Cascading Style Sheets

Giuseppe Della Penna

Università degli Studi di L'Aquila
giuseppe.dellapenna@univaq.it
http://people.disim.univaq.it/dellapenna

> *Questo documento si basa sulle slide del corso di Web Engineering, riorganizzate per una migliore esperienza di lettura. Non è un libro di testo completo o un manuale tecnico, e deve essere utilizzato insieme a tutti gli altri materiali didattici del corso. Si prega di segnalare eventuali errori o omissioni all'autore.*

1. Introduzione ai CSS

1.1. La parte visuale del web

I Cascading Style Sheets (CSS) sono un linguaggio utilizzato per **definire l'aspetto visivo/percettivo (*presentazione*) dei documenti HTML** .

CSS descrive come gli elementi definiti da HTML debbano essere rappresentati su **schermo**, su **carta**, nel **parlato** o su altri supporti.

Su media visivi, come gli schermi o la carta, CSS può essere usato ad esempio per definire il tipo di carattere, il colore, le dimensioni e la spaziatura del contenuto, dividerlo in colonne o aggiungere animazioni e altre caratteristiche decorative.

Su media diversi, come il parlato, CSS può essere usato per definire il tono e la velocità della voce usata per pronunciare un contenuto.

A partire dalla versione 3, i CSS sono diventati molto più potenti e il loro sviluppo è continuo.

I CSS (o dei loro derivati) possono essere usati anche per definire aspetti visivi al di fuori del web, ad esempio per le interfacce grafiche delle applicazioni desktop.

1.2. Fogli di Stile nei documenti HTML

Un foglio di stile CSS è un documento di testo, costituito da una serie di *regole di stile*.

Un documento può avere anche più di un foglio di stile associato.

Un foglio di stile, in generale, può essere *incorporato* in un documento o *collegato* ad esso. In HTML:

- i fogli di stile CSS incorporati vanno inseriti nella sezione <head> all'interno di tag <style> aventi type="text/css".
- è anche possibile incorporare una regola di stile di stile inerente uno specifico elemento scrivendola direttamente nell'attributo style dello stesso.
- i fogli di stile CSS possono essere importati da una risorsa esterna inserendo nella sezione <head> dei tag <link> aventi type="text/css", rel="stylesheet" e href="uri_del_foglio_di_stile".

Fogli di Stile Multipli

Se si specificano più fogli di stile per lo stesso documento, CSS in generale li combina in ordine di inclusione.

Tuttavia, in HTML è possibile specificare l'inclusione *condizionale* di un foglio di stile (o un gruppo di fogli) in un documento in base a vari criteri:

- Il media di output
- Le preferenze dell'utente

Media Types

Quando si incorpora o collega un foglio di stile a un documento HTML, è possibile specificare nell'elemento <style> o <link> l'attributo *media*.

Il valore di media è una lista separata da virgole di nomi di media descriptors, che identificano il tipo di output per il quale il foglio di stile è adatto:

- `screen` (default)
- `tty`, `tv`, `projection`, `handheld`: indicano diversi tipi di terminali di tipo visivo
- `print`: il foglio di stile selezionato per la stampa (utile ad esempio per eliminare o combiare colori o elementi opzionali presenti nel rendering su schermo)
- `aural`, `braille`: rendering per sintetizzatori vocali o braille
- `all`: tutti i tipi di media

Fogli di Stile Alternativi

In HTML è possibile dotare di una pagina di tre diversi tipi di stile per lo stesso tipo di media:

- **Stile persistente**: sono sempre caricati dal browser. Gli stili incorporati nel documento sono sempre persistenti.

- **Stile preferito**: si tratta dello stile di default che verrà combinato a quello persistente, se presente. Si indica inserendo l'attributo `title="nome\_stile"` all'interno del tag <link>.
- **Stili alternativi**: sono stili che possono essere caricati alternativamente a quello preferito, a seconda delle preferenze dell'utente. Si indicano inserendo l'attributo `title="nome\_stile"` all'interno del tag <link> e modificandone l'attributo rel ad `alternate stylesheet`.

2. Regole e Cascading

2.1. Regole

Una regola CSS definisce uno **stile di formattazione** e una *classe di elementi* a cui deve essere applicato.

Lo stile di formattazione è a sua volta definito da una lista di proprietà valorizzate, con la sintassi **proprietà: valore**, poste all'interno di parentesi graffe e separate da un punto e virgola.

Le classi di elementi sono invece definite con speciali pattern detti **selettori**.

Un esempio di regola astratta è

```
SEL {P1: V1 [!important] P2: V2 P3: V3}
```

Il modificatore `!important` , opzionale, scritto dopo il valore (ma prima del separatore) di qualsiasi proprietà, serve ad aumentare la *priorità della regola* durante il processo di *cascading*, come vedremo più avanti.

Selettori Semplici

Un selettore semplice è un selettore di base seguito da zero o più *selettori di attributo*, *selettori di classe*, *selettori di ID*, *pseudo classi* e *pseudo elementi*.

I selettori di base sono due:

- Il **selettore universale** (`*`) fa match con ogni elemento.
- I **selettori di tipo** (stringhe rappresentanti nomi di elementi) fanno match con gli elementi aventi il nome corrispondente.

Selettori di Attributo

Ad ogni selettore di base possono essere posposti dei selettori di attributo. I possibili selettori di questo tipo sono:

- `[A]` (l'elemento deve avere un attributo A)
- `[A=V]` (l'elemento deve avere un attributo A con valore V)

- `[A~=V]` (l'elemento deve avere un attributo A il cui valore è una lista separata da spazi contenente V)
- `[A|=V]` (l'elemento deve avere un attributo A il cui valore è una lista separata da '-' contenente V)

Selettori di Classe

Quando si lavora con HTML, è disponibile una speciale sintassi abbreviata, detta selettore di classe, per lavorare con l'attributo *class* degli elementi

La sintassi `S.C`, applicabile a qualsiasi selettore semplice S, è equivalente a `S[class~=C]`.

Come caso speciale, è possibile scrivere il selettore di classe da solo (mentre in generale i selettori di attributo devono seguire un altro selettore valido), sottintendendo, prima di esso, il selettore universale: `.C` è equivalente a `*.C`, cioè `*[class~=C]`

Selettori di ID

In XML (e in HTML) è possibile assegnare ad ogni elemento un ID univoco.

Questo ID può essere usato nei CSS per applicare formattazione a un elemento specifico.

Il selettore ID può essere posto dopo ogni selettore di base ed ha la sintassi `S#ID`, dove S è un selettore semplice.

Il selettore `S#ID` fa match con l'elemento che corrisponde al selettore S ed ha l'ID specificato

È possibile (e molto comune) usare i selettori di ID da soli, proprio come quelli di classe, sottintendendo un selettore universale: `#ID` equivale a `*#ID`, cioè all'elemento (di qualunque tipo) avente l'ID specificato.

Pseudo classi

Le *pseudo classi* permettono di identificare elementi in base ad alcune loro particolari proprietà.

- `:first-child` (l'elemento è il *primo figlio* del suo genitore)
- `:link` (*link non visitato*)
- `:visited` (*link visitato*)
- `:hover` (elemento attualmente *indicato dall'utente*, ad esempio passandovi sopra il mouse)
- `:focus` (elemento attualmente *con focus*, cioè che accetta input da tastiera)
- `:active` (elemento correntemente *attivato*, ad esempio da un click del mouse)

Pseudo elementi

È infine possibile applicare formattazione ad elementi fittizi, non propriamente facenti parte del documento e/o delimitati da tag. Questi elementi si chiamano *pseudo elementi*.

- `:first-line` (la prima riga del blocco di testo contenuto nell'elemento)
- `:first-letter` (il primo carattere del blocco di testo contenuto nell'elemento)
- `:before` , `:after` (indicano le posizioni precedente e successiva all'elemento. Si usano un congiunzione con la proprietà CSS *content*)

Combinazioni di Selettori

Due selettori S e T possono essere combinati in un terzo selettore in vari modi:

- `S T` (spazio intermedio)
  Questo selettore fa match con gli elementi indicati da T solo se sono discendenti di un elemento che fa match con S
- `S > T`
  Questo selettore fa match con gli elementi indicati da T solo se sono figli di un elemento che fa match con S
- `S + T`
  Questo selettore fa match con gli elementi indicati da T solo se seguono immediatamente un elemento che fa match con S, e condividono con questo lo stesso genitore.
- `S, T`
  Questo selettore fa match con gli elementi indicati da S e da T (or logico o *raggruppamento di selettori*)

2.2. Calcolo dei Valori delle Proprietà di Stile

Durante il rendering, il processore CSS deve determinare lo stile da assegnare **a ciascun elemento del documento**.

- Questo implica calcolare il valore di **ogni singola proprietà** di stile che l'elemento può avere.

Per calcolare il valore di una specifica proprietà P per un elemento E, CSS procede come segue:

1. **Regole di cascading**
   La proprietà ha uno stile specificato tramite regole CSS?
2. **Ereditarietà**
   La proprietà può essere ereditata dall'elemento padre?
3. **Valore di default**
   La proprietà assume il valore dato dallo stylesheet di default.

2.3. Cascading

In un foglio di stile è possibile (e a volte molto utile) che esistano più regole che fanno match con gli stessi elementi.

Inoltre CSS prevede, oltre al foglio di stile dato dall'autore (cioè quello associato al documento), altri

due fogli di stile da considerare:

- Foglio di stile *utente*. L'utente può fornire delle regole di stile, come ad esempio la dimensione di base dei caratteri.
- Foglio di stile del *browser*. Ogni browser deve avere un suo foglio di stile di default.

Quando CSS calcola il valore di *ogni singola proprietà di stile*, prende in considerazione tutte le regole che fanno match con l'elemento da formattare in tutti i fogli di stile, e ne seleziona una sola con il metodo della **cascata**.

Regole di Selezione

Quando più regole fanno match con lo stesso elemento, l'effettivo valore di ciascuna proprietà viene selezionato tramite la seguente procedura.

1. **Priorità di Origine**
   1.1. Valore !important del foglio di stile utente
   1.2. Valore !important del foglio di stile autore
   1.3. Valore del foglio di stile autore
   1.4. Valore del foglio di stile utente
   1.5. Valore di default del foglio di stile del browser
2. **Specificità** *(per proprietà con la stessa priorità di origine)*
   - Un selettore più specifico per un determinato elemento ha la precedenza su uno più generico.
   - In HTML, le regole inserite nell'attributo *style* dell'elemento sono considerate con la massima specificità.
3. **Ordine** *(per proprietà con stessa priorità di origine e specificità)*
   - Una regola ha la precedenza su quelle che la *precedono*.

2.4. Ereditarietà

Molte proprietà (si veda la specifica) vengono automaticamente *ereditate* dai figli di un elemento, se non esistono regole specifiche che ne calcolano un valore diverso.

Questo comportamento di default è molto utile quando si creano fogli di stile complessi.

Ad esempio, se si specifica un font per i tag P, tutti i tag in esso contenuti (ad esempio B) avranno lo stesso font, a meno che non venga loro assegnato (globalmente o individualmente) uno stile che specifichi un font diverso.

È inoltre possibile forzare l'ereditarietà di una proprietà specificando (ove possibile) la parola chiave `inherit` come valore della proprietà stessa.

3. Proprietà CSS di Base

3.1. Elementi Base del Linguaggio CSS

Misure ed Unità di Misura

Le misure vengono espresse nel linguaggio CSS da numeri (anche decimali, e in alcuni casi negativi) seguiti immediatamente dal codice dell'unità di misura.

Inserire uno spazio tra il numero e l'unità di misura rende di solito illeggibile la proprietà!

La misura zero può essere specificata anche senza un codice di unità.

Esistono due classi di unità di misura: *relative* e *assolute*.

Le unità *assolute* sono `in` (pollici), `cm` (centimetri), `mm` (millimetri), `pt` (punti = 1/72 di pollice), `pc` (pica = 12 punti) e `px` (pixel del dispositivo: è una misura assoluta ma relativa al dispositivo).

- Sono utili solo quando il dispositivo di output è unico e precisamente definito.

Le unità *relative* sono `em` (font-size attuale), `ex` (x-height attuale). Nei browser più moderni, troviamo anche `rem` (font-size dell'elemento radice), `vw` (1% della larghezza della finestra del browser) , `vh` (1% dell'altezza della finestra del browser), `vmin` (1% della dimensione minore della finestra del browser), `vmax` (1% della dimensione maggiore della finestra del browser), `ch` (larghezza del carattere zero nel font attuale).

Sono molto indicate nel caso in cui si desideri far adattare automaticamente le dimensioni al dispositivo e alle sue impostazioni (es. stampante o browser con varie dimensioni di carattere)

In molti casi, le misure possono anche essere espresse come *percentuali*. La misura di riferimento è di solito quella della stessa proprietà nell'elemento contenitore.

Colori

Il colori possono essere definiti nel linguaggio CSS tramite la specifica numerica RGB o attraverso il loro nome proprio .

Le *specifica RGB* permette di definire qualsiasi colore attraverso il valore delle sue tre componenti *rossa*, *verde* e *blu*.

Le *stringhe RGB esadecimali* hanno la forma `#RRGGBB`, in cui ciascuna coppia di cifre rappresenta il valore esadecimale della rispettiva componente.

La forma abbreviata `#RGB` rappresenta il numero in cui ogni componente ha come valore quello della cifra corrispondente ripetuto per due volte.

Le *stringhe RGB decimali*, invece, si ottengono utilizzando il costrutto `rgb(R,G,B)` , dove R, G e B sono numeri compresi tra 0 e 255 o percentuali (intese come frazioni del valore massimo 255).

Infine, i colori per cui è definito un *nome proprio* sono `maroon` (#800000), `red` (#ff0000), `orange` (#ffA500), `yellow` (#ffff00), `olive` (#808000), `purple` (#800080), `fuchsia` (#ff00ff), `white` #ffffff, `lime` (#00ff00), `green` (#008000), `navy` (#000080), `blue` (#0000ff), `aqua` (#00ffff), `teal` (#008080), `black` (#000000), `silver` (#c0c0c0) e `gray` (#808080)

Shortand Properties

Il linguaggio CSS dispone di molte proprietà che spesso vengono impostate in gruppo, come ad esempio le tre proprietà che definiscono un bordo (colore, spessore, stile) o le proprietà di un font (famiglia, dimensione, peso, ...).

Per questo motivo, sono disponibili anche le cosiddette *proprietà shortand*, che permettono, con la loro particolare sintassi, di impostare con una sola operazione i valori di diverse proprietà.

Nell'impostare una proprietà shorthand, i valori di ogni singola proprietà "componente" vengono scritti di seguito, separati da uno spazio.

Se una o più proprietà vengono omesse nell'impostazione shorthand, il loro valore è considerato quello di default.

Ad esempio, la proprietà CSS font può essere usata per impostare contemporaneamente tutte le proprietà font-style,font-variant, font-weight, font-size, line-height e font-family.

3.2. Bordi

La maggior parte degli elementi possono essere dotati di un bordo sui quattro lati del loro box. Ogni bordo può possedere differenti caratteristiche (colore, spessore, tratteggio). È inoltre possibile, nel caso dei bordi delle tabelle, specificare come I bordi adiacenti debbano combinarsi.

Nota: I bordi delle tabelle specificati tramite CSS sono indipendenti da quelli mostrati tramite l'attributo `border` di <table>.

I colori dei bordi possono essere specificati tramite nomi simbolici (ad es. white), o tramite le loro componenti rgb in esadecimale (ad es. #FFFFFF) o decimale (ad es. rgb(255,255,255))

Lo spessore dei bordi è un valore che può essere specificato in una qualsiasi delle unità di misura accettate dai CSS (ad es. px, pt, mm, ...)

I principali stili per i bordi sono `dotted`, `dashed`, `solid`, `double`, `groove`, `ridge`, `inset`, `outset`. Tuttavia, la maggior parte dei browser supporta solo solid, dotted e dashed.

Per tutte le proprietà esistono degli shorthand che permettono di impostare gli stessi valori per tutti i lati contemporaneamente e/o di specificare le tre proprietà (colore, spessore, stile) con un'unica istruzione.

Proprietà CSS

- `border` (`border-top` , `border-right` , `border-bottom` , `border-left` **Valori**: { *spessore, stile, colore* }
- `border-color` (`border-top-color` , `border-right-color` , `border-bottom-color` , `border-left-color`)
  Valori: *colore* | `transparent`
- `border-style` (`border-top-style` , `border-right-style` , `border-bottom-style` , `border-left-style`)
  Valori: `none` * | *nome stile bordo* | `inherit`
- `border-width` (`border-top-width` , `border-right-width` , `border-bottom-width` , `border-left-width`)
  Valori: *misura*
- `border-collapse`
  Valori: `collapse` | `separate` *
  Elementi: tabelle ed elementi interni inline

Bordi smussati con i CSS3

Prima dell'avvento dei CSS3, per smussare gli angoli dei bordi era necessario usare trucchi complessi o addirittura immagini. Ora è possibile usare la proprietà `border-radius` (**prima è comunque necessario dotare l'elemento di un bordo con le proprietà appena viste**)

- `border-radius` (`border-top-left-radius` , `border-top-right-radius` , `border-bottom-right-radius` , `border-bottom-left-radius`)
  Valori: *raggio* | *percentuale* | `initial` | `inherit`

Initial corrisponde al valore di default, che in questo caso è zero.

Bordi con immagini con i CSS3

È anche possibile creare bordi usando immagini al posto delle normali line.

- `border-image` (`border-image-source` , `border-image-slice` , `border-image-width` , `border-image-outset` , `border-image-repeat`)
- `border-image-source`
  Valori: `none` * | *url immagine* | `initial` | `inherit`
- `border-image-slice`
  Valori: *numero*| *percentuale* | `fill` | `initial` | `inherit`
  L'immagine sorgente è divisa in nove parti: quattro angoli, quattro lati e il centro. Si possono specificare fino a quattro valori, quelli omessi sono uguali all'ultimo specificato. Questi valori determinano il modo in cui l'immagine verrà tagliata nelle nove parti, indicando una distanza dall'alto, destra, basso e sinistra dell'immagine.
- `border-image-width`
  Valori: *numero* | *percentuale* | `fill` | `initial` | `inherit`

Imposta la dimensione dell'imagine nel bordo (che può essere ridimensionata rispetto a quella prelevata dalla source). Si possono specificare fino a quattro valori.

- `border-image-outset`
  Valori: *numero* | *lunghezza* | `fill` | `initial` | `inherit`
  Specifica quanto l'imagine si estende oltre il contorno naturale dell'elemento (border box). Si possono specificare fino a quattro valori.
- `border-image-repeat`
  Valori: `stretch` * | `repeat` | `round` | `initial` | `inherit`
  Specifica se le immagini debbano essere ripetute, arrotondate o allungate per coprire l'intero bordo.

3.3. Sfondo

Tutti gli elementi di tipo *blocco* possono essere dotati di uno sfondo, costituito da un colore solido o da un'immagine

Nel caso di sfondi costituiti da immagini, il file da utilizzare deve essere indicato tramite il costrutto url('...') ed è possibile specificare:

- In che posizione far apparire l'immagine rispetto all'elemento di cui è sfondo.
- Se l'immagine dovrà essere ripetuta per riempire l'intera superficie messa a disposizione dall'elemento.
- Se l'immagine debba "scorrere" insieme al contenuto della finestra o rimanere fissa.

Grazie all'elevata versatilità degli sfondi, questi vengono spesso utilizzati per scopi impropri, ad esempio per creare effetti grafici (bottoni, elementi strutturali della pagina, ecc) che non sarebbero realizzabili importando semplicemente le immagini tramite il costrutto <img> di HTML.

Proprietà CSS

- `background-color` (colore di fondo)
  Valori: *colore* | `transparent`
- `background-attachment` (ancoraggio dello sfondo)
  Valori: `scroll` | `fixed` | `inherit`
- `background-image` (immagine di fondo)
  Valori: *url* | `none` | `inherit`
- `background-position` (posizione dell'immagine di fondo)
  Valori: `left top` | `left center` | `left bottom` | `right top` | `right center` | `right bottom` | `center top` | `center center` | `center bottom` | *x% y%* | *misura* | `inherit`
- `background-repeat` (estensione dell'immagine di fondo)
  Valori: `repeat` | `repeat-x` | `repeat-y` | `no-repeat` | `inherit

Ombreggiature con i CSS3

Oltre al colore di sfondo, con i CSS3 è possibile dotare ogni elemento di un'ombra.

- `box-shadow`
  Valori: *offset_orizzontale offset_verticale raggio_sfumatura* [*raggio_diffusione*] *colore*

Gli offset (anche negativi) indicano dove e di quanto l'ombra fuoriuscirà dall'elemento.

Il raggio di sfumatura determina la dimensione dell'area di sfumatura dell'ombra (numero di pixel oltre all'offset).

Il raggio di diffusione viene opzionalmente usato per aumentare o diminuire la dimensione dell'ombra (oltre a offset+sfumatura).

Per ombreggiare il testo, si usa la proprietà specifica text-shadow.

3.4. Formattazione

caratteri

- `color` (colore carattere)
  Valori: *colore*
- `font-family` (tipo di carattere)
  Valori: uno o più nomi di font, separati da virgole, in ordine di priorità
- `font-size` (dimensione carattere)
  Valori: *misura*
- `font-style` (corsivo)
  Valori: `normal` | `italic` | `oblique`
- `font-variant` (maiuscoletto)
  Valori: `normal` | `small-caps`
- `font-weight` (grassetto)
  Valori: `normal` | `bold` | `bolder` | `lighter`
- `text-decoration` (sottolineatura)
  Valori: `none` | `underline` | `overline` | `line-through`
- `text-transform` (maiuscole/minuscole)
  Valori: `capitalize` | `uppercase` | `lowercase` | `none`

paragrafi

- `line-height` (altezza delle righe)
  Valori: *misura* | `normal`
- `text-align` (allineamento orizzontale paragrafi)
  Valori: `left` | `right` | `center` | `justify`
- `vertical-align` (allineamento verticale paragrafi)
  Valori: `top` | `middle` | `bottom`

Elementi: celle di tabelle

- `text-indent` (rietro sinistro paragrafi)
  Valori: *misura*
- `word-spacing` (spazio tra parole)
  Valori: *misura* | `normal`
- `letter-spacing` (spazio tra lettere)
  Valori: *misura* | `normal`

3.5. Liste

Tramite CSS è possibile creare liste puntate e numerate di tipo semplice, con immagini standard o numeri (arabi, romani, ecc.) come punti elenco, usando l'attributo `list-style-type`.

La funzionalità più avanzata resa disponibile tramite l'attributo `list-style-image` prevede l'uso di immagini come punti elenco. In questo caso, l'attributo `list-style-type` viene ignorato.

Quando si creano liste di tipo personalizzato con immagini, bisogna sempre regolare i rientri, il padding e i margini degli elementi in modo da ottenere l'effetto visivo desiderato.

A seconda del browser, il margine e/o il padding dell'elemento list-item determinano il rientro dell'elemento stesso e/o lo spazio tra il punto elenco e il testo associato.

Gli attributi di tipo list possono essere applicati a tutti gli elementi per la cui proprietà *display* sia impostata al valore *list-item*.

Proprietà CSS

- `list-style-type` (tipi standard di punto elenco)
  Valori: `disc` | `circle` | `square` | `decimal` | `decimal-leading-zero` | `lower-roman` | `upper-roman` | `lower-greek` | `lower-latin` | `upper-latin` | `armenian` | `georgian` | `lower-alpha` | `upper-alpha` | `none`
- `list-style-image` (punto elenco immagine)
  Valori: *uri* | `none`
- `list-style-position` (posizione del punto elenco rispetto al testo)
  Valori: `inside` | `outside`*

3.6. Contenuto Dinamico

Prefissi, Suffissi, Virgolette e Contatori

- `quotes`
  Valori: `none` | (*stringa stringa*)+
  Indica le virgolette (aperta e chiusa) da usare per questo elemento, se necessario. È possibile specificare più coppie, una per ciascun livello di nidificazione delle virgolette stesse.

- `counter-reset`
  Valori: (*stringa* [*intero*])+
  Azzera (o imposta al numero dato) il valore dei contatori indicati, relativi all'elemento.
- `counter-increment`
  Valori: (*stringa* [*intero*])+
  Incrementa di uno (o del numero dato) il valore dei contatori indicati, relativi all'elemento.
- `content`
  Valori: `none` | (*stringa* | `counter(C,S)` | `open-quote` | `close-quote`)+
  Si applica ai soli pseudo elementi `:before` e `:after`, e specifica il testo che andrà inserito rispettivamente prima o dopo un elemento. I valori di `open-quote` e `close-quote` sono quelli impostati dall'attributo `quotes`. *C* può essere il nome di un qualunque contatore visibile dall'elemento. *S* (opzionale) è uno dei valori definiti per la proprietà `list-style-type`.

3.7. Cursori

L'uso della proprietà `cursor` permette di definire la forma che deve avere il mouse quando passa su un certo elemento della pagina.

Quando si creano delle interface ricche, con elementi interattivi (cliccabili, spostabili, modificabili) è importante dotarli del cursore giusto, per "suggerire" all'utente la modalità di interazione con l'element stesso.

- `cursor`
  Valori: [*uri*,] (`auto` | `crosshair` | `default` | `pointer` | `move` | `e-resize` | `ne-resize` | `nw-resize` | `n-resize` | `se-resize` | `sw-resize` | `s-resize` | `w-resize` | `text` | `wait` | `help` | `progress`)
  Imposta la forma del mouse quando è sopra l'elemento.

La lista di URI, opzionale, indica una o più risorse esterne da usare come cursore. Il browser utilizzerà la prima di cui riconosce il formato.

In ogni caso, è necessario fornire anche uno dei cursori standard, come unico valore o come ultima scelta nella lista.

4. CSS Box Model

4.1. Controllare la Generazione dei Box

È possibile specificare in che modo debba essere generato il box associato a un elemento.

`display`
Valori: `inline` | `block` | `grid` | `flex` | `inline-block` | ... | `none`

- **Block** genera un box di tipo blocco (che occupa uno spazio orizzontale e verticale separato dagli

altri box, come gli elementi p o div)
- **Inline** genera un box inline, che entra a far parte del flusso esterno senza interromperlo (come gli elementi b o span)
- **Flex** e **Grid** attivano le nuove modalità di posizionamento (*flexbox* e *grid*) che vedremo più avanti
- **Inline-block** genera un box inline, che però può essere dotato di larghezza e altezza tramite le rispettive proprietà CSS (`width` , `height`), proprio come i box di tipo blocco
- Ci sono poi molti altri valori che permettono al blocco di "comportarsi" come un certo elemento html, ad esempio un elemento di una lista (*list-item*), una tabella (*table*), ecc.
- **None** disattiva la generazione del box, rimuovendo l'elemento associato dal documento a tutti gli effetti.

4.2. Mostrare e Nascondere Elementi

Dopo aver generato il box relativo a un elemento, si può specificare se il contenuto del box debba essere o meno renderizzato.

`visibility`
Valori: `visible` * | `hidden`

- **Visible** (default) mostra l'elemento
- **Hidden** nasconde l'elemento.

Impostando la proprietà ad hidden, il box dell'elemento non viene rimosso dal flusso del documento, per cui il suo ingombro viene comunque considerato nel calcolo del layout.

4.3. Gestione del Contenuto

Generalmente il contenuto di un blocco è limitato alle dimensioni del blocco stesso. Il contenuto può però essere più grande del suo contenitore.

In questi casi, si può specificare come trattare la parte che fuoriesce dal contenitore.

`overflow`
Valori: `visible` * | `hidden` | `scroll` | `auto`

- `visible` (default) lascia che il contenuto extra sia renderizzato anche fuori dal contenitore.
- `hidden` nasconde la parte di contenuto che fuoriesce dal contenitore.
- `scroll` fa apparire delle barre di scorrimento all'interno del contenitore, in modo che il contenuto possa essere gestito tramite scrolling. Le barre appariranno in ogni caso, anche se il contenuto non fuoriesce.
- `auto` fa apparire delle barre di scorrimento all'interno del contenitore, in modo che il contenuto possa essere gestito tramite scrolling, solo se questo fuoriesce dal contenitore.

A partire dai CSS3, esistono anche le proprietà `overflow-x` e `overflow-y` , con gli stessi valori ma

associati alla gestione del solo overflow orizzontale o verticale.

Margini e Spazi

- `margin` (`margin-right`, `margin-left`, `margin-top`, `margin-bottom`)
  Valori: *misura* | `auto`
  Il margine è lo spazio vuoto lasciato tra il bordo del box di un oggetto e quello degli oggetti circostanti.
- `padding` (`padding-top`, `padding-right`, `padding-bottom`, `padding-left`)
  Valori: *misura*
  Il padding è lo spazio vuoto lasciato tra il bordo del box di un oggetto e il contenuto del box stesso.

In molti casi, margin e padding producono lo stesso effetto visivo, ma bisogna sempre usare l'attributo logicamente più adatto allo scopo.

Impostando i margini orizzontali (`margin-right`, `margin-left`) di un elemento al valore `auto`, il browser lo *centrerà* nel suo contenitore.

4.4. Dimensionamento

La dimensione di ogni elemento può essere impostata in vari modi, sovrascrivendo completamente o vincolando la dimensione naturale calcolata dal browser.

Nel primo caso, è possibile specificare misure o percentuali, che si intendono applicate alle corrispondenti dimensioni del contenitore.

Nel secondo caso, è possibile specificare dimensioni minime o massime, anch'esse espresse come misure o percentuali.

Proprietà di Dimensionamento

- `width`, `height`
  Valori: *misura* | `auto` * | `inherit`
  Impostano rispettivamente l'ampiezza e l'altezza dell'elemento. Il valore `auto` corrisponde alla dimensione naturale, calcolata attraverso le altre proprietà dell'elemento.
- `max-height`, `max-width`, `min-height`, `min-width`
  Valori: *misura* | `inherit`
  Impostano le dimensioni minime o massime dell'elemento.
- `box-sizing`
  Valori: `content-box` | `border-box`
  Determina come l'ampiezza e l'altezza dell'elemento sono effettivamente calcolate, prendendo o meno in considerazione il padding e lo spessore del bordo.
- Nella modalità `content-box` (default), i valori assegnati a `width` e `height` sono applicati al solo spazio interno disponibile per i contenuti, quindi il padding e il bordo incrementano tali

dimensioni, rendendo spesso a tutti gli effetti il box più grande di quanto impostato.
- Al contrario, in modalità `border-box` i valori assegnati a `width` e `height` determineranno sempre le effettive dimensioni finali del box: in altre parole, tali dimensioni saranno distribuite tra bordo, padding e spazio interno. *Solitamente questa modalità è quella più utile quando gli elementi devono essere posizionati in modo preciso tra loro*.

4.5. Posizionamento

Gli elementi possono essere posizionati in quattro modi diversi, specificabili con l'attributo CSS `position`:

- `static`: l'oggetto viene posto nella sua posizione naturale, data dal flusso del testo (default).
- `relative`: l'oggetto viene posizionato alle coordinate impostate tramite gli attributi `left`, `right`, `top` e `bottom` *relative alla sua posizione naturale*.
- `absolute`: l'oggetto viene posizionato alle coordinate impostate tramite gli attributi `left`, `right`, `top` e `bottom` *relative all'angolo superiore sinistro del suo più diretto contenitore con posizionamento non statico*.
- `fixed`: l'oggetto viene posizionato alle coordinate impostate tramite gli attributi `left`, `right`, `top` e `bottom` *relative all'angolo superiore sinistro del viewport*.

Gli elementi posizionati in maniera absolute o fixed si dicono **rimossi dal flusso del testo**. La loro posizione non dipende più dagli elementi che li circondano, anche se, nel caso absolute, continuano a scorrere insieme al resto della pagina.

Proprietà di Posizionamento

- `position`
  Valori: `static` | `relative` | `absolute` | `fixed`
  Determina il tipo di posizionamento dell'elemento
- `top`, `left`, `right`, `bottom`
  Valori: *misura*
  Determina la posizione dell'elemento, in base alle regole definite dal valore della proprietà `position`
- `z-index`
  Valori: *numero* | `auto` | `inherit`
  Determina la posizione dell'elemento (posizionato) sull'asse Z. Valori maggiori spostano l'elemento verso l'utente.

4.6. Floats

Tramite la tecnica di floating è possibile rimuovere degli elementi dal flusso del testo e posizionarli in maniera dinamica sul bordo sinistro o destro del loro contenitore.

Gli elementi posizionati tramite floating si distribuiscono sempre al meglio in relazione allo spazio a loro disposizione.

Il testo all'esterno degli elementi floating scorre intorno al loro margine in maniera automatica.

Questo tipo di effetto è spesso usato per creare menu, layout a colonne, ecc.

Proprietà per i Floats

- `float`
  Valori: `left` | `right` | `none`
  Imposta l'oggetto come floating sul lato sinistro o destro del contenitore. Il valore `none` disabilita il floating.
- `clear`
  Valori: `left` | `right` | `both`
  La proprietà `clear` impostata su un elemento fa sì che tutti I float del tipo specificato (sinistri, destri o entrambi) vengano disposti prima dell'elemento stesso.

5. CSS Flexbox

I flexbox sono una nuova modalità di disposizione degli oggetti, ormai stabilmente implementata in tutti i browser, e ampiamente utilizzata anche per la realizzazione di layout complessi.

I flexbox sono contenitori a **disposizione unidirezionale**, cioè gli elementi vengono posti uno dopo l'altro su un solo asse, anche se possono opzionalmente andare a capo. Per layout a griglia esistono i **CSS grids**, tuttavia *essendo questi ultimi ancora implementati in maniera instabile*, si preferiscono i flexbox per realizzare (con qualche accorgimento) anche layout a griglia.

Nella definizione di un layout flexbox dobbiamo prendere in considerazione due elementi:

- Il contenitore del layout, o *flex container*
- Gli elementi disposti nel layout, o *flex items*

5.1. Contaniers

Per rendere un elemento un flex container si utilizza la proprietà nota `display`, che va in questo caso impostata al valore `flex` (o `inline-flex`, se il contenitore deve essere di tipo inline).

Tutti gli elementi nidificati direttamente in un flex container sono invece considerati *flex items*

Per configurare come i flex items verranno disposti nel container possiamo avvalerci di varie proprietà:

- `flex-direction`
  definisce l'asse principale della disposizione. Le direzioni possibili sono `row` (da sinistra a destra),

`row-reverse` (da destra a sinistra), `column` (dall'alto in basso) e `column-reverse` (dal basso in alto).

- `flex-wrap`
  permette ai flex item di andare a capo, se non entrano tutti sull'asse principale. I valori sono `nowrap` (default), `wrap` (a capo dall'alto in basso in modalità row) e `wrap-reverse` (a capo dal basso in alto in modalità row).

Per configurare come i flex items si **allineeranno nel container** rispetto all'asse principale e quello secondario possiamo avvalerci delle seguenti proprietà:

`justify-content`
definisce l'allineamento `sull'asse principale`. I valori possibili sono:

- `flex-start`, `flex-end` : gli item sono disposti a partire dall'inizio/dalla fine dell'asse (ad es. da sinistra/destra nella direzione row).
- `start`, `end` : gli item sono disposti a partire dall'inizio/dalla fine della direzione di scrittura (ad es. da sinistra/destra in modalità RTL).
- `left`, `right` : gli item sono disposti a partire da sinistra/destra, se questo non è in conflitto con la flex-direction.
- `center` : gli item sono centrati sull'asse.
- `space-between` : gli item sono disposti sull'asse da un estremità all'altra con spaziatura uniforme.
- `space-around` : gli item sono disposti sull'asse in maniera che ognuno abbia la stessa quantità di spazio intorno a sé.
- `space-evenly` : gli item sono disposti sull'asse in maniera che lo spazio tra gli item e lo spazio tra gli item e i bordi del container siano uguali.

`align-items`
definisce come gli item sono disposti **sull'asse secondario *di ciascuna riga***. I valori possibili sono:

- `flex-start`, `flex-end` : gli item sono allineati all'inizio/fine dell'asse secondario di ciascuna riga (ad es. dall'alto/basso nella direzione row).
- `center` : gli item sono centrati sull'asse secondario di ciascuna riga.
- `stretch` : gli item sono allungati sull'asse secondario in modo che tutti gli item *di ogni riga* abbiano la stessa altezza
- `baseline` : gli item sono disposti in modo che lo loro linea di base sia allineata su ciascuna riga.

`align-content`
definisce l'allineamento ***di tutte e righe*** **sull'asse secondario**. I valori possibili sono gli stessi di `justify-content`. Ad esempio, questa proprietà permette di distribuire lo spazio tra le righe quando la `flex-direction` è `row` e gli elementi vanno a capo grazie a `flex-wrap`.

5.2. Items

Ogni flex item può essere configurato singolarmente, possibilmente sovrascrivendo alcune delle proprietà globali del container, tramite le seguenti proprietà:

- `order`
  definisce l'ordine degli item, che sono disposti nel container seguendo l'ordine numerico dato da questa proprietà (che può avere anche valori negativi). Se omessa, vale ovviamente l'ordinamento interno al container.
- `flex-grow`
  definisce la possibilità dell'item di espandersi sull'asse principale, se necessario. Il valore numerico indica la frazione dello spazio da occupare che verrà ceduta allo specifico item. Ad esempio, se tre elementi hanno grow uguale a 1, 2, 1, il 50% dello spazio (2/4) verrà ceduto al secondo item, e il 25% (1/4) agli altri due. *Per default, gli item non si espandono*.
- `flex-shrink`
  come flex-grow, ma gestisce il restringimento degli item nel caso lo spazio a disposizione sia insufficiente. Il numero in questo caso indica la frazione dello spazio necessario che verrà "sottratta" allo specifico item.
- `flex-basis`
  indica la dimensione base dell'item, prima che lo spazio eventualmente vuoto gli venga redistribuito. Può essere una misura anche relativa. *Il supporto a questa proprietà, in particolare in combinazione con altre proprietà come width, min-width, ecc. sono ancora poco avanzati*.
- `flex`
  è una shorthand per impostare `flex-grow`, `flex-shrink` e `flex-basis`.
- `align-self`
  permette di modificare la modalità di allineamento `align-items` del container per questo specifico elemento.

6. CSS Grids

I grids sono una funzionalità CSS avanzata che permette di **definire delle griglie** (righe per colonne) **e distribuirvi del contenuto**. Si tratta della "soluzione definitiva" al problema dei notissimi *layout a griglia*, attualmente realizzati con i floats o i flexbox usando dei trucchi.

Tuttavia, si tratta di una specifica ancora non ben implementata nei browser, e non è sicuro adottarla, ad esempio, per realizzare layout al posto della soluzione flexbox.

In una grid si configura per prima cosa un *container* suddividendone lo spazio interno con una *griglia virtuale* (che non corrisponde ad alcun elemento HTML, come avviene invece nel caso delle griglie realizzate con altre tecniche o con le tabelle)

Gli elementi interni al *grid container* (*grid items*) andranno ad occupare gli spazi della griglia in base alle direttive ad essi associate, o automaticamente. In quest'ultimo caso, *gli elementi si distribuiranno naturalmente, nell'ordine dato, cercando di occupare tutte le celle, riga per riga*.

Per rendere un elemento un grid container si utilizza la proprietà nota `display`, che va in questo caso impostata al valore `grid` (o `inline-grid`, se il contenitore deve essere di tipo inline).

6.1. Righe e colonne

La struttura della griglia interna al container può essere definita in vari modi.

È possibile definire **le righe e le colonne** specificandone la dimensione tramite le proprietà `grid-template-columns` e `grid-template-rows`.

Ciascuna proprietà ha come valore *una sequenza di misure* (che possono anche essere percentuali o la parola chiave *auto*), ad esempio le proprietà

```
grid-template-columns: auto 50px 20px 40px;
grid-template-rows: 25% 100px auto
```

definiscono una griglia con quattro colonne (di cui la prima prenderà tutto lo spazio che "avanza") e tre righe.

Se in un grid container si inseriscono più righe di quelle dichiarate, ne verranno create automaticamente di nuove. In tal caso, la loro dimensione potrà essere specificata tramite la proprietà `grid-auto-rows`.

È possibile **assegnare dei nomi alle estremità di ciascuna cella (bordo)** inserendoli tra parentesi quadre. *Attenzione! Non stiamo dando nomi alle celle, ma ai loro bordi!* Ad esempio

```
grid-template-columns: [c1 cstart] auto [c2 c1end] 50px [c3] 20px [c4] 40px [cend];
grid-template-rows: [r1 rstart] 25% [r2] 100px [r3] auto [rend]
```

definiscono la stessa griglia dell'esempio precedente, in cui l'estremità sinistra della prima colonna si chiama "c1" o "cinizio", la sua estremità destra (che è pure l'estremità sinistra della seconda colonna) si chiama "c2" o "c1fine" e così via.

Lo stesso discorso vale per le estremità superiori e inferiori, definite nella specifica delle righe. Non è necessario specificare nomi per tutti I bordi.

Nello specificare le dimensioni delle righe o colonne è possibile usare anche la speciale unità di misura `fr` (*fraction*), che indica una frazione dello spazio disponibile.

Per definire gli spazi tra le righe e le colonne si possono usare le proprietà `grid-column-gap` e `grid-row-gap`.

6.2. Aree

È inoltre possibile **dare dei nomi alle vere e proprie celle della griglia** (sempre dopo averle definite

con le due proprietà `grid-template-rows` e `grid-template-columns`), il che spesso risulta essere concettualmente più chiaro, usando la proprietà `grid-template-areas`. Ad esempio

```
grid-template-areas: "header header header header" "main main
. sidebar" "footer footer footer footer"
```

Il valore della proprietà è costituito da una sequenza di stringhe (tra virgolette), ognuna delle quali rappresenta una riga.

Ciascuna stringa contiene nomi di area/cella, separati da spazi. Un punto indica una cella senza nome (che probabilmente rimarrà vuota)

Ripetendo lo stesso nome, si crea un'area che si espande (*span*) su più celle.

Il numero di righe e colonne specificate deve corrispondere a quanto definito con `grid-template-rows` e `grid-template-columns`.

6.3. Template

È infine possibile (ma complesso) riassumere le tre proprietà appena viste in un'unica definizione usando la proprietà `grid-template`, ad esempio

```
grid-template: [r1 restart] "header header header header" 25% [r2] "main main . sidebar"
100px [r3] "footer footer footer footer" auto [rend] / [c1 cstart] auto [c2 c1end] 50px [c3] 2
```

La parte di specifica che precede lo slash (/) definisce le righe (dando eventuali nomi ai bordi superiore e inferiore) con la specifica delle relative aree e la loro dimensione.

La parte di specifica dopo lo slash definisce le colonne di tutte le righe esattamente come fatto con `grid-template-columns`.

6.4. Allineamento

Per configurare come il contenuto inserito nelle celle si allineerà rispetto ai bordi delle stesse possiamo avvalerci delle seguenti proprietà:

`justify-items`
definisce l'allineamento orizzontale del contenuto delle celle. I valori possibili sono:

- `start`, `end`: gli elementi sono disposti all'inizio/alla fine della rispettiva cella.
- `center`: gli elementi sono centrati sulla rispettiva cella.
- `stretch`: gli elementi sono allungati in modo da occupare tutta la larghezza della rispettiva cella (default).

`align-items`

definisce l'allineamento verticale del contenuto delle celle. I valori possibili sono gli stessi di `justify-items`.

Omettiamo altre proprietà più avanzate, in quanto poco supportate.

Ogni cella può avere valori propri per le proprietà di allineamento, che sovrascrivono quelle globali: ad esempio `justify-self` e `align-self`.

6.5. Items

È possibile specificare in quale cella (o celle) dovrà essere disposto ciascun item, invece che lasciare che questo sia disposto in base all'ordine naturale.

La prima modalità di disposizione fa riferimento ai bordi delle righe e delle colonne (definite da grid-template-rows e grid-template-columns) tramite le quattro proprietà `grid-column-start`, `grid-column-end`, `grid-row-start`, `grid-row-end`, ad esempio

```
<div style="grid-column-start: cstart; grid-column-end: cend">
 C1
</div>
<div style="grid-column-start: c2; grid-column-end: c4; grid-row-start: r2; grid-row-end: rend
 C2
</div>
<div style="grid-column-start: c4; grid-column-end: cend; grid-row-start: r2; grid-row-end: r3
 C3
</div>
<div style="grid-column-start: c4; grid-column-end: cend; grid-row-start: r3; grid-row-end: re
 C4
</div>
```

Con riferimento alla griglia definita nelle slides precedenti,

- C1 si estenderà in orizzontale dal bordo "cinizio" al bordo "cfine" e in verticale occuperà la prima riga (non essendo state specificate anche grid-row-start e grid-row-end, ma in questo caso il risultato potrebbe variare),
- C2 si estenderà in orizzontale dal bordo "c2" al bordo "c4" e in verticale dal bordo "r2" al bordo "rfine",
- C3 si estenderà in orizzontale dal bordo "c4" al bordo "cfine" e in verticale dal bordo "r2" al bordo "r3",
- C4 si estenderà in orizzontale dal bordo "c4" al bordo "cfine" e in verticale dal bordo "r3" al bordo "rfine".

Queste proprietà hanno anche una sintassi più complessa, che qui non viene approfondita.

La seconda modalità di disposizione degli item in una grid fa riferimento alle aree definite con `grid-template-areas` tramite la proprietà `grid-area`, ad esempio

```
<div style="grid-area: header"> C-h </div>
<div style="grid-area: main"> C-m </div>
<div style="grid-area: sidebar"> C-s </div>
<div style="grid-area: footer"> C-f </div>
```

Con riferimento alla griglia definita nelle slides precedenti,

- C-h occuperà le quattro celle della prima riga chiamate "header",
- C-m occuperà le due celle della seconda riga chiamate "main",
- C-s occuperà la cella all'estrema destra della seconda riga chiamata "sidebar",
- C-f occuperà le quattro celle della terza riga chiamate "footer".

Questa proprietà ha anche una sintassi più complessa, che qui non viene approfondita.

7. Responsive Design con i CSS

7.1. CSS Media Queries

Tra le caratteristiche introdotte da CSS3 che sono supportate ormai da tutti i browser moderni ci sono le *media queries* .

È possibile usare le media queries

- All'interno dell'attributo `media` del tag <link>, per importare un certo foglio di stile solo se la query è soddisfatta, oppure
- Direttamente nei fogli di stile racchiudendo insiemi di regole all'interno di parentesi graffe, precedute dalla *at-rule* `@media`

Una media query è composta da un *media type* (*screen, print* , ecc.) messa in AND con una serie di *media features* , il cui supporto può variare nei browser.

- `orientation` : [`portrait` | `landscape`]
- `width` : *misura*, `min-width` : *misura*, `max-width` : *misura*
- `prefers-color-scheme` : [`light` | `dark`]

È possibile mettere in OR più media queries separandole con una virgola, ad esempio

```
@media screen and (min-width: 300px), screen and (orientation: landscape) {...}
```

Questa media query è vera se stiamo visualizzando su schermo e la dimensione orizzontale è almeno 300 pixel oppure l'orientamento dello schermo è orizzontale.

7.2. Responsive Design

Il responsive design è una tecnica di progettazione dei layout resa famosa dall'articolo del 2010 di Ethan Marcotte su A List Apart: http://alistapart.com/article/responsive-web-design

Con questa tecnica il layout di un sito web **si adatta dinamicamente alle dimensioni e alle caratteristiche del dispositivo di output**.

I tre componenti di un responsive design sono

- i layout fluidi (tipicamente a griglia),
- le CSS3 media queries
- le immagini flessibili

Prima delle CSS media queries, un design responsive era ottenibile solo con l'ausilio di design fluidi supportati da script, mentre ora è possibile ottenere effetti molto più avanzati utilizzando i soli fogli di stile.

Applicando variazioni specifiche ai fogli di stile tramite le media queries, è possibile ad esempio nascondere elementi, riposizionarne altri, diminuire bordi e spaziature, ecc.

Breakpoints

I cosiddetti *responsive breakpoints* sono i valori limite di certe proprietà (tipicamente l'ampiezza del *viewport*) al di sotto e al di sopra dei quali intervengono le media queries a modificare la struttura visiva del sito. Ad esempio:

- Default (browser "normali")

Si usa un design liquido, con ampiezze percentuali (possibilmente anche per le immagini). Un design liquido rende il layout robusto anche per i dispositivi che non supportano ancora le media queries.

- Viewport da 768 a 959 pixel (tablet):

```
@media only screen and (min-width: 768px) and (max-width: 959px)
```

Si eliminano padding e spaziature "creative", si diminuisce leggermente la dimensione del carattere, ecc.

- Viewport più piccolo di 768 pixel (cellulari):

```
@media only screen and (max-width: 767px)
```

Si usa un "fixed", ad esempio di 320 pixel, oppure si inserisce una min-width sugli elementi principali

per evitare che si riducano troppo. Si eliminano le colonne linearizzandone il contenuto, si nascondono gli elementi secondari (parte di header e footer, ecc.), si mostrano menu più compatti...

7.3. Un Layout a Griglia con i Floats

Vedremo ora come realizzare un **layout liquido a griglia**, utilizzabile in tutti i casi si desideri un allineare elementi in righe e colonne, usando solo i *floats*.

Questo tipo di costruzione è alla base dei layout **responsive**, perché si adatta alle dimensioni del browser. È anche possibile **nidificare** questo tipo di griglie, per ottenere effetti molto complessi.

Sorgenti:

- La base di questo tipo di layout è il 960 grid system (http://960.gs/) che tuttavia è fixed (larghezza 960 pixel) e non usa media queries.
- La parte con media queries è tratta da Skeleton (http://www.getskeleton.com/), che tuttavia rimane fixed per ampiezze superiori a 960 pixel.
- Infine, l'ispirazione per la versione fluida della griglia è tratta da http://www.designinfluences.com/fluid960gs/.
- Si veda anche l'articolo introduttivo sui *fluid grids* di Ethan Marcotte: http://alistapart.com/article/fluidgrids.

Le righe

Il design permette di disporre un numero massimo di **16 colonne su ogni riga**.

Le celle di ciascuna riga hanno tra loro una **spaziatura** dell'1%.

Le classi *container* e *row* sono, rispettivamente, i contenitori dell'intero layout e delle singole righe:

```
.container { position: relative; width: 98%; padding: 0;}
```

Il contenitore per la tabella ha un piccolo margine extra. Non è necessario ed è possibile inserire direttamente le righe

```
.row { margin-bottom: 10px; }
```

Le righe hanno una leggera spaziatura, che può essere rimossa

Le colonne

La classe *column* definisce le proprietà comuni delle colonne:

```
.container .column,
.container .columns { float: left; display: inline; margin-left: 1%; margin-right: 1%; }
```

Le colonne *alpha* e *omega* (prima e ultima) non hanno spaziatura sinistra e destra:

```
.column.alpha,
.columns.alpha { margin-left: 0; }

.column.omega,
.columns.omega { margin-right: 0; }
```

Le classi *one*, *two*, ecc. permettono di definire l'ampiezza di ciascuna cella come numero di colonne occupate (1-16):

```
.container .one.column { width: 4.25%; }
.container .two.columns { width: 10.5%;}
```

Le dimensioni per le celle da tre a sedici colonne sono rispettivamente: 16.75%, 23%, 29.25%, 35.5%, 41.75%, 48%, 54.25%, 60.5%, 66.75%, 73%, 79.25%, 85.5%, 91.75%, 98%

Incapsulamento dei floats

Per concludere, qualche trucco per assicurarsi che le colonne float rimangano all'interno della propria riga, su ogni browser:

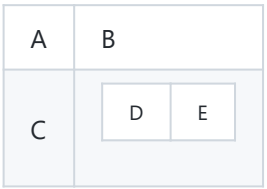
```
.row:before,
.row:after {
 content: '\0020';
 display: block;
 overflow: hidden;
 visibility: hidden;
 width: 0; height: 0;
}

.row:after { clear: both; }

.row{ zoom: 1; }
```

Esempio

```
<div class="row">
 <div class="three columns">
  A
 </div>
 <div class="thirteen columns">
  B
 </div>
</div>
<div class="row">
 <div class="three columns">
  C
 </div>
 <div class="thirteen columns">
  <div class="row">
   <div class="eight columns">
    D
   </div>
   <div class="eight columns">
    E
   </div>
  </div>
 </div>
</div>
```



Media queries

Ecco come il layout tabulare può *diventare lineare* al di sotto di una certa ampiezza. Disabilitando il *floating* e non costringendo l'ampiezza, le colonne "andranno a capo" ponendosi una sotto l'altra!

In più, in quest'esempio "blocchiamo" la dimensione dell'intero layout per impedire che scenda al di sotto di una certa soglia minima.

```
@media only screen and (max-width: 767px) {
.container {width: 300px;}
.container .columns, .container .column {margin: 0;}
.container .one.column,
.container .one.columns, ... {float: none; width: auto;}
}
```

7.4. Un Layout a Griglia con i Flexbox

Vedremo ora come realizzare un **layout liquido a griglia** utilizzando i flexbox.

Le righe

La classe *container* resta identica al caso dei floats:

```
.container { position: relative; width: 98%; padding: 0;}
```

La classe *row* attiva il flexbox positioning al suo interno:

```
.row {
 display: flex;
 flex: 1 0 auto;
 flex-direction: row;
 flex-wrap: wrap;
 margin-bottom: 10px;
}
```

La riga viene dichiarata come contenitore flex (`display: flex`) i cui figli diretti saranno allineati in orizzontale (`flex-direction`), ma andranno a capo se necessario (`flex-wrap`).

Inoltre, la riga viene dotata anche delle proprietà di un flex child (vedi dopo), in modo da poter essere nidificata nelle celle.

Le colonne

La classe *column* definisce le proprietà comuni delle colonne:

```
.container.column,
.container.columns {
 display: flex;
 flex: 1 0 auto;
 flex-direction: column;
 max-width: 100%;
 margin-left: 1%;
 margin-right: 1%;
}
```

La proprietà `flex` (shortcut per `flex-grow` , `flex-shrink` e `flex-basis`) regola il comportamento degli elementi flex nel contenitore.

`flex-grow` (qui 1) indica la proporzione con cui l'elemento crescerà rispetto agli altri, mentre `flex-shrink` indica la proporzione con cui si restringerà (qui 0 indica che l'elemento non si restringerà oltre la sua dimensione base).

`flex-basis` (`auto`) imposta la dimensione di base sulla base della quale verrà distribuito lo spazio rimanente.

Le proprietà `display` e `flex-direction` servono a configurare le celle anche come contenitori per altri gruppi flex (le righe, da cui `flex-direction` impostato a `column`)

- Le classi *one*, *two*, ecc. permettono di definire l'ampiezza di ciascuna cella come numero di colonne occupate (1-16) e sovrascrivono `flex-basis`:

```
.container.one.column { flex-basis: 4.25%; max-width: 4.25%;}
```

Le dimensioni per le celle da tre a sedici colonne sono rispettivamente: 16.75%, 23%, 29.25%, 35.5%, 41.75%, 48%, 54.25%, 60.5%, 66.75%, 73%, 79.25%, 85.5%, 91.75%, 98%

Media queries

Le media queries sono adattate di conseguenza, disabilitando il flex quando occorre:

```
@media only screen and (max-width: 767px) {
.container { width: 300px; }
.container .columns, .container .column { margin: 0; flex: none }
.container .one.column,… { flex: none; display: block; width: auto; max-width: 100%;}
.row { flex: none; display: block; width: 100% }
}
```

7.5. Compatibilità Cross-Browser

Stylesheet di Reset

Molti browser applicano alle proprietà dei vari elementi **default diversi**, corrispondenti al loro *stylesheet di default* .

Per realizzare CSS crossbrowser, può essere utile **azzerare queste differenze** e creare i fogli di stile a partire da una **base minima comune**. Questo si può ottenere inserendo all'inizio del foglio di stile una specifica come la seguente. (http://meyerweb.com/eric/tools/css/reset/)

```
html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code, del,
dfn, em, img, ins, kbd, q, s, samp, small, strike,
strong, sub, sup, tt, var, b, u, i, center, dl, dt, dd,
ol, ul, li, fieldset, form, label, legend, table,
caption, tbody, tfoot, thead, tr, th, td, article,
aside, canvas, details, embed, figure, figcaption,
footer, header, hgroup, menu, nav, output, ruby,
section, summary, time, mark, audio, video
{ margin: 0;
  padding: 0;
  border: 0;
  font-size: 100%;
  font: inherit;
  vertical-align: baseline;
}

article, aside, details,
figcaption, figure, footer,
header, hgroup, menu, nav, section /* HTML5 */
{ display: block; }

body
{ line-height: 1; }

ol, ul
{ list-style: none; }

blockquote, q
{ quotes: none; }

blockquote:before,
blockquote:after,
q:before, q:after
{ content: ''; content: none; }

table
{ border-collapse: collapse; border-spacing: 0; }
```

8. Riferimenti

Cascading Style Sheets (CSS)
https://www.w3.org/Style/CSS

9. Esempi

Di seguito trovate un elenco dei principali esempi mostrati o sviluppati durante le lezioni. Questi esempi sono tutti disponibili su GitHub, all'indirizzo [https://github.com/orgs/WebEngineering-Univaq], e sono *parte integrante* delle lezioni stesse, in quanto mostrano l'effettivo uso delle nozioni illustrate in aula e riportate su questa documentazione (dove, quando possibile, troverete dei riferimenti a questi esempi).

La lista che segue può non essere sempre aggiornata: nel repository potrete spesso trovare utili nuovi esempi appena sviluppati.

Esempi di base

- css_inclusion_print.html
  Mostra come inserire un foglio di stile specifico per la stampa in un documento HTML
- css_selectors.html
  Mostra vari esempi d'uso e combinazione dei selettori CSS
- css_cascade.html
  Esempi ed esercizi sul cascading nei CSS
- css_properties_common.html
  Mostra l'uso di varie proprietà base dei CSS come `font`, `color`, `background`, `border`
- css_3_webfonts.html
  Mostra l'uso della at-rule CSS `@font-face`
- css_3_round_borders.html
  Mostra l'uso della proprietà CSS `border-radius`
- css_3_imageborders.html
  Mostra l'uso della proprietà CSS `border-image`
- css_3_shadows.html
  Mostra l'uso delle proprietà CSS `box-shadow` *e* `text-shadow`
- css_menu_multilev_static.html
  Mostra come trasformare una semplice lista in un menu usando la proprietà `list-style` *insieme ad altre proprietà base dei CSS*
- css_properties_misc.html
  Mostra l'uso di varie proprietà avanzate dei CSS come `content`, `quotes`, `counter-increment`, `cursor`

Box model e posizionamento

- css_properties_display.html
  Mostra l'uso della proprietà CSS `display`
- css_properties_overflow.html
  Mostra l'uso della proprietà CSS `overflow`
- css_properties_positioning_relative.html
  Mostra come usare il posizionamento relativo (`position: relative` *)*

- css_properties_positioning_absolute.html
  Mostra come usare il posizionamento assoluto (`position: absolute`)
- css_properties_positioning_fixed.html
  Mostra come usare il posizionamento fosso (`position: fixed`)
- css_properties_positioning_float.html
  Mostra l'uso delle proprietà CSS `float` e `clear`
- css_menu_multilev_flyout.html
  Mostra come trasformare una semplice lista in un menu flyout usando le proprietà `list-style`, `display` e `position` insieme ad altre proprietà base dei CSS

Flexbox e Grid

- css_3_flexbox.html
  Mostra le varie proprietà relative ai flexbox
- css_3_grid.html
  Mostra le varie proprietà relative alle grid

Argomenti avanzati

- css_3_transitions.html
  Mostra l'uso della proprietà CSS `transition`
- css_3_animation.html
  Mostra l'uso della proprietà CSS `animation` (e della at-rule `@keyframes`)
- css_sprites.html
  Mostra come usare la tecnica dei CSS sprites per scaricare più immagini in modo efficiente
- css_3_variables.html
  Mostra l'uso delle variabili css e dei costrutti `var()` e `calc()`

Title: Cascading Style Sheets
Author: Giuseppe Della Penna, *University of L'Aquila*
Version: 20240220