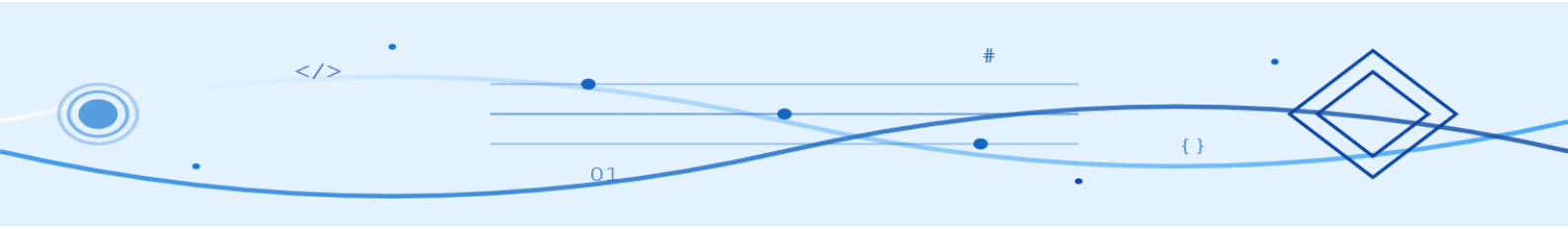


INTRODUZIONE

A HTML5 E CSS3

Docente:
Fulvia Grignaschi
fulvia.grignaschi@gmail.com
cell:340 640 51 52

Sommario

HTML

- HTML = HyperText Markup Language
- Markup Language ⇒ Usa “tags”
- In HTML i tag sono definiti con parentesi angolari (< >)
- Esempio ⇒ ** ** Content of the paragraph ** **

Tags

- La maggior parte dei tag ha sia quello di chiusura che di apertura
  Esempio ⇒ This is the content ** **

- Alcuni sono “self-closing”
  Esempio ⇒ **

- I tag possono essere annidati: chiamiamo quello esterno genitore, quelli interni figli
  Esempio ⇒
   ⇐ genitore
   … ⇐ figlio

- Ogni tag **non “self-closing”** può avere un numero illimitato di figli

Ricordarti sempre di chiudere i tag, altrimenti potrebbero verificarsi comportamenti imprevisti

Attributi

- Coppie di informazioni “nome = valore” contenute nello start tag (tag di apertura) con una sintassi di questo tipo:

<tag attrib1= “val1” attrib2= “val2” >

- Alcuni attributi sono validi per tutti i tag (id, class, style)
- Altri sono specifici di alcuni tag
- Esempio:

Come il browser legge i tag e gli attributi

Essi non vengono mai visualizzati dal browser, piuttosto viene visualizzato il loro “effetto” sul testo della pagina Web.
Questo avviene grazie all'operazione di rendering effettuata da un apposito “motore” incorporato in ogni browser.
Se qualche tag è inesistente o semplicemente scritto male, contiene errori di sintassi o per qualsiasi motivo non viene riconosciuto dal browser, viene da questo semplicemente ignorato senza che sia restituita allo sviluppatore o all’utente finale nessuna segnalazione di errore.
Molte volte i programmi di navigazione moderni riescono a interpretare correttamente anche pagine web contenenti errori e imperfezioni nell’uso dei tag ma questa non è una buona scusa per non essere attenti e precisi nella scrittura del codice.

Commenti e loro utilizzo

- sintassi: <!--commento --> (trattini alti)
- descrivono una porzione di codice che non deve essere renderizzata
- utili per appuntarsi qualcosa, soprattutto se si lavora in team, o per rendere più chiara una parte di codice oppure per nascondere temporaneamente una porzione di codice

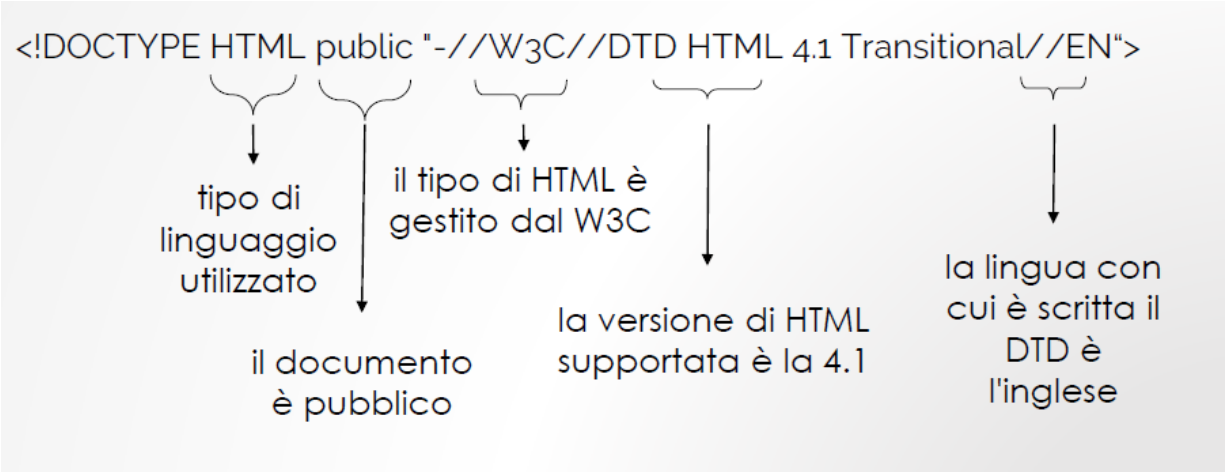
L’utente che visita il sito come abbiamo detto non vedrà il codice commentato che sarà invece visibile visualizzando il codice sorgente della pagina (tasto destro >> visualizza sorgente pagina)

Elementi di codice base di una pagina html

Doctype

Ogni linguaggio della famiglia SGML (standard generalized markup language) deve rispettare certi requisiti fra i quali quello che tutti i simboli siano definiti e descritti usando un DTD (Document Type Definition).
Il DTD per l'HTML definisce i marcatori disponibili e il modo di usarli.

Ecco come doveva essere scritto il doctype prima dell'avvento di HTML5



Oggi invece è sufficiente scrivere

```
<!DOCTYPE html>
```

per comunicare al browser che il contenuto è scritto in html 5

Head & body

- Ogni pagina HTML dopo il doctype ha un **root tag <html>**
- Dentro questo tag solitamente troviamo 2 figli:
    - **<head>** ⇐ Contiene la configurazione della pagina
    - **<body>** ⇐ Contiene il contenuto della pagina

Head

- Informazioni tipiche che puoi impostare nel tuo tag <head>
    - Titolo della pagina: <title> viene mostrato nella tab della scheda del browser
    - Collegamenti verso file esterni (CSS, Fonts, Script): <link>
    - Metadati: <meta> informazioni utili ad applicazioni esterne (es. motori di ricerca) o al browser (es. lingua, codifica dei caratteri utile per la visualizzazione di alfabeti non latini)
- Ricorda: **<head> == configurazione pagina**

Body

- Informazioni tipiche che puoi mettere nel tuo <body>
    - Testo, paragrafo, sezione
    - immagini, links
    - Elenchi, script, tabelle
- Ricorda: **<body> == contenuto, ciò che vedrà l'utente**.

Quindi la **struttura base** di una pagina html sarà la seguente:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
</head>
<body>

</body>
</html>
```

Charset

La direttiva <meta charset="UTF-8"> è un elemento HTML fondamentale che specifica come il browser deve interpretare la codifica dei caratteri di una pagina web. Vediamo insieme perché è così importante.

In primo luogo, dobbiamo capire cosa significa "codifica dei caratteri". Quando scriviamo del testo sul computer, ogni carattere (lettere, numeri, simboli) viene convertito in una sequenza di bit che il computer può elaborare. UTF-8 è uno standard di codifica che definisce come effettuare questa conversione.

UTF-8 è particolarmente importante perché:

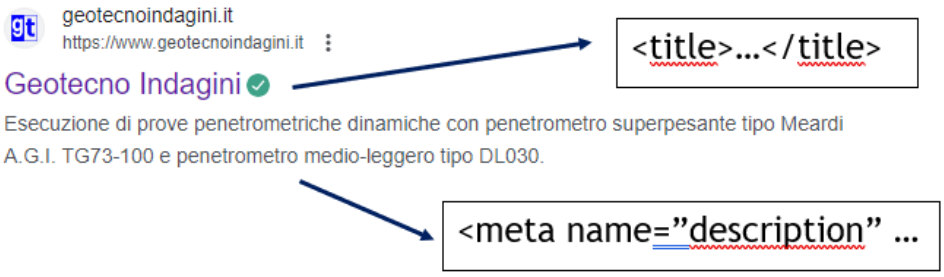
1. Supporta tutti i caratteri Unicode, permettendo quindi di visualizzare correttamente testo in qualsiasi lingua (italiano, cinese, arabo, emoji, ecc.)
2. È retrocompatibile con ASCII, lo standard più vecchio usato per l'inglese
3. È efficiente nell'uso della memoria, usando solo i byte necessari per ogni carattere

Quando inseriamo `<meta charset="UTF-8">` nell'`<head>` della nostra pagina HTML, stiamo dicendo al browser: "Interpreta tutti i caratteri in questa pagina usando la codifica UTF-8".
Senza questa direttiva, il browser potrebbe:

- Interpretare erroneamente i caratteri speciali
- Mostrare simboli strani al posto delle lettere accentate
- Non visualizzare correttamente caratteri non-ASCII

Per esempio, senza una corretta codifica UTF-8, la parola "perché" potrebbe apparire (soprattutto su browser poco aggiornati) come "perchÃ©" o altri simboli incomprensibili.
Vuoi che approfondiamo qualche aspetto particolare della codifica dei caratteri o hai altre domande su come gestire il multilinguismo nelle pagine web?

Title e description



Favicon

```
<link rel="icon" type="image/x-icon" href="/images/favicon.ico">
```

Viewport

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Tramite questa direttiva stiamo dicendo al browser che il sito dovrà essere visualizzato in riferimento alla larghezza del nostro dispositivo. Così facendo, ci mostrerà lo spazio coincidente alla risoluzione del nostro dispositivo!
Analizziamo ogni parte:

width=device-width: dice al browser di impostare la larghezza della pagina uguale alla larghezza dello schermo del dispositivo. E’ importantissimo specificare sempre questo attributo con questo valore!

initial-scale: definisce il livello di zoom iniziale della pagina. Può assumere valori da 0 a 10

Risultati con diversi valori di initial-scale:

- initial-scale=1.0 (valore standard):
  Il contenuto viene mostrato nelle sue dimensioni naturali
  Un pixel CSS corrisponde esattamente a un pixel del dispositivo

È l'impostazione più comune e raccomandata per la maggior parte dei siti

- initial-scale=0.5:
  La pagina viene "rimpicciolita" al 50%
  Più contenuto è visibile sullo schermo
  Il testo diventa più piccolo e potrebbe essere difficile da leggere
  Utile quando vuoi dare una "vista d'insieme" della pagina

- initial-scale=1.5:
  La pagina viene ingrandita del 50%
  Il contenuto appare più grande e più facile da leggere
  Meno contenuto è visibile contemporaneamente sullo schermo
  Può essere utile per contenuti che richiedono maggiore leggibilità

- initial-scale=2.0:
  La pagina viene ingrandita al doppio
  Il contenuto occupa molto più spazio
  Ottimo per accessibilità quando serve un testo molto grande
  Ma richiede molto più scrolling per vedere tutto il contenuto

Potremmo anche settare i valori di

- minimum-scale: (da 0 a 10.0) Indica il limite minimo di zoom-out che l'utente può effettuare
- maximum-scale: (da 0 a 10.0) Indica il limite minimo di zoom-in che l'utente può effettuare
- user-scalable: (yes/no) Permette di bloccare o lasciare libera la possibilità di zoomare.

Nota: Il viewport meta tag funziona in modo diverso tra desktop e mobile, e questo è un aspetto fondamentale da comprendere.
Su desktop, il viewport meta tag ha un impatto molto limitato, e questo può creare confusione. Vediamo perché:

Su desktop, il browser già gestisce la finestra in modo nativo quindi quando ridimensioni la finestra del browser, il contenuto si adatta naturalmente. Questo succede perché i desktop hanno un "viewport virtuale" che corrisponde alla dimensione della finestra del browser. L'initial-scale in questo caso ha poco effetto perché il browser desktop mantiene il controllo dello zoom indipendentemente.
Al contrario, sui dispositivi mobili, il viewport meta tag è cruciale perché i browser mobile fanno qualcosa di particolare: senza istruzioni specifiche, cercano di mostrare l'intera larghezza di una pagina web (spesso progettata per desktop) rimpicciolendo tutto. È come se cercassero di far entrare un poster in una cartolina.

TAG PRINCIPALI

Headings H1->H6

- h1, h2, h3, h4, h5, h6 sono “**tag per i titoli**”
- Meglio avere **solo un h1 per pagina**, sarà l'elemento più importante per i motori di ricerca
- Devono essere usati secondo un criterio logico gerarchico. H1 è più importante di h2 che può essere utilizzato per i sotto argomenti di h1. Lo stesso vale per h3 che può essere utilizzato per i sotto argomenti di h2 e via dicendo. Questo è molto importante dal punto di vista SEO
- Esempio:

  <h1> QUESTO E’ IL TITOLO </h1>

  <h2> Questo è un sottotitolo </h2>

Paragrafo p

- p è il tag paragrafo
- Di solito contiene testo
- Esempio: Questo è un paragrafo

Span

- span è un **contenitore di testo generico**

- Di solito viene utilizzato per definire lo stile di certe porzioni di testo
  Esempio: Questo è un testo

Liste ul ol

- ⇒ lista non ordinata
- ⇒ lista ordinata
- Entrambi sono genitori di <li> (list items)

Esempio : **<ul>**
**
** primo elemento **
**
**
** secondo elemento **
**
**
**

Container generico div

- ⇒ è un elemento di “blocco”, cioè si espande per tutta la larghezza della pagina
- Può essere utilizzato per lo **styling e l'organizzazione degli elementi nella tua pagina**
- È uno dei tag più utilizzati

Esempio: ** **
 Great paragraph
 Second text
** **

Immagine img

- <img> ⇒ img viene utilizzato per eseguire il rendering di un'immagine in una pagina
- Richiede l'attributo src per impostare l'immagine
- Richiede l'attributo alt che verrà mostrato quando l'immagine non è disponibile/accessibilità/SEO
- <img> è un **self-closing tag**

-

Esempio:

Siti dove scaricare immagini gratuite: https://unsplash.com/, https://pixabay.com/it/, https://www.freepik.com/.

Link o anchor a

- <a> ⇒ collega un documento interno o esterno
- Richiede l'attributo **href** per impostare l'URL dell'elemento a cui fare riferimento
- Tutti i figli di <a> saranno cliccabili
- Esempio: **** Go to Google ****
- **href=”#”** >> ricarica la pagina
- **href=”#top”** >> riporta all’inizio della pagina
- **href=”#nomeID”** >> porta a una sezione o a un elemento della pagina che ha come attributo id=”nomeID”. Questo tipo di link viene chiamato bookmark ed è molto usato nei siti one page.

Tel:, mailto: e https://wa.me/

Il prefisso **tel:** non è un tag HTML, ma un protocollo utilizzato all'interno dell'attributo href di un link <a>. Viene usato per creare collegamenti cliccabili a numeri di telefono.

Quando un utente clicca su un link tel: su un dispositivo mobile, si apre l'app del telefono con il numero già inserito.
Su desktop, potrebbe aprire un'applicazione di chiamata come Skype o chiedere quale app usare per la chiamata.

Esempio:
Chiama il nostro ufficio

Similmente, **mailto:** non è un tag HTML ma un protocollo usato nell'attributo href di un link <a>. Serve per creare collegamenti che aprono il client di posta elettronica predefinito dell'utente con un indirizzo email già inserito.

Quando un utente clicca su un link mailto:, si apre il suo client email predefinito con l'indirizzo del destinatario già compilato.

Si possono anche precompilare altri campi come oggetto e corpo dell'email.

Esempio base:

```
<a href="mailto:esempio@email.com">Invia un'email</a>
```

Esempio più avanzato

```
<a href="mailto:esempio@email.com?subject=Richiesta%20Informazioni&body
=Salve,%20vorrei%20alcune%20informazioni.">Richiedi informazioni</a>
```

Similmente per avviare una chat whatspp si può usare https://wa.me/numero di telefono

Entities

Le entities HTML sono codici speciali utilizzati per rappresentare caratteri riservati in HTML o caratteri che potrebbero essere difficili da digitare direttamente.

Le entità permettono quindi di visualizzare correttamente simboli speciali che altrimenti potrebbero essere interpretati come codice HTML.

> Un'entità inizia sempre con '&' e termina con ';'.

Può essere scritta in due modi:

Nome: &nome; (es. < per <)

Numero: &#codiceNumerico; (es. < per <)

Entità comuni:

< : minore di (<)

&gt>: maggiore di (>)

& : e commerciale (&)

" : virgolette doppie (")

'' : apostrofo (')

  : spazio vuoto che viene sempre renderizzato come uno spazio. Nota: il browser se aggiungiamo n spazi con la barra spaziatrice ne renderizza sempre solo uno

Le entità sono utili quando si lavora con testo in HTML, specialmente per garantire che il contenuto sia visualizzato correttamente in tutti i browser.
Vantaggio: L'uso delle entità rende il codice HTML più robusto e compatibile con diversi sistemi.

Tabelle

- <table> ⇒ contenitore principale
- <tr> ⇒ rappresenta le righe
- <td> ⇒ rappresenta una cella di dati
- <th> ⇒ rappresenta una cella contenente un'intestazione
- <caption> ⇒ rappresenta una didascalia, spesso il titolo della tabella
- <thead> ⇒ aggiunge una intestazione separata alla tabella
- <tbody> ⇒ rappresenta il corpo principale della tabella
- <tfoot> ⇒ crea un piè di pagina separato per la tabella
- Esempio:

```
<table>
    <tr>
        <td>Cella 1</td>
        <td>Cella 2</td>
        <td>Cella 3</td>
    </tr>
    <tr>
        <td>Cella 4</td>
        <td>Cella 5</td>
        <td>Cella 6</td>
    </tr>
</table>
```

Nota: in css per evitare il doppio bordo tipico delle tabelle html usare la regola

```
border-collapse:collapse
```

applicato al tag <table>

Form

Immagina un form come un modulo cartaceo che devi compilare.

Il form HTML è esattamente questo: un modo strutturato per raccogliere informazioni dagli utenti e inviarle da qualche parte. In genere questo “qualche parte” è il server.

Possiamo quindi pensarlo come una conversazione tra l'utente e il server

Un form è definito dall'elemento **<form>** e può includere vari tipi di controlli per l'input degli utenti, come campi di testo, pulsanti di opzione, checkbox, menu a discesa e pulsanti di invio.

Elemento <form>

L'elemento **<form>** è il contenitore principale per tutti i controlli di input. Ecco gli attributi principali:

- **action**: Specifica l'URL a cui i dati del form saranno inviati. E’ come l'indirizzo su una busta: dice al browser dove inviare i dati
- **method**: Definisce il metodo HTTP da utilizzare per inviare i dati (il tipo di spedizione). I valori comuni sono **GET** e **POST**. Il metodo GET passa i dati in chiaro nell’url, il metodo POST (quasi sempre utilizzato) cripta invece i dati.

Esempio:

```
<form action="/submit-form" method="post">
    <!-- Elementi del form -->
</form>
```

Elementi di Input Principali

<input>

L'elemento **<input>** è il più versatile tra gli elementi di input. Gli attributi principali includono:

- **type**: Specifica il tipo di input. Valori comuni sono **text**, **password**, **email**, **number**, **file**, **checkbox**, **radio**, **submit**, **reset**, **button**, **hidden**.
- **name**: Assegna un nome al controllo di input, utile per identificare il dato inviato.
- **value**: Imposta il valore iniziale dell'input.
- **placeholder**: Testo di suggerimento visibile quando il campo è vuoto.
- **required**: Indica che il campo è obbligatorio.
- **readonly**: Rende il campo di sola lettura.
- **disabled**: Disabilita il campo.

Esempio:

```
<input type="text" name="username" placeholder="Enter your username" required>

<input type="password" name="password" placeholder="Enter your password" required> il testo immesso è mascherato

<input type="email" name="email" placeholder="Enter your email"> il testo immesso è mascherato

<input type="number" name="age" min="0" max="120">

<input type="file" name="profile-picture">

<input type="checkbox" name="subscribe" value="newsletter">

<input type="radio" name="gender" value="male"> Male

<input type="radio" name="gender" value="female"> Female

<input type="submit" value="Submit">

<input type="reset" value="Reset">

<input type="button" value="Click me">

<input type="hidden" name="user-id" value="12345">
```

<textarea>

L'elemento **<textarea>** è utilizzato per l'input di testo multi-linea.

- **rows**: Numero di righe visibili.
- **cols**: Numero di colonne visibili.
- **name**: Assegna un nome al controllo.

Esempio:

```
<textarea name="comments" rows="4" cols="50" placeholder="Enter your comments"></textarea>
```

<select> e <option>

L'elemento **<select>** crea un menu a discesa.

- **name**: Assegna un nome al controllo.
- **multiple**: Permette la selezione di più opzioni.

L'elemento **<option>** definisce le opzioni disponibili.

Esempio:

```
<select name="country">
    <option value="usa">United States</option>
    <option value="canada">Canada</option>
    <option value="mexico">Mexico</option>
</select>

<select name="colors" multiple>
    <option value="red">Red</option>
    <option value="green">Green</option>
    <option value="blue">Blue</option>
</select>
```

<button>

L'elemento **<button>** crea un pulsante.

- **type**: Specifica il tipo di pulsante (**button**, **submit**, **reset**).
- **name**: Assegna un nome al pulsante.
- **value**: Imposta il valore inviato con il pulsante.

Esempio:

```
<button type="submit">Submit</button> (richiama la action del form)
<button type="reset">Reset</button> (svuota i campi)
<button type="button">Click me</button>
```

Elementi di Raggruppamento

<fieldset> e <legend>

L'elemento **<fieldset>** raggruppa elementi correlati all'interno di un form, mentre l'elemento **<legend>** fornisce una leggenda per il gruppo.

Esempio:

```
<fieldset>
```

```
        <legend>Personal Information</legend>
        <label for="fname">First name:</label>
        <input type="text" id="fname" name="firstname" required><br>
        <label for="lname">Last name:</label>
        <input type="text" id="lname" name="lastname" required>
    </fieldset>
```

Etichette e Associazioni

<label>

L'elemento **<label>** associa un'etichetta ad un elemento di input, migliorando l'accessibilità e rendendo possibile la presa del focus dell'input anche selezionando la label.

for: Collegamento all'**id** dell'elemento di input.

Esempio:

```
<label for="username">Username:</label>
<input type="text" id="username" name="username" required>
```

Validazione

- **pattern**: Espressione regolare per validare l'input.
- **maxlength**: Lunghezza massima accettata per l'input.
- **min** e **max**: Valori minimo e massimo per input numerici.

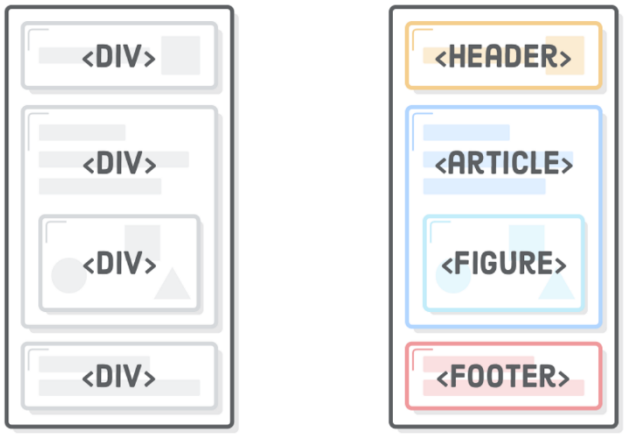
Altri attributi utili

- **step**: Incrementi validi per input numerici o range.
- **autofocus**: Imposta il focus automatico all'elemento quando la pagina viene caricata.

HTML5 e i tag semantici

HTML5 sostituisce la ripetizione di all'interno della pagina con alcuni tag semantici:

- **Header**: definisce la parte iniziale di un documento
- **Footer**: definisce la parte finale di un documento
- **Nav**: contiene i link del menu di navigazione e viene solitamente inserito all'interno del tag <header>
- **Main**: definisce il contenuto principale di un documento. Può esserci solo un main all'interno del body
- **Article**: porzioni di testo
- **Section**: definisce una sezione del documento
- **Aside**: definisce un contenuto di minore importanza
- **Figure**: immagine che è inclusa all'interno di una sezione semantica



CSS

- **Css** =>Cascading Style Sheet
- Definisce un insieme di regole su come il DOM (Document object model) viene tradotto in forma visiva
- Le regole **Cascading** style descrivono la **priorità di come vengono visualizzati gli stili su una pagina**
- Le regole possono essere semplici come definire colori, dimensioni, caratteri e applicare concetti di design tradizionali come contrasto, allineamento, prossimità, ecc. oppure permettono di creare animazioni e gestire la responsività di una pagina web

Come definire gli stili

- **Inline** usando attribute style="..."

   Lorem Ipsum
- Usando **<style> tag** in <head>

  <style> selector { /* rules */ } </style>
- **Importando un file esterno**

  <link rel="stylesheet" href="styles.css" />

Anatomia di una regola css

```
Selettore {
    proprietà 1:valore 1;
    proprietà 2:valore 2;
    ……
    proprietà n:valore n;
}
```

Alcuni esempi di styling

- **color**: red; // imposta il colore del testo su rosso

- **background-color**: blue; // imposta il colore di sfondo sul blu
- **margin-top**: 10px; // imposta il margine in alto su 10 pixel
- **font-weight:** bolder; // imposta il testo in grassetto
- **border:** 1px solid red; // crea un bordo rosso di 1 pixel attorno all'elemento
- **padding**: 10px; // creare un padding di 10 pixel per il contenuto dell'elemento
- **text-align**: center; // allinea il contenuto al centro

https://developer.mozilla.org/en-US/docs/Web/CSS/Reference

Selettori in base al tag

- Puoi specificare regole per ogni dato elemento con un tag specifico
  esempio:

```
div {
    border: 1px solid red;
} // darà un bordo di 1px colore rosso a tutti i div della pagina
```

- Queste regole si applicheranno a TUTTI i tag della pagina
- Il nome del tag sta creando un "selettore" che sceglierà come target tutti i tag con lo stesso nome sulla pagina

Selettori in base a tag multipli

- È possibile creare selettori più complessi specificando più di un nome di tag
  esempio:

```
div p {
    color: purple;
} // tutti i paragrafi contenuti dentro un div avranno il colore viola
```

Selettori in base agli attributi class e id

Spesso la selezione per tag non rappresenta la soluzione migliore. Potresti per esempio voler selezionare un elemento specifico o una serie di elementi con caratteristiche comuni ma magari rappresentati da tag diversi (o dallo stesso tag).

Ecco gli attributi class e id ci vengono in aiuto.

- Ogni elemento html può avere un solo attributo id il cui valore deve essere unico in tutto il documento.
- Ogni elemento html può avere uno o più attributi class che possono essere comuni ad altri elementi.

Pseudo-Classi CSS

I selettori per pseudo-classi sono dei selettori CSS preceduti da due punti e servono per selezionare degli elementi in base a determinate condizioni.
Andiamo per ordine: che cos' è una pseudo-classe? Ti è mai capitato di passare il mouse su un elemento e vederlo cambiare? Ecco: molto probabilmente, in quel caso, quello che è cambiato è lo stato dell'elemento che è passato dallo stato normale allo stato di hover. Hover, è una pseudo-classe, ovvero una classe che definisce il modo in cui verrà raffigurato un elemento in un determinato stato.
Le pseudo-classi sono un elemento molto utile per conferire dinamicità.

Vediamo qual è la sintassi corretta e, successivamente, quali sono le principali pseudo-classi, divise per gli elementi del linguaggio html a cui appartengono.

```
button {
  border: solid red 2px;
  background-color: transparent;
  color : red;
}
button:hover {
/*prorietà css da modificare*/
  background-color: red;
  color : white;
}
```

Eseguendo il codice precedente, possiamo vedere come, al passaggio del mouse sopra un bottone, questo cambi il colore del testo e il background.

Principali pseudo classi:

Pseudo classi tag anchor <a>

:link – Seleziona solo i tag <a> **con all'interno l'attributo href**.

:**visited** – Seleziona i link che sono già stati visitati dall'utente, ovviamente in base alla cronologia del browser.

:**active** – Seleziona un link nel momento in cui viene attivato, ovvero, nel momento in cui viene cliccato e non è ancora stato rilasciato il tasto.

Pseudo classi tag di input

:**focus** – Seleziona il campo di input nel momento in cui l'utente ha cliccato per scriverci. è molto utile per dare uno stile diverso a un tag di input quando l'utente clicca e quindi far capire che può scriverci all'interno.

:**target** – La pseudo-classe target è usata insieme agli ID e corrisponde quando il tag hash nell'URL corrente corrisponde a quell'ID. Quindi, se ci si trova all'URL www.nomeSito.it/#contatti, il selettore #contatti:target corrisponderà. Questo può essere estremamente potente.

:**enabled** – Seleziona gli input che son di default nello stato abilitato.

:**disabled** – Seleziona gli input che hanno l'attributo "disabled".

:**checked** – Seleziona i checkbox nel momento in cui vengono selezionati.

:**indeterminate** – Seleziona gli input radio che non sono ancora stati selezionati.

:**required** – Seleziona gli input che hanno l'attributo "required".

:**optional** – Seleziona gli input che non hanno l'attributo "required" e sono quindi opzionali.

Pseudo Classi in base alla posizione dell'elemento

:**first-child** – Seleziona il primo figlio all'interno di un elemento

:**last-child** – Seleziona l'ultimo figlio all'interno di un elemento

:**nth-child()** – Seleziona gli elementi in base alla posizione all'interno di un elemento.

even / odd – Seleziona rispettivamente tutti gli elementi pari o tutti gli elementi dispari

n – Inserendo nelle parentesi tonde un numero selezionerai solo l'elemento che è in quella posizione

:**nth-type**() – Funziona come :nth-child ma si usa in contesti in cui gli elementi dello stesso livello sono di tipo diverso. Ad esempio, se all'interno di un ci fossero una serie di paragrafi e una serie di immagini.

:**first-of-type** – Seleziona il primo elemento di un determinato tipo di elemento.

:**last-of-type** – Seleziona l'ultimo elemento di un determinato tipo di elemento.

:**nth-last-of–type()** – Funziona con nth-type ma partendo da sotto.

:**nth-last-child()** – Funziona con nth-child ma partendo da sotto.

:**only-of-type** – Seleziona l'elemento se è l'unico di quel tipo all'interno dell'elemento padre

```
<div>
  <ul> <!-- ul:only-of-type -->
  <li>primo</li> <!-- li:first-child | li:nth-child(odd) -->
<li>secondo</li> <!-- li:nth-child(2) | li:nth-child(even) -->
<li>terzo</li> <!-- li:last-child | li:nth-child(odd) -->
</ul>
  <div>Primo div</div> <!-- div div:first-of-type -->
<p>primo paragrafo</p>
<div>secondo div</div> <!-- div div:last-of-type -->
  <p>secondo paragrafo</p> <!-- p:nth-of-type(2) -->
</div>
```

Pseudo-Elementi CSS

Gli pseudo-elementi sono degli elementi che non esistono nel linguaggio di markup html, ma vengono creati tramite dei selettori css che sono preceduti da dei doppi due punti (::). Non essendo elementi DOM reali non possono essere selezionati con JavaScript o manipolati come normali elementi HTML.

I principali sono **::after** e **::before**

L’unica differenza tra i due è che con “after” il contenuto creato con il linguaggio css si posizionerà dopo l’elemento interessato, mentre con “before”, prima.

Esempio:

```
h1::before {
  content: ‘prima’;
}
h1::after {
  content : ‘dopo’;
}
```

Come puoi notare, ai fianchi del tuo titolo si sono aggiunti la parola “prima” a sinistra e “dopo” a destra.

Attraverso la proprietà content puoi, quindi, specificare il contenuto dello pseudo-elemento, vediamo quali sono i possibili valori da inserire:

- **testo** – Puoi inserire qualsiasi tipo testo come valore del content e questo apparirà prima o dopo l’elemento
- **immagine** – Puoi inserire come valore l’url di un’immagine (content : url(percorso/per/immagine.jpg)), questa, verrà inserita con le esatte dimensioni dell’immagine e non sarà possibile ridimensionarla, per questo motivo non è molto utilizzato.
- niente – (content : “”)Possiamo inserire un contenuto vuoto se vogliamo creare un elemento grafico oppure un elemento con una dimensione prestabilita e un’immagine di sfondo.

Altri pseudo elementi utili sono:

::first-letter - Seleziona la prima lettera di un elemento
::first-line - Seleziona la prima riga di un elemento
::marker - Controlla l'aspetto dei punti elenco o numeri nelle liste
::placeholder - Modifica l'aspetto del testo placeholder negli input

::selection - Stilizza il testo quando viene selezionato dall'utente

Approfondimento: L'uso dei due punti doppi (::) negli pseudoelementi CSS ha una storia interessante e una ragione pratica molto importante che aiuta a comprendere meglio come funziona CSS.
Quando CSS2 fu introdotto, sia gli pseudoelementi che le pseudoclassi usavano un singolo due punti (:). Con l'arrivo di CSS3, gli sviluppatori del linguaggio si resero conto che era importante poter distinguere chiaramente tra questi due concetti diversi, perché funzionano in modi fondamentalmente differenti.
Pensiamo a questa distinzione come a due categorie diverse:

Le pseudoclassi (un solo :) rappresentano stati speciali di elementi esistenti. È come descrivere "quando" un elemento è in una certa condizione - per esempio, :hover quando il mouse è sopra l'elemento.
Gli pseudoelementi (due ::) invece creano effettivamente nuovi elementi virtuali che non esistono nell'HTML. È come aggiungere "cosa" nuova alla pagina - per esempio, ::before crea un nuovo contenuto prima dell'elemento.

Per fare un'analogia pratica, è come la differenza tra:

Descrivere quando una porta è aperta (:hover - pseudoclasse)
Aggiungere effettivamente una maniglia alla porta (::before - pseudoelemento)

Il doppio due punti è stato introdotto proprio per rendere questa distinzione immediatamente visibile nel codice. È come se CSS ci stesse dicendo: "Attenzione, qui non stiamo solo modificando qualcosa che esiste, stiamo creando qualcosa di nuovo!"
È interessante notare che per retrocompatibilità, molti pseudoelementi funzionano ancora con un singolo due punti, ma usare la notazione doppia è considerata la best practice moderna perché:

Rende il codice più chiaro e auto-documentante

Aiuta altri sviluppatori a capire immediatamente che stanno lavorando con un pseudoelemento

Si allinea con le specifiche moderne di CSS

Riepilogo di selettori e pseudoclassi

:active	a:active	Selects the active link
::after	p::after	Insert something after the content of each <p> element
::before	p::before	Insert something before the content of each <p> element
:checked	input:checked	Selects every checked <input> element
:default	input:default	Selects the default <input> element
:disabled	input:disabled	Selects every disabled <input> element
:empty	p:empty	Selects every <p> element that has no children (including text nodes)
:enabled	input:enabled	Selects every enabled <input> element
:first-child	p:first-child	Selects every <p> element that is the first child of its parent
::first-letter	p::first-letter	Selects the first letter of every <p> element
::first-line	p::first-line	Selects the first line of every <p> element
:first-of-type	p:first-of-type	Selects every <p> element that is the first <p> element of its parent
:focus	input:focus	Selects the input element which has focus
:fullscreen	:fullscreen	Selects the element that is in full-screen mode
:hover	a:hover	Selects links on mouse over
:in-range	input:in-range	Selects input elements with a value within a specified range
:indeterminate	input:indeterminate	Selects input elements that are in an indeterminate state
:invalid	input:invalid	Selects all input elements with an invalid value
:lang(*language*)	p:lang(it)	Selects every <p> element with a lang attribute equal to "it" (Italian)
:last-child	p:last-child	Selects every <p> element that is the last child of its parent
:last-of-type	p:last-of-type	Selects every <p> element that is the last <p> element of its parent
:link	a:link	Selects all unvisited links
::marker	::marker	Selects the markers of list items
.class1.class2	.name1.name2	Selects all elements with both *name1* and *name2* set within its class attribute
.class1 .class2	.name1 .name2	Selects all elements with *name2* that is a descendant of an element with *name1*
#id	#firstname	Selects the element with id="firstname"
*	*	Selects all elements
element	p	Selects all <p> elements
element.class	p.intro	Selects all <p> elements with class="intro"
element,element	div, p	Selects all elements and all <p> elements
element element	div p	Selects all <p> elements inside elements
element>element	div > p	Selects all <p> elements where the parent is a element
element+element	div + p	Selects the first <p> element that is placed immediately after elements
element1~element2	p ~ ul	Selects every element that is preceded by a <p> element
[*attribute*]	[target]	Selects all elements with a target attribute
[*attribute=value*]	[target="_blank"]	Selects all elements with target="_blank"
[*attribute~=value*]	[title~="flower"]	Selects all elements with a title attribute containing the word "flower"
[*attribute|=value*]	[lang|="en"]	Selects all elements with a lang attribute value equal to "en" or starting with "en-"
[*attribute^=value*]	a[href^="https"]	Selects every <a> element whose href attribute value begins with "https"
[*attribute$=value*]	a[href$=".pdf"]	Selects every <a> element whose href attribute value ends with ".pdf"
[*attribute*=value*]	a[href*="w3schools"]	Selects every <a> element whose href attribute value contains the substring "w3schools"

:not(*selector*)	:not(p)	Selects every element that is not a <p> element
:nth-child(*n*)	p:nth-child(2)	Selects every <p> element that is the second child of its parent
:nth-last-child(*n*)	p:nth-last-child(2)	Selects every <p> element that is the second child of its parent, counting from the last child
:nth-last-of-type(*n*)	p:nth-last-of-type(2)	Selects every <p> element that is the second <p> element of its parent, counting from the last child
:nth-of-type(*n*)	p:nth-of-type(2)	Selects every <p> element that is the second <p> element of its parent
:only-of-type	p:only-of-type	Selects every <p> element that is the only <p> element of its parent
:only-child	p:only-child	Selects every <p> element that is the only child of its parent
:optional	input:optional	Selects input elements with no "required" attribute
:out-of-range	input:out-of-range	Selects input elements with a value outside a specified range
::placeholder	input::placeholder	Selects input elements with the "placeholder" attribute specified
:read-only	input:read-only	Selects input elements with the "readonly" attribute specified
:read-write	input:read-write	Selects input elements with the "readonly" attribute NOT specified
:required	input:required	Selects input elements with the "required" attribute specified
:root	:root	Selects the document's root element
::selection	::selection	Selects the portion of an element that is selected by a user
:target	#news:target	Selects the current active #news element (clicked on a URL containing that anchor name)
:valid	input:valid	Selects all input elements with a valid value
:visited	a:visited	Selects all visited links

Specificità

Per specificità si intende la priorità che una regola ha sulle altre che si riferiscono allo stesso selettore. La proprietà con valore di specificità maggiore avrà la precedenza sulle altre; nel caso in cui due o più regole, come in precedenza, abbiamo tale valore uguale, verrà assegnata l'ultima inserita nell'ordine a cascata.

Semplificando in una regola generale dalla regola più importante verso la meno importante abbiamo

Style inline >> ID >> Class, Pseudo Class, Attributo >> Elemento, pseudo elemento

Il valore di specificità di una regola CSS può essere espresso sotto-forma di numero a quattro cifre. Come nel nostro sistema numerico, l'importanza delle cifre è ordinata da sinistra verso destra quindi un valore più alto indica una specificità più alta.

Come appena detto, il valore è costituito da un numero a quattro cifre. Data una specificità come la seguente:
a, b, c, d

le seguenti regole consentono di assegnare un valore ad ogni cifra, partendo da sinistra:

Assegna valore 1 ad a se la regola è contenuta all'interno dell'attributo style, 0 altrimenti.
Aggiungi 1 a b per ogni ID presente nella regola, 0 altrimenti.
Aggiungi 1 a c per ogni classe, pseudo-classe e attributo presente nella regola, 0 altrimenti.
Aggiungi 1 a d per ogni elemento e pseudo-elemento contenuto nella regola.
N.B. il selettore universale*, non ha alcuna specificità, ed il suo valore finale è 0.

Esempio:

```
li {}                 /* a=0, b=0, c=0, d=1 -> 0001 */
ul li {}              /* a=0, b=0, c=0, d=2 -> 0002 */
ul#menu li {}         /* a=0, b=1, c=0, d=2 -> 0102 */
ul#menu li.active {} /* a=0, b=1, c=1, d=2 -> 0112 */
#menu {}              /* a=0, b=1, c=0, d=0 -> 0100 */
style="color:red"     /* a=1, b=0, c=0, d=0 -> 1000 */
```

```
*                  /* a=0, b=0, c=0, d=0 -> 0000 */
li:first-line {}   /* a=0, b=0, c=0, d=2 -> 0002 */
ul ol+li {}        /* a=0, b=0, c=0, d=3 -> 0003 */
li.red.level {}    /* a=0, b=0, c=2, d=1 -> 0021 */
h1 + *[rel=up] {} /* a=0, b=0, c=1, d=1 -> 0011 */
```

Un metodo semplice per calcolare la specificità:

- Assegna 1000 se la regola è contenuta all'interno dell'attributo style, 0 altrimenti.
- Aggiungi 100 per ogni ID presente nella regola, 0 altrimenti.
- Aggiungi 10 per ogni classe, pseudo-classe e attributo presente nella regola, 0 altrimenti.
- Aggiungi 1 per ogni elemento e pseudo-elemento contenuto nella regola.

La clausola !important

Esempio:

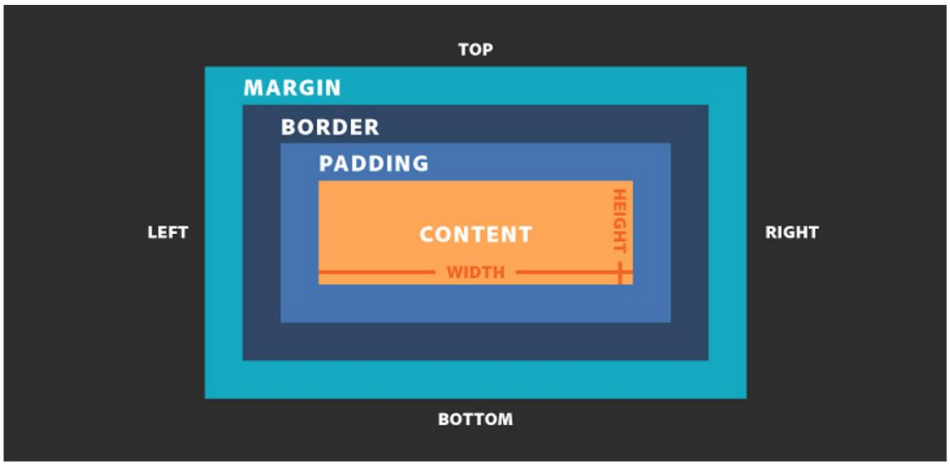
p { color: red !important; }

p { color: green; }

assegna un valore di precedenza alla regola che prevale su tutte le specificità viste in precedenza. Il suo utilizzo è utile quando si ha a che fare con fogli di stile CSS molto complessi e lavorare con le specificità classiche richiede troppo impegno.

Box model

Possiamo considerare tutti gli elementi DOM come scatole, ogni scatola è composta dal **contenuto**, dal **padding**, dal **bordo** e dal **margine**.
Questo è conosciuto come Box Model.



Unità di misura in CSS

- **px** = dimensione assoluta in pixel. font-size: 10px imposterà la dimensione del carattere su 10px
- **em** = relativo alla misura del font impostato nel genitore. 1 em = stessa misura del font del genitore, 0,5 em = metà, 2 em = doppio
- **rem** = relativo alla misura del font impostato nel tag html principale. 1rem = misura predefinita del font per la pagina, 2rem = dimensione doppia
- **vh / vw** = rispetto alla misura della finestra

Colori in CSS

- **Colornames**: **colori predefiniti che puoi usare usando il loro nome**. Esempio: blu, gainsboro, rosa, viola.
- **RGB**: puoi impostare il colore usando la formula RGB e passando i valori per Rosso Verde e Blu. Esempio: colore: rgb(255,0,128)

- **Hexadecimal**: scorciatoia per RGB, si specifica rosso, verde e blu attraverso il loro valore esadecimale. Esempio: colore: #FFAAEF

Regole ereditarie

- Alcune regole, se impostate sull'elemento genitore, vengono automaticamente ereditate sugli elementi figlio

  esempio:
   Lorem Ipsum // il testo di p sarà in rosso

- Questo non si applica a tutte le regole, ma solo ad alcune di esse

  esempio:
   Lorem Ipsum // p non avrà un bordo

https://developer.mozilla.org/en-US/docs/Web/CSS/Reference

L'aspetto del testo

- Un insieme di caratteri studiati per essere raggruppati tra di loro in modo "esteticamente coerente" formano un "**carattere tipografico**

- Il **Font** è, invece, un file che contiene tutti i caratteri tipografici di un determinato "stile di scrittura".

- Più caratteri tipografici molto simili possono essere raggruppati in **famiglie** o famiglie di font.

- Css ci permette di agire su diverse proprietà che definiscono l'aspetto dei caratteri. Le principali sono:

  - **font-family:** si riferisce al nome del carattere. Es: Arial, Roboto, Poppins… Accetta solo "caratteri web" (supportati dal browser)

- **font-size**: si riferisce alla dimensioni del testo che sono espresse nelle diverse unità di misura che abbiamo già visto (px, em, rem..,)
- **font-weight**: si riferisce allo “spessore” del carattere. Accetta sia valori semantici (regular, bold, bolder…) che numeri (100-900). Se si imposta un determinato valore e questo non è supportato da quel carattere verrà utilizzato il valore più vicino.

Posizionamento dei contenuti

Flexbox

Il layout Flexbox (Box Flessibili) è un modo per **gestire, allineare e distribuire lo spazio tra elementi contenuti all'interno di un elemento contenitore, anche nei casi in cui la dimensione degli elementi non sia nota a priori**, e senza utilizzare delle media query. Flexbox è alla base di Bootstrap, noto framework utilizzato per lo sviluppo di interfacce e template HTML responsive.

Flexbox lavora con due assi:

l'**asse principale**: di default è l'asse orizzontale, da sinistra a destra
l'**asse trasversale**: di default è l'asse verticale dall'alto verso il basso.
Utilizzando questi assi gli elementi possono essere disposti, in relazione tra loro, all'interno del box contenitore. CSS Flexbox assicura che lo spazio tra questi elementi venga riempito in modo adeguato.

Come utilizzare il layout Flexbox
Il layout Flexbox funziona attribuendo ad **un elemento contenitore "parent"** (generalmente un "div") la **capacità di modificare la lunghezza, larghezza o ordine dei suoi elementi "figli" secondo regole precise. Saranno gli elementi "figli" ad essere flessibili**.

Ad esempio, possiamo decidere di allinearli da sinistra verso destra, o da destra verso sinistra, o di centrarli nel div contenitore, o di espanderli fino a riempire tutto lo spazio disponibile oppure restringerli in modo tale da prevenire l'overflow, ...

Struttura di un layout Flexbox
Facciamo subito un esempio. Creiamo un file html ("flexbox.html") con all'interno il seguente codice.

```
<!doctype html>
<html lang="it">
  <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
        <style>
        body {
            background-color:#ffffff;
            color:#000000;
        }
        .container {
          background-color: #cccccc;
        }
        .child {
           margin:15px;
           padding:50px;
           background-color:#ff0000;
           text-align: center;
        }
        </style>
        <title>Esempio flexbox</title>
  </head>
  <body>
        <div class="conta            iner">
                <div class="child" id="box1">Box 1</div>
                <div class="child" id="box2">Box 2</div>
                <div class="child" id="box3">Box 3</div>
        </div>
  </body>
</html>
```

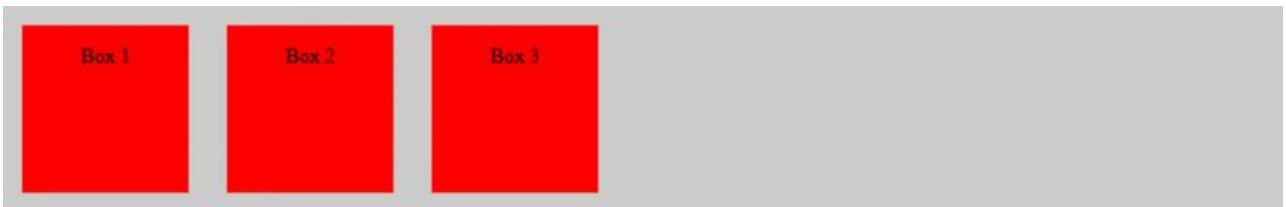
Nell'html il nostro focus sarà sul contenitore "div", con classe "container", che contiene tre "div".

Prova a testare questa pagina web, e vedrai il risultato: i tre div si presentano uno sotto l'altro

Adesso applichiamo un modello Flexbox a questa struttura aggiungendo la classe "container" a cui assegniamo una proprietà "display" con valore "flex".

```
 .container {
  display: flex;

  background-color: #cccccc;
}
```

Riapri la pagina web e vedrai i tre div "figli" affiancati in orizzontale, all'interno del div "contenitore".

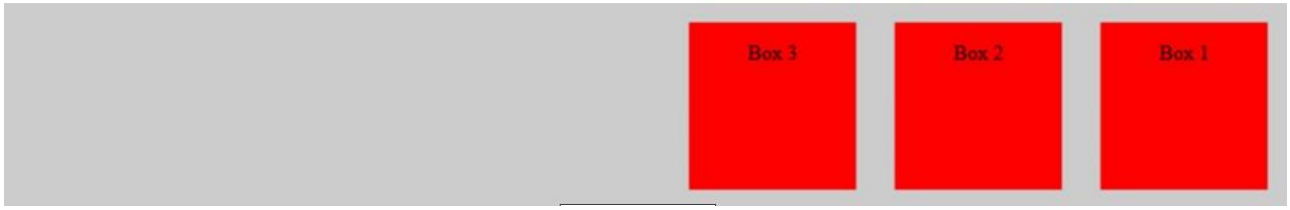


Adesso vediamo come trasformare questa struttura in Flebox, attraverso l'utilizzo di una serie di proprietà da applicare al div "container".

flex-direction

Per affiancare questi div in orizzontale, da sinistra verso destra, cioè come indicati nel codice html, si utilizza la proprietà "**flex-direction**" con valore "**row**", come indicato nel nostro css, ed è il comportamento di default di una Flexbox, per cui anche omettendolo il risultato sarebbe stato il medesimo.

Se vogliamo affiancarli da destra verso sinistra, quindi ribaltandone l'ordine, utilizziamo il valore "**row-reverse**". L'ordine sarà quindi inverso rispetto a quello indicato nel codice html.



E' possibile anche allineare i div in verticale, dall'alto verso il basso, con il valore "**column**"

e dal basso verso l'alto con "**column-reverse**", per cui il primo elemento è in basso.

Ricapitolando, i possibili valori di flex-direction sono:

row: da sinistra verso destra (default)
row-reverse: da destra verso sinistra
column: dall'altro verso il basso
column-reverse: dal basso verso l'alto

justify-content

La proprietà "justify-content" serve a definire **come allineare gli elementi "child" sull'asse principale del contenitore**, cioè sull'asse orizzontale.

Per centrare i 3 elementi utilizziamo il valore **center**

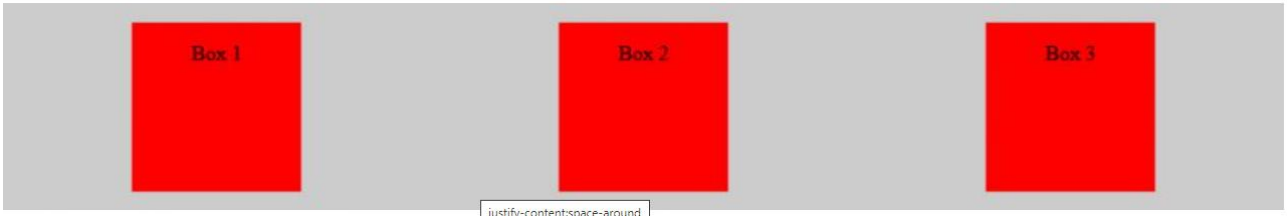
```
 .container {
  display: flex;
  flex-direction: row;
  background-color: #cccccc;
  justify-content:center;
}
```

Ecco il risultato

Per allineare i "child" sul lato sinistro si utilizza **flex-start**

Per allineare i "child" sul lato destro si utilizza **flex-end**

Per distribuire equamente i "child" nel "container", con la stessa spaziatura tra loro, si utilizza **space-around**



Per distribuire i "child" in modo uniforme, con il primo child posizionato all'inizio del "container" e l'ultimo alla fine del "container", si utilizza **space-between**

Ricapitolando, i possibili valori di flex-direction sono:

- **center**: posiziona i child centralmente sull'asse orizzontale, come visto nell'esempio precedente
- **flex-start**: posiziona i child sul lato sinistro
- **flex-end**: posiziona i child sul lato destro
- **space-around**: tutti e tre i child sono distribuiti equamente e con la stessa spaziatura tra uno e l'altro, cioè lo spazio intorno ai child è distribuito in modo uniforme
- **space-between**: lo spazio tra i child è distribuito in modo uniforme, con il primo child è posizionato all'inizio del "container" e l'ultimo alla fine del "container" cioè sul lato destro
- **space-evenly**: lo spazio tra i child è distribuito in modo uniforme anche rispetto al bordo (stessa distanza)
- **align-items**

Mentre proprietà "**justify-content**" serve a definire come allineare gli elementi "child" sull'**asse principale del contenitore**, di default quello orizzontale, "**align-items" lavora in modo simile ma sull'asse verticale**.

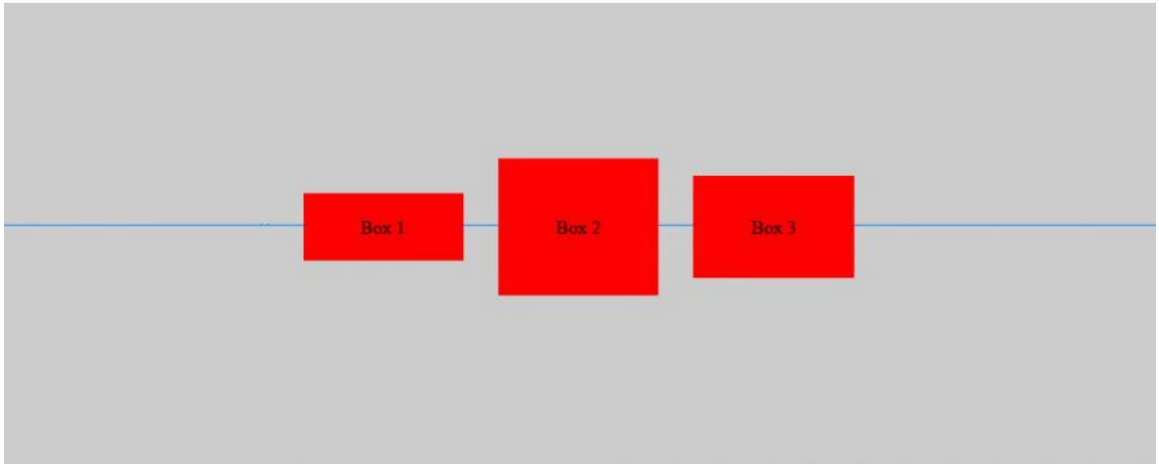
La proprietà "align-items" serve a stabilire **come allineare i child sull'asse trasversale** del contenitore, quindi generalmente in verticale.

Cominciamo con il dare una altezza al div contenitore (400px), inoltre variamo l'altezza dei vari "child", così da capire l'impatto di questo tipo di allineamento, ed utilizziamo la

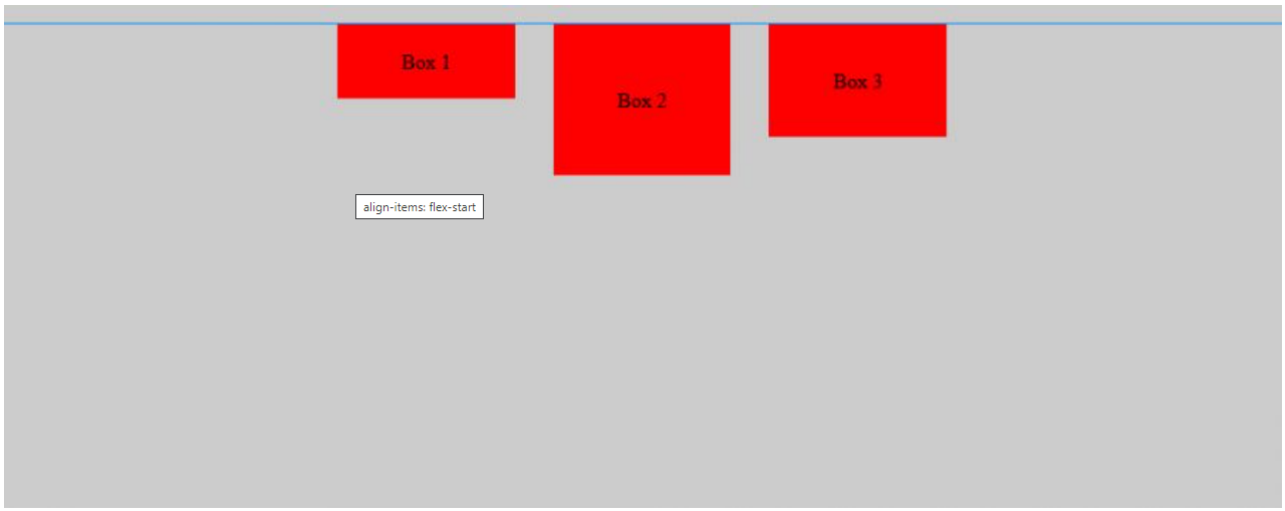
proprietà "align-items" con valore "center" per centrare verticalmente gli elementi "child" rispetto al contenitore

```
.container {
  display: flex;
  flex-direction: row;
  background-color: #cccccc;
  height:400px;
  justify-content:center;
  align-items: center;
}

#box1{padding:20px 50px;}
#box2{padding:50px 50px;}
#box3{padding:35px 50px;}
```

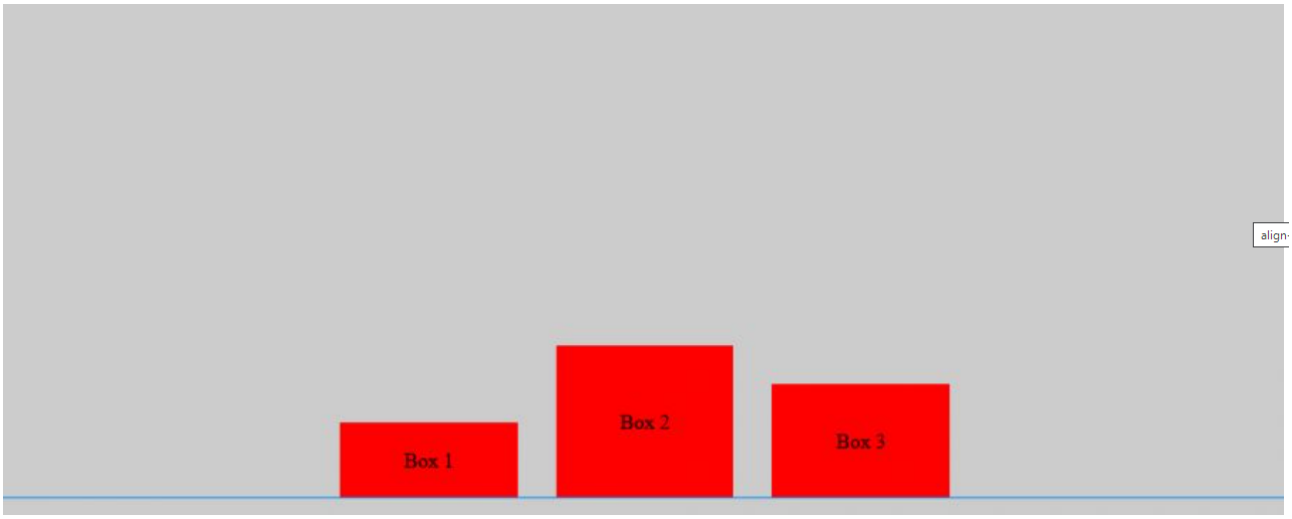
Ecco il risultato, in cui evidenziamo la linea rispetto alla quale vengono allineati i "child"



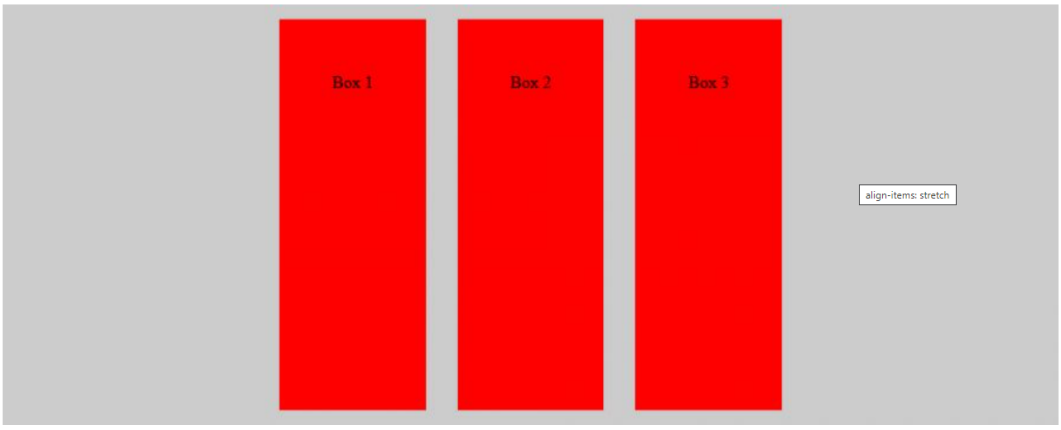
Per posizionare i "child" vicino al lato superiore del "container" utilizziamo flex-start



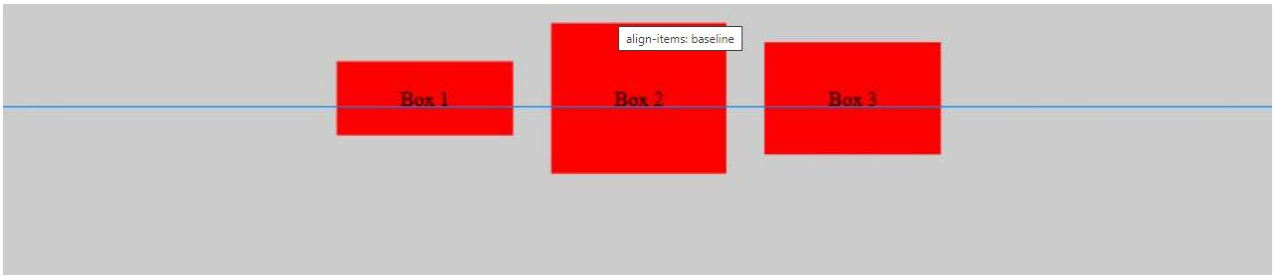
Per posizionare i "child" vicino al lato inferiore del "container" utilizziamo flex-end



Se vogliamo che i "child" occupino tutta l'altezza del "container" utilizziamo **stretch**



Se vogliamo che i "child" vengano allineati rispetto alla baseline del loro testo, utilizziamo **baseline**



Ricapitolando i possibili valori per gestire l'allineamento verticale:

- **center**: i "child" sono posizionati centralmente sull'asse verticale
- **flex-start**: i "child" sono posizionati tutti vicino al lato superiore del "container"
- **flex-end**: i "child" sono posizionati tutti vicino al lato inferiore del "container"
- **stretch**: i "child" occupano tutta l'altezza del "container"
- **baseline**: i "child" sono allineati sulla linea di base del testo contenuto in essi

Nota: Se il "container" dispone di un valore "height", allora il valore di default è "flex-start", altrimenti è "stretch".

flex-wrap

Questa proprietà specifica se i "child" devono essere disposti su un'unica riga oppure su più righe qualora lo spazio della riga sia finito.

Per capire questo concetto cambiamo esempio ed aggiungiamo qualche elemento "child" al nostro html:

```
<div class="container">
        <div class="child box1">1</div>
        <div class="child box2">2</div>
        <div class="child box3">3</div>
```

```
            <div class="child box4">4</div>
            <div class="child box5">5</div>
            <div class="child box6">6</div>
</div>
```

Modifichiamo anche lo stile in questo modo

```
.container{
  background-color: #cccccc;
  display:flex;
  flex-direction:row;
  flex-wrap:wrap;
}

.child{
  font-size:25px;
  text-align:center;
  padding:15px;
  width:40%;
}

.box1{background:green;}
.box2{background:blue;}
.box3{background:red;}
.box4{background:magenta;}
.box5{background:yellow;}
.box6{background:pink;}
```

Normalmente, quando i "child" occupano più dello spazio consentito dalla schermata, quelli in eccesso andranno automaticamente su una nuova riga: con flexbox possiamo cambiare questa regola.

Assegnando, come in questo esempio, a **flex-wrap** il valore **nowrap**, gli elementi resteranno sulla stessa riga comprimendosi

Il valore di default è nowrap, che manda a capo gli elementi che non stanno sulla riga

Mentre con wrap-reverse, la disposizione degli elementi avviene dal basso verso l'alto

Ricapitolando, i possibili valori della proprietà "flex-wrap" sono:

- **nowrap**: per non mandare nessun elemento del parent su un'altra riga
- **wrap**: per mandare a capo gli elementi che non ci stanno sulla stessa riga
- **wrap-reverse**: la disposizione degli elementi avviene dal basso verso l'alto

NOTA: è possibile utilizzare la proprietà flex-flow come sintesi abbreviata per indicare allo stesso tempo flex-direction e flex-wrap.

align-content

Se il "container" ha **"child" disposti su più righe** (quando "flex-wrap: wrap"), la proprietà "align-content" definisce **l'allineamento di ogni riga del "container**". Quindi il focus non è sui "child" ma sulle righe.

Utilizziamo il valore center se vogliamo che tutte le righe si posizionino al centro del contenitore.

Modifichiamo l'esempio precedente, modificando lo stile del "container" in questo modo

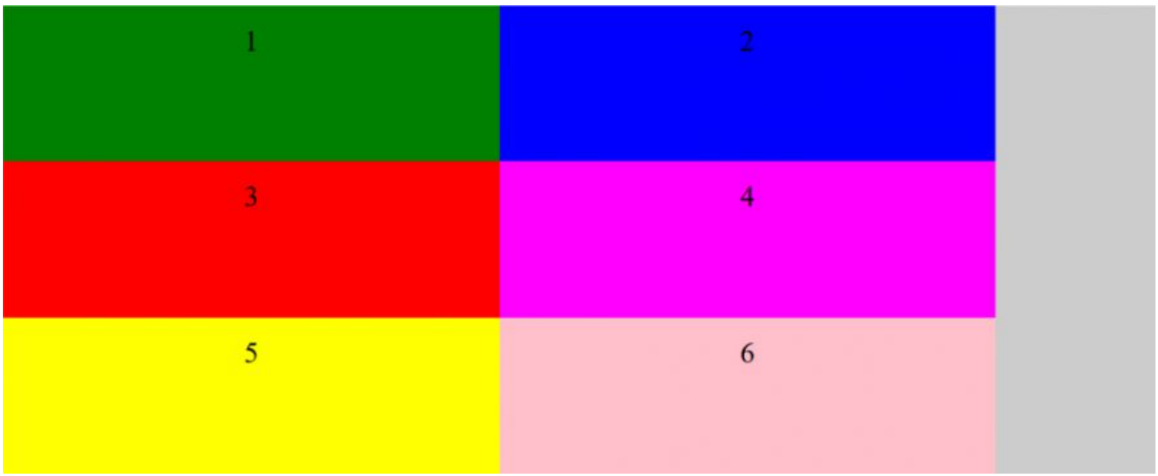
```
.container{
   background-color: #cccccc;
   display:flex;
   height:400px;
   flex-wrap:wrap;
   align-content:center;
}
```

Ecco il risultato

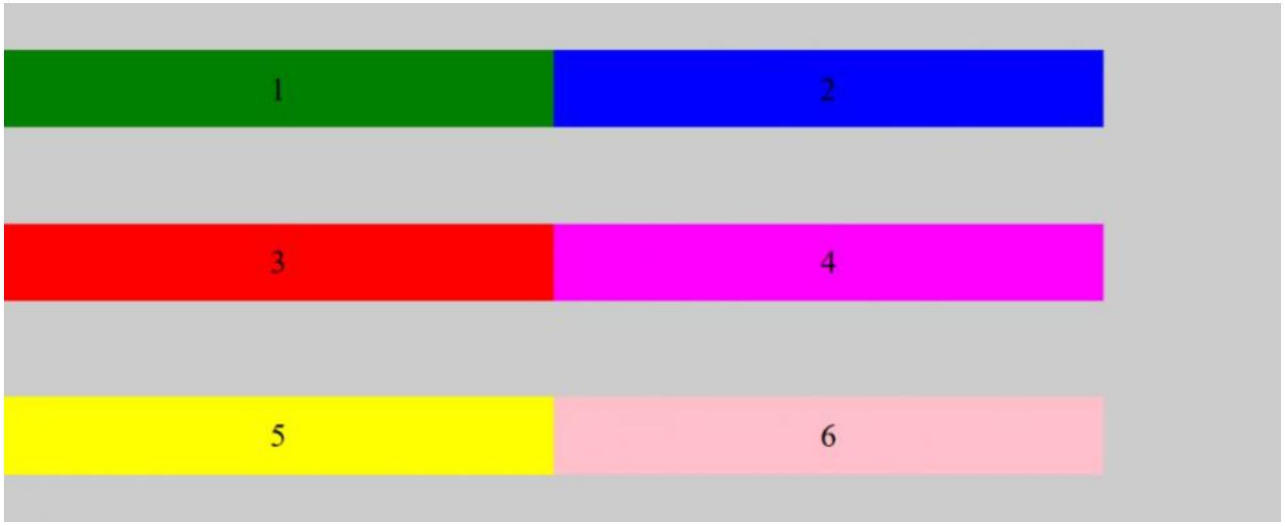
Utilizzando il valore flex-start, tutte le righe vengono posizionate all'inizio del "container".

Utilizzando il valore flex-end, le righe vengono posizionate alla fine del "container".

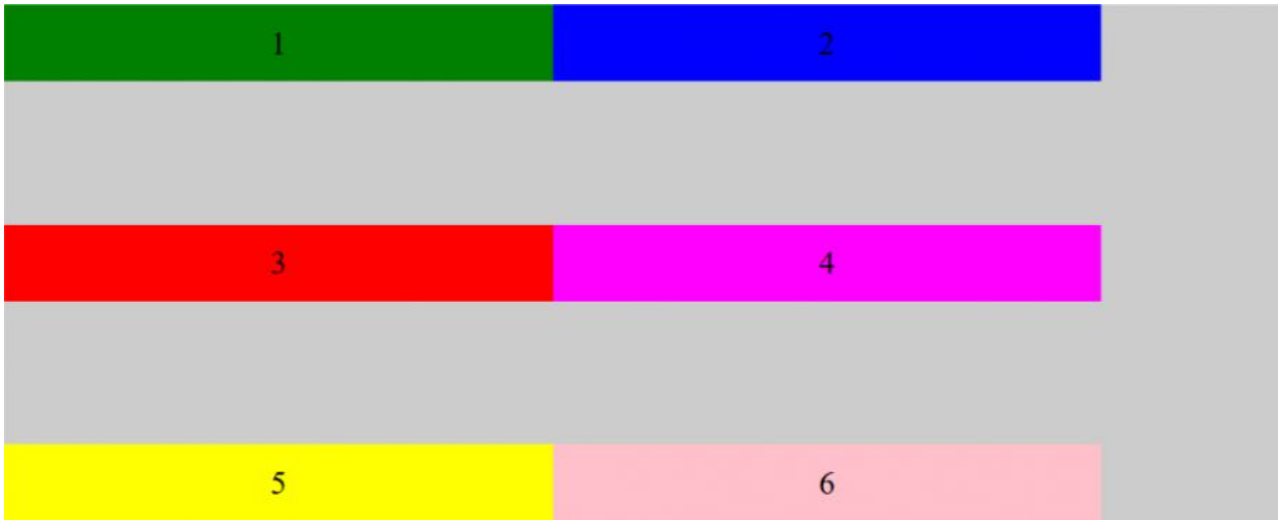
Utilizzando il valore stretch le righe si allungheranno fino a riempire tutto lo spazio rimanente.



Utilizzando il valore space-around le righe avranno la stessa spaziatura (comprese la prima e l'ultima riga).



Infine, utilizzando il valore space-between distribuiamo le righe in modo uniforme con la prima riga posizionata all’inizio del "container" e l’ultima alla fine del "container".



Ricapitolando, align-content può assumere i seguenti valori

- **center**: tutte le righe si posizionano al centro del "container"
- **flex-start**: tutte le righe vengono posizionate all'inizio del "container"
- **flex-end**: le righe vengono posizionate alla fine del "container".
- **stretch**: le righe si allungheranno fino a riempire tutto lo spazio rimanente.
- **space-around**: le righe avranno la stessa spaziatura (comprese la prima e l'ultima riga).
- **space-between**: distribuiamo le righe in modo uniforme con la prima riga posizionata all'inizio del "container" e l'ultima alla fine del "container".

Fino ad adesso abbiamo visto proprietà da applicare al div "container". Adesso passiamo ad alcune proprietà da applicare agli elmenti "child".

Order

Con la proprietà order possiamo modificare l'ordine in cui verranno visualizzati i div "child" rispetto al "container" padre.

Ripartiamo da un esempio con quattro "child " in cui vogliamo che il box1 venga visualizzato come secondo elemento, il box 2 come quarto, il box 3 come primo, e il box 4 come terzo:

```
<!doctype html>
<html lang="it">
  <head>
```

```
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
        <title>Esempio flexbox</title>
        <style>
        body {
            background-color:#ffffff;
            color:#000000;
        }
         .container {
          display: flex;
          flex-direction: row;
          background-color: #cccccc;
        }
        .child {
           margin:15px;
           font-size:2em;
           background-color:#ff0000;
           text-align: center;
        }
        #box1 {order:2;}
        #box2 {order:4;}
        #box3 {order:1;}
        #box4 {order:3;}
        </style>
  </head>
  <body>
        <div class="container">
           <div class="child" id="box1">Box 1</div>
           <div class="child" id="box2">Box 2</div>
           <div class="child" id="box3">Box 3</div>
           <div class="child" id="box4">Box 4</div>
        </div>
        </div>
  </body>
</html>
```

Ecco il risultato

Di **default** la proprietà order ha **valore 0**.

Approfondimento: poiché la proprietà order può essere difficile da interpretare correttamente riporto la traduzione dal sito di mozilla per una maggiore comprensione.

La proprietà order è progettata per disporre gli elementi in gruppi ordinali. Ciò significa che agli elementi viene assegnato un numero intero che rappresenta il loro gruppo. Gli elementi vengono quindi posizionati in ordine visivo in base a tale numero intero, iniziando con i valori più bassi. Se più di un articolo ha lo stesso valore intero, all'interno di quel gruppo gli articoli vengono disposti secondo l'ordine di origine.

Ad esempio, ho 5 elementi flessibili e assegno valori alla proprietà ordine come segue:

Elemento di origine 1:order: 2
Elemento di origine 2:order: 3
Elemento di origine 3:order: 1
Elemento di origine 4:order: 3
Elemento di origine 5:order: 1

Questi elementi verranno visualizzati nella pagina nel seguente ordine:

Elemento di origine 3:order: 1
Elemento di origine 5:order: 1
Elemento di origine 1:order: 2
Elemento di origine 2:order: 3
Elemento di origine 4:order: 3

Prova a cambiare flex-direction in row-reverse e guarda cosa succede: la linea di partenza viene cambiata in modo che l'ordine inizi dal lato opposto.

Gli elementi flessibili hanno un valore predefinito di 0, pertanto gli elementi con un valore intero maggiore di 0 verranno visualizzati dopo tutti gli elementi a cui non è stato assegnato un valore di order esplicito. Ecco perché se attribuisco per esempio al primo elemento di 5 il valore 1 e non assegno nessun ordine agli altri, questo verrà posizionato in 5 posizione in quanto i primi quattro avendo ordine 0 verranno posizionati prima.

Puoi anche utilizzare valori negativi con order, il che può essere molto utile. Se si desidera visualizzare per primo un articolo e lasciare invariato l'ordine di tutti gli altri articoli, è possibile assegnare all'articolo l'ordine -1. Poiché è inferiore a 0, l'elemento verrà sempre visualizzato per primo.

Attenzione: il valore della proprietà order modifica l'ordine in cui gli elementi appaiono visivamente. **Non modifica l'ordine sequenziale di navigazione degli elementi**.

Pertanto, se un utente passa da un elemento all'altro per esempio utilizzando il tasto tab della tastiera, potrebbe ritrovarsi a saltare nel layout in modo molto confuso.

flex-grow

Come possiamo vedere nell'esempio precedente, tolti i margini impostati ai vari "child", esiste dello spazio vuoto in eccesso nel "container".

La proprietà flex-grow consente di impostare il fattore di ingrandimento (di crescita) di un elemento "child" rispetto agli altri, in modo da riempire tutto lo spazio in eccesso, ed ammette un valore numerico (non negativo).

Se, ad esempio, vogliamo che il child "box1" abbia un fattore di crescita del doppio rispetto agli altri child, utilizziamo la proprietà flex-grow con valore 2.

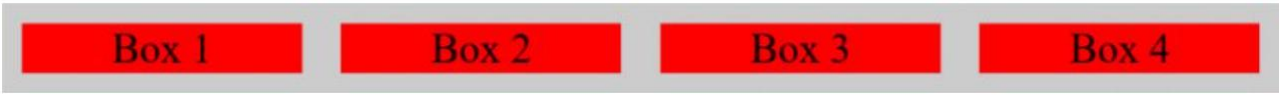
```
#box2 {flex-grow:2;}
```

Se vogliamo che tutti gli elementi abbiano lo stesso fattore di crescita utilizziamo, per tutti, un flex-grow con valore 1.

```
#box1{flex-grow:1;}
#box2{flex-grow:1;}
#box3{flex-grow:1;}
#box4{flex-grow:1;}
```

oppure semplicemente

```
.child{flex-grow:1;}
```



Il valore di default è 0

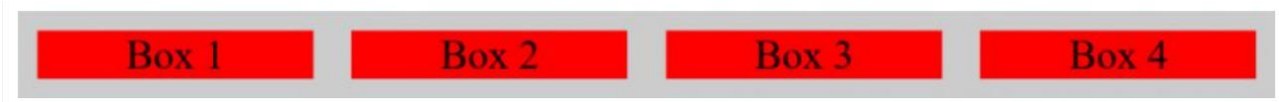
flex-basis

Con flex-grow abbiamo visto come definire un fattore di ingrandimento di un elemento "child" (o più elementi) rispetto agli altri, in modo da riempire tutto lo spazio in eccesso. In questo modo i child si dimensionano in "automatico" .

E' tuttavia possibile **impostare manualmente le dimensioni di un "child"** utilizzando la proprietà flex-basis. Ad esempio, togliamo il flex-grow a tutti i child ed definiamo che ognuno di essi occupi il 25% dello spazio disponibile nel "container".

```
.child {
   margin:15px;
   padding:50px;
   background-color:#ff0000;
   text-align: center;
   flex-basis:25%
}
```

Questo è il risultato



Si possono indicare valori in percentuale, come nel nostro esempio, oppure un valore numerico.

Vediamo di combinare assieme l'utilizzo di flex-basis e flex-grow. Impostiamo a tutti i child un flex-basis del 20%, ed assegnamo al box2 un flex-grow con valore 2

```
.child {
        margin:15px;
        font-size:2em;
        background-color:#ff0000;
        text-align: center;
        flex-basis:20%
}

#box2{flex-grow: 2;}
```

In questo caso il flex-grow è sì il fattore di crescita rispetto agli altri elementi, ma prendendo come riferimento il valore impostato per la proprietà flex-basis, ed il risultato sarà il seguente

Nota: Il valore di flex-basis default è auto e divide lo spazio tra i vari elementi sulla base della proprietà "flex-grow".

Nota: nel caso in cui è impostato "flex-direction:row", oppure se non è specificato nulla, la lunghezza si riferisce alla larghezza (width), altrimenti se è impostato "flex-direction:column" allora la lunghezza si riferisce all'altezza (height).

flex-shrink

Funziona allo stesso modo di flex-grow ma impostando il **fattore di riduzione**, e definisce come devono restringersi i "child" quando non c'è abbastanza spazio nel "container". Ammette un valore numerico (non negativo).

Di **default il suo valore è 1** e fa si che i child vengano equamente distribuiti nello spazio ("container") disponibile.

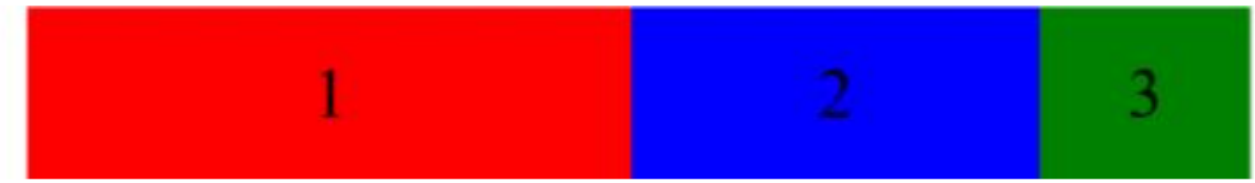
Se ad esempio un elemento ha la proprietà flex-shrink impostata a 2 allora verrà ridotto del doppio rispetto a quello con flex-shrink uguale a 1.

Vediamo questo esempio

```
<!doctype html>
<html lang="it">
  <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
        <title>Esempio flexbox</title>
        <style>
        body {
            background-color:#ffffff;
            color:#000000;
        }
        .container {
                width:100%;
                max-width: 960px;
                background-color:#cccccc;
                font-size: 32px;
                display: flex;
                flex-direction: row;
                }
        .child {
                width:320px;
                text-align:center;
```

```
                padding:20px;
        }
        #box1{background-color:red; flex-shrink:1;}
        #box2{background-color:blue; flex-shrink:2;}
        #box3{background-color:green; flex-shrink:3;}

        </style>
  </head>
  <body>
        <div class="container">
          <div class="child" id="box1">Box 1</div>
          <div class="child" id="box2">Box 2</div>
          <div class="child" id="box3">Box 3</div>
        </div>
        </div>
  </body>
</html>
```

Aprite questo esempio nel vostro browser e poi.... riducete lo schermo e vedrete come i tre box di riducono in modo differente.



flex

Questa proprietà è semplicemente un modo compatto per indicare in un'unica regola i valori per "**flex-grow", "flex-shrink" e "flex-basis**", dove il secondo e terzo parametro sono opzionali e i valori default sono 0, 1 e auto.

align-self

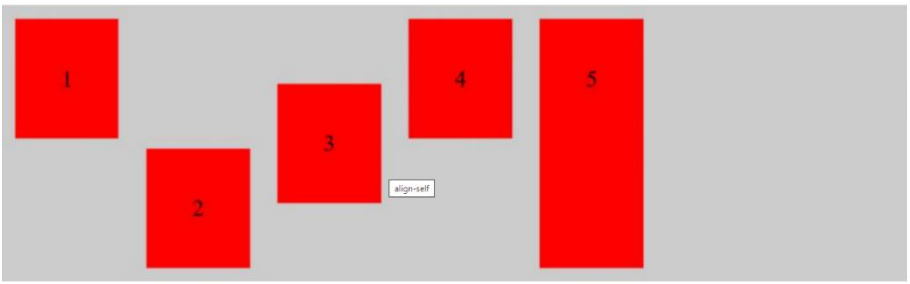
Questa proprietà consente di specificare, per un singolo "child" l'allineamento sull'asse trasversale, cioè in verticale. In definitiva questa proprietà sostituisce l'allineamento impostato dalla proprietà "align-items" impostato nel "container", se presente. Ammette i seguenti valori: flex-start, flex-end, center, baseline, stretch.

Ecco un esempio

```
<!DOCTYPE html>
<html lang="it">
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
<style>
.container {
```

```
  display: flex;
  flex-direction: row;
  background-color: #cccccc;
  height:400px;
}
.child {
  margin:15px;
  padding:50px;
  background-color:#ff0000;
  text-align: center;
  font-size:25px;
}
#box1 {align-self: flex-start;}
#box2 {align-self: flex-end;}
#box3 {align-self: center;}
#box4 {align-self: baseline;}
#box5 {align-self: stretch;}
</style>
</head>
<body>
    <div class="container">
        <div class="child" id="box1">1</div>
        <div class="child" id="box2">2</div>
        <div class="child" id="box3">3</div>
        <div class="child" id="box4">4</div>
        <div class="child" id="box5">5</div>
    </div>
</body>
</html>
```

Ecco il risultato



Position

La proprietà position imposta il modo in cui un elemento viene posizionato in un documento. Le proprietà , top, left, bottom e right determinano la posizione finale degli elementi posizionati.
La proprietà position può assumere questi valori principali:

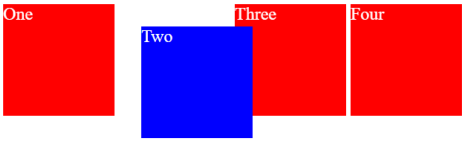
Static:
L'elemento viene posizionato secondo il normale flusso del documento. Le proprietà top, right, bottom, left non hanno alcun effetto. Di default tutti gli elementi hanno position static per cui difficilmente si specifica nel css.

Relative:
L'elemento viene posizionato in base al flusso normale del documento e quindi **spostato rispetto a se stesso** in base ai valori di top, right, bottom, e left.
In pratica gli elementi posizionati relativamente vengono spostati di una determinata quantità rispetto alla loro posizione normale all'interno del documento, ma senza che l'offset influisca sugli altri elementi.

Esempio:

```
<div class="box" id="one">One</div>
<div class="box" id="two">Two</div>
<div class="box" id="three">Three</div>
<div class="box" id="four">Four</div>
```

```
.box {
  display: inline-block;
  width: 100px;
  height: 100px;
  background: red;
  color: white;
}

#two {
  position: relative;
  top: 20px;
  left: 20px;
  background: blue;
}
```



Questo valore crea un nuovo contesto di stacking quando il valore di z-index non è auto. Il suo effetto sugli elementi table-*-group, table-row, table-column, table-cell, e table-caption non è definito.

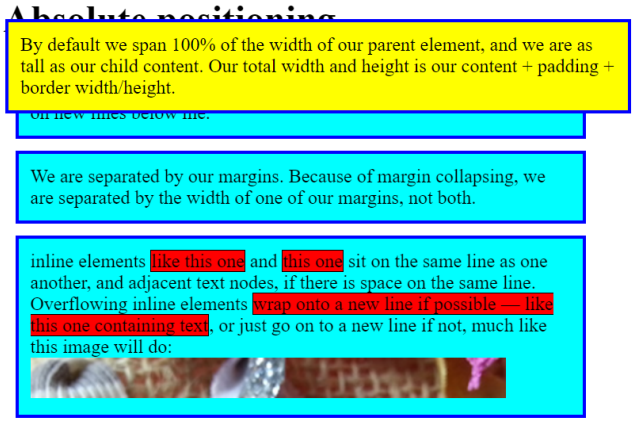
Absolute:

Gli elementi posizionati relativamente rimangono nel normale flusso del documento. Al contrario, un **elemento posizionato in modo assoluto viene escluso dal flusso**; quindi gli altri elementi vengono posizionati come se non esistesse. L'elemento posizionato in modo assoluto è posizionato **rispetto al suo antenato più vicino posizionato** (cioè l'antenato più vicino che non è static). Se non esistesse un antenato posizionato l'elemento si posizionerebbe relativamente al viewport

Esempio

```
<h1>Absolute positioning</h1>

<p>
  I am a basic block level element. My adjacent block level elements sit on new  lines below me.
</p>

<p class="positioned">
  By default we span 100% of the width of our parent element, and we are as tall  as our child content. Our total width and height is our content + padding +  border width/height.
</p>

<p>
  We are separated by our margins. Because of margin collapsing, we are  separated by the width of one of our margins, not both.
</p>

<p>
  inline elements <span>like this one</span> and <span>this one</span> sit on  the same line as one another, and adjacent text nodes, if there is space on the same line. Overflowing inline elements <span>wrap onto a new line if possible — like this one containing text</span>,  or just go on to a new line if not, much like this image will do:  <img src="long.jpg" />
</p>
```

```
body {
  width: 500px;
  margin: 0 auto;
}

p {
  background: aqua;
  border: 3px solid blue;
  padding: 10px;
  margin: 10px;
}

span {
  background: red;
  border: 1px solid black;
}

.positioned {
  position: absolute;
  background: yellow;
  top: 30px;
  left: 30px;
}
```



Fixed:

L'elemento viene **rimosso dal normale flusso di documenti e non viene creato spazio per l'elemento nel layout di pagina.** L'elemento è **posizionato rispetto al viewport**. La

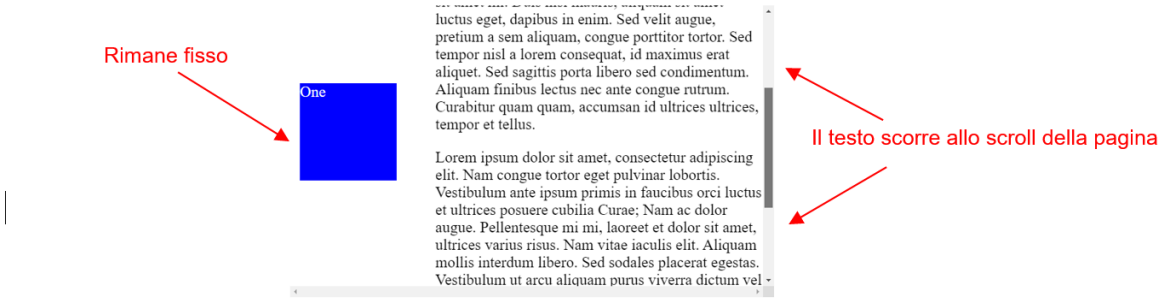
sua posizione finale è determinata dai valori di top, right, bottom, e left. L'elemento **rimane fisso in quella posizione anche allo scrollare degli altri elementi della pagina**.

Esempio

```
<div class="outer">
  <p>
    Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam congue torto    eget pulvinar lobortis. Vestibulum ante ipsum primis in faucibus orci luctus    et ultrices posuere cubilia Curae; Nam ac dolor augue. Pellentesque mi mi, laoreet et dolor sit amet, ultrices varius risus. Nam vitae iaculis elit.    Aliquam mollis interdum libero. Sed sodales placerat egestas. Vestibulum ut    arcu aliquam purus viverra dictum vel sit amet mi. Duis nisl mauris, aliquam    sit amet luctus eget, dapibus in enim. Sed velit augue, pretium a sem    aliquam, congue porttitor tortor. Sed tempor nisl a lorem consequat, id  maximus erat aliquet. Sed sagittis porta libero sed condimentum. Aliquam  finibus lectus nec ante congue rutrum. Curabitur quam quam, accumsan id ultrices ultrices, tempor et tellus.
  </p>
  <p>
    Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam congue tortor  eget pulvinar lobortis. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Nam ac dolor augue. Pellentesque mi mi,
    laoreet et dolor sit amet, ultrices varius risus. Nam vitae iaculis elit.  Aliquam mollis interdum libero. Sed sodales placerat egestas. Vestibulum ut  arcu aliquam purus viverra dictum vel sit amet mi. Duis nisl mauris, aliquam  sit amet luctus eget, dapibus in enim. Sed velit augue, pretium a sem aliquam, congue porttitor tortor. Sed tempor nisl a lorem consequat, id maximus erat aliquet. Sed sagittis porta libero sed condimentum. Aliquam finibus lectus nec ante congue rutrum. Curabitur quam quam, accumsan id ultrices ultrices, tempor et tellus.
  </p>
  <div class="box" id="one">One</div>
</div>
```

```
* {
  box-sizing: border-box;
}
.box {
  width: 100px;
  height: 100px;
  background: red;
  color: white;
}
#one {
  position: fixed;
  top: 80px;
  left: 10px;
  background: blue;
}
.outer {
  width: 500px;
```

```
  height: 300px;
  overflow: scroll;
  padding-left: 150px;
}
```



Sticky:

Un elemento posizionato in modo sticky viene trattato come **posizionato relativamente finché non supera una soglia specificata, a quel punto viene considerato fisso finché non raggiunge il confine del suo genitore**. Per esempio:

```
#one {
  position: sticky;
  top: 10px;
}
```

La regola CSS precedente posizionerebbe l'elemento con id one relativamente fino a quando il viewport non viene fatto scorrere in modo tale che l'elemento si trovi a meno di 10 pixel dalla parte superiore. Oltre tale soglia, l'elemento viene fissato a 10 pixel dall'alto.

Un uso comune di questo tipo di posizionamento è per rendere le header dei siti fisse oppure per gestire le intestazioni di un elenco in ordine alfabetico. L'intestazione "B" verrà visualizzata appena sotto gli elementi che iniziano con "A" finché non verranno fatti scorrere fuori dallo schermo. Invece di scorrere fuori dallo schermo con il resto del contenuto, l'intestazione "B" rimarrà fissa nella parte superiore del viewport finché tutti gli elementi "B" non saranno usciti dallo schermo, a quel punto sarà coperta dalla "C" intestazione e così via.

È necessario specificare una soglia con almeno uno tra top, right, bottom o left affinché il posizionamento permanente si comporti come previsto. Altrimenti, sarà indistinguibile dal posizionamento relativo.

Esempio:

```
<dl>
  <div>
    <dt>A</dt>
    <dd>Andrew W.K.</dd>
    <dd>Apparat</dd>
    <dd>Arcade Fire</dd>
    <dd>At The Drive-In</dd>
    <dd>Aziz Ansari</dd>
  </div>
  <div>
    <dt>C</dt>
    <dd>Chromeo</dd>
    <dd>Common</dd>
    <dd>Converge</dd>
    <dd>Crystal Castles</dd>
    <dd>Cursive</dd>
  </div>
  <div>
    <dt>E</dt>
    <dd>Explosions In The Sky</dd>
  </div>
  <div>
    <dt>T</dt>
    <dd>Ted Leo &amp;amp; The Pharmacists</dd>
    <dd>T-Pain</dd>
    <dd>Thrice</dd>
    <dd>TV On The Radio</dd>
    <dd>Two Gallants</dd>
  </div>
</dl>
```

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

dl > div {
```

```
  background: #fff;
  padding: 24px 0 0 0;
}

dt {
  background: #b8c1c8;
  border-bottom: 1px solid #989ea4;
  border-top: 1px solid #717d85;
  color: #fff;
  font:
    bold 18px/21px Helvetica,
    Arial,
    sans-serif;
  margin: 0;
  padding: 2px 0 0 12px;
  position: -webkit-sticky;
  position: sticky;
  top: -1px;
}

dd {
  font:
    bold 20px/45px Helvetica,
    Arial,
    sans-serif;
  margin: 0;
  padding: 0 0 0 12px;
  white-space: nowrap;
}

dd + dd {
  border-top: 1px solid #ccc;
}
```

Per il risultato consultare https://developer.mozilla.org/en-US/docs/Web/CSS/position

La proprietà z-index

I posizionamenti (assoluti, relativi o fissi che siano) permettono di sistemare o traslare elementi lungo due dimensioni (verticale e orizzontale). C'è in realtà nei CSS una sorta di profondità, o **terza dimensione**: lo z-index, appunto. Stabilisce la disposizione degli

elementi posizionati lungo l'asse perpendicolare allo schermo. **Ha effetto solo sugli elementi con position diversa da static**.

Lo z-index assume valori interi senza unità di misura, che possono essere positivi o negativi. **Elementi con z-index maggiore vengono disposti sopra** (o ancor meglio, davanti) ad elementi con z-index minore. Il valore di default è 0.

Animazioni e trasformazioni

Per applicare effetti di animazione e/o trasformazione agli elementi html dobbiamo servirci di tre proprietà fondamentali:

- transition
- transform
- animation

Transition CSS

La proprietà transition consente di ottenere un passaggio graduale tra due stati nella stessa proprietà.

Viene principalmente utilizzata con le pseudo-classi come hover, active o focus.

Possiamo specificare 4 proprietà per la transizione da uno stato ad un altro :

transition-property: la proprietà sulla quale applicare la transizione;

transition-duration: la durata della transizione;

transition-timing-function: la curva temporale della transizione;

transition-delay: il ritardo con cui far iniziare la transizione;

Queste 4 proprietà sono raggruppabili in un’unica shorthand la proprietà “transition”

Esempio:

```
.button {
  background : red;
  border: solid 2px red;
  color: black;
  transition-property: all;
  transition-duration: 0.5s;
```

```
  transition-timing-function: linear;
}
.button:hover {
  background: orange;
  border: solid 2px orange;
  color: white;
}
```

In questo semplice esempio, vediamo come all'hover sul bottone il suo colore di background, di bordo e di testo cambierà in modo graduale secondo le caratteristiche da noi date.

Notiamo, inoltre, che anche quando il mouse lascerà il bottone le proprietà torneranno nello stato iniziale con una transizione; questo è dovuto al fatto che abbiamo specificato la transizione nello stato di default e non nella pseudo-classe.

Come dicevamo, esiste anche un altro metodo – più breve – per scrivere tutte queste proprietà, ovvero la proprietà transition.

Vediamo in questo esempio come si trasforma il codice.

```
.button {
  background: red;
  border: solid 2px red;
  color: black;
  transition: all 0.5s linear;
}
.button:hover {
  background: orange;
  border: solid 2px orange;
  color: white;
}
```

Nota la keyword "all" utilizzata nella proprietà transition-property: questa attribuisce una transizione a tutte le proprietà dell'elemento.

Se, ad esempio, sostituissimo la keyword "all" con "background" la transizione verrebbe applicata solo alla proprietà background, mentre le proprietà border e color cambierebbero immediatamente, dando quasi un effetto di "scatto" all'occhio.

Transform CSS

La proprietà transform ci permette di trasformare un elemento in 2D o in 3D.

Le trasformazioni che possiamo effettuare in 2D ci permettono di :

- traslare
- ruotare
- ridimensionare
- distorcere

Le trasformazioni che possiamo effettuare in 3D ci permettono di cambiare la prospettiva di un elemento.

Concentriamoci sullw trasformazioni che possiamo effettuare in 2D che sono quelle più utilizzate:

Traslazioni

```
.transform {
  transform: translate(x,y);
}
```

Con questa funzione possiamo spostare un elemento; accetta due parametri (x e y) che corrispondono allo spostamento sull'asse x – ovvero quello orizzontale – e sull'asse y – ovvero quello verticale -.

Abbiamo anche la possibilità di traslare solo sull'asse X o sull'asse Y con

translateY(y)

translateX(x)

Rotazioni

```
.transform {
  transform: rotate(angolo);
}
```

Con questa funzione possiamo far ruotare un elemento; accetta come parametro un angolo di rotazione – ad esempio "120deg" -, sia positivo che negativo.

Ridimensionamento

```
.transform {
  transform: scale(x,y);
}
```

Con questa funzione possiamo ridimensionare un elemento in scala; i valori da inserire saranno un numero intero o con la virgola a seconda di quanto vogliamo ridimensionare lungo l'asse orizzontale e lungo l'asse verticale.

Esempio:

```
.transform {
  transform: scale(1.5,2);
}
```

Nel caso in cui il ridimensionamento sia uguale sia per l'asse orizzontale che per il verticale possiamo inserire anche un unico valore.

Distorsione

```
.transform {
  transform: skew(angolo-x,angolo-y);
}
```

Con questa funzione possiamo distorcere la visualizzazione di un elemento; i valori da inserire saranno un angolo per la distorsione orizzontale e uno per quella verticale.

Per tutte le trasformazioni viste fino ad ora, possiamo aggiungere anche la proprietà "**transform-origin**" che indicherà il punto di "ancoraggio" ovvero il punto di origine della trasformazione. Di default questo punto coincide con l'angolo in alto a sinistra.

Animation

Le animazioni sono probabilmente lo strumento di progettazione di animazioni web più potente che CSS3 ha da offrire.

Le animazioni consentono di modificare gradualmente lo stile degli elementi della pagina, andando a definire più "keyframes" all'interno dell'animazione.

A differenza della transizione, la proprietà animation ci dà molto più controllo sulle fasi intermedie dell'animazione.

Possiamo creare un'animazione attraverso la regola @keyframes

I keyframes all'interno di questa regola determinano il tipo di stile applicato a un elemento in un dato momento, fondamentalmente rappresenta il "codice" che esegue l'animazione.

Vediamo come creare un'animazione con @keyframes

```
@keyframes bounce {
```

```
from {
    transform: translate(0, 0);
}
40% {
    transform: translate(0, -30px) scaleY(1.1);
}
70% {
    transform: translate(0, -15px) scaleY(1.05);
}
to {
    transform: translate(0, 0);
}}
```

In questo modo abbiamo creato un'animazione, che abbiamo chiamato "bounce", che simulerà il rimbalzare di una palla. Come possiamo notare, abbiamo creato 4 keyframes dove l'elemento verrà modificato.

Il primo, ovvero from, indica il punto di partenza del nostro elemento e possiamo sostituirlo anche con 0%.

L'ultimo, ovvero to, invece, indica il punto finale dell'animazione e possiamo anche sostituirlo con 100%.

Gli altri due keyframes ovvero 40% e 70% indicano invece degli stati intermedi della nostra animazione; questi non sono obbligatori, a differenza dello stato iniziale e finale dell'animazione.

Possiamo aggiungere quanti keyframes vogliamo alla nostra animazione e "complicarla" a nostro piacimento.

Una volta creata un'animazione con un nome, non ci resta che assegnarla ad un elemento.

```
.bounce-animated {
   animation-name : bounce;
   animation-duration: 1s;
}
```

Abbiamo moltissime proprietà che ci permettono di modificare un'animazione ma queste due – animation-name e animation-duration – sono obbligatorie per far partire la nostra animazione.

Vediamole, adesso, tutte :

animation: modalità abbreviata che serve ad impostare tutte le proprietà che seguono;

animation-name: (obbligatorio) indica il nome dell'animazione da applicare a un elemento;

animation-duration: (obbligatorio) specifica quanto dura l'animazione;

animation-delay: (opzionale) indica il ritardo in termini di tempo con cui inizia l'animazione ;

animation-direction: (opzionale) specifica la direzione dell'animazione; i possibili valori sono:

normal : da 0% a 100%

reverse: da 100% a 0%;

alternate: da 0% a 100% e da 100% a 0% in base a quante volte viene ripetuta l'animazione

alternate-reverse : da 100% a 0% e da 0% a 100% in base a quante volte viene ripetuta l'animazione

animation-iteration-count: (opzionale) specifica quante volte l'animazione deve essere ripetuta, i possibili valori sono:

infinite : l'animazione viene ripetuta all'infinito

numero : sarà il numero di volte in cui la nostra animazione verrà ripetuta.

animation-fill-mode: (opzionale) indica lo stile da applicare prima che l'animazione parta e una volta terminata l'animazione; i possibili valori sono:

forwards : una volta terminata l'elemento avrà lo stile dell'ultimo keyframe

backwards : una volta terminata l'elemento avrà lo stile del primo keyframe

both : quando l'animazione termina mantiene lo stile del kayframe nel quale è terminata

animation-timing-function: identifica la curva di velocità dell'animazione.

Media Query: gestire la responsività

Le media query sono una funzionalità di CSS3 che consente di applicare stili in modo condizionale in base a una serie di parametri del browser e del sistema operativo

La sintassi di una media query è la seguente

```
@media media type and (condition: breakpoint) {
    // regole CSS
}
```

Possiamo creare regole per diversi tipi di media con una varietà di condizioni. Nel momento in cui il tipo di media corrisponderà e la condizione sarà verificata le regole definite nella media query saranno applicate, altrimenti saranno ignorate.

Tipi di media

Se non applichiamo un tipo di media, la regola @media influenza tutti i tipi di dispositivi come default. Se invece vogliamo specificare o diversificare il comportamento in base ai tipi di media dobbiamo definirli dopo la proprietà @media. Ci sono molti tipi di dispositivi, ma possiamo raggrupparli in quattro categorie:

all—per tutti i tipi di media

print—per le stampanti

screen—per gli schermi di computer, tablet e smartphone

speech—per gli screen reader

Per esempio, quando voglio applicare la regola solo agli schermi, utilizzo la parola chiave screen dopo @media. Se poi, come quasi sempre accade, voglio specificare le dimensioni dello schermo devo concatenare la prima condizione (il fatto che si applichi solo agli schermi) con la seconda (dimensione dello schermo) utilizzando la parola **and;**

Esempio:

```
.text {
  font-size: 14px;          Regola definita nel css a livello generale
}
@media screen and (max-width: 600px) { //oppure @media screen and (width:<=600px) {
      .text {
        font-size: 16px;
      }
```

Regola valida solo nel caso di schermi con larghezza uguale o inferiore a 600px (sovrascrive la regola precedente quando le condizioni sono verificate. Ecco perché è importante **scrivere le media query dopo le regole css generali**

}

Breakpoints

I breakpoints sono larghezze personalizzabili che determinano il comportamento del layout responsivo in base alle dimensioni del dispositivo o dell'area visibile

Riprendendo la media query che abbiamo visto prima

@media screen and (max-width: **600px**)

@media screen and (width<= **600px**)

600 px rappresenta un breakpoint

Quali breakpoint utilizzare?

Non esistono breakpoint standard.

I più comuni e minimali possono essere:

- fino a **600** (o **576**) px -> per dispositivi extra small come smartphone
- fino a **768** px -> per tablet in modalità verticale
- fino a **992** px -> per tablet in modalità orizzontale e portatili
- **oltre 992** px -> per tutti gli altri dispositivi

Possiamo anche definire i breakpoint attribuendo loro nomi significativi e utilizzando le custom-media queries

```
/* Definizione dei breakpoint con nomi semantici */
    @custom-media --mobile (width < 768px);
    @custom-media --tablet (768px <= width < 1024px);
    @custom-media --desktop (width >= 1024px);

/* Utilizzo */
    @media (--mobile) {
      .header {
        padding: 1rem;
      }
    }
```

FINE MODULO

INTRODUZIONE
A HTML5 E CSS3