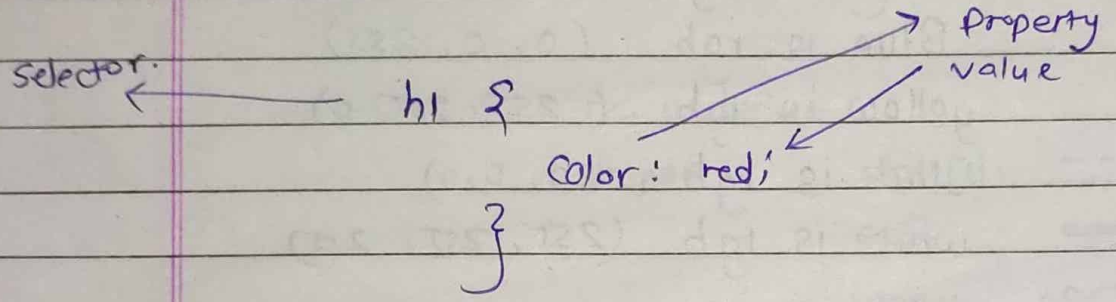
CSS

Page No.
Date

* Cascading Style Sheet
  : styling & designing
  - It is used to describle style of a document.
  - colour, font, size, bg, buttons, formats.
  - We have to link css in the head.

* format



* Types of Including styles
  ① Inline style / Inline css
  `<h1 style="color: red">New</h1>`
  : It is directly added in HTML heading / in element.

  ② Internal style : In the head tag. with style tag.

  ③ External style : In css file (seperate document)

* Color properties
  ① foreground color : The upper color of text i.e. text color is always foreground color.
  Whenever we use color property it will be applied to foreground color.
  ② background color : The bg color used to set backside color of any background text or any entity.

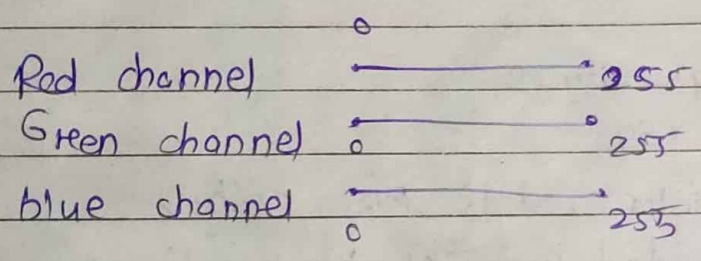
  → background-color : red; (background : red;)
  → color: red;

* Color systems : ① Browser only can recognise 140 - 150 colors.
  • which colors we don't know we can used color codes for them.

② RGB systems :

Red Green Blue

eg: Red is rgb (255, 0, 0)
Green is rgb (0, 255, 0)
Blue is rgb (0, 0, 255)
yellow is rgb (255, 255, 0)
Black is rgb (0, 0, 0)
white is rgb (255, 255, 255)



• we also can use color picker & directly use that generated code.

③ Hex code / Hex / Hexadecimal codes.
: The color code of 6 digits with using hashtag.
: Similar to RGB ⇒ # FF FF FF
R G B
: 16 characters are used : 16 Letters
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

White # ffffff ⇒ #fff
Black: # 000000 ⇒ #000
green # 00ff00 ⇒ #0f0

* font weight : lite or bold font [from 100-900]
default or normal is 400.
Normal = 400, bold = 700

* text-decoration :
text decoration: 1] underline abc
2] overline abc
3] line-through abc

: red underline
: shapes : dotted ---
solid ▬▬
wavy ≈≈
double ═══

: text-decoration : none
⇒ removes each effect created

* line-height
① line-height : normal
② line-height : 2.5 ⇒ 2.5 × normal.

* letter spacing
① letter-spacing : normal
② letter-spacing : 10 px (px - pixels)

* font sizes ~~&~~ units in CSS.

Absolute units

Those units whose values are fixed means don't change with any situations.

- Px
- pt
- pc
- mm
- cm
- in

Relative units.

in relation with other values

- %
- em
- rem
- ch
- vh
- vw + many

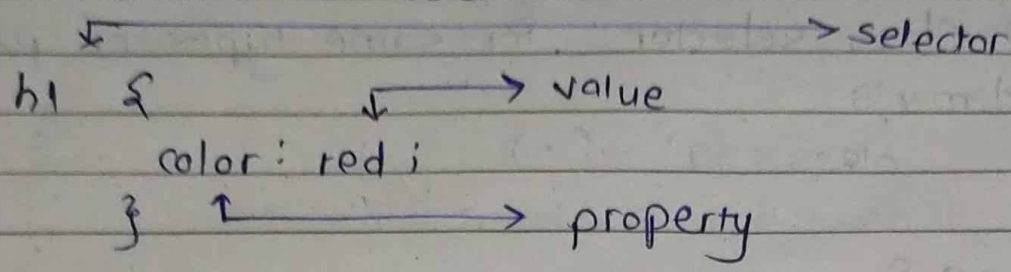
: 1 inch = 96 px
: 1 inch = 72 pt
: 1 inch = 12 pt

rem : related to parent
em : current size related new size

* Pixels (px)
: mostly used
: not suitable for responsive websites

* font-family
font-family : arial;
font-family :

* Basic format of CSS



Types of selectors :

① Universal selector : To select each & every element . (Asterisk symbol = *)

```
* {
    font-family : Arial ;
  }
```

② Id selector : To apply on a specific element
id = "color". (Hashtag or hex = #)

```
#color
  { . . .
  }
```

③ Element selector : To apply on multiple elements of same type. Also can be applied by using comma.

eg:

```
h1
  {
    color : black ;
  }
```

} for all h1 element

```
h1 , h2 , h3 .
  {
    color : black ;
  }
```

} for all h1, h2, h3 elements.

④ Class selector : for same types of particular elements

class = 'a' (.)

```
.a {
    color : red;
}
```

⑤ Descendant selector :

grand parents → parents → children

⇒ childrens are descendants of parents & gp.

⇒ parents are descendants of gp.

```
<div> <p> </p> </div>
```

p is descendant of div.

⇒ ① div p

```
{
    color : black;
}
```

⇒ we also can give with the class selector.

class = "new"

```
.new p
{
    color : red;
}
```

⇒ more levels than 2.

```
nav ul li {
    color : red;
}
```

⇒ nav ul li a { -- }

or

nav a { -- }

Page No.:
Date. / /

• Combinator

① Adjacent Sibling Combinator :
Multiple selector's combination.

```
<p></p>
<h3></h3>
<h3></h3>
<div><h3></h3><div>
```

Here if we only want to apply css on that underlined h3. then sibling or adjacent combinator is used.

sibling combinator.

```
p + h3 {
        property : value;
       }
```

```
<p class="new"></p>
<h3></h3>
<div class "new><p></p>
<h3></h3>
<p></p>
```

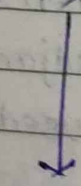
→ with using class

```
.new+h3 {
         color:red;
        }
```

② Child combinator
: direct descendant

⇒ only for paragraph ie. the element which comes immediately it only applied to that element. not others.
ie. only for direct child

eg: div > P {
color : red;
}

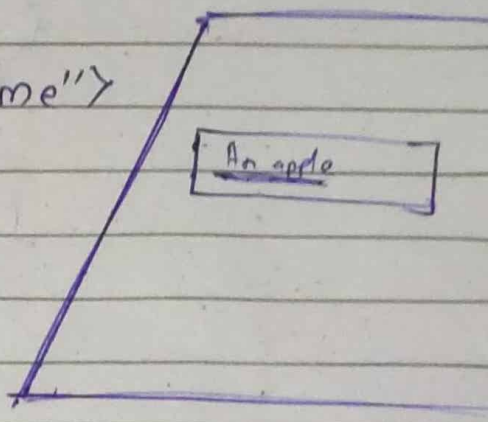


child combinator

⑥ Attribute selector.
: Any element having presence of attribute.

eg①
<input type="text" placeholder="name">

⇒ input[type="text"] {
color: red;
}



eg: ②
<P> My name is Anu </P>

⇒ div[id="new"] P {
color: red;
}

* Pseudo class.
: applied on a special state of selected elements.

① hover
button: hover
{
color: red;
}

② active [after click]
button: active
{
color: pink;
}

③ checked [used for radio buttons]

```
input[type="radio"] : checked + label
    {
      font-weight : bold;
      color : red;
    }
```

④ nth-of-type

```
div : nth-of-type (2)
        {
          color : red;
        }
```

for even = 2n
for odd = 3n

```
P: nth-of-type (3)
        {
          color : black;
        }
```

* Pseudo Elements
: style to a specific part of any element.

:: first-letter
:: first-line
:: selection (the selected area style changes)
(the area which will be selected)

eg:- [1] h1 :: first-letter
{
color: red;
}

[2] p:: first-line
{
color-purple;
}

③ P:: selection
{
background-color : red;
}

CSS - Cascading Stylesheet.

* Cascading
  : Cascade algorithm's job to declare / determine CSS-correct values by selecting CSS declarations in an order.

eg : If there are two values declared for h2 then the cascading property says that the value which is applied lastly it gets applied to the final value. (element value dedared at last)

```
h2 {
    color: red;
    }
h2 {
    color; blue;}
```

Blue color will be applied.

- This will only be applicable for same element.

* Specificity. [selector specificity]
  : Algorithm that calculates weight that is applied to CSS.

100th	10, 10, 10th	1●, 1●th
id	class, attribute & pseudo-class	element & pseudo-element
1	1+1+1	1+1

⊛ Priority :
id > class > element

eg: ① h2 {
background-color: red;
}

specificity ⇒ 0 (id) | 0 (class, attri & pesudo-class) | 1 (element & pesudo element) } (0,0,1)

② .myClass: hover {
color: red;
}

specificity ⇒ 0 (id) | 10+10 (class, attri & pseudo class) | 0 (ele & pseudo element) } (0,20,0)

③ # myid {
color: red
}

specificity ⇒ 100 (id) | 0 (class, attri & pseudo class) | 0 (ele & pseudo element) } (1,0,0)

* Inline specificity : It has highest priority.

Inline > Id > class attribute & pseudo classes > element & pseudo element

* !Important : This is most specific thing in doc.
: After this the value is fixed to it & other values will not effect to it.

```
h2 {
    color: red !important;
}
```

* Inheritance in CSS

Some some qualities which inherits to next generation this is inheritance.

```
<div> <p>
        <ul>
            <li> a </li>
            <li> b </li>
        </ul>
    <p>
</div>
```

```
div
{
  color : red;
}
```

then color of each element will be changed in div (presented every element) this is due to inheritance property.

: Any element which doesn't having any property it tries to take properties from parents (if they don't have property) then grandparents & so on.

: All elements doesn't follow this rule

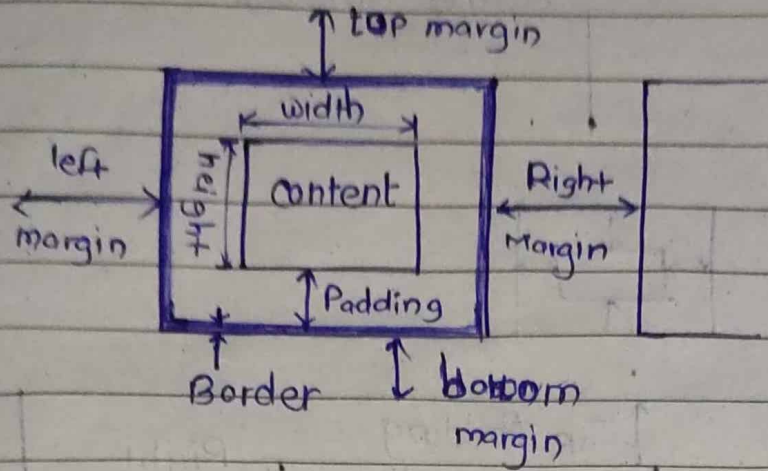
eg: Buttons, inputs

(for allowing them we have to write that property name & allow inheritance)

ie.
```
input {
    color: inherit;
}
Button {
    color: inherit;
}
```

: Some properties does not inherits eg: width, height, border

* Box Model in CSS.



: If there is no content this is not applicable.

① height : The height of content area.

```
p {
    height : 100 px;
}
```

② width : width of content area.

```
p {
    width : 100 px;
}
```

③ Border : (i) Border-width

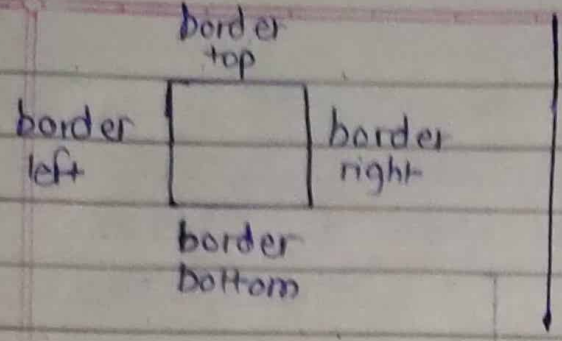
: Border is set to outerlining of element not to content.

(ii) Border-style : ---- dash , ... dotted , —— Solid . = double

(iii) Border-color : any

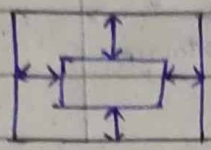
[(Border : width - style - color) border shorthand)

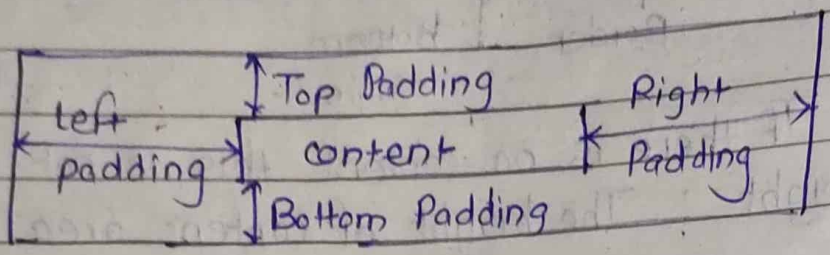
border : 2px solid black;



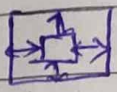
Border radius = 100% ⇒ ○
Border radius = 10 px □

* Padding





Shorthands : • [1] for four sides:



padding : 40 px;

[2] For two two sides

padding: 20px 20px;
↓ top & bottom
↳ left & right

[3] top | left & right | bottom

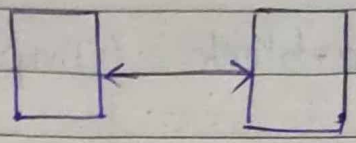
padding : 10 px 20px 30 px;

• ④ top | right | bottom | Left (clockwise)

padding : 10px 20px 30px 40px;

⑤ or differently padding-left, padding-right, padding-top, padding-bottom

Page No.:
Date. / /

* Marging  (bet border of two different elements)

• Shorthands : same as padding.

* Display : If we want to change the property of any element of behaving it as inline or block then this is used.

eg: span {
display : block;
}

⇒ Here the span which is inline will behave as block.

• Span element : It doesn't applies top & bottom margin & ~~padding~~.

BLOCK	INLINE
① Starts on new line	① Continues to that line.
② Takes full width of browser window.	② Takes required space.
③ Will apply widhth & height	③ Don't accept width & height
④ Works on all sides	④ Works horizontally not vertically. (for margin)
⑤ Margins work on all sides	⑤ Padding works on all sides but may overlap for top & bottom. on other elements.

Date: / /

Display : Inline-block (Imp).

Elements behave as inline but all styles will be applied of block.

Units

Absolute	Relative
- px (pixels)	- %
pt	- em
pc	- rem
cm	- ch
mm	vh
in	vw + many

Imp Relative elements (mostly used)

① % (percentage):

→ Used for setting the size of element with relative to the parent.

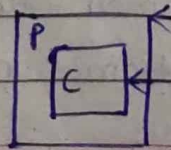
: Child's size will be set according to relation with parent.

: size can be changed according with parent size

② Em

→ (i) In case of typographical properties like font-s it relates to the parent font size.

eg:-



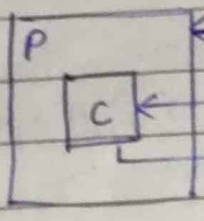
Parent's font size = 10px

child's font size = 2em = 2×10
= 20px

or 3em = 3×10px = 30px

(ii) In case of other properties like width or other it relates to font size of element itself.

eg:-



Parent font size = 5px

child's font size = 2em = 10px

padding = 2em = 2x10px = 20px

- Drawback of fm : As if we go from parent to child then in case of nesting of multiple elements the size of elements can get double by it's parent any case of 2em or greater. And in case of 0.5em it gets smaller.

eg :- ① section div1 div2 div3
P_1 P_2 P_3 C

if section { font size : 10px ; }
div {
fs : 2em; }

section
div1
div2
div3

② in case of 0.5em } section
div1
div2
div3

That's why we can use root em.

③ Rem (root em)

: Root em means this property relates to root element.

eg: ① section div1 div2 div3
P_1 P_2 P_3 C
(G) (G)

if section size is 10px then if
div = 2em then all div1 = div2 = div3 = 2em
= 2x 10px = 20px

* CSS - Next steps

* Alpha Channel (opacity)
(Range 0-1)
(i) rgba (255, 255, 255, 1) ⇒ full opacity white

(ii) hex code :- #000000a1 ;

: Alpha channel is only used for color.

* Opacity (Range from 0 to 1)
: Sets for any element

```
div {
(i) opacity: 0.5;
    }
```

* CSS transition
: transform of one state to another

transition : 1s or [1 minisec ⇒ 1ms]
⇒ transition-duration : 1s
this is same (transition is shorthand)

* Transition shorthand

property name | duration | timing-function | delay

eg:

transition: background-color 2s ease-in-out 0s ;

(i) Duration : this is for how much time should the activity take.

(ii) transition delay : this is for the timing of delay after that the activity should start.
→ if delay is 1s then that trasition activity will start after 1sec.

(iii) ease-in-out, ease-in, ease-out, ease, linear, step, step-end.

CSS transform.
- Rotate, scale, skew, translate an element.

(i) Rotate : Rotate is done by angle rotation we mostly use degree rotation. [other radians, turns]

eg:
```
transform: rotate (45deg);
```

This transition occurs default in clockwise direction.

(ii) Scale: scale is like zoom in or zoom out. transform can done in both 2D & 3D.

eg:
```
transform: scale(2);          [both(x & y)]
transform: scale(0.5, 0.5)    [x & y]
transform: scale (2,2,3)      [[x,y,z] 3D]
transform: scalex (0.1)
```

(iii) translate : Change the position from one plate to another.

transform : translate (50px , 50px)
transform : translateX (50px)
transform : translateY (-50px)

(iv) Skew : tilt the element (in the form of deg)

transform : skew (50deg)

No. of in one :→
transform : translateX (50px) skew (40deg)

* Box Shadow
: It adds shadow effect around any element's frome.

box-shadow : x-offset yoffset blurradius color;

box-shadow: 2px 2px 10px black;

: A 3D like effect of image will be created with box shadow.

* Bg image : Apply image to background.

background-image : url (—)
[: Here in url you have to write name of img / soure of img]

Page No.
Date / /

To make size adjustment of img we use bg size property.

① background-size : cover;

(i) Cover : Crops & adjusts img
(ii) auto : increase pixels
(iii) contain : fits img & in remaining space it adds once again image.
(In this image will be added multiple times)

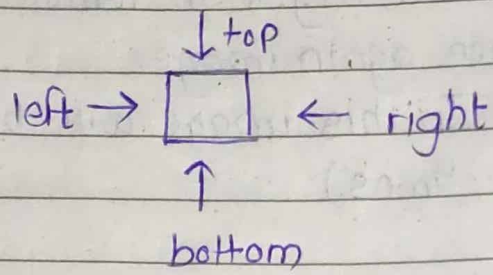
* Card Hover effect

Page No.
Date: / /

* Position

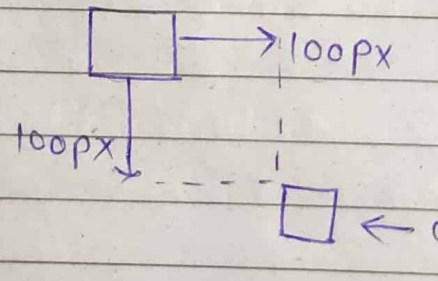
How an element is positioned in a document.

(i) Static
(ii) Relative
(iii) absolute
(iv) fixed



① Static : (Default value) top, right, bottom, left & z-index properties have no effect on it.
(element does not moves with static position it remains in original position)

② Relative : Offset is relative to itself.



③ Absolute : It is removed from normal doc flow & position is changed/ can be given according to it's nearest parent. If there is no parent the relative to body.
It does not creates different space of the element.

④ fixed : Element is removed from normal doc. flow, no space is created for that.

• It is positioned to relative to initial containing block?

(: sticky)

* FLEXBOX

: flexible box Layout (Responsive)

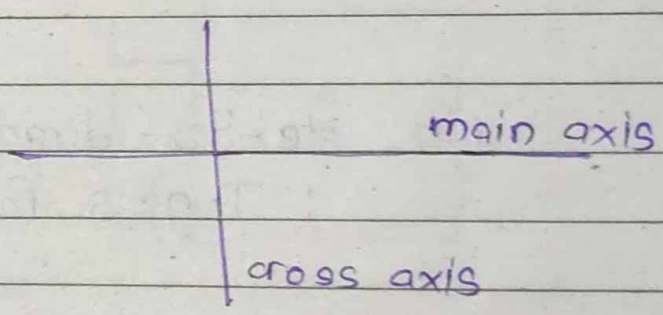
: 1D layout method to arrange items in rows or columns.

⇒ □□□□□ → rows

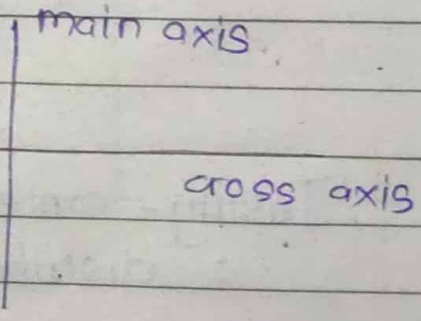
□
□
□
□ → columns

• The Flex Model

If direction Row ⇒



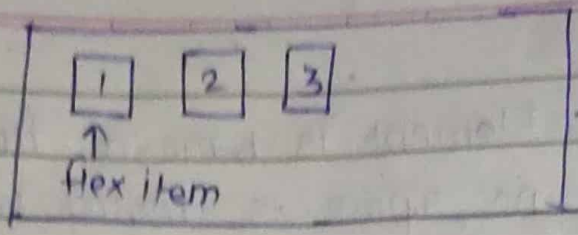
In direction column ⇒



: axis means simply direction

Date / /

eg:




```
flex container {
    display : flex;
        }
```

* Flexbox Direction
  : In which manner items will be placed ie flex elements will be placed into the flex container.

(i) flexbox-direction : row
  : It goes from left to right [1] [2] [3]

(ii) flexbox-direction : column
  : It goes from top to bottom
  [1]
  [2]
  [3]

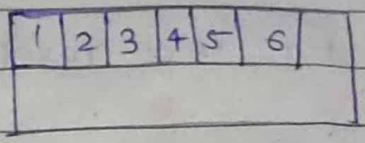
(iii) flex-box-direction: row-reverse
  : It goes from right to left [3] [2] [1]

(is) flexbox-direction: column-reverse
  : It goes from bottom to top
  [3]
  [2]
  [1]

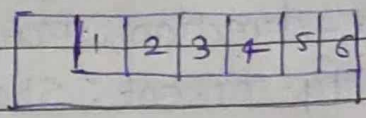
* Justify-content
  : distribut space betn content items along main axis.
  : justify means arranging on the axis.

- for Row :→

(i) justify content: start



(ii) justify content: end



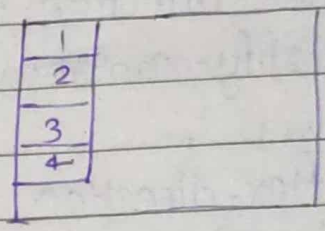
(iii) justify-content : flex-start (similar to (i))

(iv) justify-content : flex-end (similar to (ii))

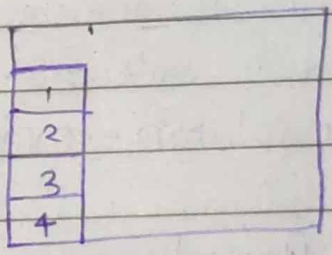
- for column :→

(i) justify content: start

justify content: flex-start

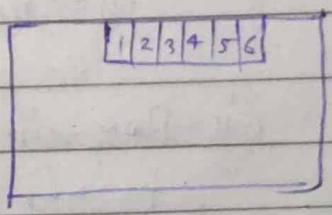


(ii) justify content : end
justify content : flex-end

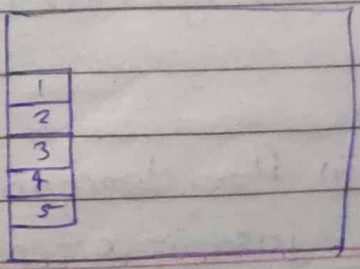


- Middle for both

(i) for row : (flex-direction: row)
justify content: center

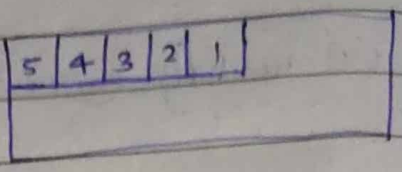


(ii) flex-direction: column;
justify-content: center

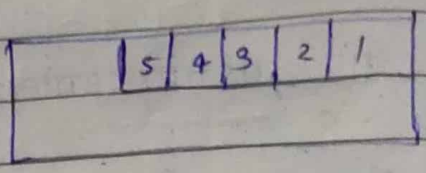


- for row-reverse :→

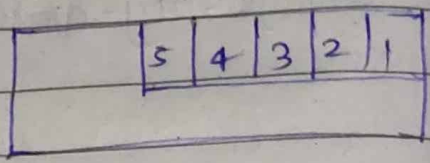
(i) flex-direction : row-reverse;
justify-content : start;



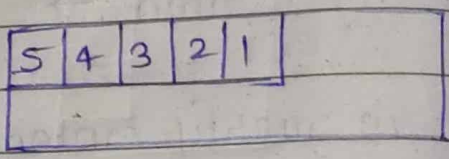
(ii) flex-direction: row-reverse;
justify-content : end;



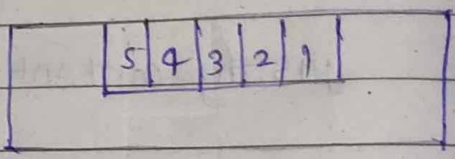
(iii) flex-direction: row-reverse;
justify-content: flex-start;



(iv) flex-direction : row-reverse;
justify-content: flex-end;

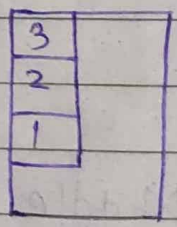


(v) flex-direction: row-reverse;
justify-content : center;

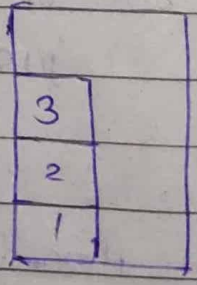


- for column-reverse :→

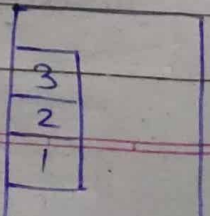
(i) flex-direction : column-reverse;
justify-content : start;
justify-content : flex-end;



(ii) flex-direction - column-reverse;
justify-content : end;
justify-content : flex-start;

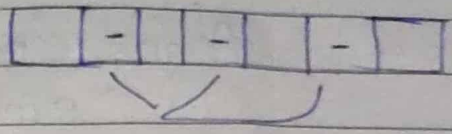


(iii) flex-direction : column-reverse;
justify-content : center;



Page No.:
Date: / /

- justify-content : space-between



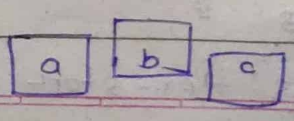
- justify-content : space-around
  [half on last sides than middle]
- justify-content : space-evenly
  [all even spaces]
- justify-con

* Flex Wrap
  : If the elements overflow the flexwrap can be used.
  (i) no-wrap : it doesn't goes to next line it wraps content inthat line by decreasing dimensions default

  (ii) wrap : by assuming dimensions it will go to nex line & if there it doesn't fits then to ~~next line~~ nex line & so on.

  (iii) wrap-reverse : cross axis in rev direction.

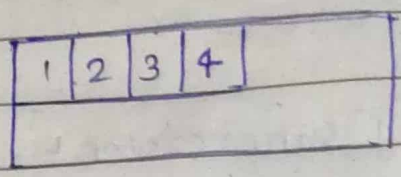
* Align-items :- Align items by cross axis
  (i) flex-start :- This aligns to the flex starting.
  (ii) Align-items : flex-end : to the flex ending.
  (iii) Align-items : center : in middle
  (iv) align-items : baseline : [this works by the ~~ating~~ aligning of content in a same line] [rather than size]



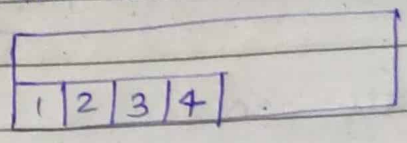
Page No.:
Date: / /

* Align-content
: similar to justify content but it works on cross-axis.

- For Row
  (i) align-content: start
  align-content: flex-start.

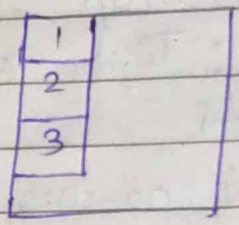


  (ii) align-content: end
  align-content: flex-end

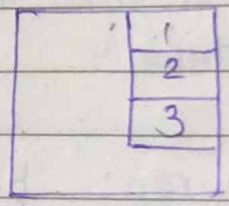


- for column

(i) align-content: start
align-content: flex-start

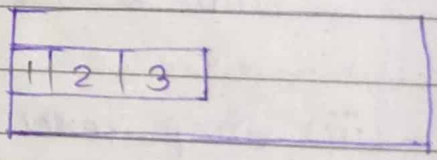


(ii) align-content: end
align-content: flex-end

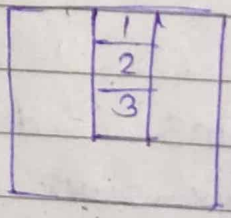


- center.

(i) for row
align-content: center;

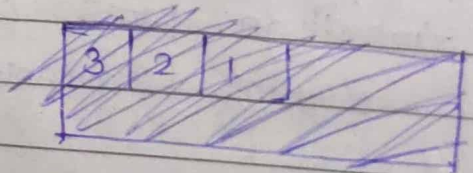


(ii) for column
align-content: center;

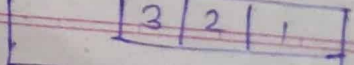


- For row-reverse.

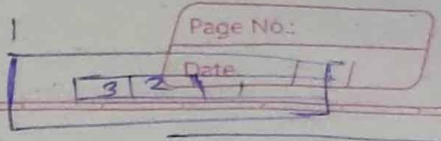
(i) ~~align-content: start end~~
~~align-items~~:



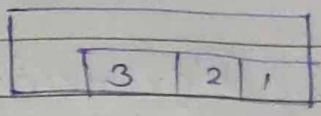
(ii) align-~~items~~ content: flex-start



(ii) align-content: flex-center

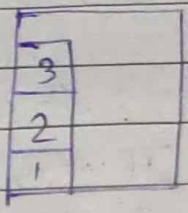


(iii) align-content : flex-end

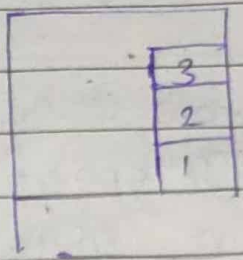


- column-reverse

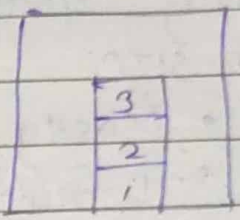
(i) align-items : flex-start



(ii) align-items: flex-end



(iii) align-items : flex-middle



* align-content: space-between same as justify but on cross axis.

- align-content: space-around

align-content: space-evenly

align-content: baseline

Align-self (higher-priority) : along cross axis for individual item.

align-self : flex-start
flex-end
ceter
baseline [baseline to other elements]

Page No:
Date: / /

* Flex-sizing

- Flex-basis : this sets initial main size of flex item.
  (for individual item).
  this happens along flex-direction.
  If this for row then width will change &
  if this is for column then its height will change

- Flex-grow :
  It specifies howmuch of the flex containers remaining space should be assigned to flex items main size.
  - Mostly unitless.
  - Default value is 1
  - Individual element

eg: flex-grow : 1

The remaining all space will be occupied by that element.

- If More than one element has given flex-grow value than on the basis of their proportion they will auto-distribute in the space themselves.

- flex-shrink
  : If more space is needed then flex-shrink to an element can be applied.
  : default is 1.
  : individual item

eg:- flex-shrink : 2

- flex shorthand

(i) flex-grow / flex-shrink / flex-basis
flex : 2 2 100px;

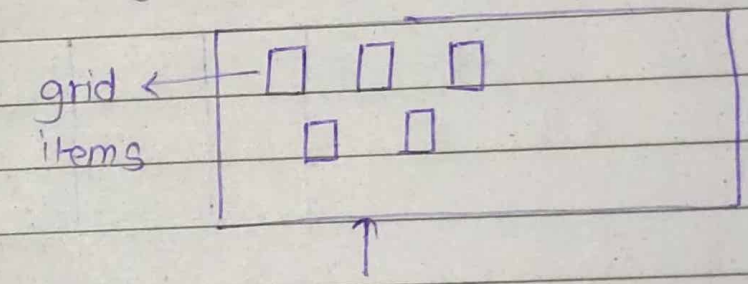
(ii) flex grow / flex basis
flex : 2 100px;

(iii) flex : 2 [flex-grow(unitless)]

(iv) flex : 100px [flex-basis(unit)]

CSS Grid
: setting container grid will make all children grid items

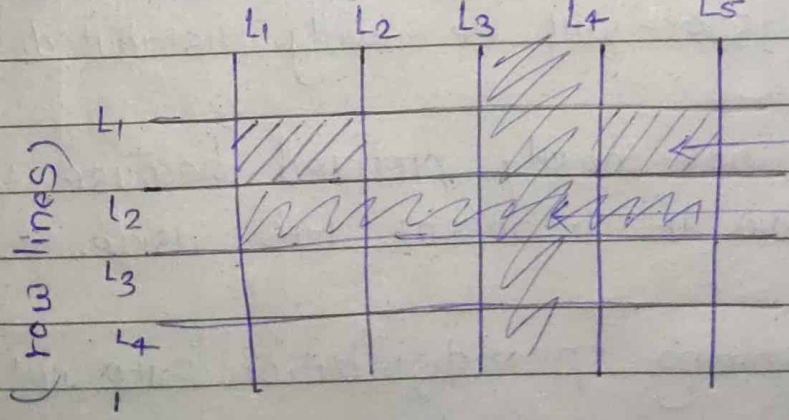
container {
display: grid;
}



grid container

: only parent children will get grid items as similar to flex.

* Grid model (column lines)



• Grid gets created in 2D model

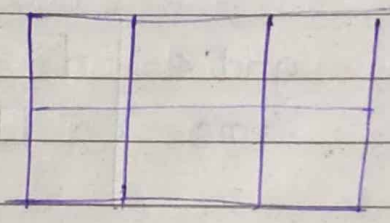
Date. / /

* Grid template
This defines lines & track sizing
: [track +1 = lines]

: with grid template we can make a path for placing items by giving sizes of column track & row track.

eg:
grid-template-column: 50px 50px 50px;
grid-template-row: 50px 50px;

(grids/cell) = n x m [n=columns, m=rows]
= 3 x 2 = 6 grid cells



eg: If we use auto one time then one column/row track will be created & equally dismbuted between the container space.

No. of auto s in section will creat that times tracks & space will be equally distributed.

: Auto is not mostly prefered because the content will overflow if it doesn't fits inside.

: for remaining space distribution auto will be used.

: If item's height or width is not specified then the item will directly go as 1st cell, 2nd cell & so on

* Repeat (count, 1fr)
  : 1 fraction of 100% width → available space

Means eg:-
grid-template-rows: repeat (3, 1fr)
[depends upon container height & width]
: This will create three equal rows & the remaining space will distribute bet'n remaining items.
[height for rows & width for columns will make]

. Grid Layout is preferred for 2D use mostly preferrable.

grid-template-rows : 1fr 1fr 1fr ;

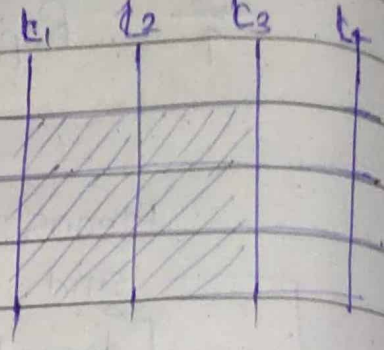
* grid-gap

row-gap : 10px ;
column-gap : 10px ;
grid-gap : 10px 20px ; [grid-gap : rowgap colgap ;

grid-gap : 10px [for both]

Page No.:
Date: / /

* Grid-columns
  : Used for placing item.
  : item's starting & ending position inside column.



grid-column-start : line-number
grid-column-end : line number

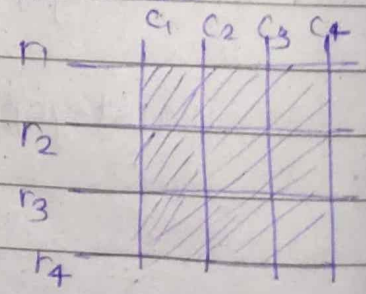
eg: grid-column-start : 1
grid-column-end : 3

- grid-column : start col / end col
  ⇒ grid-column : 1 / 3

- grid-column : start col / span number
  (this span no. means upto which column you want mean if we want to go on upto 4th column then span no. from 1 will be 3)

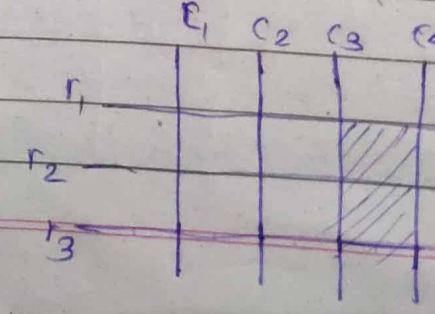
  ⇒ grid-colum : 1 / span 2

* Grid-Rows
  : same as column but will work for rows.



grid-row-start : 1
grid-row-end : 4
grid-row : 1/4;
grid-row : 1/span 3;

grid-row : span 2
(for 3rd cell (c_3-c_4) r_1-r_2)



* z-index
: overlapping elements
: It is used to stack level change.
: If we want any element more visible while overlapping then by using z-index we can make it

: default is 0. (auto = 0)

: If we want to set z-index the position of element should be relative, or other
[[it shound't be static or default]]

: positive means outer/upper level
: negative means lower/down level

* Media Queries
: Help to create responsive website.

@media
[There are environments like height, width, another orientation : (i) landscape (ii) portrait]

media-features : width (of viewport)

If we want any change in website related to width on mobile (ie. mobile's width is less than computer & then when we use site on mobile then no. of features will not visible property so to change them) media-queires are used.

device

```
@media (width: 400px)
  {
    div {
      color: red;
    }
```

⇒ : means when the size of mobile ie. width will go to 400px the color of text will be changed to red.

: we use mostly: max-width,
min-width.

eg:
```
@media (min-width: 400px)
  {
    div {
      color: red;
    }
  }
```

this means minimum width of device should be 400 & greater ther text red color will be apped.

```
@media (max-width: 400px)
  {
    div {
      color: red;
    }
  }
```

This means max device width should be 400 & less than that so text-red color will be applied.

other syntax

⇒
```
@media (min-width: 400px) and (max-width: 500px)
  {
    div {
      color: red;
    }
  }
```

* Basic design principles
(for better looking websites)

Color Theory

[1] Monochromatic color : Similar colors but differe shades (subtle look).

[2] Complementary : for impactful color combination (opposite colors in color wheel)

[3] Triadic : High contrast combination (three triangle colors) (in color wheel)

- Color combinatic site [canva.com color wheel]

* Typography
: Style & appearance [color, fonts
- google fonts (https://fonts.google.com) 1513
- Icons (i) fontawesome (fontawesome) [link : cdnjs]
(ii) Google icons [[static.iconfont] - link]

BOOTSTRAP.

* Original toolkit (same as template) to use in your site.
  - prebuild components are available

Defn:- Bootstrap is a powerful, feature-packed frontend toolkit. To build anything from prototype to production - in minutes.

- To add (either download bootstrap) or add cdn link.

- CDN (content delivery network)

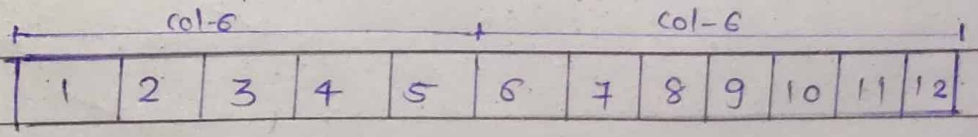
* Container Layout
  Similar as any box. The size-responsiveness is added automatically.
  - container, container-md, container-fluid

* Button, Button group
* Badges: Badges are simply the notification msgs / emails / some new addition which highlighted by badge.

eg: G (21) ← Badge. Heading [new]

* Alert: (Any pop-up window)
* Navbar

* Grid System



- If two columns: then equally space divided col-6 col-6
- If three: col-4 col-4 col-4