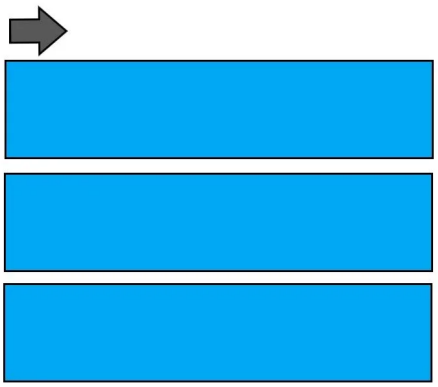
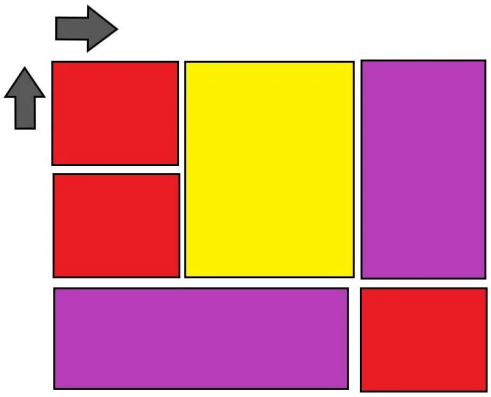
Grid

Introduction: Flex Vs Grid



Flexbox is one-dimensional

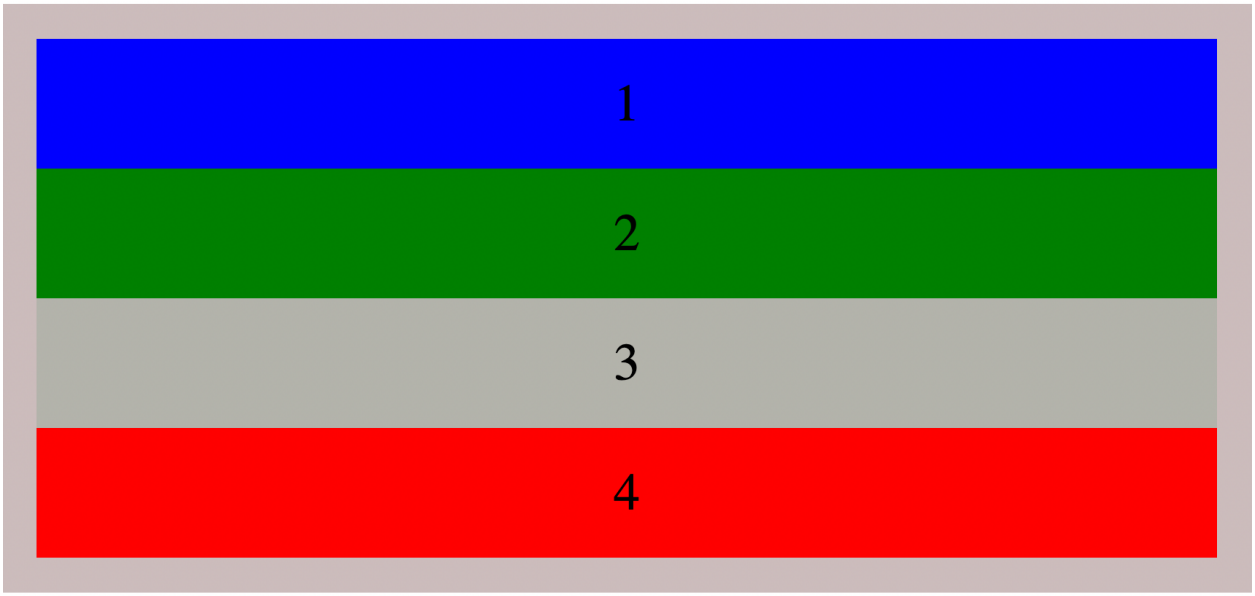


CSS Grid is two-dimensional

Basic Grid :

```
<style>
      #container {
        display: grid;
      }
    </style>
  </head>
  <body>
    <div id="container">
      <div>1</div>
      <div>2</div>
      <div>3</div>
      <div>4</div>
      <div>5</div>
      <div>6</div>
    </div>
  </body>

Note: some styles are hidden like background color etc
```

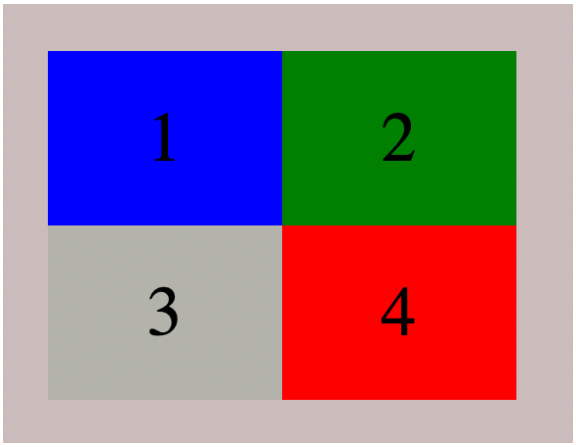
Output:



grid-template-columns

- Defines the columns and rows of the grid with a space-separated list of values.

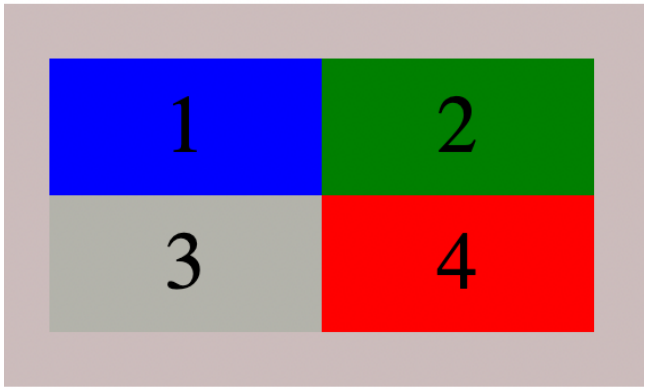
```
<style>
      #container {
        display: grid;
        grid-template-columns: 100px 100px; // 100px represents column size
      }
    </style>
  </head>
  <body>
    <div id="container">
      <div>1</div>
      <div>2</div>
      <div>3</div>
      <div>4</div>
      <div>5</div>
      <div>6</div>
    </div>
  </body>

Note: some styles are hidden like background color etc
```



grid-template-rows

- We can also specify height of each row by using `grid-template-rows` property

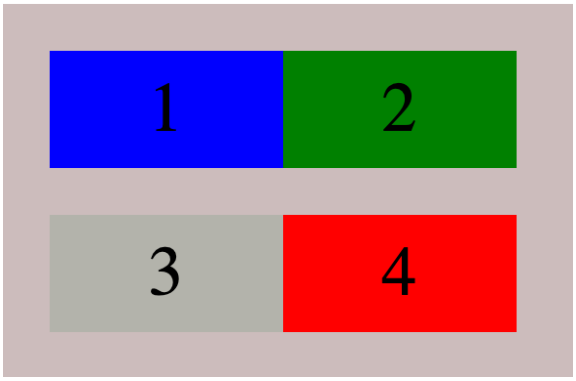
```
<style>
      #container {
        display: grid;
        grid-template-columns: 100px 100px; // 100px represents column size
        grid-template-rows: 50px 50px ; // 100px represents row height
      }
    </style>
  </head>
  <body>
    <div id="container">
      <div>1</div>
      <div>2</div>
      <div>3</div>
      <div>4</div>
      <div>5</div>
      <div>6</div>
    </div>
  </body>

Note: some styles are hidden like background color etc
```



- To get gap between each item, we can use `grid-row-gap`

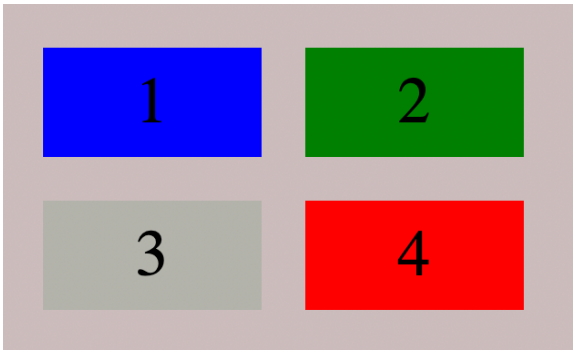
```
<style>
      #container {
        display: grid;
        grid-template-columns: 100px 100px;
        grid-template-rows: 50px 50px;
        grid-row-gap:20px
      }
    </style>
  </head>
  <body>
    <div id="container">
      <div>1</div>
      <div>2</div>
      <div>3</div>
      <div>4</div>
      <div>5</div>
      <div>6</div>
```

```
    </div>
  </body>
```



- `grid-column-gap`

```
<style>
      #container {
        display: grid;
        grid-template-columns: 100px 100px;
        grid-template-rows: 50px 50px;
        grid-row-gap:20px;
        grid-column-gap:20px
      }
    </style>
  </head>
  <body>
    <div id="container">
      <div>1</div>
      <div>2</div>
      <div>3</div>
      <div>4</div>
      <div>5</div>
      <div>6</div>
    </div>
  </body>
```



- Shorthand notation for grid-gap

```
<style>
      #container {
```

```
        display: grid;
        grid-template-columns: 100px 100px;
        grid-template-rows: 50px 50px;
        /* grid-row-gap:20px;
        grid-column-gap: 20px; */
        gap:20px 20px
      }
    </style>
  </head>
  <body>
    <div id="container">
      <div>1</div>
      <div>2</div>
      <div>3</div>
      <div>4</div>
    </div>
  </body>
```

repeat():

- The `repeat()` CSS function represents a repeated fragment of the track list, allowing a large number of columns or rows that exhibit a recurring pattern to be written in a more compact form.
- Syntax : `repeat(no_of_times,size)`
- for eg:
  - grid-template-columns: 100px 100px can be written as `repeat(2,100px)`
- Now the above code can be changed as

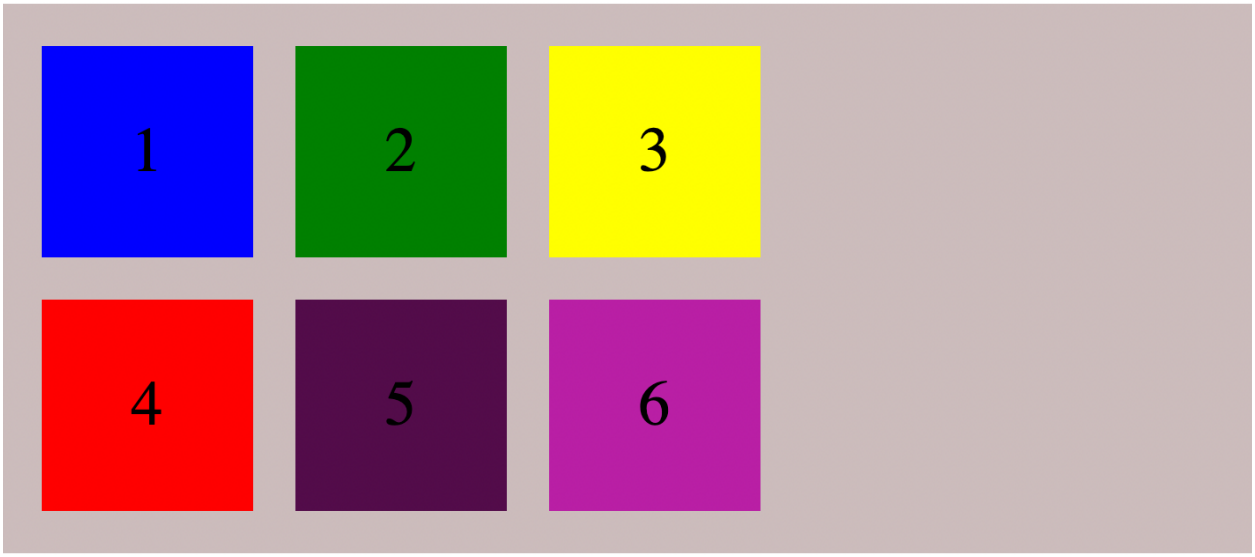
```
<style>
      #container {
        display: grid;
        grid-template-columns: repeat(2,100px);
        grid-template-rows: repeat(2,50px);
        gap:20px 20px
      }
    </style>
  </head>
  <body>
    <div id="container">
      <div>1</div>
      <div>2</div>
      <div>3</div>
      <div>4</div>
    </div>
  </body>
```

- Now lets try to build a layout which has 2 rows and 3 columns with different backgrounds

```
#container {
        display: grid;
        grid-template-columns: 100px 100px 100px;
        grid-template-rows: 100px 100px;
        gap:20px 20px
      }
    </style>
  </head>
  <body>
```

```
    <div id="container">
      <div>1</div>
      <div>2</div>
      <div>3</div>
      <div>4</div>
      <div>5</div>
      <div>6</div>
    </div>
  </body>
```

Output:



Live code: Codepen

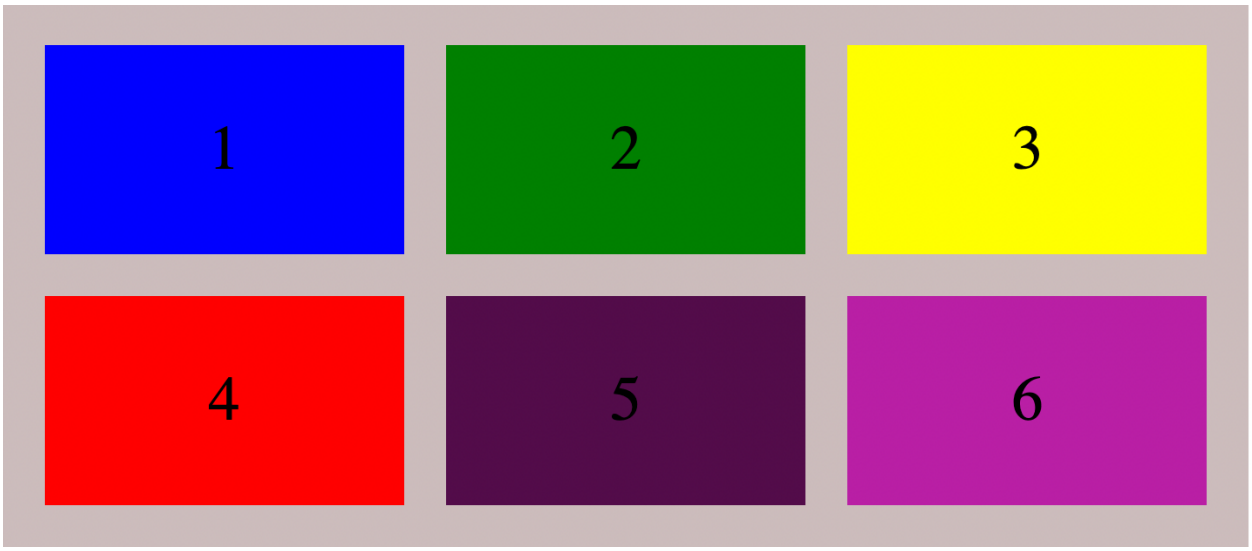
Units of grid-template-columns:

```
Pixels
  grid-template-columns: 100px 100px or repeat(2,100px)

Percentages
  grid-template-columns: 50% 50% or repeat(2,50%)
  // each column will take 50% width in reference to its parent

  grid-template-columns: 1fr 1fr or repeat(2,1fr)
  // each column will take 1 fraction width in reference to its parent
```

```
<style>
      #container {
        display: grid;
        grid-template-columns: repeat(3,1fr);
        grid-template-rows: 100px 100px;
        gap:20px 20px
      }
    </style>
  </head>
  <body>
    <div id="container">
      <div>1</div>
      <div>2</div>
```

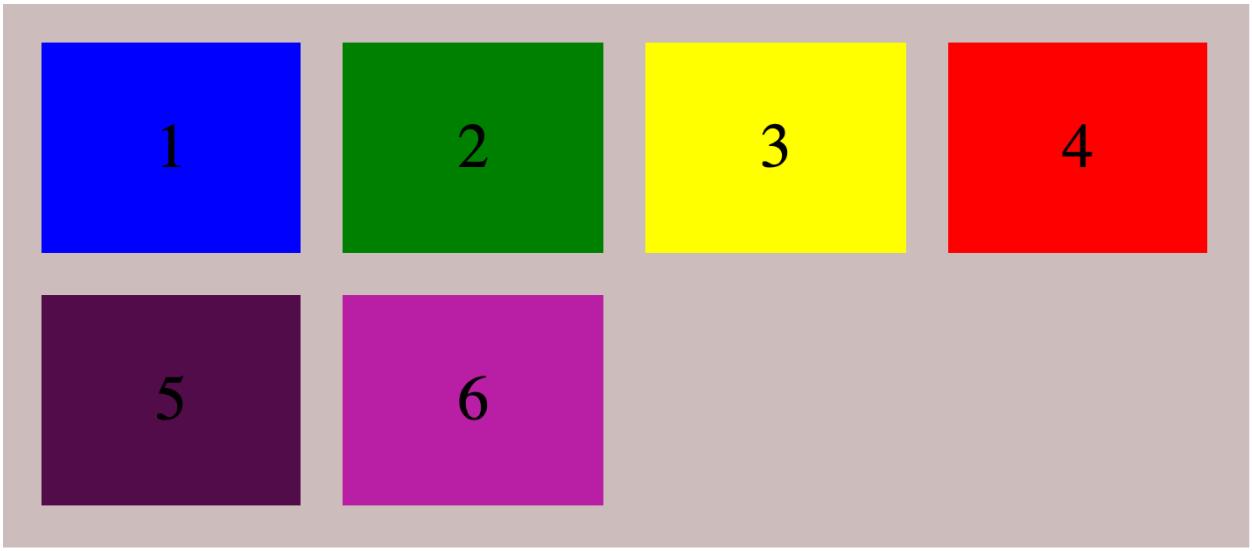
```
      <div>3</div>
      <div>4</div>
      <div>5</div>
      <div>6</div>
    </div>
  </body>
```

Output: Dividing each block into 3 fractions




```
<style>
      #container {
        display: grid;
        grid-template-columns: repeat(3,1fr);
        grid-template-rows: 100px 100px;
        gap:20px 20px
      }
    </style>
  </head>
  <body>
    <div id="container">
      <div>1</div>
      <div>2</div>
      <div>3</div>
      <div>4</div>
      <div>5</div>
      <div>6</div>
    </div>
  </body>
```

Output: Dividing each block into 4 fractions



Specificity:

- In Selectors lecture we have seen scores of various selectors

Scores of various selectors

Selector Name	Score
Inline	1000
Id	100
Class	10
Tag	1
Universal	0

- We have also learned few more selectors like
  - Attribute Selector
  - Pseudo Class
    - :hover

        - :nth-child
        - :focus
    - Pseudo Element
        - ::first-line
        - ::after

- What about scores of above selectors?

Scores of various selectors

Selector Name	Score
Inline	1000
Id	100
Class,attribute,pseudo classes	10
Tag, pseudo element	1
Universal,combinators (+, > , ~,)	0

What is Specificity?

- If there are two or more CSS rules that point to the same element, the selector with the highest specificity value will "win", and its style declaration will be applied to that HTML element.
- Think of specificity as a score/rank that determines which style declaration are ultimately applied to an element.
- Let's take an example

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Document</title>
    <style>
      /* Specificity score - 100 points */
```

```
      #masai {
        background-color: teal;
      }
      /* specificity score - 1 point */
      h1 {
        background-color: red;
      }
    </style>
  </head>
  <body>
    <div>
      <h1 id="masai">Masai school</h1>
    </div>
  </body>
</html>
```

- In the above example id has more specificity score compared to tag, so background of `teal` will be applied.
- Now, what if I use combinators to style `h1` tag

```
<style>
      /* Specificity score - 100 points */
      #masai {
        background-color: teal;
      }

      /* specificity score - 1 point */
      h1 {
        background-color: red;
      }

      /* specificity score - 1+1 = 2 points */
      div > h1 {
        background-color: yellow;
      }

      /* specificity score - 1+100 = 101 points */
      div > #masai {
        background-color: pink;
      }
    </style>
```

- Now since `div>#masai` has highest specificiy score of 101, `pink` will be applied

The table below shows some examples on how to calculate specificity values:

Selector	inline Style (1000)	#id (100)	.class (10)	tag (1)	**Calculation**	Specificity Value
ul.class	0	0	10	1	0+0+10+1	11
h1 + p	0	0	0	1+1	0+0+0+2	2
h1 ~ p::first-letter	0	0	0	1+1+1	0+0+0+3	3
.heading p.main	0	0	10+10	1	0+0+20+1	21
#heading p ul li .disabled li	0	100	10	1+1+1+1	100+10+4	114

Selector	inline Style (1000)	#id (100)	.class (10)	tag (1)	**Calculation**	Specificity Value
.heading p.main ul li.disabled li	0	0	10+10+10	1+1+1+1	0+0+30+4	34
#navbar p#demo	0	100+100	0	1	0+200+1	201
<p style="color: pink;">	1000	0	0	0	1000	1000