CSS Grid

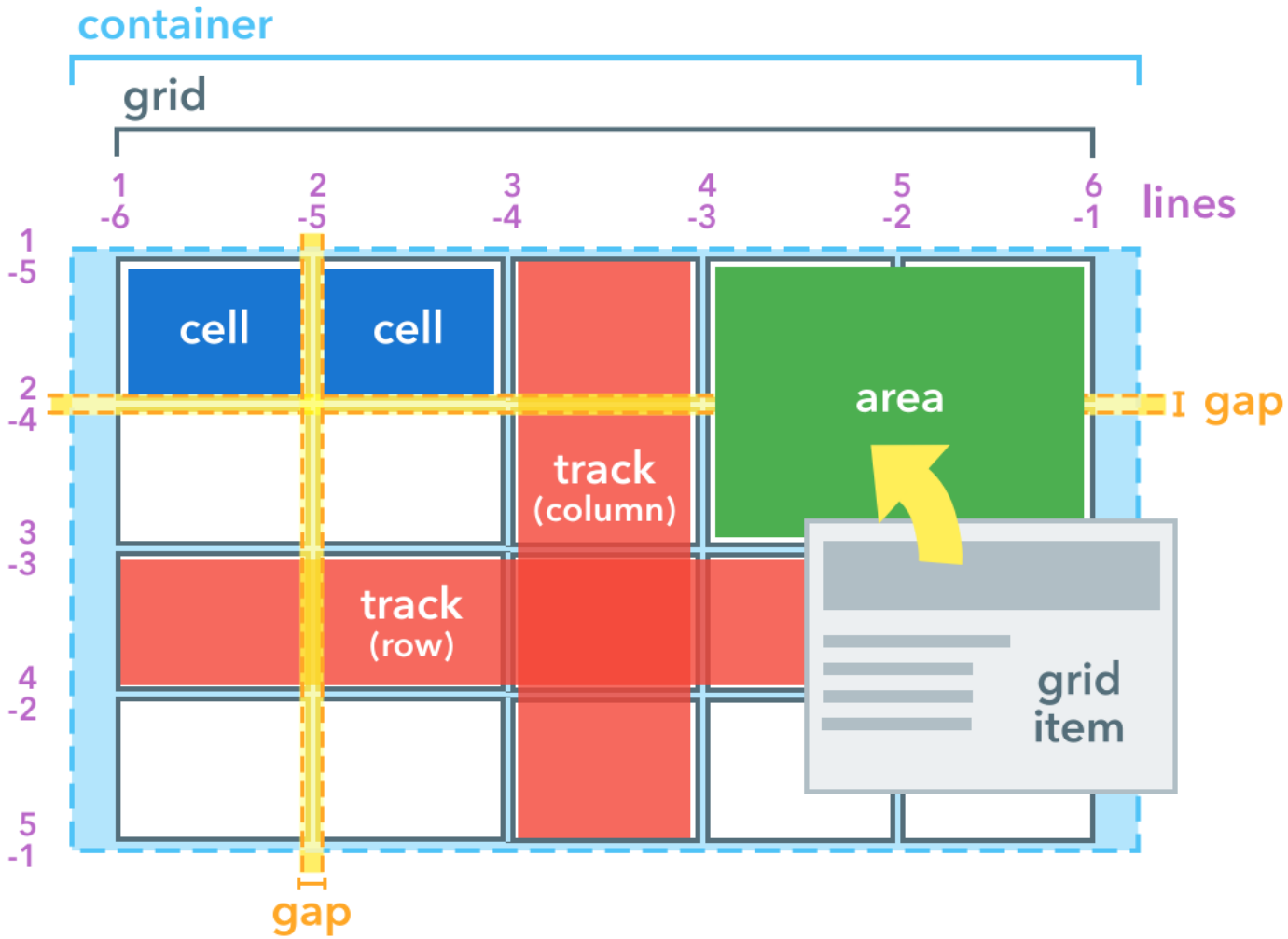
What is Grid

Grid Layout is a layout model for CSS that has powerful abilities to control the sizing and positioning of boxes and their contents. Grid Layout is optimized for 2-dimensional layouts: those in which alignment of content is desired in both dimensions.

Features

- Fixed and flexible track sizes: Fixed track sizes – using pixels or flexible sizes with percentages or with the new fr unit designed for this purpose.
- Item placement: Place items into a precise location on the grid using line numbers, names or by targeting an area of the grid.
- Creation of additional tracks to hold content
- Alignment control
- Control of overlapping content

Grid Concepts



Terminology

- A **grid container** establishes an independent grid formatting context for its contents.
- **Grid lines** are the horizontal and vertical dividing lines of the grid. A grid line exists on either side of a column or row. They can be referred to by numerical index, or by an author-specified name.
- A **grid cell** is the intersection of a grid row and a grid column. It is the smallest unit of the grid that can be referenced when positioning grid items.
- A **grid area** is the logical space used to lay out one or more grid items. A grid area consists of one or more adjacent grid cells. It is bound by four grid lines
- **Grid track** is a generic term for a grid column or grid row—in other words, it is the space between two adjacent grid lines.
- **grid items** are boxes representing its in-flow contents.

Flexbox vs Grid

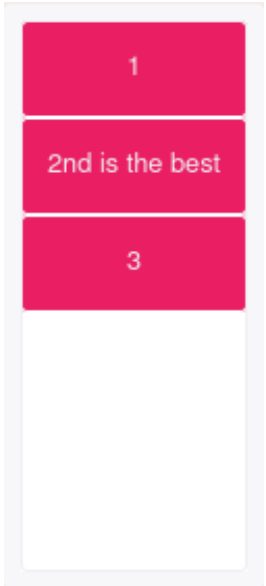
Flexbox	Grid
one-dimensional, Flexbox only deals with either columns or rows. greater control over alignment and space distribution between items.	Grid has two-dimension layout capabilities which allow flexible widths as a unit of length.
Layout First. CSS Grids helps you create the outer layout of the webpage. You can build complex as well responsive design with this.	Content First.. Flexbox mostly helps align content & move blocks.
The Flexbox layout is best suited to application components and small-scale layouts,	Grid layout is designed for larger-scale layouts that are not linear in design.
flexbox is best when you have a bunch of stuff of different sizes and just want a reasonable in proportion layout from them.	If you start constraining all your flex items with a width, then more often than not it will be easier to use grid.

Grid Containers

Create a grid container by setting the display property with a value of grid or inline-grid.

display: grid/inline-grid

Grid items are placed in rows by default and span the full width/max-content width of the grid container when display is grid/inline-grid respectively .



Code Demo

Explicit Grid

The three properties **grid-template-rows**, **grid-template-columns**, and **grid-template-areas** together define the **explicit** grid of a grid container by specifying its explicit grid tracks.

grid-template-rows: 10px 20%

A row track is created for each value specified for grid-template-rows.

grid-template-columns: 10px 40%

A column track is created for each value specified for grid-template-columns.

grid-template-areas: "header header"
"content sidebar"

Named Grid Areas. A row is created for every separate string listed for the grid-template-areas property, and a column is created for each cell in the string.

Grid Track Sizes

<length> A non-negative length, giving the width of the column.

<percentage> Is a non-negative <percentage> value relative to the inline size of the grid container. If the size of the grid container depends on the size of its tracks, then the percentage must be treated as auto. The intrinsic size contributions of the track may be adjusted to the size of the grid container and increase the final size of the track by the minimum amount that would result in honoring the percentage.

<flex> Is a non-negative dimension with the unit fr specifying the track's flex factor. Each <flex>-sized track takes a share of the remaining space in proportion to its flex factor. When appearing outside a minmax() notation, it implies an automatic minimum (i.e. minmax(auto, <flex>)).

auto As a maximum represents the largest max-content size of the items in that track. As a minimum represents the largest minimum size of items in that track (specified by the min-width/min-height of the items). This is often, though not always, the min-content size. If used outside of minmax() notation, auto represents the range between the minimum and maximum described above. This behaves similarly to minmax(min-content,max-content) in most cases.

View Specification  Code Demo

max-content Is a keyword representing the largest maximal content contribution of the grid items occupying the grid track.

min-content Is a keyword representing the largest minimal content contribution of the grid items occupying the grid track.

Exercises

- Create a grid with rows using max-content min-content and fit-content.
-

Flexible Lengths: the fr unit

- A *flexible length* or *<flex>* is a dimension with the *fr* unit, which represents a fraction of the leftover space in the grid container.
- Tracks sized with fr units are called *flexible tracks* as they flex in response to leftover space similar to how flex items with a zero base size fill space in a flex container.
- <leftover space> = The total maximum size of all non-flexible track sizing rows or columns - the available space.
- **Each column or row's share of the leftover space can be computed as the column or row's <flex> * <leftover space> / <sum of all flex factors>.**

fr

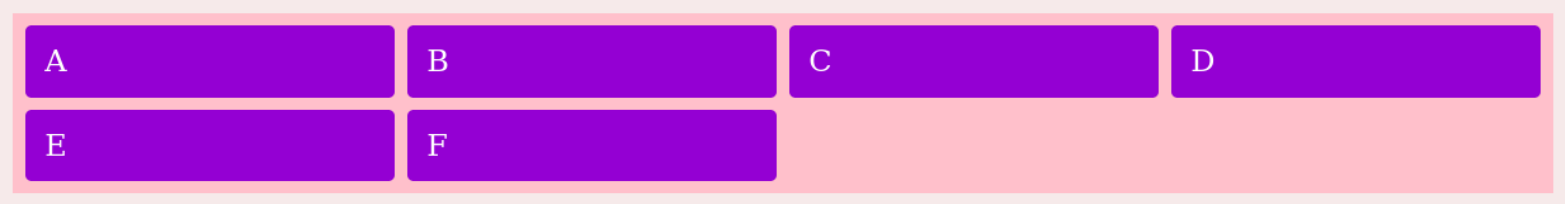
When the available space is infinite I.e. grid container's width or height is , not specified.

The used size of each flex-sized grid track is computed by determining the max-content size of each flex-sized grid track and dividing that size by the respective flex factor to determine a "hypothetical 1fr size". The maximum of those is used as the resolved 1fr length (the *flex fraction*), which is then multiplied by each grid track's flex factor to determine its final size.

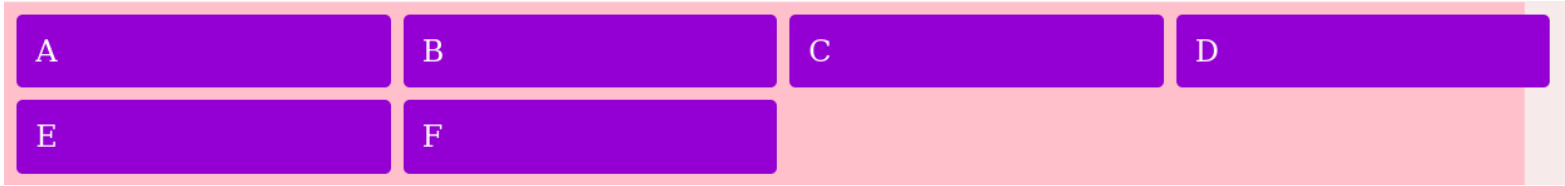
Note: **<flex> values are not <length>s (nor are they compatible with <length>s, like some <percentage> values), so they cannot be represented in or combined with other unit types in calc() expressions.**

fr vs percentage

grid-template-columns: 1fr 1fr 1fr 1fr



grid-template-columns: 25% 25% 25% 25%



Both have grid-gap of 10px. There’s no overflow on the x-axis with fr because setting each column to fr takes 10px into account automatically and subtracts it from the total width available for each column.

fr vs auto

grid-template-columns: 300px 1fr;

Imagine a fixed sidebar and main content area that takes up the rest of the space. That 1fr column will take up any remaining space left behind by the fixed 300px column.

auto value would do the same, but it isn’t quite as robust since its size is based on the content inside, if the content is less, it won't fill up the space.

Grid Gaps(Gutters)

- The grid-column-gap and grid-row-gap properties create gutters between columns and rows.
- Grid gaps are only created in between columns and rows, and not along the edge of the grid container
-

- Example: row-gap, column-gap, grid-gap shorthand demo
-

Minimum and Maximum Grid Track Sizes

Tracks sizes can be defined to have a minimum and/or maximum size with the minmax() function. **minmax(min, max)** is a functional notation that defines a size range.

The minmax() function accepts 2 arguments: the first is the minimum size of the track and the second the maximum size. Alongside length values, the values can also be auto, which allows the track to grow/stretch based on the size of the content.

If max is smaller than min, then max is ignored and the function is treated as min. **As a maximum, a <flex> value sets the track's flex factor. It is invalid as a minimum.**

Smaller containers/screens takes minimum width, larger containers/screens takes maximum width.

```
grid-template-columns: repeat(auto-fill, minmax(min(10rem, 100%), 1fr));
```

Repeating Grid Tracks

Define repeating grid tracks using the repeat() notation. This is useful for grids with items with equal sizes or many items.

The generic form of the repeat() syntax is, approximately,

repeat(number of columns/rows, <track list of the column/row widths we want>);

The first argument specifies the number of repetitions. The second argument is a track list, which is repeated that number of times.

Exercise: Template area + grid template rows/columns which will take effect

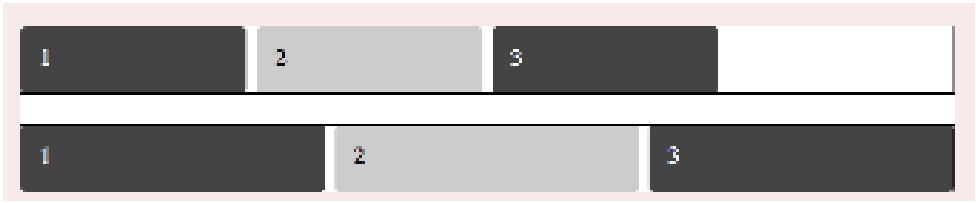
Code Demo

Auto-fill vs auto-fit

when *auto-fill* or *auto-fit* is given as the repetition number,
if the grid container has a definite size or max/min size in the relevant axis, then the number of repetitions is the largest/smallest possible positive integer that does not cause the grid to overflow the content box of its grid container. Otherwise, the specified track list repeats only once.

after grid item placement any empty repeated tracks are collapsed. An empty track is one with no in-flow grid items placed into or spanning across it. A *collapsed track* is treated as having a fixed track sizing function of 0px, and the gutters on either side of it.

```
grid-template-columns: repeat(auto-fill, minmax(200px,1fr));

grid-template-columns: repeat(auto-fit, minmax(200px,1fr));
```



Exercise

- Create Grid with fixed width
- Create Grid with Variable widht
- Create Grid Fixed and Variable widths

Grid auto placement

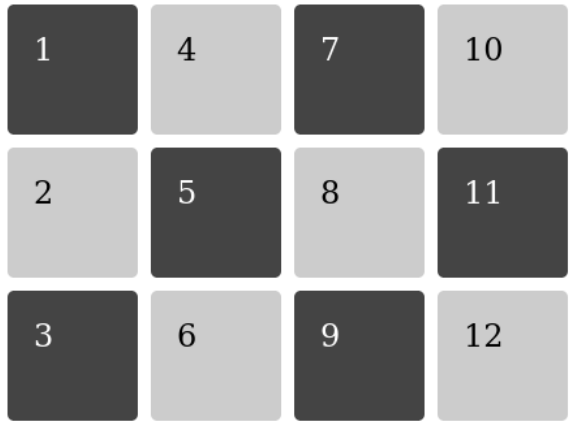
- If a child element has been placed then the auto-placement algorithm will place that first and then work out what to do with any child elements that have not been placed.

- Code Demo

Grid auto-flow

- The default behavior of Grid Auto Flow is to layout the elements by row, You can change this behavior by setting the grid-auto-flow property to column.

```
grid-auto-flow: column;
```



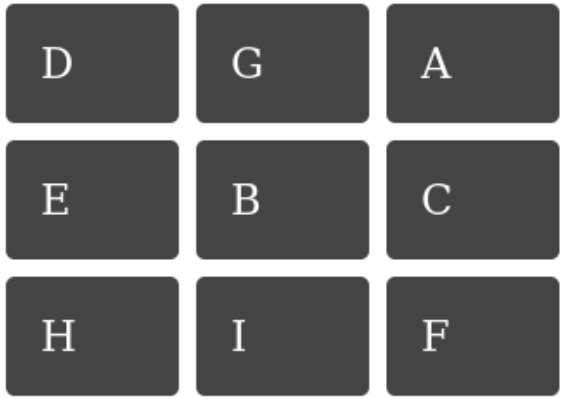
Positioning Items by Grid Line Numbers

You can place the grid celss using grid line numbers or grid-column, grid-row or grid-area.

Code Demo

Exercise

Position the items as shown in the grid.

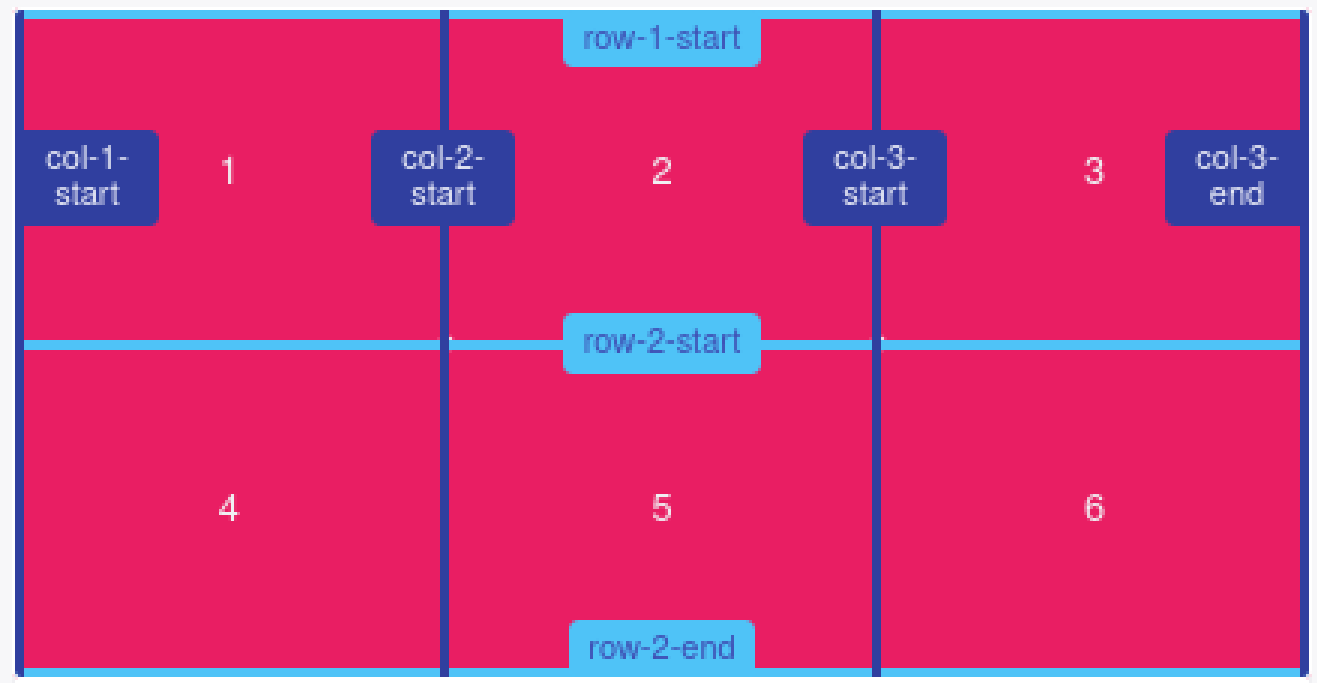


Naming Grid Lines

Grid lines can be named when defining the grid with the grid-template-rows and grid-template-columns properties. In line names, avoid keywords that appear in the specification (e.g. span) to not cause confusion.

Assigned line names must be wrapped in square brackets [name-of-line] and placed relative to the grid tracks.

```
grid-template-rows:    [row-1-start] 1fr [row-2-start]
                       1fr [row-2-end];
grid-template-columns: [col-1-start] 1fr [col-2-start]
                     1fr [col-3-start] 1fr [col-3-end];
```



Positioning grid items with Line Names

With named grid lines, items can be positioned by line names and numbers.

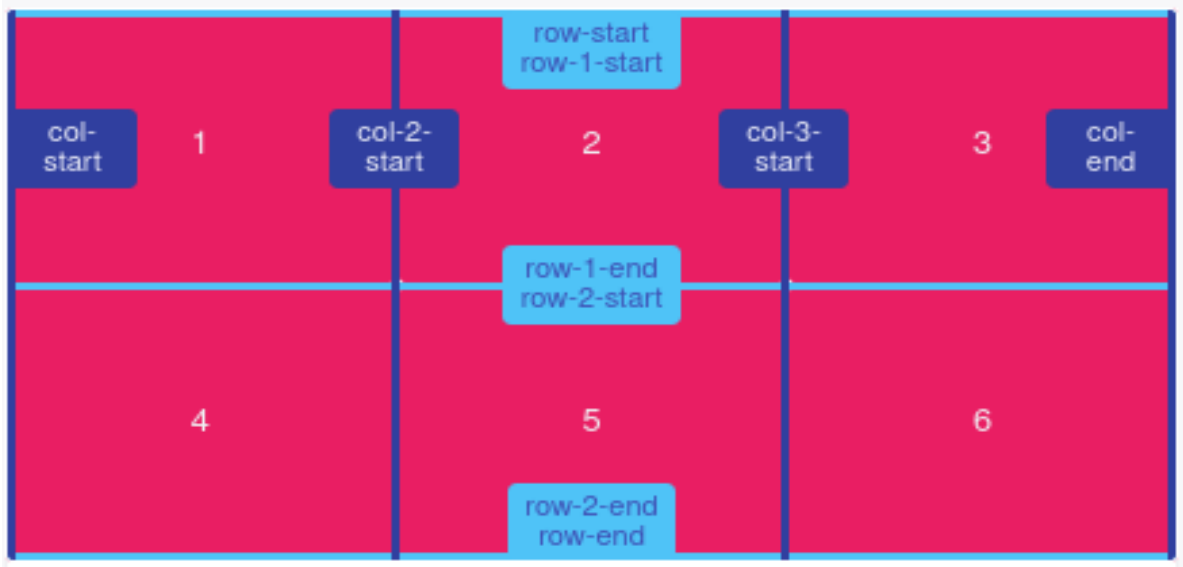
```
grid-row-start:    row-2-start;
grid-row-end:      row-end;
grid-column-start: col-2-start;
grid-column-end:   col-end;
```

Or using grid-row and grid-column shorthand properties

```
grid-row:    row-2-start / row-end;
grid-column: col-2-start / col-end;
```

Multiple Names to Grid Lines

- Multiple names can be assigned to grid lines by adding names within square brackets and separating each with a whitespace.



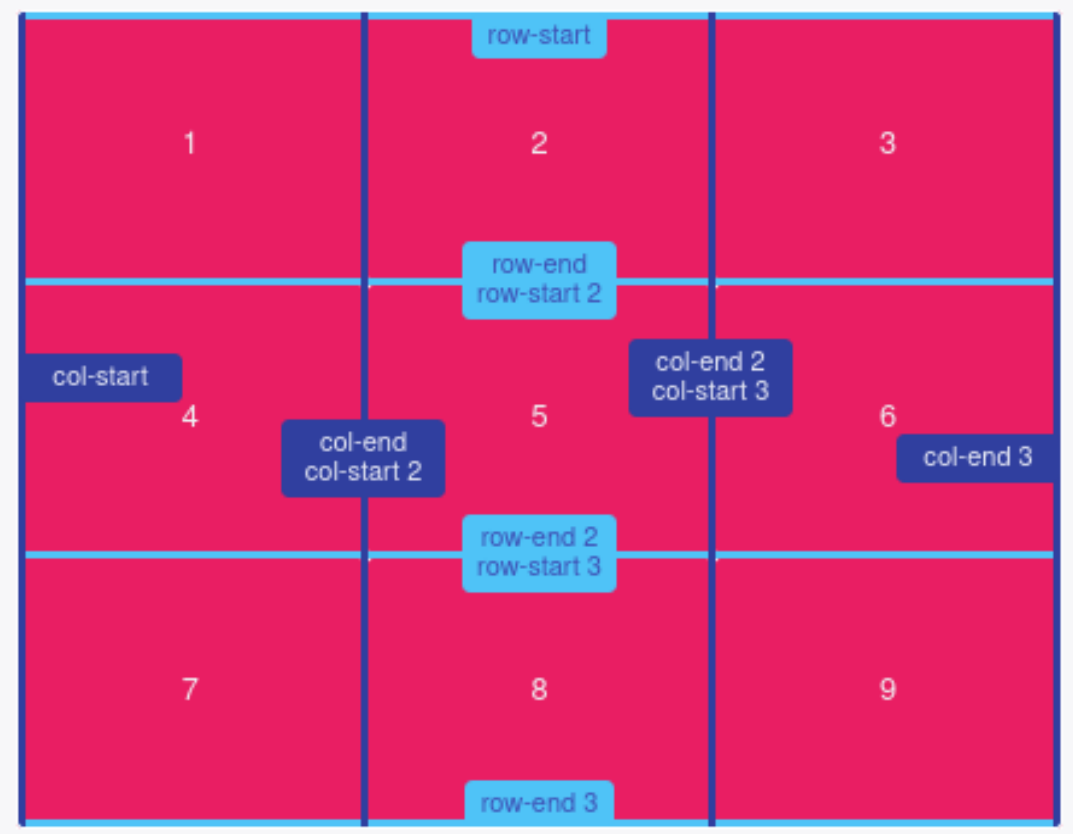

```
grid-template-rows: [row-start row-1-start] 1fr [row-1-end
                    row-2-start] 1fr [row-2-end row-end];

grid-template-columns: [col-start] 1fr [col-2-start] 1fr
                       [col-3-start] 1fr [col-end];
```

Grid Lines with Same Name

Lines can be assigned the same name with the repeat() function. This can save you time from having to name each line in track definitions.

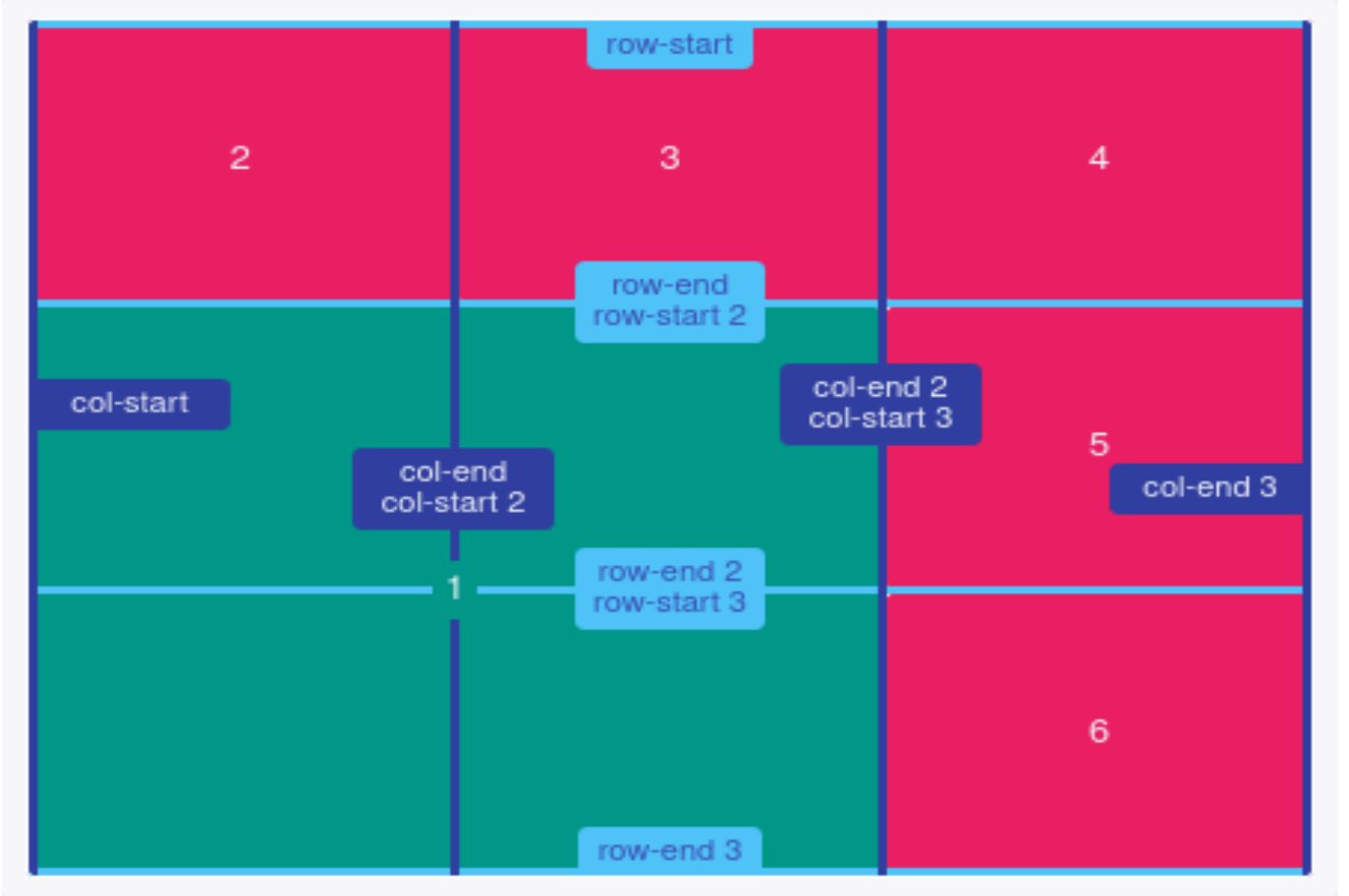
Lines with the same name are also assigned the a line's position/name's occurrence number, which allows it to be uniquely identified from another line with the same name.

```
grid-template-rows: repeat(3, [row-start]
                    1fr [row-end]);
grid-template-columns: repeat(3, [col-start]
                    1fr [col-end]);
```



Positioning grid items with Same Line Names

```
grid-row:    row-start 2 / row-end 3;
grid-column: col-start / col-start 3;
```



Exercise

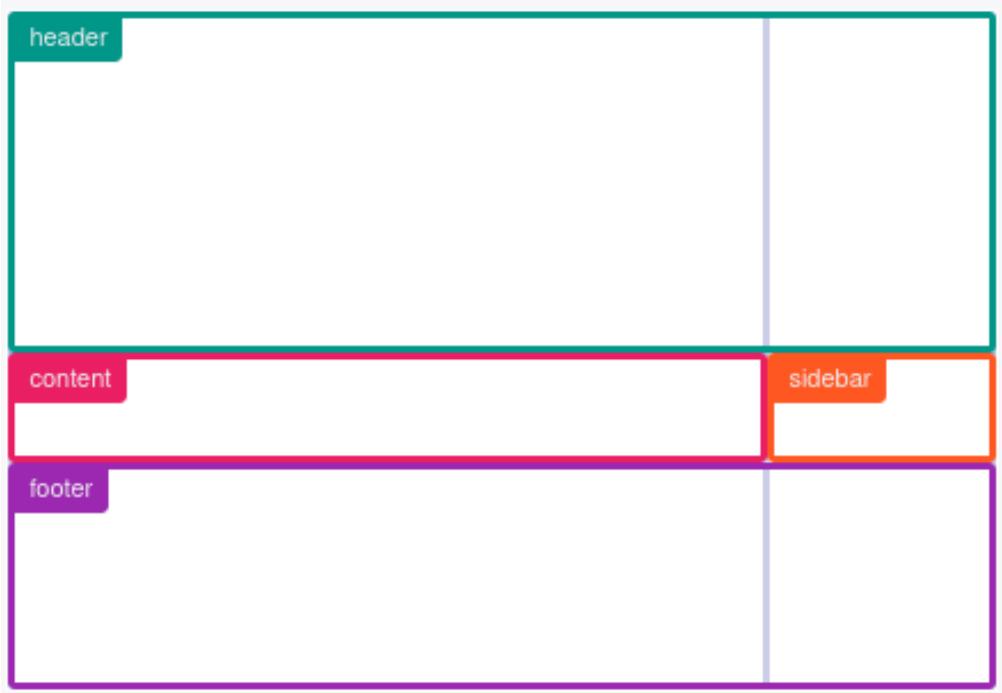
- 1. Multiple Names to the same column, position using them.
- 2. Repeat same name and position using them

Grid Areas

Use the grid-template-areas property to describe where on the Grid these elements should sit.

```
grid-template-areas:   "header header"
                       "content sidebar"
                       "footer footer";
grid-template-rows:    150px 1fr 100px;
grid-template-columns: 1fr 200px;
```

[Code Demo]



Positioning with Grid Areas

grid-area: a;

/* Equivalent to grid-row-start: a; grid-column-start: a; grid-row-end: a; grid-column-end: a; */

grid-area: 1 / 3 / -1;

/* Equivalent to grid-row-start: 1; grid-column-start: 3; grid-row-end: -1; grid-column-end: auto; */

grid-area: header-start / sidebar-start / footer-end / sidebar-end;

/*Equivalent to grid-row: header-start / footer-end; grid-column: sidebar-start / sidebar-end;*/

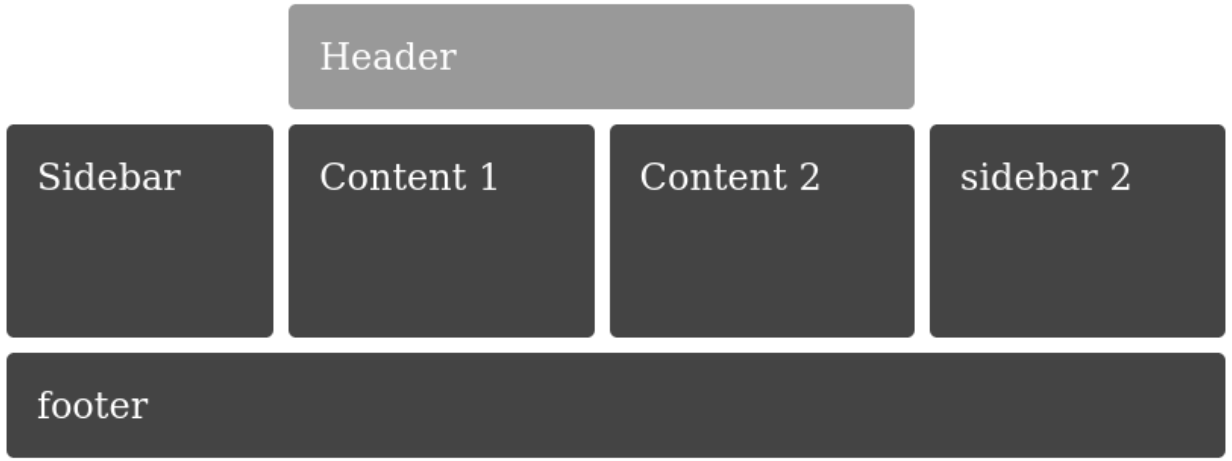
Exercise

Write the same properties using grid-row, grid-column for these

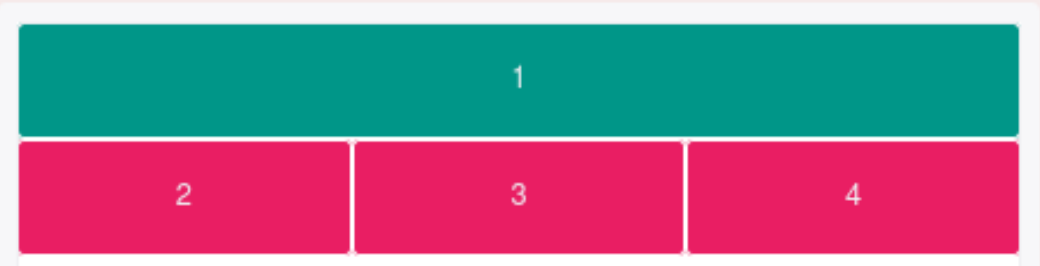
grid-area: auto;

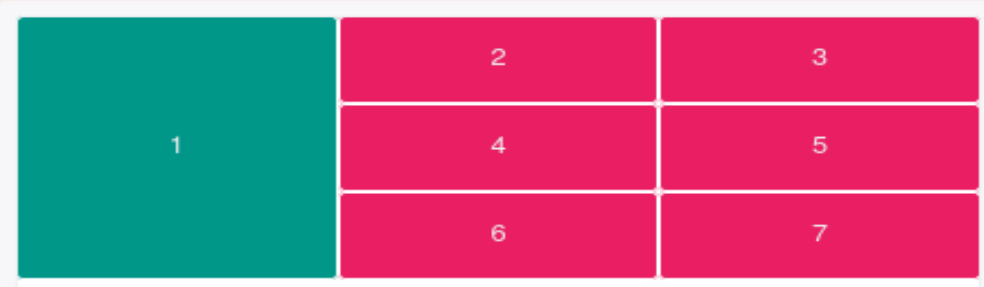
grid-area: 2 / 4;

Create this layout in tablet and larger screens and stack these up in the mobile screens.



Spanning Items Across Rows and Columns






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

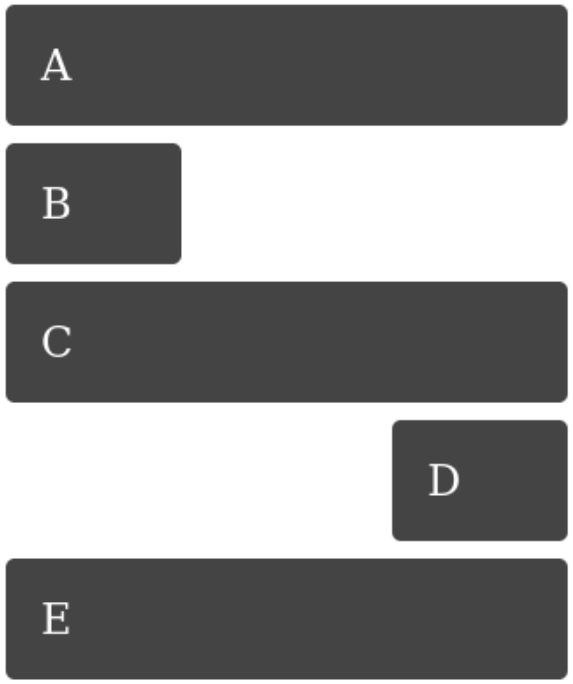
```
.a {
  grid-column: 1 / span 2;
}
```

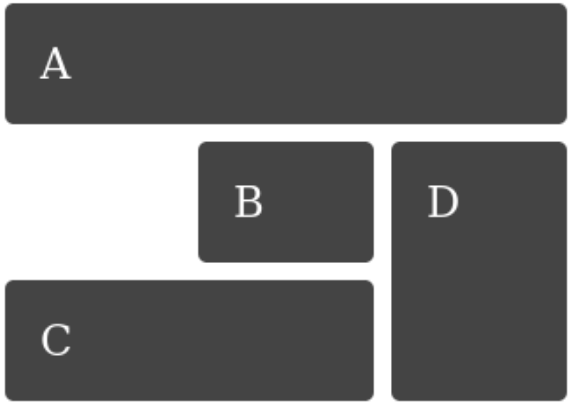
Using "span" keyword or by specifying grid line numbers or grid-row, grid-area as seen in the above slide.

Code Demo

Exercise

- Creat these layouts





Implicit Grid

If we don’t declare grid with grid-template-rows|columns|areas or place an item outside of the explicit grid the browser will create an **Implicit Grid** line or lines in order to hold that item.

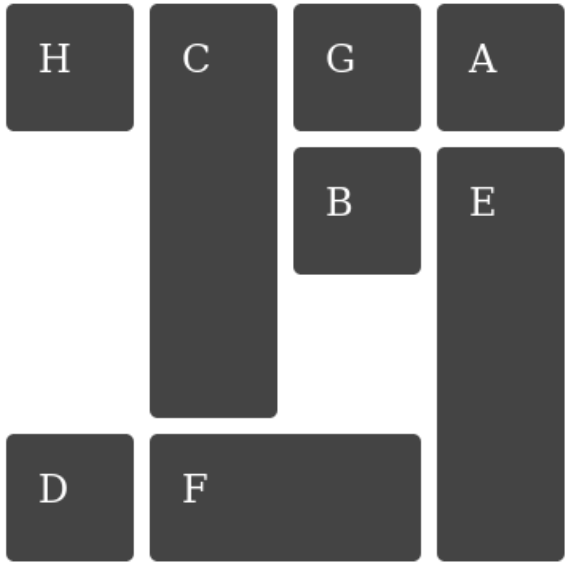
By default the implicit grid tracks created by the implicit lines will be auto sized. You can size the tracks with the grid-auto-columns and grid-auto-rows properties.

- grid-auto-columns: 100px
- grid-auto-rows: 100px

 Code Demo

Exercise

- Create a gird with 3 explicit rows and 3, 4, 5 columns of width 100px in 1,2,3rd row respectively
- Create an implicit grid , row: 5rem, column: 30%



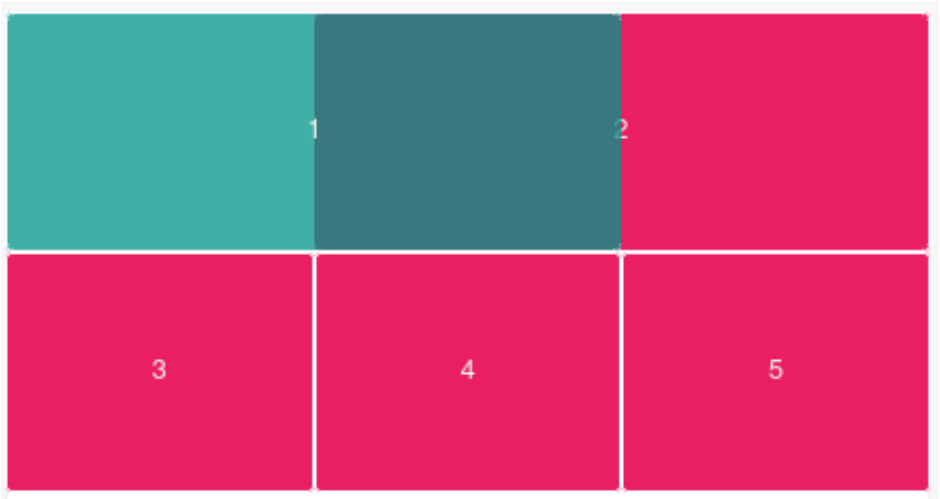
Layering and Ordering

A key thing in Grid is that the **order** of items in the source does not matter, as long as they are children of the element that has been declared as a grid.

layer items in the Grid using z-index to control the order that they stack.

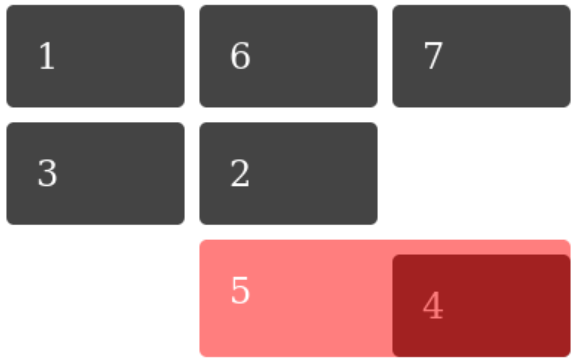
```
.item-1 {
    grid-column-start: 1;
    z-index: 1;
}
.item-2 {
    grid-column-start: 2;
}
```

[Code Demo]



Exercise

Order these boxes using order property and add an overlay as shown here



Box alignment align-items, align-self

Grid items can be aligned in the block dimension (perpendicular to the inline dimension) by using the align-self property on the grid item or align-items property on the grid container.

align-items : {stretch| start| end| center}
align-self: {stretch| start| end| center}

Both properties work only when there is enough space for the items to move.

Code Demo

Box Alignment justify-items, justify-self

Grid items can be aligned in the inline dimension(in horizontal dimension) by using the justify-self property on the grid item or justify-items property on the grid container.

- justify-items : {stretch| start| end| center}
- justify-self: {stretch| start| end| center}

- Both properties work only when there is enough space for the items to move.

Exercise

1) align-items, align-self

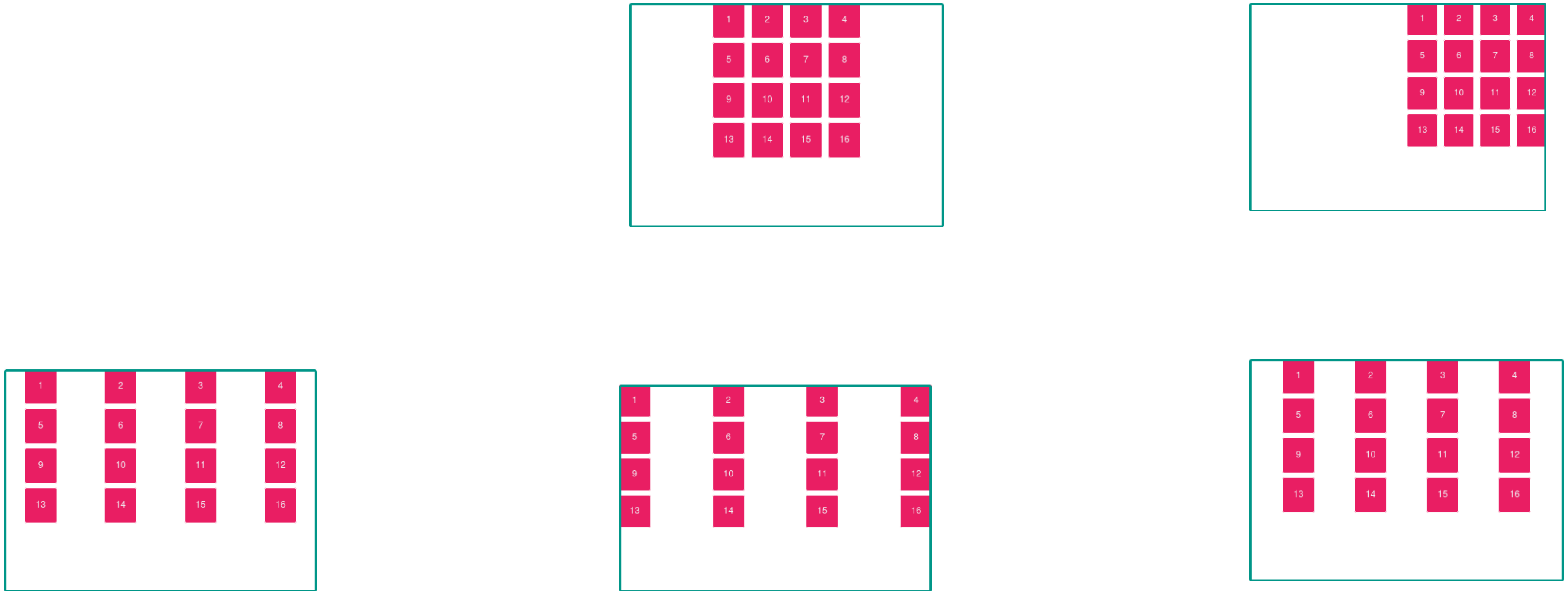
Create a 4x4 grid and align all grid items at the end while the box 2 and box 3 to be aligned at the start and center respectively.

2) Justify-items, justify-self

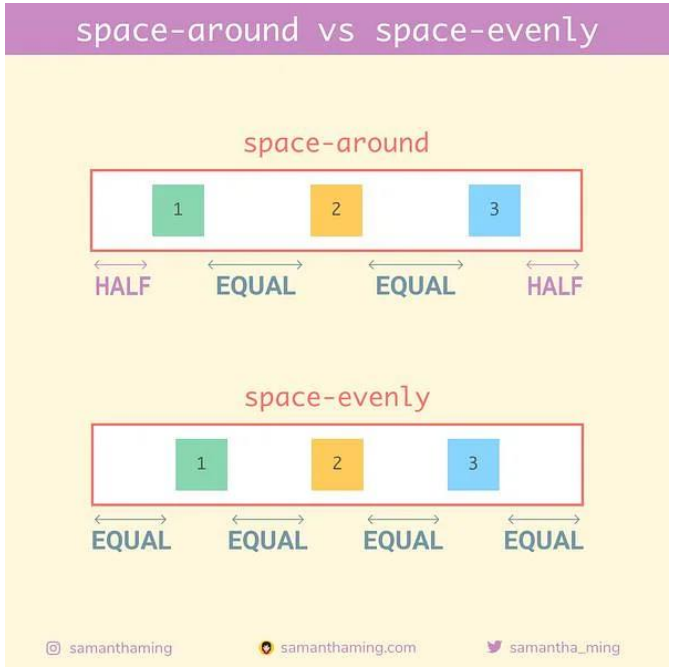
Create a 4x4 grid and align all grid items at the end while the box 2 and box 3 to be aligned at the start and center respectively.

Aligning Grid Tracks – justify-content

Grid tracks can be aligned relative to the grid container along the column axis using justify-content.

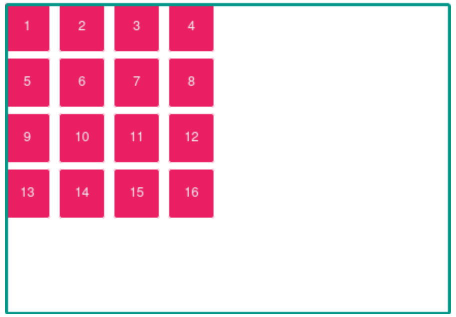


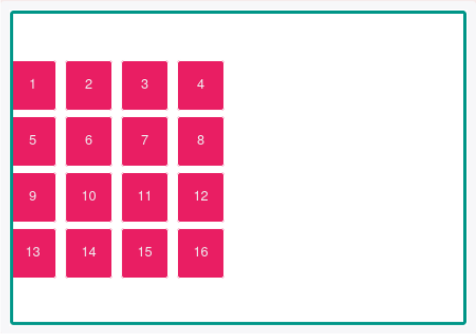
Space-around vs space-evenly

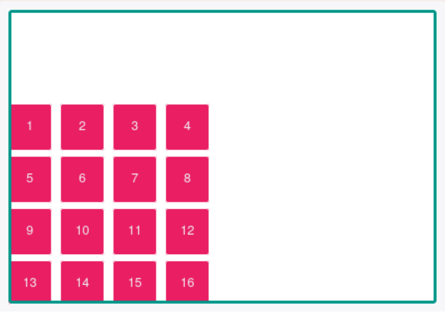


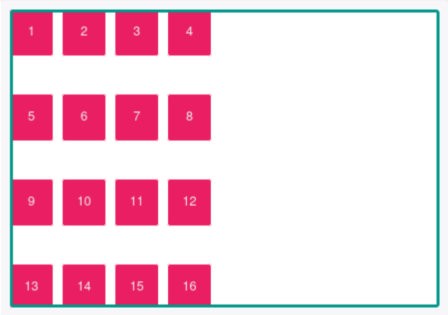
Aligning Grid tracks- align-content

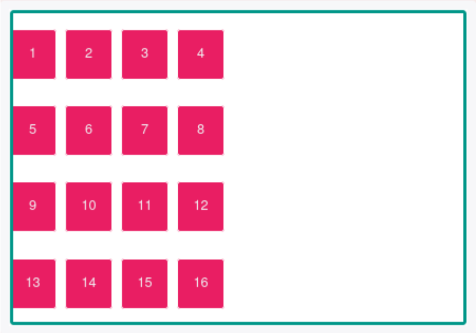
Grid tracks can be aligned relative to the grid container along the row axis using align-content

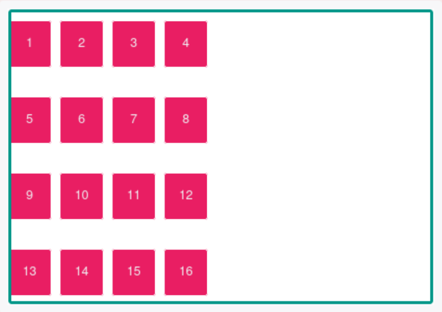












Things to know about Grid Containers

Grid containers are not block containers, and so some properties that were designed with the assumption of block layout don't apply in the context of grid layout. In particular:

- float and clear have no effect on a grid item. However, the float property still affects the computed value of display on children of a grid container, as this occurs *before* grid items are determined.
- vertical-align has no effect on a grid item.
- the ::first-line and ::first-letter pseudo-elements do not apply to grid containers, and grid containers do not contribute a first formatted line or first letter to their ancestors.

References

- https://gridbyexample.com/examples/
- https://learncssgrid.com/