“CSS Inheritance”

¿Qué es CSS Inheritance (Herencia en CSS)?

La **herencia en CSS** es el proceso mediante el cual **ciertas propiedades aplicadas a un elemento padre se transmiten automáticamente a sus elementos hijos**.
Esto significa que, si se define, por ejemplo, un color de texto o una fuente en el <body>, esos valores se aplicarán también a todos los elementos dentro de él (como párrafos, encabezados o listas), **a menos que se sobrescriban** con estilos propios.

En resumen:

La herencia en CSS hace que algunos estilos se propaguen hacia abajo en la jerarquía del documento.

Ventajas de la herencia en CSS

1. **Menos código repetido:** no es necesario aplicar el mismo estilo a cada elemento hijo.
2. **Diseños más consistentes:** mantiene una apariencia uniforme en toda la página.
3. **Fácil mantenimiento:** cambiar el estilo del padre actualiza automáticamente a los hijos.
4. **Mayor legibilidad:** el CSS se vuelve más limpio y organizado.

Desventajas

1. **No todas las propiedades son heredables:** solo algunas (como color o font-family) se heredan por defecto.
2. **Sobreescritura involuntaria:** si no se tiene cuidado, un cambio global puede afectar muchos elementos.
3. **Difícil de rastrear en proyectos grandes:** los valores heredados pueden venir de varios niveles superiores.

Casos de uso comunes

- Definir estilos de **texto, color o fuente** globales en el <body>.
- Mantener una **apariencia uniforme** en secciones o contenedores.
- Simplificar el CSS en diseños con **muchos elementos anidados**.

Ejemplo

Html:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>CSS Inheritance</title>
  <link rel="stylesheet" href="css.css">
</head>
<body>
  <h1>Título principal</h1>
  <p>
    Este texto hereda el color y la fuente del <code>body</code>.
    <a href="#">Este enlace usa su propio color.</a>
  </p>

  <div class="padre">
    Contenedor padre ,es el de color azul.
    <div class="hijo">
      Hijo (color por defecto, pero borde heredado)
```

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

CSS:

```
/* --- Herencia general del body --- */
body {
  font-family: 'Segoe UI', sans-serif;
  color: #2c3e50; /* Se hereda a h1 y p */
 }

 /* --- Elemento que sobrescribe color heredado --- */
 a {
  color: #e74c3c;
  text-decoration: none;
 }

 /* --- Ejemplo de herencia forzada y controlada --- */
 .padre {
  color: darkblue;
  border: 3px solid darkblue;
  padding: 15px;
 }

 .hijo {
  border: inherit;  /* hereda el borde del padre */
```

```
  color: initial;   /* restaura color por defecto */
}
```

Propiedades que sí se heredan normalmente

Categoría	Propiedades heredables
Texto	color, font-family, font-size, font-style, font-weight, letter-spacing, line-height, text-align
Listas	list-style-type, list-style-position, list-style-image
Otros	visibility, cursor, quotes

Propiedades que NO se heredan por defecto

Categoría	Propiedades no heredables
Caja (Box model)	margin, padding, border, width, height
Fondo	background-color, background-image
Posicionamiento	position, top, left, z-index
Display / Grid / Flex	display, grid-template, flex-direction

Resumen general

Concepto	Detalle
Nombre	CSS Inheritance (Herencia en CSS)
Tipo	Mecanismo de propagación de propiedades
Ventajas	Menos código, consistencia visual, mantenimiento fácil
Desventajas	No todas las propiedades se heredan, posibles conflictos
Casos comunes	Estilos de texto, color, fuente, consistencia de diseño
Palabras clave	inherit, initial, unset

“Responsive Design”

¿Qué es Responsive Design (Diseño Adaptable)?

El **Responsive Design** o **Diseño Responsivo** es una técnica de desarrollo web que permite que una página **se adapte automáticamente al tamaño y tipo de pantalla** del dispositivo (computadora, tableta o teléfono).
Su objetivo es que **el contenido siempre se vea bien**, sin importar el tamaño o la orientación de la pantalla.

Esto se logra usando **media queries**, **unidades flexibles** (como %, vh, vw, fr) y **diseños fluidos** con herramientas como **Flexbox** o **CSS Grid**.

Ventajas

1. **Mejor experiencia de usuario** en cualquier dispositivo.
2. **Un solo sitio** se adapta a todos los tamaños de pantalla.
3. **Mejor posicionamiento SEO**, ya que Google prioriza sitios responsivos.
4. **Mantenimiento más fácil**, sin necesidad de versiones separadas para móvil y escritorio.

Desventajas

1. **Más trabajo inicial** en el diseño y pruebas.
2. **Posibles problemas de rendimiento** si no se optimizan bien las imágenes.
3. **Diseños complejos** pueden requerir múltiples ajustes o media queries adicionales.

Casos de uso comunes

- Sitios web que deben funcionar bien tanto en **computadoras como en móviles**.
- **Tiendas en línea**, blogs o portafolios.
- **Aplicaciones web** o paneles administrativos con interfaz adaptable.
- **Landing pages** que deben mantener un diseño atractivo sin importar el dispositivo.

Ejemplo de código

HTML :

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Responsive Design</title>
```

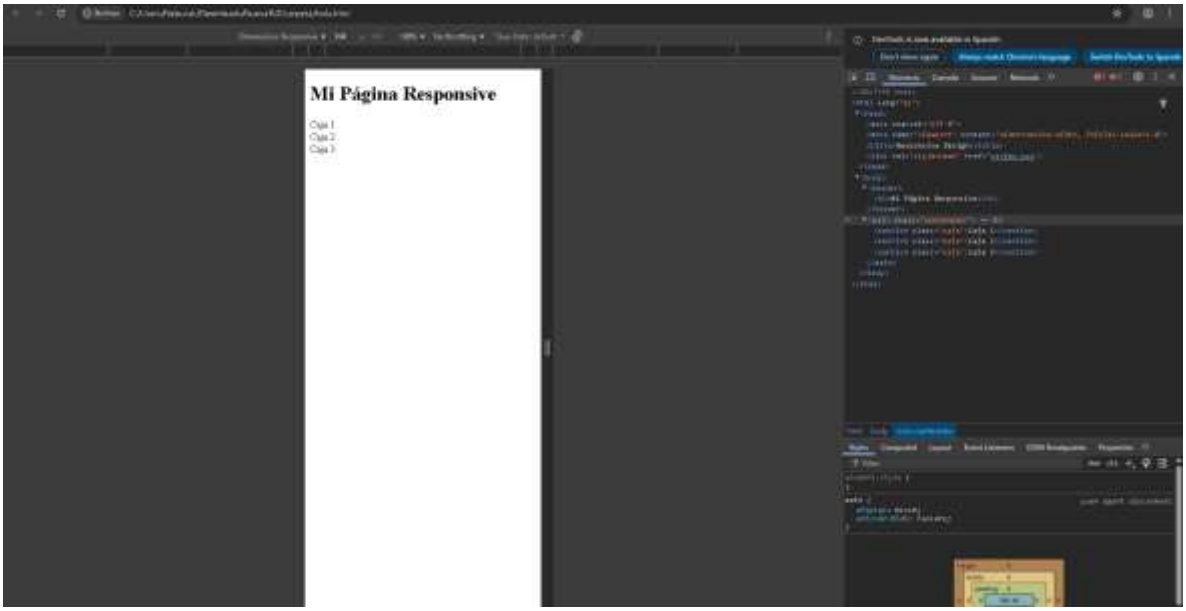
```
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <header>
    <h1>Mi Página Responsive</h1>
  </header>

  <main class="contenedor">
    <section class="caja">Caja 1</section>
    <section class="caja">Caja 2</section>
    <section class="caja">Caja 3</section>
  </main>
</body>
</html>
```

CSS :

```
/* --- Estilos generales --- */
body {
  font-family: Arial, sans-serif;
  margin: 0;
  background: #f5f5f5;
  text-align: center;
}
```

```
header {
  background: #3498db;
  color: white;
  padding: 20px;
}

/* --- Contenedor con tres cajas --- */
.contenedor {
  display: flex;
  justify-content: center;
  gap: 20px;
  padding: 20px;
}

.caja {
  background: #2ecc71;
  color: white;
  padding: 30px;
  flex: 1;
  border-radius: 8px;
}
```

```
/* --- Diseño responsive: las cajas se apilan en pantallas pequeñas --- */
@media (max-width: 600px) {
  .contenedor {
    flex-direction: column; /* cambia de fila a columna */
  }
}
```



Resumen general

Concepto	**Detalle**
Nombre	Responsive Design (Diseño Adaptable)
Definición	Técnica que adapta el diseño web a distintos tamaños de pantalla
Ventajas	Mejor experiencia, un solo sitio, SEO optimizado, mantenimiento fácil
Desventajas	Más trabajo inicial, posibles ajustes extra

Casos comunes	Páginas web, tiendas, blogs, paneles y landing pages
Herramientas comunes	Media queries, Flexbox, Grid, unidades flexibles (%, vh, vw, fr)

Animations

¿Qué son las CSS Animations?

Las **CSS Animations** permiten crear **movimientos y transiciones visuales** en los elementos de una página web sin usar JavaScript.
Estas animaciones se definen con la regla @keyframes, que describe cómo cambia el estilo de un elemento a lo largo del tiempo.

En pocas palabras, permiten **dar vida a los elementos**: moverlos, cambiar sus colores, agrandarlos, rotarlos, etc.

Ventajas

1. **Más atractivas visualmente**: hacen la experiencia del usuario más dinámica.
2. **Rendimiento optimizado**: las animaciones CSS son procesadas por el navegador de forma eficiente.
3. **Fáciles de aplicar**: no requieren JavaScript.
4. **Control detallado**: se pueden ajustar duración, retraso, repeticiones y dirección.

Desventajas

1. **Animaciones complejas** pueden volverse difíciles de controlar solo con CSS.
2. **Mal uso** puede afectar el rendimiento en dispositivos débiles.

3. **Menor control lógico** comparado con animaciones hechas en JavaScript.

Casos de uso comunes

- Efectos de entrada o salida de elementos (como menús o modales).
- Indicadores de carga o botones interactivos.
- Banners o anuncios animados.
- Destacar elementos importantes (como mensajes o alertas).

Ejemplo de código

HTML

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>CSS Animations</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <h1 class="titulo">Animación con CSS</h1>
  <div class="caja"></div>
</body>
</html>
```

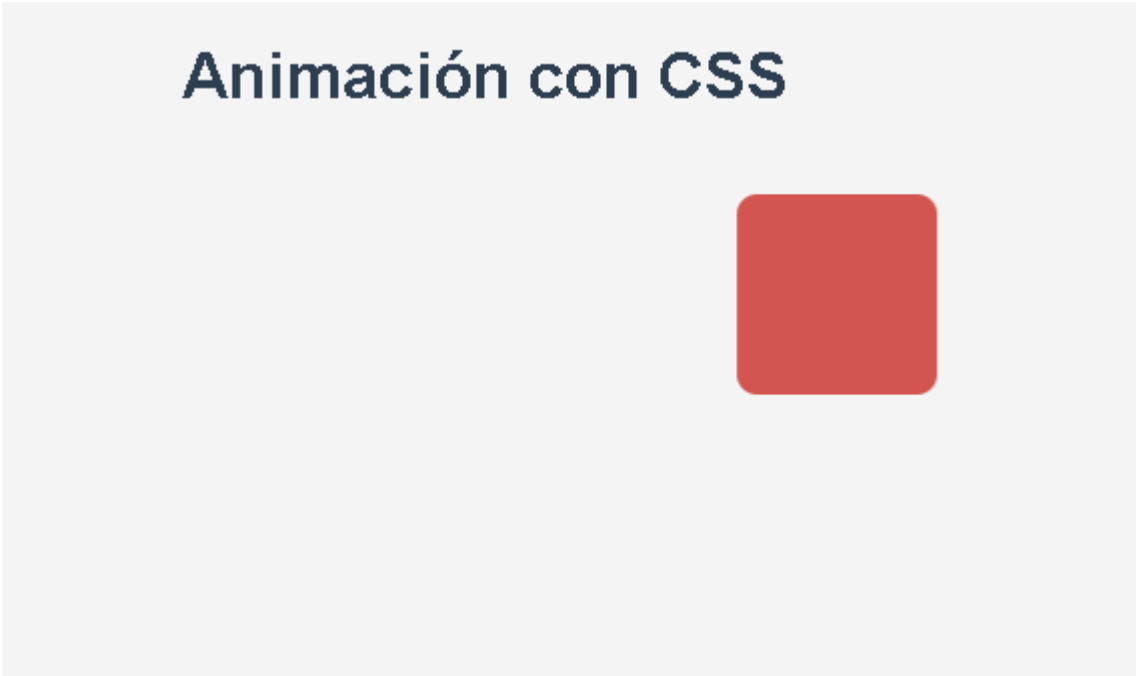
CSS

```
/* --- Estilos generales --- */
body {
  font-family: Arial, sans-serif;
  text-align: center;
  margin-top: 50px;
  background: #f4f4f4;
}

.titulo {
  color: #2c3e50;
}

/* --- Caja que se animará --- */
.caja {
  width: 100px;
  height: 100px;
  background: #3498db;
  margin: 40px auto;
  border-radius: 10px;

  /* Activamos la animación */
```

```
  animation: mover 3s infinite alternate ease-in-out;
}

/* --- Definición de la animación con keyframes --- */
@keyframes mover {
  from {
    transform: translateX(0);
    background: #3498db;
  }
  to {
    transform: translateX(200px);
    background: #e74c3c;
  }
}
```

Qué hace el ejemplo:

- La caja azul se mueve suavemente hacia la derecha, cambia a rojo y vuelve, repitiéndose infinitamente.



Resumen general

Concepto	**Detalle**
Nombre	CSS Animations
Definición	Permiten crear movimientos visuales mediante la regla @keyframes

Ventajas	Fluidez, rendimiento alto, fácil de aplicar, control total de tiempo
Desventajas	Difíciles de manejar si son muy complejas, menos control lógico
Casos comunes	Efectos visuales, botones, loaders, banners
Propiedades clave	animation-name, animation-duration, animation-iteration-count, animation-delay, animation-timing-function

“ CSS Variables”

¿Qué son las CSS Variables?

Las **CSS Variables** (también llamadas *Custom Properties*) son valores personalizados que puedes definir en un lugar del código y reutilizar en todo el archivo CSS.
Se declaran con el prefijo -- y se accede a ellas usando la función var().

Esto permite **mantener estilos más organizados, fáciles de cambiar y coherentes** a lo largo del diseño.

Ventajas

1. **Reutilización de valores** (colores, tamaños, fuentes, etc.).
2. **Mantenimiento más fácil**, ya que solo cambias el valor en un lugar.
3. **Temas dinámicos** (modo oscuro/claro, cambio de color principal, etc.).
4. **Compatibles con JavaScript**, se pueden modificar en tiempo real.

Desventajas

1. **No son compatibles con navegadores muy antiguos** (como IE11).
2. **Mal uso puede complicar el código**, si se abusa de demasiadas variables.
3. **No son constantes**, su valor depende del *scope* donde se declaran.

Casos de uso comunes

- Definir **paletas de colores globales**.
- Crear **temas personalizables** (modo oscuro o claro).
- Mantener coherencia entre **tamaños, bordes o fuentes**.
- Cambiar estilos dinámicamente con JavaScript.

Ejemplo de código

HTML

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>CSS Variables</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <header>
```

```
    <h1>Uso de Variables CSS</h1>
  </header>

  <main class="contenedor">
    <section class="caja">Caja 1</section>
    <section class="caja">Caja 2</section>
  </main>
</body>
</html>
```

CSS

```
/* --- Declaración de variables globales --- */
:root {
  --color-principal: #3498db;
  --color-secundario: #2ecc71;
  --color-texto: #ffffff;
  --radio-borde: 10px;
  --espaciado: 20px;
}

/* --- Estilos generales --- */
body {
  font-family: Arial, sans-serif;
```

```
  background: #f5f5f5;
  margin: 0;
  text-align: center;
}

header {
  background: var(--color-principal);
  color: var(--color-texto);
  padding: var(--espaciado);
}

/* --- Cajas reutilizando las variables --- */
.contenedor {
  display: flex;
  justify-content: center;
  gap: var(--espaciado);
  padding: var(--espaciado);
}

.caja {
  background: var(--color-secundario);
  color: var(--color-texto);
  padding: var(--espaciado);
```

```
  border-radius: var(--radio-borde);
  flex: 1;
}
```

Qué hace el ejemplo:

- Define variables globales en :root (como colores y bordes).
- Las reutiliza en todo el CSS, haciendo fácil cambiar el tema o color base sin tocar varias líneas.

Resumen general

Concepto	**Detalle**
Nombre	CSS Variables (Custom Properties)
Definición	Valores personalizados que se pueden reutilizar en todo el CSS
Ventajas	Reutilización, fácil mantenimiento, temas dinámicos, integración con JS
Desventajas	Incompatibilidad con navegadores antiguos, abuso puede confundir

Casos comunes	Paletas de colores, temas oscuros/claros, estilos dinámicos
Sintaxis básica	--nombre-variable: valor; → uso con var(--nombre-variable)

React (visión general)

React es una **librería de JavaScript** creada por **Meta (Facebook)** que se utiliza para construir **interfaces de usuario** de manera **declarativa** y **basada en componentes**.

Su idea principal es dividir la interfaz en **componentes independientes** que gestionan su propio estado y luego se combinan para formar aplicaciones completas.

React utiliza un sistema llamado **Virtual DOM**, que es una representación ligera del DOM real del navegador. Esto le permite **actualizar solo las partes necesarias** de la página cuando cambian los datos, haciendo el proceso más eficiente.

El código de React suele escribirse con una sintaxis llamada **JSX**, que permite combinar HTML y JavaScript de manera más intuitiva dentro de los componentes.

Además, React es **solo la capa de vista** dentro de una aplicación (es decir, se encarga únicamente de mostrar los datos y actualizar la interfaz). Sin embargo, puede integrarse con otras herramientas para manejar rutas, datos o lógica más compleja.

React se utiliza tanto para aplicaciones web como móviles (a través de **React Native**) y es una de las tecnologías más populares en el desarrollo moderno de interfaces.

Angular (visión general)

Angular es un **framework de desarrollo web** creado y mantenido por **Google**, diseñado para construir **aplicaciones web dinámicas y de una sola página (SPA)**.

A diferencia de React, que es una librería, Angular es un **framework completo**, lo que significa que ofrece todo lo necesario para desarrollar una aplicación: manejo de rutas, comunicación con servidores, validación de formularios, inyección de dependencias, entre otros.

Angular utiliza **TypeScript** (un superconjunto de JavaScript) como lenguaje principal, lo que permite escribir código más estructurado y con tipado estático.

Su arquitectura se basa en el uso de **componentes**, **módulos** y **servicios**, lo que facilita la organización del código y la reutilización de elementos dentro de la aplicación.

También incluye un sistema de **data binding** (enlace de datos) que permite sincronizar automáticamente el estado de la interfaz con los datos del programa.

Angular ofrece una herramienta llamada **Angular CLI (Command Line Interface)** que ayuda a crear, compilar y desplegar proyectos de forma rápida y automatizada.

Django (visión general)

Django es un **framework de desarrollo web de alto nivel** escrito en **Python**, diseñado para crear aplicaciones web de forma rápida, segura y escalable.

Sigue el patrón **MTV (Model-Template-View)**, una variante del clásico MVC, donde:

- **Model** maneja la estructura y conexión con la base de datos.
- **Template** define la presentación o interfaz del usuario.

- **View** controla la lógica que conecta los modelos con las plantillas.

Django incluye muchas herramientas integradas, como un **sistema de autenticación**, un **panel de administración automático**, manejo de formularios, validaciones y un **ORM (Object Relational Mapper)** que permite trabajar con bases de datos usando código Python en lugar de SQL directo.

Además, está diseñado con el principio de **"Don't Repeat Yourself (DRY)"**, que promueve escribir menos código redundante y mantenerlo más limpio.

Django es mantenido por la comunidad de código abierto y la **Django Software Foundation**, y se utiliza en numerosos proyectos profesionales y educativos.

Flask (visión general)

Flask es un **microframework de desarrollo web** escrito en **Python**, diseñado para crear aplicaciones web de manera **simple y ligera**.

A diferencia de frameworks más completos como Django, Flask se centra únicamente en **proporcionar las herramientas básicas para crear un servidor web y gestionar rutas**. Todo lo demás, como bases de datos, autenticación o plantillas avanzadas, se puede agregar mediante **extensiones opcionales**, lo que ofrece gran flexibilidad al desarrollador.

Flask utiliza el patrón **Wsgi (Web Server Gateway Interface)** y generalmente se combina con **Jinja2**, un motor de plantillas que permite generar HTML dinámico a partir de datos del servidor.

Es ampliamente usado para **APIs REST**, prototipos rápidos y aplicaciones web sencillas, gracias a su simplicidad y facilidad de aprendizaje.

Ruby on Rails (visión general)

Ruby on Rails (o simplemente **Rails**) es un **framework de desarrollo web** escrito en **Ruby**, diseñado para crear aplicaciones web de manera **rápida y estructurada**.

Rails sigue el patrón **MVC (Model-View-Controller)**:

- **Model**: gestiona la lógica de los datos y la base de datos.
- **View**: define la presentación o interfaz que ve el usuario.
- **Controller**: maneja la lógica que conecta los modelos con las vistas.

Rails incluye muchas herramientas integradas, como **sistema de enrutamiento**, **gestión de base de datos con ActiveRecord**, **migraciones**, y un **sistema de plantillas** para generar HTML dinámico.

Está diseñado con los principios de **"Convención sobre Configuración"** (minimiza la necesidad de configuraciones manuales) y **"Don't Repeat Yourself (DRY)"** (promueve la reutilización y claridad del código).

Rails es mantenido por la comunidad de Ruby y se utiliza en aplicaciones web de todo tipo, desde prototipos rápidos hasta proyectos a gran escala.

Vue.js (visión general)

Vue.js es un **framework progresivo de JavaScript** para construir interfaces de usuario.
El término *progresivo* significa que puedes usar Vue **solo en una parte de la página** o integrarlo en aplicaciones completas según sea necesario.

Vue se basa en **componentes**, lo que permite dividir la interfaz en piezas reutilizables con su propio estado y lógica.
Usa una sintaxis declarativa y un **sistema de enlace de datos (data binding)** que mantiene sincronizada la interfaz con los datos del programa.

Además, Vue cuenta con un **DOM virtual** que optimiza la actualización de la interfaz, y puede integrarse con herramientas del ecosistema para **rutas**, **estado global** y **renderizado del lado del servidor**.

Vue está diseñado para ser **simple de aprender** y **flexible**, permitiendo a los desarrolladores usarlo desde proyectos pequeños hasta aplicaciones SPA complejas.