <head> del HTML.
* **Descripción:** El código CSS se escribe en un archivo .css separado.
* **Ventajas:**
* Separación clara de contenido y estilo.
* Reutilización: Un solo archivo CSS puede estilizar múltiples páginas HTML, facilitando cambios masivos en todo el sitio.
* Mejora la velocidad de carga al permitir que el navegador aplique estilos más rápido.
* **Recomendación:** Es la forma predeterminada y más aconsejable. Se sugiere colocar la etiqueta <link> temprano en el <head> para una carga visual óptima.

1. **Bloque de Estilos Interno**

* **Cómo:** Escribiendo el código CSS directamente dentro de una etiqueta <style> ubicada en el <head> del documento HTML.
* **Ejemplo:**

<head>

<style>

div { background: hotpink; color: white; }

</style>

</head>

* **Ventajas:** Útil para estilos específicos de una sola página que no se reutilizarán.
* **Desventajas:** Rompe la separación de contenido y estilo si se usa excesivamente, y no permite la reutilización fácil de estilos en otras páginas. Se le debe dar prioridad al método de archivo externo.

1. **Estilos en Línea**

* **Cómo:** Aplicando el código CSS directamente en el atributo style de una etiqueta HTML específica. Las propiedades se separan con ; .
* **Ejemplo:** <p style="color: red; padding: 8px">¡Hola amigo lector!</p>
* **Ventajas:** Muy específico para aplicar estilos a un **elemento único** o para inyectar variables CSS dinámicamente.
* **Desventajas:** Se considera una **mala práctica** si se abusa, ya que los estilos quedan asociados a una sola etiqueta, no son reutilizables y dificultan el mantenimiento.

**Sintaxis General.**

El código CSS se basa en una serie de conceptos fundamentales que son cruciales para entender cómo se aplican los estilos a los documentos web.

**Conceptos Clave de CSS:**

* **Selector:** El elemento o elementos del documento HTML a los que se le aplicará un estilo específico.
* **Propiedad:** La característica principal que se desea modificar (ej. Color, background, font-size).
* **Valor:** El valor concreto asignado a una propiedad, que define su comportamiento o apariencia (ej. Red, blue, 16px).
* **Regla:** Un par de propiedad y su valor asociado, que define una característica de estilo (ej., color: red,).
* **Comentario:** Fragmento de texto entre /\* y \*/ que el navegador ignora. Sirve para añadir anotaciones o aclaraciones para los desarrolladores.

**Estructura General de un Archivo CSS:**

Un documento CSS se organiza en bloques de reglas, siguiendo este formato:

selector {

propiedad: valor; /\* Comentario \*/

propiedad2: valor2;

}

Ejemplo Práctico:

p {

color: red; /\* Color de texto \*/

}

En este ejemplo:

* p es el **selector** (aplica a todas las etiquetas <p> en el HTML).
* color es la **propiedad**.
* red es el **valor**.
* color: red; es una **regla**.
* /\* Color de texto \*/ es un **comentario**.

Cada regla se termina con un **punto y coma (;)**. El último punto y coma en un bloque de reglas es opcional.

**Sintaxis Avanzada y Buenas Prácticas en CSS Documento CSS**

Los documentos CSS suelen ser mucho más complejos que un simple ejemplo, conteniendo cientos de reglas con diversos selectores, propiedades y valores. Cada bloque de declaración comúnmente incluye múltiples reglas.

Ejemplo de Sintaxis con Múltiples Reglas:

/\* Ejemplo 1 \*/

h1 {

color: blue;

font-size: 2em;

text-align: center;

}

/\* Ejemplo 2 \*/

.mi-clase {

background-color: #f0f0f0;

padding: 10px;

border-radius: 5px;

}

Buenas Prácticas para Escribir Código CSS:

A medida que el código CSS crece, puede volverse complejo de controlar y mantener. Para evitar esto, es fundamental seguir estas buenas prácticas, que, aunque no son obligatorias para el navegador, son esenciales para la legibilidad y el mantenimiento por parte de los desarrolladores:

1. **Una Regla por Línea:** Escribe cada par propiedad: valor; en una línea separada. Esto mejora drásticamente la legibilidad y facilita las modificaciones.

**Ejemplo:**

selector {

propiedad1: valor1;

propiedad2: valor2;

}

1. **Usa Identación:** Después de abrir un bloque de reglas con {, identa las propiedades hacia la derecha (con espacios o tabulaciones). Esto hace que la estructura sea visualmente clara.

**Ejemplo:**

selector {

propiedad: valor; /\* Las propiedades están indentadas \*/

}

1. **Siempre el Último Punto y Coma:** Aunque el último ; de un bloque de reglas es opcional, se aconseja escribirlo siempre. Esto mantiene la coherencia y ayuda a prevenir errores o descuidos al añadir o reordenar reglas en un futuro.

**Ejemplo:**

selector {

propiedad1: valor1;

propiedad2: valor2; /\* Punto y coma opcional, pero recomendado \*/

}

Seguir estos consejos mejora sustancialmente la legibilidad del código y es considerado un cumplimiento obligatorio entre desarrolladores para entender rápidamente el código ajeno y el propio.

**Uso de Navegadores - Importancia de Conocer los Navegadores para CSS**

Es fundamental entender cómo funcionan los navegadores web al aplicar CSS porque cada navegador interpreta y renderiza el código CSS de manera ligeramente diferente.

**Razones Clave:**

* **Consistencia y Compatibilidad:** Lo que funciona bien en un navegador puede no hacerlo en otro. Conocer estas diferencias (conocidas como “compatibilidad entre navegadores”) nos permite escribir código CSS que se vea y funcione consistentemente en diversas plataformas.
* **Mejora la Experiencia del Usuario:** Asegurar la compatibilidad garantiza que todos los usuarios tengan una buena experiencia, independientemente del navegador que utilicen.
* **Accesibilidad:** Contribuye a que el sitio sea accesible para una mayor audiencia.
* **Aprovechar Nuevas Características:** Comprender las capacidades de cada navegador nos permite utilizar al máximo las últimas características y tecnologías de CSS para crear diseños modernos y atractivos.

**Herencia – Herencia de Propiedades en CSS**

En CSS, algunas propiedades se heredan de los elementos HTML padres a los hijos, cambiando el valor por defecto de estos últimos. Otras propiedades, sin embargo, no se heredan.

**Cómo Funciona la Herencia:**

Cuando una propiedad es heredable (ej., color o font-family), si las aplicas a un elemento padre, sus elementos hijos adoptarán automáticamente ese valor, a menos que una regla más específica les dé un valor diferente.

**Ejemplo:**

<div class="container">

<p>Texto del padre</p>

<div class="child">

<p>Texto del hijo</p>

</div>

</div>

<style>

  .container {

    color: green; /\* Propiedad heredable \*/

    border: 1px solid red; /\* Propiedad NO heredable \*/

  }

</style>

* **color: green; (Heredable):** Tanto el texto dentro de .container como el texto dentro de .child aparecerán en verde. El hijo hereda el color del padre porque no hay una regla más específica para el hijo.
* **border: 1px solid red; (NO Heredable):** Solo el elemento .container tendrá un borde rojo. El elemento .child no hereda este borde.

**Razón de la Herencia Selectiva:**

La herencia no ocurre con todas las propiedades CSS, solo con aquellas donde este comportamiento es deseable. Por ejemplo, si los bordes fueran heredables, todos los elementos anidados tendrían su propio borde, lo cual rara vez es el efecto buscado. La herencia selectiva evita comportamientos visuales no deseados y simplifica la aplicación de estilos lógicos.

**Modelo en Cascada – Concepto Clave**

La Cascada de CSS es uno de los conceptos más importantes y fundamentales de CSS, que incluso da sentido a la “C” de sus siglas (Cascading Style Sheets).

Aunque puede ser un concepto algo avanzado para principiantes, comprenderlo es crucial. Se basa en cómo CSS resuelve conflictos cuando múltiples reglas intentan aplicar estilos a un mismo elemento.

Para entender completamente la cascada, es útil tener un buen conocimiento de los selectores CSS y de las propiedades CSS.

**¿Qué es la Cascada CSS? – Resolviendo Conflictos de Estilos**

La Cascada de CSS es el algoritmo o conjunto de reglas que utiliza el navegador para decidir qué estilos CSS aplicar a un elemento HTML cuando existen ambigüedades o conflictos. No es tan simple como aplicar siempre el último estilo definido.

La cascada resuelve estos conflictos considerando varios factores de prioridad:

1. **Importancia:** Algunos estilos tienen una importancia intrínseca mayor o menor que otros.
2. **Origen:** Los estilos pueden provenir de diferentes orígenes (ej., navegador, usuario, autor de la página), cada uno con su propia prioridad.
3. **Orden de Aparición:** El orden en que se define y procesa el CSS influye en la resolución de conflictos.
4. **Especificidad:** Cuanto más especifico sea un selector CSS, mayor será su prioridad para aplicar un estilo.
5. **Capas:** Los estilos CSS pueden organizarse en capas, lo que permite fusionarlos de forma controlada.

Entender estos puntos es clave para predecir y controlar cómo se aplican los estilos en tus páginas web.

**Importancia - !important en CSS: Forzando la Prioridad de Estilos.**

En CSS, puedes añadir la palabra clave !important al final de una regla para otorgarle la máxima prioridad al momento de ser evaluada por el navegador, superando a otras reglas de estilo.

**Funcionamiento:**

Cuando una propiedad tiene !important, su valor prevalece sobre otros estilos, incluso si esos otros estilos son más específicos o aparecen posteriormente en el código.

**Ejemplo:**

<head>

<style>

  .text {

    color: blue; /\* Inicialmente azul \*/

  }

  .text {

    color: red!important; /\* ¡Este prevalece! \*/

  }

</style>

</head>

<body>

<div class="text">Texto del elemento</div>

</body>

En este caso, aunque color: blue; esté definido primero, el texto se mostrará en **rojo** debido a !important. Si hay varias reglas con !important, la que aparezca **en último lugar** será la que prevalezca.

**Advertencia sobre su Uso:**

* **!important es una herramienta potente para "forzar" estilos, pero su uso indiscriminado es una MALA PRÁCTICA.**
* Debe **evitarse** salvo en casos muy específicos y justificados, ya que puede dificultar el mantenimiento y la depuración del CSS, al romper las reglas de la cascada y la especificidad.

**Origen – Origen de las Hojas de Estilo en la Cascada CSS**

La Cascada de CSS considera el origen de las hojas de estilo al determinar la prioridad de los estilos. Aunque como desarrolladores nos enfocaremos principalmente en el “CSS de autor”, es importante conocer los tres orígenes y su jerarquía.

Los orígenes se ordenan de menor a mayor importancia de la siguiente manera:

1. **Agente de Usuario:** 
   * Son los estilos CSS que el navegador web aplica por defecto a los elementos (ej., un <h1> es grande y en negrita por defecto del navegador).
2. **CSS de Usuario:**
   * Son los estilos CSS que un usuario final añade para personalizar su experiencia de navegación (ej., mediante extensiones como UserStyles para aplicar temas o ajustes visuales personales).
3. **CSS de Autor:**
   * Son los estilos CSS que el desarrollador (yo) crea e implementa directamente en la página web. Estos son los que generalmente controlamos.

**Comportamiento con !important:**

Si una propiedad CSS tiene declarado !important en cada origen, el orden de importancia se invierte:

* En este caso, el CSS del agente de usuario con !important tendría la máxima prioridad, seguido por el CSS de usuario con !important, y finalmente el CSS de autor con !important.

**Orden de Aparición – Orden de Aparición y Especificidad en la Cascada CSS**

El orden de aparición de las reglas CSS y su especificidad son dos factores cruciales para que el navegador decida que estilos aplicar cuando hay conflictos.

**Orden de Aparición:**

Cuando se aplican estilos de diferentes orígenes o en diferentes ubicaciones, el navegador procesa el CSS en un orden determinado. Generalmente, el último estilo definido (o el que aparece más tarde en el código) para un mismo selector suele prevalecer, si no hay otros factores de prioridad.

**Ejemplo 1: Diferentes formas de incluir CSS:**

<html>

<head>

<link rel="stylesheet" href="index.css" /> <style>

   .text { color: red } /\* Prevalece sobre el de index.css si está aquí \*/

</style>

</head>

<body>

<div class="text">¡Desde Mi web puedes ver mi contenido!</div>

</body>

</html>

Si index.css define color: green; para .text y el bloque <style> define color: red; para .text, el texto será **rojo** porque la regla en el <style> aparece después y sobreescribe la anterior. Esto también aplica si hay múltiples etiquetas <link> o múltiples reglas en el mismo bloque <style>.

**Ejemplo 2: Mismas reglas en la misma hoja de estilos:**

<style>

 .text {

  padding: 8px;

  color: blue; /\* Este color se aplicará \*/

  background-color: grey;

 }

.text { /\* Si hubiese otro .text aquí con 'color: red', ese prevalecería \*/

/\* color: red; \*/

}

</style>

Si dentro de la misma hoja de estilos se define la misma propiedad para el mismo selector varias veces, **prevalece siempre la última regla definida**.

**Especificidad: Resolviendo Conflictos Complejos**

El orden de aparición solo es decisivo cuando los selectores tienen la misma especificidad. Sin embargo, cuando hay diferentes selectores apuntando al mismo elemento, entra en juego la especificidad CSS. El navegador calcula cuál selector es “más específico” y, por lo tanto, tiene mayor prioridad.

**Ejemplo de Conflicto por Especificidad:**

<div id="element" class="block">Texto del elemento</div>

<style>

div { color: red; } /\* Selector de tipo \*/

#element { color: blue; } /\* Selector de ID (¡más específico!) \*/

.block { color: green; } /\* Selector de clase \*/

</style>

En este caso, el elemento <div> tiene un ID (#element) y una clase (.block). Los estilos se aplicarían en un orden de prioridad determinado por la especificidad:

* Un selector de **ID (#element) es más específico** que un selector de clase (.block).
* Un selector de **clase (.block) es más específico** que un selector de tipo (div).

El navegador calculará la especificidad de cada selector para determinar cuál tiene la máxima prioridad y aplicar su estilo correspondiente.

**Variables CSS – Custom Properties.**

Las Custom Properties, o Variables CSS, permiten definir valores reutilizables en tu hoja de estilos, haciendo el código más limpio y fácil de mantener.

**Definición de una Custom Property:**

Para definir una variable CSS, se usa el prefijo de dos guiones (--) antes del nombre de la variable. Es común definirlas en la pseudoclase :root, que representa el elemento raíz del documento (normalmente <html>), para que estén disponibles globalmente.

**Sintaxis:**

:root {

--nombre-de-variable: valor;

}

**Ejemplo:**

:root {

--background-color: black; /\* Define una variable para el color de fondo \*/

}

**Utilización de una Custom Property:**

Para usar una variable CSS, se emplea la función var(), pasando el nombre de la variable como argumento.

**Sintaxis:**

propiedad: var(--nombre-de-variable);

**Ejemplo:**

body {

background-color: var(--background-color); /\* Utiliza la variable definida en :root \*/

color: white;

}

**Importancia del Ámbito:**

Es crucial entender que una Custom Property puede tener diferentes valores dependiendo del ámbito (donde fue definida en el árbol DOM). Por ejemplo, si defines –color-principal en un div específico, solo los hijos de ese div (y el propio div) lo heredarán, a menos que se defina globalmente en :root.

**Valor por Defecto (Recomendado):**

Es muy recomendable que la función var() tenga dos parámetros:

1. La Custom Property en cuestión.
2. Un valor por defecto que se utilizará si la variable no esta definida en el ámbito actual.

**Sintaxis:**

propiedad: var(--nombre-de-variable, valor-por-defecto);

**Ejemplo:**

.element {

background: var(--color-principal, blue); /\* Si --color-principal no existe, usa blue \*/

}