



Lab Hours - Lecture 3

# **HTML + CSS**

**Layout: Grid & FlexBox**

Contact: [ellepuntopi.98@gmail.com](mailto:ellepuntopi.98@gmail.com)

# Programma di oggi:



# Esercizio Guidato - Raccolta Differenziata

HTML

CSS

Scrivere il codice HTML (e CSS) per visualizzare nel browser il contenuto qui a fianco.

In questo esercizio non siete liberi di fare come volete. Dovremo utilizzare un **Layout FlexBox** per l'intero contenuto, a parte per le singole schede di riciclo, le quali andranno organizzate tramite un **Layout Grid**.

Come al solito è tutto sulla repo GitHub.

DIFFERENZIATA

CATEGORIE DI RICICLO

PLASTICA

PLASTIC



CARTA

RECYCLING



METALLO

METAL



NOT RECYCLABLE



MISTO

UNDIFFERENTIATED

VETRO

RECYCLING

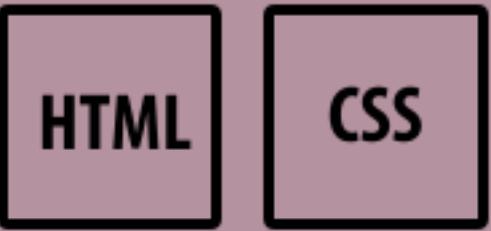


ORGANICO



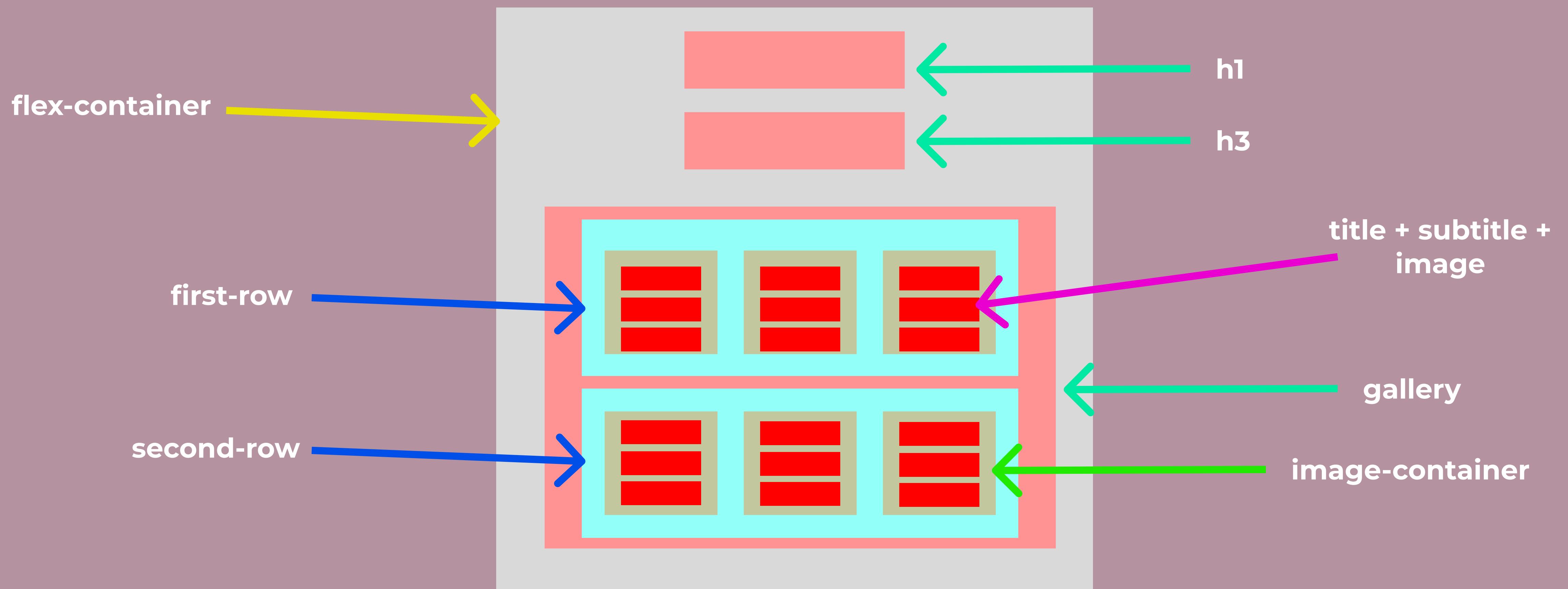
BIOWASTE

# Esercizio Guidato - Raccolta Differenziata (1)



Come prima cosa ispezioniamo insieme l'HTML.

Potete vedere - partendo dal generale per poi arrivare allo specifico - che il file *index.html* è organizzato nel modo seguente:



# Esercizio Guidato - Raccolta Differenziata (2)



Partiamo dunque dall'organizzazione dell'elemento più “esterno” (il container più grande): **flex-container**.

```
.flex-container {  
    display: flex;  
    flex-direction: column;  
    align-items: center;  
    gap: 20px;  
    width: 100%;  
    max-width: 800px;  
}
```

Come vedete, l'elemento presenta la proprietà **display**, con valore **flex**. Questo significa che stiamo impostando l'elemento per contenere al suo interno delle **flexbox**.

**flex-direction** è utile per organizzare le box in colonna (**column**), mentre **align-items** posiziona al centro (**center**) ogni sotto-elemento contenuto al suo interno.

**gap** definisce invece lo spazio tra ogni elemento contenuto nell'oggetto **flex-container**.

# Esercizio Guidato - Raccolta Differenziata (3)



Una volta organizzati in colonna gli elementi **h1**, **h3**, e **gallery**, proseguiamo ed entriamo ancora di più nel dettaglio. Ora proviamo ad organizzare l'elemento **gallery**, contenente i due container **first-row** e **second-row**.

```
.gallery {  
    display: flex;  
    flex-direction: column;  
    /* flex-wrap: wrap; */  
    justify-content: center;  
    align-items: center;  
    gap: 20px;  
    width: 100%;  
    max-width: 700px;  
}
```

Di nuovo, per ottenere la struttura iniziale dobbiamo posizionare in colonna i due sotto-elementi di **gallery**. Per farlo utilizziamo di nuovo **display-flex**.

Questa volta aggiungiamo anche **justify-content**, che ci permette di aggiustare la posizione degli elementi lungo l'asse principale di riferimento, che in questo caso è quella verticale (data da 'flex-direction: column;')

Considerate che:

- > align-items lavora sull'asse secondario (quello perpendicolare alla direzione fornita da flex-direction)
- > justify-content lavora sull'asse principale (quello di flex-direction)

# Esercizio Guidato - Raccolta Differenziata (4)



Ora dobbiamo posizionare gli elementi contenuti in **first-row** e **second-row**. Per farlo utilizziamo di nuovo un **display:flex**, ma questa volta la direzione sarà **row**, e **non più column**, questo perché vogliamo che i tre elementi contenuti in ognuna delle due **row** siano uno di fianco all'altro.

```
#first-row, #second-row {  
    display: flex;  
    flex-direction: row;  
    flex-wrap: nowrap;  
    gap: 20px;  
    justify-content: space-around;  
    width: 100%;  
}
```

In questo caso potete vedere anche la proprietà **flex-wrap**, che definisce se gli elementi all'interno del container possono andare a capo (**wrap**) o no (**nowrap**). In questo caso li vogliamo tutti e tre sulla stessa riga, senza andare a capo. Questo potrebbe essere un problema nel momento in cui lo schermo si rimpicciolisce, ma per risolvere la questione vedrete le **media-query** durante la prossima lezione.

Infine, utilizziamo **justify-content: space-around** per definire il posizionamento orizzontale. **space-around** distribuisce equamente lo spazio intorno ad ogni sotto-elemento.

# Esercizio Guidato - Raccolta Differenziata (5)

HTML

CSS

Siamo arrivati al layout degli elementi presenti all'interno di **first-row** e **second-row**, ovvero gli elementi con classe **image-container**.

Questi andranno organizzati tramite un layout di tipo **grid**, per il quale andiamo a definire le tre macro-aree di interesse: **top**, **middle**, e **bottom**.

```
.image-container {  
    text-align: center;  
    background-color: white;  
    border: 2px solid black;  
    padding: 15px;  
    border-radius: 10%;  
    display: grid;  
    gap: 2px;  
    grid-template-rows: auto auto auto;  
    grid-template-areas:  
        "top"  
        "middle"  
        "bottom";  
    box-shadow: 10px 8px 4px 0 rgba(0, 0, 0, 0.2);  
}
```

Oltre ad allineare il testo al centro, a definire un colore di sfondo, un bordo, un padding, ed un raggio di curvatura per gli angoli, andiamo anche a definire **display: grid**, che ci permette di organizzare il contenuto su una griglia.

Definiamo un **gap** tra gli elementi della griglia. Successivamente specifichiamo le dimensioni delle tre aree (o righe in questo caso) che abbiamo definito prima. Per farlo utilizziamo **grid-template-rows** e impostiamo le dimensioni su **auto**. Questo permette agli elementi di occupare esattamente lo spazio necessario per essere mostrati a schermo.

In ultimo definiamo l'ombra dei box tramite **box-shadow** (*displacement orizzontale, displacement verticale, raggio di sfocatura, raggio di diffusione, colore*).

# Esercizio Guidato - Raccolta Differenziata (5.b)



Una volta definita la nostra griglia, dovremo attribuire le giuste aree ai giusti elementi:

```
.top .title {  
    font-size: 25px;  
    letter-spacing: 2px;  
    font-weight: bold;  
    grid-area: top; /* Assegna l'area 'top' */  
}  
  
.top .subtitle {  
    font-size: 18px;  
    letter-spacing: 1px;  
    grid-area: middle; /* Assegna l'area 'middle' */  
}  
  
.top img {  
    grid-area: bottom; /* Assegna l'area 'bottom' */  
}
```



**N.B.:** stiamo accedendo ai vari sotto-elementi di classi specifiche tramite una notazione nested: entriamo prima nella classe **.top**, e poi nei vari elementi (**.title**, **.subtitle**, e **img**).

# Correzione Esercizi in Classe

Adesso vedremo insieme i due esercizi che vi ha dato il professor Di Iorio in classe durante l'ultima lezione.

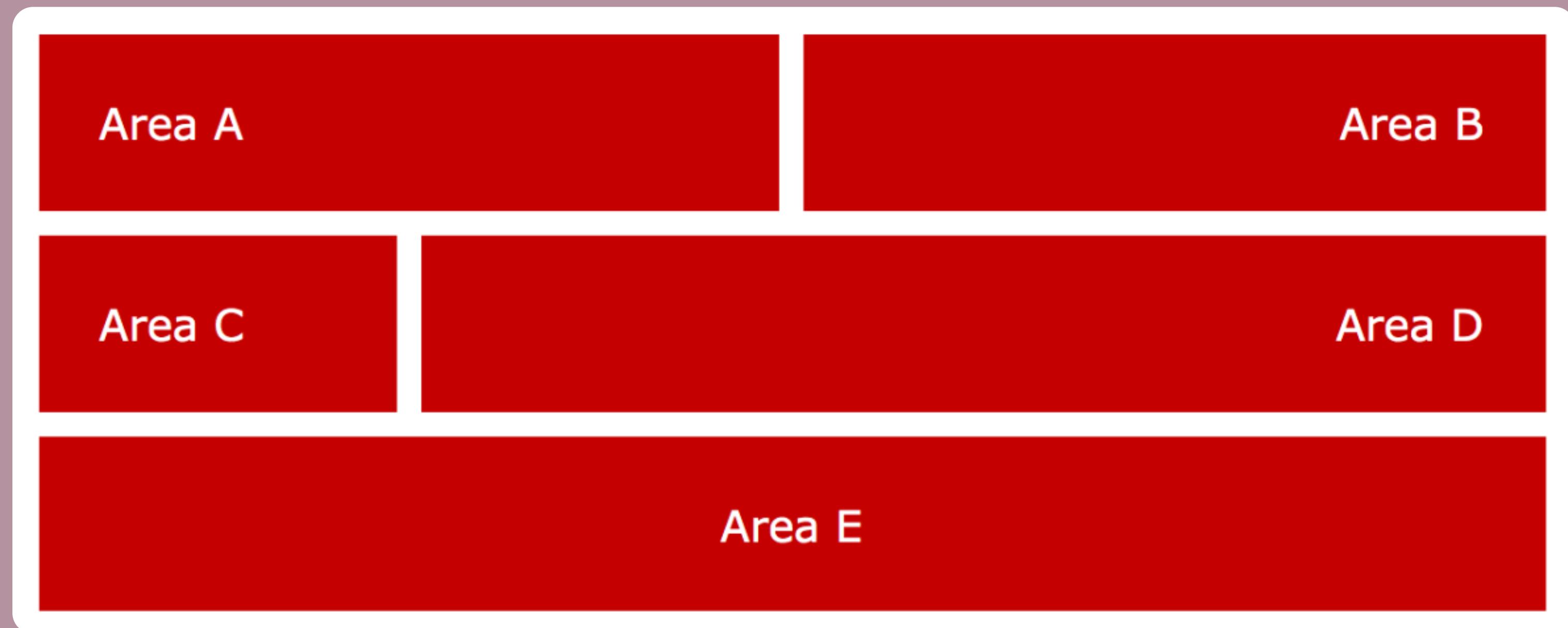
Il codice HTML è lo stesso per entrambi gli esercizi, e potete trovarlo sul GitHub sotto la cartella **EserciziClasse** della lezione di oggi.

# Correzione Esercizi in Classe - Esercizio 1



Scrivere il codice CSS per visualizzare su un browser la seguente griglia di layout

- larghezza pari alla larghezza del viewport
- misure esatte dei gap non rilevanti, colori non rilevanti
- usare layout Grid, **con e senza named areas**



# Correzione Esercizi in Classe - Esercizio 1 (1)



Faremo insieme la parte con le griglie normali (senza grid-areas).

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

La prima cosa da notare è l'applicazione di **box-sizing: border-box**; a tutti gli elementi. Quando impostate **box-sizing: border-box**; a qualcosa, state *dicendo al browser di includere il padding e il bordo nel calcolo della larghezza e dell'altezza dell'elemento.*

Questo significa che:

- **La larghezza totale dell'elemento sarà sempre uguale al valore che avete impostato per la larghezza**, anche se aggiungete padding e bordi. **Lo stesso vale per l'altezza.**
- **Il padding e il bordo sono inclusi all'interno delle dimensioni dell'elemento**, invece di aggiungerle all'esterno. In altre parole, *il padding e il bordo riducono la larghezza disponibile per il contenuto dell'elemento, ma non aumentano la larghezza totale dell'elemento.*

# Correzione Esercizi in Classe - Esercizio 1 (2)

HTML

CSS

Andiamo ora a definire la **grid**.

```
#main {  
    display: grid;  
    grid-template-columns: repeat(4, 1fr);  
    grid-gap: 10px;  
    width: 100vw;  
    max-width: 100vw;  
    padding: 5px;  
    height: 50vh;  
    margin: auto;  
    font-family: Verdana;  
}
```

Come vedete definiamo solo le colonne: **4 colonne, dove ognuna occupa una frazione equamente suddivisa dello spazio a disposizione**. Questo è possibile grazie a:

**grid-template-columns: repeat(4, 1fr);**  
**repeat()** permette di specificare un numero di volte che un certo pattern di dimensioni (di colonna in questo caso) dovrebbe essere ripetuto. La sintassi è la seguente:

**repeat(numero\_di\_ripetizioni, dimensione);**  
Successivamente, specifichiamo una larghezza massima (il totale della larghezza del viewport = **100vw**) e la larghezza effettiva (**width**).

**margin: auto;** non è necessario in questo caso, perché stiamo già occupando tutto lo spazio. Altrimenti sarebbe servito a centrare il contenuto.

# Correzione Esercizi in Classe - Esercizio 1 (3)



Fatto questo, definiamo le singole celle, dando però prima uno stile applicabile ad ogni cella tramite il selettore nested **#main div {...}**.

```
#main div {  
    background-color: #cc0000;  
    color: #fff;  
    width: 100%; /* Ogni div occupa l'intero spazio concessogli */  
}  
  
#a {  
    grid-column: 1 / span 2; /* da colonna 1, occupa due colonne */  
    grid-row: 1; /* prima riga */  
}  
  
#b {  
    grid-column: 3 / span 2; /* da colonna 3, occupa due colonne */  
    grid-row: 1; /* prima riga */  
}  
  
#c {  
    grid-column: 1; /* occupa solo la prima colonna */  
    grid-row: 2; /* seconda riga */  
}
```

Come vedete, ogni **id** (cella) occupa uno spazio ben preciso. Ed è proprio tramite la definizione delle righe occupate dalle celle (**grid-row**) che andiamo a definire il numero di righe totali della griglia.

**Le notazioni possibili sono 3:**

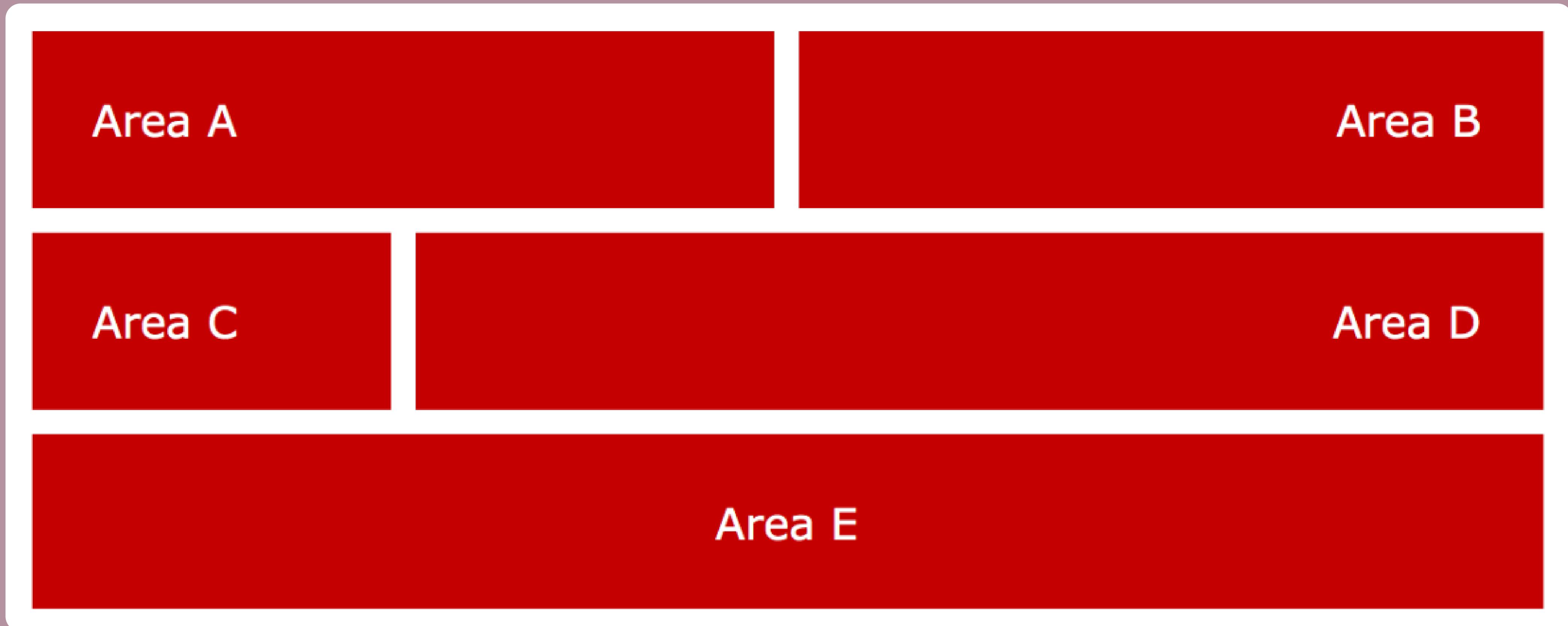
- numero singolo***: l'elemento occupa esattamente quella colonna, o riga;
- numero / span***: l'elemento, a partire dalla posizione *numero*, si estende per posizioni *span*;
- x / y***: l'elemento si estende da posizione *x* a posizione *y*.

# Esercizi in Classe - Esercizio 1

HTML

CSS

L'esercizio 1 è così terminato nella sua forma *normal-grid*. Ora provate voi a rifare lo stesso esercizio, però utilizzando le **grid-areas**.

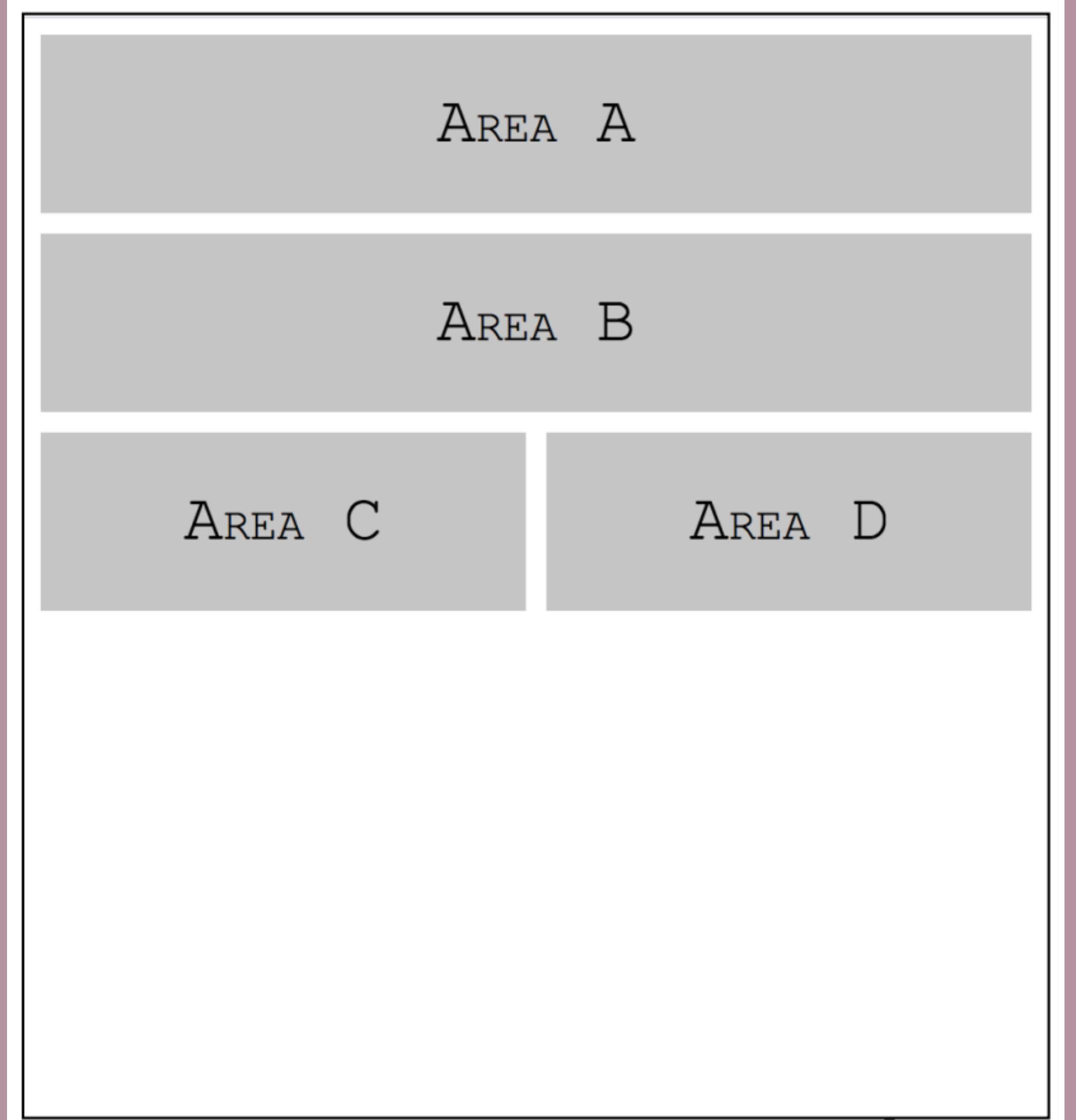


# Esercizi in Classe - Esercizio 2



Con lo stesso codice HTML, vi chiedo ora di ricostruire l'immagine seguente tramite un **layout flex** (no grid, no modifica HTML).

Successivamente lo correggeremo insieme.

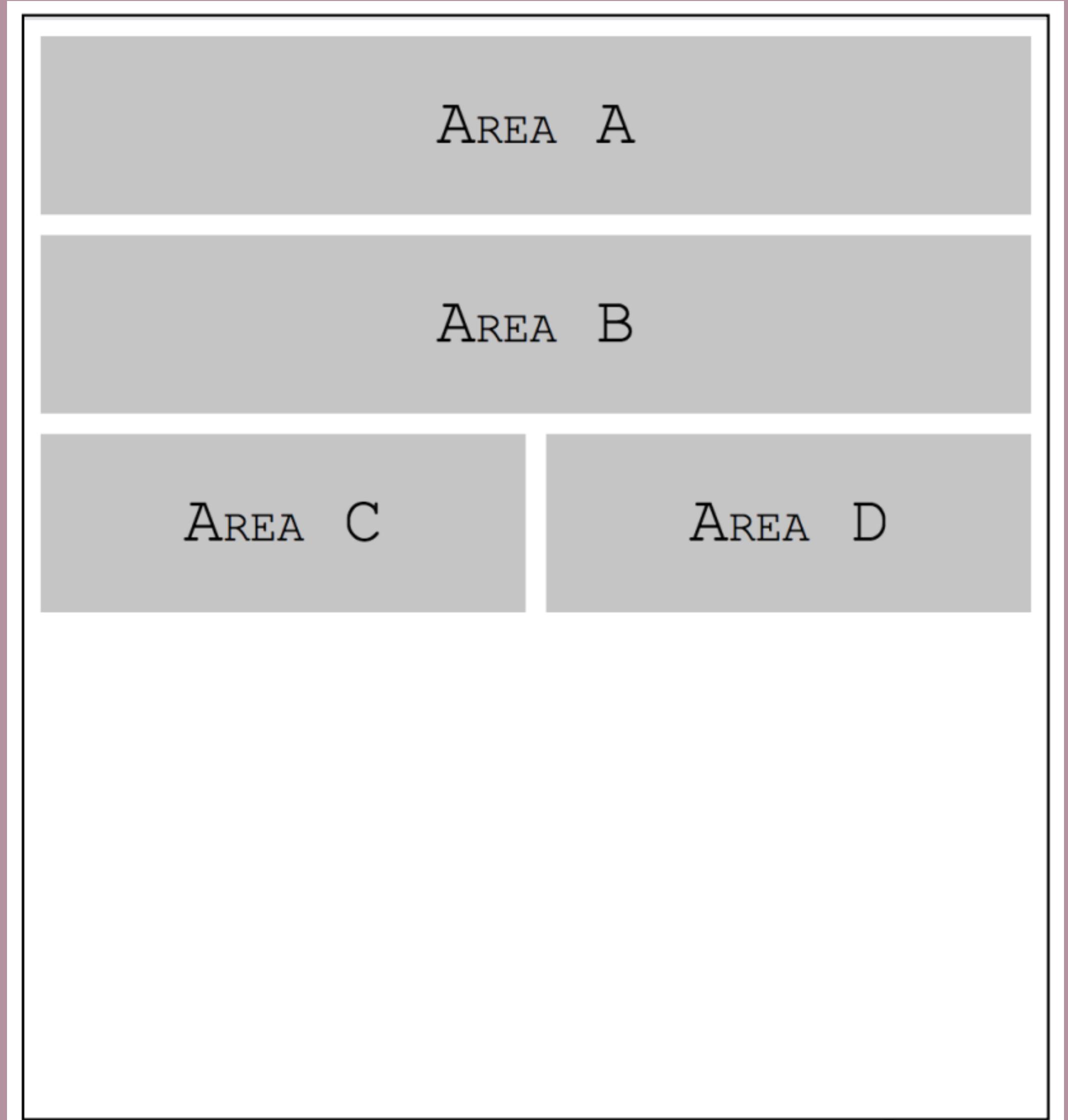


# Esercizi in Classe - Esercizio 2 (1)



Infine, provate a ricostruire lo stesso layout con **Grid**.

Successivamente lo correggeremo insieme.



HTML

CSS

Fine