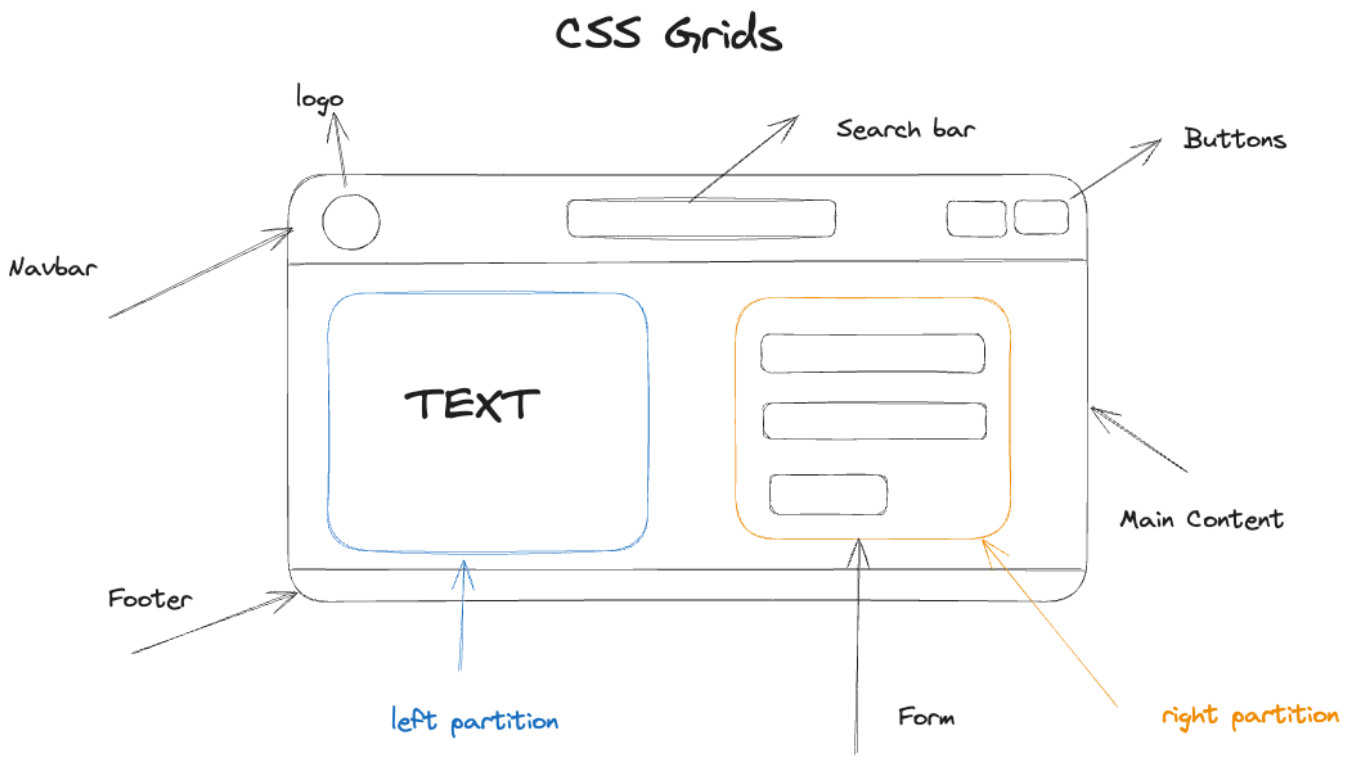
CSS Grids

Preparing layouts in websites is one of the most important concepts that we need to master. When we describe layout then we mainly refer to arrangements of the components in out web pages.



The above image refers to the arrangement of different components we are referring to. Here different UI elements are arranged at specific places and order.

Ways to do arrangements using CSS

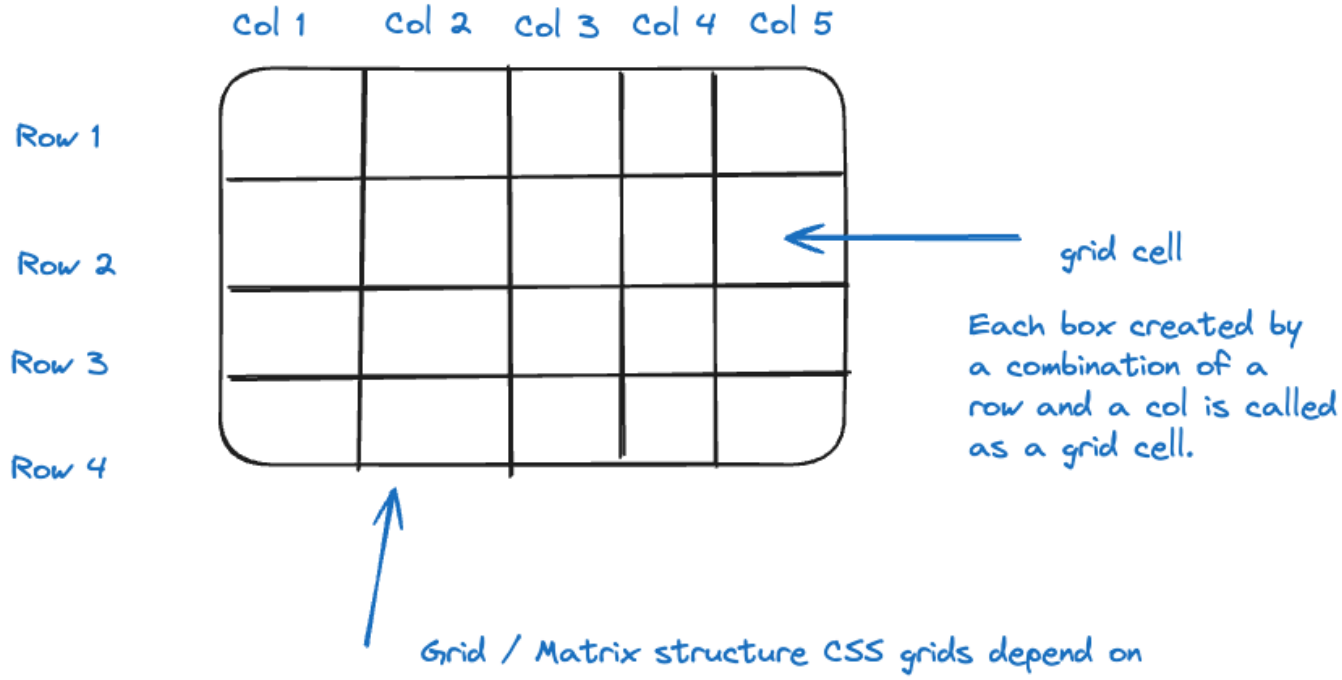
Flex box is one very useful way to prepare these arrangements using CSS. An alternative to flex box for preparing UI component arrangements and layouts is **CSS GRIDS**.

FYI, before CSS Grids and flex box, developers used to rely on tables, floats, positions etc to do the arrangements.

What is CSS Grids ? Why should we learn CSS Grids ?

CSS Grids are an alternate way to prepare layouts and arrange UI elements in a web page. CSS Grids is a 2D (two dimensional) layout system. Flex box on the other hand is a 1D (One dimensional) layout system. Means, Flex box only gives us control in one dimension of the layout.

CSS grids create a grid / matrix like layout which comprises of rows and columns using which we can arrange the elements in these rows and columns. This gives more granular control on the content we want to display.



CSS Grid Terminologies

Grid Container

If you already know about flex box, then you must've heard of flex container. Similar to that we have Grid container. Grid container is the parent element inside which we can activate CSS grids. Once done, any child element inside this container will be controllable by CSS grids.

To enable a parent element with CSS grids we can use the display CSS rule and allocate it a value grid.

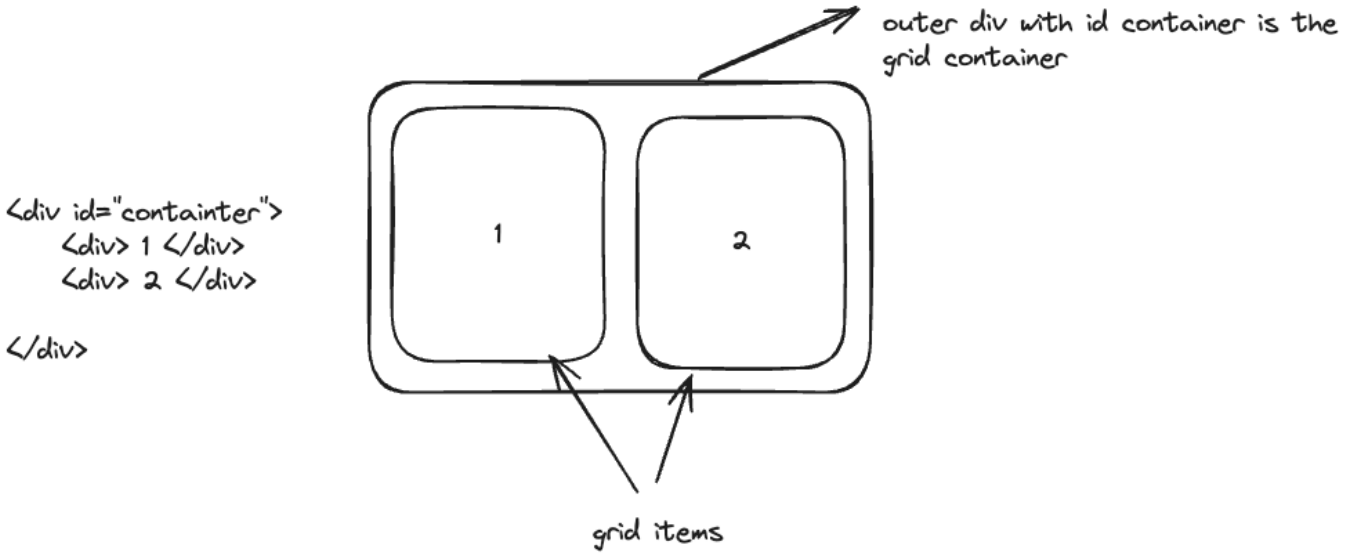
```
SELECTOR_FOR_PARENT_ELEMENT {
        display: grid;
}
```

With the help of `display` rule we can activate `grid` by allocation the `grid` value to the `display` property.

Grid Item

The child elements inside the Grid container are called as Grid items. It is very similar to that of flex items. After allocation the grid value to display property of a parent, we will be able to

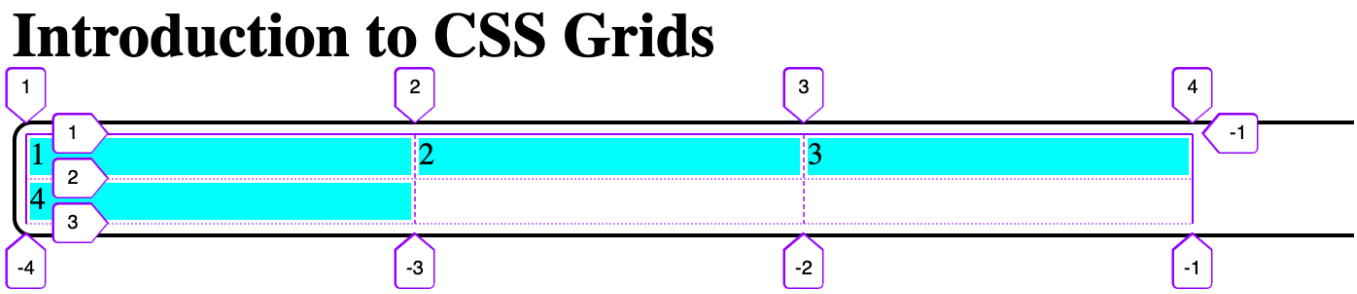
control these Grid items.



To make a similar implementation we can write

```
#container {
        display: grid;
        padding: 5px;
        border: 2px solid black;
        border-radius: 10px;
        grid-template-columns: 100px 100px;
}
```

Grid Lines

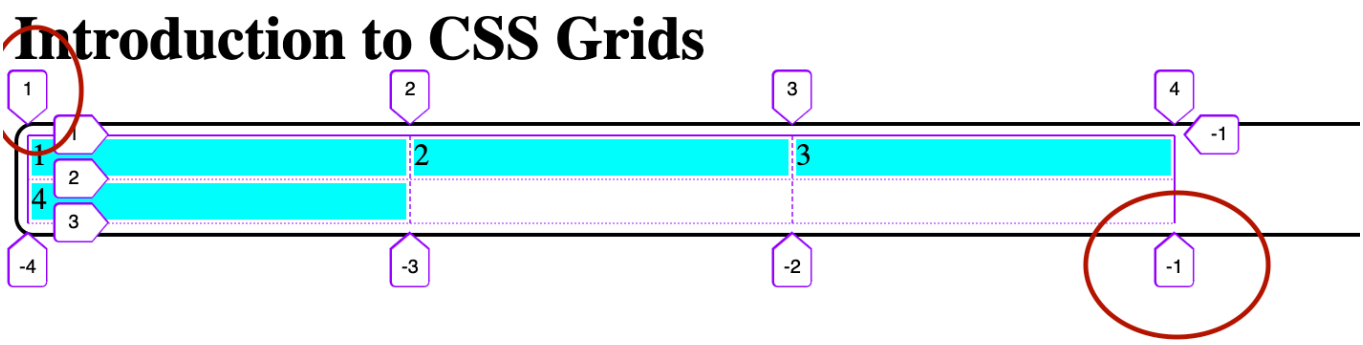


CSS Grid Lines are a bunch of horizontal and vertical lines using which our whole grid / matrix is divided into rows and columns.

- Horizontal lines separate the rows
- Vertical lines separate the columns

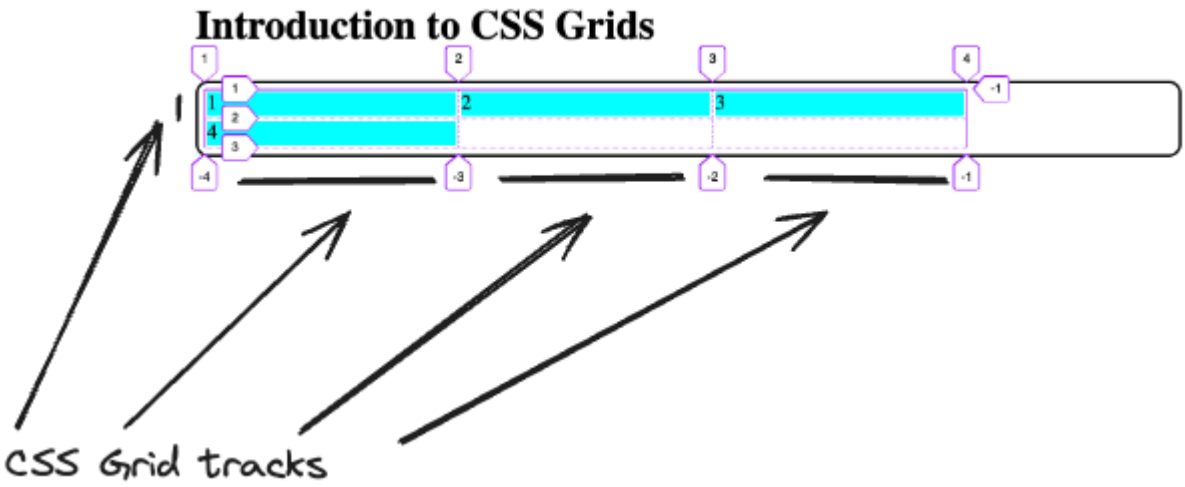
Every line has a number starting from 1 (i.e. 1 refers to first line, 2 refers to second line and so on). Then there are numbers for reverse ordering as well starting from -1 (i.e. -1 refers to the

last line, -2 refers the second last line and so on).



These lines play a very very important role in preparing the layout using css grids. For most of the rules we might need to define the start and end of a cell, which can be done using these line numbers.

CSS Grid tracks



The css grid tracks refers to the space between adjacent grid lines (horizontal or vertical anyone). These actually helps us to visualise the space distribution among rows and columns. This will be the place where actually some elements will be rendered in the grid.

Note:

To specify size of a row or a column inside `grid-template-rows` or `grid-template-columns` we can use **fractional units** as well.

Fractional Units

Fractional Units or **fr** is also one way to mention size of the column width or row height in terms of available space in the grid container.

CSS Grid rules

To control the orientation of the grid-container i.e. number of columns and rows, along with size of each column and row, we need to master 3 important properties:

- grid-template-columns
- grid-template-rows
- grid-auto-flow

grid-template-columns

Using this rule, we will be able to define the number of columns in the CSS Grid and along with that, we can also specify size of each column.

To use this rule and define the number of columns and size of each column, we have to just give n-number of values space-separated to this CSS rule.

```
Selector_of_the_element {
        display: grid; /* This activates the grid in the parent container*/
        grid-template-columns: 20px 20px 30em 40%; /* 4 values space
sepaarted */
}
```

In the above property we have given 4 different values to `grid-template-columns`. This means that there should be `4` columns and width of the columns should be:

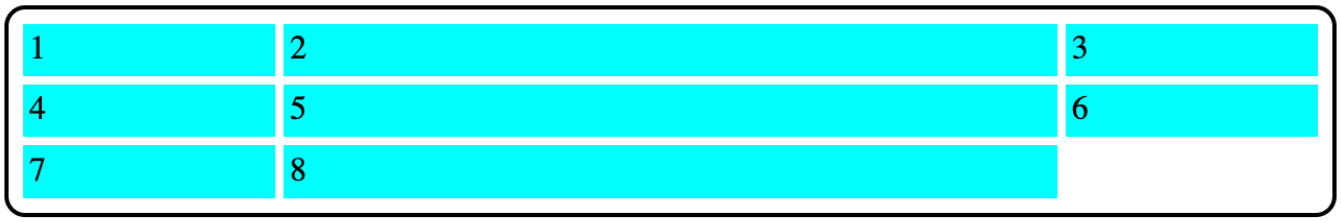
- 20px for first column
- 20px for second column
- 30em for third column
- 40% of parent width for fourth column

Size of each column can be different and can be mentioned in different units as well.

```
<div id="container">
        <div class="item">
                1
        </div>
        <div class="item">
                2
        </div>
        <!-- 6 more divs with item class having similar sequence number in
```

```
text -->
</div>
```

```
#container {
        display: grid; /* grid layout */
        /* more styling*/
        grid-template-columns: 20% 60% 20%;
}
```



In the above CSS, because we provided only 3 values to the grid-template-columns property, we have only 3 columns with the corresponding width.

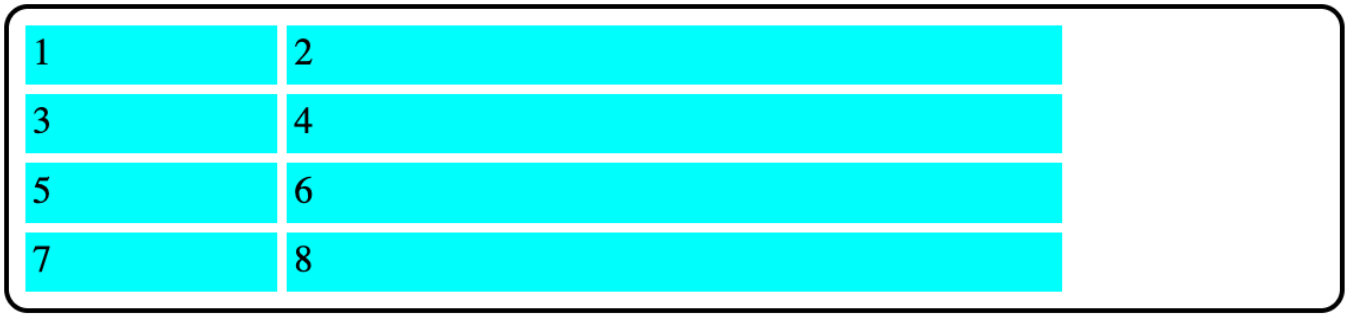
grid-auto-flow

This property decides the flow of the remaining elements after the decided number of columns or rows. **This only controls extra elements which are not coming within the number of rows / columns defined.**

For example, if by using `grid-template-columns` we decided to have 2 columns of 50px wide length, and total grid items are 8, so the first two items will be arranged in each column one by one, but what about other ?
So the remaining ones are actually arranged based on this flow.

So if the value of `grid-auto-flow` is `row` (and row is the default value also), then all the remaining items will go in the next row.

```
        display: grid; /* grid layout */
        grid-template-columns: 20% 60%; /* only 2 columns defined */
        grid-auto-flow: row; /*Default value, Because of this, remaining
elements are gone to a new row*/
```



And if the the value of `grid-auto-flow` is `column` then all the remaining items will go in a column.

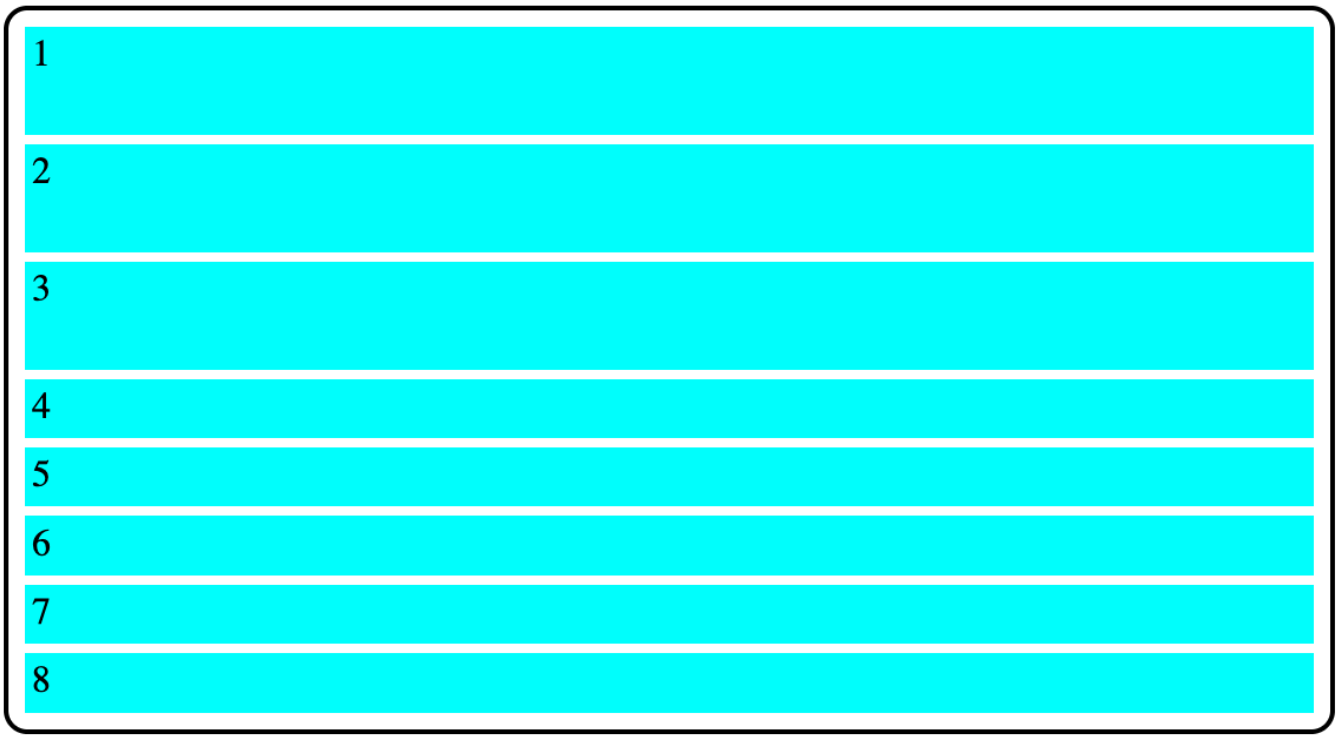
```
        display: grid; /* grid layout */
        grid-template-columns: 20% 60%; /* only 2 columns defined */
        grid-auto-flow: column; /* Because of this, remaining elements are
gone to a new column*/
```

grid-template-rows

Using this rule, we will be able to define the number of rows in the CSS Grid and along with that, we can also specify size of each row.

But because `grid-auto-flow` is by default having a value a `row` all the remaining elements will go to each new row.

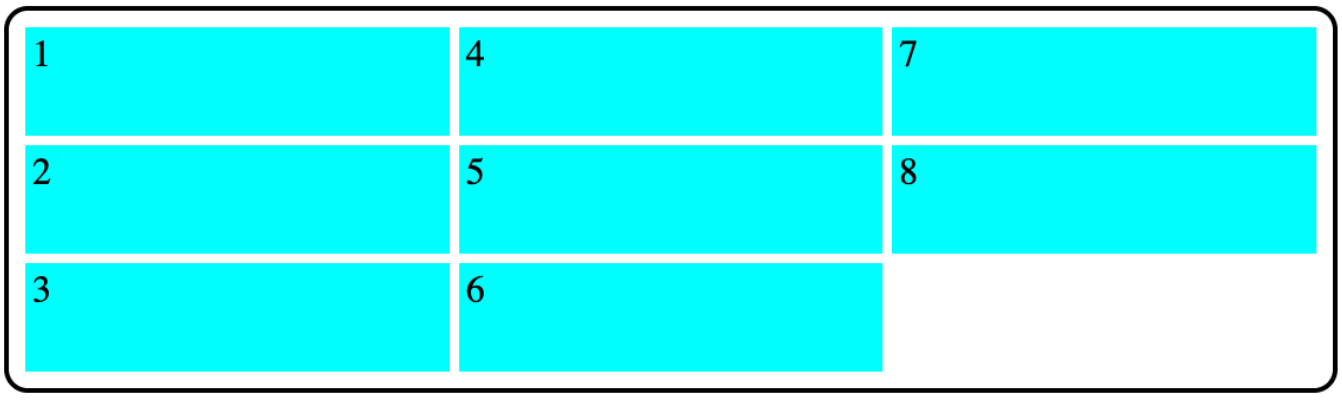
```
        display: grid; /* grid layout */
        grid-template-rows: 50px 50px 50px; /* only 2 columns defined */
```



So now because we defined only 3 values in grid-template-rows, so we have height of first 3 rows as 50px and all the remaining elements will go to a new row as `grid-auto-flow` is `row`.

But if we add `grid-auto-flow` as `column`, then we will be having only 3 rows with `50px` height and all the remaining elements will go new column.

```
        display: grid; /* grid layout */
        grid-template-rows: 50px 50px 50px; /* only 2 columns defined */
        grid-auto-flow: column; /* Because of this, remaining elements are
gone to a new column*/
```



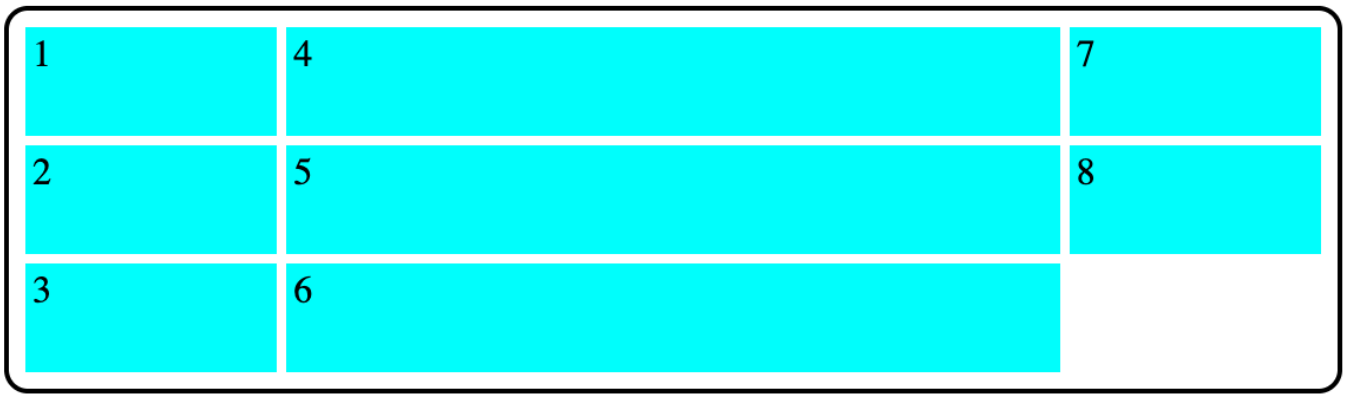
Or, if we have to avoid `grid-auto-flow` we can define both `grid-template-columns` and `grid-template-rows` to define the number of columns and rows specifically.

```
        display: grid; /* grid layout */
        grid-template-columns: 20% 60%; /* only 2 columns defined */
        grid-template-rows: 50px 50px 50px;
```

You can see here, `grid-template-columns` is having 2 values 20% and 60% meaning we only want two columns. That's good, and we have `grid-template-rows` with 50px each for 3 rows. Meaning we want to have 3 rows. But why is the 4th row coming ?
This is because the default `grid-auto-flow` is row, meaning extra elements should be adjusted in a new row. So 7 and 8 will go to a new row.

So if we will give `grid-auto-flow` as `column` here, does that mean extra elements will now go to columns ? yes.

```
        display: grid; /* grid layout */
        grid-template-columns: 20% 60%; /* only 2 columns defined */
        grid-template-rows: 50px 50px 50px;
        grid-auto-flow: column;
```

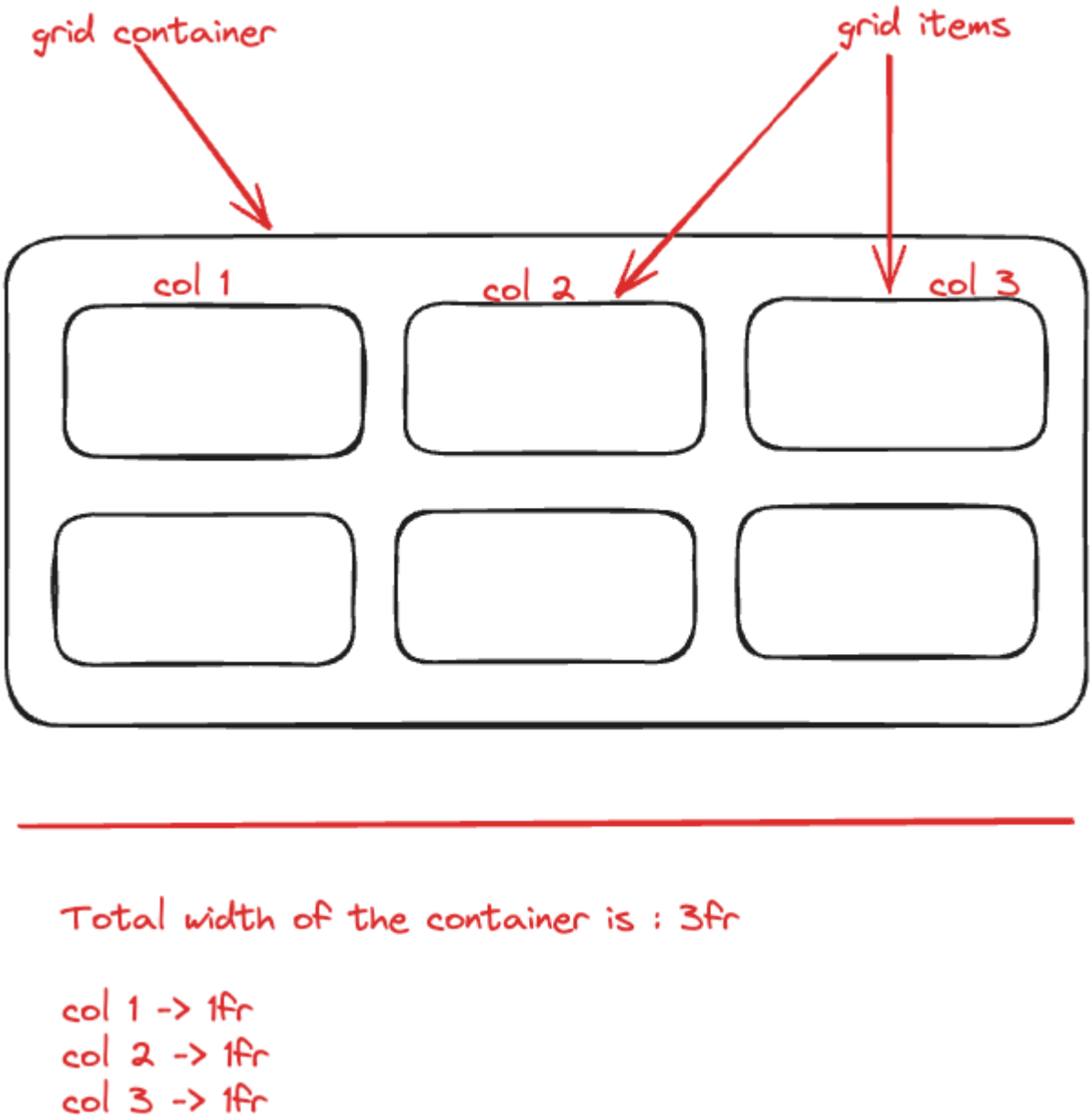


fractional units with grid-template

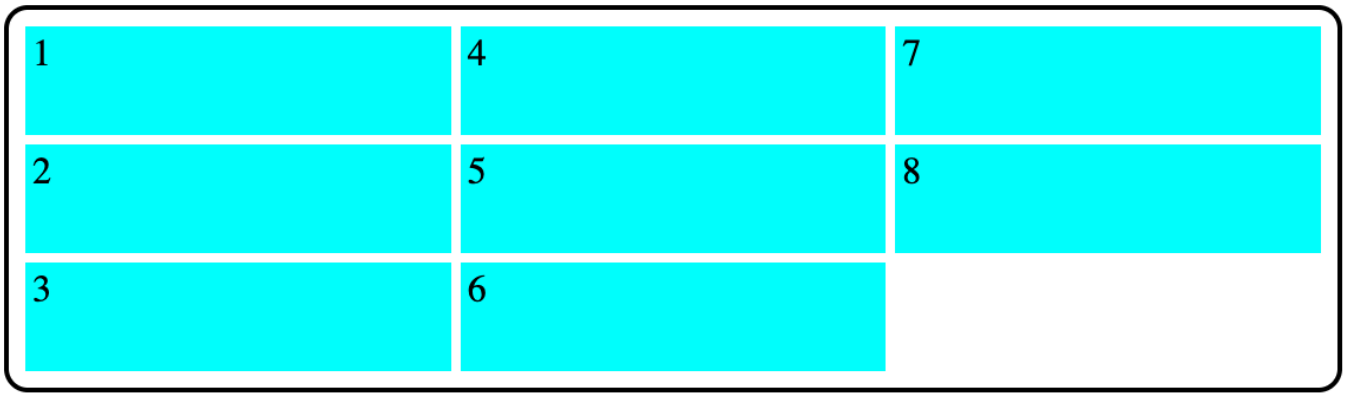
As we know fractional units can be used to tell the size of column width or row height w.r.t the space available in the container.

```
display: grid; /* grid layout */
grid-template-columns: 1fr 1fr 1fr;
grid-template-rows: 50px 50px 50px;
grid-auto-flow: column;
```

In the code above, we have mentioned 3 values in grid-template-columns that means we expect 3 columns. Each value is 1fr, 1fr represent the fractional unit of each column i.e. 1/3 of the total space. How ?



If size of the total container is 3fr, then 1fr went to first column, 1fr went to 2nd column and 1fr went to third column.

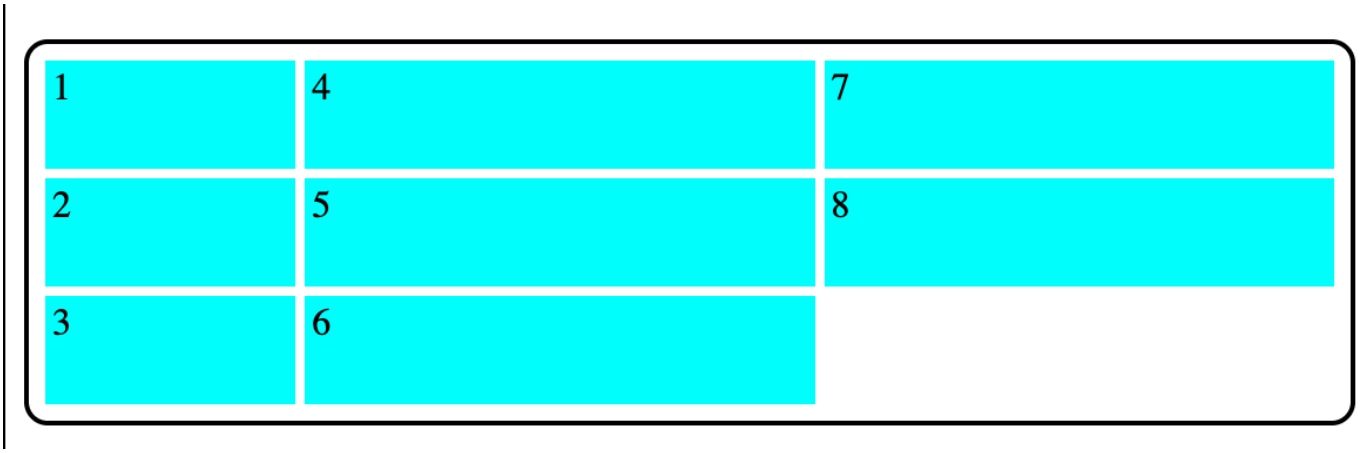


For more examples, we can refer the below:

Example 1

```
display: grid; /* grid layout */
grid-template-columns: 1fr 2fr 2fr;
grid-template-rows: 50px 50px 50px;
grid-auto-flow: column;
```

Here we can total 5fr space (1 + 2 + 2) available in the container, among which the 1/5 of the space i.e. 1fr goes to column 1, then 2/5 of the space i.e. 2fr goes to second column and the last 2/5 of the space i.e. again 2fr goes to the last column.

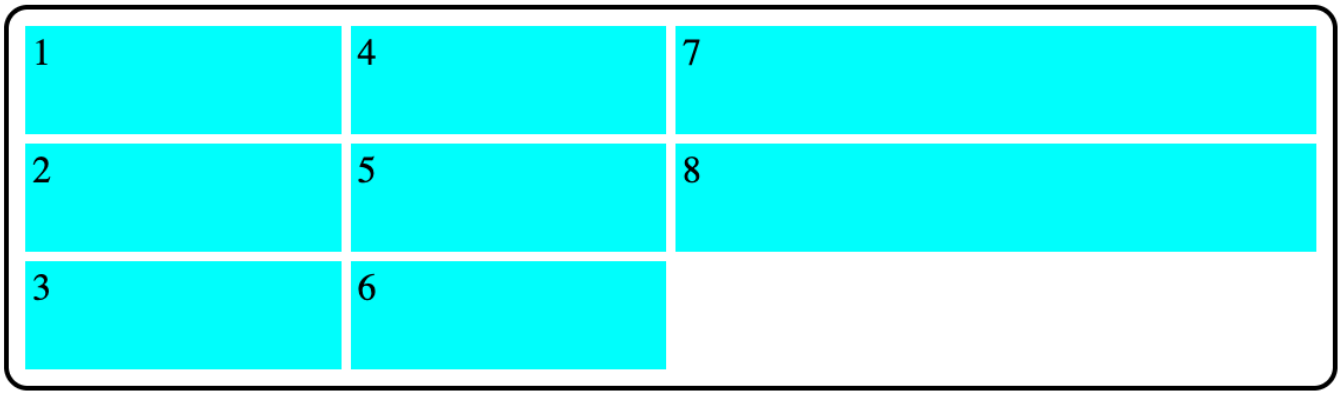


Example 2

```
display: grid; /* grid layout */
grid-template-columns: 1fr 1fr 2fr;
grid-template-rows: 50px 50px 50px;
grid-auto-flow: column;
```

Here we can total 4fr space (1 + 1 + 2) available in the container, among which the 1/4 of the space i.e. 1fr goes to column 1, then 1/4 of the space i.e. 1fr goes to second column and the

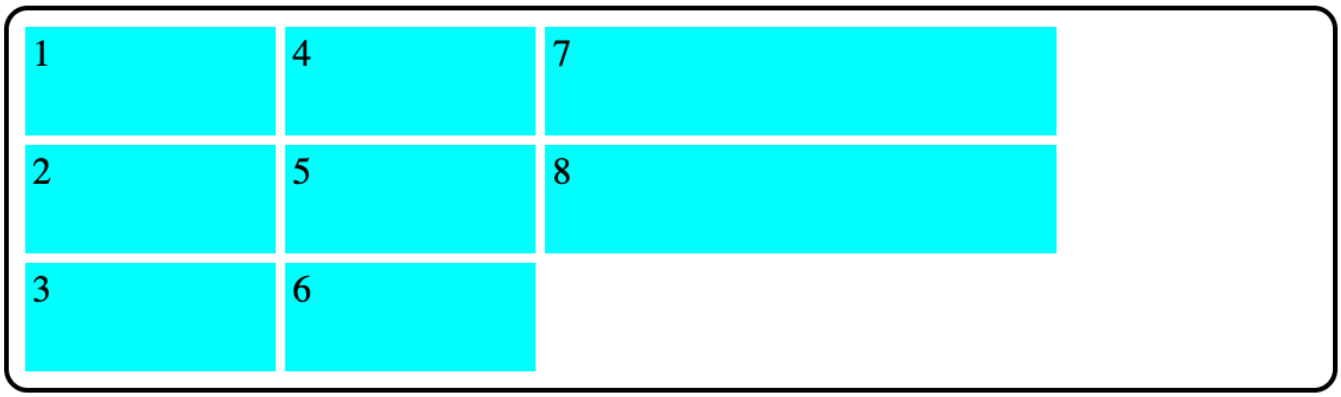
last 2/4 of the space i.e. 2fr goes to the last column.



Example 3

```
        display: grid; /* grid layout */
        grid-template-columns: 1fr 1fr 2fr 1fr;
        grid-template-rows: 50px 50px 50px;
        grid-auto-flow: column;
```

Here we can total 5fr space (1 + 1 + 2 + 1) available in the container, among which the 1/5 of the space i.e. 1fr goes to column 1, then 1/5 of the space i.e. 1fr goes to second column, then 2/5 of the space i.e. 2fr goes to the third column and remaining 1/5 goes to fourth column.



But why the last column is empty ?

It is because of the `grid-auto-flow`, as it is equal to `column` the filling of grid cells happen in column fashion i.e. all extra values goes to next column.
Now beyond 7 and 8 we don't have enough items to fill the space, but because we have mentioned the spacing of 4 columns, the last column stays empty but also keeps it's size entact, just because we have mentioned the size of the column manually as 1fr.

Example 3

```
        display: grid; /* grid layout */
        grid-template-columns: 1fr 1fr 2fr 1fr;
```

Here we can total 5fr space (1 + 1 + 2 + 1) available in the container, among which the 1/5 of the space i.e. 1fr goes to column 1, then 1/5 of the space i.e. 1fr goes to second column, then 2/5 of the space i.e. 2fr goes to the third column and remaining 1/5 goes to fourth column.

Here the `grid-auto-flow` is row, so all the elements are first filled in the row 1 and all extra moves to next row, hence no empty columns.

So this was it for the basics of CSS grids. In the further upcoming set of parts, we will talk about more intermediate and advanced concepts related to CSS grids