

CLASE 4 · 29/04/2026 · VIRTUAL

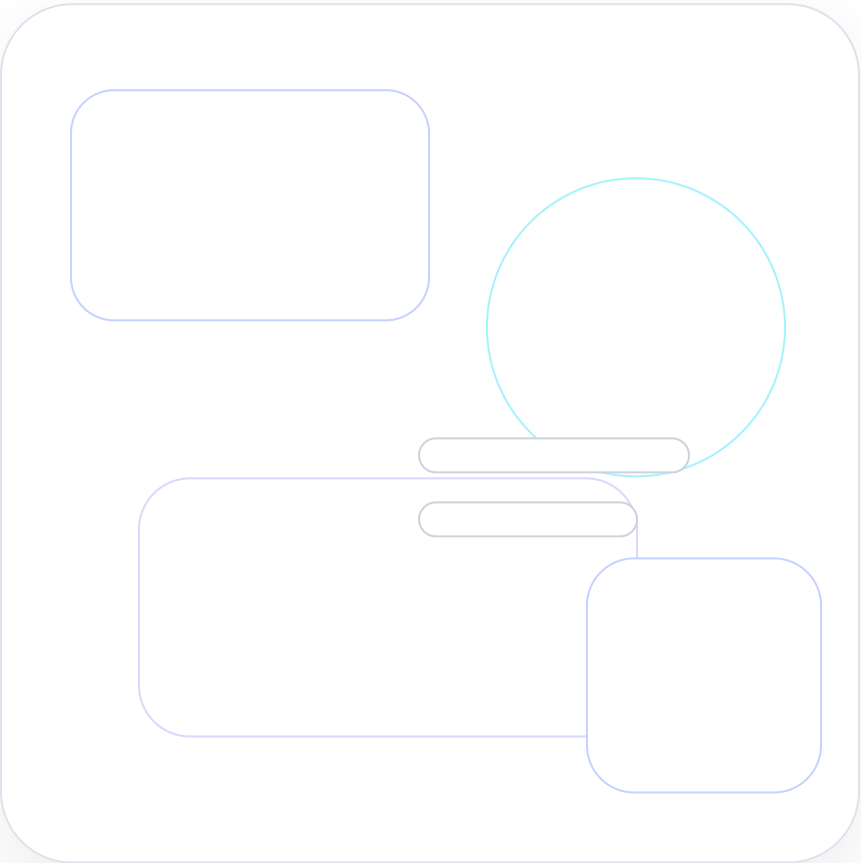
Introducción a CSS

Selectores, reglas, cascada, herencia y especificidad. La clase donde HTML deja de ser solo estructura y empieza a convertirse en interfaz.

CSS INICIAL

SELECTORES

BUENAS PRÁCTICAS



UBICACIÓN EN LA CURSADA

Venimos de construir estructura

- En HTML definimos títulos, párrafos, listas, enlaces, imágenes, tablas y formularios.
- Eso responde a la pregunta: **¿qué contenido hay?**
- Hoy empezamos a responder: **¿cómo se ve ese contenido?**

```
<h1>Cursos disponibles</h1>
<p>Elegí una opción para anotarte.</p>
<button>Inscribirme</button>
```

Lectura docente: HTML nos dio el esqueleto. CSS aparece cuando queremos que ese contenido tenga jerarquía visual: qué se destaca, qué se agrupa, qué parece botón y qué parece texto secundario.

OBJETIVOS

Qué deberían poder hacer al final

Entender CSS

Comprender qué problema resuelve y por qué conviene separar estructura y diseño.

Escribir reglas

Usar selector, propiedad y valor sin mezclar todo dentro del HTML.

Resolver conflictos

Interpretar cascada, herencia y especificidad para saber por qué gana una regla.

Idea para remarcar: No buscamos memorizar propiedades sueltas. Buscamos entender la lógica: seleccionar algo, aplicarle una regla y saber resolver cuando dos reglas chocan.

La meta no es decorar: es controlar la interfaz

IDEA BASE

HTML, CSS y JavaScript no hacen lo mismo

HTML

Define la estructura: títulos, párrafos, formularios, enlaces, tablas.

Contenido

CSS

Define la presentación: colores, tamaños, espaciado, distribución visual.

Diseño

JavaScript

Agrega comportamiento: eventos, validaciones, cambios dinámicos.

Interacción

Clave: Separar responsabilidades ayuda a mantener el código. Si mañana cambia el diseño, no debería ser necesario reescribir toda la estructura HTML.

DEFINICIÓN

Qué es CSS

Cascading Style Sheets

CSS significa hojas de estilo en cascada. Sirve para indicar cómo debe presentarse visualmente un documento HTML.

1. Selecciono un elemento.
2. Indico una propiedad visual.
3. Asigno un valor.

Mi primera página

El mismo contenido puede cambiar por completo cuando le aplicamos estilos.

Continuar

Lectura simple: CSS funciona como una lista de instrucciones visuales. El navegador lee esas instrucciones y decide cómo dibujar cada elemento en pantalla.

ANTES Y DESPUÉS

El mismo HTML, otra presentación

HTML sin CSS

```
<h1>Inscripción a cursos</h1>
<p>Completá tus datos.</p>
<button>Enviar</button>
```

El navegador aplica estilos por defecto. Funciona, pero se ve básico.

HTML con CSS

```
h1 { color:#1d4ed8; }
p { color:#475569; }
button {
  background:#2563eb;
  color:white;
}
```

El contenido gana jerarquía visual, contraste y apariencia más cuidada.

Para explicar en vivo:

El HTML no cambió. Lo único que cambió fue la hoja de estilos. Esa separación es una de las bases del desarrollo Front-End.

SINTAXIS

La estructura de una regla CSS

selector { propiedad: valor; }

La regla se lee así: “A los elementos que coincidan con este selector, aplica estas propiedades visuales”.

```
p {  
  color: red;  
  font-size: 18px;  
}
```

Cómo leerlo: el selector apunta al elemento, las llaves agrupan las instrucciones y cada línea interna modifica una característica visual concreta.

LECTURA DE CÓDIGO

Cómo se interpreta una regla

Selector	h1 indica qué elemento del HTML queremos modificar.
Propiedad	color, font-size o background-color indican qué aspecto visual cambia.
Valor	blue, 32px o #f3f4f6 indican cómo queda esa propiedad.
Declaración	Cada par propiedad: valor; es una declaración.

Detalle importante: una declaración incompleta o mal escrita puede hacer que el navegador ignore esa línea. Por eso conviene escribir de forma prolija desde el inicio.

FORMAS DE INCLUIR CSS

Tres formas de aplicar estilos

En línea

Se escribe dentro del atributo `style`. Funciona, pero mezcla contenido y diseño.

Interno

Se escribe en una etiqueta `<style>` dentro del HTML. Útil para ejercicios chicos.

Externo

Se guarda en un archivo `.css`. Es la forma más ordenada y recomendable.

Recomendación práctica:

para ejercicios chicos podemos usar CSS interno, pero desde ahora conviene acostumbrarse al archivo externo porque es lo que más se parece a un proyecto real.

CSS EN LÍNEA

Funciona, pero no conviene abusarlo

```
<h1 style="color: red;">
  Título importante
</h1>

<p style="font-size: 18px;">
  Texto del párrafo
</p>
```

- El estilo queda mezclado con el contenido.
- Si hay muchos elementos, se vuelve repetitivo.
- Modificar el diseño después es más difícil.
- Sirve para pruebas rápidas, no como forma principal.

Problema real: si repetimos estilos en línea, después cambiar un color o tamaño obliga a buscar elemento por elemento. Es rápido al principio, caro después.

CSS INTERNO

Una mejora, pero todavía dentro del HTML

```
<head>
  <style>
    h1 { color: green; }
    p  { color: #555; }
  </style>
</head>
```

Cuándo puede servir

Para una explicación, un ejercicio chico o una prueba rápida. Pero si el sitio crece, conviene separar los estilos en otro archivo.

Uso razonable: el CSS interno sirve para mostrar una demo rápida en una sola página, pero no es ideal cuando el sitio empieza a crecer.

Más ordenado que inline, menos escalable que externo

CSS EXTERNO

La forma recomendada

INDEX.HTML

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

ESTILOS.CSS

```
body {
  font-family: Arial, sans-serif;
}

h1 {
  color: navy;
}
```

Buenas prácticas: el archivo `estilos.css` debería estar vinculado desde el `head` . Si el nombre o la ruta están mal, la página carga sin estilos.

SELECTORES

El selector responde: ¿a quién le aplico el estilo?

Etiqueta

p, h1, button. Afecta a todos los elementos de esa etiqueta.

Clase

.card, .error. Reutilizable y muy usada en CSS real.

ID

#principal. Apunta a un único elemento. Tiene más peso.

Descendente

.card h2. Aplica a elementos dentro de otros elementos.

Pregunta que siempre hay que hacerse: antes de escribir propiedades, definí a quién querés modificar. Esa decisión se toma con el selector.

Seleccionar bien evita estilos accidentales

SELECTOR DE ETIQUETA

Afecta a todos los elementos iguales

```
p {  
  color: #444;  
  font-size: 18px;  
}
```

```
<p>Primer párrafo.</p>  
<p>Segundo párrafo.</p>  
<p>Tercer párrafo.</p>
```

Los tres párrafos reciben el mismo estilo. Es útil para estilos generales, pero peligroso si queremos modificar solo un caso puntual.

Riesgo: el selector de etiqueta es cómodo, pero puede ser demasiado amplio. Si ponés estilo a todos los `p` , afecta todos los párrafos de la página.

Simple, amplio y poco específico

PROBLEMA TÍPICO

No siempre queremos afectar todo

CSS demasiado general

```
p {  
  color: red;  
}
```

Todos los párrafos se vuelven rojos, incluso los que no son mensajes de error.

CSS más preciso

```
.error {  
  color: red;  
}
```

Solo los elementos marcados con la clase `error` se ven rojos.

Lectura del problema: cuando un estilo es demasiado general, terminás modificando partes que no querías. Las clases aparecen para dar precisión.

El selector correcto reduce efectos no deseados

SELECTOR DE CLASE

La herramienta principal para diseñar componentes

```
<p class="error">
  El campo nombre es obligatorio.
</p>
```

```
.error {
  color: red;
  font-weight: bold;
}
```

En HTML se escribe **class="error"**. En CSS se selecciona con punto: **.error**.

Regla mental:

si algo se va a repetir o representa un tipo de componente, usá una clase. Ejemplo: `.boton` , `.card` , `.mensaje-error` .

Clase en HTML · Punto en CSS

REUTILIZACIÓN

Una misma clase puede repetirse

<pre><p class="destacado">Curso de HTML</p> <p class="destacado">Curso de CSS</p> <p class="destacado">Curso de JavaScript</p></pre>	<pre>.destacado { background-color: #fef3c7; padding: 10px; border-left: 4px solid orange; }</pre>
--	--

La ventaja es clara: escribimos el estilo una vez y lo aplicamos donde haga falta.

Por qué importa: una clase reutilizable evita copiar y pegar estilos. Cambiás la clase una vez y se actualizan todos los elementos que la usan.

COMPOSICIÓN

Un elemento puede tener varias clases

```

<p class="mensaje error">
  No se pudo enviar el formulario.
</p>

<p class="mensaje exito">
  Formulario enviado correctamente.
</p>

```

```

.mensaje {
  padding: 12px;
  border-radius: 6px;
  font-weight: bold;
}

.error { color: #991b1b; }
.exito { color: #166534; }

```

Idea de composición: una clase puede dar la estructura general y otra puede cambiar el estado. Por ejemplo: `.mensaje` define la caja; `.error` define el color.

SELECTOR DE ID

El ID identifica un elemento único

```
<section id="principal">
  <h1>Bienvenidos</h1>
</section>
```

```
#principal {
  background-color: #e0f2fe;
  padding: 30px;
}
```

En HTML se escribe **id="principal"**. En CSS se selecciona con numeral: **#principal**. Conviene no abusar de IDs para estilos.

Criterio: el ID sirve para identificar una sección única, pero para estilos suele ser menos flexible. No conviene resolver todo con IDs.

ID: único y con mucha especificidad

CLASE VS ID

Para estilos, normalmente convienenen clases

Clase

Se puede repetir en muchos elementos. Es flexible y reutilizable.


```
.boton { ... }
.card { ... }
```

ID

Está pensado para identificar un elemento único. Tiene más peso y puede complicar conflictos.


```
#menuPrincipal { ... }
#formularioAlta { ... }
```

Decisión práctica: en CSS inicial, priorizá clases. Los IDs dejalos para casos realmente únicos o para navegación interna con anclas.

SELECTOR DESCENDENTE

Seleccionar elementos dentro de otros

```
<section class="noticia">
  <h2>Título de la noticia</h2>
  <p>Texto de la noticia.</p>
</section>
```

```
.noticia h2 {
  color: darkblue;
}
```

Esto significa: “Seleccioná los **h2** que estén dentro de un elemento con clase **noticia**”. No afecta a todos los h2 de la página.

Cómo se lee: `.card h2` significa “los h2 que estén adentro de una card”. No afecta a todos los h2 del documento.

SELECTOR AGRUPADO

Aplicar el mismo estilo a varios selectores

Repetitivo

```
h1 { font-family: Arial; }  
h2 { font-family: Arial; }  
h3 { font-family: Arial; }
```

Mejor

```
h1, h2, h3 {  
  font-family: Arial, sans-serif;  
  color: #111827;  
}
```

La coma agrupa selectores. No significa “dentro de”; significa “a estos selectores por igual”.

Para ordenar código: agrupar selectores evita repetición cuando varios elementos comparten la misma regla visual.

Menos repetición, más claridad

SELECTOR UNIVERSAL

El selector que alcanza a todo

```
* {  
  box-sizing: border-box;  
}
```

- El asterisco selecciona todos los elementos.
- Es útil para configuraciones generales.
- No conviene usarlo para cualquier cosa porque impacta en toda la página.

Un uso común es **box-sizing: border-box;**, porque facilita el cálculo de tamaños cuando luego trabajemos con cajas, márgenes, bordes y espaciado.

Cuidado: el selector universal afecta todo. Es útil para ajustes globales, pero no para resolver estilos específicos.

Poderoso, pero hay que usarlo con criterio

CASCADA

Qué pasa cuando dos reglas compiten

```
p {  
  color: blue;  
}
```

```
p {  
  color: red;  
}
```

Resultado

El texto se ve rojo.

1. Ambas reglas apuntan al mismo elemento.
2. Tienen la misma especificidad.
3. Gana la que aparece después.

La cascada no es azar: si dos reglas tienen la misma fuerza, gana la que aparece más abajo. Por eso el orden del archivo CSS importa.

Si todo pesa igual, gana lo último

CASCADA REAL

CSS también combina reglas

```
.boton {
  background-color: gray;
  color: white;
}

.boton {
  background-color: blue;
}
```

Enviar

El fondo cambia a azul, pero el texto sigue blanco porque la segunda regla no reemplazó esa propiedad.

Detalle fino: CSS no reemplaza todo el bloque anterior; combina declaraciones. Si solo cambiás el fondo, el color de texto anterior puede seguir vigente.

HERENCIA

Algunas propiedades pasan del padre a los hijos

```
body {
  font-family: Arial, sans-serif;
  color: #333;
}
```

```
<body>
  <h1>Título</h1>
  <p>Texto de la página</p>
</body>
```

El título y el párrafo heredan la tipografía y el color desde **body**. Esto evita repetir la misma regla para cada texto.

Ventaja: la herencia permite definir fuente y color general en **body** y no repetirlo en cada título, párrafo o lista.

HEREDA / NO HEREDA

No todas las propiedades se comportan igual

Suelen heredarse

- color
- font-family
- font-size
- line-height
- text-align

No suelen heredarse

- margin
- padding
- border
- background-color
- width / height

Por qué no se hereda todo: si márgenes, bordes y fondos pasaran automáticamente a todos los hijos, cada página sería visualmente inmanejable.

Si todo se heredara, el diseño sería caótico

ESPECIFICIDAD

Cuando no gana el último, gana el más específico

```
p {  
  color: blue;  
}
```

```
<p class="texto">  
  Este párrafo tiene clase.  
</p>
```

```
.texto {  
  color: red;  
}
```

El texto se ve rojo porque **.texto** es más específico que **p**. La clase pesa más que la etiqueta.

La pregunta clave: cuando algo no cambia aunque esté más abajo, probablemente hay un selector más específico ganando.

JERARQUÍA BÁSICA

Peso aproximado de los selectores

1 Etiqueta p, h1, button	2 Clase .texto, .card, .boton	3 ID #principal, #aviso	4 Inline style="..." tiene mucho peso.
---	--	--	---

Escala mental: etiqueta pesa poco, clase pesa más, ID pesa todavía más y el estilo en línea suele imponerse. No abusos de los niveles fuertes.

Para empezar, esta regla mental alcanza

EJEMPLO COMPLETO

¿Qué color gana?

```
<p id="aviso" class="mensaje">
  La inscripción cierra el viernes.
</p>
```

```
p { color: blue; }

.mensaje { color: orange; }

#aviso { color: red; }
```

Respuesta

Gana el rojo, porque **#aviso** es un selector de ID y tiene más especificidad que la clase y que la etiqueta.

Ejercicio de lectura: antes de mirar la respuesta, identifica qué selectores afectan al mismo elemento y ordenalos por fuerza.

ADVERTENCIA

Cuidado con **!important**

```
p {  
  color: red !important;  
}
```

```
.texto {  
  color: blue;  
}
```

- **!important** fuerza una declaración.
- Puede resolver algo rápido, pero ensucia el mantenimiento.
- Si se usa mucho, después nadie entiende por qué gana cada regla.
- Para aprender, conviene evitarlo.

Advertencia: **!important** puede parecer una solución rápida, pero suele ocultar un problema de organización o especificidad.

ERRORES COMUNES

Pequeños errores que rompen el estilo

Olvidar el punto

```
error { color:red; }
```

Esto busca una etiqueta llamada error.

Confundir ID y clase

```
#mensaje  
.mensaje
```

No son iguales. Uno busca ID, el otro clase.

Olvidar ;

```
color: blue  
background: white;
```

Puede cortar la lectura correcta de las declaraciones.

Método para depurar: si un estilo no aparece, revisá primero tres cosas: vínculo al CSS, selector correcto y sintaxis con llaves, dos puntos y punto y coma.

BUENAS PRÁCTICAS

Cómo escribir CSS más mantenible

Usar CSS externo El HTML queda más limpio y los estilos se reutilizan mejor.

Nombrar claro `.boton-principal` comunica más que `.azul` o `.caja1`.

Preferir clases Las clases son más flexibles que los IDs para diseñar componentes.

No abusar Evitar estilos en línea, IDs innecesarios y `!important`.

Profesionalización: CSS mantenible no es solo que “funcione”; es que otro pueda leerlo, entenderlo y modificarlo sin romper todo.

EJEMPLO INTEGRADOR

HTML base: una página de cursos

```

<header class="encabezado">
  <h1>Cursos disponibles</h1>
  <p>Elegí una opción para comenzar a estudiar.</p>
</header>

<main>
  <section class="card">
    <h2>HTML inicial</h2>
    <p>Aprendé a estructurar páginas web.</p>
    <a class="boton" href="#">Ver curso</a>
  </section>

  <section class="card destacada">
    <h2>CSS inicial</h2>
    <p>Aprendé a diseñar páginas web.</p>
    <a class="boton" href="#">Ver curso</a>
  </section>
</main>

```

Lectura del ejemplo: este HTML está preparado para ser diseñado: tiene encabezado, contenido principal, tarjetas y botones con clases claras.

EJEMPLO INTEGRADOR

CSS para esa página

```
body {
  font-family: Arial, sans-serif;
  color: #1f2937;
  background-color: #f3f4f6;
}

.encabezado {
  background-color: #2563eb;
  color: white;
  padding: 30px;
  text-align: center;
}
```

```
.card {
  background-color: white;
  margin: 20px;
  padding: 20px;
  border: 1px solid #ddd;
  border-radius: 12px;
}

.card h2 { color: #1d4ed8; }
.destacada { border: 2px solid #2563eb; }
```

Lectura del CSS: primero se definen estilos generales y después estilos de componentes. Ese orden ayuda a que el archivo sea más fácil de recorrer.

ANÁLISIS

Qué conceptos aparecen en el ejemplo

body

Selector de etiqueta. Define estilos generales que pueden heredarse.

.card

Clase reutilizable para crear cajas visuales de contenido.

.card h2

Selector descendente. Solo afecta títulos dentro de tarjetas.

.destacada

Clase adicional para modificar una tarjeta particular sin duplicar HTML.

Conexión conceptual: en este ejemplo conviven etiqueta, clase, selector descendente, herencia y cascada. No son temas separados: trabajan juntos.

No es magia: cada estilo tiene una razón

FORMULARIO

Aplicamos CSS a lo visto la clase anterior

```

<form class="formulario">
  <label for="nombre">Nombre</label>
  <input id="nombre" type="text">

  <label for="email">Email</label>
  <input id="email" type="email">

  <button class="boton">Enviar</button>
</form>

```

```

.formulario {
  background: white;
  padding: 20px;
  border-radius: 12px;
}

.formulario label {
  display: block;
  margin-top: 12px;
}

.formulario input {
  width: 100%;
  padding: 10px;
}

```

Conexión con la clase anterior: los formularios ya tenían estructura correcta; ahora CSS mejora legibilidad, separación y apariencia de controles.

MINI EJERCICIO GUIADO

Resolver estilos básicos

Consigna

Dado este HTML, escribir CSS para que el título sea azul, la presentación tenga 18px, el dato sea gris y el botón tenga fondo azul con texto blanco.

```
<h1>Mi perfil</h1>
<p class="presentacion">Soy estudiante de desarrollo web.</p>
<p class="dato">Me interesa aprender Front-End.</p>
<a class="boton" href="#">Contactar</a>
```

Consigna mínima: no hace falta que quede perfecto visualmente. Lo importante es que el CSS esté separado, sea legible y use clases con sentido.

Primero pensar selectores, después escribir propiedades

POSIBLE SOLUCIÓN

Una forma correcta de resolverlo

```

h1 {
  color: blue;
}

.presentation {
  font-size: 18px;
}

.dato {
  color: gray;
}

.boton {
  background-color: blue;
  color: white;
  padding: 10px 14px;
  text-decoration: none;
}

```

Comparación esperada: puede haber muchas soluciones correctas. Lo que se evalúa es que el selector elegido corresponda al HTML y que el resultado sea coherente.

Puede haber otras soluciones válidas

ACTIVIDAD PRÁCTICA

Crear una página con CSS externo

Enunciado

1. Crear un archivo **index.html**.
2. Crear un archivo **estilos.css**.
3. Vincular el CSS con `<link rel="stylesheet" href="estilos.css">`.
4. Diseñar una página con encabezado, sección de presentación, tabla y formulario.
5. Usar al menos tres clases CSS y un selector descendente.

Actividad integradora: esta práctica obliga a unir lo visto: estructura HTML, clases, archivo externo y reglas CSS básicas.

La entrega debe mostrar estructura y estilo separados

CRITERIOS

Qué voy a mirar en la actividad

Vinculación Que el archivo CSS externo esté correctamente conectado al HTML.	Selectores Que se usen etiquetas, clases y al menos un selector descendente.	Orden Que el código sea legible, con nombres claros y sin estilos mezclados innecesariamente.
--	--	---

Criterio de corrección: un diseño simple pero ordenado vale más que muchos colores aplicados sin criterio.

No se evalúa “que quede perfecto”, sino que esté bien construido

ORGANIZACIÓN

Cómo ordenar el archivo CSS

1. Base

Estilos generales: body, tipografías, colores base y fondo.

2. Layout

Zonas grandes: header, main, section, footer.

3. Componentes

Card, botón, formulario, tabla, mensaje.

4. Ajustes

Casos puntuales y estados visuales.

Orden sugerido: estilos generales arriba, componentes después y ajustes específicos al final. Eso reduce conflictos innecesarios.

CSS ordenado se corrige más fácil

NOMBRES DE CLASES

Nombrar bien también es programar

Poco claro

- .caja1
- .rojo
- .grande
- .nuevo

El nombre describe una apariencia suelta o no dice nada.

Más claro

- .card-curso
- .boton-principal
- .mensaje-error

El nombre explica el rol del elemento dentro de la interfaz.

Nombres útiles: una buena clase dice qué representa el elemento, no solo cómo se ve hoy. Mejor `.boton-principal` que `.azul`.

CASCADA

Cómo decidir qué regla gana

Orden de análisis

1. Primero miro si el selector es más específico.
2. Si tienen la misma especificidad, gana el que aparece después.
3. Si una propiedad no se redefine, se conserva el valor anterior.
4. Si aparece `!important`, rompe la lógica normal y por eso conviene evitarlo.

Procedimiento: cuando haya conflicto, no pruebes al azar. Mirá especificidad, orden de aparición y si hay propiedades heredadas involucradas.

EJERCICIO RÁPIDO

¿Qué se hereda?

```
main {
  color: #333;
  font-family: Arial;
  border: 2px solid black;
  padding: 20px;
}
```

Pregunta

Dentro de `main` hay un `h1` y un `p`. ¿Qué reciben los hijos?

1. Reciben color y tipografía.
2. No reciben automáticamente borde ni padding.

Respuesta esperada: se heredan principalmente propiedades de texto. Los bordes y fondos quedan en el elemento donde fueron declarados.

EJERCICIO RÁPIDO

¿Qué selector gana?

```

button {
  background: gray;
}
.boton {
  background: blue;
}
#enviar {
  background: green;
}

```

```

<button id="enviar" class="boton">
  Enviar
</button>

```

Respuesta

Gana `#enviar`, porque el selector de ID tiene más especificidad que la clase y que la etiqueta.

Respuesta esperada:

si el botón tiene clase, la clase suele ganar frente al selector de etiqueta. Si aparece un ID, probablemente gane el ID.

FORMULARIO

Un formulario ya empieza a parecer interfaz

```

.formulario {
  max-width: 520px;
  margin: 30px auto;
  background: white;
  padding: 24px;
  border-radius: 14px;
}

```

```

.formulario input {
  width: 100%;
  padding: 12px;
  border: 1px solid #ccc;
  border-radius: 8px;
}

```

Detalle visual: con pocas reglas un formulario pasa de “HTML crudo” a una interfaz más clara. Todavía no vimos layout avanzado, pero ya mejora mucho.

TABLAS

También podemos mejorar tablas

```
table {
  width: 100%;
  border-collapse: collapse;
}
th, td {
  border: 1px solid #ddd;
  padding: 12px;
}

th {
  background: #eef2ff;
  color: #1e1b4b;
  text-align: left;
}
```

Aplicación concreta: las tablas sin CSS suelen ser difíciles de leer. Bordes, espaciado y encabezados ayudan a interpretar la información.

Las tablas no tienen por qué verse crudas

REGLA ÚTIL

Un pequeño reset inicial

```
* {
  box-sizing: border-box;
}
body {
  margin: 0;
  font-family: Arial, sans-serif;
}
```

Para qué sirve

Ayuda a trabajar con tamaños más predecibles y elimina el margen por defecto del navegador.

No arregla todo, pero da un punto de partida más controlado.

Por qué sirve: `box-sizing: border-box` hace más predecible el tamaño de las cajas. Lo vamos a valorar más cuando veamos box model.

DEMO SUGERIDA

Orden para mostrarlo en vivo

Secuencia recomendada

1. Abrir un HTML sin CSS.
2. Crear `estilos.css`.
3. Vincularlo con `link`.
4. Aplicar estilos a `body`.
5. Crear clases para `card`, botón y formulario.
6. Mostrar un conflicto de especificidad y resolverlo.

Para la demo: conviene mostrar primero la página sin estilos. Así se entiende mejor qué aporta cada regla que se va agregando.

CHECKLIST

Antes de entregar la actividad

Archivos

Están index.html y estilos.css en la misma carpeta.

Vínculo

El HTML tiene correctamente el link al CSS.

Clases

Hay clases claras y reutilizables.

Orden

El código se puede leer sin adivinar qué quiso hacer el autor.

Antes de subir: si el CSS no carga, casi siempre es por ruta incorrecta o archivo mal nombrado. Revisar eso ahorra mucho tiempo.

PUENTE

Lo que falta para diseñar mejor

Color

Vamos a elegir paletas y contrastes con más criterio.

Espaciado

Vamos a controlar márgenes, padding y separación visual.

Box model

Vamos a entender cómo se calcula el tamaño real de cada caja.

Puente: hoy aprendimos a aplicar reglas. La próxima clase vamos a controlar con más precisión medidas, espacios y cajas.

CIERRE CONCEPTUAL

Qué nos tiene que quedar

Ideas clave

- CSS permite controlar la presentación visual del HTML.
- Una regla CSS tiene selector, propiedad y valor.
- Las clases son fundamentales para reutilizar estilos.
- La cascada, la herencia y la especificidad explican por qué gana una regla.

CSS no es “poner colores”: es darle forma visual a una interfaz.

Próxima clase: colores, tipografías, tamaños, márgenes, bordes, espaciado y box model.

Cierre: cuando algo “no se ve como esperaba”, CSS casi siempre tiene una explicación: selector, cascada, herencia o especificidad.