CSS

Selettori

Un selettore CSS è una **regola che individua gli elementi HTML** a cui applicare uno stile.

- Tipo: `p`, `div`, `h1`, `h2`, ecc...
- Classe: `.classe`
- ID: `#id`
- Attributo: `type="text"`
- Discendenza:
  - Discendente: `div p`
  - Figlio Diretto: `div > p`
  - Fratello Adiacente: `div + p`
  - Fratelli Generici: `div ~ p`

Pseudo-Classi

Parole chiavi precedute da `:` che si aggiungono a un selettore per definire uno stato particolare di un elemento senza modificarne l'html.

- `:hover`: mouse sopra l'elemento
- `:focus`: quando l'elemento ha il focus
- `:first-child`: primo figlio di un genitore

Ereditarietá

Meccanismo per cui alcuni stili si trasmettono automaticamente dagli elementi genitori ai figli senza ridefinirli

```
body{
    color: red;
}
```

Tutti i testi all'interno del body, quindi anche quelli contenuti all'interno di un <p> o avranno font color rosso, a meno che non vengano ridefiniti.

Cascade

Meccanismo con cui il browser decide quali regole CSS applicare quando ce ne sono piú di una per lo stesso elemento.
L'ordine di priorità:
1. Stile di default del browser
2. Regole CSS esterne
3. Regole CSS interne <Style>
4. Stili Inline style="Qualcosa"
5. Regole con `!important`

Se ci sono più regole con uguale importanza, prevale quella con **più specificità**. Se anche la specificità è uguale, vince **l'ultima dichiarazione** letta dal browser.

Specificitá

Regola che il browser utilizza per decidere quale stile di applicare quando ci sono piú regole in conflitto sullo stesso elemento.
Il calcolo é molto semplice:

Tipo Selettore	Valore Specificitá
Inline Style (style=" ")	1000
ID (#id)	100
Classe (.classe)	10
Tipo Elemento (p, h1, h2, ecc...)	1
Universale (*)	0

Esempio Calcolo Specificitá:

```
/* Specificità = 1 */
p {
  color: blue;
}

/* Specificità = 10 */
.intro {
  color: red;
}

/* Specificità = 100 */
#header {
  color: green;
}
```

```
/* Specificità = 1000 */
<div style="color: black;">...</div>
```

In questo caso, se questi stili si appliccassero ad un `<p id="header class="intro style="color: black;>"`, il colore finale del font sará nero perché la specificitá dello style inline ha valore piú alto

> **Importante**
>
> Le specificitá possono anche essere sommate:
>
> ```
> p .class{
>     color: black
> }
> ```
>
> In questo caso avremmo una specificitá di valore 11, 1 per selettore tipo e 10 per selettore classe

Formattazione del Testo

```
body{
    font-family: "Times New Roman", Times, serif;
}
```

- `"Times New Roman"` → Prima scelta: Font contenenti gli spazi → Va inserito tra virgolette
- `Times` → Seconda scelta, nel caso il primo non sia disponibile
- `serif` → Scelta generica di fallback, si usa se i font precedenti non sono supportati

> **Importante**
> Le famiglie generiche di font (`serif` , `sans-serif` , `mono-space` , `cursive` , ecc) non hanno maiuscole

Font Ospitati Localmente:

- Salvare il file del font nel tuo progetto
- Usare `@font-face` nel CSS per dichiarare il font
- Il browser scarica i file font dal tuo server e li usa per mostrare il testo

```
@font-face{
    font-family: 'MioFont';
    src: url('fonts/MioFont.ttf') format(ttf);
}
body{
    font-family: 'MioFont', sans-serif;
}
```

Font Ospitati Esternamente:

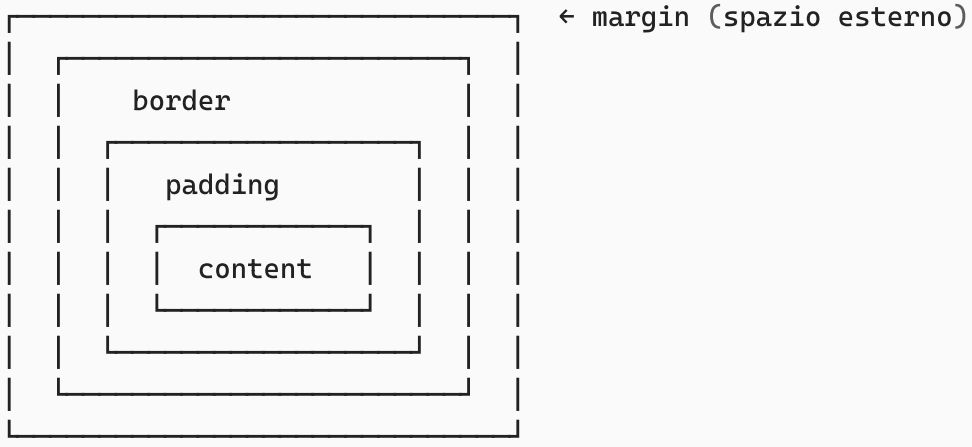
- I font non sono nel sito, ma su un server remoto
- Si usano link o import da CDN (Content Delivery Network)
- La tua pagina richiede i font via internet e il browser li scarica

```
HTML
<link href="https://fonts.googleapis.com/css2?family=Roboto&display=swap"
rel="stylesheet">
```

```
CSS
body {
  font-family: 'Roboto', sans-serif;
}
```

Box

Modello a scatola con cui il browser rappresenta ogni elemento HTML
Ogni elemento é visto come un rettangolo composto da quattro aree concentriche:



- *Content:* Contenuto vero e proprio,come testo, immagini, ecc
- *Padding:* Spazio interno tra il contenuto e il bordo

- *Border:* Il bordo visibile dell'elemento
- *Margin:* Spazio esterno tra l'elemento e ció che lo circonda

```
div {
  width: 200px;
  height: 100px;
  padding: 10px;
  border: 2px solid black;
  margin: 20px;
}
```

Posizione degli elementi

Mediante la regola `position` é possibile assegnare agli elementi determinate posizioni all'interno della pagina web.

- `static` → elementi visualizzati regolarmente all'interno del documento (Regola di Default)
- `relative` → elemento spostato rispetto alla posizione static
- `absolute` → elemento posizionato rispetto al contenitore
- `fixed` → elemento posizionato rispetto alla finestra web

Flexbox-Display

Proprietá che trasforma un contenitore in un contenitore flessibile, permettendo di gestire in modo semplice il layout deglie elementi figli.

```
.container{
    display: flex;
}
```

- Gli elementi figli di `.container` diventeranno flex items.
- L'impostazione predefinita é orizzontale

Proprietá principali:

Proprietá	Descrizione
`flex-direction`	Direzione degli elementi: `row` (Default), `column`, `row-reverse`, `column-reverse`
`justify-content`	Allineamento Orizzontale

Proprietá	Descrizione
`align-items`	Allineamento Verticale
`flex-wrap`	Se gli elementi devono andare a capo `wrap` o restare su riga `nowrap`

```css
.container {
  display: flex;
  flex-direction: row;
  justify-content: space-between;
  align-items: center;
}
```

Dispositivi Mobile

I dispositivi mobile possono presentare caratteristiche dello schermo, come risoulzione e dimensione, molto diverse fra loro. Questo implica che durante la costruzione della pagina web, un'intera sezione della progettazione va dedicata alla costruzione delle regole CSS che rendano il contenuto piú flessibile. L'obiettivo é quello di fornire layout diversi a seconda della dimensione della finestra di visualizzazione.
L'approccio basato sul Responsive Web Design si occupa di questo.
Tecniche:

- Controllo viewport
- Controllo layout con media query
- Stili fluidi → utilizzo di `%` come unitá di misura per definire la grandezza degli elementi

> **Nota**
>
> Strategia mobile-first: Definire prima le regole CSS per i mobile e poi tramite `@media(min-width:_)` impostare le regole per i desktop.