[Flex] ⇒ Its a flex box property
⇒ for one dimensional display
e.g. rows and columns

[Grid] ⇒ For 2-dimensional display
⇒ use of cells.

Block vs Inline Element (Display Property)

⇒ display is most imp property in CSS to controll layout.

⇒ Most common display values are:
display: block, inline, inline-block

⇒ every element have default display value.

[block]
- is a standard block-level element.
- It starts from new line
- Takes all the space
- other block elements are form, header, footer, section

[inline]
- is an inline element.
- An inline element only occupy space as much a it required.
- <a> is an other example of inline element

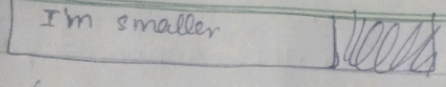
[inline-block]
⇒ does not force to start with new line.
⇒ However allow element to occupy all possible space.

⇒ [none] is another common display value. Its different from visibility.
⇒ As visibility: hidden; means that element is hidden on web page but still occupping some space
⇒ But display: none; is like the element never exists.

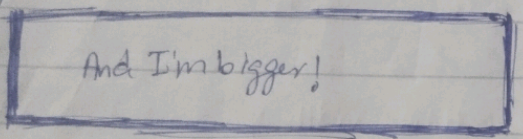
Box Model

⇒ While setting width of element, it can appear and occupy more space on screen and look bigger.
⇒ The element's border and padding stretching the element beyond specified width.

```
.simple{
    width: 50px;
    margin: 20px auto;
```




```
.fancy{
    width: 500px;
    margin: 20px auto;
    padding: 50 px;
    border-width: 10px;
}
```



For generations, we have to calculate/ do math, for displaying element according desired width.

Box-sizing

As due to drawbacks of actual box model, box-sizing property is used to display element with desired width

```
*{
    -webkit-box-sizing: border-box;
    -moz-box-sizing: border-box;
    box-sizing: border-box;
}
```

Positioning

⇒ Determines element's position within a doc.
⇒ top, left, right, bottom

There are five main values for positioning:

1. Static

⇒ defualt position of all element in html
⇒ no effected by top, ... bottom properties

2. Relative

⇒ position relavent to original position in normal document flow
⇒ still space is occupied of original location

3. Absolute:

⇒ position relavent to ~~most~~ nearest positioned element (rather than static)

⇒ If there is no ancestor, then intial element mostly <html> is reference point.

4. Fixed :

Elemend is positioned to viewport, remove from normal flow html doc.

5. Sticky:

Its hybrid of relevant & Fixed, Thes element has relevant position until it reached to specific scroll threshold then it behave fixed on screen.

Flexbox layout

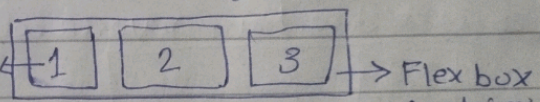
⇒ flexible layout

⇒ consiste of

- Flex Container - the parent
- Flex Items - items inside the container

Flexbox

⇒ A container become flexible by setting display property to flex



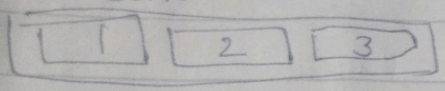
Flex-Direction

⇒ Display direction of flex item

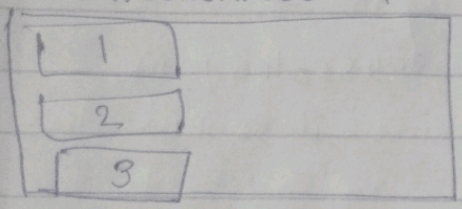
⇒ could be

- row
- column
- column-reverse
- row-reverse

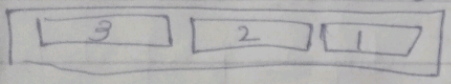
→ flex-direction: row



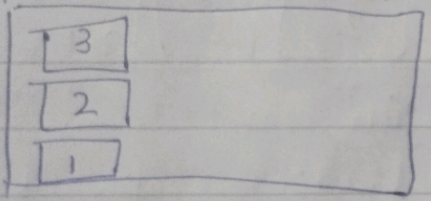
→ flex-direction: column



→ flex-direction: reverse row



→ flex-direction: reverse column

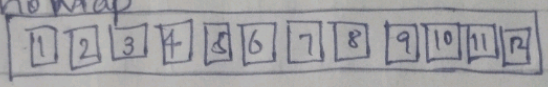


Flex-wrap

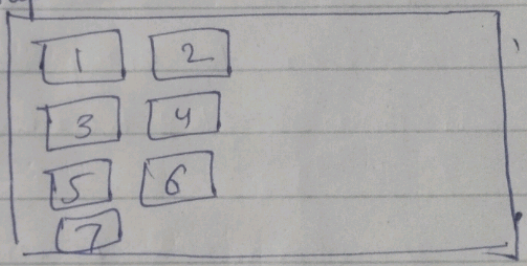
this property specify wether item should be wrap or not, if there is not enough space for them.

- nowrap
- wrap
- wrap-reverse

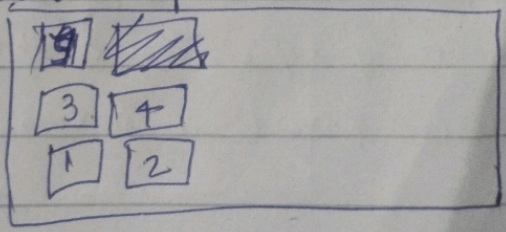
nowrap



wrap



reverse-wrap



Flex-flow Property

⇒ for setting both flex-direction & flex-wrap

flex-flow: row wrap;

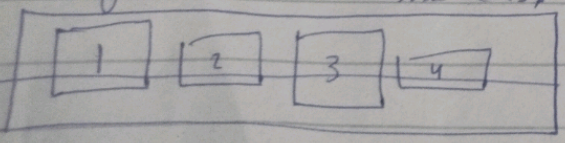
Justify Content

⇒ this property is used to align items when they don't occupy all available space on main axis (horizontally)

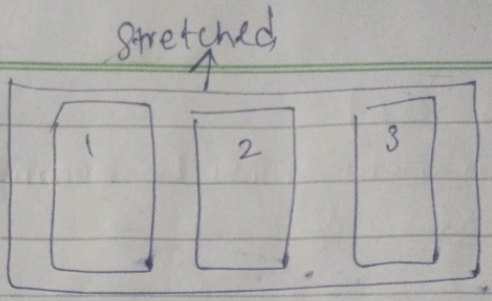
- center
- flex-start
- flex-end
- space-around
- space-between
- space-evenly

Align Item

⇒ Align item at cross_axis

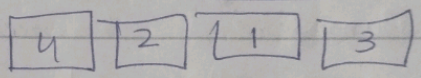


Baseline



Flex Items

Order Property




```
<div>
  <div style="order: 3">1</div>
  <div style="order: 2">2</div>
  <div style="order: 4">3</div>
```

flex-grow	flex-shrink	flex-basis
grow faster relative to other items	specify how item will shrink	intial length of item

Align-Self

⇒ align-item will be overrided by align-self

Grid Layout

⇒ grid-based layout system with rows and columns

⇒ for creating complex web layout

⇒ for responsive structure

⇒ element become grid container when its display property set to grid / inline-grid.

align-content: Vertically aligns whole grid inside container

align-items: Align content along column axis

align-self: for specific grid-item alignment

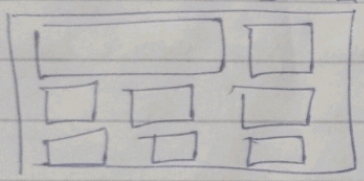
column gap: Specify gap b/w column.

gap: A property both for row & column gap

grid-area

Grid-Column Line

grid column-start: 1 ← grid is
grid column-end: 3 ← combo of



1 / span2 → and
↓ start

same for row

Grid-Template Columns
Grid-Template Rows

Define how many rows and columns you want and their sizes.

grid-template-columns: 200px 200px 200px
grid-template-rows: 100px 150px

grid-template-columns: 1fr, 2fr;
(2 column, second is double of first)

Grid Area:

- .header { grid-area: header; }
- .sidebar { grid-area: sidebar; }
- .content { grid-area: content; }
- .footer { grid-area: footer; }

grid-template-area:
"header header header"
"sidebar content content"
"footer footer footer"

Media Queries (@media)

⇒ Used to apply CSS rules only when screen matches certain conditions

```
@media(max-width: 600px){
    body{
        background-color: lightblue;
    }
}
```

Fluid Layouts

% relative to parent size

vh: viewport height (1vh = 1% of screen height)

vw: Viewport width (1vw = 1% of screen width)

minmax (150px , 200 px)

at least 150px but will expand

em: relative to parent's font-size

1.5em 1.5 × (selected font for ~~whole~~ doc) parent

rem: relative to root (html)

Mobile-First Approach

Design for mobile devices at first then for other devices