15 CSS: Especificidad (Specificity)

Una vez que la Cascada ha determinado el origen y la importancia (`!important`) de las reglas CSS en conflicto, el siguiente factor decisivo es la **Especificidad** del selector utilizado en cada regla.

La Especificidad es, esencialmente, una **puntuación o peso** que el navegador asigna a cada selector CSS. Cuando dos o más reglas válidas (del mismo origen e importancia) intentan aplicar un valor a la misma propiedad del mismo elemento, **la regla con el selector de mayor especificidad será la que prevalezca.**

Piensa en la especificidad como una medida de cuán "específico" es un selector al apuntar a un elemento. Un selector de ID es más específico que un selector de clase, que a su vez es más específico que un selector de etiqueta.

Cómo se Calcula la Especificidad:

Los navegadores calculan la especificidad utilizando (generalmente) un sistema de tres o cuatro componentes. Una forma común de visualizarlo es como `(A, B, C)` o a veces `(A, B, C, D)`, donde:

- **A (IDs):** Cuenta el número de **selectores de ID** (`#mild`) en el selector compuesto.
- **B (Clases, Atributos y Pseudoclases):** Cuenta el número total de **selectores de clase** (`.miClase`), **selectores de atributo** (`[type="text"]`, `[href^="https://"]`) y **pseudoclases** (`:hover`, `:focus`, `:nth-child()`, etc.).
- **C (Elementos y Pseudoelementos):** Cuenta el número total de **selectores de tipo (etiqueta)** (`p`, `div`, `h1`) y **pseudoelementos** (`::before`, `::after`, `::first-letter`, etc.).
- *(D - Inline Styles):* A veces se incluye una categoría implícita aún más alta para los **estilos en línea** (`style="..."`). Estos casi siempre ganan sobre las reglas en hojas de estilo (a menos que se use `!important`). Por simplicidad, a menudo se consideran por separado o como el valor más alto (ej: 1,0,0,0 si usamos 4 componentes).

Comparación de Puntuaciones - ¡NO ES BASE 10!

Este es el punto **más crucial** para entender la especificidad: la comparación de estas puntuaciones **no es como comparar números normales**. Se compara componente por componente, de izquierda a derecha (A, luego B, luego C).

1. Compara los valores de **A (IDs)**. El selector con el valor de A más alto gana, **sin importar** los valores de B y C.
2. Si los valores de A son iguales, compara los valores de **B (Clases, etc.)**. El selector con el valor de B más alto gana, **sin importar** el valor de C.
3. Si los valores de A y B son iguales, compara los valores de **C (Elementos)**. El selector con el valor de C más alto gana.
4. Si A, B y C son todos iguales, entonces se aplica el último criterio de la cascada: el **Orden en el Código Fuente**.

Ejemplo:

- Un selector con especificidad `(0, 1, 0)` (una clase) **siempre** ganará a un selector con especificidad `(0, 0, 11)` (once elementos), porque `1` en la columna B es mayor que `0` en la columna B. ¡No importa que 11 sea mayor que 1 en términos numéricos generales!

- Un selector con especificidad `(1, 0, 0)` (un ID) **siempre** ganará a un selector con especificidad `(0, 25, 15)` (veinticinco clases/atributos/pseudo-clases y quince elementos), porque `1` en la columna A es mayor que `0`.

Valores de Especificidad de Selectores Comunes:

Tipo de Selector	Cálculo (A, B, C)	Ejemplo de Selector	Especificidad Resultante
Estilo en Línea	(Implícito Alto)	`style="..."`	`(1, 0, 0, 0)` (si 4 comp)
ID	+1 en A	`#nav`	`(1, 0, 0)`
Clase	+1 en B	`.boton`	`(0, 1, 0)`
Atributo	+1 en B	`[type="checkbox"]`	`(0, 1, 0)`
Pseudoclase	+1 en B	`:hover`	`(0, 1, 0)`
Elemento (Tipo)	+1 en C	`div`	`(0, 0, 1)`
Pseudoelemento	+1 en C	`::before`	`(0, 0, 1)`
Selector Universal	+0 (Ninguno)	`*`	`(0, 0, 0)`
Combinadores (`+` , `>` , `~` , ` `)	+0 (Ninguno)	`div > p`	(Suma de `div` y `p`)
Pseudoclase `:where()`	+0 (Ninguno)	`:where(.clase)`	`(0, 0, 0)`
Pseudoclases `:is()` , `:not()` , `:has()`	(Variable)	`:is(.a, #b)` `:not(.c)`	(Especificidad del argumento *más específico* dentro de ellas)
Herencia	N/A	(Valor heredado)	(No tiene especificidad)

Ejemplos de Cálculo de Especificidad:

- `p { ... }` → `(0, 0, 1)` (Un elemento)
- `a:hover { ... }` → `(0, 1, 1)` (Una pseudoclase `:hover` = B, un elemento `a` = C)
- `.menu li a { ... }` → `(0, 1, 2)` (Una clase `.menu` = B, dos elementos `li` , `a` = C)
- `#sidebar h2.titulo { ... }` → `(1, 1, 1)` (Un ID `#sidebar` = A, una clase `.titulo` = B, un elemento `h2` = C)
- `body #content .data img:hover { ... }` → `(1, 2, 2)` (Un ID `#content` = A, una clase `.data` y una pseudoclase `:hover` = B, dos elementos `body` , `img` = C)
- `[data-tipo="importante"] { ... }` → `(0, 1, 0)` (Un selector de atributo = B)
- → `(0, 0, 0)` (Selector universal)

Resolviendo un Conflicto con Especificidad:

Revisemos el ejemplo del inicio de la unidad:

```
<p id="intro" class="destacado">Este es un párrafo importante.</p>
```

```
/* Regla 1: Selector 'p' */
p { color: black; } /* Especificidad: (0, 0, 1) */

/* Regla 2: Selector '.destacado' */
.destacado { color: blue; } /* Especificidad: (0, 1, 0) */
```

```
/* Regla 3: Selector '#intro' */
#intro { color: red; } /* Especificidad: (1, 0, 0) */
```

1. Todas son reglas del autor, sin `!important`.
2. Comparamos especificidades:
   - `#intro` tiene `(1, 0, 0)`
   - `.destacado` tiene `(0, 1, 0)`
   - `p` tiene `(0, 0, 1)`
3. Comparando la columna A (IDs): `1` es mayor que `0`.
4. **Resultado:** La regla `#intro { color: red; }` gana porque su selector tiene la mayor especificidad. El texto del párrafo será **rojo**.

Consideraciones Adicionales:

- **`!important`:** Como vimos en la Cascada, una declaración con `!important` **ignora la especificidad**. Solo será sobrescrita por otra declaración también con `!important` de mayor prioridad en el origen (Usuario > Autor) o igual prioridad pero posterior en el código.
- **Estilos en Línea:** Tienen una especificidad implícita muy alta (1,0,0,0), superando a IDs, clases y elementos definidos en hojas de estilo externas o internas (a menos que se use `!important` en estas últimas).
- **`:where()` vs `:is()`:** La pseudoclase `:where()` es especial porque *no añade especificidad*, permitiendo crear selectores complejos sin aumentar su peso. En contraste, `:is()` toma la especificidad de su argumento más específico.
