POSITIONING GRID ELEMENT

Every track has a start line and an end line. The lines are numbered from left to right and top to bottom starting at 1. Grid lines are what we use to position grid items.

`grid-column-start` and `grid-column-end`.
`grid-row-start` and `grid-row-end`

These properties allow us to use our existing grid lines to tell items how many rows and columns each item should span across.

Example:

```
#living-room {
  grid-column-start: 1;
  grid-column-end: 6;
  grid-row-start: 1;
  grid-row-end: 3;
```

A. Shorthand

`grid-column`	shorthand for 'grid-column-start' and 'grid-column-end'
`grid-row`	shorthand for 'grid-row-start' and 'grid-row-end'

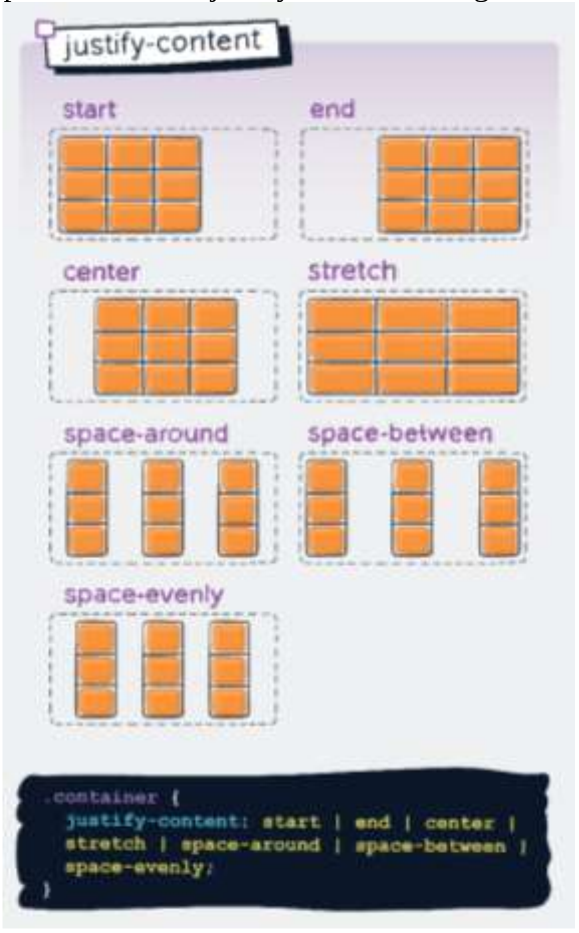
`grid-area` : `grid-row-start` / `grid-column-start` / `grid-row-end` / `grid-column-end`

Example:

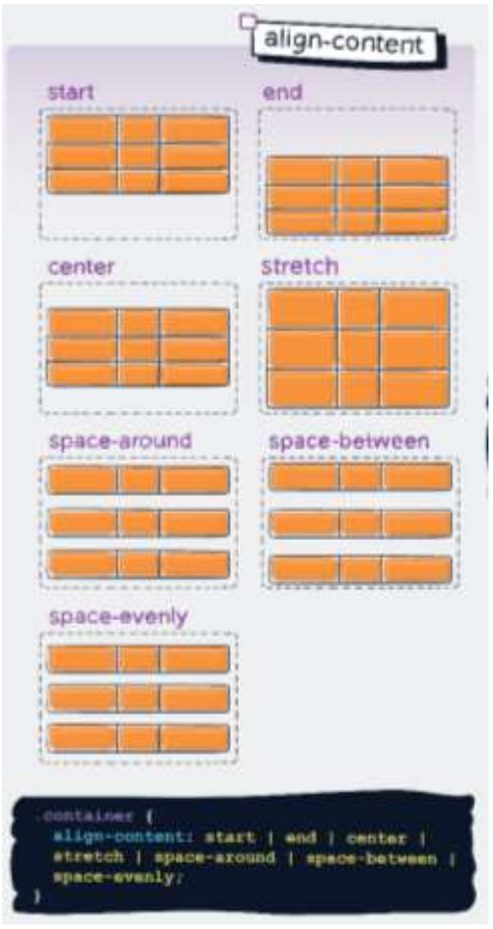
```
#living-room {
  grid-area: 1 / 1 / 3 / 6;
```

B. Other Container Properties

- place-content: <justify-content> <align-content>

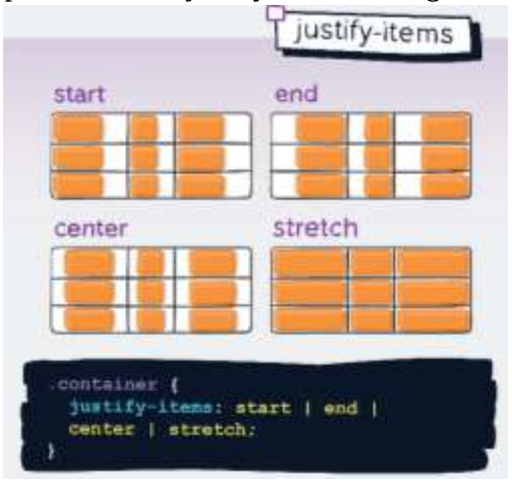


```
.container {
  justify-content: start | end | center |
  stretch | space-around | space-between |
  space-evenly;
}
```

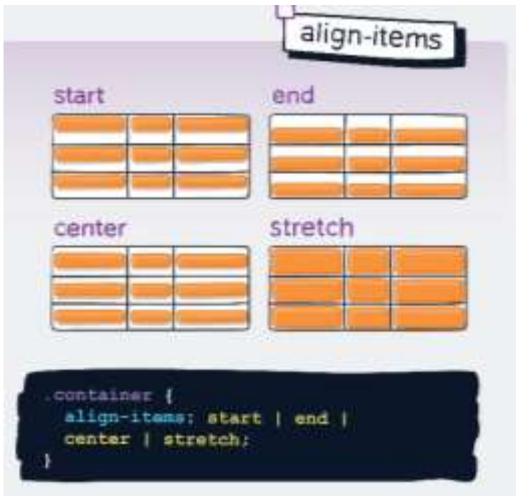


```
.container {
  align-content: start | end | center |
  stretch | space-around | space-between |
  space-evenly;
}
```

➢ place-items: <justify-items> <align-items>



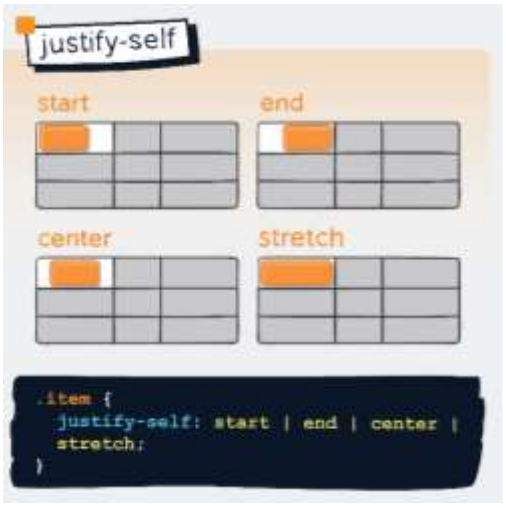
```
.container {
  justify-items: start | end |
  center | stretch;
}
```



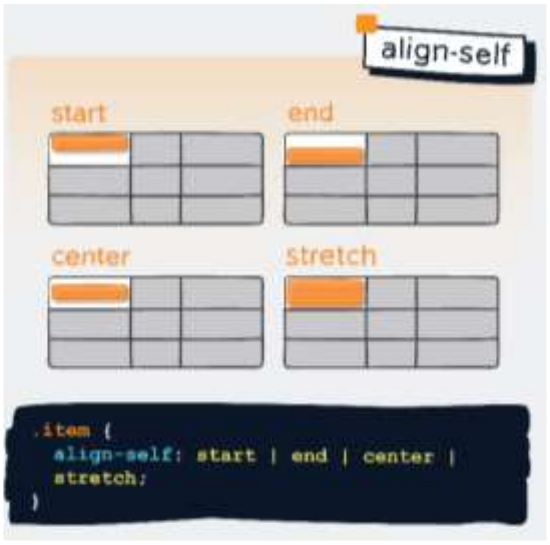
```
.container {
  align-items: start | end |
  center | stretch;
}
```

C. Other Grid-Item Properties

➢ place-self: <justify-self> <align-self>



```
.item {
  justify-self: start | end | center |
  stretch;
}
```



```
.item {
  align-self: start | end | center |
  stretch;
}
```

D. grid-template-areas

To define grid layout using area-name. Example:

```
.container {
  display: grid;
  grid-template-areas:
    "header header header"
    "sidebar main main"
    "footer footer footer";
}

.header {
  grid-area: header;
}
.sidebar {
  grid-area: sidebar;
}
.main {
  grid-area: main;
}
.footer {
  grid-area: footer;
}
```

Artinya ada 3 baris & 3 kolom

'header' menempati 3 kolom di baris pertama.
'sidebar' menempati 1 kolom di baris kedua, diikuti oleh main yang menempati 2 kolom di baris yang sama.
'footer' menempati 3 kolom di baris ketiga.

Contoh lain:

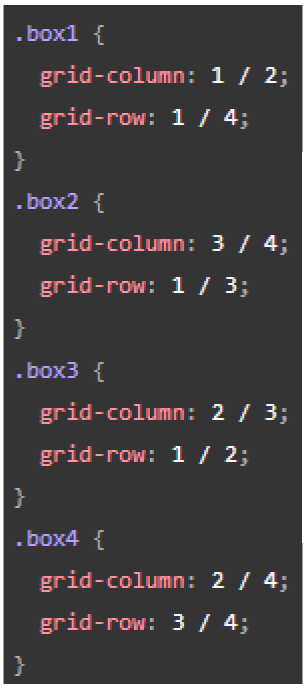
```
.grid-container {
  display: grid;
  grid-template-areas: "myArea myArea . . .";
```

Item dengan nama ‘myArea’ menempati 2 kolom di baris pertama(baris pertama memiliki 5 kolom.

Note: titik (.) artinya area dikosongkan/tidak diisi element apapun.

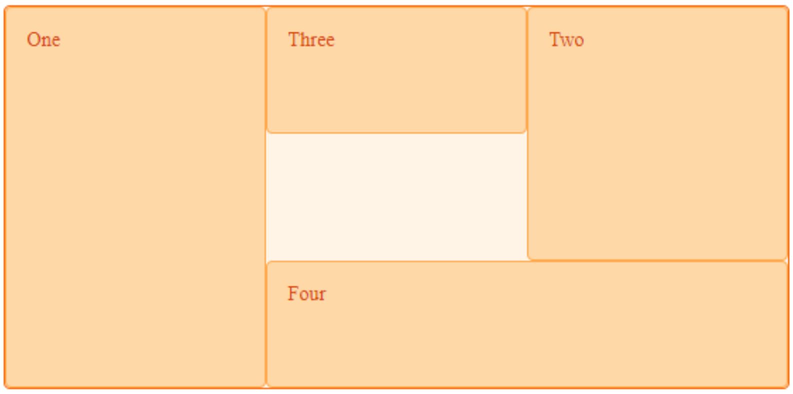
E. Default Span

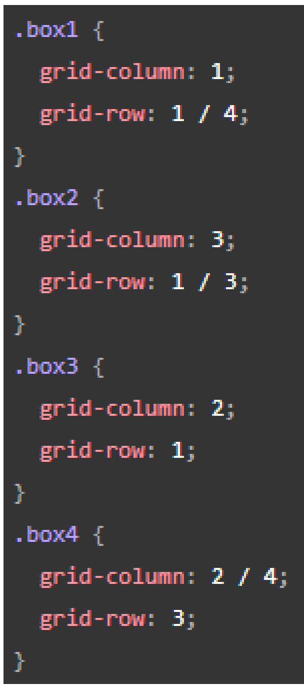
Grid defaults to spanning one track. So if an item only spans one track you can omit the 'grid-column-end' or 'grid-row-end' value.



```
.box1 {
  grid-column: 1 / 2;
  grid-row: 1 / 4;
}
.box2 {
  grid-column: 3 / 4;
  grid-row: 1 / 3;
}
.box3 {
  grid-column: 2 / 3;
  grid-row: 1 / 2;
}
.box4 {
  grid-column: 2 / 4;
  grid-row: 3 / 4;
}
```

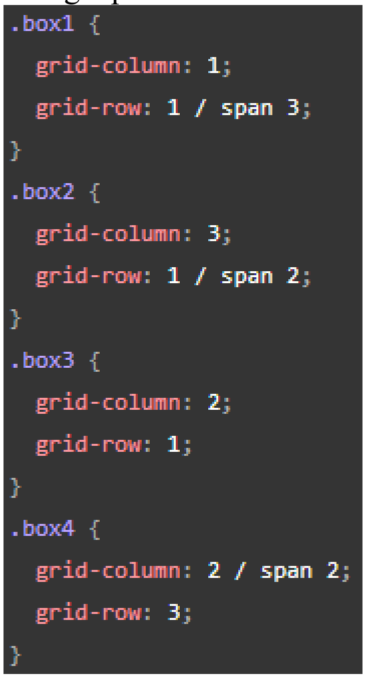
code on the left same as the code **on the right**





```
.box1 {
  grid-column: 1;
  grid-row: 1 / 4;
}
.box2 {
  grid-column: 3;
  grid-row: 1 / 3;
}
.box3 {
  grid-column: 2;
  grid-row: 1;
}
.box4 {
  grid-column: 2 / 4;
  grid-row: 3;
}
```

➢ Using 'span'

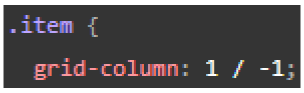


```
.box1 {
  grid-column: 1;
  grid-row: 1 / span 3;
}
.box2 {
  grid-column: 3;
  grid-row: 1 / span 2;
}
.box3 {
  grid-column: 2;
  grid-row: 1;
}
.box4 {
  grid-column: 2 / span 2;
  grid-row: 3;
}
```

F. Counting Backwards

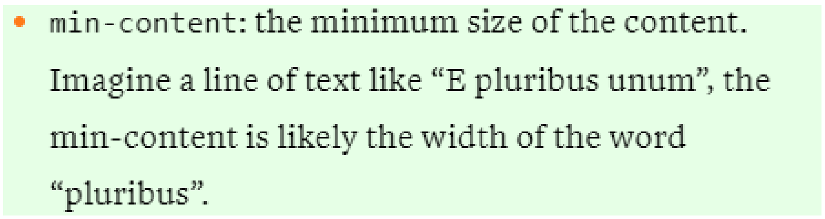
We can count backwards from the end of the grid(right-hand column line and final row line). These lines can be addressed as -1, and you can count back from there, so the second last line is -2.

This is useful for 'stretching an item across the grid':



```
.item {
  grid-column: 1 / -1;
```

G. Sizing keywords

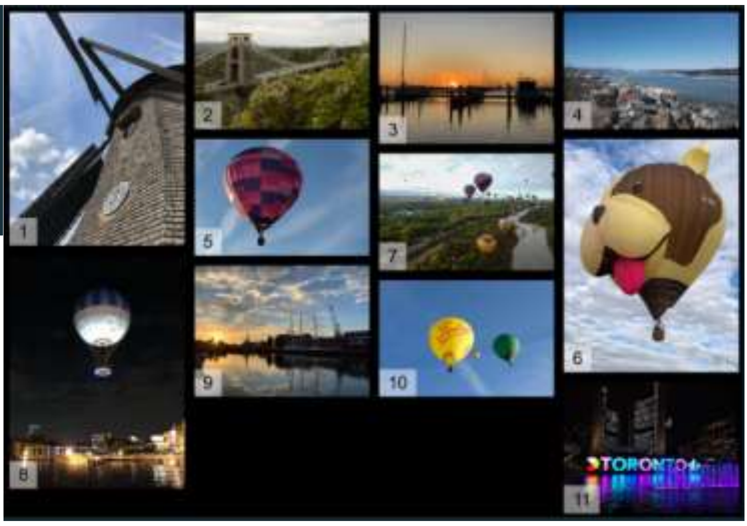


- `min-content`: the minimum size of the content. Imagine a line of text like "E pluribus unum", the min-content is likely the width of the word "pluribus".

- `max-content`: the maximum size of the content. Imagine the sentence above, the `max-content` is the length of the whole sentence.
- `auto`: this keyword is a lot like `fr` units, except that they “lose” the fight in sizing against `fr` units when allocating the remaining space.
- Fractional units: see next file

H. Masonry Grid Layout(Using CSS not JS)

```css
.container {
  display: grid;
  grid-template-columns: repeat(4, 1fr);
  grid-template-rows: masonry;
```



This will creates four-column grid, with the rows set to masonry. The masonry items will be displayed in the four columns of my grid axis.
To use masonry layout, one of your grid axes needs to have the value masonry(masonry axis). The other axis will have rows or column tracks defined as normal(grid axis).

```css
@supports (grid-template-rows: masonry) {
  /* masonry code here */
}
```

For fallback behavior because some browser no support masonry.