

Responsive Web Design

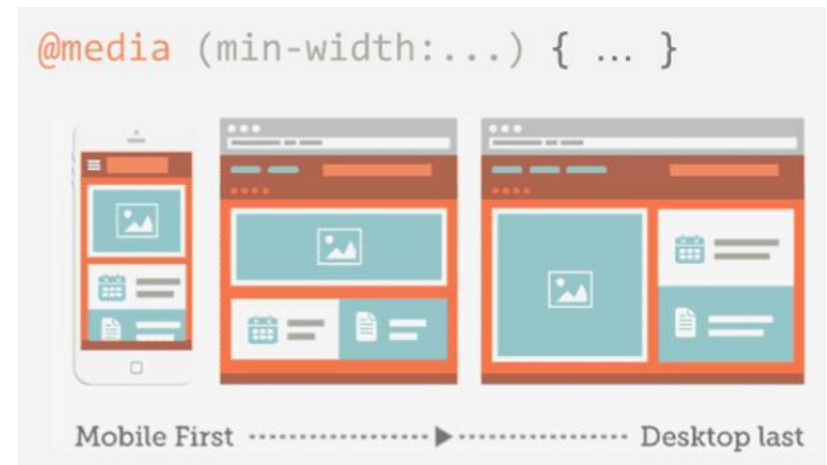
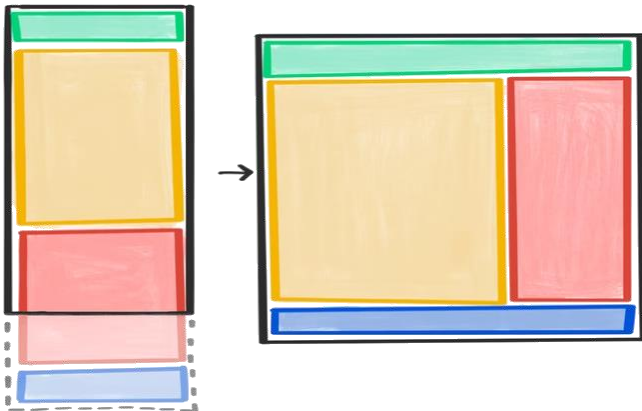


Outline

1. Flexbox
2. Grid
3. Media Queries
4. Common Layout Patterns

Responsive Web Design (RWD)

- RWD is an approach to **serve different layouts for different screen sizes**
 - **Optimize the viewing experience on range of devices:** mobile, desktop, tablet, TV...
 - Can be accomplished using CSS **grid/flexbox** & **media queries**
 - **Mobile-first layouts** work well on all screen widths: start with single column layout for smaller screens



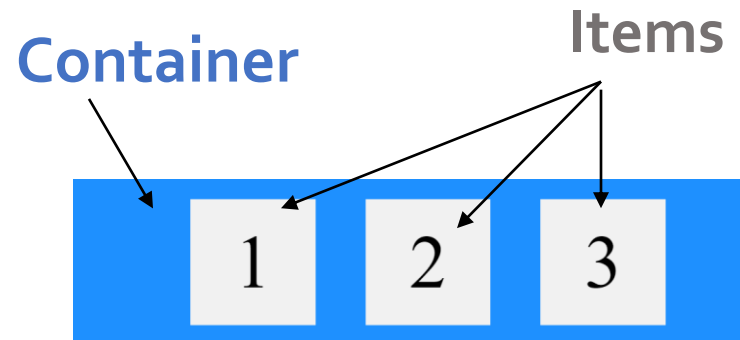


Flexbox

- The Flexbox provide an efficient way to **lay out, align** and **distribute space** among items in a container
 - Defines **one-dimensional layout**
 - A flex container expands items to fill available free space or shrinks them to prevent overflow

```
.flex-container {  
  display: flex;  
  gap: 1rem;  
  justify-content: center;  
}
```

```
<div class="flex-container">  
  <div>1</div>  
  <div>2</div>  
  <div>3</div>  
</div>
```



https://www.w3schools.com/css/css3_flexbox.asp

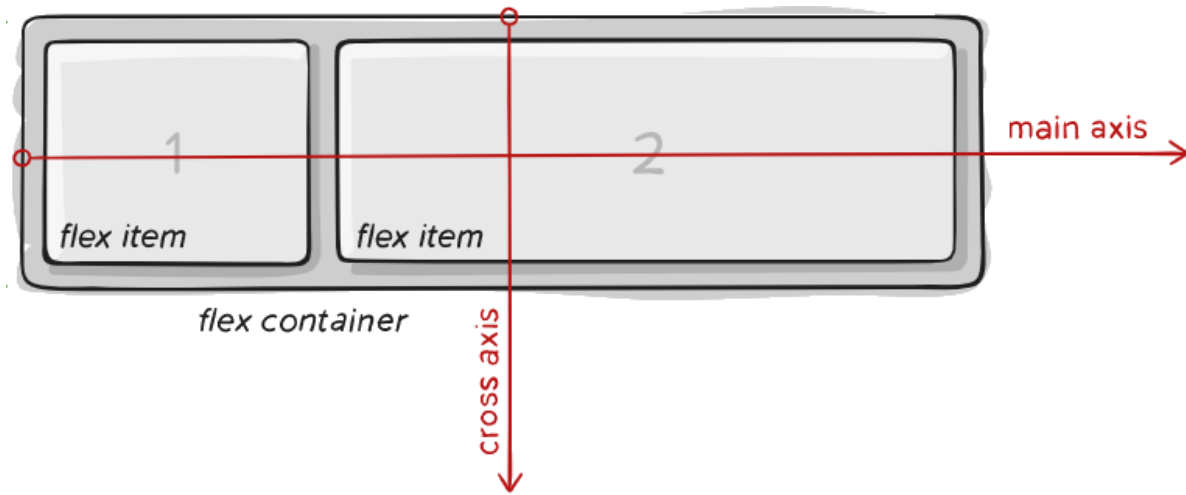
Flex Container Properties

- **flex-direction**: either row (default) or column
- **flex-wrap**: By default, **nowrap**: all flex items will be on 1 line. Assign **wrap** to allow flex items to wrap onto multiple lines
- **justify-content**: aligns and arranges flex-items along the main axis
- **align-items**: aligns items within a flex line, along the cross-axis
- **align-content**: aligns and manages spacing between **multiple lines** when items wrap

flex-direction: sets the main-axis.



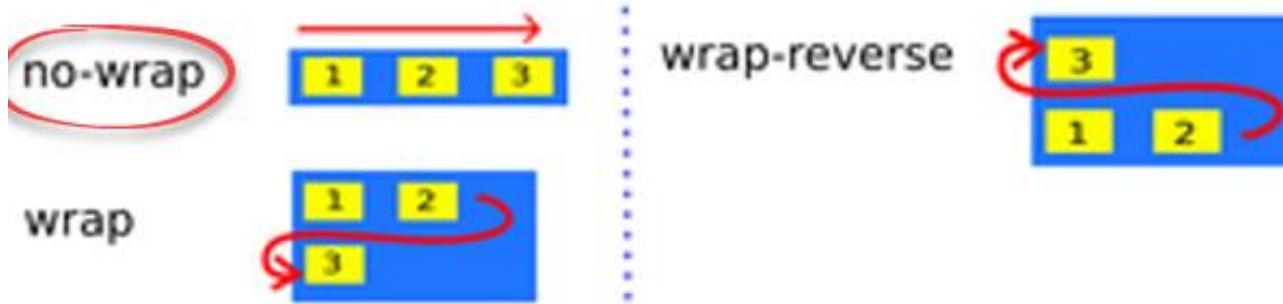
row (default):
horizontal
alignment



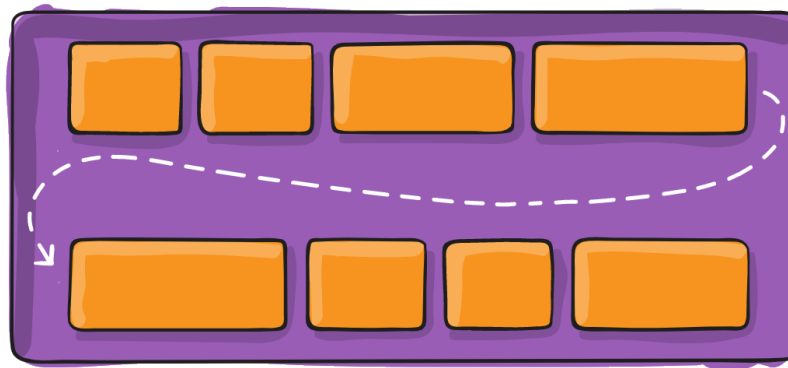
column: vertical alignment



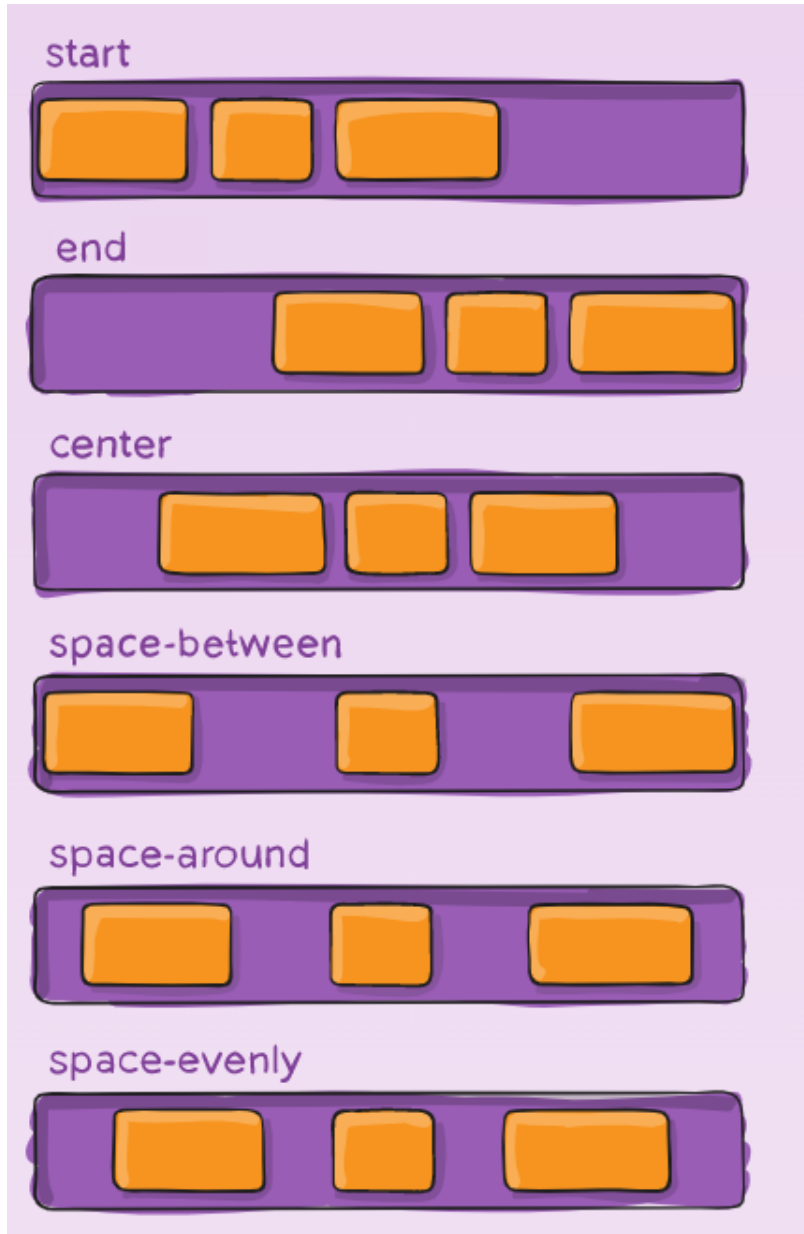
flex-wrap: allows the items to wrap as needed.



- **nowrap** (default): all flex items will be on one line
- **wrap**: flex items will wrap onto multiple lines

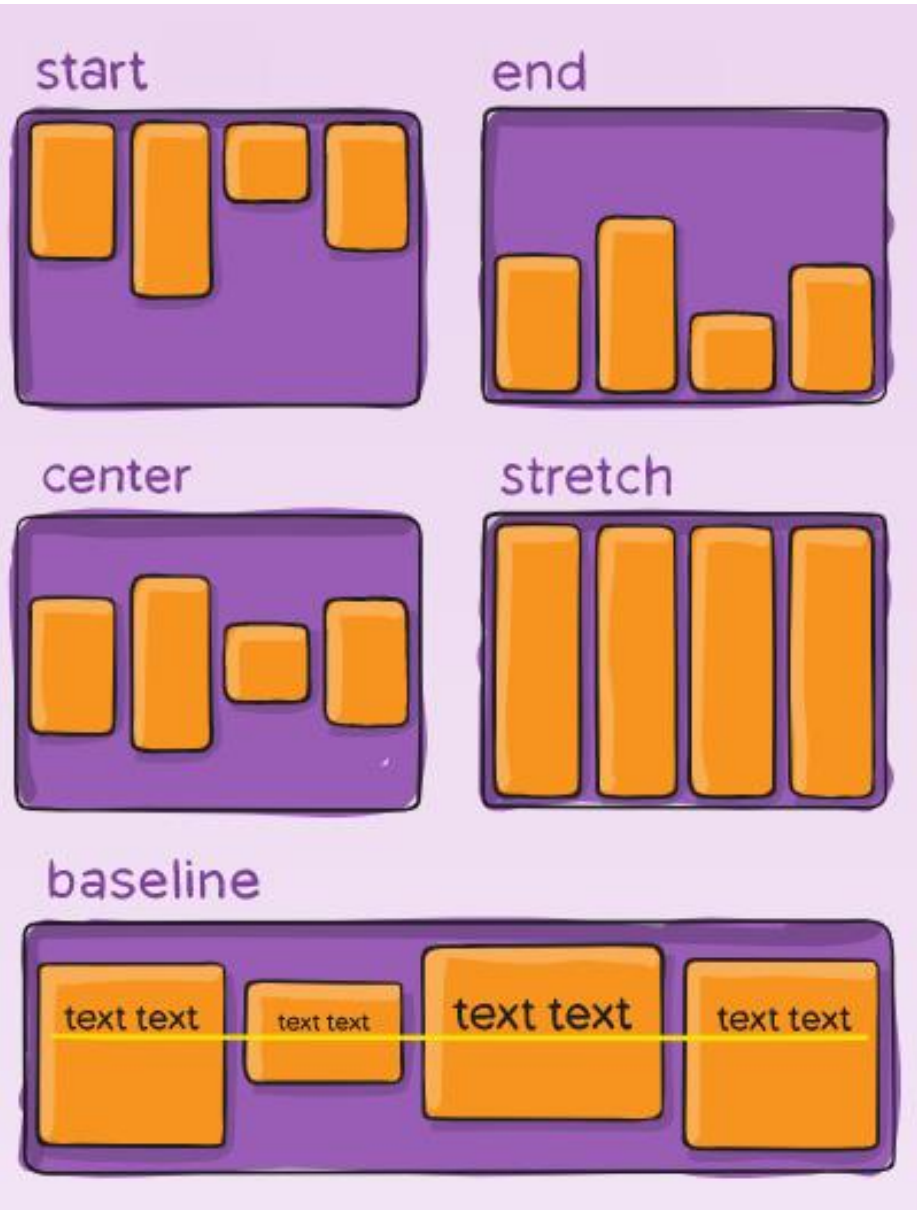


justify-content



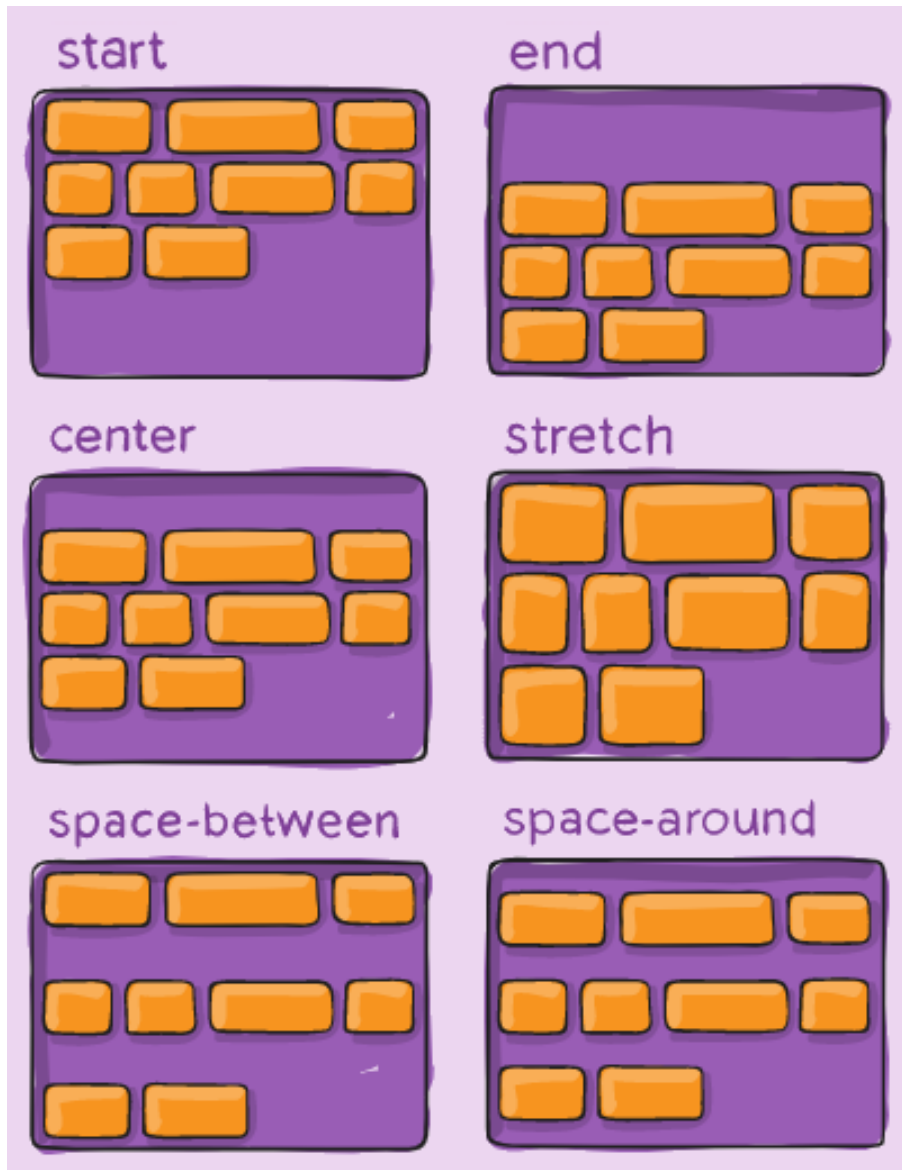
- Align items and distribute extra leftover space along the **main axis**
- **start** is the **default**: items are packed toward the start

align-items



- Aligns items within a flex line, along the cross-axis
- stretch** is the **default**: flex items stretch to fill the flex line

align-content



- Aligns and distributes extra leftover free space **between the lines** when items wrap
- stretch** is the **default**: lines stretch to fill the container

flex items - order & flex-basis

order: changes the order of flex items.

```
.item {  
  order: 3 // the default is 0  
}
```




Try it: https://www.w3schools.com/css/tryit.asp?filename=trycss3_flexbox_order_2

order: changes the display order of the flex item

flex-basis: defines the flex item default size before remaining space is distributed. It accepts:

- specific values : pixels, rem, %
- auto: defaults to width or height property
- content : automatic sizing based on its content

A dark gray rectangular box containing two vertical purple bars. The bar on the left is narrower than the bar on the right.

first item 20%, second item 40%

flex items - grow & shrink

flex-grow: allows item to grow using remaining space.

```
.item-1 { flex-grow: 0 }  
//default
```



```
.item-1 { flex-grow: 1 }
```



Tip: If all items have flex-grow: 1, the remaining space is distributed equally.

flex-grow: determines how the flex item is allowed to grow

flex-shrink: defines the ability for a flex item to shrink.

```
.one { flex-shrink: 1; }  
.two { flex-shrink: 2; }  
.three { flex-shrink: 3; }  
.four { flex-shrink: 4; }
```

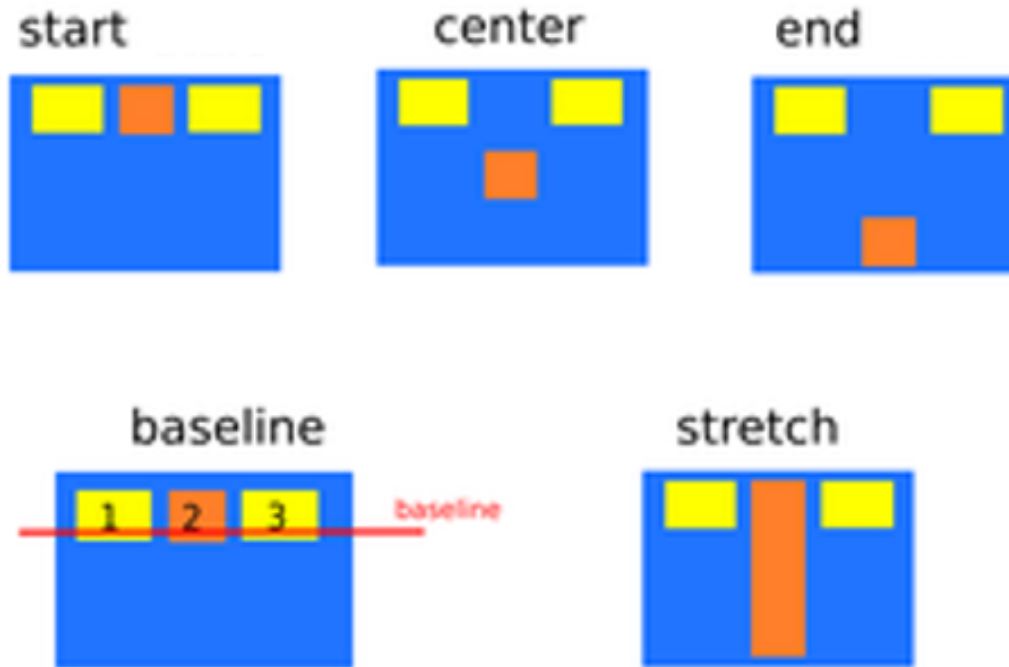


Tip: Defaults to 1. The highest the value the more it shrinks compared to siblings.

flex-shrink: allows an item to shrink if necessary

flex items - align-self

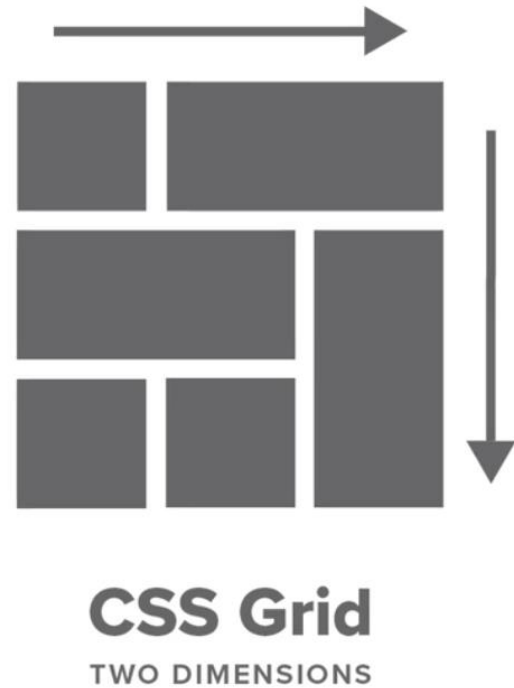
- **align-self**: overrides default alignment (or the one specified by align-items) for a specific item along the Cross Axis





CSS Grid

- CSS Grid is a **two-dimensional layout** system to design the page layout
- Can specify columns/rows template
- Grid elements can be **auto-placed** or **explicitly placed** using grid lines or grid areas
- Easy control of **space distribution** and **alignment** of items



Grid container & Grid items

- Grid **container** is defined by setting the *display* property of the container element to *grid*
- Grid item = *Element that is a direct descendant of the grid container*

```
.page {  
    display: grid;  
}
```

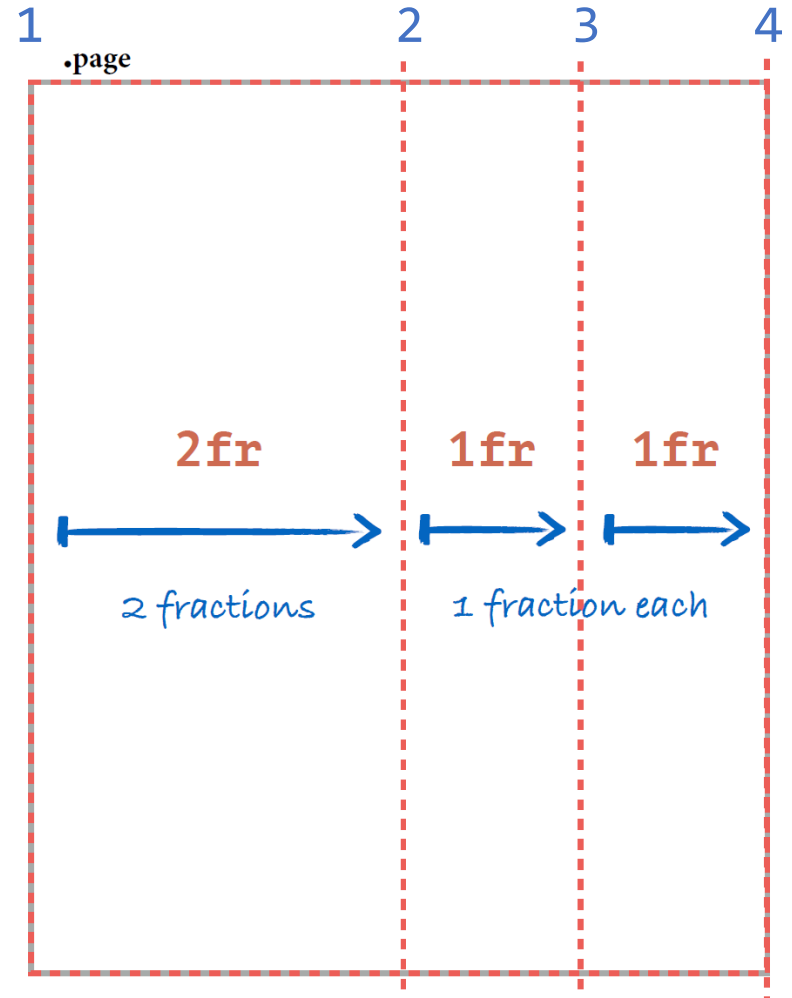
```
.page  
<div class="page">  
  <header class="head">  
  </header>  
  
  <main class="main-content">  
  </main>  
  
  <aside class="sidebar">  
  </aside>  
  
  <footer class="footer">  
  </footer>  
</div>
```

Grid Template Columns

`grid-template-columns:`
2fr 1fr 1fr;

Defines the number of **grid columns** and their desired **size** (em, px, %, **fr**)

```
.grid-container {  
  display: grid;  
  grid-template-columns: 2fr  
1fr 1fr;  
}
```

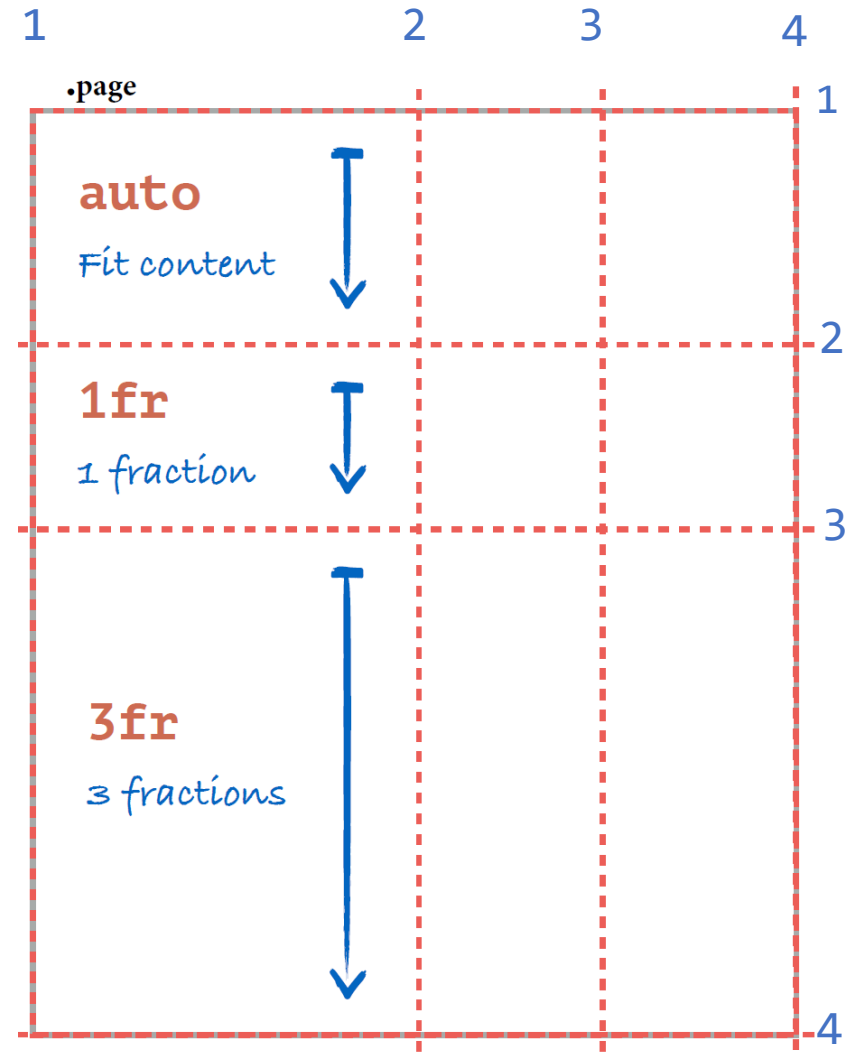


Grid rows

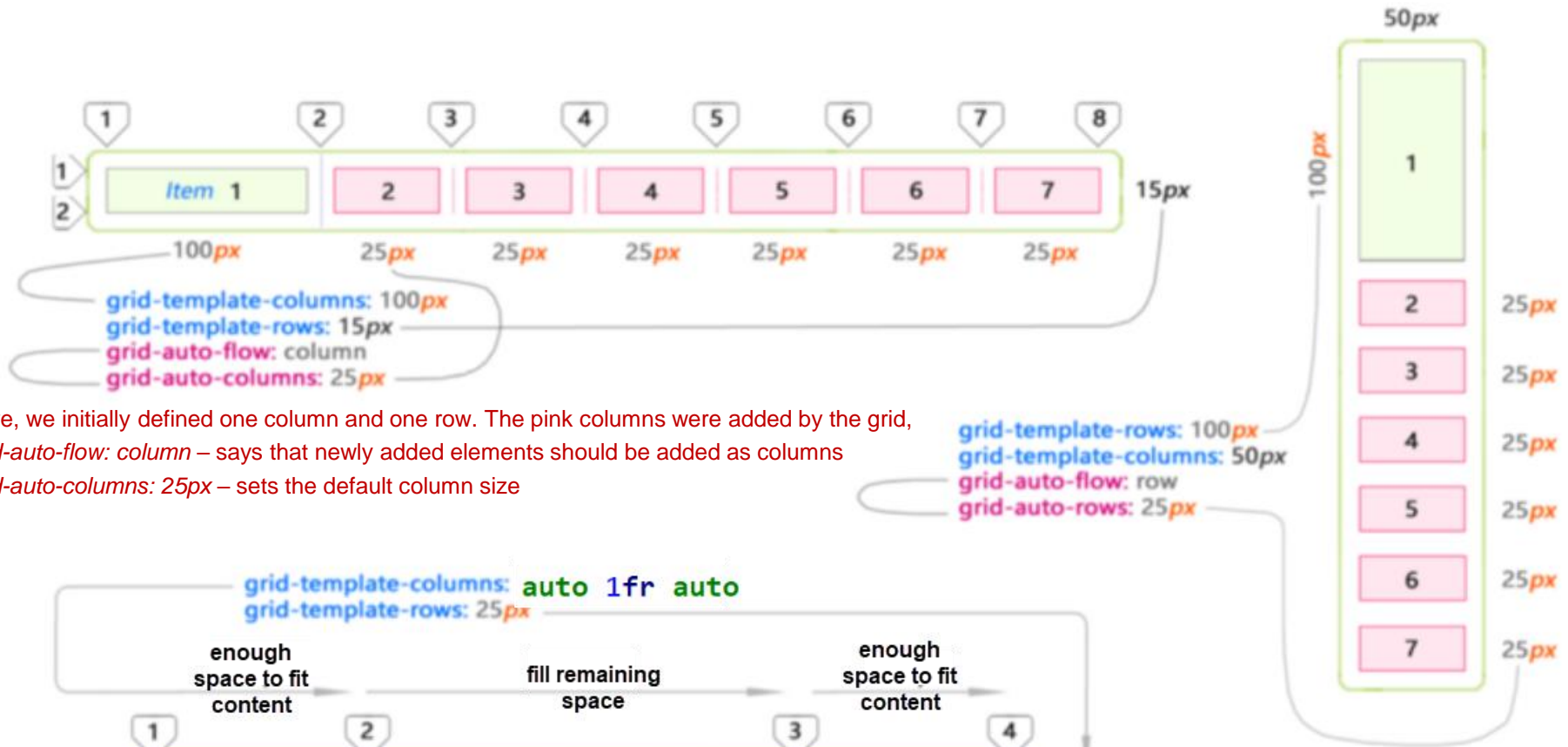
`grid-template-rows:`

`auto 1fr 3fr;`

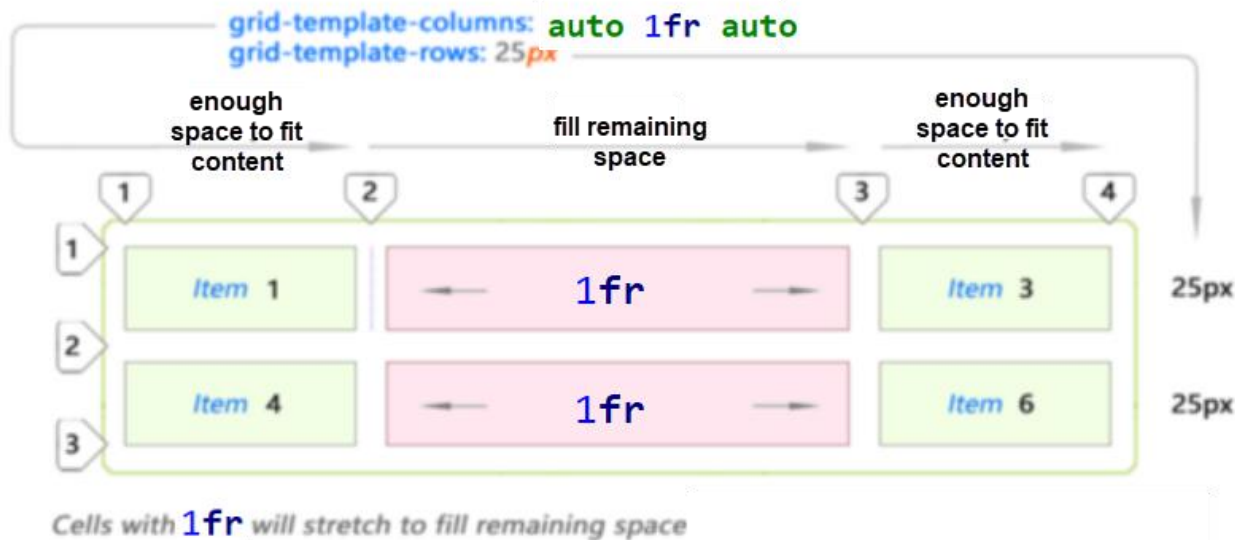
- Defines **grid rows** and their desired **size** (em, px, %, **fr**)
- Optional, only define it when really needed



grid-template-rows & grid-template-columns



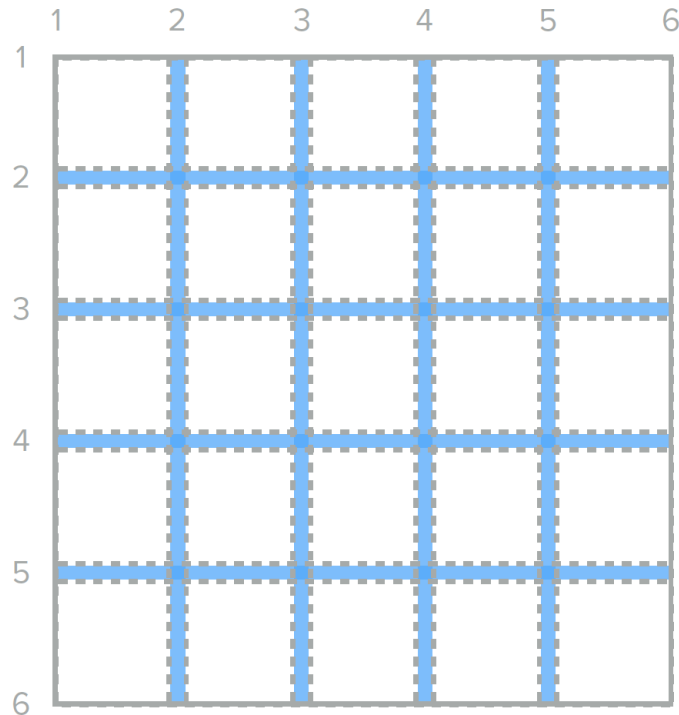
- Here, we initially defined one column and one row. The pink columns were added by the grid,
- *grid-auto-flow: column* – says that newly added elements should be added as columns
- *grid-auto-columns: 25px* – sets the default column size



grid-gap

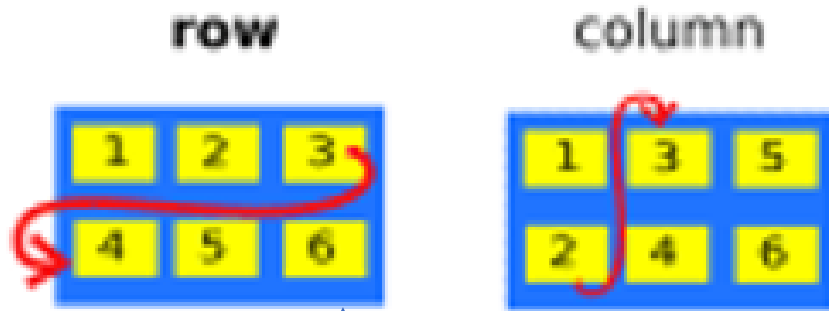
```
.page {  
  display: grid;  
  grid-gap: .5rem;  
}
```

Grid-gap defines space (i.e., gutter or gap) between grid tracks (as shown in **blue**)



grid-auto-flow

Defines how to automatically place grid items that aren't explicitly placed
(**row** is the **default**)



Grid items automatically populate grid from top left to bottom right based on HTML source order.

Placing Items using Grid Lines

```
.head {
```

```
  grid-column: 2/4;
```

```
  grid-row: 1/2;
```

```
}
```

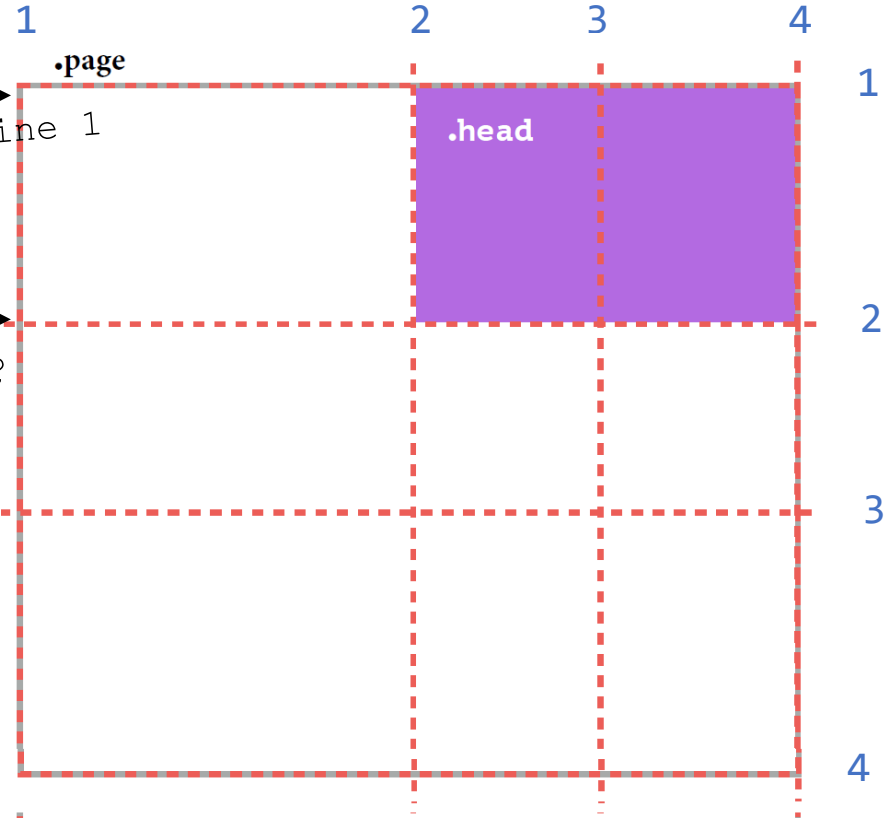
```
grid-column: 2/4;
```

```
Or grid-column: 2/ span 2;
```

Place item in the grid using the **start** and **end** grid lines for columns and rows

Start at column line 2

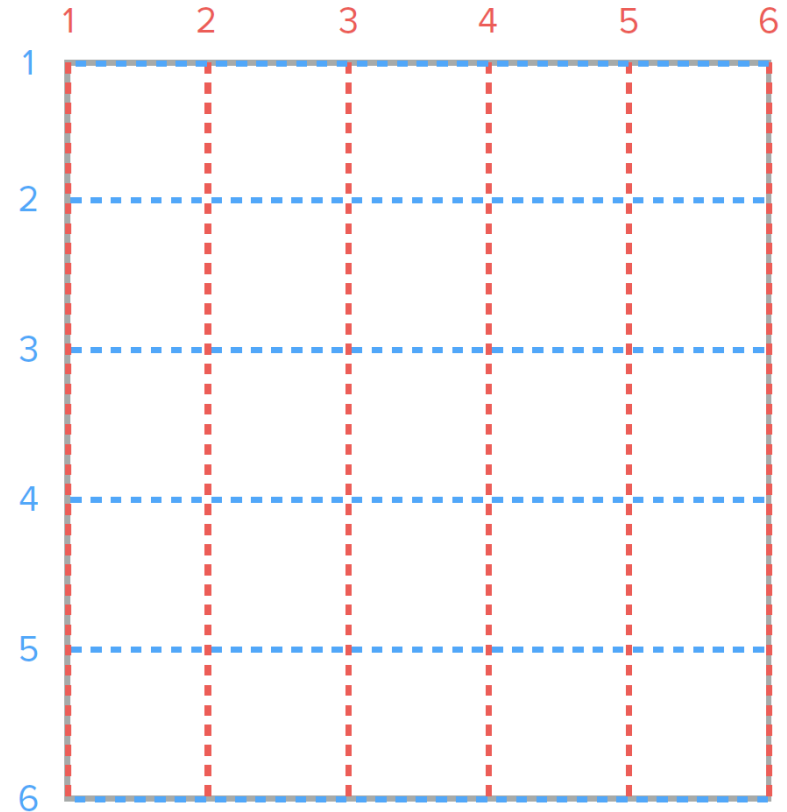
End at column line 4



grid-column: 1 / -1
=> spans all available columns

Grid line

- Horizontal (**row**) or vertical (**column**) line separating the grid into sections
- Grid lines are referenced by numbers, starting and ending with the outer borders of the grid



Example - Placing Items using Grid Lines

```
.container {  
  display: grid;  
  grid-template-columns: auto 1fr auto;  
  grid-template-rows: auto 1fr auto;  
}
```

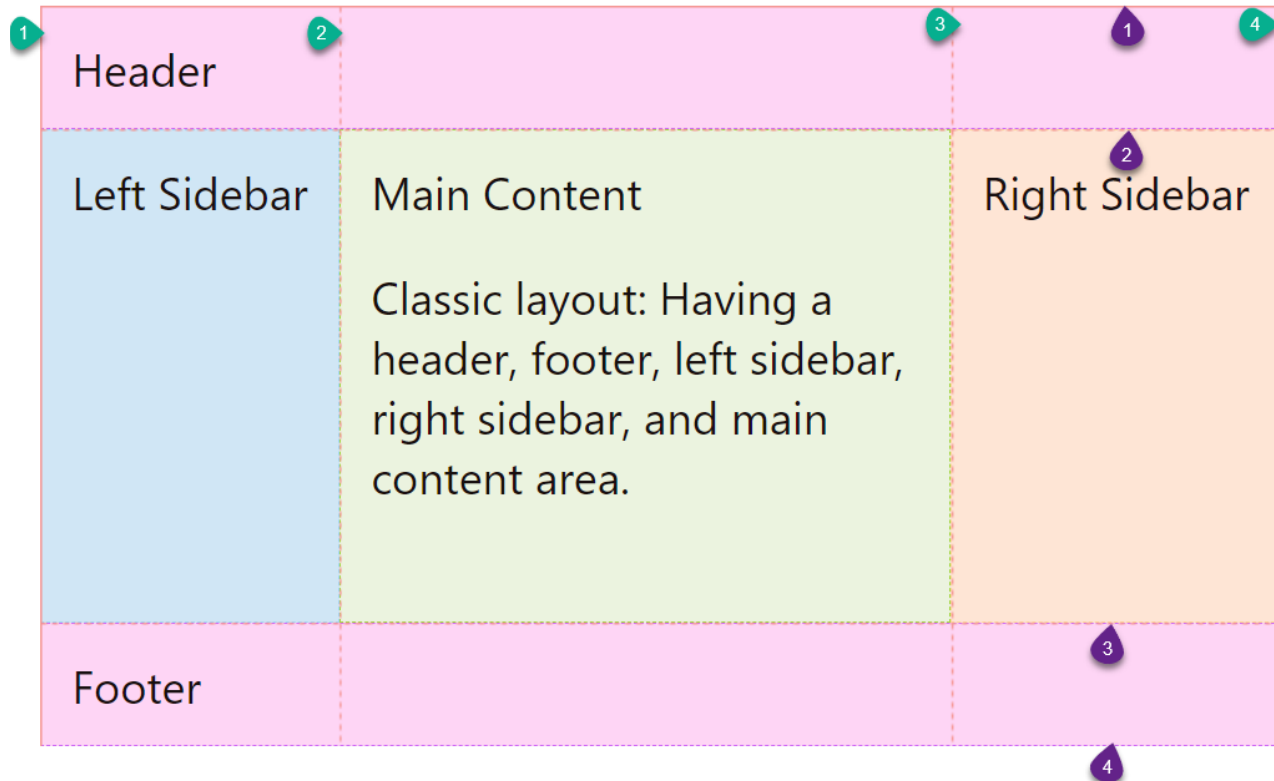
```
header {  
  grid-column: 1 / span 3;  
}
```

```
.left-side {  
  grid-column: 1 / 2;  
}
```

```
main {  
  grid-column: 2 / 3;  
}
```

```
.right-side {  
  grid-column: 3 / 4;  
}
```

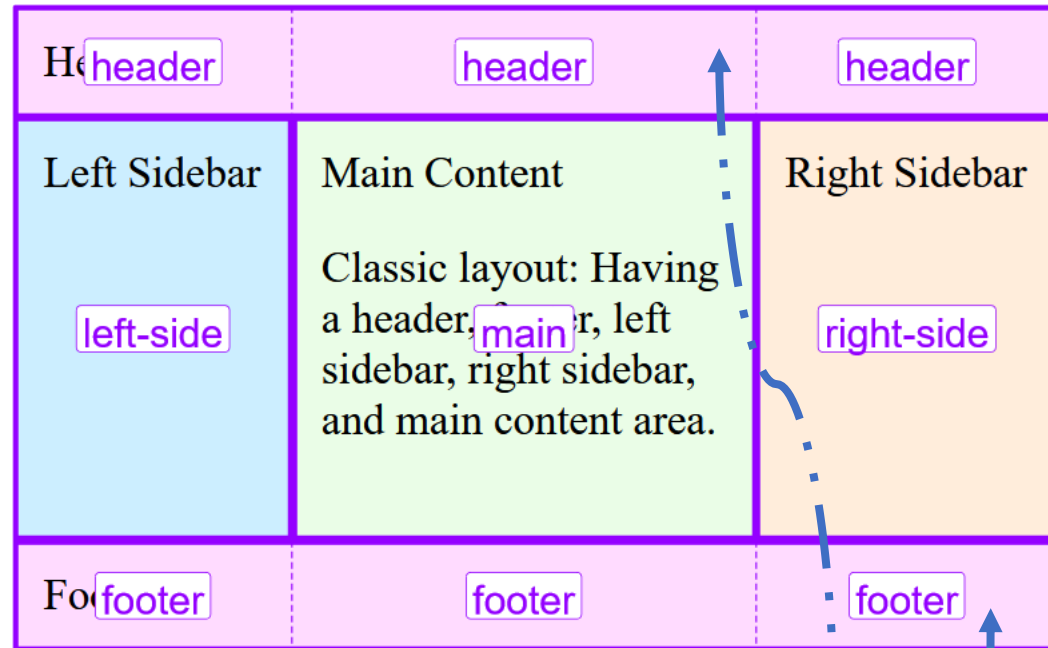
```
footer {  
  grid-column: 1 / span 3;  
}
```



Placing Items using Grid areas

grid-template-areas
is used to **define** named grid areas

Then place items in the grid areas



```
.container {  
  display: grid;  
  grid-template: auto 1fr auto  
    / auto 1fr auto;  
  grid-template-areas:  
    "header      header      header"  
    "left-side   main        right-side"  
    "footer      footer      footer";  
}
```

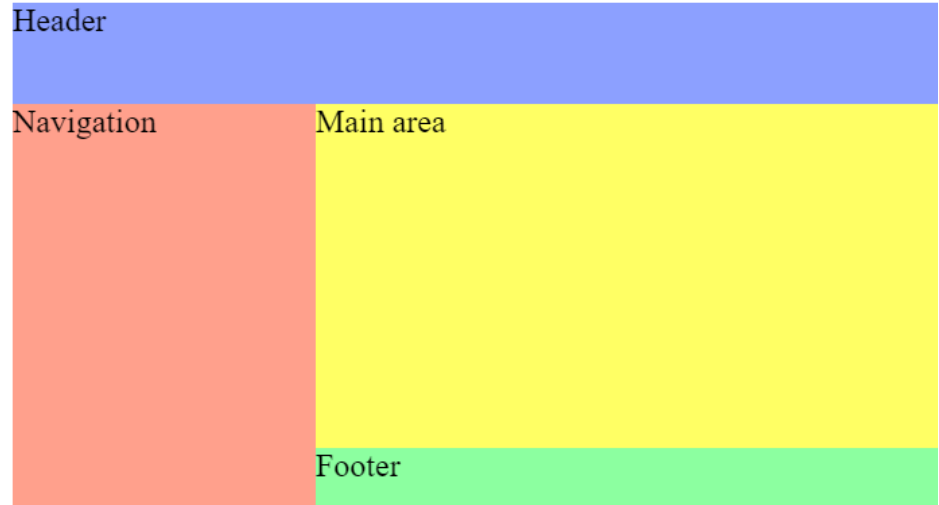
```
header {  
  grid-area: header;  
}  
  
.left-side {  
  grid-area: left-side;  
}  
  
main {  
  grid-area: main;  
}  
  
.right-side {  
  grid-area: right-side;  
}  
  
footer {  
  grid-area: footer;  
}
```

Grid areas

Result

HTML

```
<section id="page">
  <header>Header</header>
  <nav>Navigation</nav>
  <main>Main area</main>
  <footer>Footer</footer>
</section>
```



CSS

Source: <https://developer.mozilla.org/en-US/docs/Web/CSS/grid-template-areas>
https://www.w3schools.com/cssref/tryit.php?filename=trycss_grid-template-areas3

```
#page {
  display: grid;
  width: 100%;
  height: 250px;
  grid-template-areas:
    "head head"
    "nav main"
    "nav foot";
  grid-template-rows: 50px 1fr 30px;
  grid-template-columns: 150px 1fr;
}
```

```
#page > main {
  grid-area: main;
  background-color: #ffff64;
}

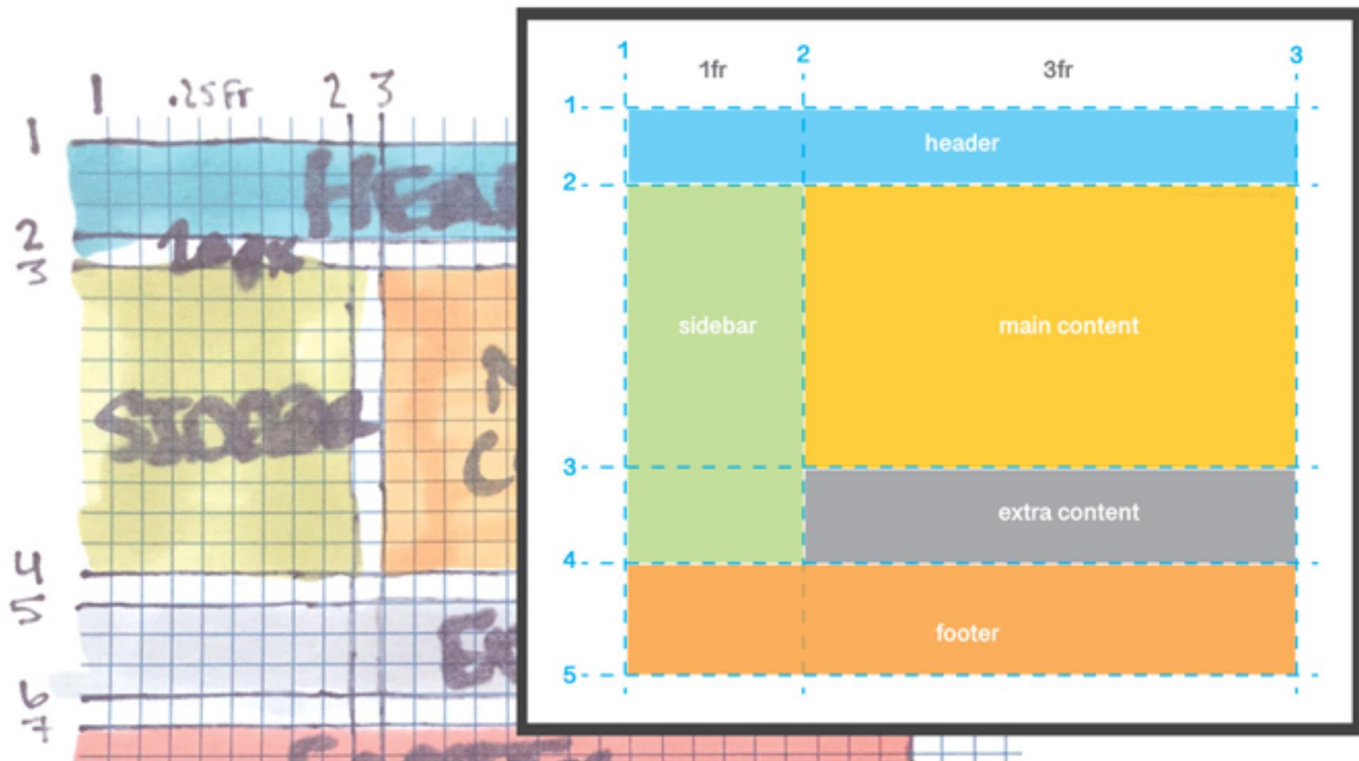
#page > footer {
  grid-area: foot;
  background-color: #8cffa0;
}
```

```
#page > nav {
  grid-area: nav;
  background-color: #ffa08c;
}

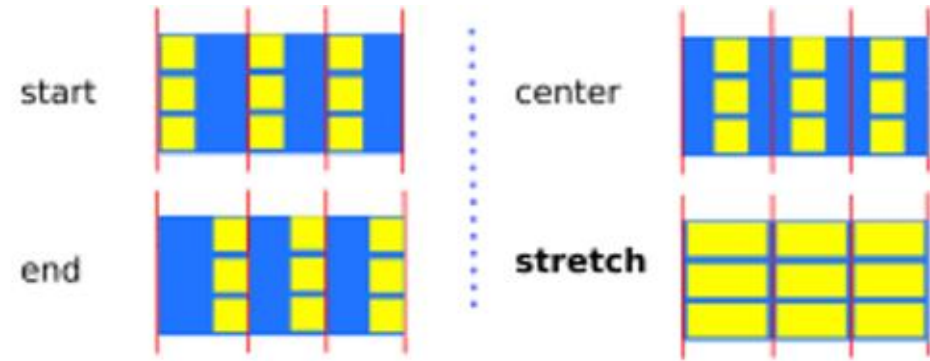
#page > header {
  grid-area: head;
  background-color: #8ca0ff;
}
```

Grid areas

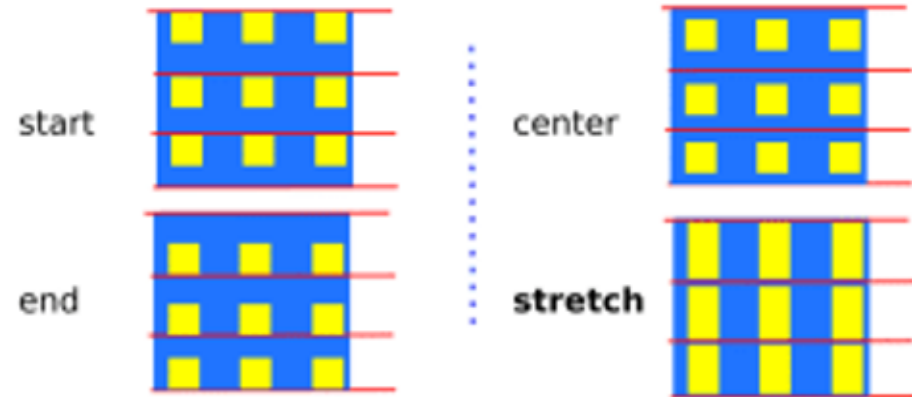
- Defining grid areas and using them to place elements is **best way** to design the page layout as it allows direct translation of the paper-based design to a CSS grid



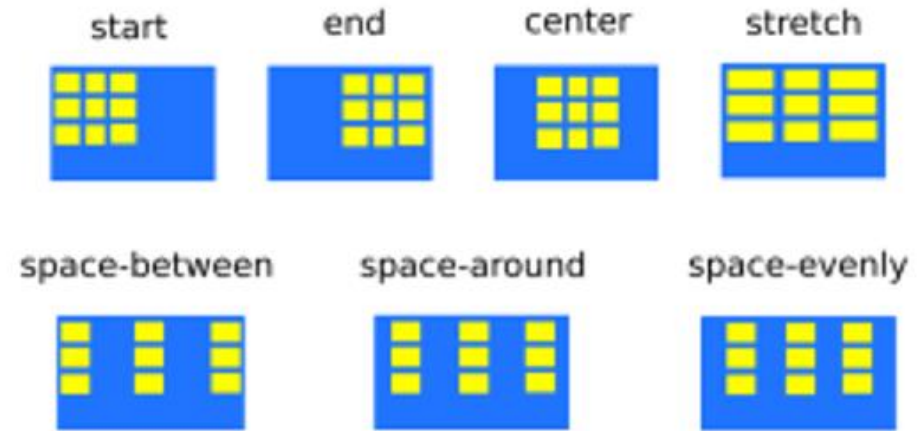
- **justify-items**
defines alignment along the **row axis**



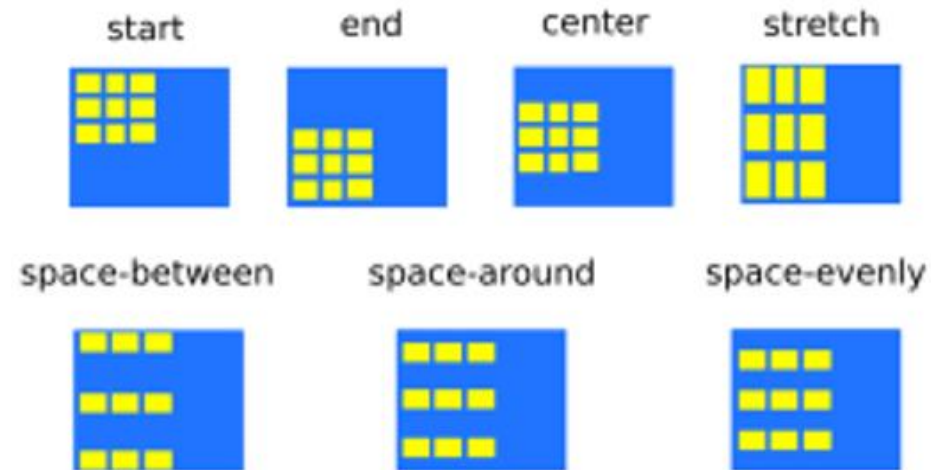
- **align-items**
defines alignment along the **column axis**



- **justify-content**
justifies **all** grid content
on **row axis** (if container has
extra space)

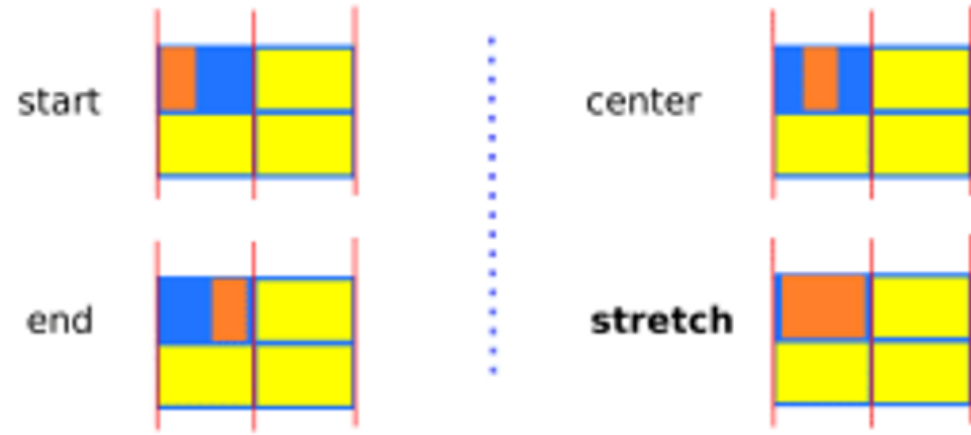


- **align-content**
justifies **all** grid content
on **column axis** (if container
has extra space)



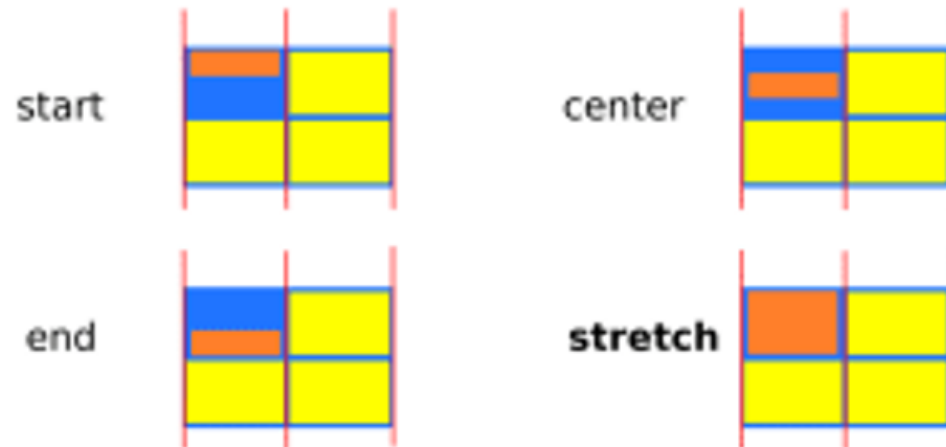
- **justify-self**

aligns **an item** inside a single cell along the **row axis**



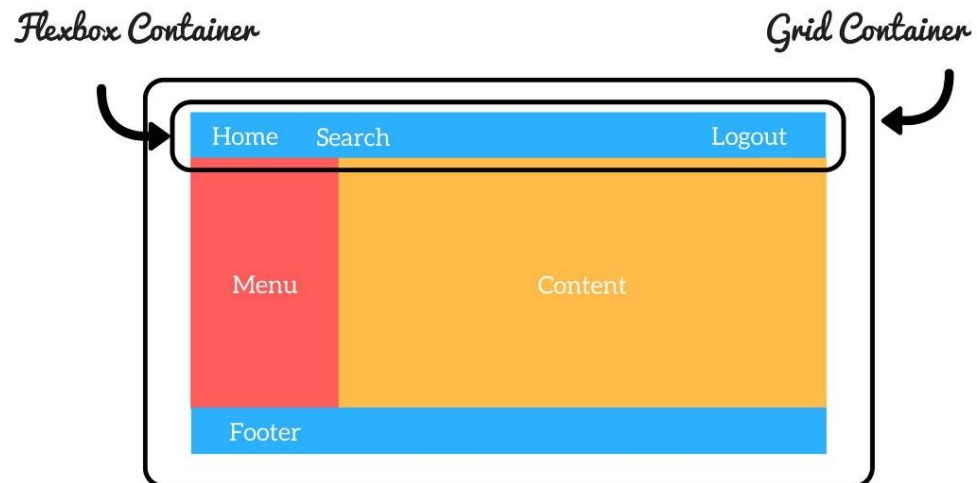
- **align-self**

aligns an item inside a single cell along the **column axis**



Grid vs Flexbox

- Grid allows defining a **two-dimensional** layout with columns and rows, unlike flexbox which is a **one-dimensional layout** (either in a column or a row).
- In practice you combine these layout models. Often you can use a Flexbox container inside a Grid container
 - Grid is often used for the overall page layout (i.e., **Macro layouts** describing the larger, page-wide organization) while the **flexbox is used for small-scale** one-dimensional layouts (e.g., menu or card layout)



Media Queries

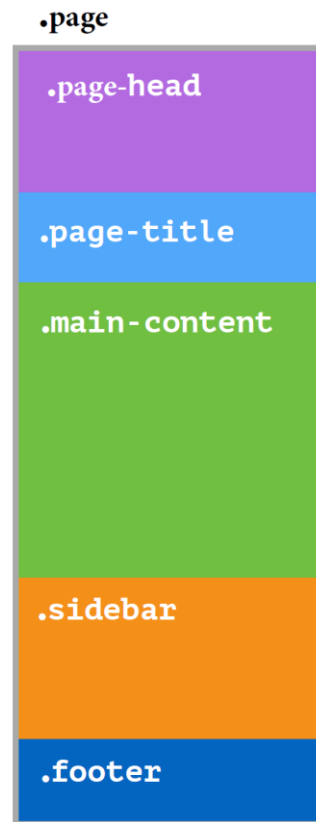
Responsive page layout using Media Queries

Use media queries to define layouts for different screen sizes

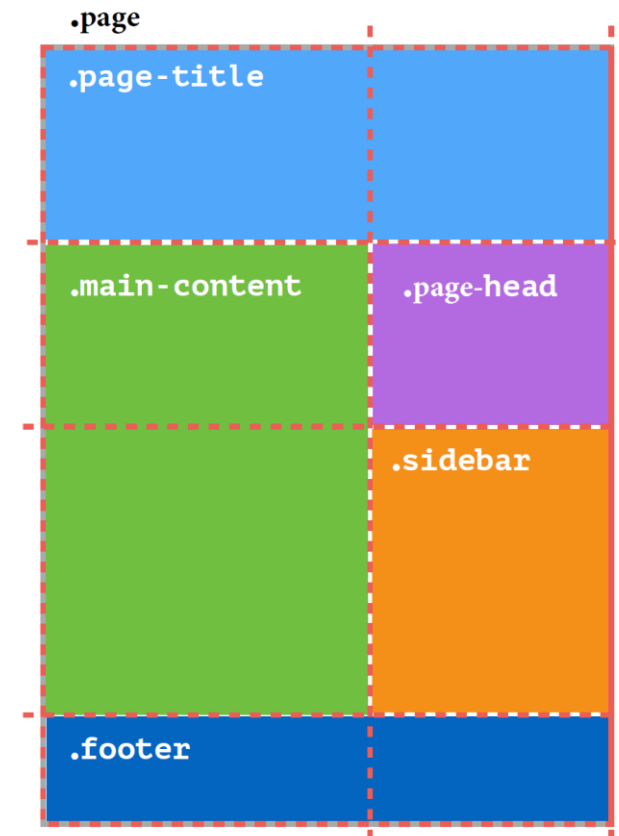
```
@media screen and (min-width: 700px) {  
  .page {  
    display: grid;  
    grid-template-columns: 2fr 1fr;  
    grid-template-areas: "title title"  
                        "main header"  
                        "main sidebar"  
                        "footer footer";  
  }  
}
```

- This example applies two-column layout once the screen width is above a specified **breakpoint**
- Media queries allow defining layouts for different screen sizes

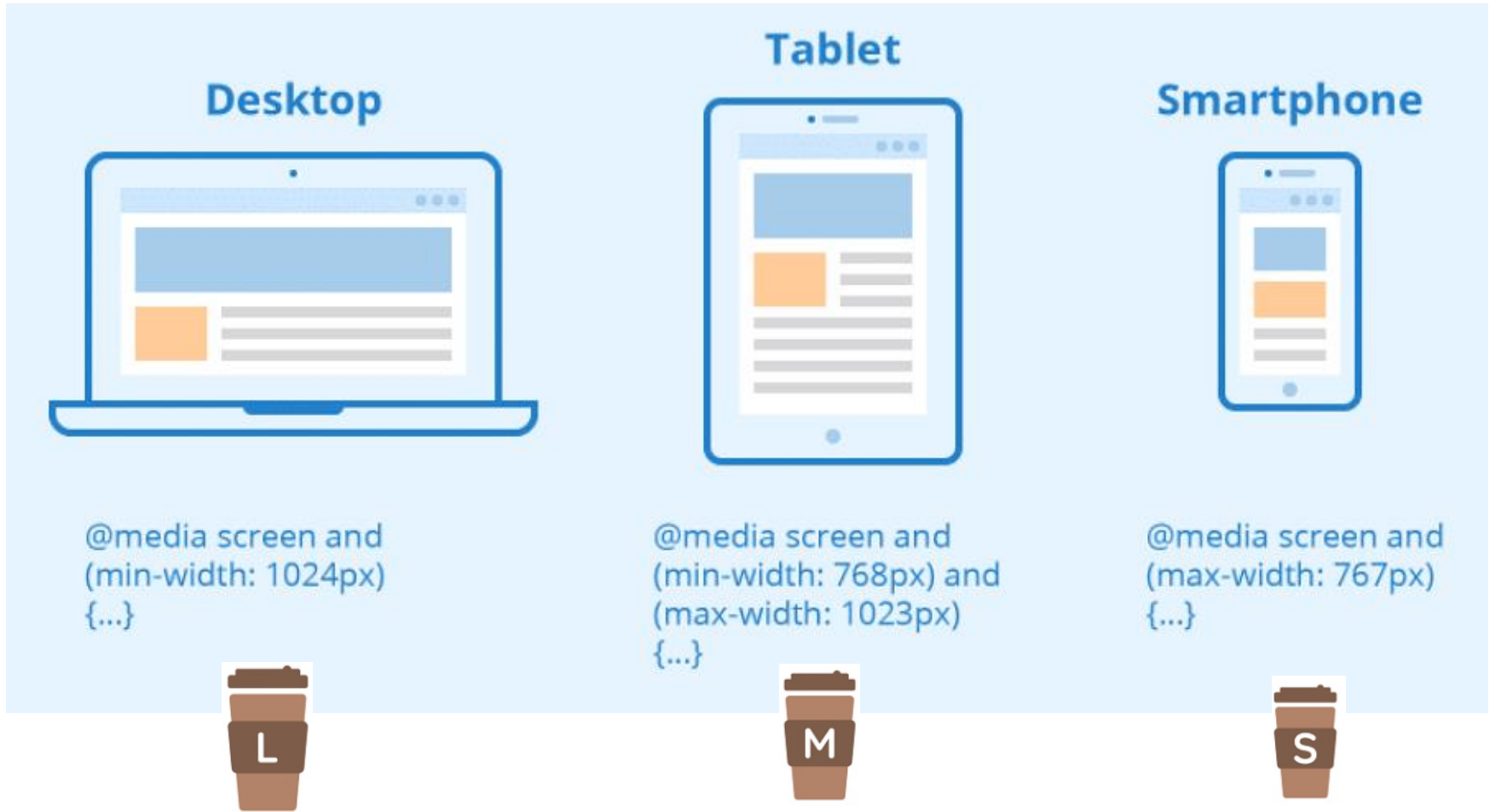
No grid



Two-column grid
(when page width $\geq 700px$)



Common breakpoints

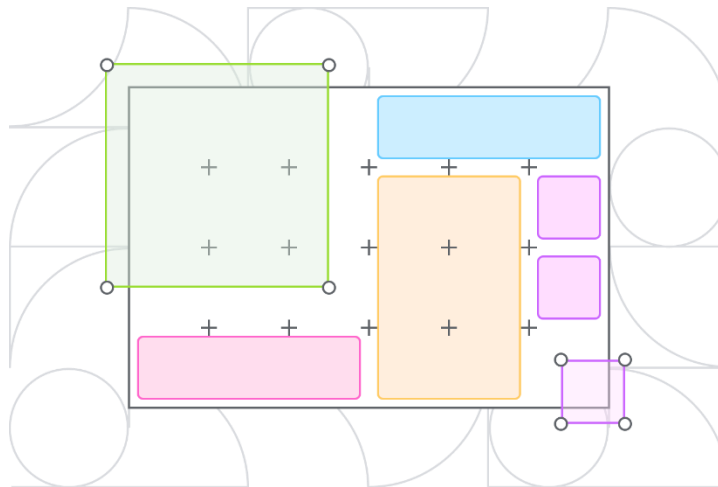


Source: <https://kinsta.com/blog/responsive-web-design/>

Common Layout Patterns

<https://web.dev/patterns/layout/>

Watch explanation in this [video](#)



Menu using a flexbox

- A website menu could be created using a `ul` element with `display: flex`

[Home](#) [About](#) [Contact us](#)

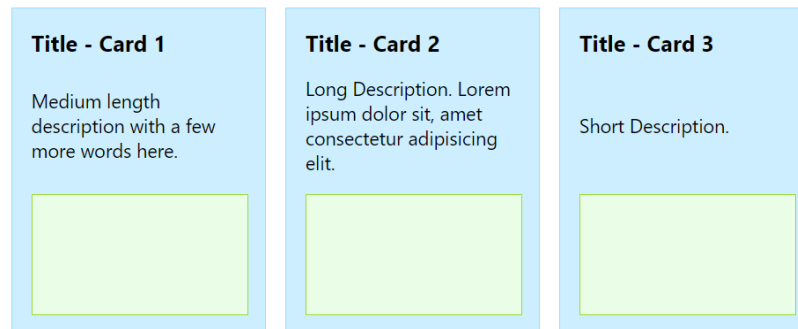
```
<nav>
  <ul>
    <li><a href="#">Home</a></li>
    <li><a href="#">About</a></li>
    <li><a href="#">Contact us</a></li>
  </ul>
</nav>
```

```
nav ul {
  width: 90%;
  display: flex;
  column-gap: 1rem;
  row-gap: 0.4rem;
  flex-wrap: wrap;
}
nav ul li {
  list-style: none;
}
```

Line-up card

justify-content: space-between

- Flexbox column card with **justify-content: space-between**
 - places the first and last child elements (e.g., title and image) at the edges of the flex container
 - the remaining space evenly distributed between the elements
 - e.g., the descriptive text in between gets placed with equal spacing to each edge



Aspect ratio Image Card

aspect-ratio: <width> / <height>

- Maintains the aspect ratio of an image in a card, while resizing the card.
- With the **aspect-ratio** property, as you resize the card, the image maintains the desired aspect ratio
 - e.g., maintains 16 x 9 aspect ratio as you resize the card

```
.card img {  
    aspect-ratio: 16 / 9;  
}
```

[Explanation & Demo](#)

Doha, Qatar



Doha is Qatar's largest city and commercial centre. It has a population of 2.4 million.

[Read more...](#)

Clamping card

`clamp(<min>, <actual>, <max>)`

- Sets an absolute min and max size, and an actual size for the card

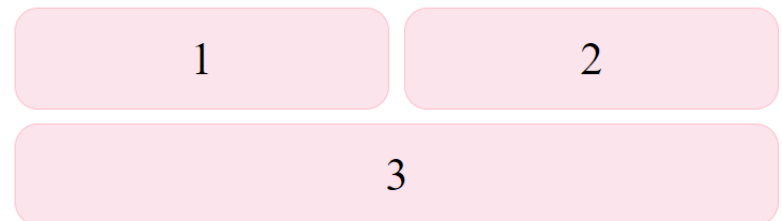
```
.card {  
  width: clamp(23ch, 40%, 46ch);  
}
```

- Min size is 23 characters, max size is 46ch, actual size is 40% of the parent width
 - Width of the card increases to the max size and decreases to its min size as the parent stretches and shrinks
 - Enables more legible layouts, as the text won't be too wide (above 46ch) or too narrow (below 23ch)

Deconstructed pancake

flex: <flex-grow> <flex-shrink> <base-width>

- Create a layout that stretches to fit the available space and wraps to the next line to maintain a minimum size (specified in base-width)
- On smaller screens, the boxes would stack nicely
 - set the value of <flex-grow> to 1 => flex items grow as you increase the screen size
 - set the value of <flex-shrink> to 1 => flex items shrink as you decrease the screen size
 - when needed boxes wrap to the next line to maintain the minimum base-width



[Explanation & Demo](#)

Pancake stack – Header-Main-Footer

`grid-template-rows: auto 1fr auto`

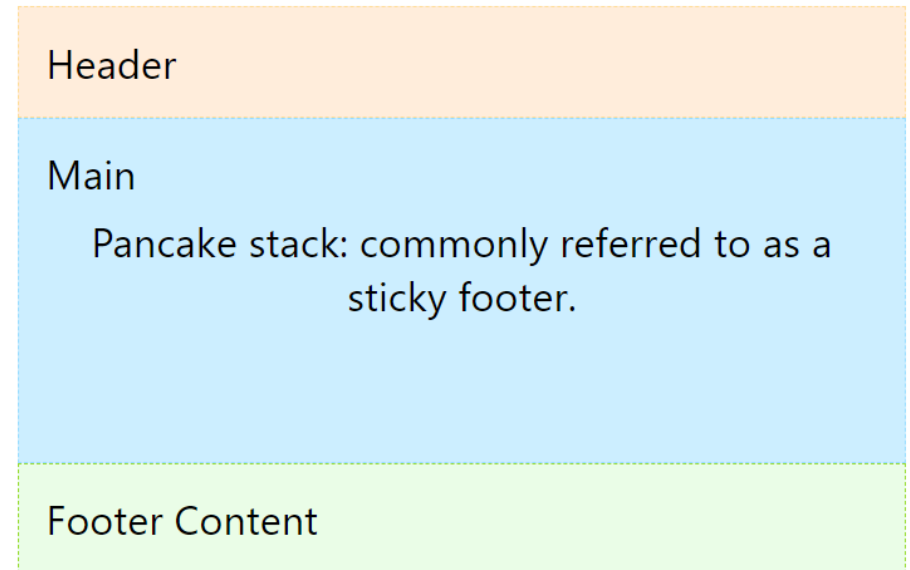
- Commonly referred to as a sticky footer

`grid-template-rows: auto 1fr auto`

- auto = auto-sized based on content

Header and footer are auto-sized based on their content

- main content area occupies the remaining space (1fr)



[Explanation & Demo](#)

Sidebar & Content

grid-template-columns: minmax(<min>, <max>) 1fr

- A layout where the sidebar is given a minimum and maximum safe area size, and the rest of the content fills the available space.

grid-template-columns:
minmax(100px, 20%) 1fr;

Min:
100px
/ Max:
20%

This main element takes the second grid position (1fr), meaning it takes up the rest of the remaining space.

- **minmax()** function is used to set the minimum sidebar size to 100px, but letting it stretch out to 20% on larger screens
 - the main content takes up the rest of the space (1fr)

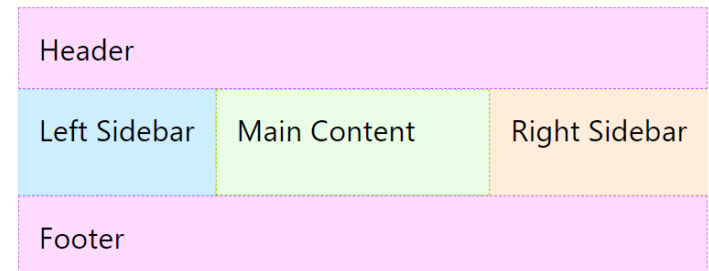
[Explanation & Demo](#)

Classic layout – Header-3 Columns-Footer

`grid-template: auto 1fr auto / auto 1fr auto`

- Classic layout with a header, footer, left sidebar, right sidebar, and main content area.
- `grid-template: auto 1fr auto / auto 1fr auto`
rows and columns templates separated by slash
 - auto = auto-sized based on content
header, footer and sidebars are auto-sized based on their content
 - main content area occupies the remaining space (1fr)
 - grid lines are used for placing the grid items

Explanation & Demo



RAM (Repeat, Auto, Minmax)

`grid-template-columns: repeat(auto-fit, minmax(<base>, 1fr))`

- A responsive layout with auto-created grid columns and automatically-placed children

`grid-template-columns: repeat(auto-fit, minmax(280px, 1fr));`

Browser!

- Use RAM (Repeat-Auto-fit-Minmax) to create **dynamic grid areas**
- I want you to **auto-create the grid columns** you decide how many you can fit using the auto-placement algorithm
- I want the columns to be minimum 280px and a maximum of **sharing the available space equality among the columns**



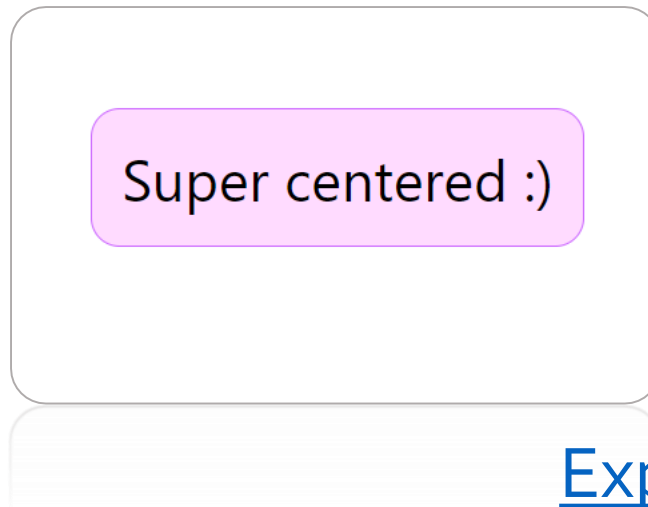
[See posted example](#)

[Explanation & Demo](#)

Super centered



place-items: center

- Use grid's **place-items: center** to center an element within its parent
 - place-items: center is a shorthand that sets both align-items and justify-items to center



[Explanation & Demo](#)

Summary

- Use Grid any time you work with *two-dimensional* layouts to divide the page into several sections having different size and position
- Use Flexbox for *one-dimensional* layout that offers space allocation between items + the ability to alter its items' width/height to best fill the available space
- Use Grid layout and Media Queries (when needed) for responsive design
- .. mastering CSS needs hands-on practice  ...

Resources

- Responsive Design Patterns
 - <https://web.dev/patterns/layout/>
 - <https://web.dev/learn/design/>
- Responsive Web Design Code Camp
 - <https://www.freecodecamp.org/learn/responsive-web-design/>
- Flexbox
 - <https://css-tricks.com/snippets/css/a-guide-to-flexbox/>
 - <https://marina-ferreira.github.io/tutorials/css/flexbox/>
- CSS Grid
 - <https://1linelayouts.glitch.me/>
 - https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout
 - <https://gridbyexample.com/learn/>
 - <https://css-tricks.com/snippets/css/complete-guide-grid/>