CSS Grids Intermediate

To make sure that we are able to provide the grid configuration in CSS properly, we can use a couple of utility functions

repeat

If we want to give the number of tracks (when we say track we are referring to column or rows) and size of each track in a shorter way we can use the repeat function. It takes 2 values:

- Number of tracks
- Size of each track -> Size can be given in any units i.e. px, em, fr etc

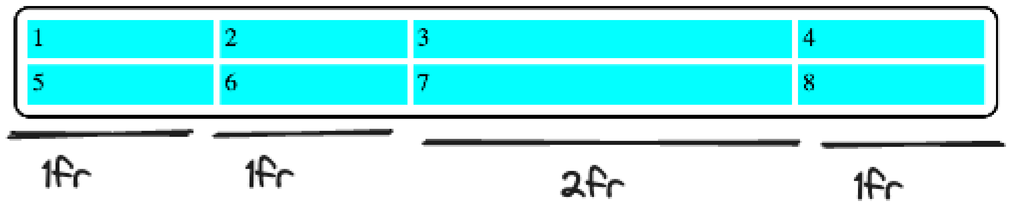
```css
grid-template-columns: repeat(5, 1fr);
```

So the above piece of code gives 5 columns (as track here is the column due to the template-columns property) and size of each column is 1fr.

In case we don't want to have same size for all the tracks we can give custom configuration also, and use `repeat` function wherever we have consecutive same values.

```css
grid-template-columns: repeat(2, 1fr) 2fr 1fr;
```

Here, we will be having 4 columns, with first two columns of size `1fr` then third column of size `2fr` and then last one with `1fr` again.

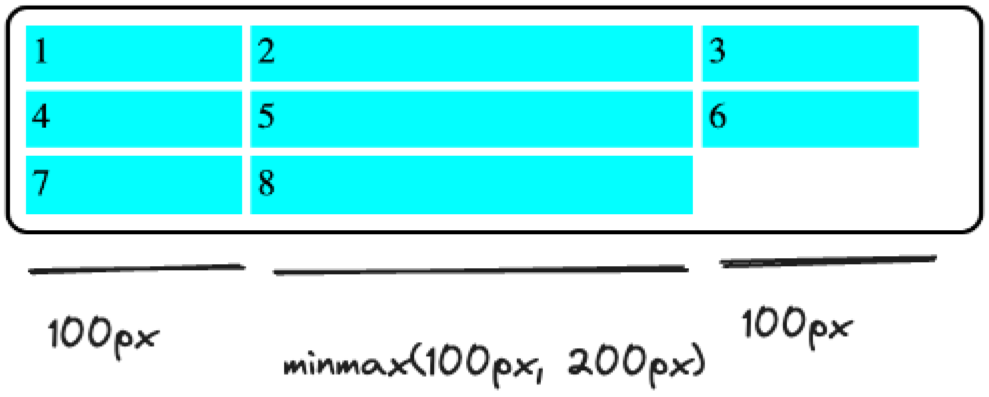


minmax

So the minmax function helps us to decide minimum and maximum track sizes. Because we might be having a lot of situations where we want to restrict the minimum size and maximum size of a column or a row, in those cases minmax function helps.
It takes 2 values:

- Minimum size of the track i.e. at-least how much should be the size of the track
- maximum size of the track i.e. at-most how much should be the size of the track

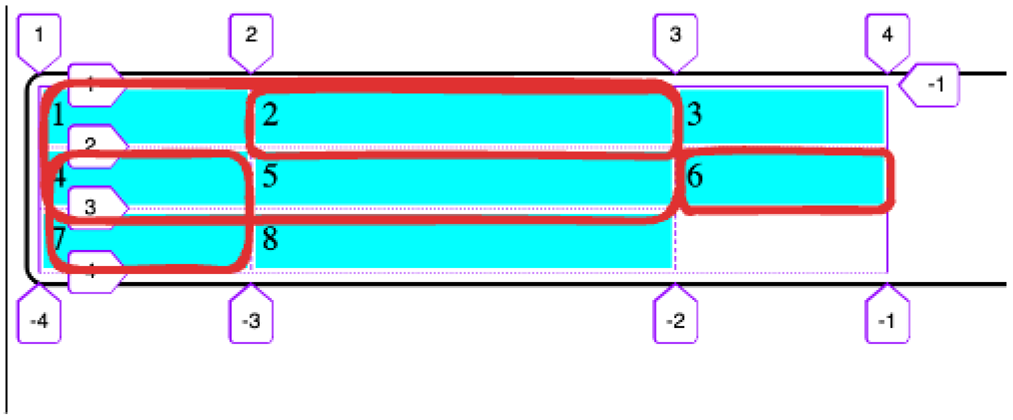
```
grid-template-columns: 100px minmax(100px, 200px) 100px;
```



Important Advanced CSS Grid Concepts

CSS Grid area

Inside the grid / matrix of the grid container any area / region surrounded by grid lines is called as grid area.
In simple terms any sub-rectangle / sub-grid / rectangular cross-section inside the CSS grid container is called as grid area.



Any of the above (and more) highlighted sub-rectangles are our grid area.

Why do we need this concept ?
So there a very interesting CSS rule to control the styling of a grid-area which is `grid-template-areas`. Using this property we can select a particular `grid-area` and give it a different style

grid-template-areas

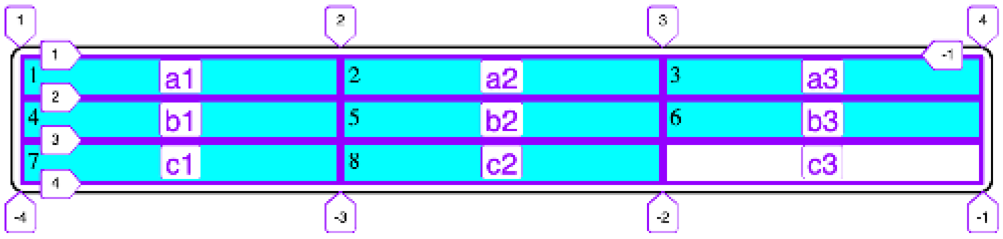
This CSS property helps us to give names to our CSS grid areas in the CSS grid.
Then these names can be later used to provide certain type of styling to the elements.

To use this property, we have to provide `mxn` number of names (at-most) row by row inside a pair of double quotes for each.

```
grid-template-columns: 1fr 1fr 1fr;
grid-template-areas:
        "a1 a2 a3"
        "b1 b2 b3"
        "c1 c2 c3";
```

In the above piece of code, we get a grid of 3 columns and 3 rows.
Now in the `grid-template-areas` property, we have put 3 pair of double quotes, each representing one row.
So,

- `"a1 a2 a3"` is the first row
- `"b1 b2 b3"` is the second row
- `"c1 c2 c3"` is the third row row



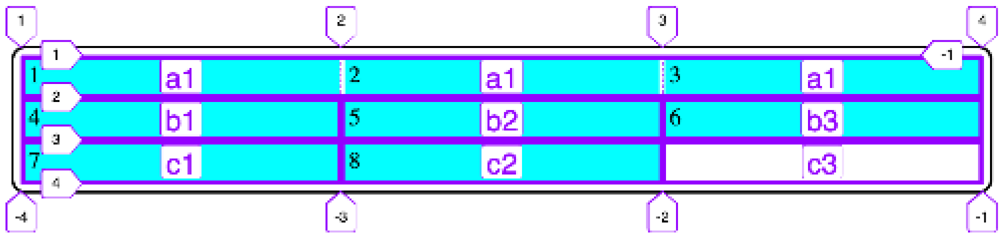
Now, here we can see that each `grid-area` is being highlighted with purple colour border.
Although there is no data in the c3 area still it will be highlighted.

If we change the configuration the grid areas can change.

Example 1

```
grid-template-columns: 1fr 1fr 1fr;
grid-template-areas:
        "a1 a1 a1"
        "b1 b2 b3"
        "c1 c2 c3";
```

Now in this piece of code, the whole first row is given the same name, hence instead of having 3 different sub-rectangle, we have the whole row as 1 sub-rectangle with the area name as `a1`.

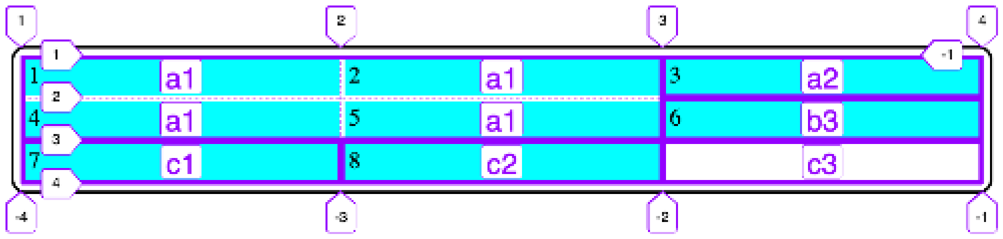


Now we can also visualise that, the purple border is now coming to the complete first row.

Example 2

```
grid-template-columns: 1fr 1fr 1fr;
grid-template-areas:
        "a1 a1 a2"
        "a1 a1 b3"
        "c1 c2 c3";
```

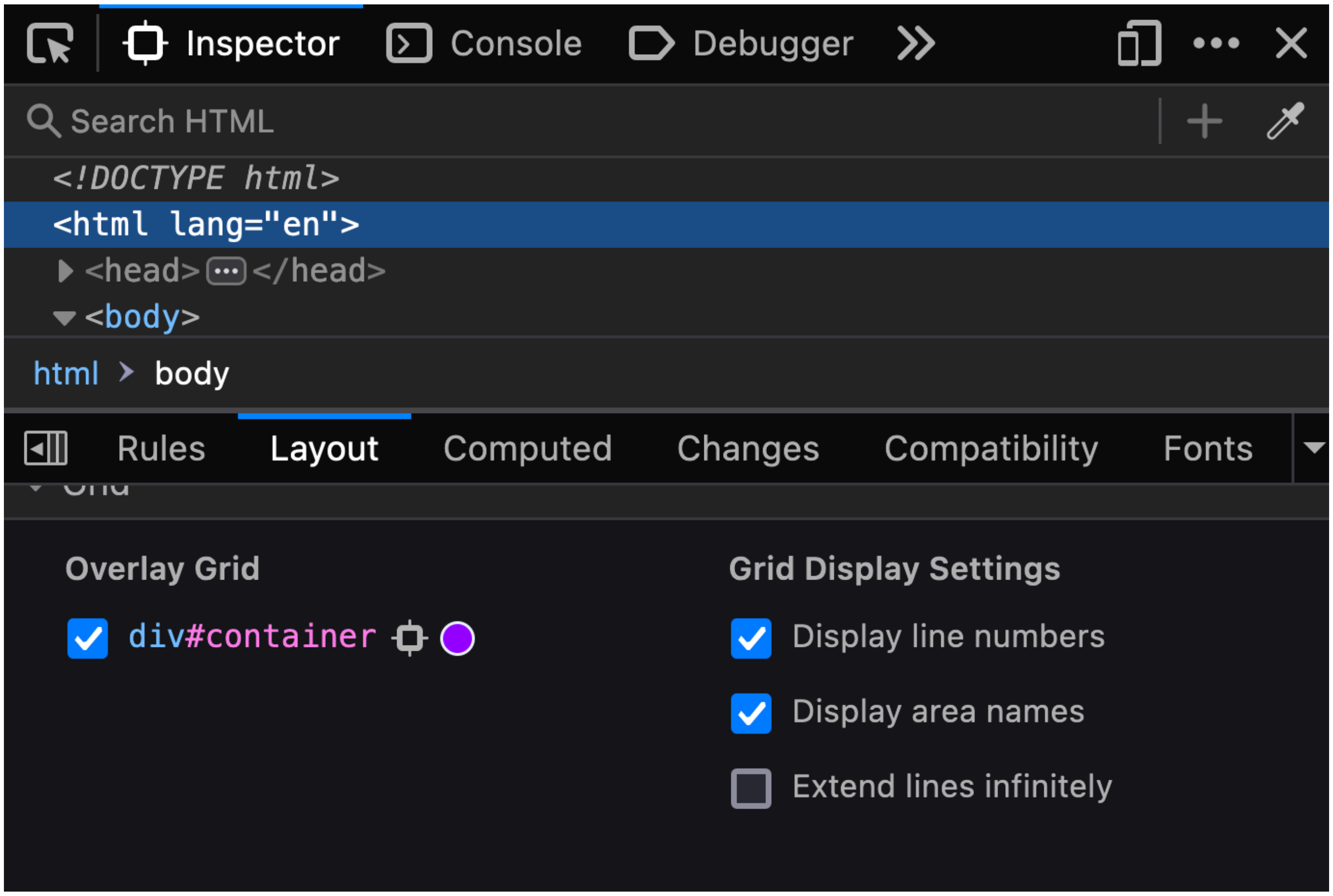
Now in this piece of code, the first row , second row, and first two columns is given the same name, we have the whole region as 1 sub-rectangle.



Now we can also visualise that, the purple border is now coming to the complete first box.

Note:

If we want to enable grid area names and lines we can do this with our dev tools.



```
<!DOCTYPE html>
<html lang="en">
  <head>⋯</head>
  <body>
```

Overlay Grid

- [✓] div#container

Grid Display Settings

- [✓] Display line numbers
- [✓] Display area names
- [ ] Extend lines infinitely

We need to go to the layout tab, and then go to the grid section and enable `Overlay Grid`. Then we can enable display line numbers and area names to get the lines and areas.

Using grid areas for styling

We can select any element in the grid, and then in the CSS rules use the `grid-area` to give a specific area and define the styles.

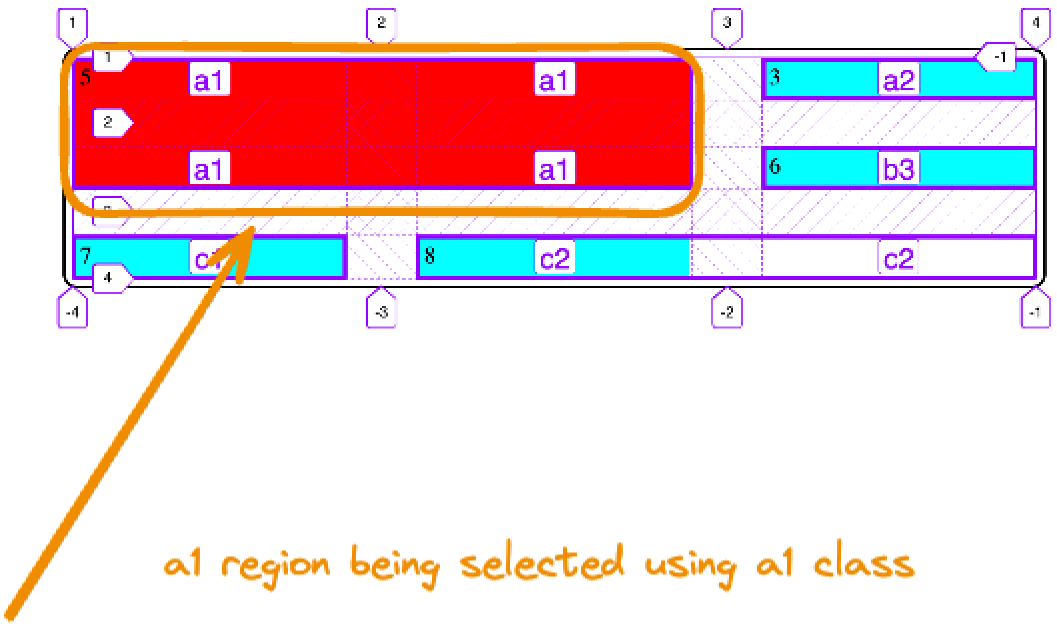
```
grid-template-columns: 1fr 1fr 1fr;
grid-template-areas:
        "a1 a1 a2"
        "a1 a1 b3"
        "c1 c2 c2";
grid-gap: 2rem 3rem;
```

Here the gap property introduces gaps in rows and columns of cells of the grid.

```
.a1 {
        grid-area: a1;
        background-color: red;
}
```

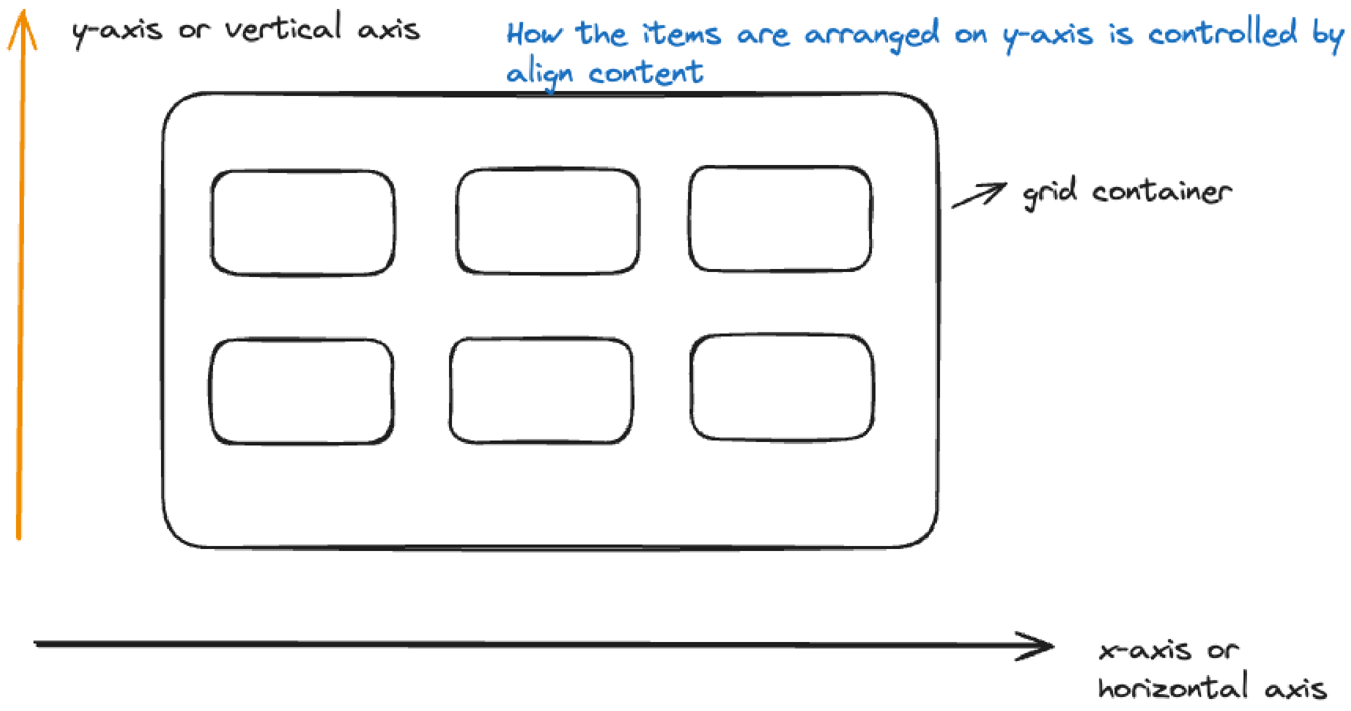
```
<div class="item a1">
        1
</div>

<div class="item a1">
        2
</div>

<div class="item">
        3
</div>

<div class="item a1">
        4
</div>

<!--- Other divs -->
```

Here we have selected the divs with `a1` class and then allocated them the styles for `a1` area.



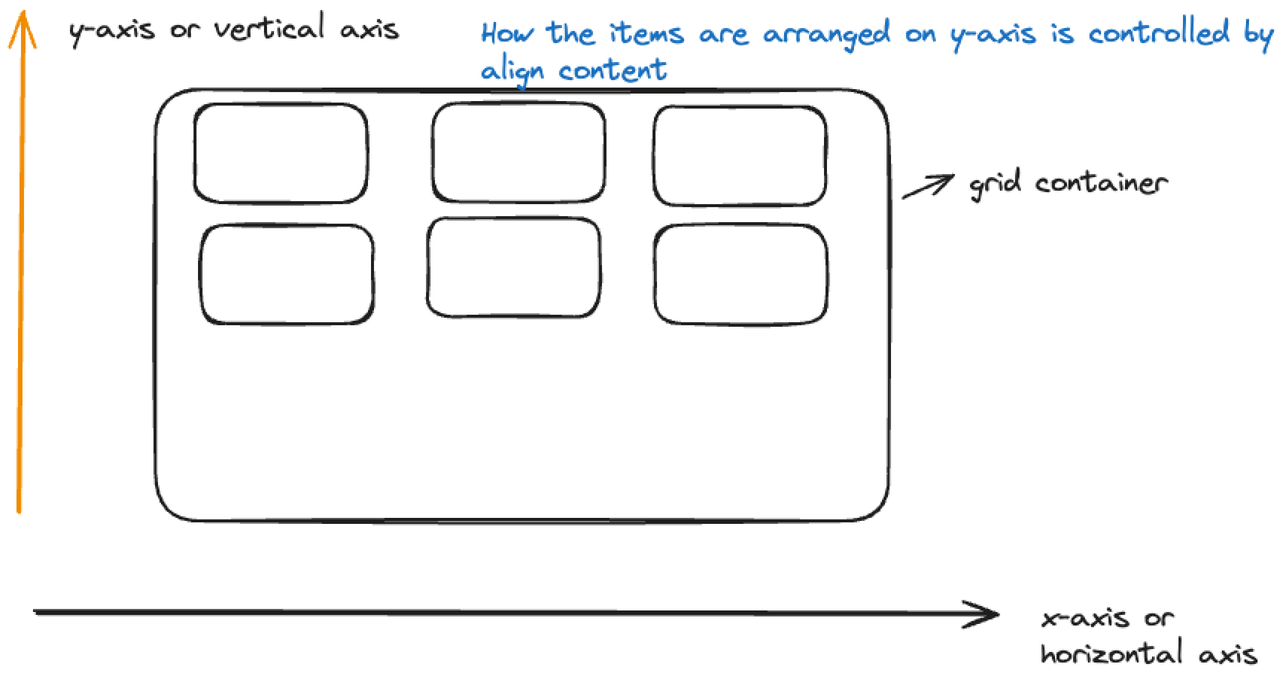
align-content

This property helps us to control the alignment or arrangement or distribution of the grid-items inside the container along the vertical axis (y-axis).

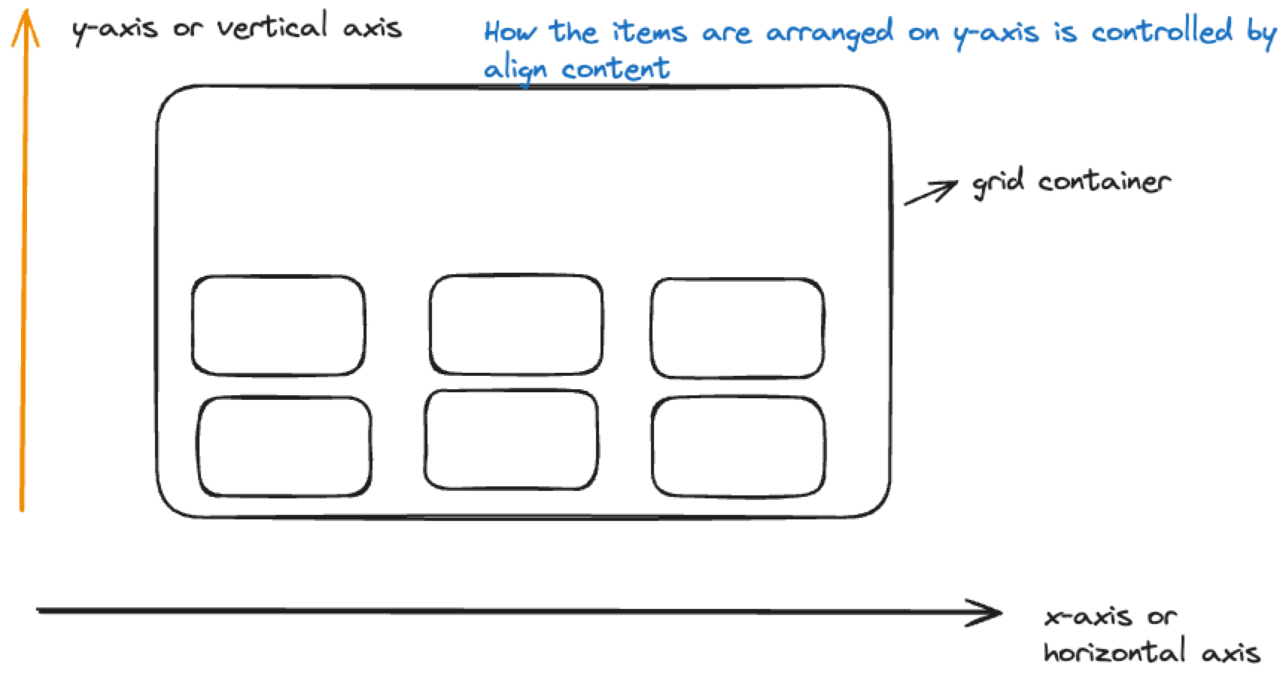

y-axis or vertical axis
How the items are arranged on y-axis is controlled by align content
grid container
x-axis or horizontal axis

So by default it's value is equal to `stretch` but we can change it to :

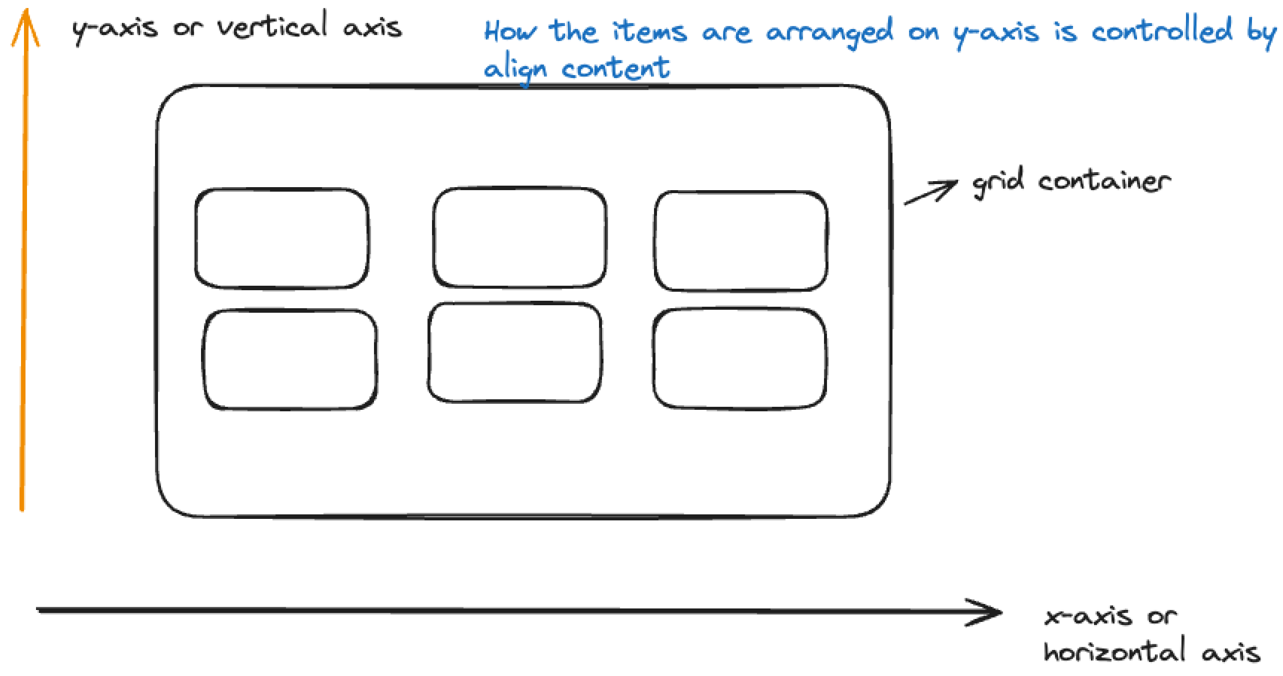
- **start** -> with this all the items are going to take their required height based on content and are pushed to the start / top of the y-axis


y-axis or vertical axis
How the items are arranged on y-axis is controlled by align content
grid container
x-axis or horizontal axis

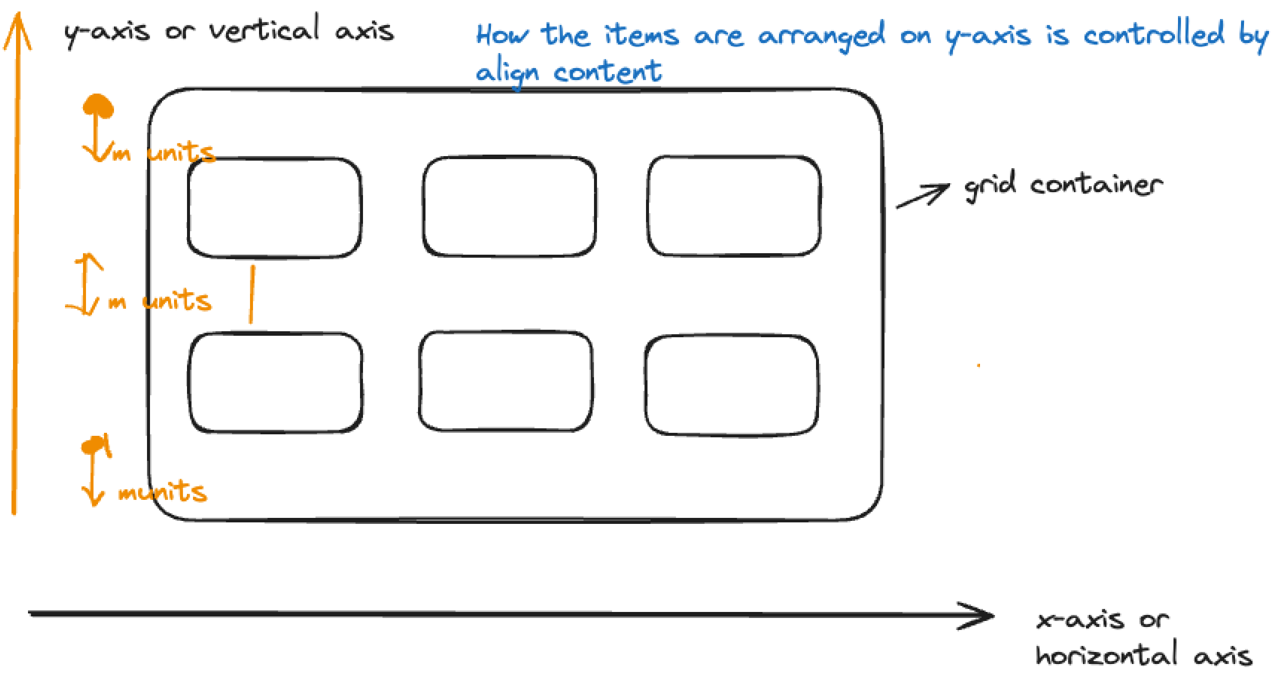
- **end** -> with this all the items are going to take their required height based on content and are pushed to the end / last of the y-axis


y-axis or vertical axis
How the items are arranged on y-axis is controlled by align content
grid container
x-axis or horizontal axis

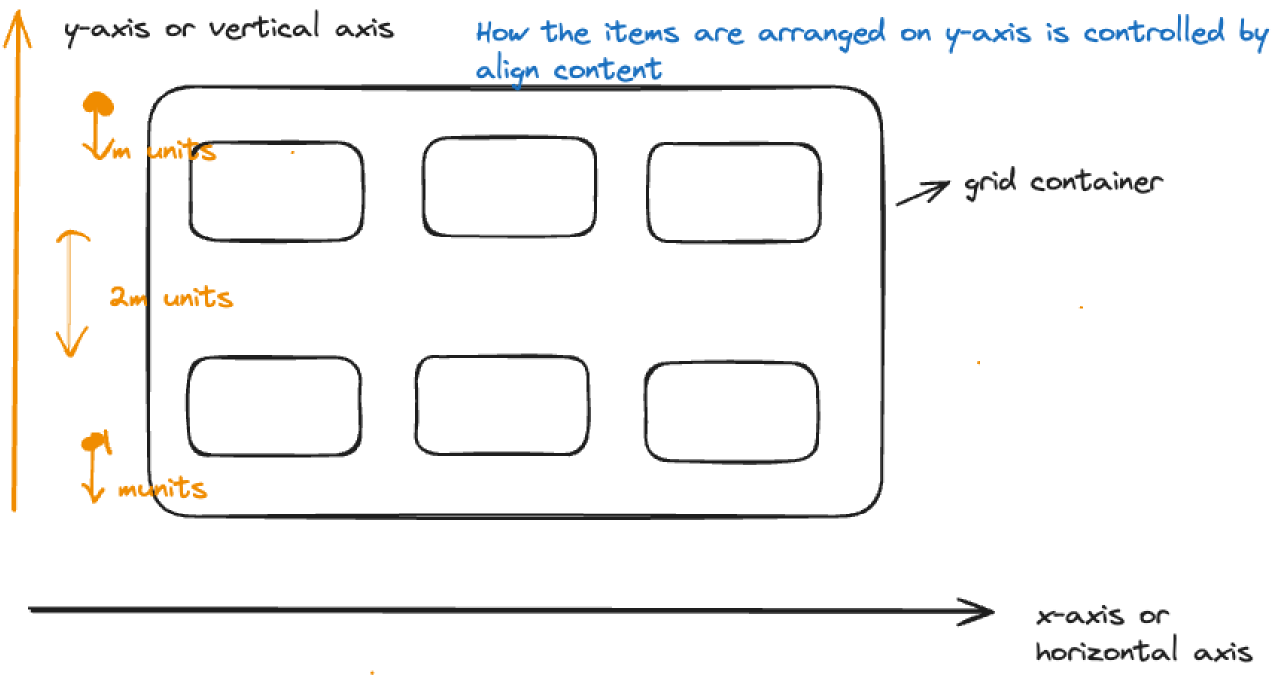
- **center** -> with this all the items are going to take their required height based on content and are pushed to the center of the y-axis


y-axis or vertical axis
How the items are arranged on y-axis is controlled by align content
grid container
x-axis or horizontal axis

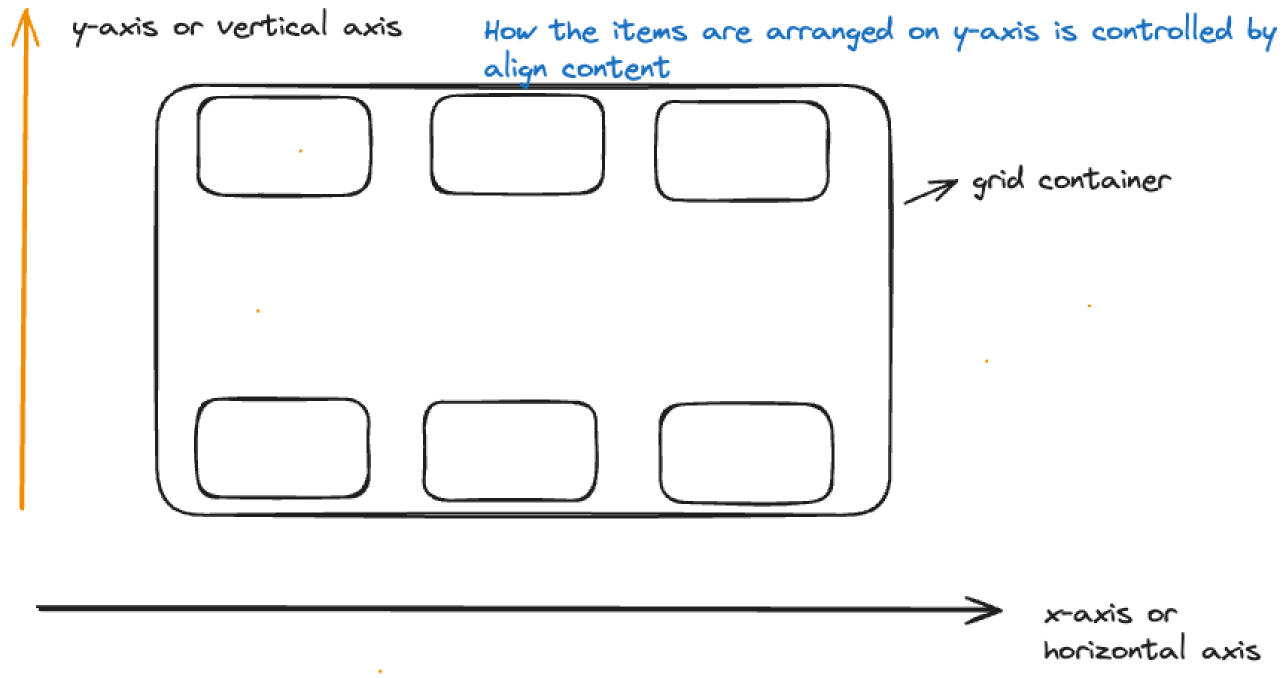
- **space-evenly** -> with this all the items are going to take their required height based on content and are pushed with equal spacing on top, bottom and between on the y-axis


y-axis or vertical axis
How the items are arranged on y-axis is controlled by align content
m units
grid container
m units
m units
x-axis or horizontal axis

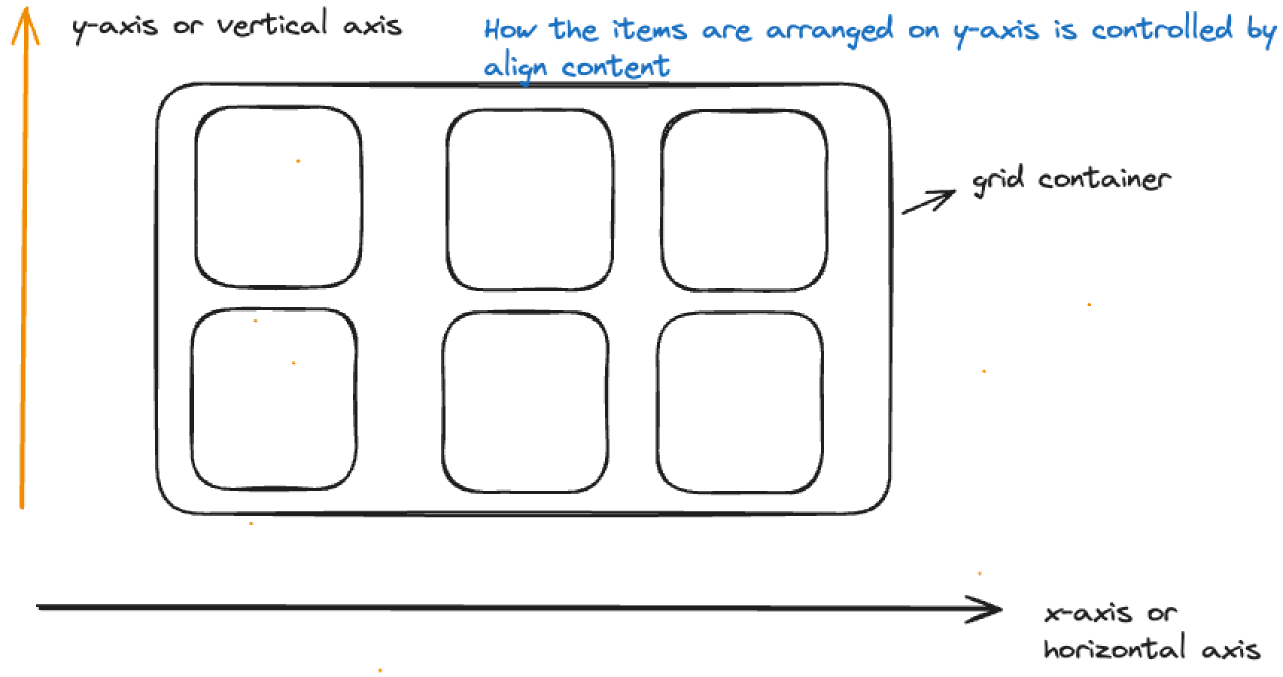
- **space-around** -> with this all the items are going to take their required height based on content and are pushed with double the distance in between when compared to top and bottom on the y-axis


y-axis or vertical axis
How the items are arranged on y-axis is controlled by align content
m units
grid container
2m units
m units
x-axis or horizontal axis

- **space-between** -> with this all the items are going to take their required height based on content and are pushed as far as possible on the y-axis. Here we have maximum possible distance between the rows.

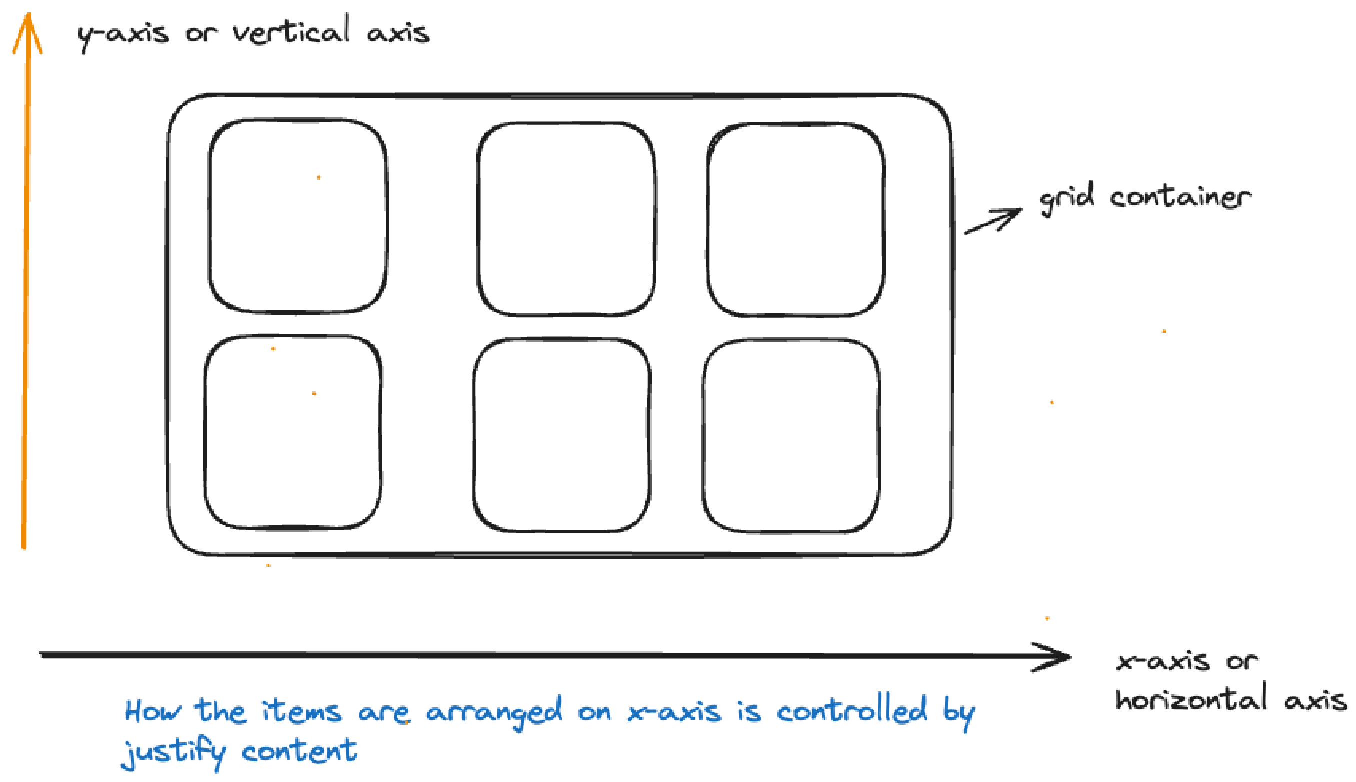

y-axis or vertical axis
How the items are arranged on y-axis is controlled by align content
grid container
x-axis or horizontal axis

- **stretch** -> This is the default value, here all the rows take as much space as possible and items stretch their height to fill the space.


y-axis or vertical axis
How the items are arranged on y-axis is controlled by align content
grid container
x-axis or horizontal axis

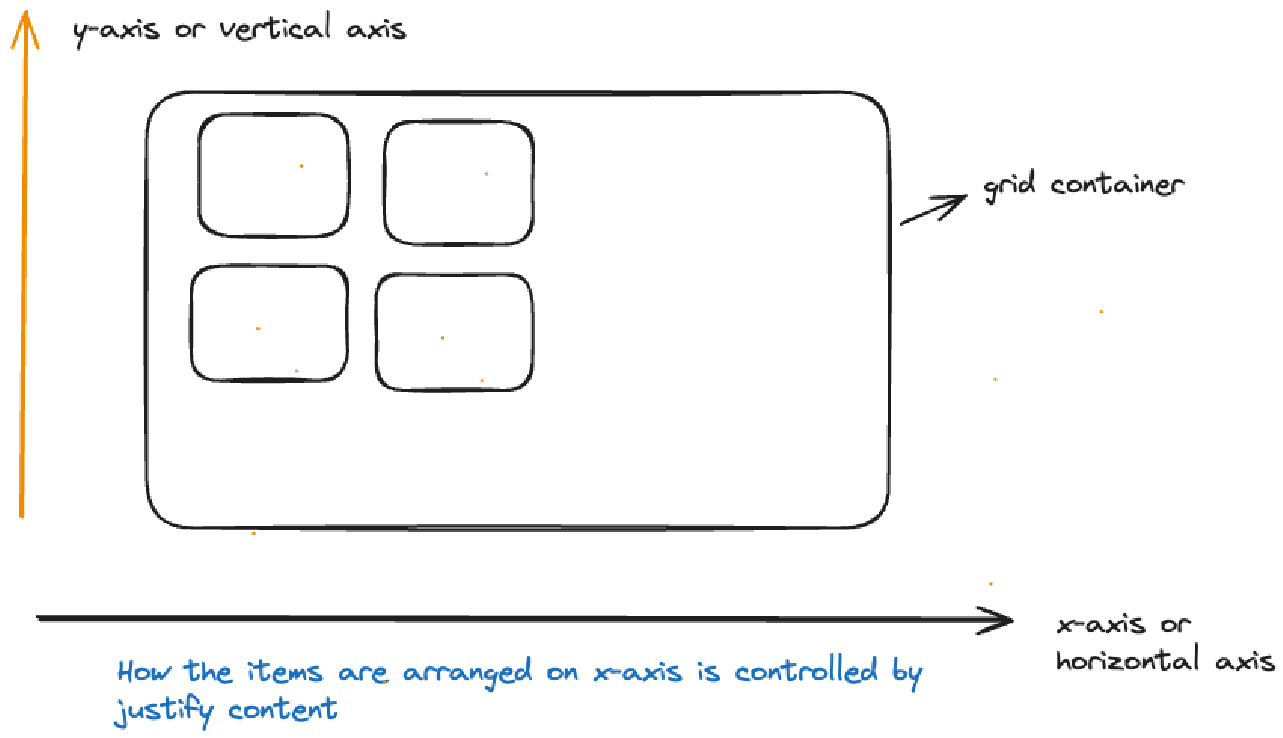
justify-content

This property helps us to control the alignment or arrangement or distribution of the grid-items inside the container along the horizontal axis (x-axis).

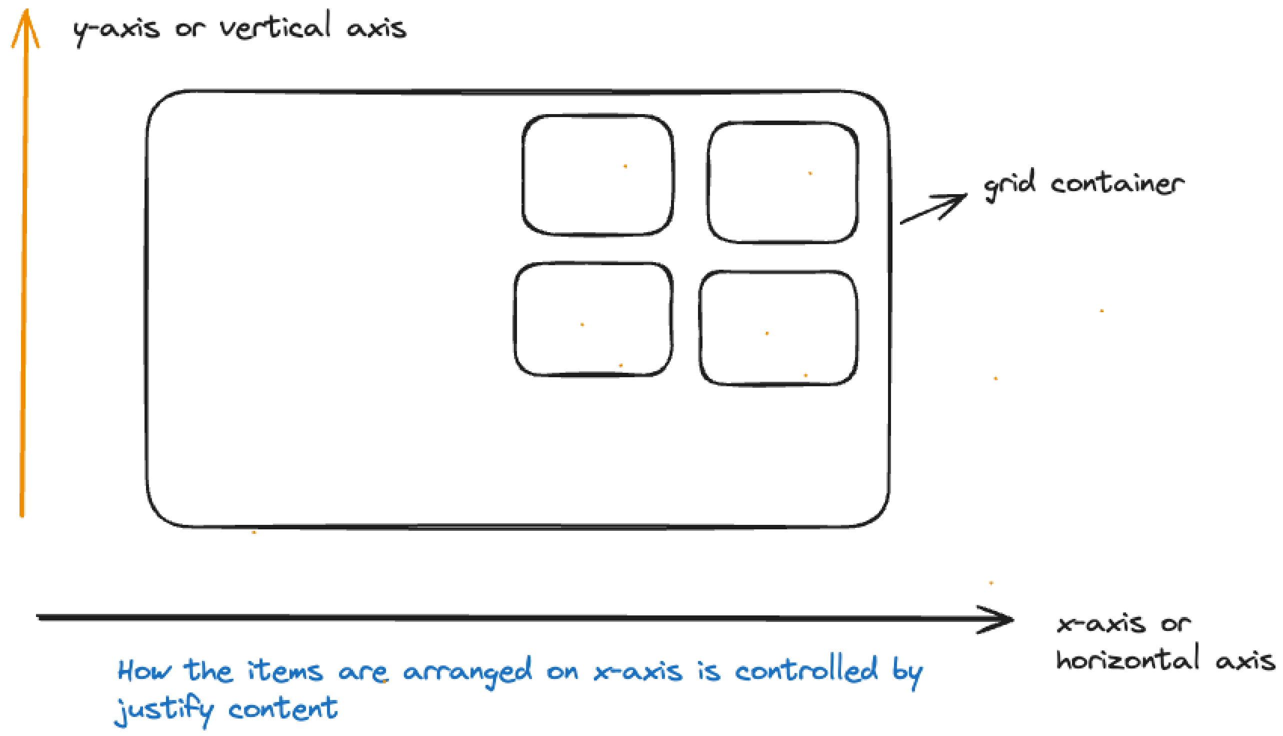

y-axis or vertical axis
grid container
x-axis or horizontal axis
How the items are arranged on x-axis is controlled by justify content

So by default it's value is equal to `start` but we can change it to :

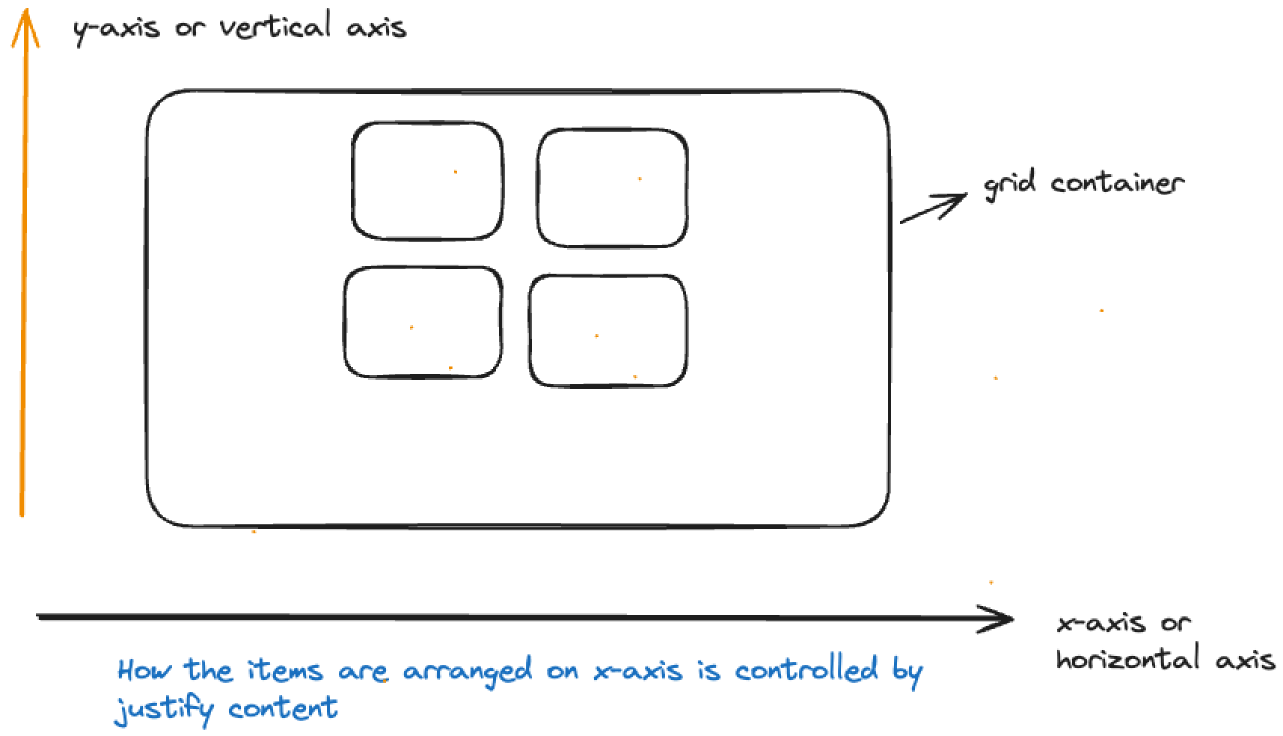
- **start** -> with this all the items are going to take their required width based on content and are pushed to the start / left of the x-axis


y-axis or vertical axis
grid container
x-axis or horizontal axis
How the items are arranged on x-axis is controlled by justify content

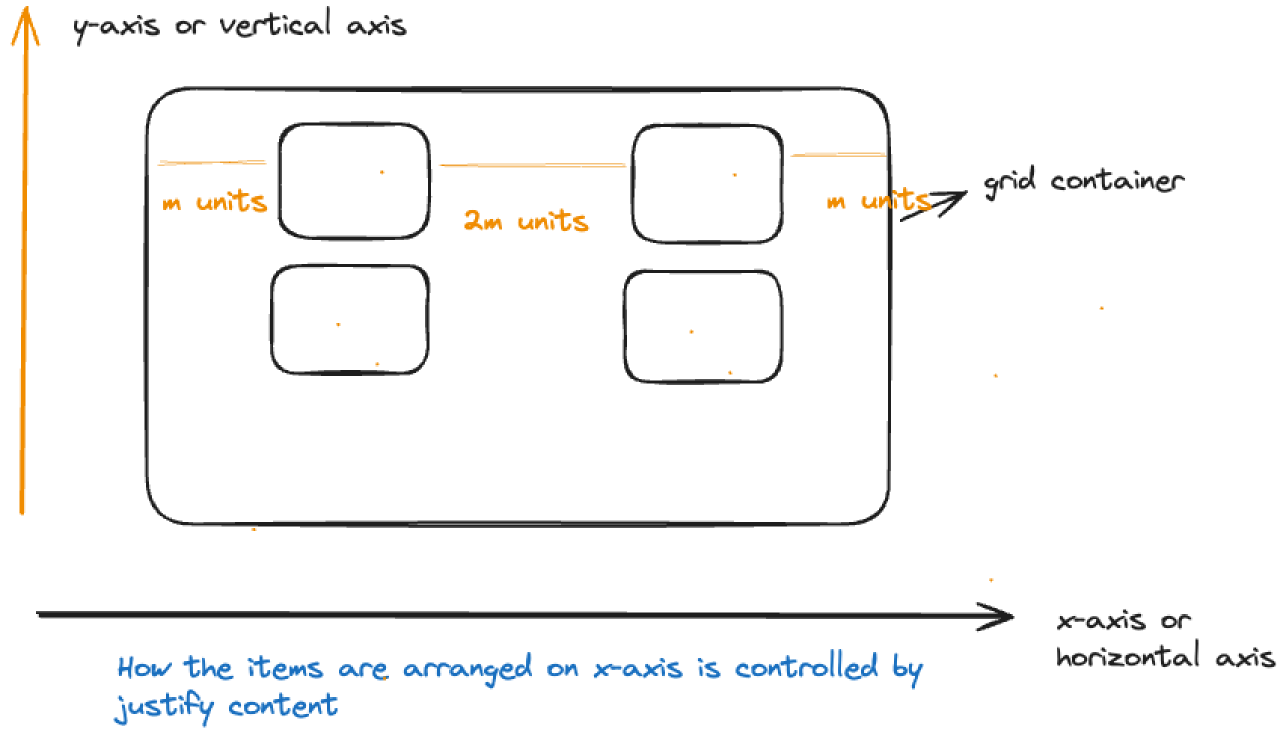
- **end** -> with this all the items are going to take their required width based on content and are pushed to the end / right of the x-axis


y-axis or vertical axis
grid container
x-axis or horizontal axis
How the items are arranged on x-axis is controlled by justify content

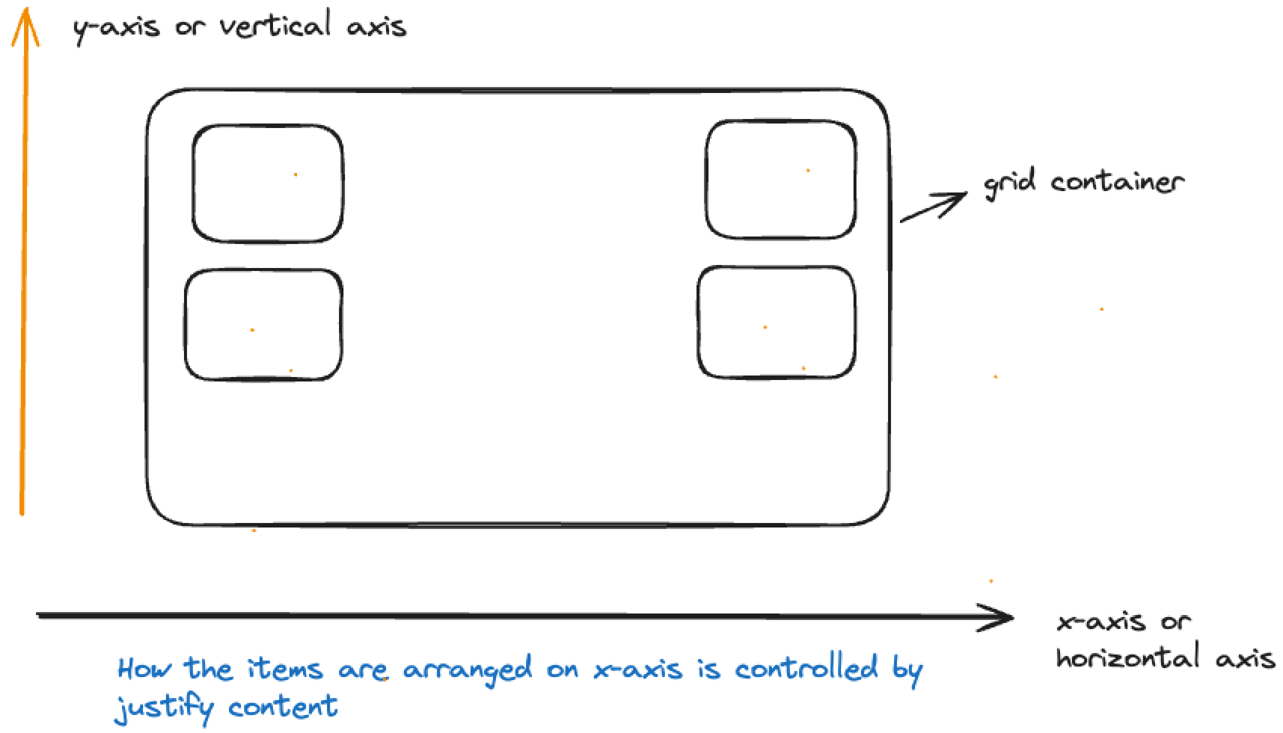
- **center** -> with this all the items are going to take their required width based on content and are pushed to the center of the x-axis


y-axis or vertical axis
grid container
x-axis or horizontal axis
How the items are arranged on x-axis is controlled by justify content

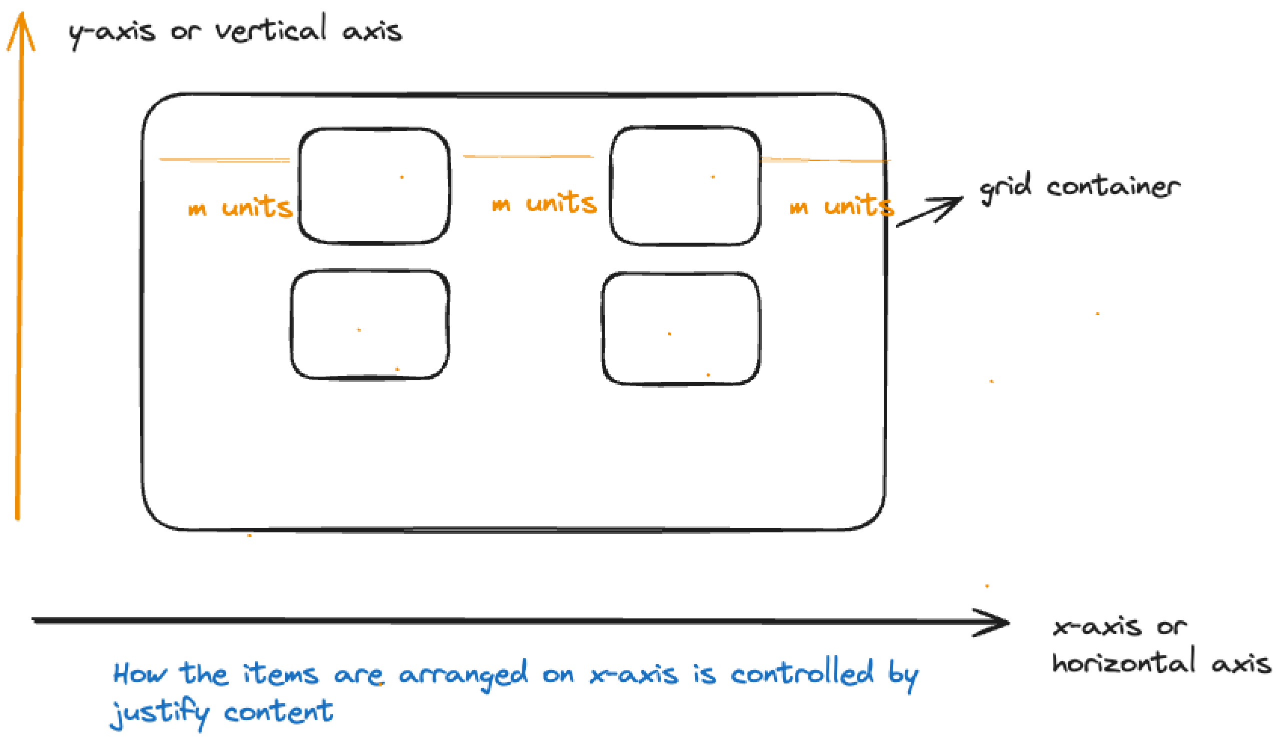
- **space-around** -> with this all the items are going to take their required width based on content and are pushed with double the distance in between when compared to left and right on the x-axis


y-axis or vertical axis
m units
2m units
m units
grid container
x-axis or horizontal axis
How the items are arranged on x-axis is controlled by justify content

- **space-between** -> with this all the items are going to take their required width based on content and are pushed as far as possible on the x-axis. Here we have maximum possible distance between the columns.


y-axis or vertical axis
grid container
x-axis or horizontal axis
How the items are arranged on x-axis is controlled by justify content

- **space-evenly** -> with this all the items are going to take their required width based on content and are pushed with equal spacing on left, right and between on the x-axis


y-axis or vertical axis
m units
m units
m units
grid container
x-axis or horizontal axis
How the items are arranged on x-axis is controlled by justify content

align-items

This property aligns the items vertically within the grid cell. It controls how the content within a cell is aligned vertically.

```css
align-items: end; /*start | end | center */
```

justify-items

This property aligns the items horizontally within the grid cell. It controls how the content within a cell is aligned horizontally.

```css
justify-items: end; /*start | end | center */
```

grid

This property is a shorthand for combining `grid-template-rows` and `grid-template-columns` together. Using this we don't need to separately mention both of these.

```css
grid: grid-template-rows / grid-template-columns;
```

So the row and column configuration is separated by a `/`.
For example:
If we have the row and column configuration like the below:

```
grid-template-columns: 100px 100px;

grid-template-rows: 50px 50px 50px 50px;
```

We can use the `grid` property and mention them together.

```
/* grid-template-columns: 100px 100px;

grid-template-rows: 50px 50px 50px 50px; */

grid: 50px 50px 50px 50px / 100px 100px; /* row config / column config*/
```

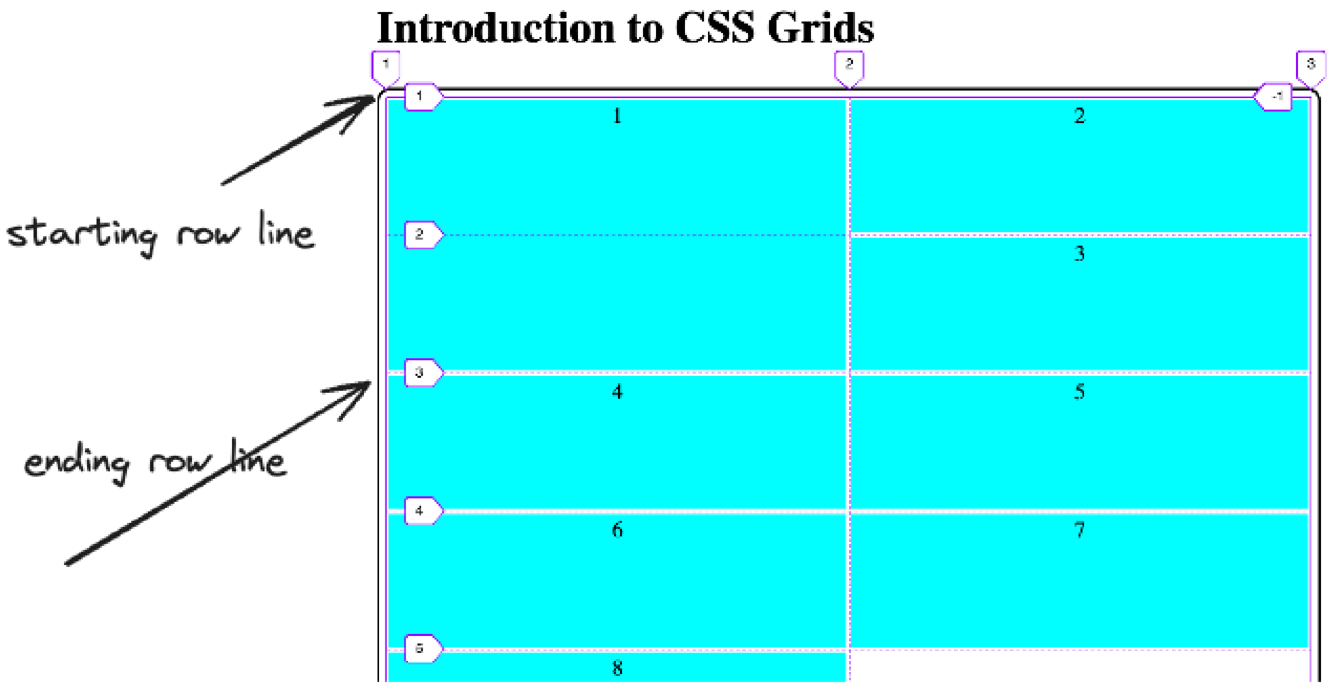
grid-row and grid-column

So these properties help us to expand a particular grid items across multiple rows or columns by specifying the line numbers for the starting and ending row / column.

```
grid-row: start-line-number / end-line-number;
grid-column: start-line-number / end-line-number;
```

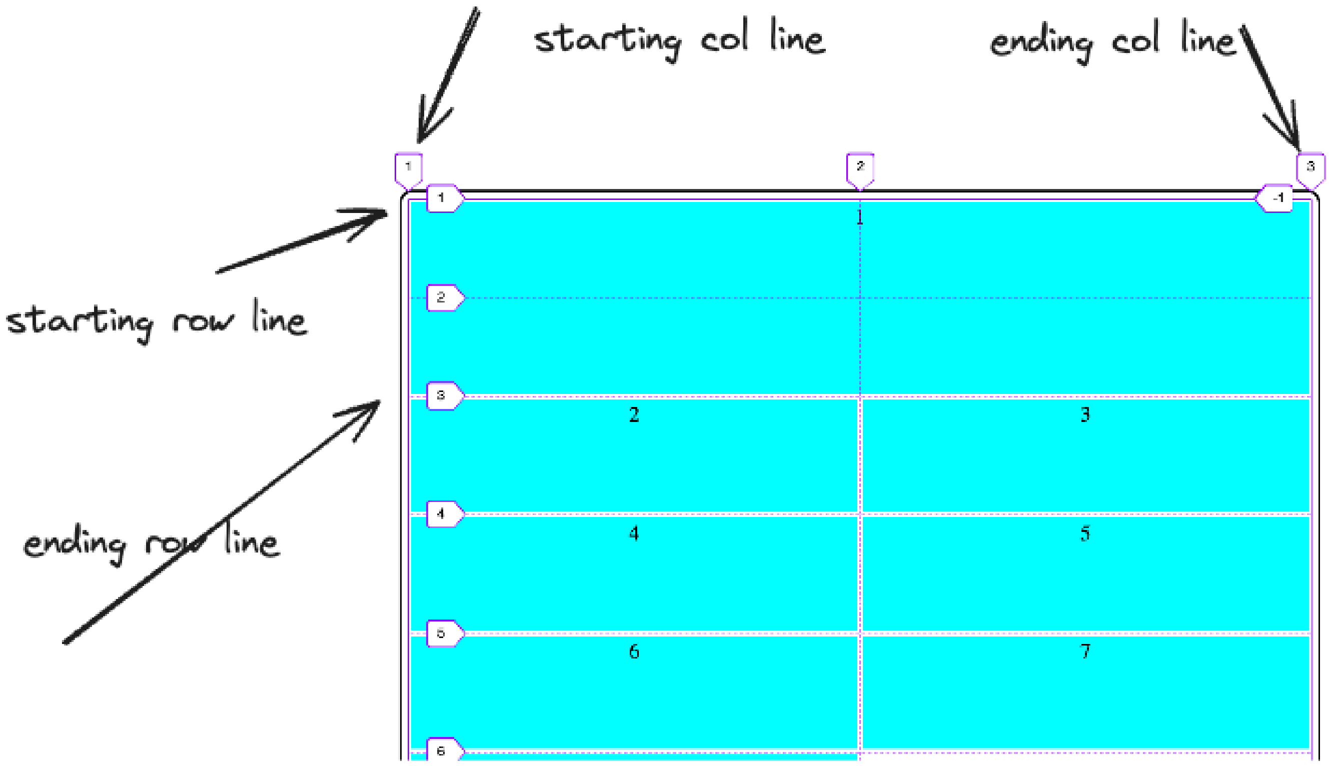
Example 1:

```
#container {
        display: grid;
        grid-template-columns: 1fr 1fr;
}
.b1 {
        grid-row: 1 / 3;
}
```



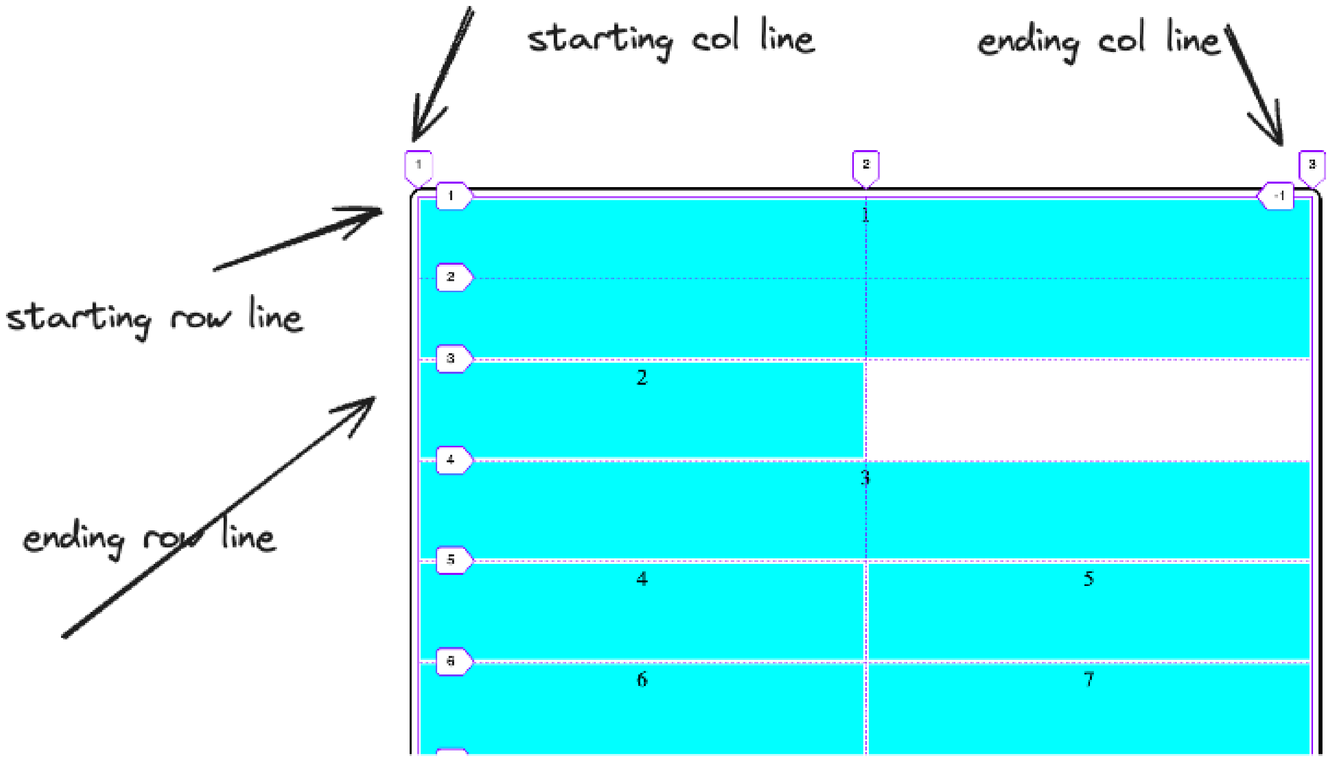
If we add for column as well then,

```
#container {
        display: grid;
        grid-template-columns: 1fr 1fr;
}
.b1 {
        grid-row: 1 / 3;
        grid-column: 1 / 3;
}
```



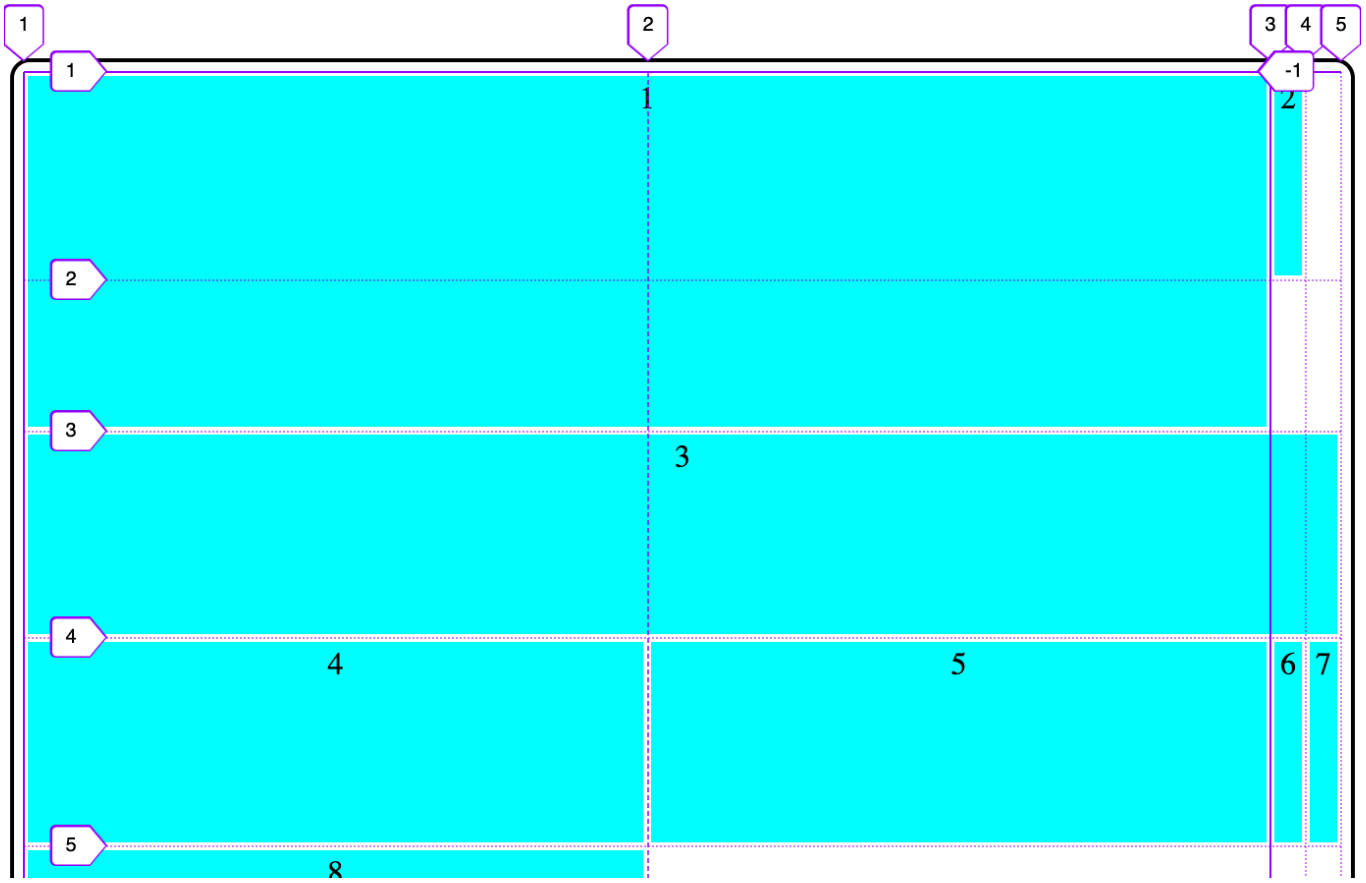
Now here, we can see `cell 3` is starting from line 2 and ending at line 3 in terms of column. We can change it to start from line 1 and end at line 3.

```
.b3 {
        grid-column: 1 / 3;
}
```



Now `cell3` has been displaced from its original position in the grid, and moved to a new line to satisfy this new configuration.

We can even give line numbers that as of now don't exist like in the above image we can say `1 / 5`. As of now line 5 doesn't exist but if we write column configuration, then more columns will be added so that we get column lines upto 5, and rearrangement of all the cells according to 4 columns will be done (4 columns get total 5 lines).



Note:

Instead of giving starting line and ending line number together slash separated in `grid-row` and `grid-column` we can give these values separately using `grid-row-start`, `grid-row-end`, `grid-col-start` and `grid-col-end` . These properties don't do any thing extra, but just is a replacement if we don't want to mention both the line numbers together.

Any other way to do the expansion instead of start and end line ?

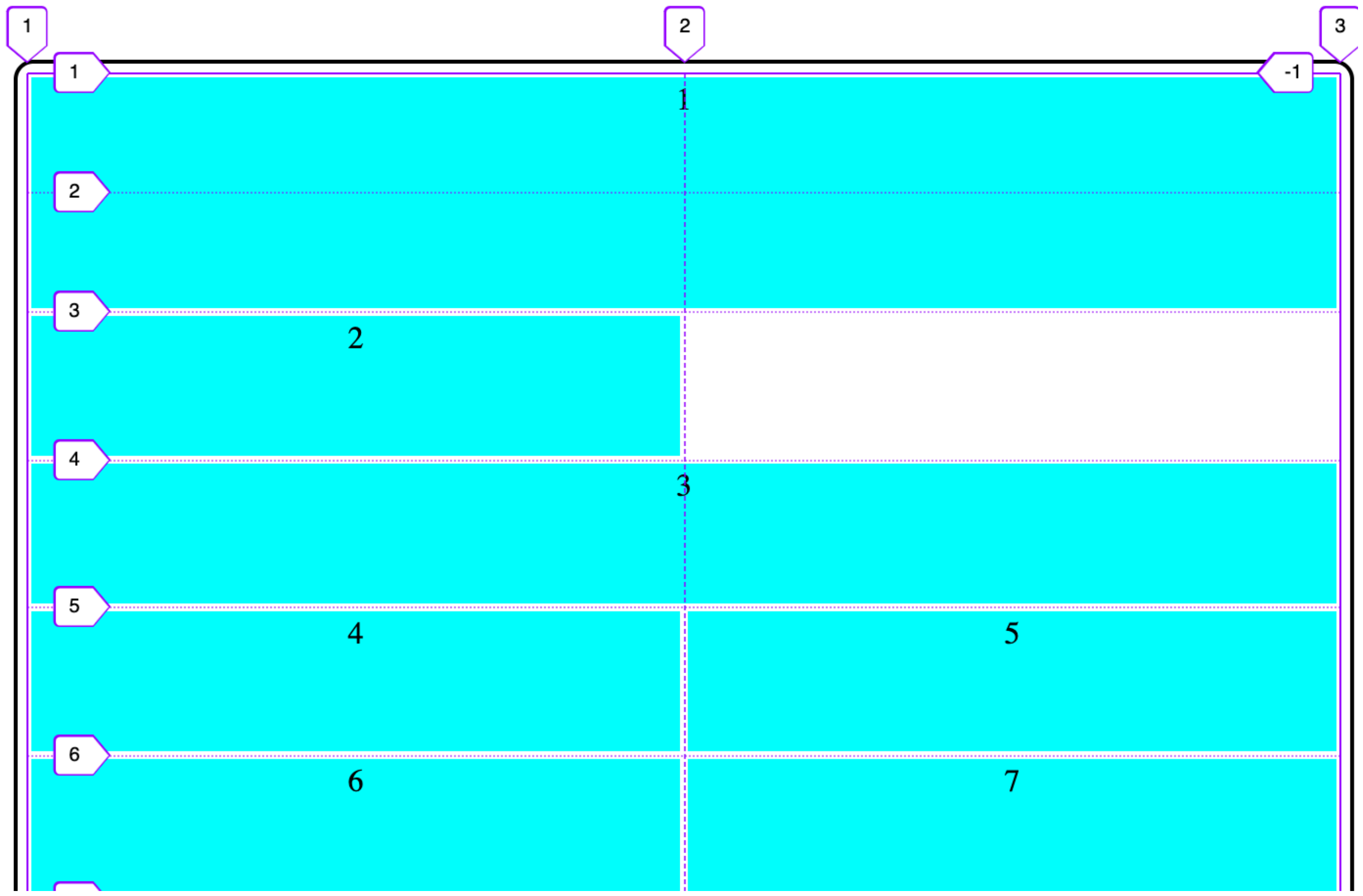
To avoid using start and end line number we can use `span <number_of_tracks>`. Using this items will be expanded with the given number of tracks.

```
.b1 {
        grid-row: 1 / 3;
        grid-column: 1 / 3;
}
```

This above configuration can be written using span like this

```
.b1 {
        grid-row: 1 / span 2;
        grid-column: span 2;
}
```

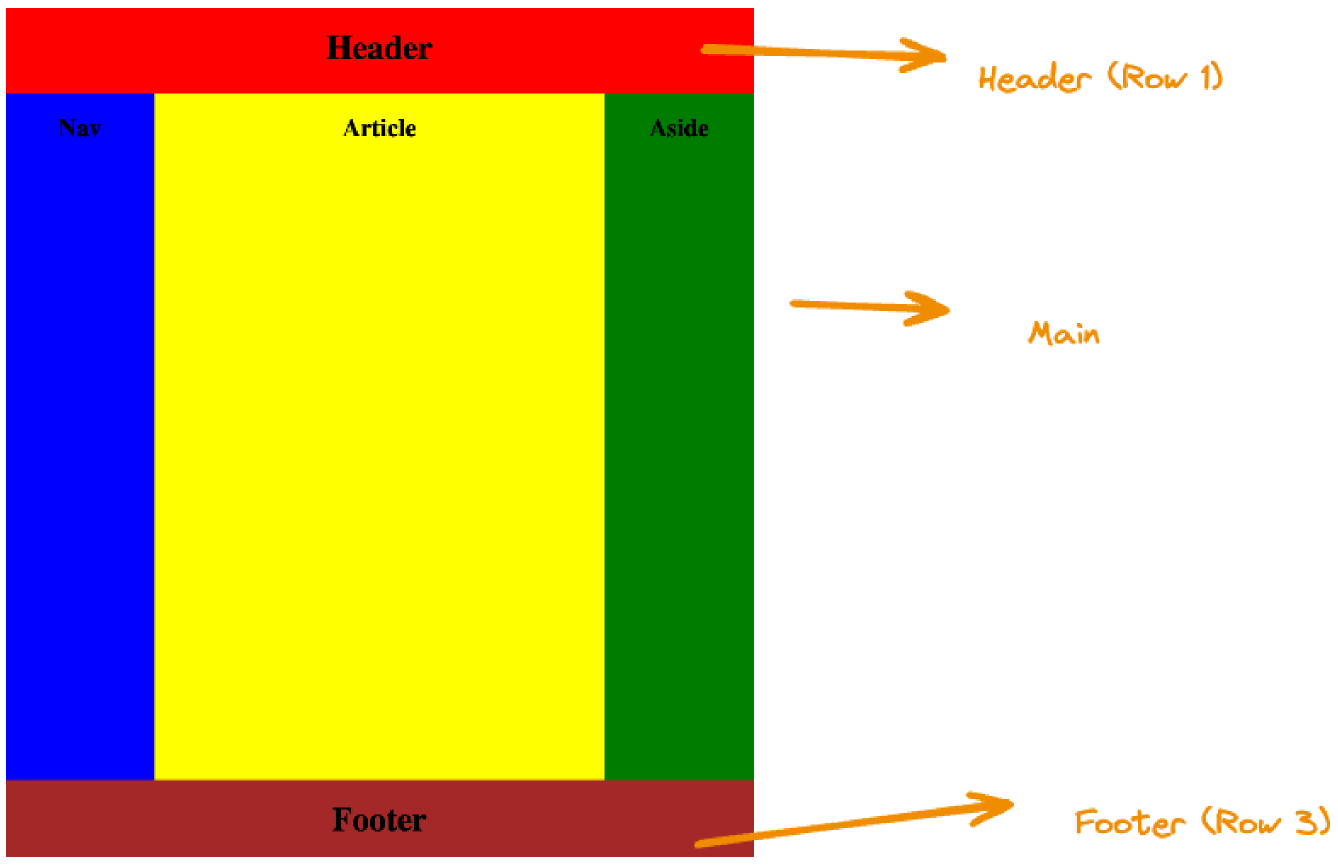
Here for the row we still mention starting line but ending line we say `span 2` that mean expand across two tracks (i.e. rows).
For columns we don't mention start and end line both and just say `span 2` which means start from your allocated line number in column and expand across two tracks (i.e. column)



Preparing Holy grail UI

The holy grail UI is a very common representation of a lot of applications specially social media apps like Facebook, Linkedin, X etc. In this UI our page is divided in 3 rows

- Header - the first row
- Main content - the second row
- Footer - the last row
  Inside the main content we have 3 columns ideally:
- Left nav
- Main Middle part
- Right nav



We will make the container get enabled with `diplay: grid` and then we will configure 3 rows using `grid-template-rows`. Here we have given first row `10vh` , second row as `80vh` and last one with `10vh`.

Then we will enable `display: grid` in the main content as well, and define grid areas in the container:

```
        "header header header"

        "main main main"

        "footer footer footer";
```

And then we will add some basic stylings.

```
#container {
        display: grid;
        text-align: center;
        grid-template-rows: 10vh 80vh 10vh;
        /* grid-template-columns: 1fr 3fr 1fr; */
        grid-auto-flow: column;
        grid-template-areas:
                "header header header"
                "main main main"
                "footer footer footer";
        }

main {
        display: grid;
        grid-area: main;
        grid-auto-flow: column;
        grid-template-columns: 1fr 3fr 1fr;
}

nav {
        background-color: blue;
}

aside {
        background-color: green;
}

article {
        background-color: yellow;
}

header {
        grid-area: header;
        background-color: red;
}

footer {
        grid-area: footer;
        background-color: brown;
}
```

```
<div id="container">
        <header>
                <h1>Header</h1>
        </header>
        <main>
                <nav>
                        <h1>Nav</h1>
                </nav>
                <article>
                        <h1>Article</h1>
                </article>
                <aside>
                        <h1>Aside</h1>
                </aside>
        </main>
        <footer>
                <h1>Footer</h1>
        </footer>
</div>
```

- The `#container` selector targets the element with the `id` attribute set to "container". It uses CSS Grid properties to define the layout of the container element.
- The `main` selector targets the `<main>` element and sets its CSS Grid properties to create a three-column layout within the main section.
- The `nav`, `aside` and `article` selectors target the respective elements and set their background colors to blue, green, and yellow, respectively.
- The `header` and `footer` selectors target the header and footer elements and set their background colors to red and brown, respectively. They also use the `grid-area` property to specify their position within the grid layout.
- The `display: grid;` property on the `#container` selector sets the container element to use CSS Grid layout.
- The `grid-template-rows` property defines the height of the rows in the grid. In this case, it sets the first and last rows to be 10% of the viewport height (`10vh`), and the middle row to be 80% of the viewport height (`80vh`).
- The `grid-auto-flow: column;` property sets the flow of grid items to be in columns.
- The `grid-template-areas` property defines the areas within the grid layout. It specifies that the header, main, and footer sections should span across all columns in their respective rows.
- The `grid-template-columns` property on the `main` selector defines the width of the columns within the main section. It sets the first and last columns to be 1/4 of the total width (`1fr`), and the middle column to be 3/4 of the total width (`3fr`).

Overall, this code creates a web page layout with a header, main section divided into three columns, and a footer. The CSS Grid properties are used to define the layout and positioning of the different sections within the grid.

We can also use `grid-row` and `grid-column` to make the same ui.

```
#container {
        display: grid;
        grid-template-columns: 1fr 3fr 1fr;
        grid-template-rows: 10vh 80vh 10vh;
```

```
main {
        display: grid;
        grid-column: 1 / span 3;
        grid-auto-flow: column;
        grid-template-columns: 1fr 3fr 1fr;
}

nav {
        background-color: blue;
}

aside {
        background-color: green;
}

article {
        background-color: yellow;
}

header {
        grid-column: 1 /span 3;
        background-color: red;
}

footer {
        grid-column: 1 /span 3;
        background-color: brown;
}
```