



*Modulo 3*

# Introduzione a CSS

*Capitolo 1*

# Cos'è il CSS

CSS è un linguaggio che serve a "vestire" le pagine web. Con il CSS possiamo cambiare colori, font, spaziature e layout.

Ci permette inoltre di rendere il nostro sito responsive e quindi accessibile da ogni tipologia di dispositivo.



HTML



HTML + CSS

# Perche serve il CSS?

Per evitare questo:

- <http://www.milliondollarhomepage.com/>
- <https://www.spacejam.com/1996/>
- <https://info.cern.ch/>
- <https://www.cfg.com/>
- <https://scaruffi.com/>
- <https://spork.org/>
- <https://web.archive.org/web/20191022005906/http://history.crs4.it/>

# Come inserire il CSS

CI sono 3 tipologie di inserimento:

- Inline
- Incorporato (Embedded)
- Esterno (Single)

# Inline

Il CSS inline viene inserito direttamente all'interno dell'elemento HTML utilizzando l'attributo Style.



```
<span style="color:#f19f19;">Francesco</span>
```

# Incorporato (Embedded)

Il CSS embedded viene inserito direttamente all'interno del tag <style> di HTML nella parte superiore delimitata da <head>.

```
<head>
  <style>
    .testo-rosso{
      color:#f19f19;
    }
  </style>
</head>
```

# Esterno (Single)

Il CSS esterno viene definito in file separati con estensione .css e viene collegato al documento HTML utilizzando il tag <link> all'interno della sezione <head>.

All'interno di questo tag sono indispensabili due attributi:



```
<link rel="stylesheet" href="style.css">
```

# Come si scrive il CSS: I Selettori

Il CSS può essere paragonato alla famosa “mira assistita” di un qualsiasi spara-tutto: puntiamo a un **elemento specifico** e **applichiamo la modifica stilistica** desiderata, proprio come un player mira a un bersaglio e fa fuoco.

Scegliamo quale elemento HTML modificare attraverso i cosiddetti **selettori**, un elemento chiave che permette di selezionare - appunto - e stilizzare specifiche parti di un documento HTML.



# La sintassi dei Selettori

Dopo aver scelto il selettore che desideriamo utilizzare, apriamo un blocco di dichiarazioni con le parentesi graffe {}.

All'interno di questo blocco, inseriamo delle coppie **proprietà-valore**, dove la proprietà specifica la regola CSS da applicare e il valore indica l'effetto che vogliamo ottenere.

```
h1 {  
    color: orange;  
}
```

# Tipologia Selettori: Universale (\*)

Il **selettore universale** in CSS è rappresentato dall'asterisco (\*). Quando viene utilizzato, corrisponde a tutti gli elementi all'interno del documento HTML.

```
* {  
    font-family: "Sofia", sans-serif;  
}
```

# Tipologia Selettori: TAG

Il **selettore per tag** è uno dei tipi di selettori più basilari in **CSS**. Consiste semplicemente nel selezionare tutti gli elementi HTML di un determinato tipo.



```
div {  
    font-size: 40px;  
}
```

# Tipologia Selettori: Classe

Il **selettore per classe** è un tipo di selettore in **CSS** che consente di selezionare tutti gli elementi HTML che hanno un determinato valore nell'attributo class.

Le **classi in HTML** sono utilizzate per raggruppare elementi che hanno caratteristiche simili e consentono di applicare le stesse regole di stile a più elementi senza dover ripetere lo stesso codice **CSS**.

```
.ciccio {  
    font-size: 40px;  
}
```

# Tipologia Selettori: ID

Il selettore per ID in CSS permette di selezionare un elemento HTML specifico basandosi sul suo attributo ID.

È importante ricordare che ogni elemento HTML può avere un solo ID, e che questo deve essere univoco all'interno del documento, fornendo un modo diretto per stilizzare l'elemento scelto.

```
#ciccio {
```

```
    font-size: 40px;
```

```
}
```

---

```
<span id="ciccio"> QUI </span>
```

# Tipologia Selettori: Discendenza

I selettori per discendenza sono un tipo di selettori CSS che consentono di selezionare un elemento HTML in base alla sua relazione di discendenza rispetto a un altro elemento all'interno della struttura del documento HTML.

Questi selettori permettono di applicare regole di stile a elementi che sono figli diretti, nipoti, o discendenti in generale di un elemento specifico.

```
div > p {  
    color: red;  
}  
  
<div>  
    <p>Test 1</p>  
</div>  
  
<div>  
    Test 2  
</div>
```

# Tipologia Selettori: Multi-selettore

Il multi-selettore in CSS è una funzionalità che consente di applicare le stesse regole di stile a più elementi, separandoli da virgole all'interno della dichiarazione CSS.



```
div, p, span {  
    color: red;  
}
```

# Principi del CSS

1. - Principio della Cascata (Cascading);
2. - Principio di Ereditarieta' (Inheritance);
3. - Principio di Specificita' (Specificity).

# Princípio della Cascata

La natura a cascata di CSS è una regola democratica che serve a determinare come vengono applicati gli stili quando, una o piu' regole, mirano allo stesso elemento.

La cascata risolve questi conflitti seguendo un ordine specifico:

1. **Importanza:** Gli stili marcati con *!important* sovrascrivono qualsiasi regola in conflitto, indipendentemente dalla specificità o dall'ordine;
2. **Specificità:** Le regole con una specificità più alta prendono il sopravvento su quelle con una specificità più bassa, anche se appaiono più tardi;
3. **Ordine:** Quando due regole hanno la stessa specificità, la regola definita più tardi nel foglio di stile prende il sopravvento.

# Princípio della Cascata: Importanza

**Importanza:** Gli stili marcati con *!important* sovrascrivono qualsiasi regola in conflitto, indipendentemente dalla specificità o dall'ordine;

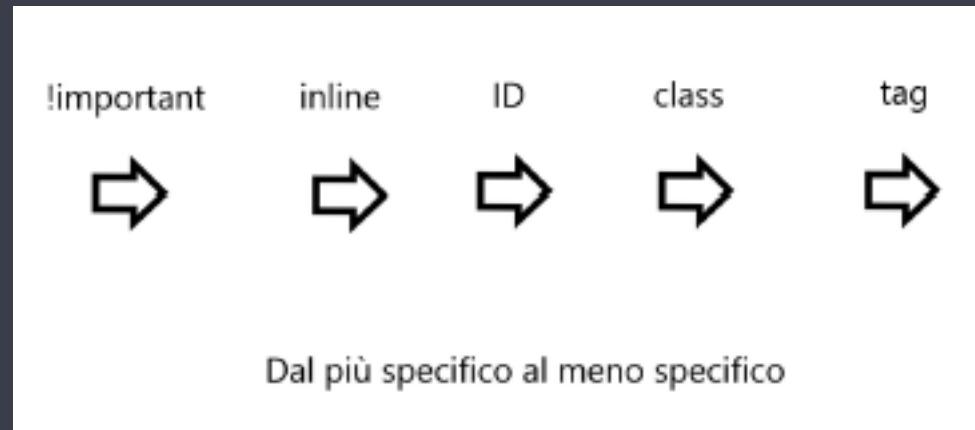
```
● ● ●

.colore-testo{
    color: red!important;
}

.colore-testo{
    color: yellow;
}
```

# Princípio della Cascata: Specificità

**Specificità:** Le regole con una specificità più alta prendono il sopravvento su quelle con una specificità più bassa, anche se appaiono più tardi.



# Princípio della Cascata: Ordine

Quando due regole hanno la stessa specificità, la regola definita più tardi nel foglio di stile prende il sopravvento:

Ultimo Piglia tutto!

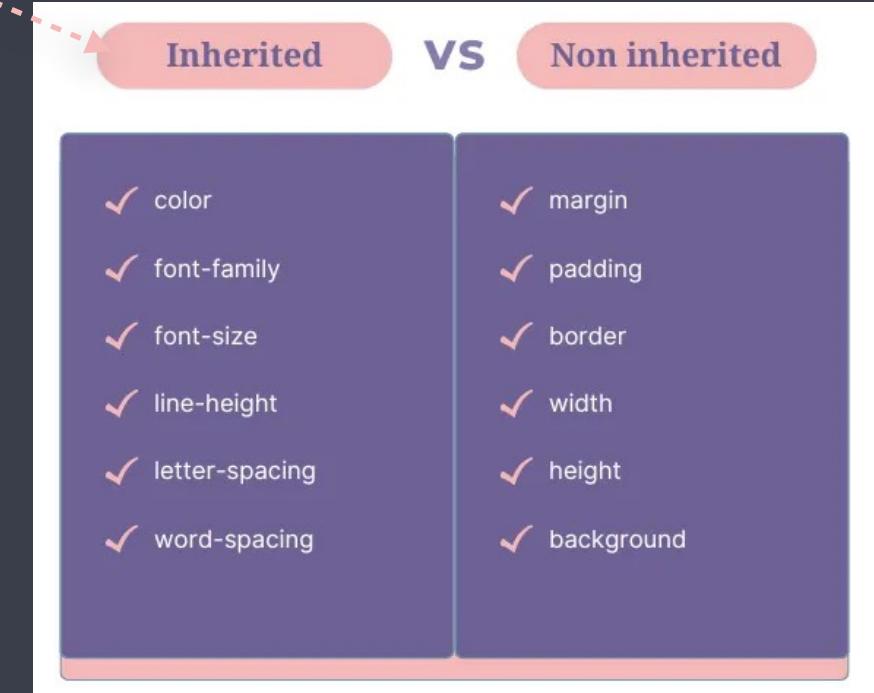
```
● ● ●  
.card {  
    color: blue;  
}  
  
.card {  
    color: red;  
}
```

# Principio di Ereditarietà

CSS ha un meccanismo di ereditarietà (*inherited*) che permette agli stili di essere trasmessi dagli elementi genitori agli elementi figli.

Ciò semplifica la stilizzazione consentendo di definire stili generali a un livello superiore e applicarli automaticamente ai discendenti.

Non tutti però!



# Principio di Specificita'

Ogni tipo di selettore ha un “peso”:

- `#id` → 100 punti
- Classi, attributi, pseudoclassi: `.classe`, `[type="text"]`, `:hover` → 10 punti
- Tag (elementi): `div`, `p`, `h1` → 1 punto
- Inline style: (es. `style=""`) → 1000 punti
- `!important` → Sovrascrive tutto (ma è meglio evitarlo se possibile)

```
.branding {  
    color: blue;  
}  
  
button {  
    color: red;  
}  
  
—  
  
<button class="branding">Ciao, Fra!</button>
```

# Proprietà del CSS: color

La proprietà `color` definisce il colore del testo di un elemento. Non influisce sul colore di sfondo o dei bordi.

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

La proprietà `color` può accettare vari tipi di valori:

- Nomi dei colori (in inglese!): red, blue, black, white...
- **Codici HEX** (esadecimali): #ff0000, #00ff00, #000000...
- **RGB**: rgb(255, 0, 0)
- **RGBA**: rgba(255, 0, 0, 0.5) (include la trasparenza)
- **HSL (*tonalità* (Hue), *saturazione* (Saturation) e *luminosità* (Lightness))**: hsl(120, 100%, 50%)

# Proprietà del CSS: height e width

Le proprietà height e width definiscono l'altezza (height) e la larghezza (width) di un elemento HTML( div, img, section...).

```
.quadrato {  
    width: 150px;  
    height: 150px;  
}
```

# Come indicare le dimensioni di Height e Width

- **Pixel (px)**: Questo è un valore assoluto che specifica la dimensione in pixel. Ad esempio, height: 200px imposta l'altezza dell'elemento a 200 pixel.
- **Percentuale (%)**: Questo è un valore relativo, imposta la dimensione dell'elemento in base a una percentuale della dimensione del suo contenitore genitore. Ad esempio, height: 50% imposta l'altezza dell'elemento al 50% dell'altezza del suo contenitore genitore.
- **REM (Root em)**: Imposta l'altezza o la larghezza dell'elemento in base alla dimensione del carattere radice (html). Un valore 1rem corrisponde alla dimensione del carattere radice. Questo è particolarmente utile quando si desidera impostare dimensioni in relazione alla dimensione del testo di base della pagina. Ad esempio, width: 10rem imposta la larghezza dell'elemento a 10 volte la dimensione del carattere radice.
- **Viewport (vw, vh)**: Il viewport è la finestra visualizzata sullo schermo di un dispositivo che mostra il contenuto di una pagina web. Questi sono valori relativi alla dimensione della finestra del browser, noti come viewport. 1vw corrisponde a 1% della **larghezza del viewport (larghezza dello schermo)**, mentre 1vh corrisponde a 1% dell'**altezza del viewport(altezza dello schermo)**. Ad esempio, width: 50vw imposta la larghezza dell'elemento alla metà della larghezza dello schermo su cui si sta visualizzando.

# min-height e min-width

Le proprietà min-height e min-width sono utilizzate per impostare la dimensione minima dell'altezza e della larghezza di un elemento HTML, rispettivamente.

Queste proprietà sono utili quando si desidera garantire che un elemento non si riduca al disotto di una determinata dimensione, garantendo una buona leggibilità o presentazione del contenuto.

```
.card {  
    min-width: 300px;  
    min-height: 150px;  
}
```

# Proprietà del CSS: background

La proprietà `background` è utilizzata per specificare uno o più aspetti dello sfondo di un elemento HTML.

Può essere utilizzata per impostare il colore di sfondo, un'immagine di sfondo, la ripetizione dell'immagine, la posizione, il colore del testo, e altre caratteristiche relative allo sfondo di un elemento.

Ecco le diverse proprietà collegate alla proprietà `background`:

- **`background-color`**: Imposta il colore di sfondo dell'elemento.
- **`background-image`**: Imposta un'immagine di sfondo per l'elemento. Questa immagine di sfondo può essere qualsiasi immagine desiderata, come un'immagine fotografica, un'illustrazione, una texture o un motivo decorativo. Necessita di una funzione `url()` all'interno del quale va inserito il percorso dal quale prendere l'immagine, che può essere sia assoluto che relativo. Ad esempio `background-image url('sfondo.jpg')`

# Proprietà del CSS: background

- **background-size**: Controlla la dimensione dell'immagine di sfondo rispetto all'elemento.
- **background-repeat**: Specifica come l'immagine di sfondo deve essere ripetuta o posizionata se è più piccola dell'elemento.
- **background-position**: Imposta la posizione iniziale dell'immagine di sfondo all'interno dell'elemento.
- **background-attachment**: Specifica se l'immagine di sfondo deve scorrere con il contenuto dell'elemento o rimanere fissa mentre il contenuto scorre.
- **background**: È una proprietà che consente di impostare tutte o alcune delle proprietà sopra elencate in un'unica dichiarazione.
- **linear-gradient**: È una funzione CSS che consente di creare transizioni fluide di colore da un colore all'altro lungo una direzione specificata. Questa funzione è ampiamente utilizzata per creare sfondi di elementi HTML, bordi, ombreggiature e altri effetti visivi.

# Proprietà del CSS: background

```
● ● ●

.immaginone {

background-color: #f0f0f0; /* colore di sfondo di base */
background-image: url('immagini/sfondo.jpg'),
    linear-gradient(to bottom right, rgba(255, 255, 255, 0.7), rgba(0, 0, 0, 0.5));
background-size: cover; /* l'immagine copre tutto l'elemento */
background-repeat: no-repeat; /* l'immagine non si ripete */
background-position: center center; /* centrata orizzontalmente e verticalmente */
background-attachment: fixed; /* lo sfondo resta fisso mentre la pagina scorre */

}

/* Oppure Shorthand */

.immaginone {
background:
    url('immagini/sfondo.jpg') center center / cover no-repeat fixed,
    linear-gradient(to bottom right, rgba(255,255,255,0.7), rgba(0,0,0,0.5));
}
```

# Proprietà del CSS: opacity

Questa proprietà è utilizzata per controllare il livello di trasparenza di un elemento HTML e dei suoi contenuti. Questa proprietà accetta un valore compreso tra 0 e 1, dove 0 rappresenta la **totale trasparenza** (elemento completamente invisibile) e 1 rappresenta l'**opacità completa** (nessuna trasparenza, elemento completamente visibile).

Accetta anche un dato percentuale (per esempio, 0.5 sarà equivalente a 50%).

```
● ● ●  


Questo paragrafo ha opacity: 1 (completamente visibile).



Questo paragrafo ha opacity: 0.5 (semi-trasparente).



Questo paragrafo ha opacity: 0.1 (quasi invisibile).


```

# Proprietà del CSS: font-family

Questa proprietà è utilizzata per specificare il tipo di carattere o i tipi di carattere preferiti per il testo all'interno di un elemento HTML.

Consente di specificare una lista di nomi di caratteri o famiglie di caratteri, in ordine di priorità, per consentire al browser di utilizzare il primo carattere disponibile dalla lista.



# Proprietà del CSS: font-family

Puoi importare una "famiglia" diversa usando Google Font: <https://fonts.googleapis.com/>

Ad esempio:

```
<head>
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link
        href="https://fonts.googleapis.com/css2?family=Silkscreen:wght@400;700&display=swap"
        rel="stylesheet">
<style>
    .silkscreen-regular {
        font-family: "Silkscreen", sans-serif;
        font-weight: 400;
        font-style: normal;
    }
</style>
</head>
<body>
    <h2 class="silkscreen-regular">Ciao ciao</h2>
</body>
```

# Proprietà del CSS: font-style

La proprietà font-style definisce lo stile con cui appare il testo:

Può avere questi valori

- font-style: normal;
- font-style: italic;
- font-style: oblique;
- font-style: oblique 16deg;

```
.testo-torre-di-pisa{  
    font-style: italic;  
}
```

# Proprietà del CSS: font-weight

La proprietà font-weight definisce **lo spessore** (peso o “ciccionaggine”) del testo. Visualizzare il testo come **normale**, **grassetto**, **più leggero** o con **valori numerici precisi**.

Può avere valori testuali:

- normal → peso normale ( $\approx 400$ )
- bold → grassetto ( $\approx 700$ )
- lighter, bolder → relativi al genitore

Oppure valori numerici:

- 100: Ultra Light
- 300: Light
- 400: Normal
- 500: Medium
- 700: Bold
- 900: Black

# Proprietà del CSS: font-size

Questa proprietà è utilizzata per impostare la dimensione del carattere del testo all'interno di un elemento HTML.

Essa controlla la grandezza del carattere, influenzando la sua altezza e larghezza visibile:

```
● ● ●  


Questo testo ha font-size: 16px (dimensione predefinita del browser).



Questo testo ha font-size: 24px (grande).



Questo testo ha font-size: 2rem (2 volte la dimensione base).


```

# Proprietà del CSS: text-align

Allinea in orizzontale il testo all'interno del suo contenitore, ad esempio a sinistra, a destra, al centro, o giustificato.

I valori accettati piu' comuni sono:

- **left**, allinea il testo a sinistra. E' il valore predefinito della maggior parte dei browser;
- **right**, allinea il testo a destra;
- **center**, centra il testo orizzontalmente all'interno dell'elemento contenitore;
- **justify**, giustifica il testo allineando il margine sinistro e destro;
- **start**, allinea il testo verso l'inizio della direzione di scrittura specificata (ad esempio, a sinistra per le lingue che scrivono da sinistra verso destra);

# Proprietà del CSS: text-align

```
● ● ●

<p style="text-align: left;">
    Testo allineato a sinistra. Il comportamento predefinito nella maggior parte dei browser.
</p>
<p style="text-align: right;">
    Testo allineato a destra. Utile ad esempio per didascalie o date.
</p>
<p style="text-align: center;">
    Testo allineato al centro orizzontalmente nel suo contenitore.
</p>
<p style="text-align: justify;">
    Testo giustificato. I margini sinistro e destro sono allineati.
</p>
<p style="text-align: start;">
    Testo allineato ad inizio rigo secondo la direzione del testo.
</p>
```

# Esercizio 1

A traccia libera:

- Recupera il progetto HTML visto nel Modulo 1
- Implementa tutte le proprietà CSS viste fino ad ora (senza utilizzare Framework)
- Utilizza tutti i principi di design e le risorse viste a lezione/extra

# Esercizi 2

Realizza un **blog (generico, scegli tu l'argomento)** con **almeno 2 pagine HTML** **collegate tra loro**, ispirandoti a un vero blog oppure inventando uno stile personale (es. viaggi nello spazio, cucina, sport ecc.).

## Pagina 1: index.html – Homepage del blog

Realizza una **Navbar** in alto con link (anche fintizi) ad altre sezioni del blog (es. Home, Articoli, Contatti).

Devi inserire:

- **Titolo principale** con il nome del blog.
- Tre card di anteprima viaggio, ognuna con:
  - Un'immagine rappresentativa del viaggio
  - Un sottotitolo (es. "3 giorni su Marte")
  - Un breve testo introduttivo
  - Un **link** (ad esempio un pulsante o a href) che porta alla **pagina dell'articolo**

### Stili richiesti (in CSS):

- Usa un font-family personalizzato a piacere
- Imposta dimensioni del testo con font-size

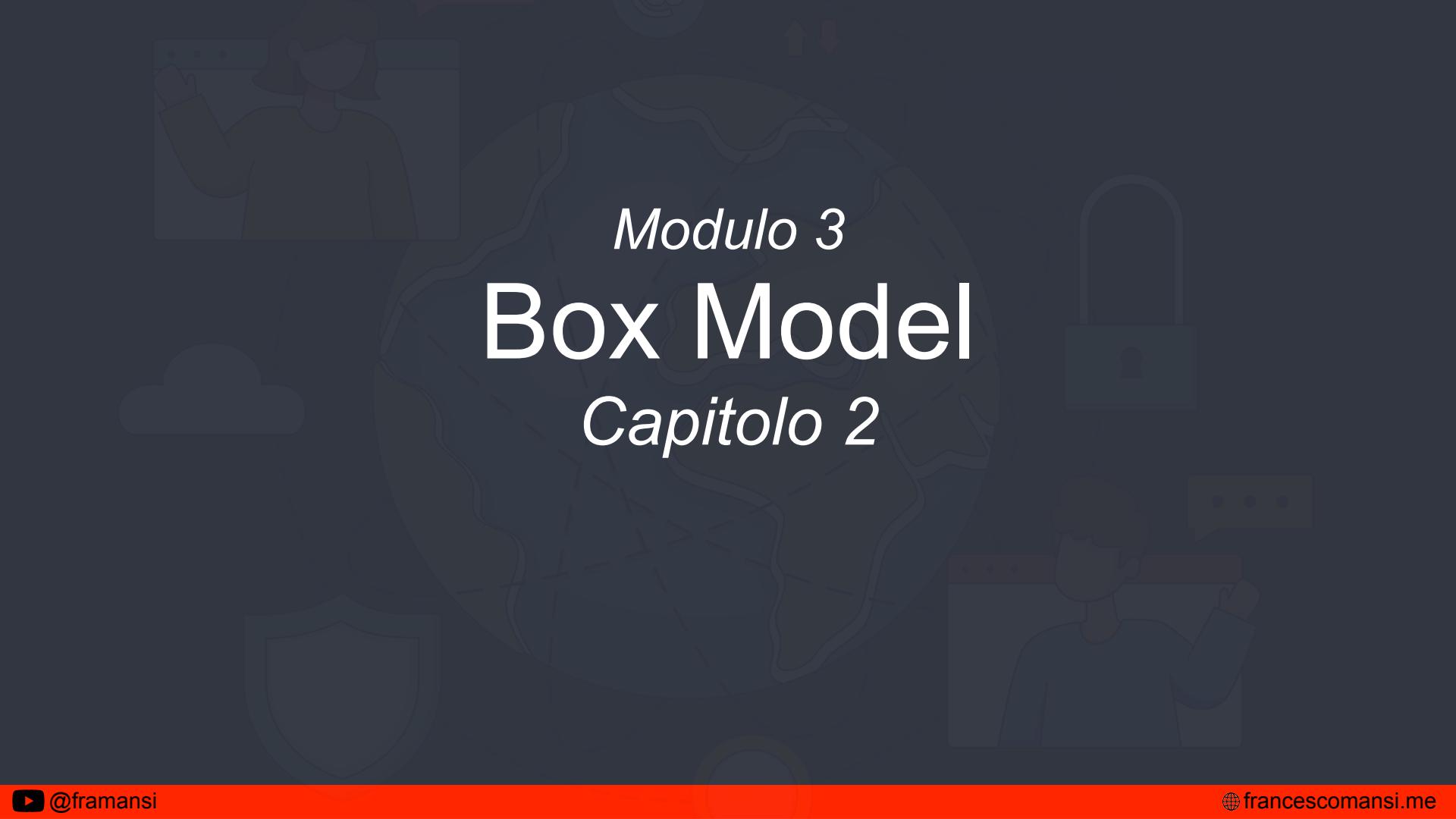
## Pagina 2: articolo.html – Pagina articolo singolo

Contenuti richiesti:

- Immagine in evidenza della destinazione
- **Lista puntata** con l'itinerario del viaggio (minimo 3 tappe)
- **Paragrafo descrittivo** con una breve narrazione della destinazione

### Stili richiesti (in CSS):

- Titolo con font-weight: bold e font-size maggiore
- Allineamento del testo (text-align) e uso di line-height
- Scopri text-shadow e text-decoration



# *Modulo 3*

# Box Model

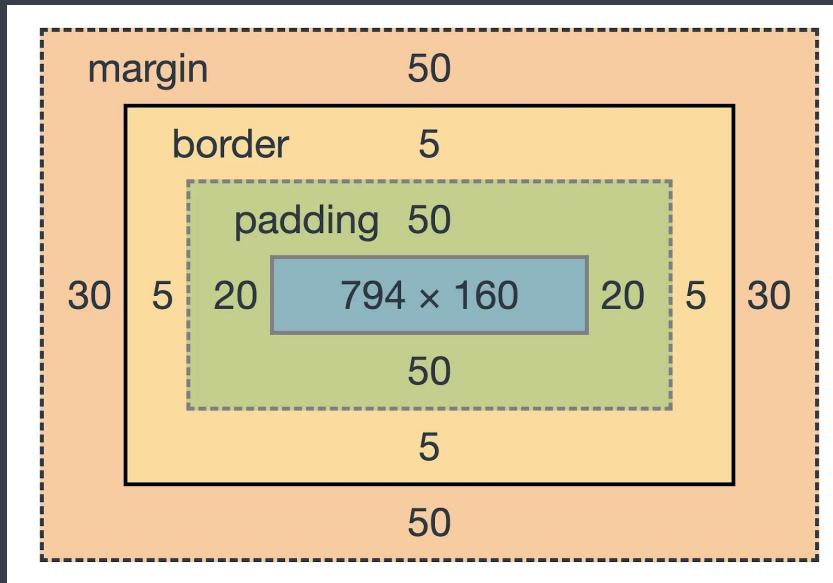
## *Capitolo 2*

# Box Model

Il box model è un concetto fondamentale nel design delle pagine web che descrive la struttura di ogni elemento HTML come un contenitore rettangolare che comprende il contenuto, il padding, il bordo e il margine.

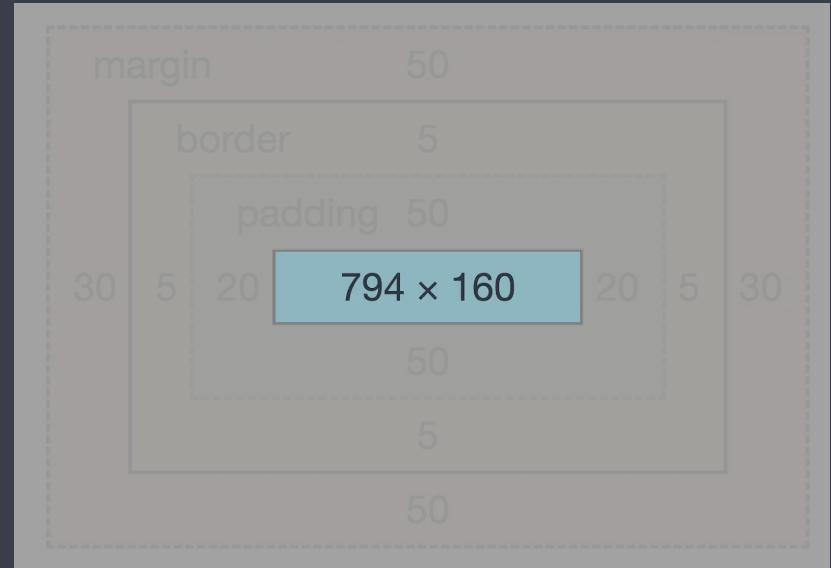
Pensa al box model come alla cornice di un quadro:  
ogni elemento HTML è il dipinto, la tela,  
all'interno di questa cornice, e il box model ne  
definisce la struttura.

Il box model è formato da diversi elementi:  
Content, padding, border e margin.



# Box Model: Content

**Content (Contenuto):** Questa è la parte interna del box, che contiene il contenuto effettivo dell'elemento, come testo, immagini, o altri elementi HTML.



# Box Model: Padding

Lo spazio tra il contenuto dell'elemento e il suo bordo. Il padding è opzionale e consente di creare spazio vuoto intorno al contenuto dell'elemento, contribuendo a separare il contenuto dal bordo, dando piu' respiro all'elemento. Il padding può essere valorizzato utilizzando diverse unità di misura per specificare la dimensione dello spazio e può essere specificato da 1 a 4 valori.



# Box Model: Padding

**Un singolo valore:** Specifica lo stesso valore di padding per tutti e quattro i lati dell'elemento (in alto, a destra, in basso, a sinistra).

**Due valori:** Il primo valore specifica il padding in alto e in basso, mentre il secondo valore specifica il padding a destra e a sinistra.

**Tre valori:** Il primo valore specifica il padding in alto, il secondo valore specifica il padding a destra e a sinistra, mentre il terzo valore specifica il padding in basso.



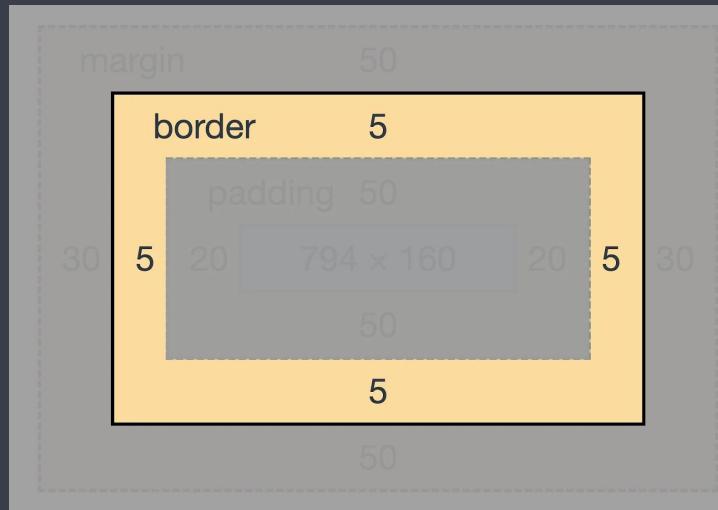
```
button {  
  padding: 5px 10px 15px 30px;  
}
```

**Quattro valori:** Specifica il padding per ciascuno dei quattro lati nell'ordine: in alto, a destra, in basso, a sinistra.

# Box Model: Border

La cornice vera e propria. Questa proprietà è una linea visibile che circonda il contenuto e il padding di un elemento HTML.

Il bordo è opzionale e può essere utilizzato per delimitare visivamente un elemento e definire i suoi confini all'interno del layout della pagina.



# Box Model: Border

Il border puo' essere utilizzato come proprietà a se' stante oppure come una serie di proprietà individuali per specificare diversi aspetti del bordo.

- **border-width**, lo spessore del bordo;
- **border-style**, lo stile del bordo (solido, tratteggiato, puntinato ecc);
- **border-color**, il colore del bordo;
- **border-radius**, la spigolosità degli angoli.

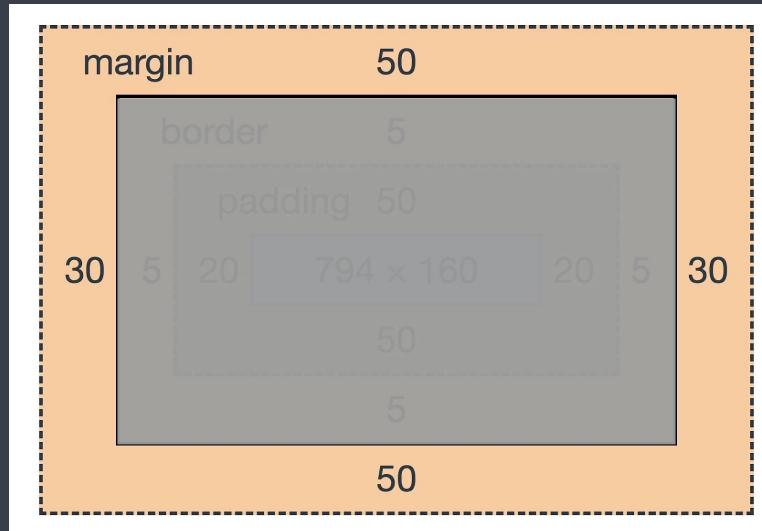
# Box Model: Margin

Lo spazio esterno tra cornice e ambiente circostante, che lo separa dall'ambiente circostante.

Il margine è quello spazio che separa l'elemento HTML dagli altri elementi nella pagina.

Il margine è opzionale e consente di creare spazio tra gli elementi, controllando la disposizione e il layout della pagina.

Il **margin** può essere **valorizzato** nello stesso identico modo  
del **padding**, quindi da **1 a 4 valori**.

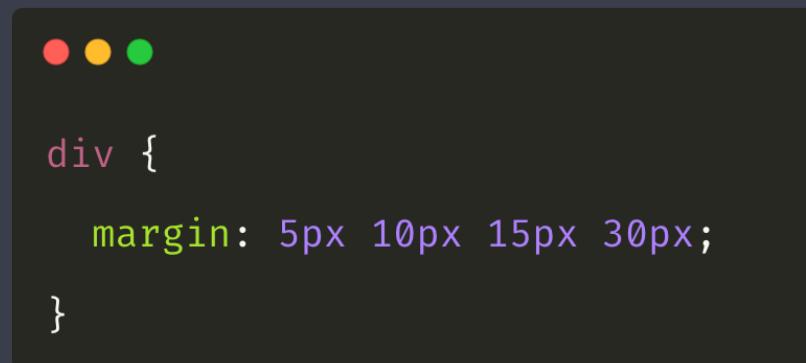


# Box Model: Margin

**Un singolo valore:** Specifica lo stesso valore di margin per tutti e quattro i lati dell'elemento (in alto, a destra, in basso, a sinistra).

**Due valori:** Il primo valore specifica il margin in alto e in basso, mentre il secondo valore specifica il margin a destra e a sinistra.

**Tre valori:** Il primo valore specifica il margin in alto, il secondo valore specifica il margin a destra e a sinistra, mentre il terzo valore specifica il margin in basso.



```
div {  
    margin: 5px 10px 15px 30px;  
}
```

**Quattro valori:** Specifica il margin per ciascuno dei quattro lati nell'ordine: in alto, a destra, in basso, a sinistra.

# Box Shadow

La proprietà box-shadow in CSS consente di **applicare ombre esterne** (o interne) agli elementi HTML, creando profondità e rilievo visivo.

Un po come visto con padding e margin, anche il box-shadow gestisce 4 parametri:

```
box-shadow: offset-x offset-y blur-radius color;
```

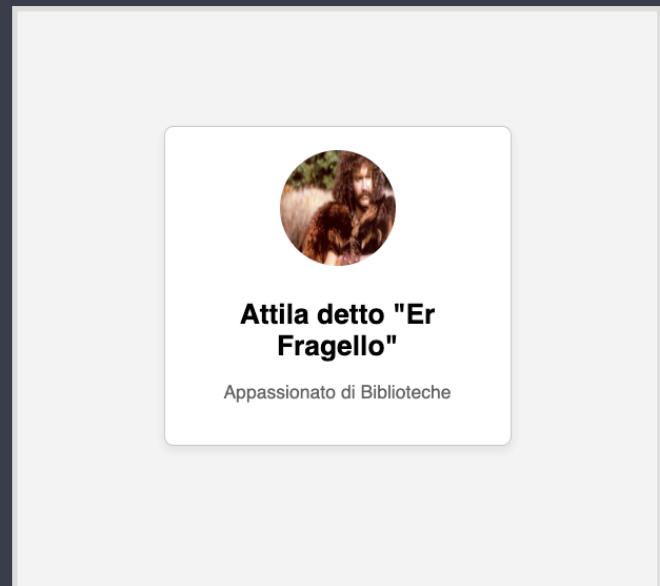
```
.ombra{  
  box-shadow: 4px 4px 10px rgba(0, 0, 0, 0.25);  
}
```

# Esercizio 1

Creare una scheda profilo utente con immagine, nome e descrizione, utilizzando padding, margin, border e box-shadow per dare struttura e profondità

## Requisiti

- Un contenitore card centrato nella pagina
- All'interno: immagine utente, nome e descrizione
- Aggiungi padding interno per distanziare il contenuto dai bordi
- Usa margin esterno per separare la card da altri elementi
- Applica un border sottile per evidenziare la cornice
- Aggiungi box-shadow per creare un effetto sollevato



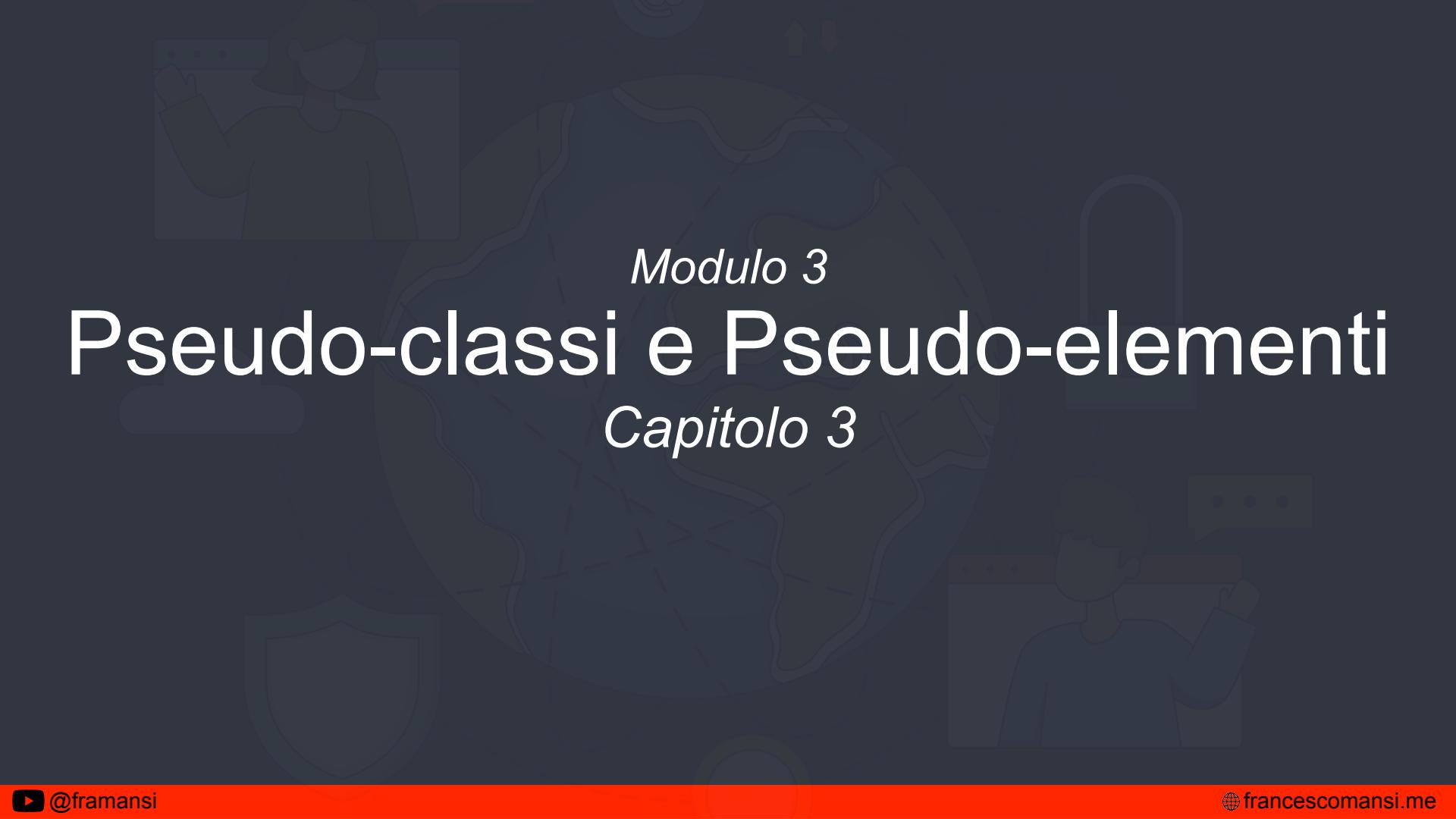
# Esercizio 2

Creare tre box affiancati per rappresentare servizi o caratteristiche, usando margin, padding, border e box-shadow, per comprendere il comportamento del Box Model in un layout orizzontale.

## Requisiti

- Un contenitore centrato con larghezza massima di 1000px
- Tre box affiancati con display flex o inline-block
- Ogni box deve avere una larghezza fissa
- Aggiungi padding interno per separare testo e bordo
- Inserisci margin tra i box per lo spazio esterno
- Applica border e box-shadow per definizione visiva





*Modulo 3*

# Pseudo-classi e Pseudo-elementi

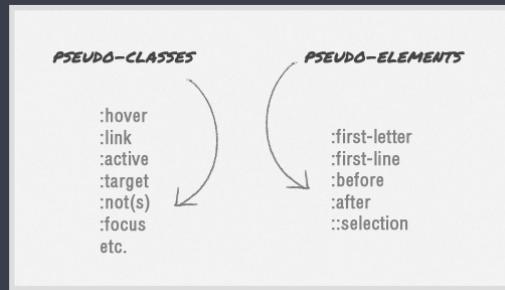
*Capitolo 3*

# Pseudo-cose

In CSS, pseudo-classi e pseudo-elementi sono strumenti che permettono di selezionare e stilizzare elementi in base a stati particolari o parti specifiche del contenuto, anche quando non sono direttamente rappresentati nel codice HTML.

Le **pseudo-classi** consentono di applicare stili a un elemento in base al suo stato o al contesto, come ad esempio quando il cursore passa sopra un link (:hover) o quando un campo del form è selezionato (:checked).

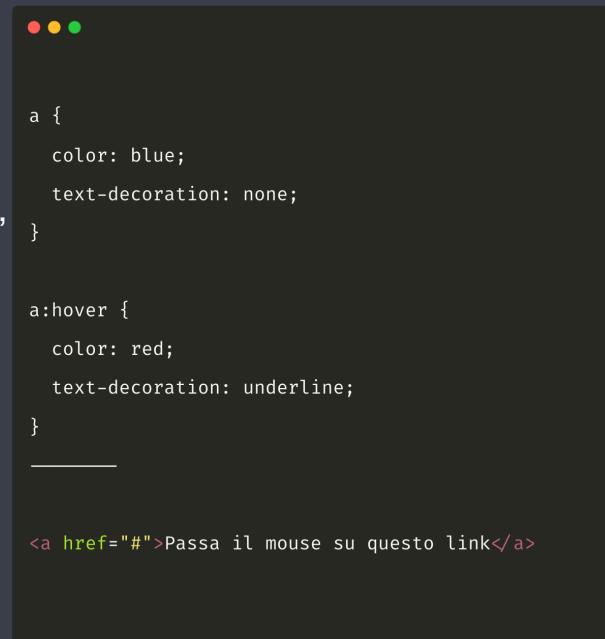
I **pseudo-elementi**, invece, permettono di creare e stilizzare parti virtuali di un elemento, come la prima lettera (::first-letter) o inserire contenuti generati prima o dopo (::before, ::after), senza modificare l'HTML.



# Pseudoclassi

Le pseudoclassi consentono di applicare stili specifici agli elementi HTML a seconda del loro stato attuale o del contesto specifico in cui si trovano.

Questo approccio offre una flessibilità notevole nella definizione del design delle pagine web, consentendoci di modificare l'aspetto degli elementi in risposta a determinate azioni dell'utente o al loro stato di interazione, come per esempio il passaggio del mouse, il click o il focus, o in base alla loro posizione nella struttura del documento, come il primo figlio di un elemento o l'unico figlio di un tipo specifico



```
a {  
    color: blue;  
    text-decoration: none;  
}  
  
a:hover {  
    color: red;  
    text-decoration: underline;  
}  
  
—  
  
<a href="#">Passa il mouse su questo link</a>
```

# Pseudelementi

Gli pseudelementi sono parti di un elemento HTML che possono essere selezionate e stilizzate tramite CSS, ma non sono presenti direttamente nel codice HTML.

Gli pseudelementi consentono di aggiungere contenuto o stili a parti specifiche di un elemento senza dover modificare direttamente il markup HTML. Vengono rappresentati da un doppio due punti (::) seguito dal nome dell'elemento.

```
● ● ●  
p ::before {  
    content: "★ ";  
    color: gold;  
}
```

<p>Questo è un paragrafo importante.</p>

*Prima del contenuto del paragrafo, viene aggiunta una stellina dorata*

*tramite ::before. Questo è un pseudo-elemento: non esiste nell'HTML, ma viene generato dal CSS.*

# Esercizi

## Traccia 1

Crea una pagina HTML che rappresenti un semplice portfolio personale con:

- una sezione con nome, descrizione e immagine profilo
- una lista di 3 progetti (titolo e breve descrizione)
- un link per il contatto email

Applica con il CSS:

- :hover per cambiare colore dei link al passaggio del mouse

# Esercizi

Traccia 1

**Mario Rossi**

Web Developer Junior



- **Portfolio HTML**  
Un sito vetrina creato solo con HTML e CSS.
- **To-Do List**  
App interattiva con JavaScript e Local Storage.
- **Responsive Landing Page**  
Pagina promozionale adattiva per mobile e desktop.

[Contattami](#)

# Esercizi

Traccia 2 — Articolo di blog con HTML semantico e pseudo-elementi. Crea una pagina HTML che simuli un articolo di blog con:

- un titolo principale (<h1>)
- un sottotitolo o data
- almeno 2 paragrafi di contenuto
- una citazione (<blockquote>)

Applica con il CSS:

- ::first-letter per ingrandire la prima lettera del primo paragrafo
- ::before per aggiungere un'icona prima della citazione
- imposta colori e padding con il box model

# Esercizi

Traccia 2

(Colori di esempio)

## Perché imparare HTML e CSS nel 2025

Scritto il 29 Maggio 2025

**H**TML e CSS sono le fondamenta del web. Anche con i framework moderni, conoscere bene questi linguaggi è ancora essenziale.

**I**n questo articolo vedremo perché investire tempo nell'apprendimento di questi strumenti porta vantaggi anche nel lungo periodo.

“Impara le regole come un professionista, così potrai infrangerle come un artista.”

# Variabili

Le variabili in CSS, anche conosciute come **custom properties**, sono meccanismi che consentono di memorizzare e riutilizzare valori specifici all'interno di un foglio di stile CSS.

Offrono una maggiore flessibilità e modularità nello sviluppo di stili CSS, consentendo di definire valori una sola volta e utilizzarli in più parti del foglio di stile.

Le variabili iniziano con un doppio trattino (-) seguito da un nome personalizzato e sono definite all'interno di un selettori CSS, spesso nel selettori :root per renderle globali all'intero documento.

Una volta definite, le variabili possono essere richiamate e utilizzate in qualsiasi parte del foglio di stile utilizzando la funzione var().

Le variabili CSS forniscono un modo potente per gestire e organizzare i valori nei fogli di stile, rendendo il codice più pulito, manutenibile e facilmente personalizzabile.

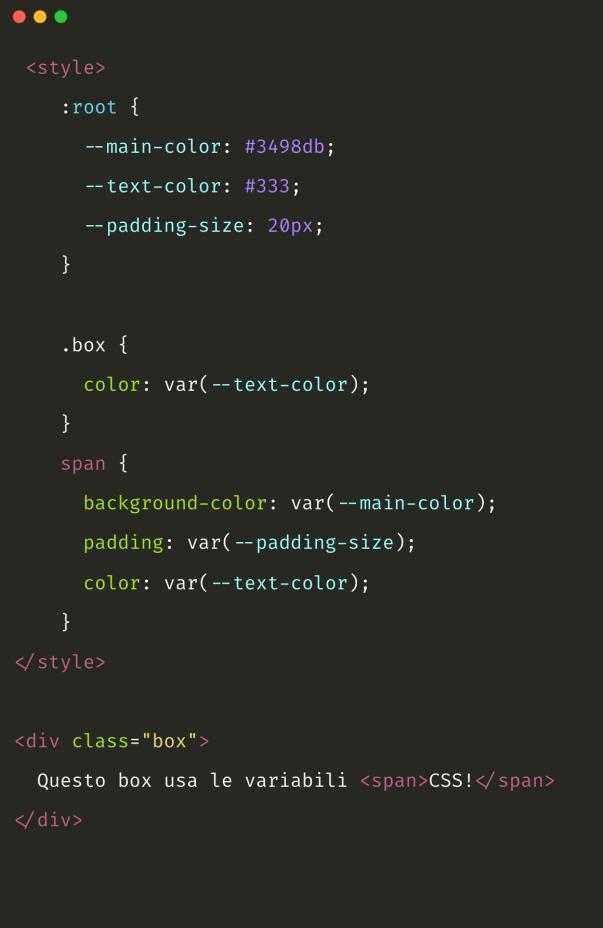


*Modulo 3*

# Variabili e Mediaqueries

*Capitolo 4*

# Variabili



```
<style>
:root {
    --main-color: #3498db;
    --text-color: #333;
    --padding-size: 20px;
}

.box {
    color: var(--text-color);
}
span {
    background-color: var(--main-color);
    padding: var(--padding-size);
    color: var(--text-color);
}
</style>

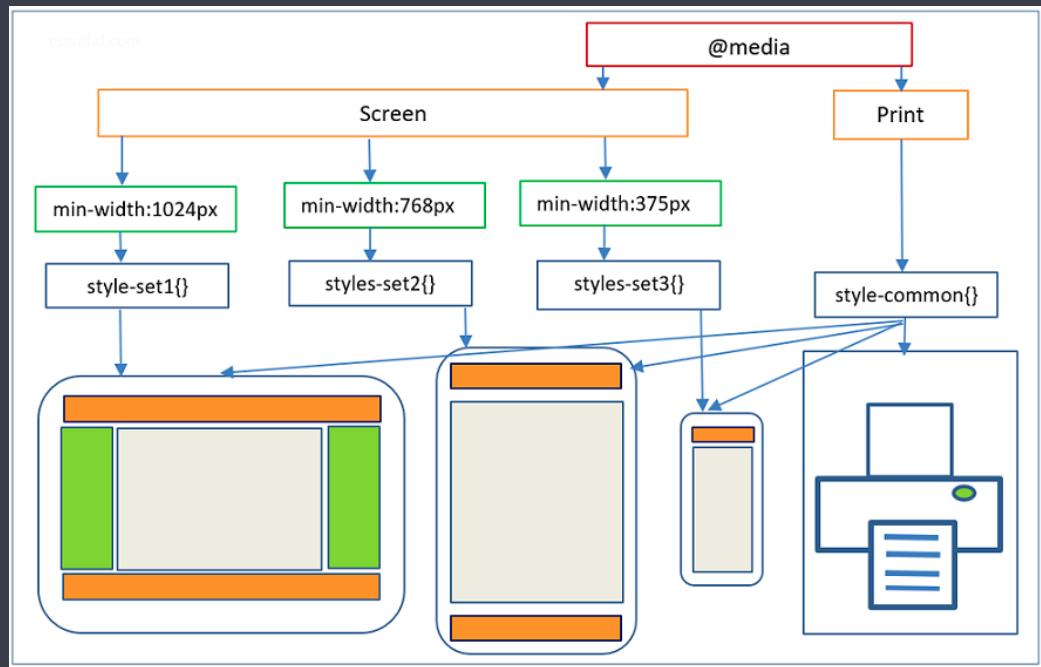
<div class="box">
    Questo box usa le variabili <span>CSS!</span>
</div>
```

# Mediaqueries

Le media queries sono un modo per applicare **stili diversi** in base alle caratteristiche del dispositivo o del viewport, come ad esempio la **larghezza dello schermo**, l'orientamento del dispositivo, la risoluzione dello schermo e molte altre proprietà.

Sono il miglior strumento che CSS ci mette a disposizione per rendere il nostro sito **responsive**.

Le **mediaqueries** vengono definite utilizzando la parola chiave `@media`, seguita da una condizione che specifica quando applicare gli stili CSS all'interno della media query.

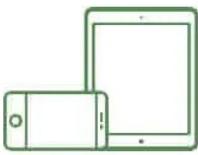


# Mediaqueries



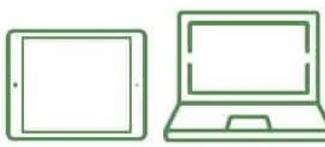
0-480

Smaller  
smartphones



481-768

Tablets & larger  
smartphones



769-1279

Laptops, larger tablets  
in landscape, and small  
desktops



1280+

Larger desktops  
and monitors



```
@media (min-width: 576px) {
```

```
  .box {
```

```
    color:red;
```

```
}
```

```
}
```

```
@media (min-width: 768px) {
```

```
  .box {
```

```
    color:blue;
```

```
}
```

```
}
```

# Esercizi

## Esercizio 1 — Media Query Responsive Box

Obiettivo:

Crea un box centrale con larghezza e colore diversi in base alla larghezza dello schermo. Se lo schermo è piccolo (max 600px), il box deve diventare più piccolo e cambiare colore.

Ridimensiona la finestra per cambiare stile

# Esercizi

## Esercizio 2 — Uso di Variabili CSS

Obiettivo:

Utilizza le custom properties per gestire colori e spaziatura in modo centralizzato. Modifica i valori in :root per cambiare facilmente l'intero stile.

Questo box usa variabili CSS!



# *Modulo 3*

# **Layout**

## *Capitolo 5*

# Il layout in CSS

Il posizionamento degli elementi HTML è fondamentale per creare layout e design dinamici e responsivi nelle pagine web.

CSS offre diverse tecniche e proprietà per controllare il posizionamento degli elementi, consentendo agli sviluppatori di creare layout complessi e adattabili.

Adesso andremo ad esplorare due dei principali metodi per posizionare gli elementi all'interno della nostra pagina tramite il CSS.

# Proprietà: display

La proprietà CSS `display` definisce come un elemento viene visualizzato nella pagina web e come interagisce con gli altri elementi circostanti. È una delle proprietà fondamentali per il controllo del layout e determina il comportamento di rendering di ogni elemento HTML.

Questa proprietà influenza direttamente il flusso del documento e le possibilità di styling, permettendo di trasformare elementi inline in block e viceversa, o di utilizzare modelli di layout più avanzati come flexbox e grid.

```
/* Sintassi base */

elemento {

    display: valore;

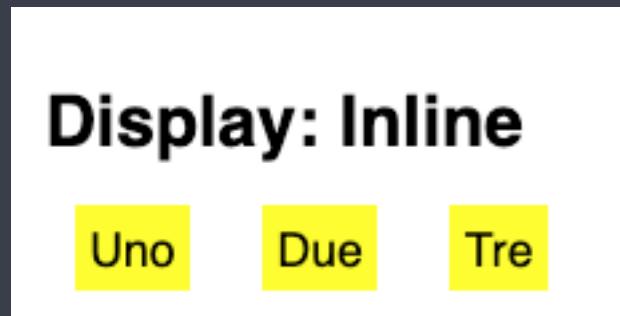
}
```

# Display Inline

Gli elementi con `display:inline` si comportano come il testo normale, posizionandosi uno accanto all'altro sulla stessa riga fino a riempire lo spazio disponibile.

Non rispettano le proprietà di larghezza e altezza, e i margini verticali non hanno effetto.

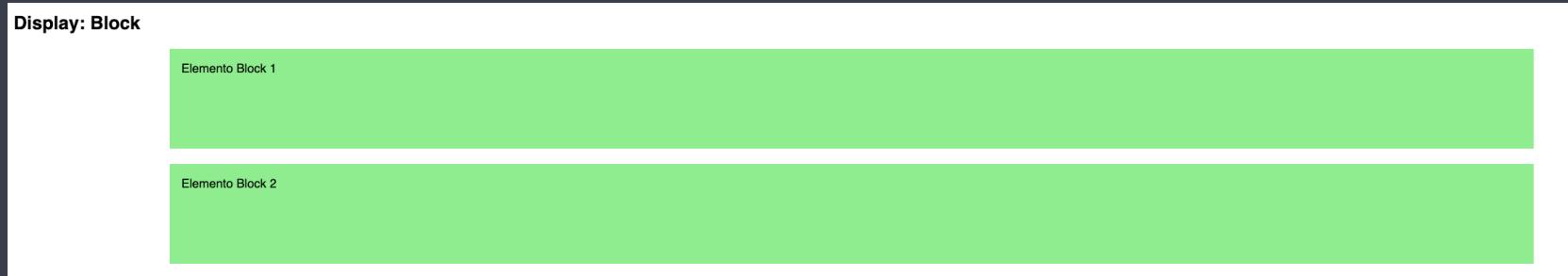
Esempi tipici di elementi inline sono `<span>`, `<a>`, `<strong>`, `<em>`. Questi elementi fluiscono naturalmente nel testo e vanno a capo automaticamente quando raggiungono il bordo del contenitore.



# Display Block

Gli elementi block occupano tutta la larghezza disponibile del contenitore genitore e iniziano sempre su una nuova riga. Accettano tutte le proprietà di dimensione e spaziatura, e creano naturalmente una struttura verticale.

Elementi come `<div>`, `<p>`, `<h1>`-`<h6>`, `<section>` sono block per default. Questo comportamento li rende ideali per creare la struttura principale di una pagina web.

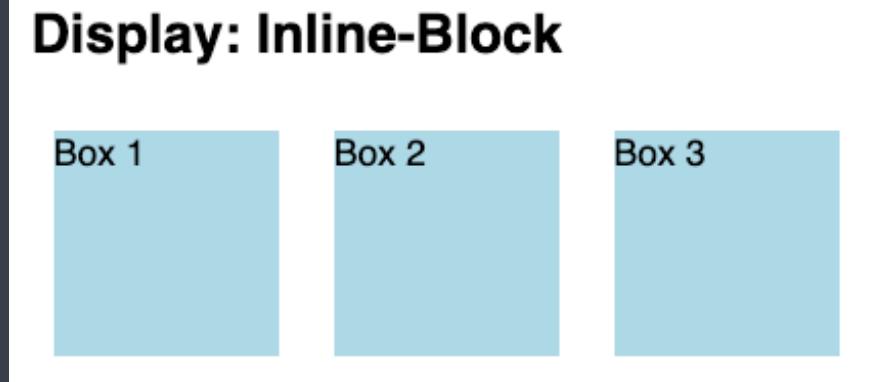


# Display Inline-Block

L'elemento `inline-block` combina le caratteristiche degli elementi inline e block.

Si posiziona sulla stessa riga come gli elementi inline, ma accetta proprietà di dimensione come larghezza, altezza, margini e padding verticali.

Questa proprietà è particolarmente utile per creare layout orizzontali mantenendo il controllo completo sulle dimensioni degli elementi, come per menu di navigazione o gallerie di immagini.



# Display Flex

La proprietà `display: flex` trasforma un elemento in un contenitore flessibile, attivando il modello di layout Flexbox. Questo permette un controllo avanzato sull'allineamento, distribuzione e ridimensionamento degli elementi figli.

Flexbox risolve molti problemi comuni del CSS tradizionale, come la centratura verticale, la distribuzione equa dello spazio e la creazione di layout responsivi. È particolarmente potente per layout unidimensionali (riga o colonna).



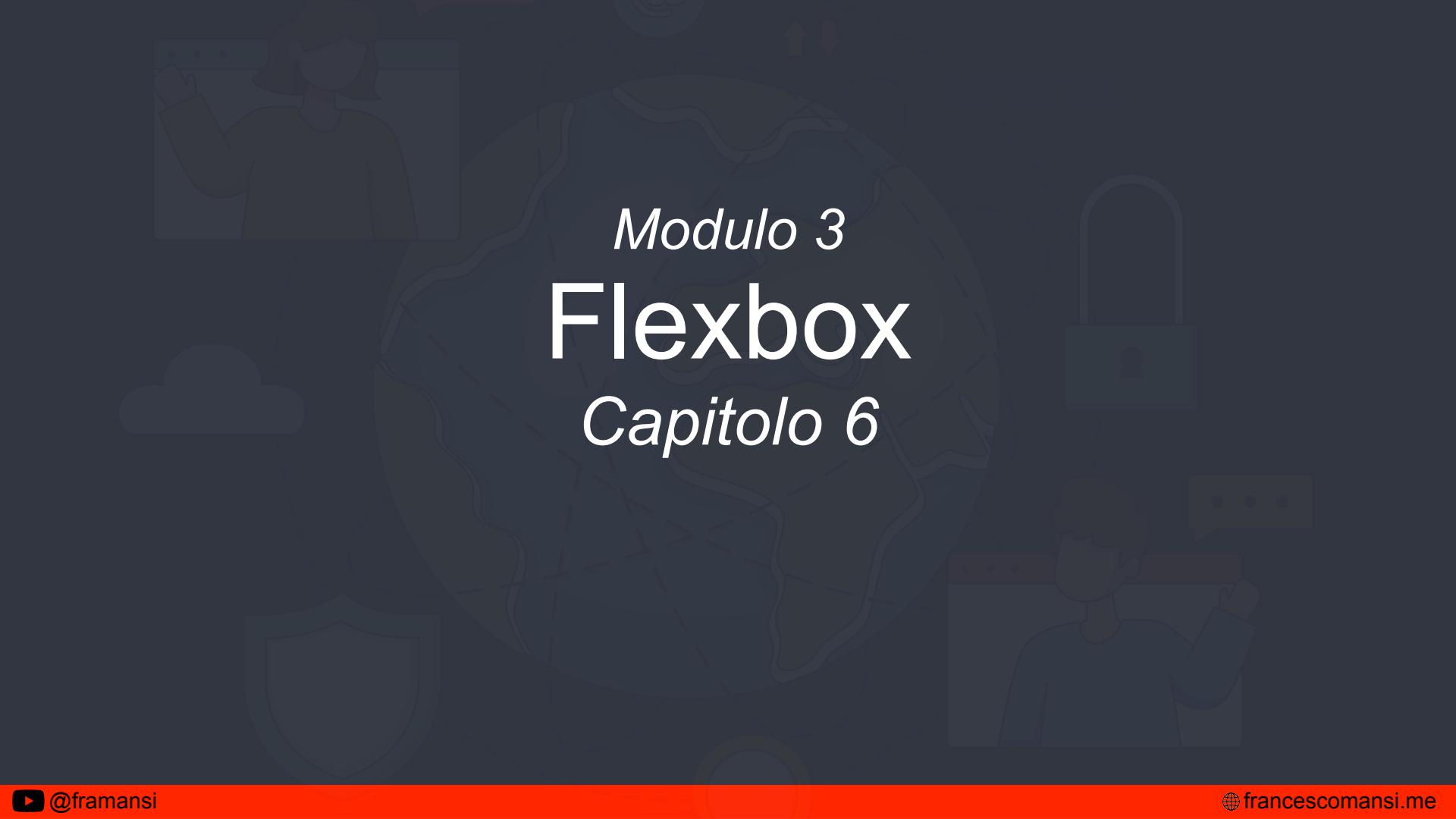
# Display None

La proprietà `display: none` rimuove completamente l'elemento dal flusso del documento, rendendolo invisibile e facendo sì che non occupi alcuno spazio. È diverso da `visibility: hidden` che mantiene lo spazio occupato dall'elemento.

Questa proprietà è fondamentale per creare interfacce dinamiche, menu a scomparsa, modali e elementi che appaiono/scompaiono in base alle interazioni dell'utente o alle media queries.

# Esercizio

Riprodurre una semplice pagina con 5 sezioni, ognuna delle quali dimostra un valore diverso della proprietà display.



# *Modulo 3*

# Flexbox

## *Capitolo 6*

# Cos'è il Flexbox

Flexbox (Flexible Box Layout) è un modello di layout CSS progettato per distribuire spazio e allineare elementi in un contenitore, anche quando le loro dimensioni sono sconosciute o dinamiche.

Funziona su due assi: principale (main axis) e trasversale (cross axis).

Il sistema flexbox è composto da un contenitore padre (flex container) e i suoi figli diretti (flex items). Il contenitore controlla il layout generale, mentre ogni figlio può avere proprietà individuali per il proprio comportamento all'interno del layout flessibile.

# Flex Direction

Questa è la proprietà che ci permette di **definire l'asse principale** del nostro layout. Ricorda: l'asse principale può essere **orizzontale** oppure **verticale**.

Se vogliamo che l'asse principale sia **orizzontale**, useremo il valore row.

Se invece lo vogliamo **verticale**, useremo column.

Inoltre, abbiamo parlato di **inizio e fine dell'asse principale** (main start e main end).

Se vogliamo invertire la direzione, basta **aggiungere il suffisso -reverse**:

- row-reverse parte da destra verso sinistra (asse orizzontale invertito)
- column-reverse parte dal basso verso l'alto (asse verticale invertito)

In questo modo possiamo controllare **il flusso e l'ordine visivo** degli elementi figli all'interno di un contenitore Flexbox.

<https://www.samanthaming.com/flexbox30/9-flex-direction/>

# Flex Wrap

La proprietà `flex-wrap` determina se gli elementi flex possono andare a capo quando non c'è abbastanza spazio nel contenitore. Per default è impostata su `nowrap`, ma può essere cambiata in `wrap` o `wrap-reverse`.

Quando è attivato il `wrap`, gli elementi che non entrano nella prima riga creano automaticamente nuove righe, mantenendo le proprietà di allineamento. Questo è essenziale per creare layout responsivi che si adattano a diverse dimensioni dello schermo.

<https://www.samanthaming.com/flexbox30/10-flex-wrap/>

# Justify Content

Questa è la proprietà che gestisce l'allineamento lungo l'asse principale. È probabilmente **la proprietà più usata** con i contenitori Flexbox.

Scegli semplicemente l'allineamento che preferisci e... **bam!** Flexbox si occupa di tutto il resto, in modo **automatico e responsive**.

Man mano che la finestra si allarga o si restringe, Flexbox **calcola tutto dietro le quinte** per mantenere sempre l'allineamento desiderato.

<https://www.samanthaming.com/flexbox30/12-justify-content-row/>

# Align Items

La proprietà justify-content serve a controllare la disposizione degli elementi lungo l'asse principale. Per gestire invece l'allineamento sull'asse trasversale, entra in gioco align-items.

L'asse trasversale (o secondario) è sempre **perpendicolare all'asse principale**: se l'asse principale è orizzontale (come accade con flex-direction: row), allora l'asse trasversale sarà verticale.

Questi concetti si basano sui **principi fondamentali di Flexbox**, che permettono di gestire layout complessi in modo ordinato e prevedibile. Tutte le nozioni di base apprese trovano ora applicazione pratica nella definizione dell'allineamento degli elementi in ogni direzione.

<https://www.samanthaming.com/flexbox30/15-align-items-row/>

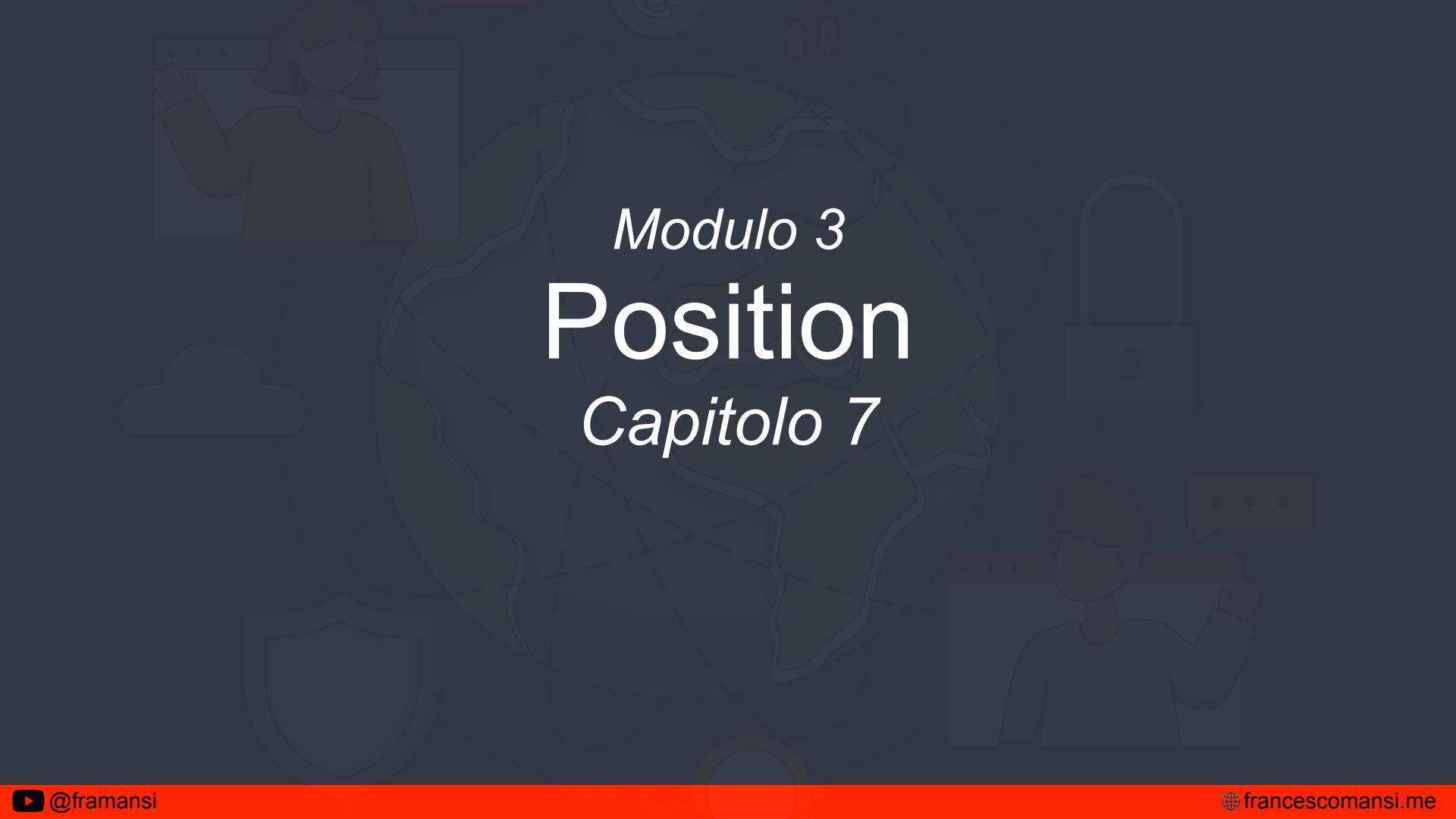
# Esercizio

Soluzione: <https://codepen.io/abbeyfitzgerald/pen/zzqqYo>



# Esercizi EXTRA

- 1) Finire tutti i 24 livelli di <https://flexboxfroggy.com/#it>
- 2) Provare anche con i 24 livelli di <https://codingfantasy.com/games/flexboxadventure>



# *Modulo 3*

# Position

## *Capitolo 7*

# Position

Le proprietà di posizionamento sono utilizzate per controllare la **posizione** di un elemento all'interno di un layout di pagina web.

Queste proprietà consentono di spostare gli elementi in modo più “libero”rispetto alla loro posizione normale nel flusso del documento HTML. Appena creato, ogni elemento acquisisce di default la position:static , andandosi a posizionare nel primo spazio disponibile in alto a sinistra.

I diversi valori della proprietà sono:

- Static
- Relative
- Absolute
- Fixed
- Sticky
- z-index

# Playground Position

<https://appbrewery.github.io/css-positioning/>

# Esercizio

Riprodurre una semplice pagina con 6 sezioni, ognuna delle quali dimostra un valore diverso della proprietà position.



*Modulo 2*

# Progetto Finale CSS

*Capitolo 8*

# Traccia

Creare un sito portfolio responsive per te stesso, usando HTML semantico e tutte le principali proprietà CSS viste.

NON usate librerie, ma pure classi css fatte a mano.

Una volta fatto tutto, caricheremo il sito su GithubPages: <https://pages.github.com/>



# *Modulo 3*

# Bootstrap

## *Capitolo 9*

# Introduzione a Bootstrap 5

Bootstrap è il framework CSS più utilizzato al mondo per lo sviluppo web responsive.

Creato originariamente da Twitter nel 2011, è ora alla versione 5 con importanti miglioramenti.

Si tratta di una libreria di componenti pre-costruiti che permette di creare interfacce web moderne senza scrivere CSS da zero.

Bootstrap 5 adotta un approccio mobile-first, garantendo che le applicazioni funzionino perfettamente su tutti i dispositivi.

Creato da Mark Otto e Jacob Thornton presso Twitter, Bootstrap fornisce una collezione di strumenti CSS e JavaScript per la progettazione di interfacce web.



# Framework vs Libreria

Framework: è una struttura di supporto predefinita che fornisce un insieme di strumenti, librerie e best practices per facilitare e standardizzare lo sviluppo di software. I framework possono essere utilizzati in vari contesti dello sviluppo software, come lo sviluppo web, mobile, di giochi, e altro. Essi aiutano a ridurre il tempo di sviluppo, migliorare la qualità del codice e assicurare la coerenza all'interno dei progetti.

Libreria: Fornisce funzioni riutilizzabili che chiami tu nel tuo codice, lasciandoti il controllo del flusso.

In sintesi, un framework impone una struttura e gestisce il flusso, mentre una libreria ti offre strumenti da usare comee quando vuoi.

# Responsive e Mobile First

Il **responsive design** si riferisce alla capacità di un sito web di adattarsi automaticamente a diverse dimensioni di schermo e dispositivi. Questo significa che il layout, le immagini, il testo e altri elementi del sito si ridimensionano e si riorganizzano in modo fluido per offrire una buona esperienza utente su qualsiasi dispositivo, che sia un desktop, un tablet o uno smartphone.

Il **mobile-first** è un approccio di progettazione e sviluppo web in cui si inizia progettando e sviluppando l'**interfaccia utente** per i dispositivi mobili prima di passare a versioni per schermi più grandi come tablet e desktop. Questo approccio riflette la crescente prevalenza dell'uso di dispositivi mobili per navigare in internet.

# Perché Scegliere Bootstrap?

Bootstrap accelera significativamente lo sviluppo web fornendo componenti pronti all'uso e stili consistenti. La sua compatibilità cross-browser elimina i problemi di inconsistenza tra diversi browser.

La documentazione è eccellente e dettagliata, con esempi pratici per ogni componente. La vasta community garantisce supporto continuo e risorse aggiuntive.

# Come Iniziare

Il modo più veloce per iniziare è utilizzare il CDN, collegando semplicemente i file CSS e JavaScript via link esterni.

Per progetti più complessi, puoi scaricare i file localmente o installarli via npm.

Bootstrap fornisce template starter pronti all'uso che includono la struttura HTML base e tutti i collegamenti necessari.

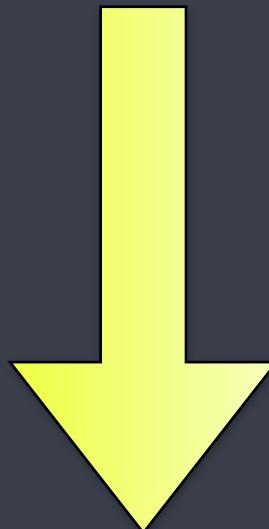
- CDN
- DOWNLOAD
- NPM

LINK: <https://getbootstrap.com/docs/5.3/getting-started/introduction/>

# Strutturare Risorse

Il browser interpreta il linguaggio di markup dall'alto verso il basso, quindi verranno prima letti gli stili e poi successivamente caricati gli script.

TOP



DOWN

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Pagina HTML</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.6/dist/css/bootstrap.min.css" rel="stylesheet">
  </head>
  <body>
    <h1>Weila!</h1>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.6/dist/js/bootstrap.bundle.min.js"></script>
  </body>
</html>
```

# Le classi di Bootstrap

Abbiamo detto che **Bootstrap** è una libreria che ci fornisce delle **classi pre-impostate** che ci permettono di fare una serie di cose soprattutto velocizzare il nostro lavoro.

Ad esempio avremo dei colori di default, delle dimensioni di default, stili di default.

Alcune di queste classi **pre-impostate** però, non devono farci distogliere l'obiettivo dall'avere applicativi sempre unici, per cui ci toccherà usare anche il nostro **CSS custom**.

Altre volte ci capiterà, dato che le **classi** di **Bootstrap** hanno intrinsecamente l'**attributo !important** dovremmo sovrascriverle con il nostro **CSS custom**, come dicevamo su.

# Ma che cos'è una classe?

L'abbiamo già visto, delle regole CSS racchiuse in un selettore.

# Bootstrap: Layout

- Breakpoints: <https://getbootstrap.com/docs/5.3/layout/breakpoints/>
- Container: <https://getbootstrap.com/docs/5.3/layout/containers/>
- Grid System: <https://getbootstrap.com/docs/5.3/layout/grid/>
- Columns: <https://getbootstrap.com/docs/5.3/layout/columns/>

# Bootstrap: Content

- Typography: <https://getbootstrap.com/docs/5.3/content/typography/>
- Image: <https://getbootstrap.com/docs/5.3/content/images/>
- Table: <https://getbootstrap.com/docs/5.3/content/tables/>
- Figures: <https://getbootstrap.com/docs/5.3/content/figures/>

# Bootstrap: Form

Introduzione ai form e altri link: <https://getbootstrap.com/docs/5.3/forms/overview/>

# Bootstrap: Components

## Components

- Accordion
- Alerts
- Badge
- Breadcrumb
- Buttons
- Button group
- Card
- Carousel
- Close button
- Collapse
- Dropdowns

- List group

- Modal

- Navbar

- Navs & tabs

- Offcanvas

- Pagination

- Placeholders

- Popovers

- Progress

- Scrollspy

- Spinners

- Toasts

- Tooltips

# Esercizio 1

**Obiettivo:** Creare una landing page responsive con navbar, jumbotron e sezione di 3 servizi.

Richieste:

- Usa il **container** Bootstrap
- Aggiungi una **navbar** responsive con brand e 3 link
- Inserisci un **jumbotron** con titolo e paragrafo
- Crea una sezione con **row + col-md-4** per 3 box (icone + testo)
- Aggiungi **padding e margin** con le utility class

# Esercizio 2

**Obiettivo:** Realizzare una scheda prodotto responsive con immagine, titolo, descrizione e bottone.

Richieste:

- Usa una **card** Bootstrap con immagine sopra
- Inserisci **card-body** con titolo, testo e bottone “Aggiungi al carrello”
- Allinea la card al centro con **d-flex justify-content-center**
- Applica classi di colore e rounded

# Esercizio 3

**Obiettivo:** Costruire un form di contatto con layout ordinato e validazione base.

Richieste:

- Usa **form-control** per input (nome, email, messaggio)
- Dividi il form in **2 colonne** su schermi grandi (col-md-6)
- Aggiungi un **bottone** con stile Bootstrap
- Inserisci **placeholder** e classi mb-3 per spaziatura
- Rendi il form **responsive**



*Modulo 3*

# Tailwind CSS

*Capitolo 10*

# Introduzione a Tailwind

Tailwind CSS è un framework CSS "utility-first" che fornisce classi di basso livello per costruire design personalizzati direttamente nel markup HTML.

A differenza dei framework tradizionali, non offre componenti pre-costruiti ma utilità atomiche.

Ogni classe corrisponde a una singola proprietà CSS, permettendo un controllo granulare del design. Questo approccio elimina la necessità di scrivere CSS personalizzato per la maggior parte dei casi d'uso.

Il framework include un design system completo con colori, spaziature, tipografia e breakpoint responsive integrati, garantendo consistenza visiva in tutto il progetto.



# Approccio Utility-First vs Component-Based

L'approccio utility-first significa costruire interfacce combinando piccole classi riutilizzabili invece di scrivere CSS personalizzato.

Ad esempio, invece di creare una classe `.header`, si usano `.text-center` `.text-blue-500` `.font-bold`.

Questa metodologia accelera lo sviluppo perché elimina la necessità di inventare nomi per le classi e di scrivere CSS da zero. Ogni utility ha uno scopo specifico e può essere riutilizzata ovunque.

Il risultato è un CSS più mantenibile e un workflow di sviluppo più veloce, specialmente per team che lavorano su progetti di grandi dimensioni.

# Perché Scegliere Tailwind CSS

Tailwind produce CSS estremamente ottimizzato in produzione grazie al sistema di purging automatico che rimuove tutte le classi non utilizzate. Questo risulta in file CSS molto più piccoli rispetto ai framework tradizionali.

Il design system integrato garantisce consistenza visiva automatica: colori, spaziature e tipografia seguono scale predefinite ma completamente personalizzabili.

Non dovrai più preoccuparti di scegliere il giusto shade di blu o la spaziatura corretta.

La curva di apprendimento è rapida perché le classi sono intuitive e corrispondono direttamente alle proprietà CSS che già conosci

# Come Iniziare

Il modo più veloce per iniziare è utilizzare il CDN, collegando semplicemente i file CSS e JavaScript via link esterni.

Per progetti più complessi, puoi scaricare i file localmente o installarli via npm.

Tailwind fornisce template starter pronti all'uso che includono la struttura HTML base e tutti i collegamenti necessari.

- CDN
- DOWNLOAD
- NPM

LINK: <https://tailwindcss.com/docs/installation/play-cdn>

# Playground

Il modo più immediato per provare Tailwind è con il suo Playground:

LINK: <https://play.tailwindcss.com>

# Prime Classi Pratiche

Questo esempio applica: sfondo blu, testo bianco, padding di 1rem, angoli arrotondati e ombra media. Ogni classe ha un significato specifico e immediatamente riconoscibile.

Le classi Tailwind sono progettate per essere leggibili: `bg-blue-500` significa "background blu, shade 500", `p-4` significa "padding 1rem", `rounded-lg` significa "border-radius large".



```
<div class="bg-blue-500 text-white p-4 rounded-lg shadow-md">  
  Il mio primo elemento Tailwind!  
</div>
```

# Sistema Colori Avanzato

Tailwind include una palette di colori con scale da 50 (molto chiaro) a 900 (molto scuro) per ogni tonalità. Ad esempio: `text-red-100` (rosso chiaro), `bg-red-500` (rosso medio), `border-red-800` (rosso scuro).

I colori disponibili includono: gray, red, yellow, green, blue, indigo, purple, pink e molti altri. Ogni colore ha 10 varianti numeriche che garantiscono contrasti appropriati e armonia visiva.

Le classi colore possono essere applicate a testo (`.text-*`), sfondi (`.bg-*`), bordi (`.border-*`) e altri elementi come `fill` e `stroke` per SVG.

**Link di riferimento:** <https://tailwindcss.com/docs/customizing-colors>

# Tipografia

Le dimensioni del testo vanno da `.text-xs` (extra small) a `.text-9xl` (extra extra large), con scale intermedie intuitive. Il peso del font è controllato da `.font-thin` a `.font-black`.

L'interlinea (line-height) usa `.leading-tight`, `.leading-normal`, `.leading-relaxed`, mentre la spaziatura delle lettere utilizza `.tracking-tighter`, `.tracking-normal`, `.tracking-wider`.

Sono disponibili anche utilità per `text-decoration` (`.underline`, `.line-through`), `text-transform` (`.uppercase`, `.lowercase`) e `text-align` (`.text-left`, `.text-center`, `.text-right`).

**Link di riferimento:** <https://tailwindcss.com/docs/font-size>

# Spaziatura

Il sistema di spaziatura usa una scala numerica dove ogni unità corrisponde a 0.25rem (4px). Quindi .p-1 = 4px, .p-2 = 8px, .p-4 = 16px, fino a .p-96 = 384px.

I prefissi determinano la direzione: .p-\* (padding completo), .px-\* (orizzontale), .py-\* (verticale), .pt-\* (top), .pr-\* (right), ecc. Lo stesso vale per i margin con .m-\*.

Valori speciali includono .p-px (1px), .p-0.5 (2px), e .p-auto per margin automatico. Questa scala garantisce spaziature armoniose e coerenti in tutto il design.

**Link di riferimento:** <https://tailwindcss.com/docs/padding>

Controllare px to rem: <https://tailwind-match.netlify.app/>

# Flexbox

Attiva flexbox con `.flex` e controlla la direzione con `.flex-row` (default) o `.flex-col`. L'allineamento principale usa `.justify-start`, `.justify-center`, `.justify-between`, `.justify-around`.

L'allineamento trasversale utilizza `.items-start`, `.items-center`, `.items-end`, `.items-stretch`. Per controllare il wrap usa `.flex-wrap` o `.flex-no-wrap`.

Le proprietà flex degli elementi figli sono gestite con `.flex-1` (grow), `.flex-auto`, `.flex-none` e `.flex-shrink` per controlli più specifici.

Link di riferimento: <https://tailwindcss.com/docs/flex>

# Grid System

Attiva CSS Grid con `.grid` e definisci le colonne con `.grid-cols-1` fino a `.grid-cols-12`. Per layout personalizzati usa `.grid-cols-[200px_1fr_100px]` con valori arbitrari.

Gli elementi figli occupano spazio con `.col-span-2`, `.col-span-3`, ecc. Per le righe usa `.grid-rows-*` e `.row-span-*`. La classe `.col-start-2` e `.col-end-4` permettono posizionamento preciso.

Le gap (spaziature tra elementi) usano `.gap-4`, `.gap-x-2` (orizzontale), `.gap-y-8` (verticale) con la stessa scala numerica delle spaziature.

**Link di riferimento:** <https://tailwindcss.com/docs/grid-template-columns>

# Mobile-First Responsive

Tailwind usa breakpoint mobile-first: `sm:` (640px+), `md:` (768px+), `lg:` (1024px+), `xl:` (1280px+), `2xl:` (1536px+). Ogni utility può essere resa responsive aggiungendo il prefisso appropriato.

Esempio: `.text-base md:text-lg lg:text-xl` crea testo che cresce con la dimensione dello schermo. `.hidden md:block` nasconde su mobile ma mostra su tablet e desktop.

I layout possono cambiare completamente: `.flex-col md:flex-row` impila verticalmente su mobile ma orizzontalmente su schermi più grandi.

Link di riferimento: <https://tailwindcss.com/docs/responsive-design>

# Stati Interattivi

Ogni utility può essere applicata a stati specifici usando pseudo-classi. `.hover:bg-blue-600` cambia il colore al passaggio del mouse, `.focus:ring-2` aggiunge un anello di focus.

Stati disponibili includono: `hover:`, `focus:`, `active:`, `disabled:`, `first:`, `last:`, `odd:`, `even:` e molti altri. Possono essere combinati con `responsive:` `md:hover:scale-105`.

Gli stati possono essere annidati: `.group-hover:opacity-100` si attiva quando si fa hover sul genitore con classe `.group`, perfetto per effetti complessi.

**Link di riferimento:** <https://tailwindcss.com/docs-hover-focus-and-other-states>

# Width, Height e Sizing

Le dimensioni usano scale diverse: valori fissi (`.w-64 = 256px`), frazioni (`.w-1/2, .w-1/3, .w-2/5`), percentuali (`.w-full = 100%`), e viewport (`.w-screen = 100vw`).

L'altezza segue le stesse regole con `.h-*`. Valori speciali includono `.w-fit` (fit-content), `.w-min` (min-content), `.w-max` (max-content).

Min e max dimensioni usano `.min-w-*`, `.max-w-*`, `.min-h-*`, `.max-h-*`. Per esempio `.max-w-prose` ottimizza la larghezza per la leggibilità del testo.

Link di riferimento: <https://tailwindcss.com/docs/width>

# Border, Border-radius e Shadows

I bordi iniziano con `.border` (1px) e crescono con `.border-2`, `.border-4`, `.border-8`. I lati specifici usano `.border-t-*`, `.border-r-*`, ecc. I colori seguono la palette standard: `.border-gray-300`.

Gli angoli arrotondati vanno da `.rounded-sm` a `.rounded-3xl`, con `.rounded-full` per cerchi perfetti. Lati specifici: `.rounded-t-lg`, `.rounded-br-xl`.

Le ombre creano profondità: `.shadow-sm`, `.shadow`, `.shadow-lg`, `.shadow-xl`, `.shadow-2xl`. Colori personalizzati con `.shadow-blue-500/50` per trasparenza.

**Link di riferimento:** <https://tailwindcss.com/docs/border-width>

# Esercizio 1

**Obiettivo:** Creare una landing page semplice e responsive con intestazione e 3 sezioni orizzontali.

- Usa container mx-auto per centrare il contenuto
- Crea un'intestazione con flex justify-between items-center (logo a sinistra, link a destra)
- Aggiungi un blocco principale con titolo (text-3xl font-bold) e paragrafo (text-gray-600)
- Sotto, crea una grid grid-cols-1 md:grid-cols-3 gap-4 per 3 box servizi (colore di sfondo, padding, bordi arrotondati)

# Esercizio 2

## Esercizio 2 – Card Prodotto

**Obiettivo:** Costruire una scheda prodotto centrata con immagine, titolo, descrizione e bottone.

- Usa un div con max-w-sm mx-auto bg-white shadow-md rounded-lg
- Inserisci un'immagine in alto con rounded-t-lg
- Aggiungi un blocco sotto con p-4: titolo (text-xl font-semibold), descrizione (text-gray-500)
- Inserisci un bottone con bg-blue-500 hover:bg-blue-600 text-white font-bold py-2 px-4 rounded mt-4

# Esercizio 3

**Obiettivo:** Realizzare un form di contatto con campi ordinati e ben spaziati.

- Crea un form centrato con max-w-2xl mx-auto p-6 bg-gray-100 rounded-lg
- Aggiungi 3 input: nome, email, messaggio (usa block w-full p-2 border rounded mb-4)
- Rendi il layout a **2 colonne su schermi grandi** con grid grid-cols-1 md:grid-cols-2 gap-4
- Aggiungi un bottone finale con bg-green-500 text-white py-2 px-6 rounded hover:bg-green-600