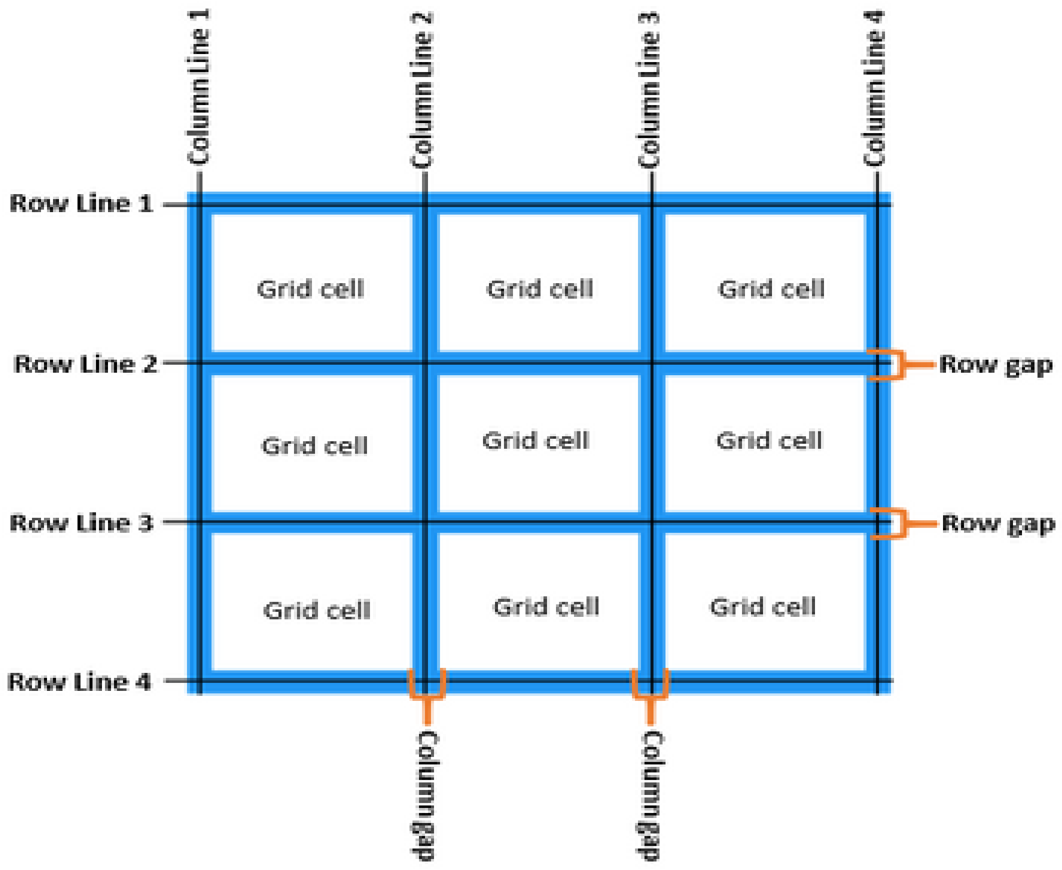
CSS Grid

- A Grid is a collection of horizontal and vertical lines, creating a pattern against which we can line up our elements
- It provides greater consistency on our website
- It is 2D layout model. Content is laid out in rows and columns.
- Grid has rows, columns and gap between each row and column. The gaps are commonly referred as gutters



Grid Vs FlexBox

Grid

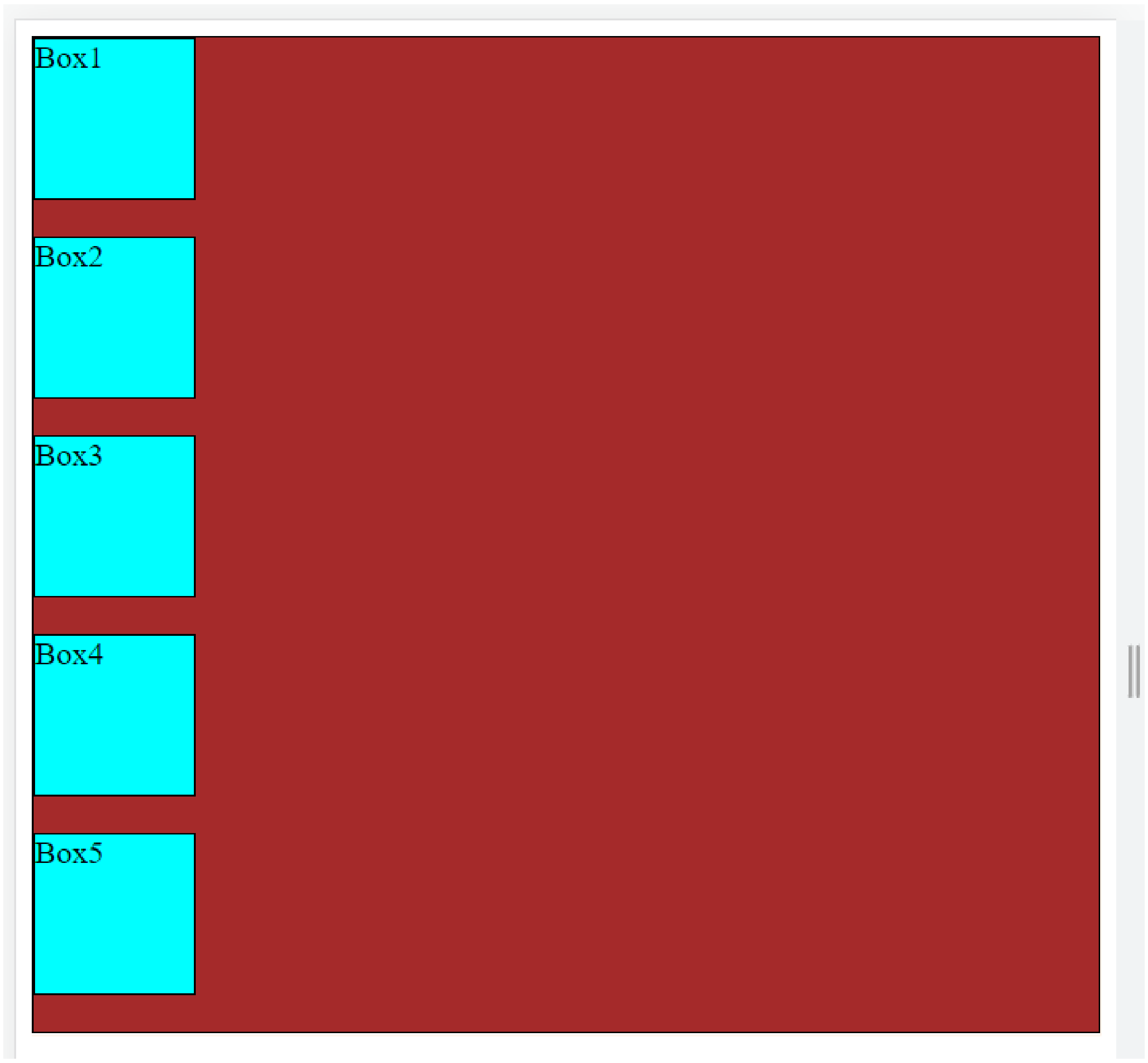
- It is 2D layout model
- It is use for layout creation

FlexBox

- It is 1D layout model.
- It is use to show how to position content / how content flows.

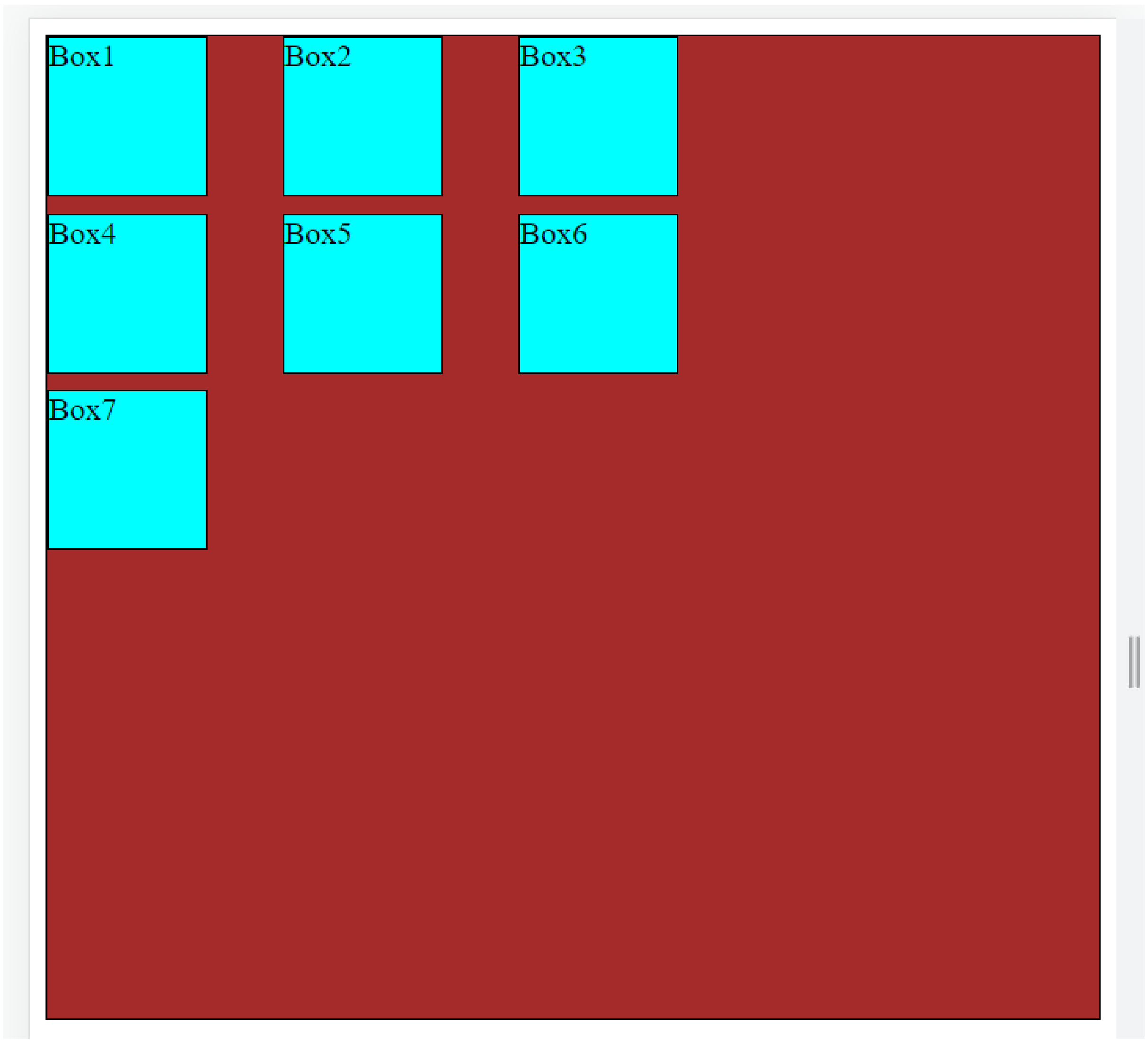
We can make a container as grid by using **display: grid;**

```css
.container{
    background-color: brown;
    height: 500px;
    border: 1px solid black;
    display: grid;
}
.box{
    background-color: aqua;
    border: 1px solid black;
    width: 80px;
    height: 80px;
}
```



Grid Container Properties

```
grid-template-columns: 120px 120px 100px;
grid-template-rows: 90px 90px 90px 90px;
```



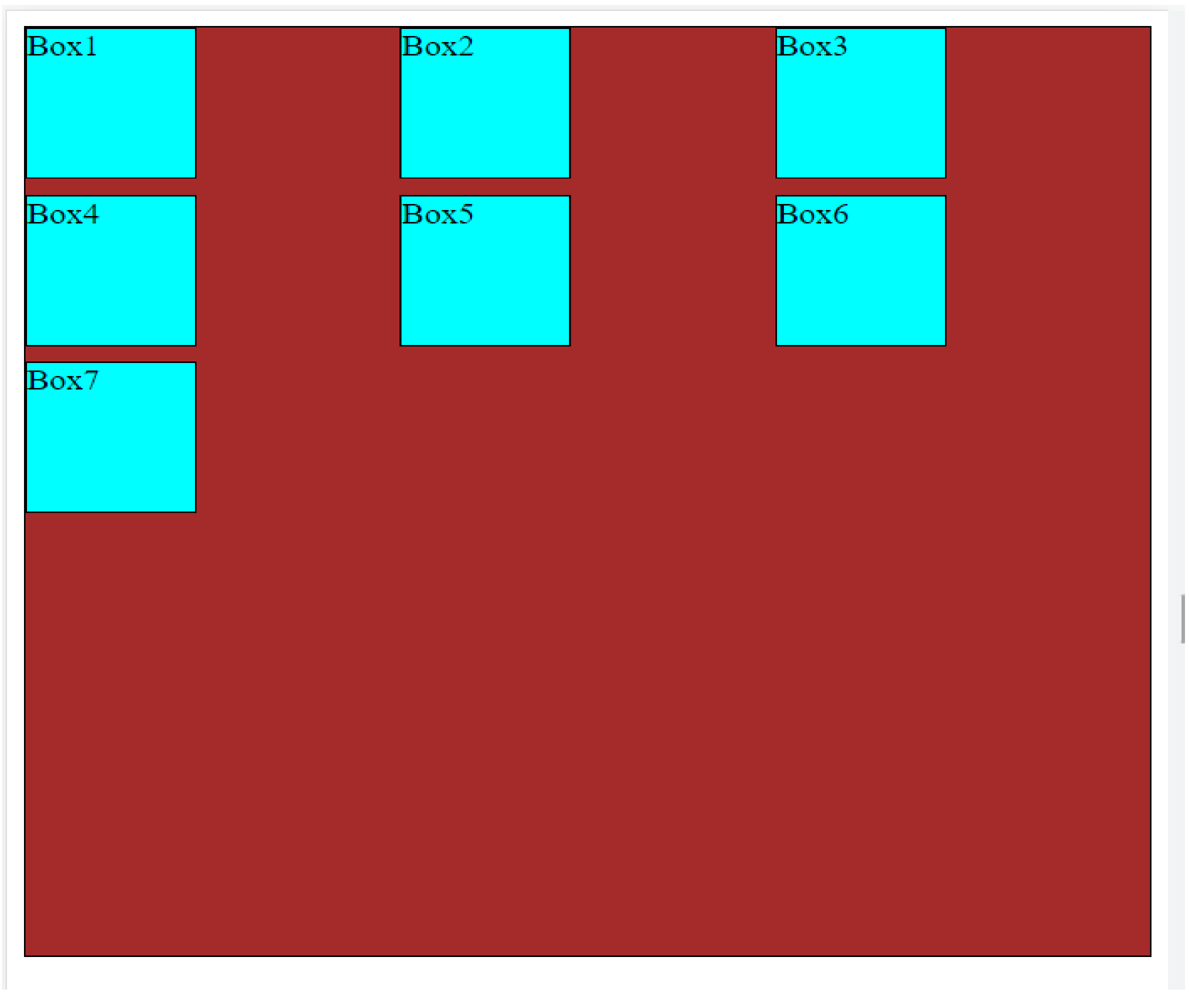
How to create flexible Grid?

We can use flexible unit “fr” to create flexible grid. It is fractional unit and **1fr means 1 part of available space**

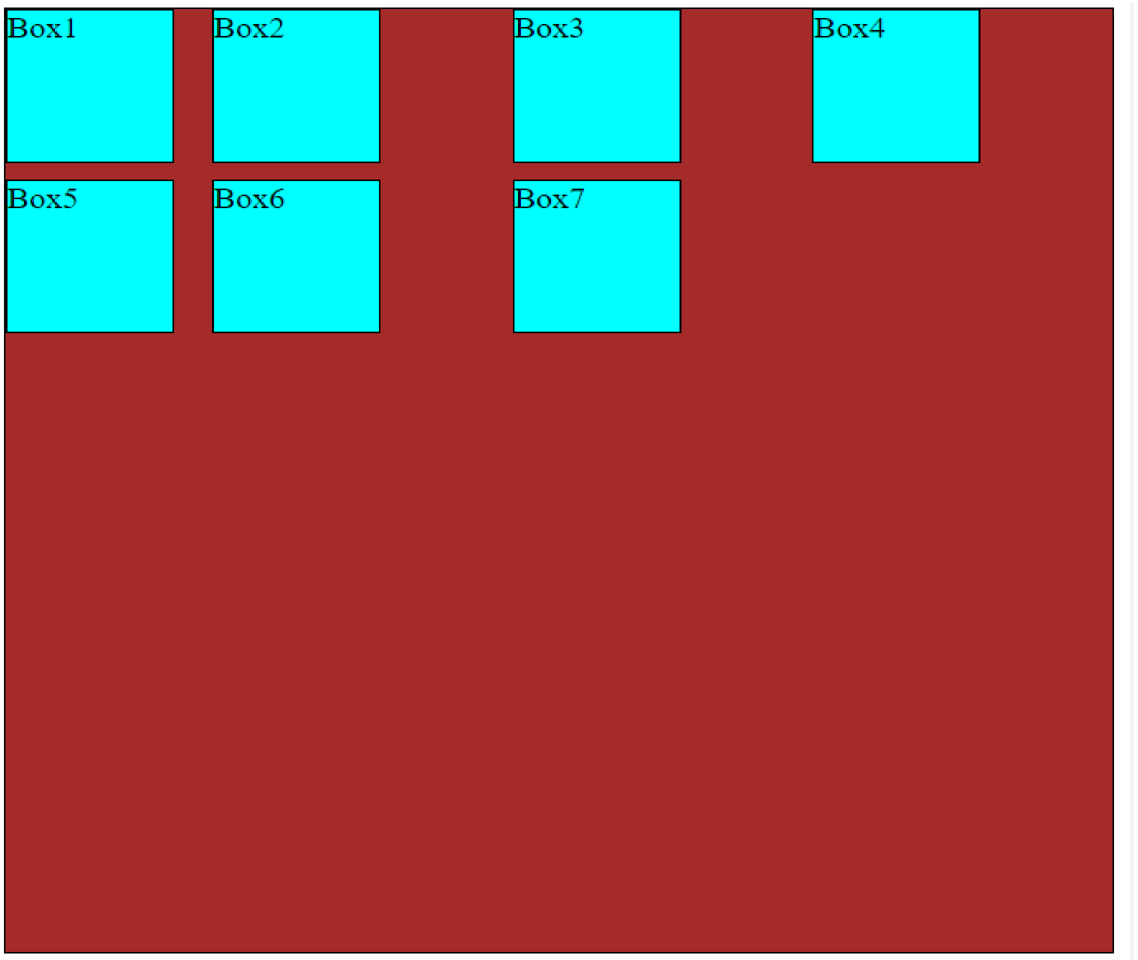
```
grid-template-columns: 1fr 1fr 1fr; //width will divide in 3 equal columns
```

OR

```
grid-template-columns: repeat(3, 1fr); //width will divide in 3 equal columns
```

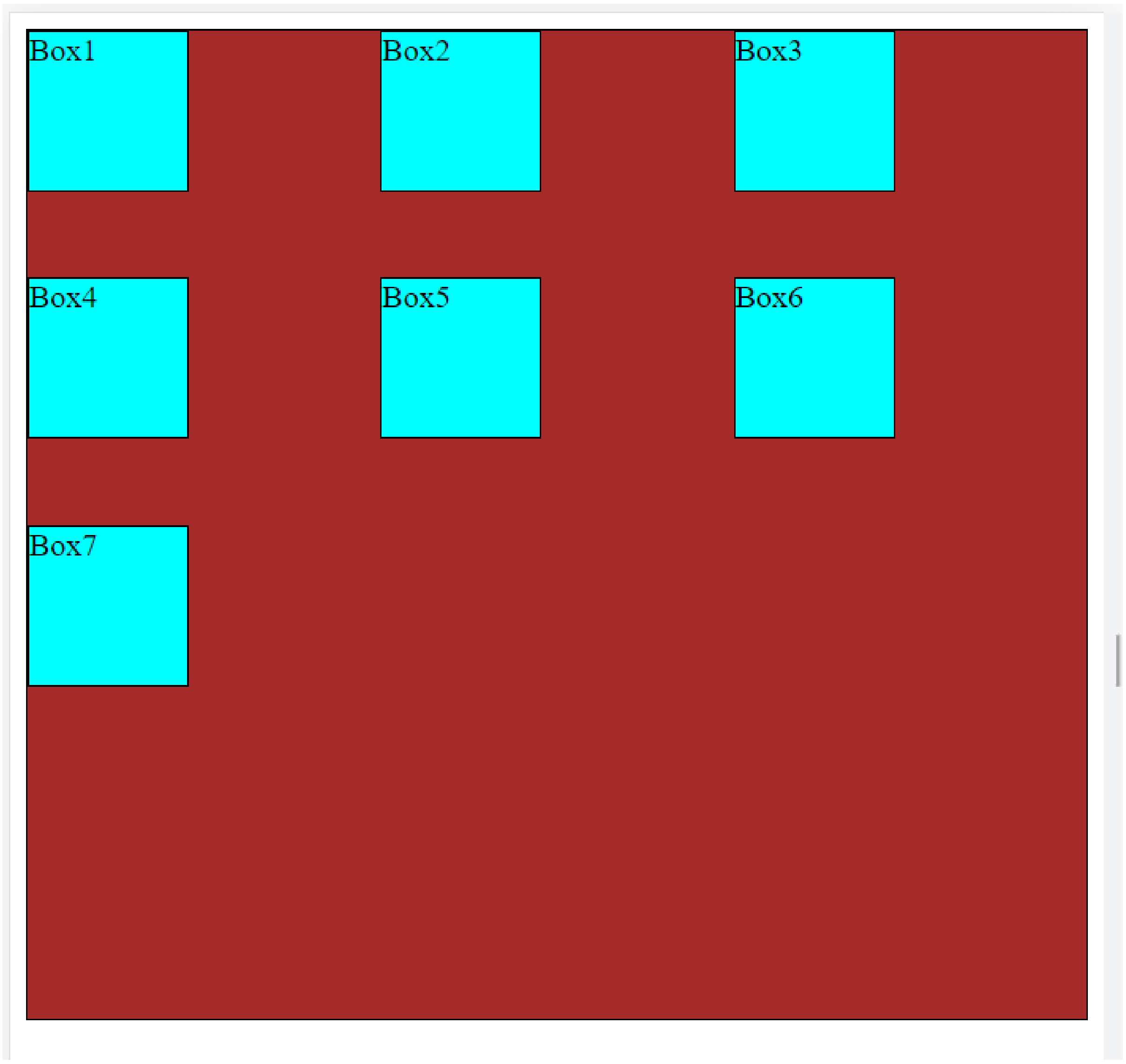

Box1
Box2
Box3
Box4
Box5
Box6
Box7

```
grid-template-columns: 100px repeat(3, 1fr); //first column will be of 100px
and other will be divided in equal parts
```



```
grid-template-columns: 100px repeat(2, 1fr) 90px; //first col will be 100px,
last col 90px and 2 columns middle having same width divided
```

```
 grid-template-columns: repeat(3, 1fr); //three cols having same width
 grid-template-rows: repeat(4, 1fr); //four rows having same width
```

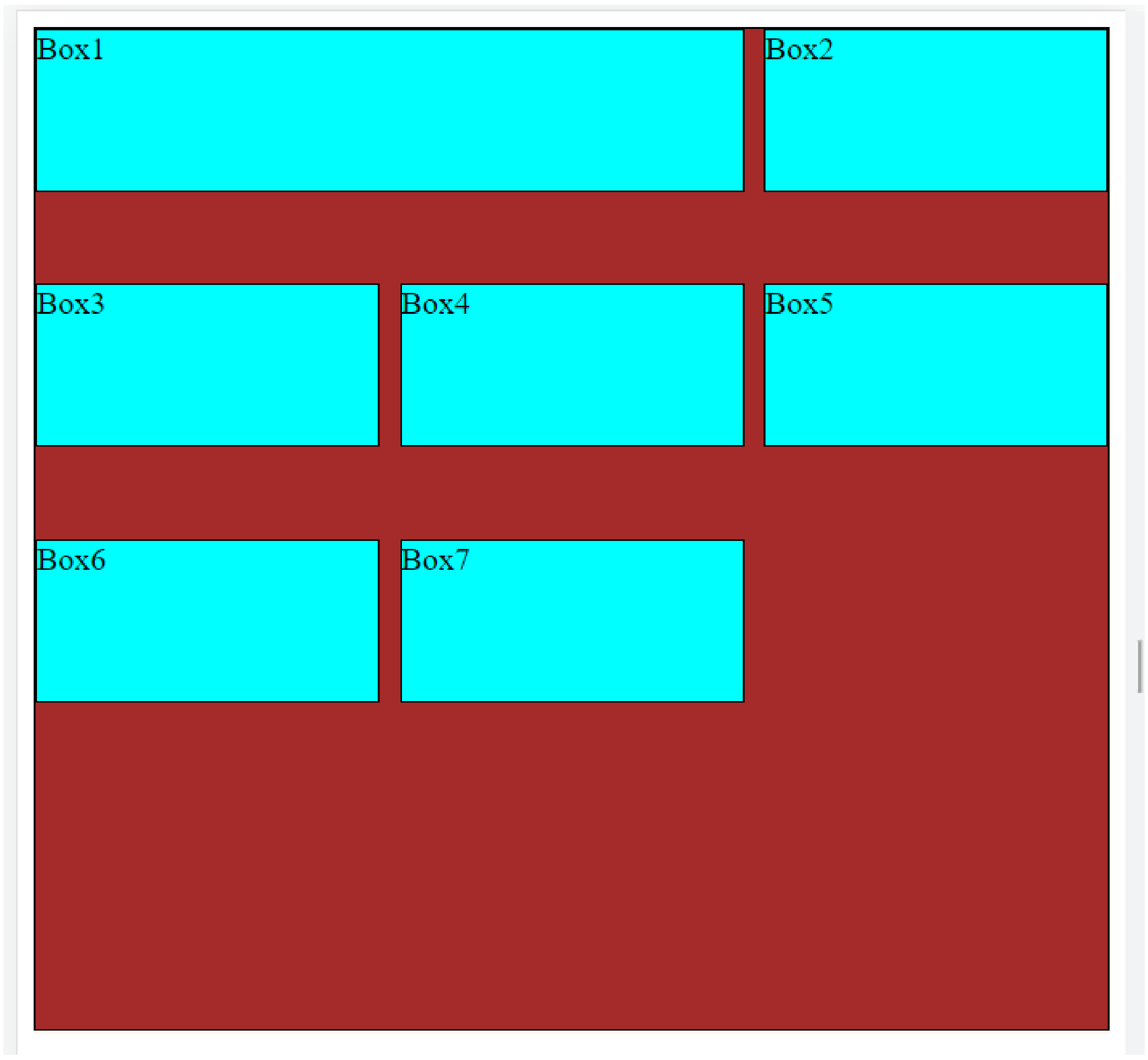


gap – adds gap between rows and columns

```
gap :10px; // add 10px gap
```

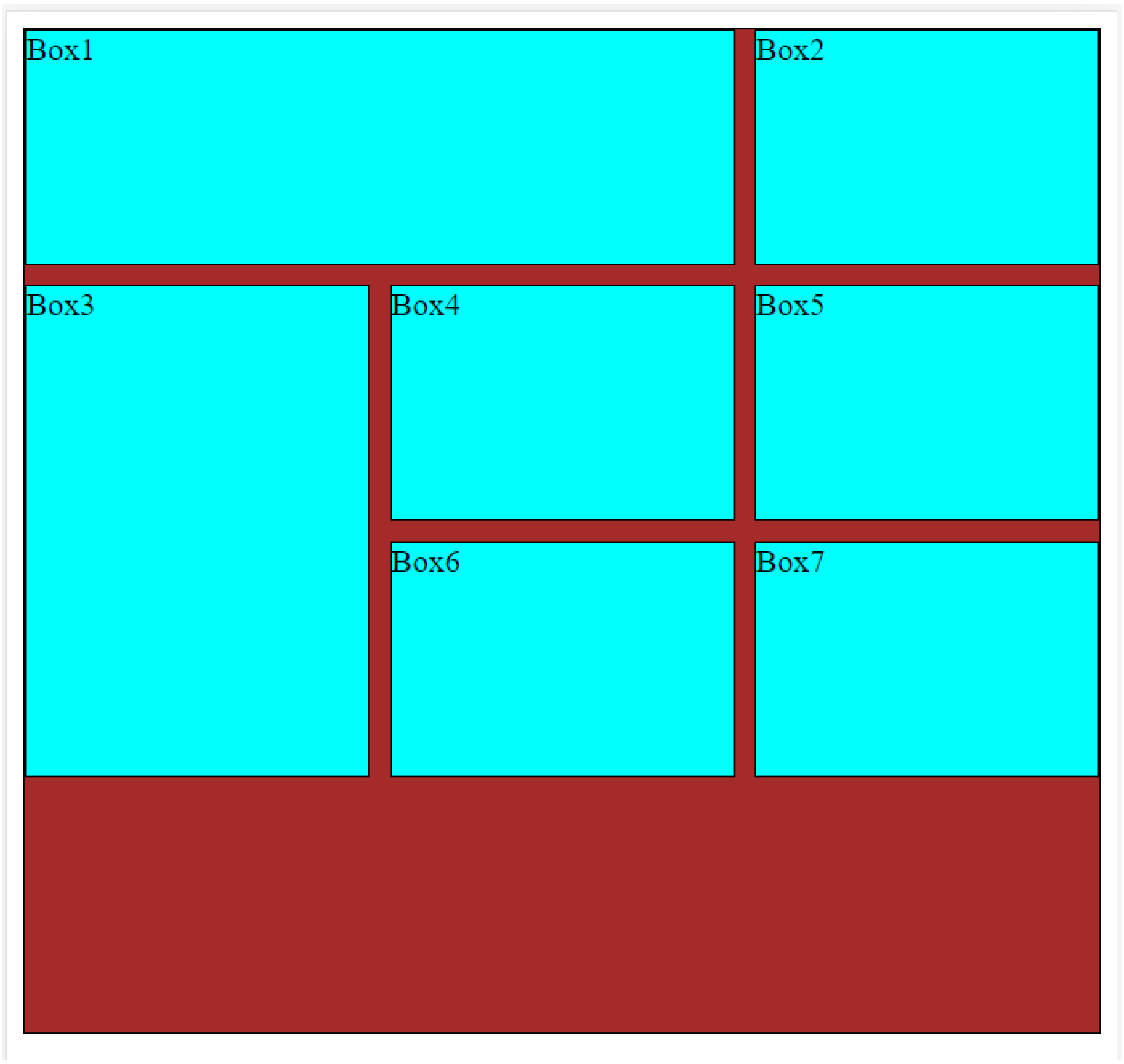
Grid Item Properties

```
#box1{
    grid-column-start: 1;
    grid-column-end: 3 ;
}
Note : Make sure you have set width explicitly for box
```



In above case, In case of overflow of number of boxes, grid layout adjust their height and add new row and items on that new row.

```
#box3{
    grid-row-start: 2;
    grid-row-end: 4;
}
//Make sure you haven’t set height explicitly
```

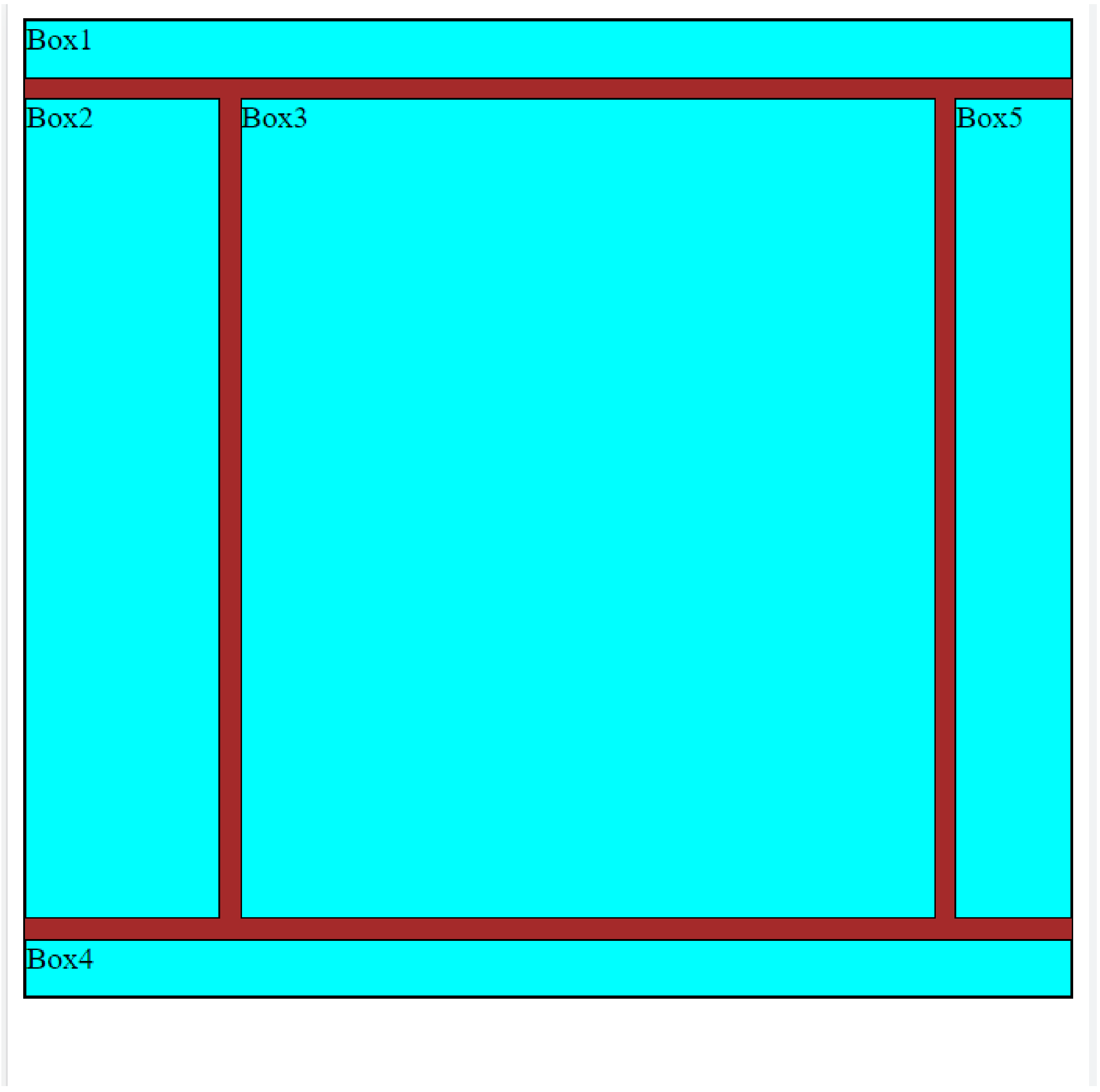


Creating Simple Layout

```css
.container{
    display: grid;
    grid-template-columns: 100px repeat(2, 1fr) 60px;
    grid-template-rows: 30px 1fr 30px;
    gap :10px;
}
```

```css
#box1, #box4{
    grid-column-start: 1;
    grid-column-end: 5;
}
#box2{
    grid-row-start: 2;
    grid-row-end: 3;
}
```

```
#box3{
    grid-column-start: 2;
    grid-column-end: 4;
    grid-row-start: 2;
    grid-row-end: 3;
}
```



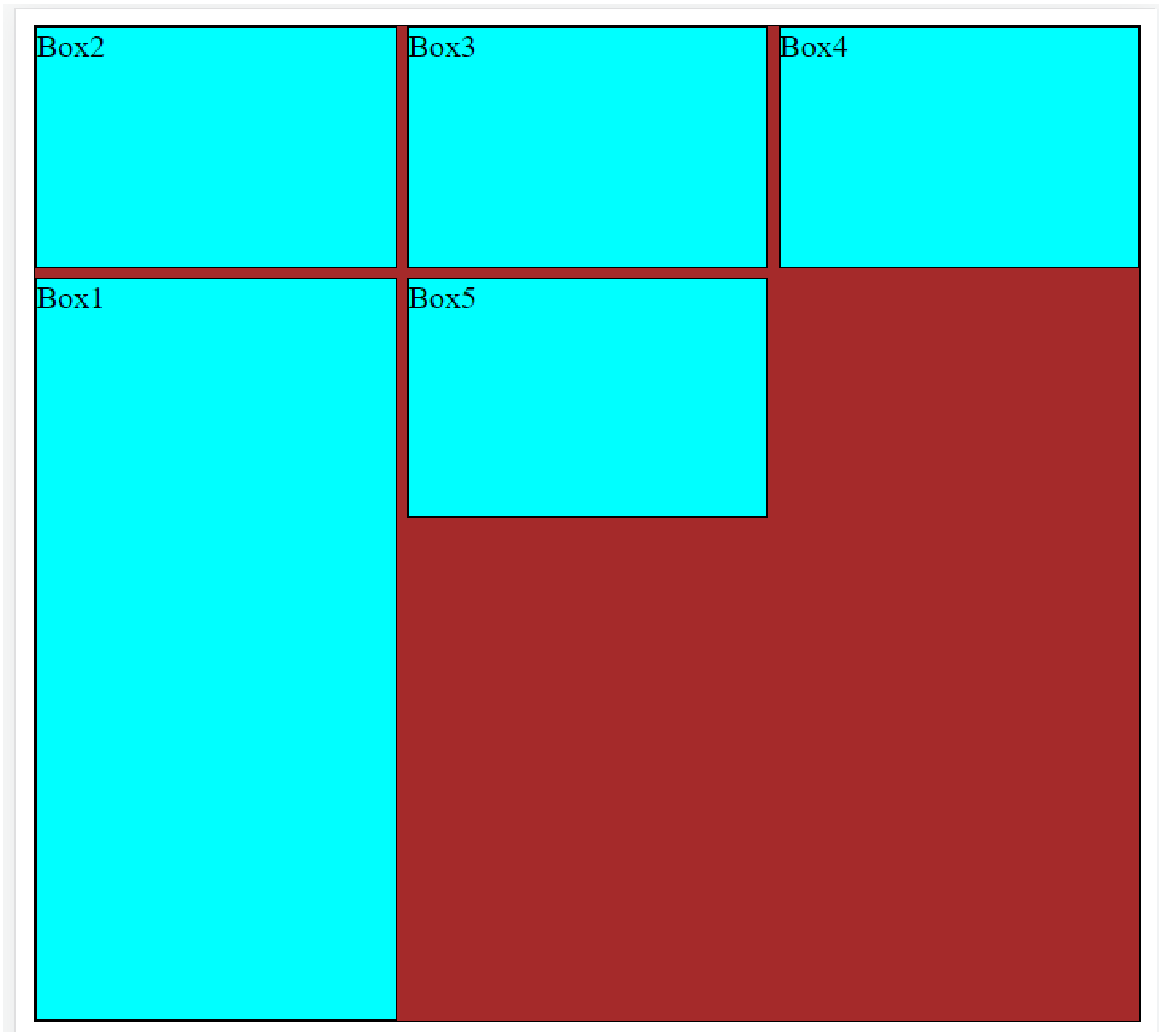
Shorthand Properties:

1) **grid-row : grid-row-start / grid row end ;**
   It is shorthand property for grid-row-start and grid-row-end.
   Grid-row-start specifies on which row to start displaying the item.
   Grid-row-end specifies on which row line to stop displaying the item, or how many rows to span

Example:

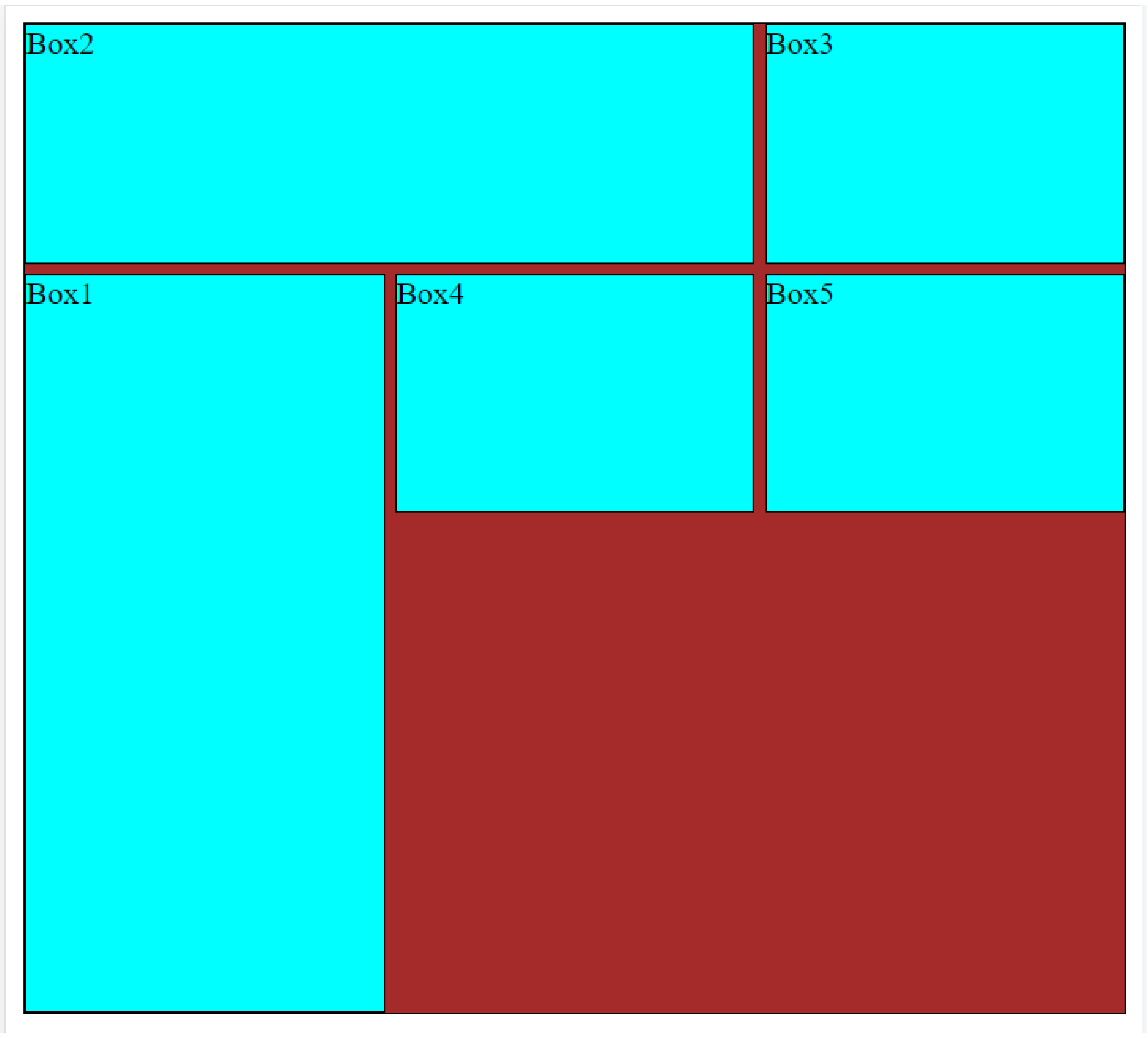
```
 .container{
      grid-template-rows: repeat(4, 1fr); //creating 4 rows
      grid-template-columns: repeat(3, 1fr); //creating 2 cols
      gap: 5px;
}
```

```
#box1{
    /* box1 will start from row2 and will span across 3 rows */
    grid-row: 2 / span 3;
}
```



2) **grid-column- grid-column-start / grid-column-end;**
It is shorthand property for grid-column start and grid-column-end.
Grid-column-start specifies on which column to start displaying an item.
Grid-column-end specifies on which column line to stop displaying an item or how many columns to span.

```
#box2{
    /*box2 will start from column1 and will span across 2 columns */
    grid-column: 1 / span 2;
}
```



Grid-Area

- Grid-area specifies the particular area of rows and columns that grid item occupies.
- Extending particular item in rows and in columns as well.
- It is grid item property.
- It is shorthand property for –

```css
.item{
grid-area: grid-row-start / grid-column-start / grid-row-end / grid-column-
end;
}
```

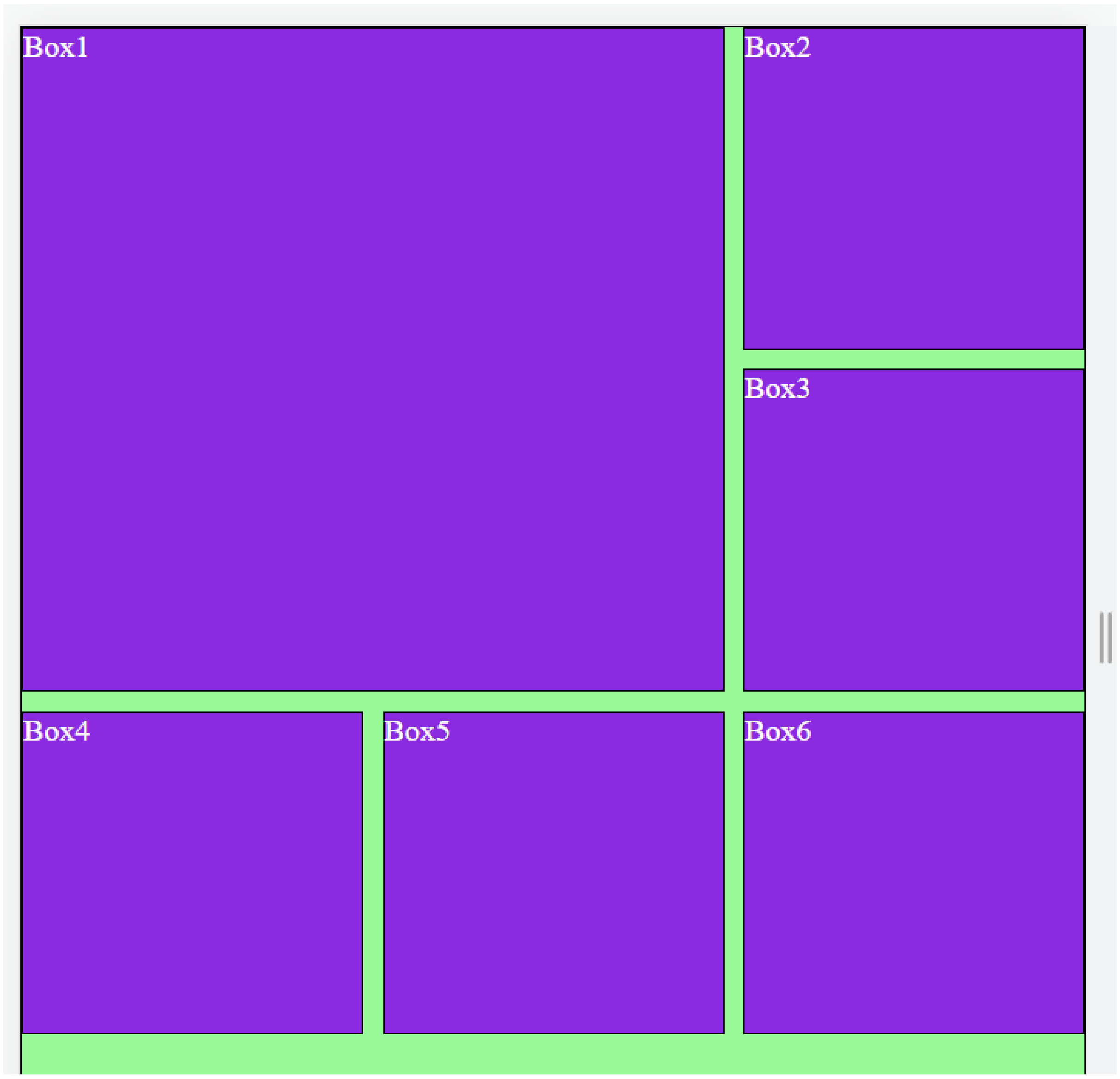
```css
.container{
    height: 700px;
    background-color: palegreen;
    border: 1px solid black;
    display: grid;
    grid-template-columns: repeat(3, 1fr); /* 3 columns */
    grid-template-rows: repeat(4, 1fr); /* 4 rows */
    gap: 10px;
}
```

```css
.box{
    background-color: blueviolet;
    border: 1px solid black;
    color: white;
    /* width: 100px;
    height: 100px; */
}
```

```css
#box1{
    /* grid-row-start: 1;
    grid-row-end: 3;
    grid-column-start: 1;
    grid-column-end: 3; */

    grid-area: 1/1/3/3;
}
```

Expanding first item in 2 columns and 2 rows as below, and other items shift to block



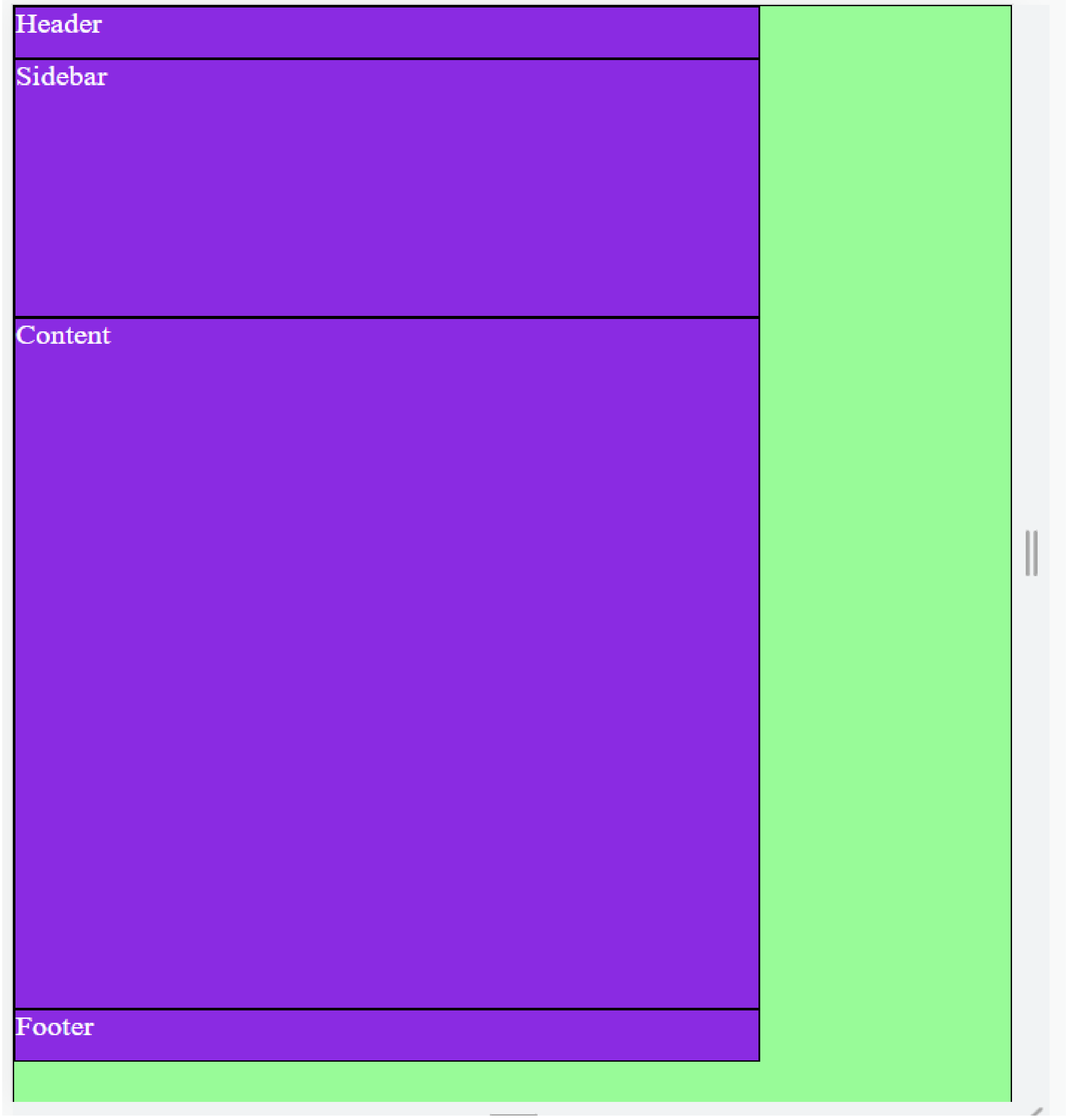
Grid-Template-Areas

- grid-template-areas property is used for naming rows and columns in grid and set it’s layout.
- It is container property.

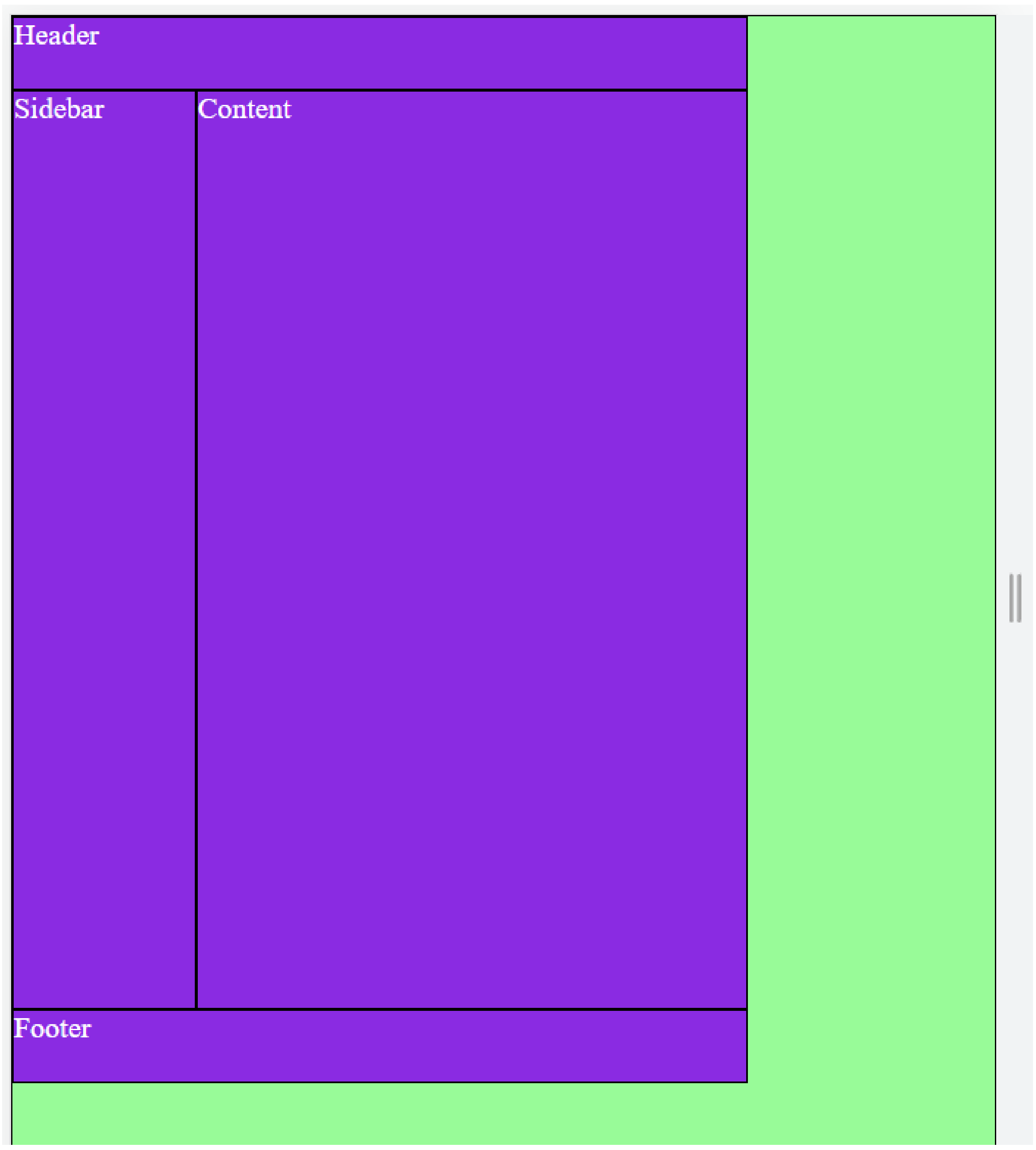
```
.container{
  grid-template-areas:
    "hd"
    "side"
    "main"
    "ft";
}
```

In HTML, there are 4 div created having below ID's

```
#header{
    grid-area: hd;
}
#sidebar{
    grid-area: side;
}
#content{
    grid-area: main;
}
#footer{
    grid-area: ft;
}
```



```
.container{
    grid-template-rows: 40px 500px 40px;
    grid-template-columns: 100px 300px;
    grid-template-areas:
    "hd hd"
    "side main"
    "ft ft";
}
//grid-area for items is defined same as above
```



Advanced Grid Concepts

1) Fr Unit – fr is a fractional unit and 1fr is 1 part of available space

2) Repeat function –

repeat (repeat_count, size) - function represent repeated fragment of cells.

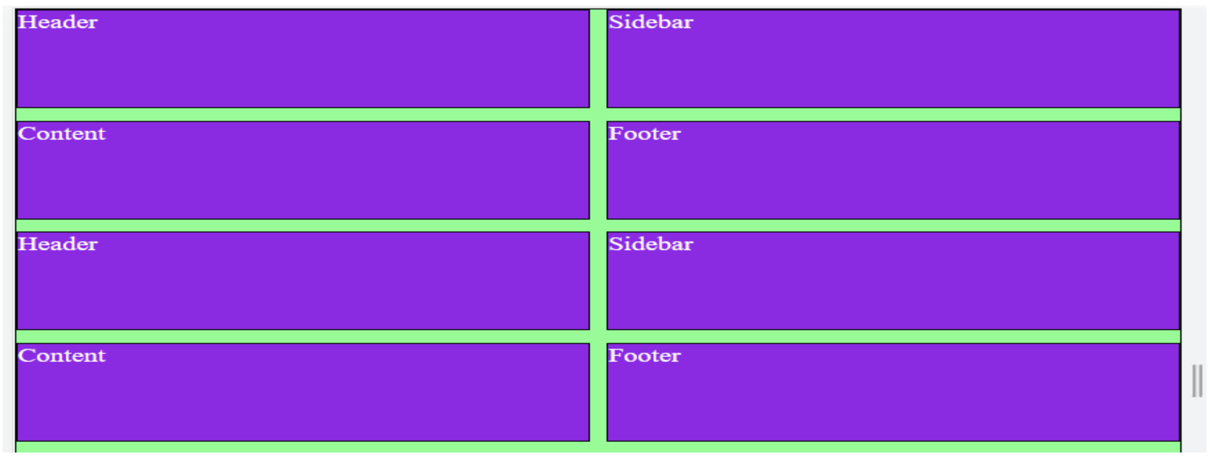
Example –

grid-template-columns: 1fr repeat (2, 60px); - first column will take 1 part of available space and other two columns will be of 60px each

3) Grid-auto-rows: minmax () –

- This property sets the size for the rows in grid container.
- This property affects only rows with the size not set. (We can use grid-template-rows property at the same time). Use either of them.
- Automatically rows gets add. When there are unknown number of rows then we can use this property.
- Minmax() – using this we can specify what will be the minimum size and maximum size of these rows.

```
.container{
    grid-auto-rows: minmax(80px, auto);
    grid-template-columns: repeat(2, 1fr);
    gap: 10px;
}
/* minimum size of each row will be 80px and maximum will be equal to
content*/
/* No row will be create having size less than 80px */
/* If number of rows increase then rows will overflow from the container */
```



Grid Properties

Justify-content -

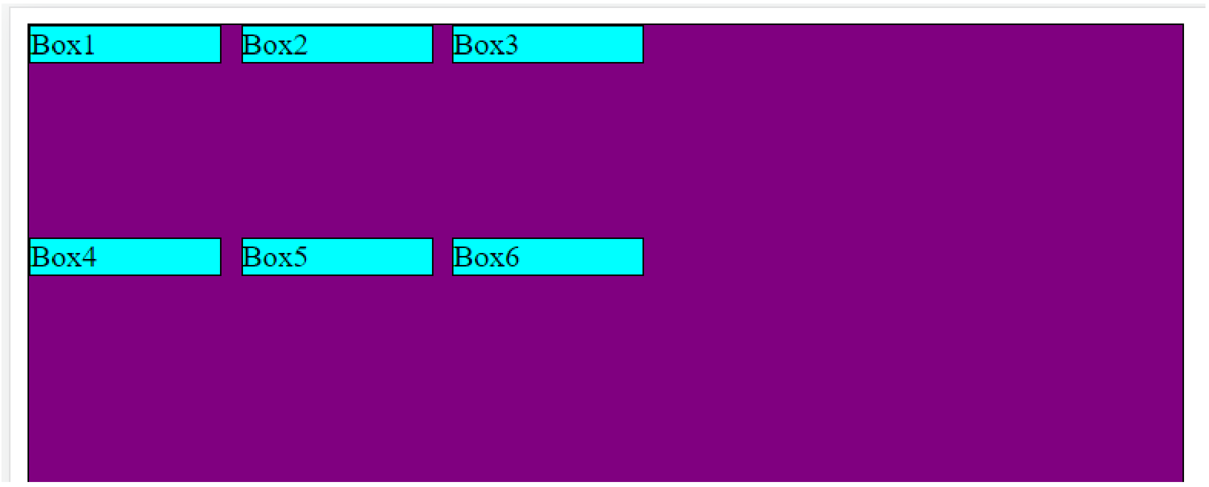
- Aligns the content on main axis ie Horizontal axis
- This property is applied on parent/container
- Works exactly same as it works in flexbox.

```
    justify-content: start;
    justify-content: end;
    justify-content: center;
    justify-content: space-around;
    justify-content: space-between;
    justify-content: space-evenly;
```

align-items –

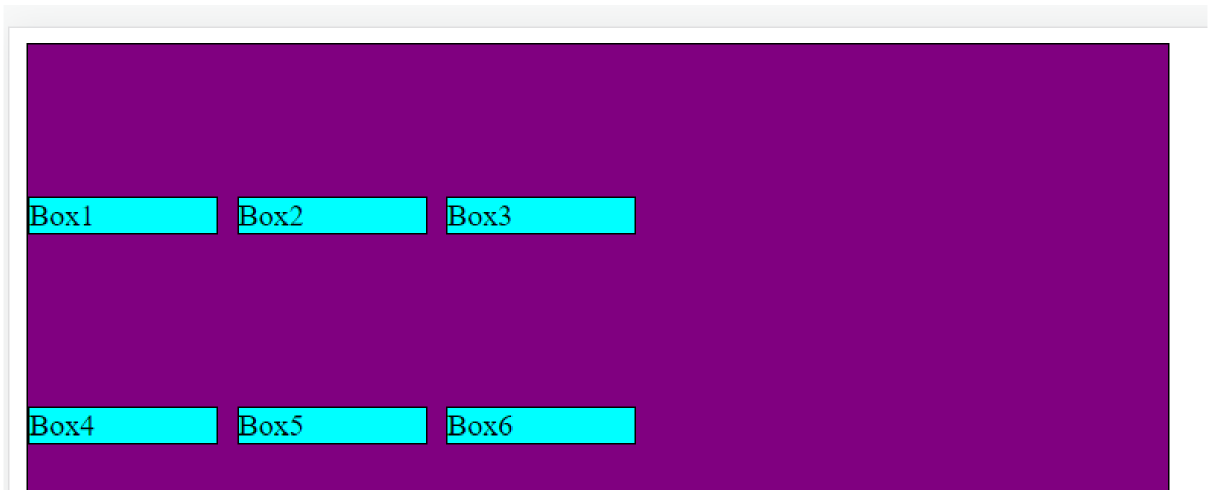
- This property is applied on parent/container
- Aligns all the items wrt the box in which they are placed in.
- According to cross/vertical axis Example -

```
align-items: flex-start; // Items will be placed at the start of box in which they are in
```



Similarly, It works for

```
  align-items: flex-end;
```

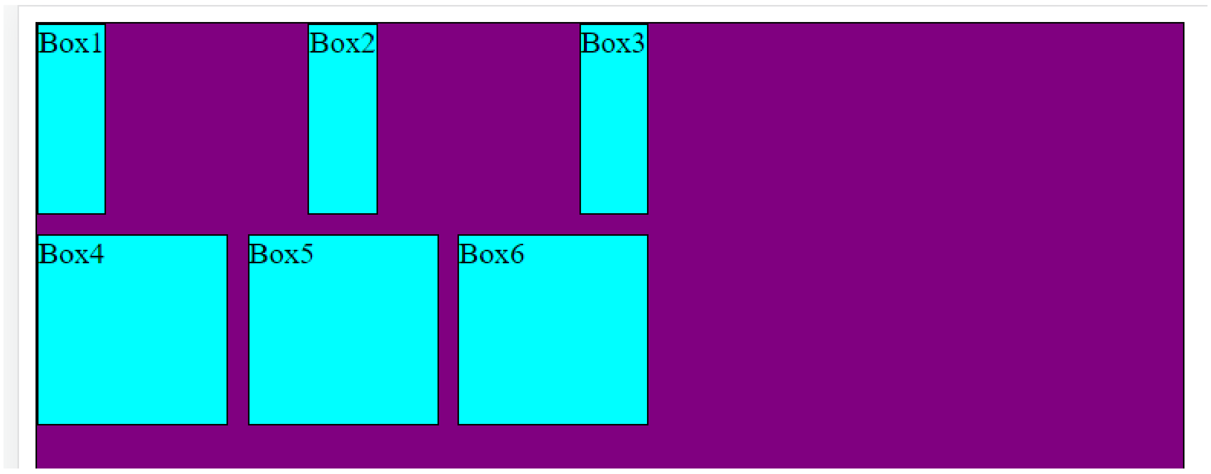


```
align-items: center; // place the item at the middle of box
align-items: baseline;// align the item according to the text inside it
align-items: stretch; //default – stretch the item in complete box
```

justify-self –

- This property is applied on particular item or child of grid container
- It helps to align the content in the particular item with respect to the box in which they are in.
- According to horizontal axis.

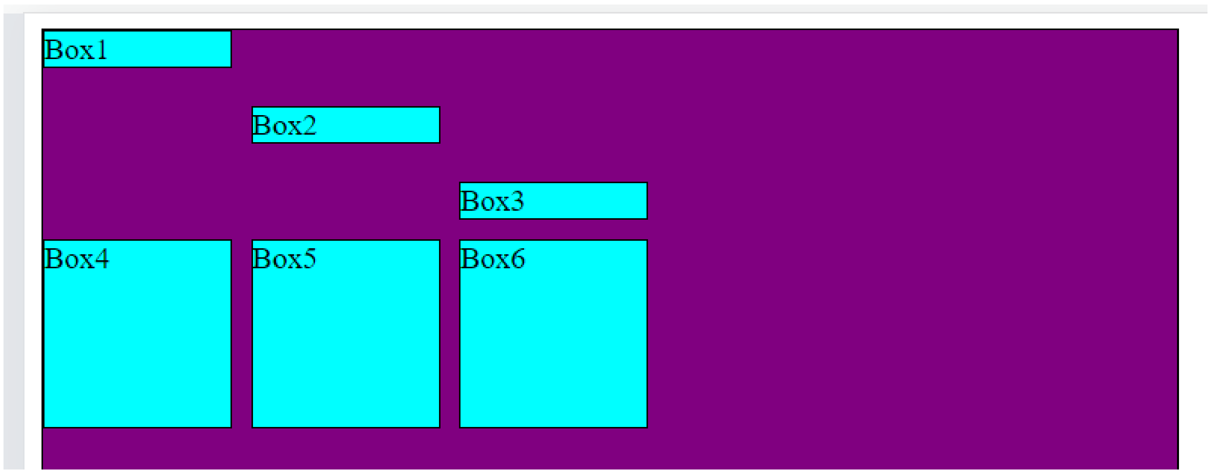
```
#box1{
   justify-self: start;
}
#box2{
    justify-self: center
}
#box3{
    justify-self: end;
}
```



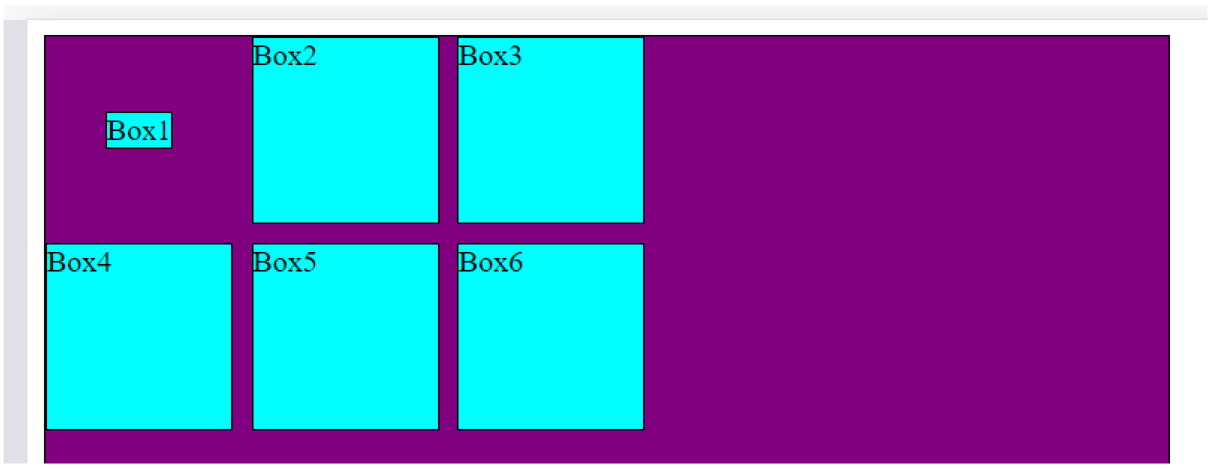
align-self –

- align the particular item with respect to the box it is in according to vertical axis

```
#box1{
   align-self: flex-start;
}
#box2{
    align-self: center;
}
#box3{
    align-self: flex-end;
}
```



```
#box1{
   justify-self: center;
   align-self: center;
}
//place the item at the center if box
```



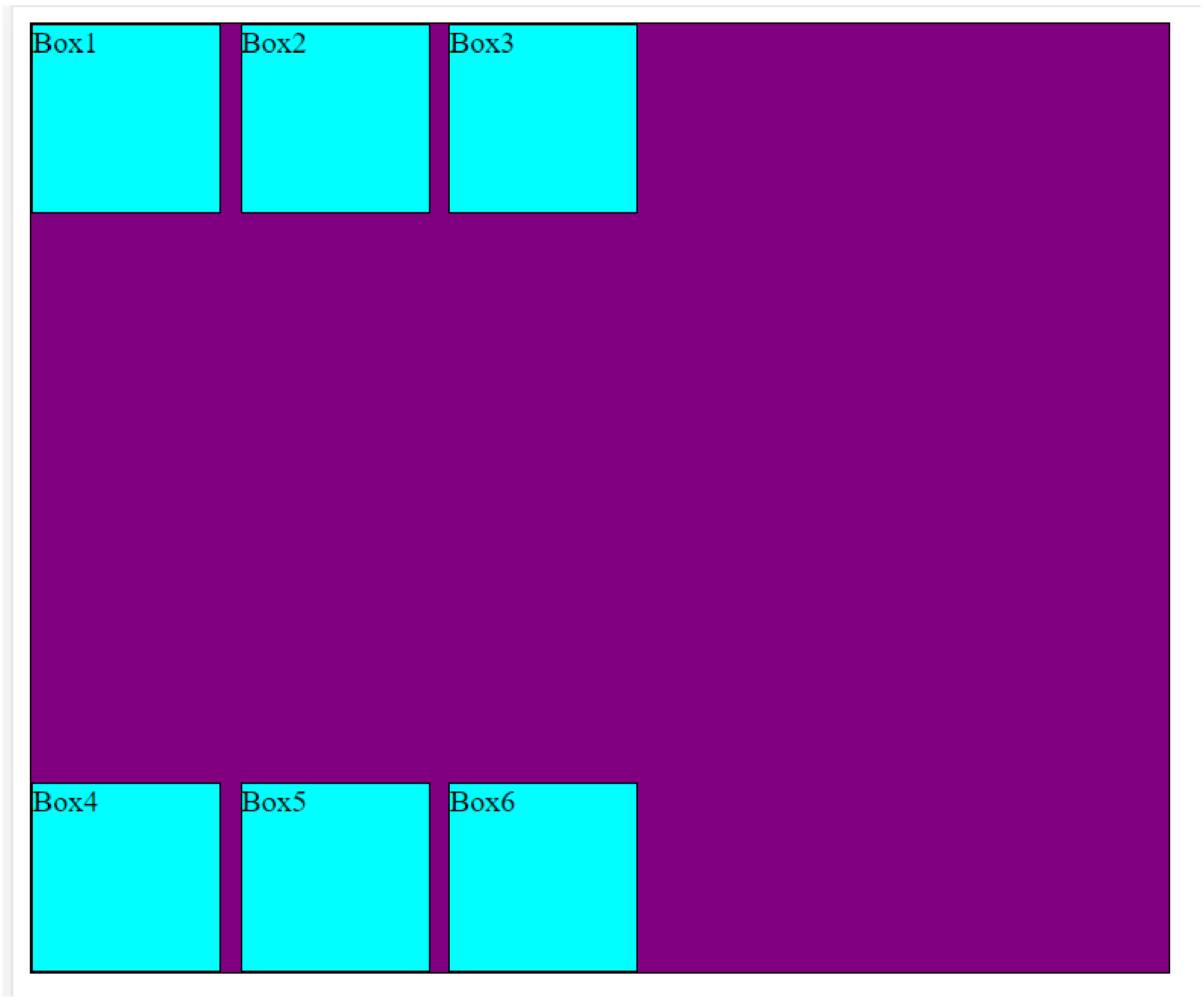
align-content –

This is container property

It place all the items according to cross axis

Works same as It works in flexbox (Distance between rows)

```
align-content: space-between;
```



```
    align-content: flex-start;
    align-content: flex-end;
    align-content: center;
    align-content: space-around;
    align-content: space-evenly;
//works exactly same as it works in flexbox
```

justify-items –

- This is container property,
- It works same like justify-self – only difference justify-self is item property and works for particular item and justify-items works for all the items.

place-items

- shorthand property for align-items and justify-items

place-self

- shorthand property for justify-self and align-self

Difference between Grid and Inline-grid

- In HTML, the difference between a grid layout and an inline-grid layout is the way in which the elements are displayed on the page.

- A grid layout uses CSS to create a grid of rows and columns, and the elements are placed within the cells of the grid. This allows for precise control over the position and size of the elements within the grid.

- An inline-grid layout, on the other hand, displays the elements as if they were inline elements (i.e. they are placed on the same line as other elements and flow with the text), but they still follow the grid layout rules for positioning and sizing.

- In general, if you want to create a layout that looks like a table, with rows and columns, you would use the grid layout. If you want the elements to flow with the text but still be positioned according to a grid layout, you would use the inline-grid layout.