CSS – Layout

Angelo Di Iorio

Università di Bologna

Layout

- Il termine **layout**, o meglio **page layout**, indica la disposizione degli oggetti in modo armonioso nella pagina
- Richiede quindi di organizzare gli oggetti e in particolare decidere le loro *dimensioni* e la loro *posizione*, in relazione agli altri oggetti e alle aree vuote
- CSS definisce diverse proprietà per controllare in modo sofisiticato il layout
- Se il contenuto richiede più spazio di quello disponibile:
  - *resizing*: si cambiano le dimensioni degli oggetti
  - *pagination*: i contenuti sono divisi su più pagine
  - *scrolling*: solo una parte dei contenuti è visibile e si fornisce all'utente un meccanismo per far scorrere i contenuti visibili

Responsive Layout

- Un **layout responsive** è ottimizzato per essere letto su tutti i dispositivi
- Obiettivo: minimizzare *resizing* e *scrolling* da parte dell'utente e facilitare l'esperienza di lettura
- La costruzione di layout responsive non richiede nuove tecnologie (e non è di per sé una tecnologia) ma è realizzata direttamente in CSS, specificando il modo in cui il layout si adatta al dispositivo di lettura
- Partiamo dai meccanismi CSS per progettare un layout, per poi arrivare alla *responsiveness*

Viewport (1)

- Il canvas è lo spazio (virtualmente infinito) di posizionamento degli elementi via CSS
- Il viewport è la parte del canvas attualmente visibile sullo schermo; è possibile posizionare elementi sul canvas anche se non visibili attraverso il viewport
- Ovviamente il viewport dipende dalla dimensione dello schermo
- I dispositivi di dimensioni ridotte usano un *viewport virtuale* per calcolare le dimensioni degli oggetti nel layout e successivamente le scalano per visualizzarle nel viewport reale
- Questa tecnica è molto utile per layout che non sono progettati come mobile-first

Viewport (2)

- Se un layout è progettato per dispositivi più piccoli e per essere modificato in base alle dimensioni del dispositivo (tramite media queries, dettagli nelle prossime slide) il viewport virtuale non funziona bene
- Partendo da una proposta di Apple, è stato introdotto l'uso di un elemento META di HTML per controllare il viewport:
- Specificando la proprietà "viewport" si possono decidere le dimensioni del viewport e forzare il livello di zoom iniziale:

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

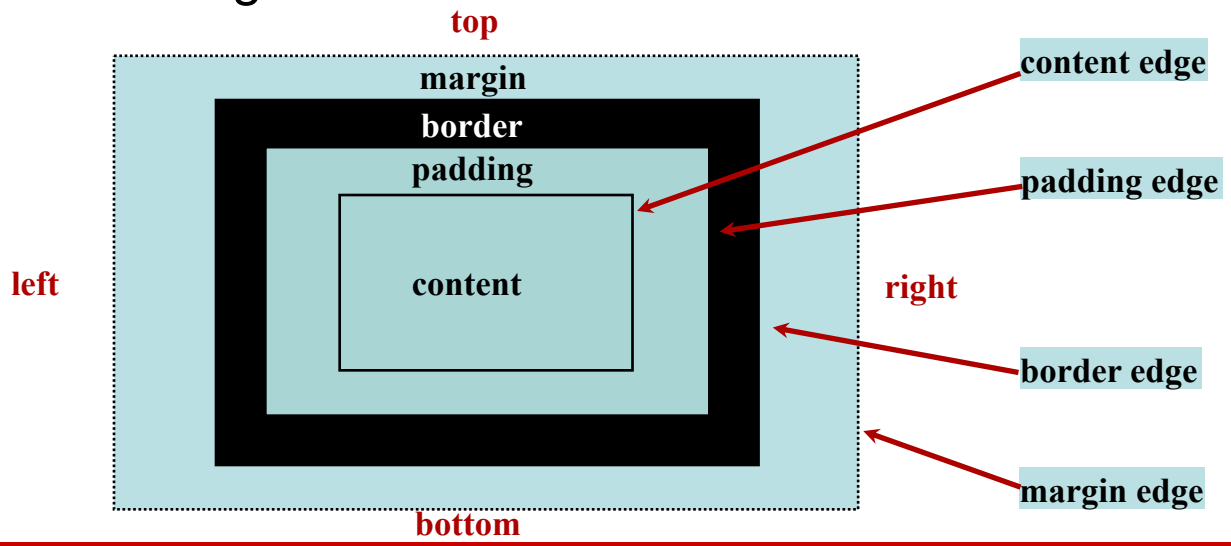
- **Suggerimento**: aggiungete sempre questa dichiarazione alle pagine e usate `initial-scale=1.0`

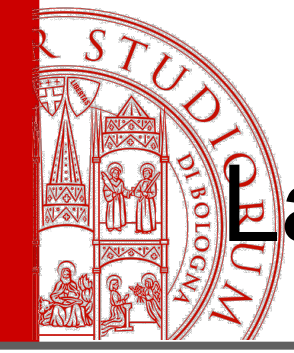
Unità di misura basate su viewport

- La più piccola unità indirizzabile sullo schermo è il *pixel (px)*
- Le dimensioni dei pixel dipendono fortemente dalla dimensione e dalla risoluzione degli schermi e sono molto diversi tra device e device e non sono un'unità affidabile.
- L'elemento `<meta name="viewport"...` è usato per controllare questo fattore di scala
- Piuttosto che usare misure in pixel (o in percentuale rispetto al contenitore, poi tradotta in pixel dal browser) si possono usare unità di misura relative al viewport:
  - **`Viewport width (vw)`**: una percentuale (1%) della larghezza attuale del viewport. Ad esempio, in un viewport largo 500 pixel, 3vw = 15px
  - **`Viewport height (vh)`**: una percentuale (1%) della altezza attuale del viewport.

Recap: box CSS

- **Margine**: la regione che separa una scatola dall'altra, sempre trasparente (ovvero ha il colore di sfondo della scatola contenitore)
- **Bordo**: la regione ove visualizzare un bordo per la scatola
- **Padding**: la regione di respiro tra il bordo della scatola ed il contenuto. Ha sempre il colore di sfondo del contenuto
- **Contenuto**: la regione dove sta il contenuto dell'elemento





Larghezza e altezza di una box

- Con le proprietà `width` e `height` si fissano larghezza e altezza delle box CSS, esprimibili con dimensioni assolute o relative
  - **Non usare dimensioni fisse!**
- Con `max-width` e `max-height` si fissano la larghezza ed altezza massime, `min-width` e `min-height` per i valori minimi
- Di default larghezza e altezza si applicano alla content-box, escludendo padding e bordi
- La proprietà **`box-sizing`** permette cambiare strategia:
  - **`border-box`**: calcola le misure sull'intera box, compresa di padding e bordo. Molto più comodo per costruire layout
  - **`content-box`** (default)

Flussi inline, blocco e float

- I flussi CSS indicano indicano il modo in cui una box è posta rispetto alle altre
- Esistono diversi flussi, tra cui quelli base sono:
  - flusso **blocco**: le scatole sono poste l'una sopra l'altra in successione verticale (come paragrafi)
  - flusso **inline**: le scatole sono poste l'una accanto all'altra in successione orizzontale (come parole della stessa riga)
  - flusso **float**: le scatole sono poste all'interno del contenitore e poi spostate all'estrema sinistra o destra della scatola, lasciando ruotare le altre intorno
- Ogni elemento HTML ha un flusso di default, che può essere sovrascritto tramite la proprietà **`display`**

Posizionamento (1)

- La posizione dipende dalle altre scatole, dallo sfondo (canvas) o dalla finestra (viewport).
- Si usa la proprietà **`position`**:
  - *Posizionamento* ***static*** (default): la scatola viene posta nella posizione di flusso normale che avrebbe naturalmente.
  - *Posizionamento* ***absolute***: la scatola viene posta nella posizione specificata indipendentemente dal flusso (e nasconde ciò che sta sotto).
    - Nota: la posizione è calcolata rispetto al primo elemento contenitore (padre o ancestor) che ha posizione assoluta o relativa
  - *Posizionamento* ***fixed***: la scatola viene posta in una posizione assoluta rispetto alla finestra (*viewport*), senza scrollare mai

Posizionamento (2)

- Altri valori della proprietà **`position`**:
  - *Posizionamento **relative***: la scatola viene spostato di un delta dalla sua posizione naturale.
    - Gli elementi figlio possono essere posizionati in modo assoluto. Si usa quindi anche senza "spostare" l'elemento ma solo per aver un contenitore per posizionare gli elementi figlio
    - Utile anche per poter usare *z-index* (vedi prossime slide) che non si applica ad elementi static
  - *Posizionamento **sticky***: la scatola viene posta nella sua posizione naturale all'interno del suo contenitore, ma durante lo scrolling sta fissa rispetto al viewport fino a che il contenitore non esce dalla vista.

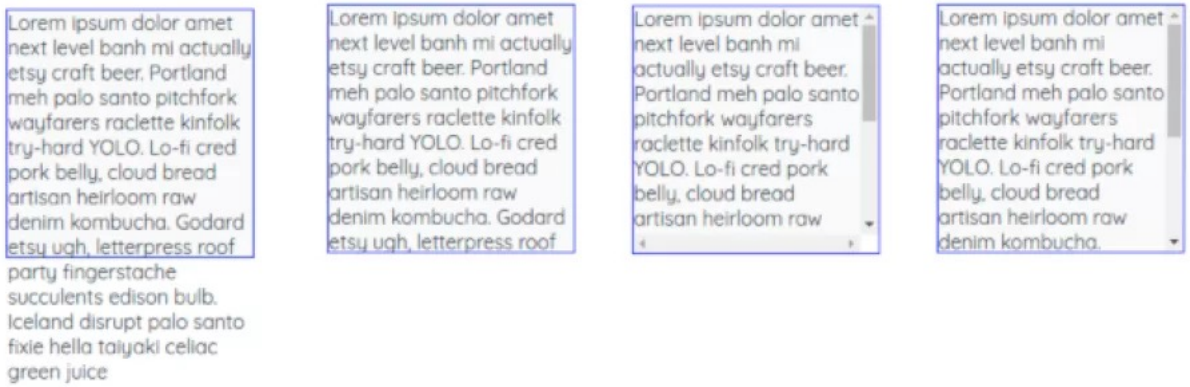
Posizionamento (3)

- Definito il tipo di posizione, si possono controllare coordinate e dimensioni con altre proprietà (fino a 4):
  - **top**, **bottom**, **left**, **right**: coordinate della scatola (rispetto al contenitore o elemento "ancestor" o viewport)
  - **width**, **height**: dimensioni usabili invece di *right* e *bottom*

- Se si indicano 4 proprietà, la scatola verrà creata delle dimensioni indicate, ma il contenuto in eccesso può uscirne (lo si gestisce con la proprietà overflow).
- ***Suggerimento****: se non sapete in anticipo quanto contenuto avrete, specificate al massimo 3 proprietà dimensionali. Gestire correttamente l'overflow di una scatola troppo piena richiede molti sforzi.*

Posizionamento: overflow

- La proprietà **`overflow`** specifica come trattare il contenuto che non sta interamente nella scatola (forse con dimensioni troppo rigide). Valori possibili:
  - ***visible***: la scatola si espande per contenere tutto il contenuto
  - ***hidden***: il contenuto extra viene nascosto
  - ***scroll***: compare una scrollbar per l'accesso al contenuto extra.
- Si possono specificare le proprietà `overflow-x` and `overflow-y` per permettere la comparsa di una sola delle scroll



Visibilità degli elementi

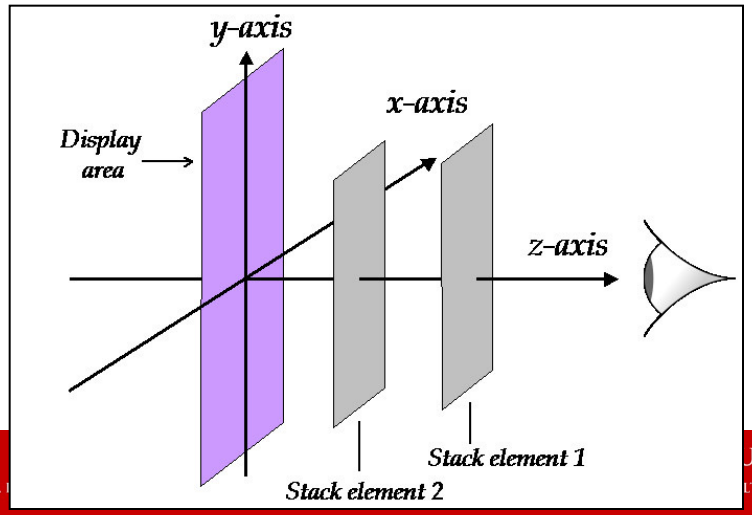
- Due modi per nascondere o rendere visibili gli elementi:
  - `visiblity: visible`; (oppure `hidden` o `collapse`); l'elemento è aggiunto al DOM ma non visibile
  - `display: none` (oppure altri valori per flussi diversi); l'elemento non è aggiunto al DOM e lo spazio che avrebbe occupato può essere occupato da altre box
- Sono utili per creare, ad esempio, menù a scomparsa o tooltip e per questo sono spesso usati in combinazione con il selettore `E:hover`

```
<span class="property">
  visibility
  <span
class="explanation">
  Proprietà CSS che …
  </span>
</span>
```

```
span.property {position: relative;}
span. explanation {
        visibility:hidden;
        position: absolute;
        left: 20px;
        top: 20px;
        z-index: 1;}
span.property:hover span. explanation {
                visibility: visible;}
```

Overlap e z-index

- Le box possono sovrapporsi, ad esempio in caso di `position:relative` o `absolute` (e dropdown o tooltip)
- La proprietà `z-index` permette di indicare la posizione di una box nella pila di box sovrapposte
- Più è alto il valore, più l'elemento è in primo piano
- Per default il background delle box è trasparente. Per coprire i contenuti sottostanti deve essere quindi indicato un colore o un'immagine



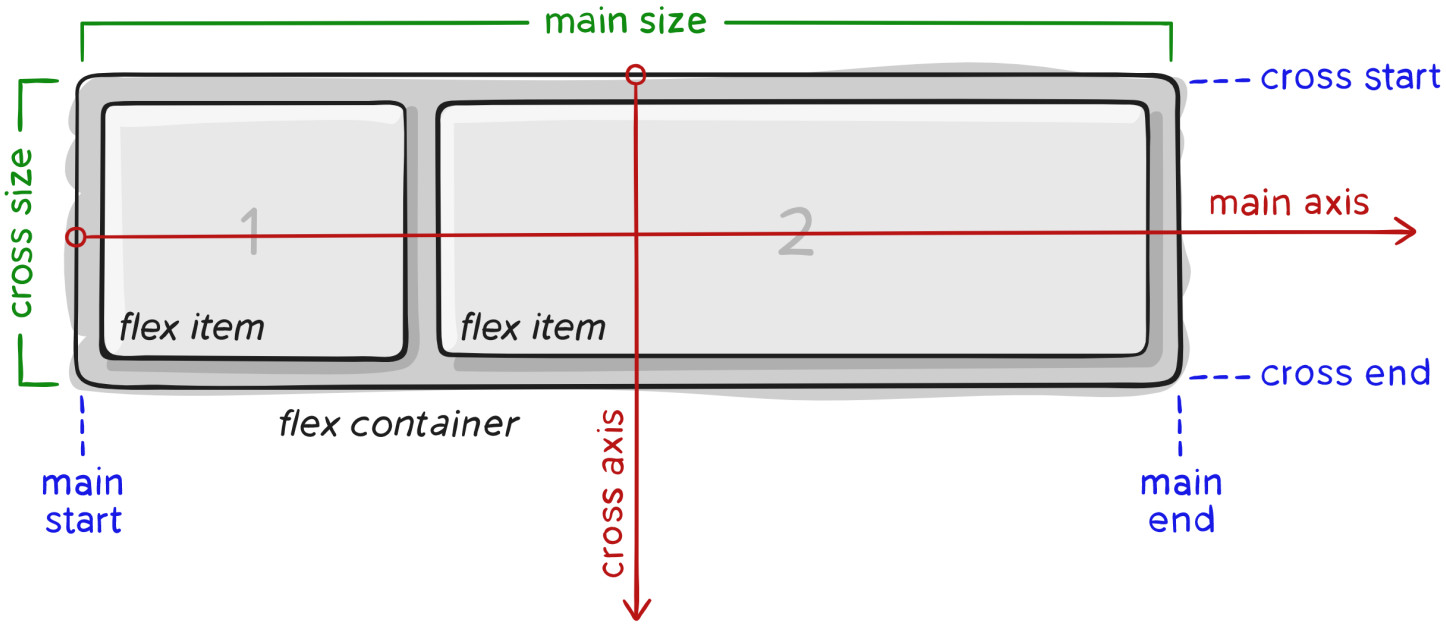
Proprietà `float`

- La proprietà float indica che una box deve essere spostata a sinistra o a destra all'interno del contenitore e le altre box ruotarle intorno
  - NOTA: non `display:float;` ma una proprietà diversa, che può assumere valori `left, right` o `none`
- Usata in combinazione con la proprietà `clear (left | right | both | none)` che indica il lato della box su cui non è possibile posizionare altri elementi di tipo float, che quindi seguiranno il normale flusso
- E' stato lo strumento principale per costruire layout liquidi
  - in combinazione con la proprietà **`width`** per indicare larghezze percentuali delle box

CSS Flexbox

- Il modulo CSS Flexbox è uno strumento estremamente flessibile per costruire layout fluidi senza usare float
- Introduce un nuovo valore per la proprietà *display*, appunto `dispay:flexbox`. Gli elementi si dispongono per usare al meglio lo spazio interno di un FlexBox (scatola flessibile)
- Due componenti: un **FlexBox container** all'interno del quale sono posizionati i **FlexBox item**
- Le proprietà di container e item sono usate per decidere:
  - direzione in cui disporre gli item
  - possibilità di disporre gli item su più linee
  - allineamento, spaziatura e ordine degli item

CSS Flexbox: terminologia



- Una guida molto chiara (e colorata!) di tutte le proprietà: https://css-tricks.com/snippets/css/a-guide-to-flexbox/

Alcune proprietà di Flex Container

flex-direction: definisce l'asse principale

- Valori: `row, column, row-reverse, column-reverse`

flex-wrap: elementi su una a più linee

- Valori: `wrap, nowrap, …`

justify-content: allineamento sull'asse principale

- Valori: `flex-start, flex-end, center, space-between, …`

align-items: allineamento sull'asse secondario

- Valori: `start, end, center, …`

gap: spazio tra i flex-items

Alcune proprietà di Flex Item

order: per modificare l'ordine degli elementi

```
order: 1      order: 3      order: -1
```

flex-grow, **flex-shrink** and **flex-basis** controllano come distribuire lo spazio rimanente dopo aver posizionato gli elementi:

- capacità di aumentare le dimensioni lungo l'asse principale
- capacità di ridurre le dimensioni
- dimensione di default prima di ridistribuire lo spazio

flex: si usa spesso come shorthand per le precedenti

```
flex: 2 1 100px      flex: 0 5      flex: 3
```

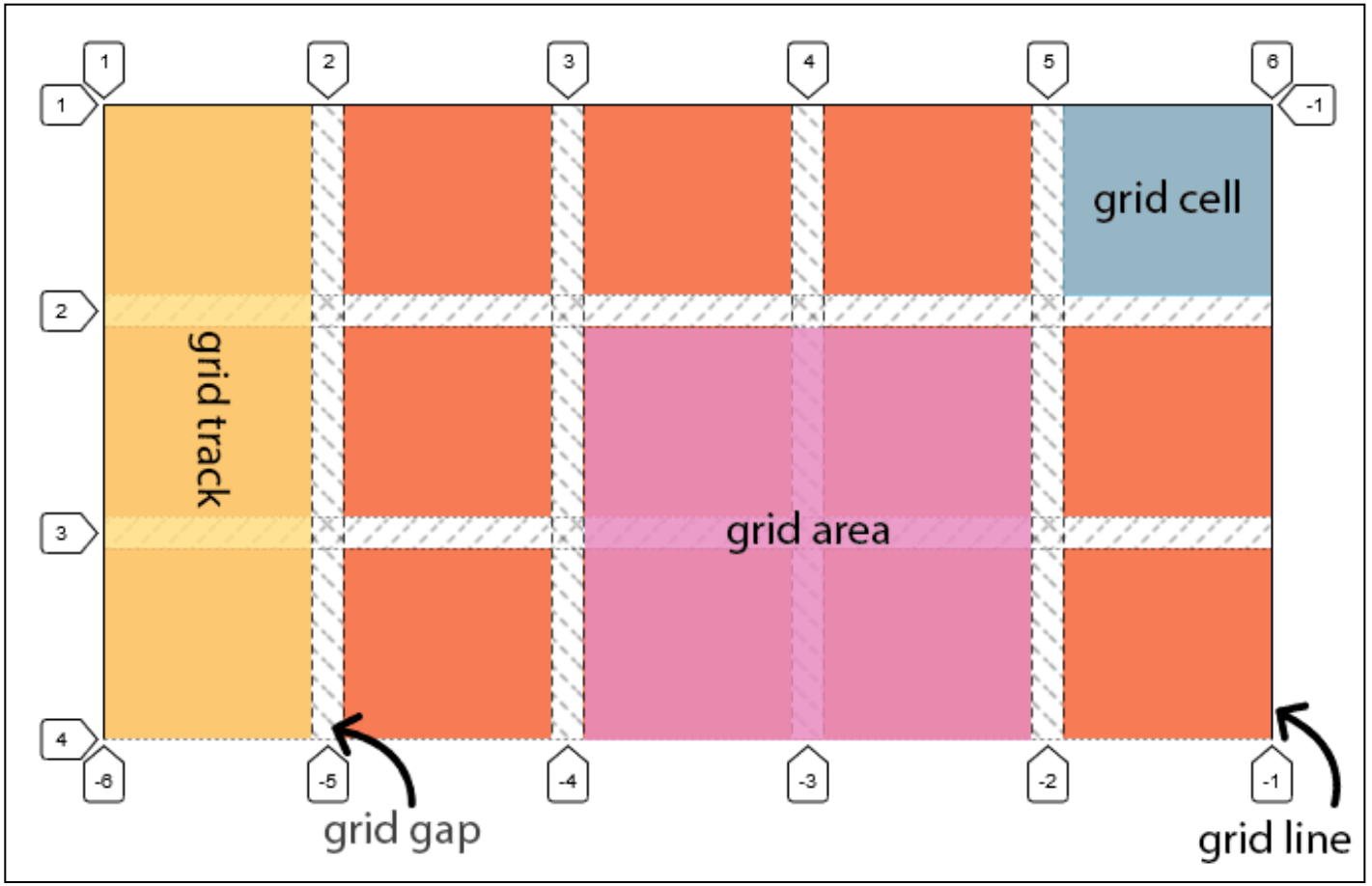
Nota: se si usa un solo valore es. `flex: 3` corrisponde a `flex: 3 1 0px`

CSS Grid

- Il modulo CSS Grid permette di costruire layout basati su una struttura a griglia, formata quindi da righe e colonne su cui sono disposti gli elementi
- Anche in questo caso si usa la proprietà `display:grid` e il layout è formato da un contenitore (**grid container**) e un insieme di **grid item**
- Le proprietà di contenitore e item permettono di controllare accuratamente il modo in cui la griglia è costruita e i contenuti sono disposti e visualizzati

- Sullo stesso sito trovate una guida su CSS Grid: https://css-tricks.com/snippets/css/complete-guide-grid/

CSS Grid: terminologia



Esempi di proprietà per Grid e Grid Item (1)

- Per definire lo spazio occupato da righe e colonne:
  - `grid-template-rows: 1fr 4fr`
  - `grid-template-columns: 100px 300px 400px`
  - `grid-template-columns: 2fr 2fr 1fr`

- Per posizionare un elemento su una o più righe e/o colonne:
  - `grid-column-start: 2 /* grid line */`
  - `grid-column-end: 3 /* grid line */`
  - `grid-row-start: 1 /* grid line */`
  - `grid-row-end: span 3 /* span across 3 grid lines */`

- Oppure usando shorthand (start / end):
  - `grid-column: 2 / 4` `grid-column: 2 / span 2`
  - `grid-row: 1 / 3` `grid-row: 1 / span 2`

Esempi di proprietà per Grid e Grid Item (2)

- Si possono anche definire template che usano i nomi delle aree per indicare su quali righe e/o colonne saranno posizionate, tramite la proprietà **`grid-template-areas`**
- Le aree saranno poi identificate tramite i valori della proprietà **`grid-area`** di ogni *grid item*
- Le dimensioni e proporzioni di righe e colonne possono essere comunque indicate da **`grid-template-rows`** e **`grid-template-columns`**
- Esempio nel file *grid-named.css*:

```
grid-template-areas: "menus menus"
                     "sidebar main"
                     "disclaimer disclaimer"
```

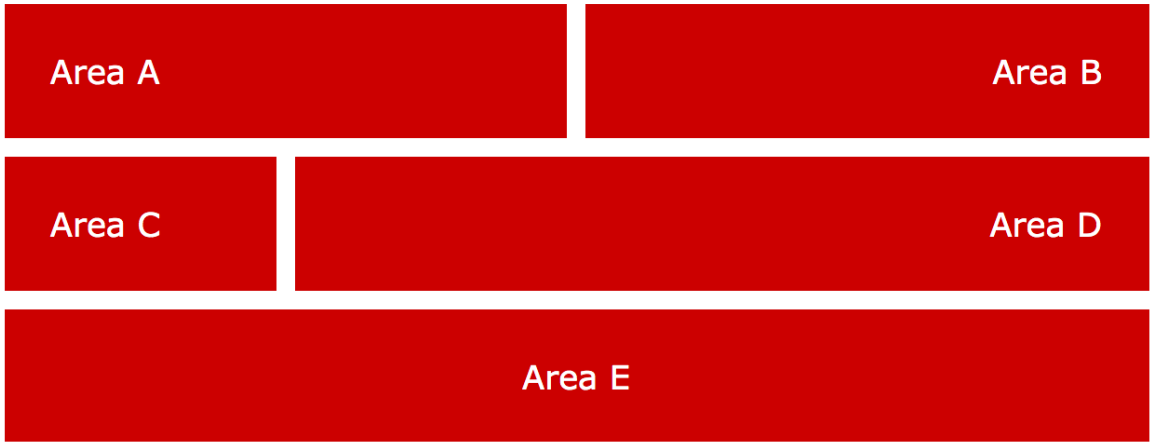
Conclusioni

- Qui abbiamo visto come costruire layout in CSS e ottenere alcuni effetti sofisticati, nella prossima lezione vedremo come rendere i layout responsive e adattarli a diversi dispositivi
- La presentazione dello stesso contenuto può cambiare radicalmente in base ai fogli di stile applicati
- Esistono moltissime risorse in rete con esempi di CSS
- Un sito che ha fatto storia: www.csszengarden.com
- E una sua rivisitazione: https://stylestage.dev/

Esercizi

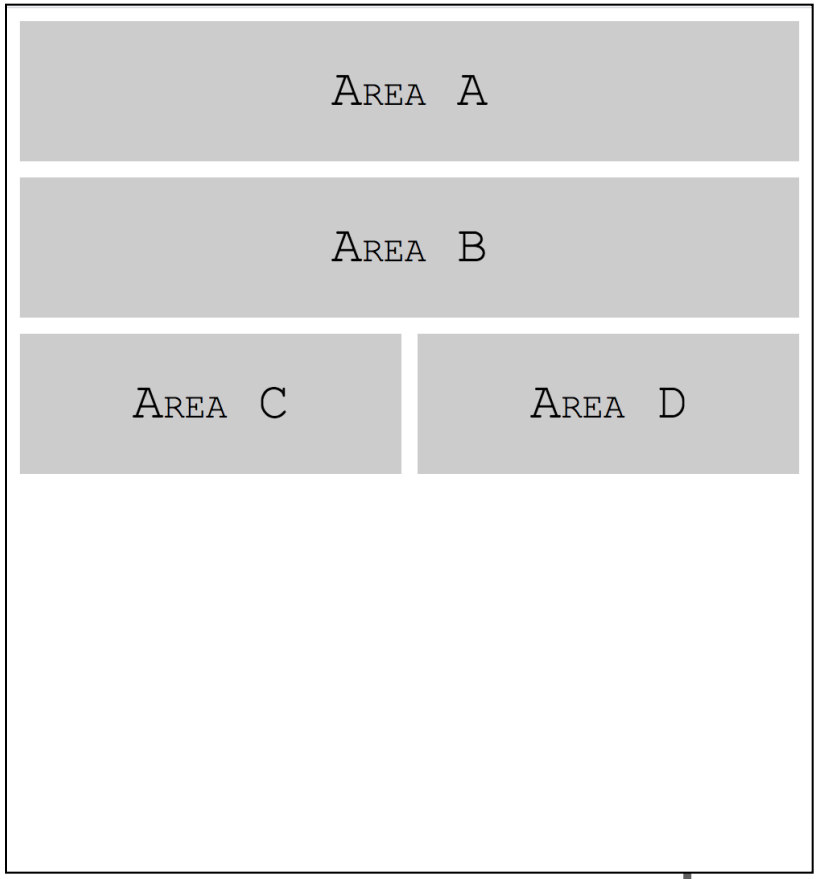
Esercizio 1

- Scrivere il codice HTML e CSS per visualizzare su un browser la seguente griglia di layout
  - larghezza pari alla larghezza del viewport
  - misure esatte del *gap* non rilevanti, colori non rilevanti
  - **usare layout Grid, con e senza *named areas***



Esercizio 2

- Scrivere un foglio di stile CSS in modo che la stessa pagina HTML sia visualizzata come mostrato a destra
  - Il layout occupa tutta la larghezza del viewport
  - colori e spazi esatti non rilevanti
- Provare con:
  - Grid
  - Flexbox (*flex-direction*?)



Esercizio 3

- Scrivere un foglio di stile CSS che applicato alla pagina precedente produca il risultato mostrato sotto
  - CSS precedente non incluso
  - altezza pari all'altezza del viewport
  - Area D posizionata sempre in basso a destra
  - colori e bordi esatti non rilevanti
  - Provare Grid, Float e Flexbox