

CSS Basics - I



CSS :- Cascading Style Sheet.

- To Style the web page.
- we add the properties
- all about appearance (visual)

Selectors in CSS :- is a way using which we can select an element(s).

1) Simple Selector.

- + Element Selector
- + class Selector
- + ID Selector

2) Pseudo-class Selector.

3) Multiple Selector.

Selector

Color : #02b3e4 ;

{ property value }

① Element Selector

↳ also known as type selector
and Tag Selector.

CSS can select HTML elements by using an element's tag name. A tag name is the word (or character) b/w HTML angle brackets.

→ It changes every Tag properties present in your Document.

Syntax :-

P S

Color: red.

7

② Class Selector

Limited selecting elements by tag name.

- grouping the tags for styling.
- giving class name.
- To select an HTML element by its class using CSS, a period (.) must be prepended to the class's name.
- Classes will NOT be unique, always use for grouping.

Example

```
<p class = "green"> ABCD </p>
```

```
<p class = "green"> ABCD </p>
```

```
<p class = "blue"> ABCD </p>
```

```
<Style>
```

```
• green {
```

```
color: green;
```

```
}
```

```
• blue {
```

```
color: blue;
```

```
}
```

```
</Style>
```

③ ID Selector

↳ Unique Selector.

→ The delineation is made by using (#) to represent an id.

→ Individual Styling.

```
<button id="button">Submit </button>
```

```
#button {
```

```
    color: red;
```

```
}
```

④ Pseudo-classes Selector :-

pseudoclass is a keyword added to a selector than can change to the special state of the selected element(s).

ex:-

```
<button> Submit </button>
```

CSS :-

```
button:hover {
```

```
    color: yellow;
```

```
}
```

↑
color will change to
yellow when mouse
cursor will go
on the button

④ Multiple Selector

↳ also called grouping selector.

Multiple Selection of elements

Example

table, tr, td {

border: 1px solid black;

}

How to add Styling to HTML?

① Inline

② Internal

③ External

} ways of adding CSS

① Inline CSS :-

Inline CSS is applied in Tag ✓

< p style="font-style: italic"> Hello </p>

! _____

inline CSS
in opening tag.

② Internal CSS :-

We use `<style>` tag so which is written in `<head>` tag.

`<head>`

`<style>`

`h1 {`

`text-shadow: 1px 1px #FF0000;`
}

`</style>`

`</head>`

`<body>`

`<h1> Internal CSS applied </h1>`

`</body>`

③ External CSS :-

We create Separate CSS File & HTML File and then link to each other.

Link using `<link>` element in HTML.

`<link>` element is written inside

`head tag.`

Syntax

```
<link rel="stylesheet"  
      href="style.css">
```



style.css

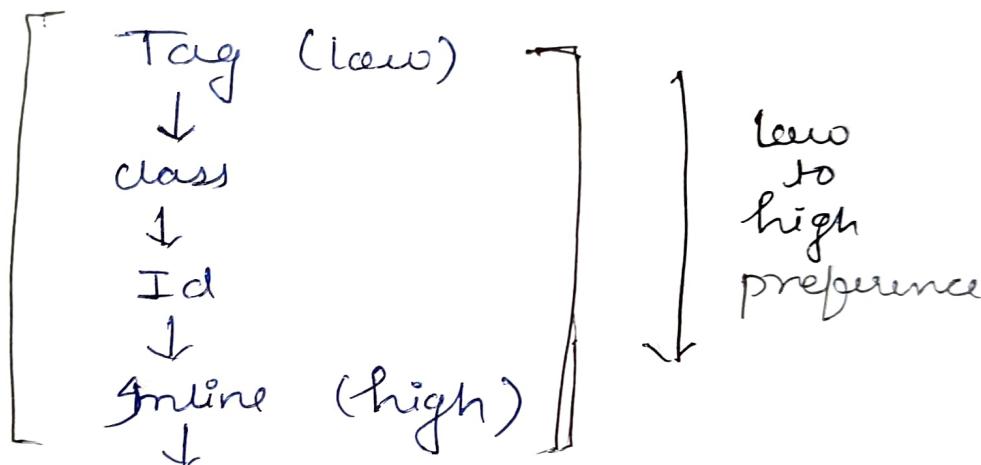


index.html

Specificity :-

- tag selector
- class selector
- id selector
- inline CSS
- ! Important (Bad practice)

To know whose Specificity is this:



! Important (important removes every Selectors & use only selector where important is written)

```
<div class="abc" id="def"> kya haal hai </div>
```

<Style>

```
#def {  
    color: red;  
}
```

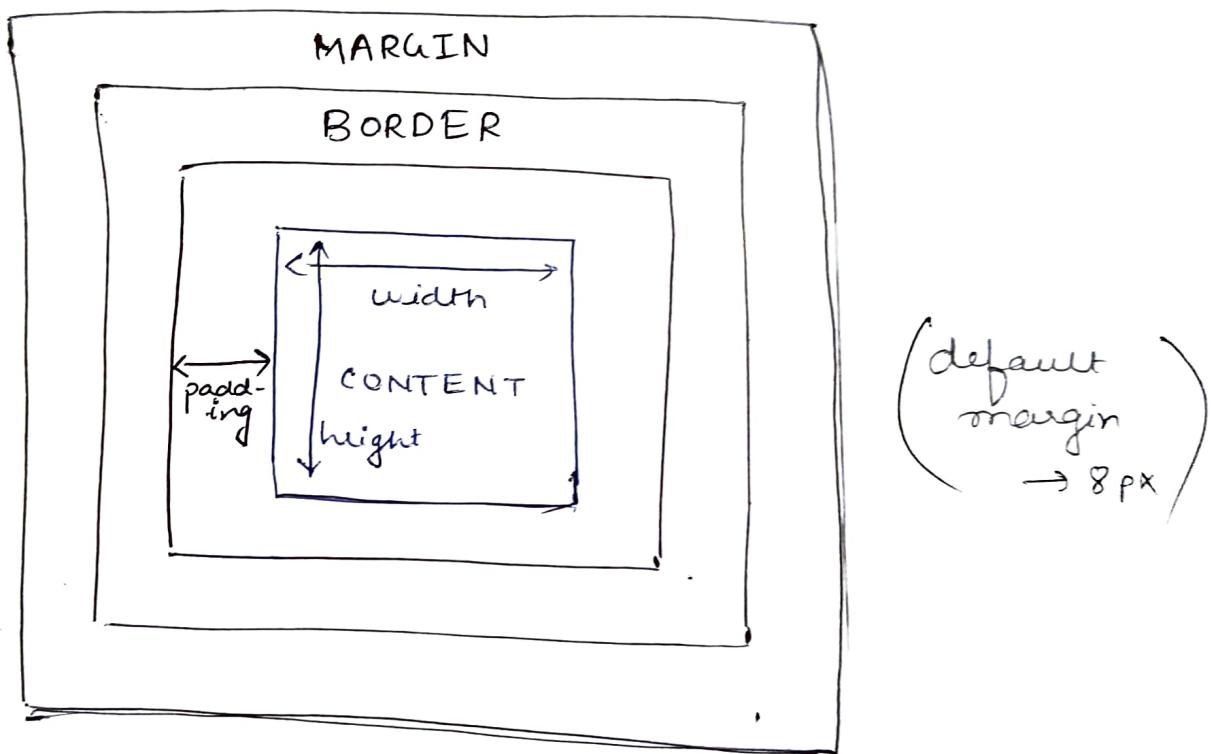
```
.abc {  
    color: blue;  
    text-style: italic;  
}
```

```
div {  
    color: green;  
}
```

color: red
&
text-style: italic
will be
applied

Box Model in CSS

- Basic building block of CSS
- every element on a page is a rectangular box & may have width, padding, borders & margins



- ✓ Space b/w Border & content → padding
- ✓ Outer side of Border Spacing → margin

$\text{margin} = 10\text{px}$
(includes all left, right, top, bottom)

margin-left
margin-right
margin-top
margin-bottom

} Specific

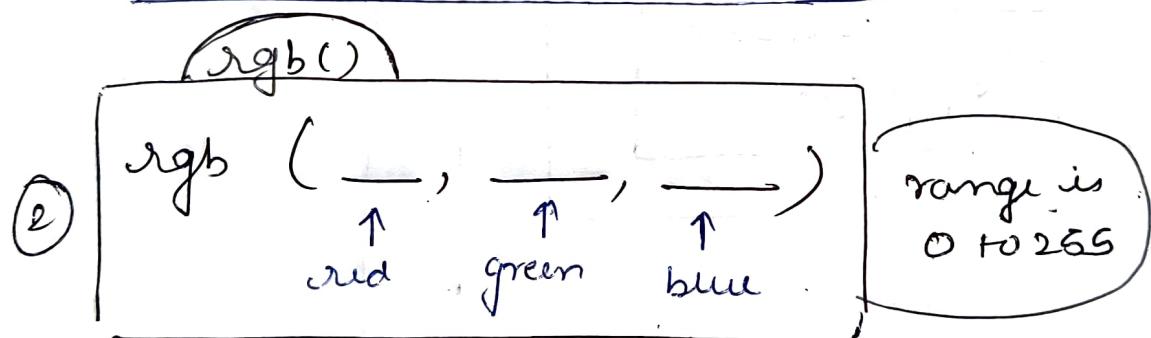
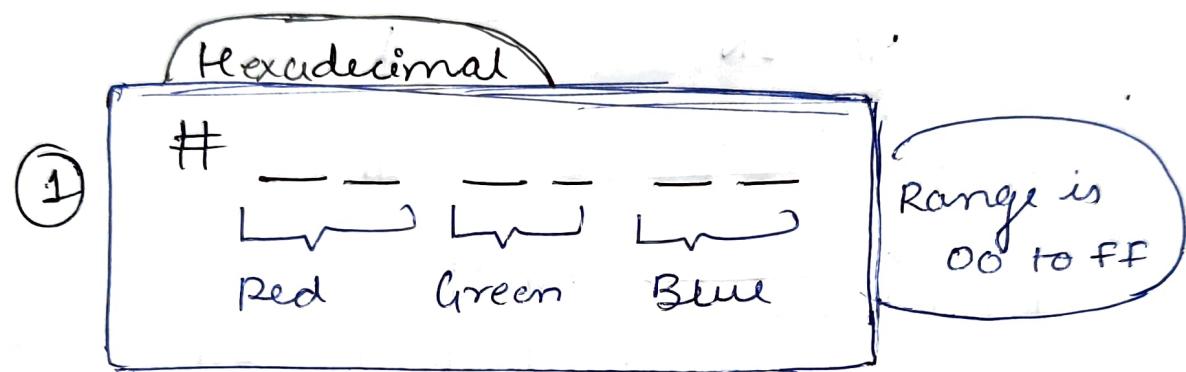
Same goes
for
padding.

Colors in CSS :-

Ways to add colors :-

(1) Hexadecimal colors (eg:- #00FFFF)

(2) RGB colors (eg:- rgb(0,0,255))



rgb() → rgb function.

③ predefined / Cross - browser Color Names .

↳ 140 predefined colors by Name

Units in CSS

$$px = \frac{1}{96} \text{ inch}$$

- (1) Absolute unit
 - mm
 - cm
 - in
- (2) Percentage unit
 - px (fixed) For all devices
 - 10% (percentage of parent element)
- (3) Relative unit

- ① Relative to font size
 - em
 - rem
- ② Relative to Document
 - vw
 - vh

✓ em → relative to parent element font size.

$$1\text{em} = 1 \times \text{parent size.}$$



$$1\text{em} = 1 \times 18 \checkmark$$

$$2\text{em} = 2 \times 18 = 36 \checkmark$$

✓ rem → relative to root (root is html)

$$1\text{rem} = 1 \times 16 = 16\text{px}$$

$$\checkmark \text{ vw} = \frac{1}{100} \times \text{width of viewport}$$

$$\checkmark \text{ vh} = \frac{1}{100} \times \text{height of viewport (display area)}$$

CSS-Basics-II

(Gradients, Shadows, Positioning, transforms
& FlexBox)

① Gradients :-

✓ Smooth transitions b/w the colors.

Types :-

- + Linear (down/up/left/right/diagonally)
- + Radial (circular center)
- + Conic (conical shape)

(i) Linear gradient

↳ background-image: linear-gradient(Red, Blue)
→ By default it is Top to Down

(to right),

(to right, Red, Blue) → Left to right

(to left, Red, Blue) → Right to left

(to top, Red, Blue) → Bottom to top

Diagonally { (to bottom right, Red, Blue) also left
(to top right, Red, Blue) also left

Directions in Linear gradient :-

- (1) Default Direction (top to Bottom)
- (2) Specific Direction (to right, to left etc. etc)
- (3) Angle (90deg, Red, Blue)
- (4) Using transparency
 - (to bottom, rgba(25, 0, 0, 0),
rgba(0, 0, 255, 1));

Transparency lies from 0 to 1)

(ii) Radial Gradient :-

background-image: radial-gradient
(Red, Blue, orange);
↑ ↑
Starting. last.

✓ Shape changing:-

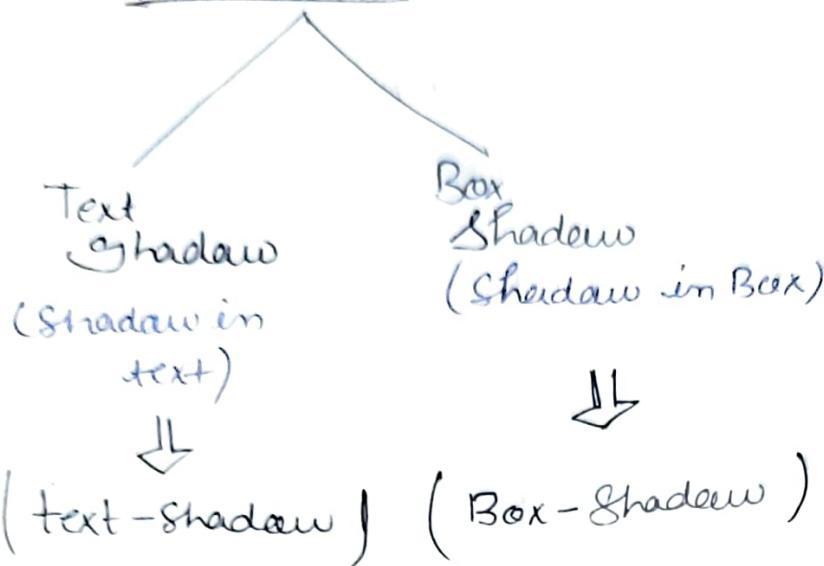
(circle, Red, Blue, orange)
↑
Shape
change.

✓ percentage :-

(yellow 5%, orange 10%, red 50%)

2

Shadows :-



Text Shadow

`text-shadow: 3px 3px 3px red;`

The diagram illustrates the components of a CSS `text-shadow` declaration. It shows the declaration `text-shadow: 3px 3px 3px red;` with arrows pointing from each part to its corresponding meaning:

- An arrow points from the first `3px` to the label "horizontal distance from text".
- An arrow points from the second `3px` to the label "vertical distance from text".
- An arrow points from the third `3px` to the label "Blur effect".
- An arrow points from the word "red" to the label "color of shadow".

✓ default shadow = text color.

Box Shadow

Box-Shadow : 10px 10px 10px ;
Horizontal → vertical
 ↓ ↓
 Blur

✓ By default shadow = text color

else,

else,  center of shadow
Box-Shadow : 10px 10px 10px red;

✓ we can also add multiple text / Box Shadow

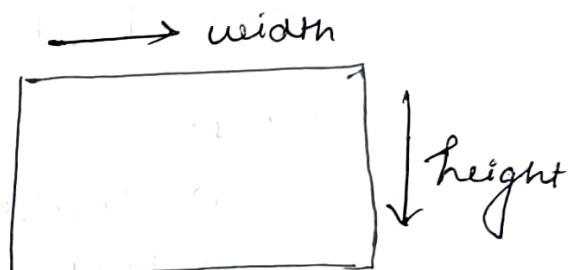
✓ By giving -ve values we can change the direction of shadow.

* Spread - Radius in Box-Shadow

H/w

③ Dimension properties

- (1) width
- (2) height
- (3) min-height
- (4) min-width
- (5) max-height
- (6) max-width



✓ Max-height :-

if max-height is 500px

& content exceeds 500px then the content will ~~overflow~~ & will get out of the Div.

Then Div will change its size and adjust.

when content > 500px (overflow)

✓ How to handle overflow?

overflow: scroll; } (Imp)

✓ Min-height
if content is less than min-height
it will be inside the div.
if content gets bigger, the div block
will automatically become large.
content < block → Same size
content > block. → increase size

④ Overflow property :-

values:-

- 1) Visible
 - 2) Hidden
 - 3) Scroll
 - 4) Auto
- } when too big
to fit in
block formatting
content.

overflow: visible (By default)

↳ visible content when overflow
from box

overflow: hidden

↳ overflow content hide.

overflow: scroll

↳ Scrolling effect in your content

overflow: auto

↳ depends on content

④ Position property :- (positioning method)

- 1) Static (default) used for an element.
- 2) Relative → adjusted away from its normal position. (left / Right / Top / bottom)
- 3) Fixed → fixed place even if Scrolled.
- 4) Absolute → positioned relative to the nearest positioned ancestor if no positional ancestor it used document body as parent.
- 5) Sticky

✓ order wise → static

✓ Relative to normal position → Relative.

✓ Fixed even if Scroll → Fixed.

→ Absolute positioned elements are removed from the normal flow, & can overlap elements.

→ Absolute → moves along with Scrolling, used document body when No Ancestor

→ Sticky → positioned based on the user's Scroll position.

toggles b/w relative & Fixed,
depending on the Scroll position.

It is positioned relative until a given offset position is met in the viewport.

⑤ 2D transforms :-

↳ allow you to move, rotate,
Scale & Skew elements.

✓ transform property methods :-

- 1) translate()
- 2) rotate()
- 3) scaleX()
- 4) scaleY()
- 5) scale()
- 6) skewX()
- 7) skewY()
- 8) skew()
- 9) Matrix()

* translate() :- (Movement)

transform: translate (100px, 200px);
 ↑ ↑
 X-axis Y-axis

* rotate() :-

transform: rotate (45deg);

* scale() :- (zooming)

transform: scale (2, 3)
 ↑ ↑
Horizontal Vertical
Scaling Scaling

✓ we can also use decimal values here
ex: (1.5)

* skew() :- (Tilt)

transform: skew (20deg)

⑥ 3D Transform :-

Same operation wrt 3D -

3 axis

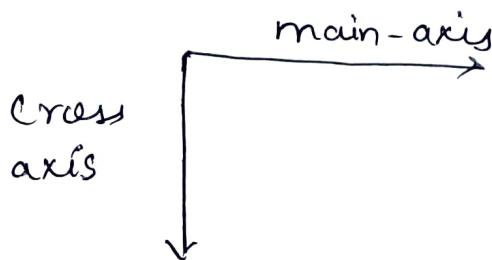
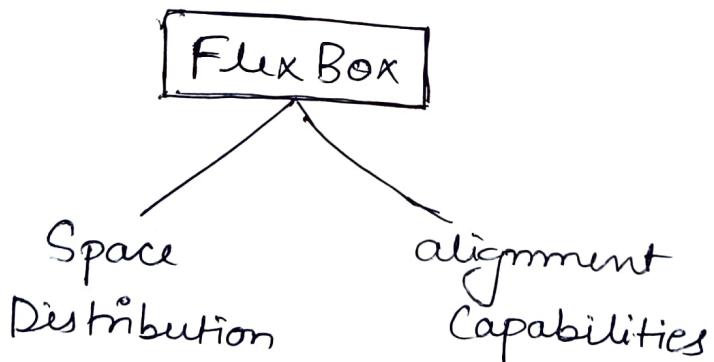
✓ Same properties will 3 axis :-

CSS - BASICS - III

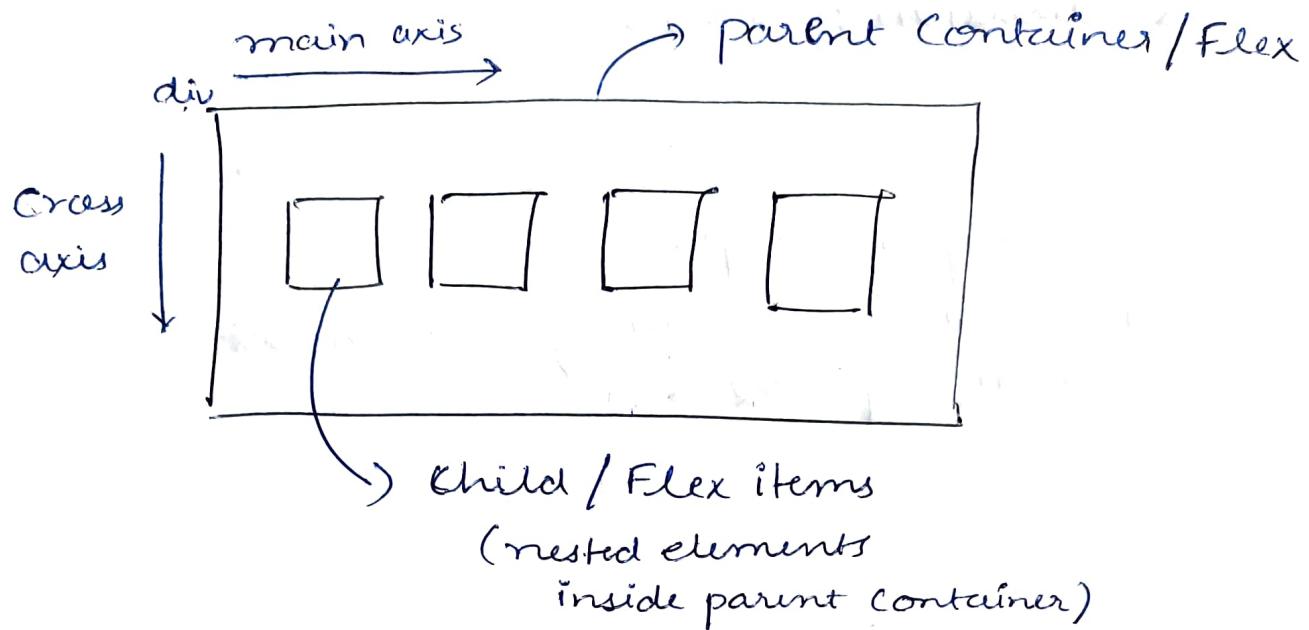
FlexBox :-

way to get more Flexibility in your layouts & to Simplify responsive layout design.

- It makes it easy to align elements on a 2D plane.
- It is a 1D layout model where we can do Space distribution b/w elements , with powerful alignment Capabilities.



parent - child relation in Flexbox



positioning tricks

- 1) Block
- 2) Inline
- 3) CSS position property
- 4) table
- 5) Flex Box (Imp)

For making Flex.

css → display: Flex;

by default it changes horizontal to
vertical.

but for manual change use.

Flex-direction : column;
flex-direction : row;
vertical; }
horizontal; }

For reverse, we use :-

Flex-direction : column-reverse;

Flex-direction : row-reverse.



✓ Flex-wrap : wrap;

this makes the width of div intact when squeezed.

another values { default → nowrap ✓
also → wrap-reverse; ✓

Flex-container properties :-

1) Flex-direction
row
row-reverse
column
column-reverse

2) Flex-wrap
wrap
no-wrap

3) Flex-basis
wrap
wrap-reverse

4) Justify-content

5) align-items

6) Align-content

Flex-item properties :-

✓ Flex-container property
is used for parent element

✓ Flex-item property
is used for nested / child
Flex items.

Flex Container properties :-

1) Flex-direction

- + flex-direction: row;
- + flex-direction: row-reverse;
- + flex-direction: column;
- + flex-direction: column-reverse;

2) Flex-wrap

- + flex-wrap: wrap
- + flex-wrap: no-wrap
- + flex-wrap: wrap-reverse.

3) Flex-Flow (Shorthand notation)

↳ we can add Flex direction & Flex wrap in one line.

Flex-flow: row wrap

4) Justify content

↳ align content in main axis

- + justify-content: flex-start (left end)
- + justify-content: flex-end (Right end)
- + justify-content: center
- + justify-content: space-around; (space b/w items)
→ space
regular
from left
right
- + justify-content: space-evenly
(equal spacing)

5) Align-items :- (alignment in cross axis)
vertically.

- + align-items: flex-start;
- + align-items: flex-end;
- + align-items: center;
- ✓ + align-items: stretch; (default)
- align-items: baseline;

✓ Clapping b/w elements / boxes -
use:

gap: 10px;

gap → Shorthand notation

↳ row-gap
↳ column-gap

6) Align-content :-

↳ handling Spacing b/w row
of boxes.

- + align-content: flex-start;
- + align-content: flex-end;
- + align-content: center;
- + align-content: space-between;
- + align-content: space-around;
- + align-content: space-evenly;

Flex-item properties :-

1) Order

- 2) Flex-grow → equal width & giving spaces to element
- 3) Flex-shrink → shrink speed (squeezing)
- 4) Flex-basis → used for giving width.
- 5) Flex → all above property in single line
- 6) Align-self.

1) Order :-

put in order with values.

```
#box1 {
```

```
    order: 3;
```

```
}
```

2) Flex-grow

```
Flex-grow: 1;
```

This property is for giving space to element.

3) Flex-shrink (by default - 1)

Shrink speed when squeezed.

The layout ..

4) Flex-basis :-

Flex-basis: 100px;

we can also use %.

a particular item will take the % width from total ✓

Width V/S Flex-basis

The overflow content gets hidden

the width increases with content.

Flex-basis makes it responsive

5) Flex :-

use to write all properties in one single line.

(order, Flex-grow, Flex-shrink & Flex-basis)

↓
all in one

Flex: 3 4 2 120px;

6) Align-self :-

Self aligning ↗

align-self : Stretch,
Flex-end,
~~flex-end~~,
Flex-start,
Center;

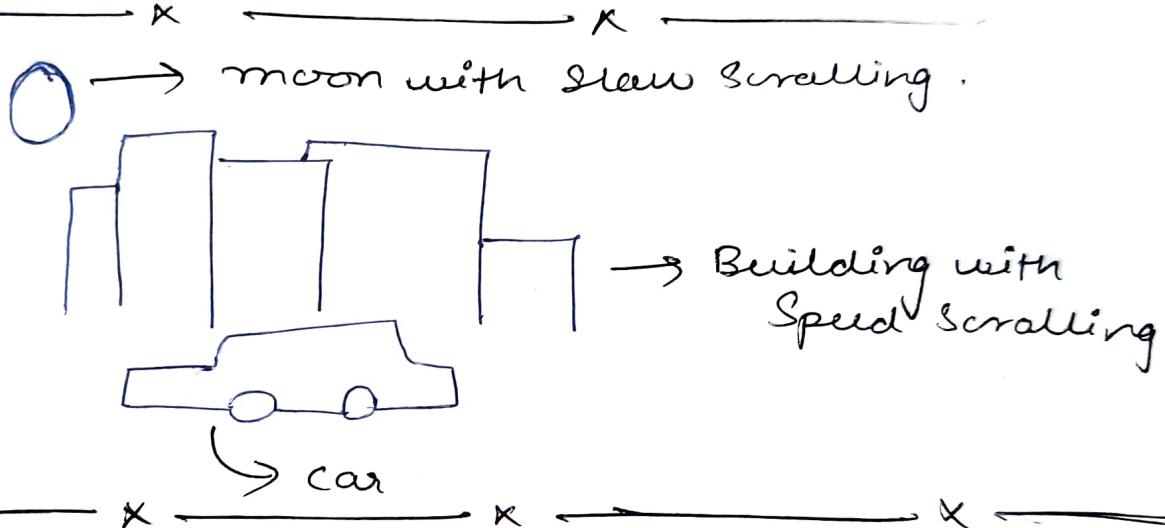
for each & every element;

Parallel - website :-

parallel Effect?



In a Single Frame when Speed of different objects is different, that effect is parallel effect.



→ First we overlap our images

→ Then, Foreground image near, Background image far (3-D effect)

→ Then when we will scroll one will go fast & one will move slow.

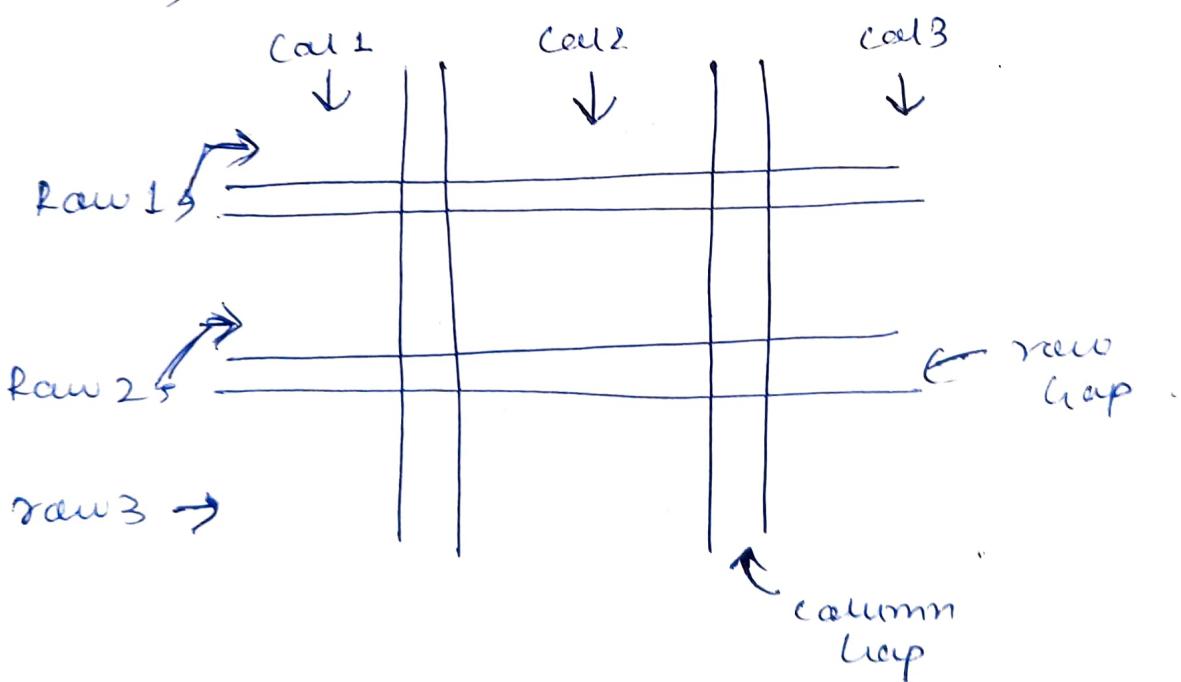
CSS - Basics - 4

CSS Grid :-

A grid is a collection of horizontal & vertical lines creating a pattern against which we can line up our design elements.

→ 2D layout, with rows & column gap

→ This is Matrix



✓ Flex-Box → 1D

✓ Grid → 2D

For Grid Column :-

grid-template-columns : repeat(3, 1fr);
can also be written
as 1fr 1fr 1fr

This means
equally
divided in 3
parts.

we can also give
values in pixel.

as 200px 250px 200px;

For Grid Row :-

grid-template-rows : repeat(4, 1fr);
↓ ↓
4 times equally
 divided

For Gap in Grid

gap : 15px;

Line-Based-placement :-

- 1) grid-column-start
- 2) grid-column-end
- 3) grid-row-start
- 4) grid-row-end
- 5) Shorthand properties : grid-row, grid-column,
grid-area.

Some more Properties

justify-content

→ start

→ space-around

→ center

align-items

→ align-items

justify-self

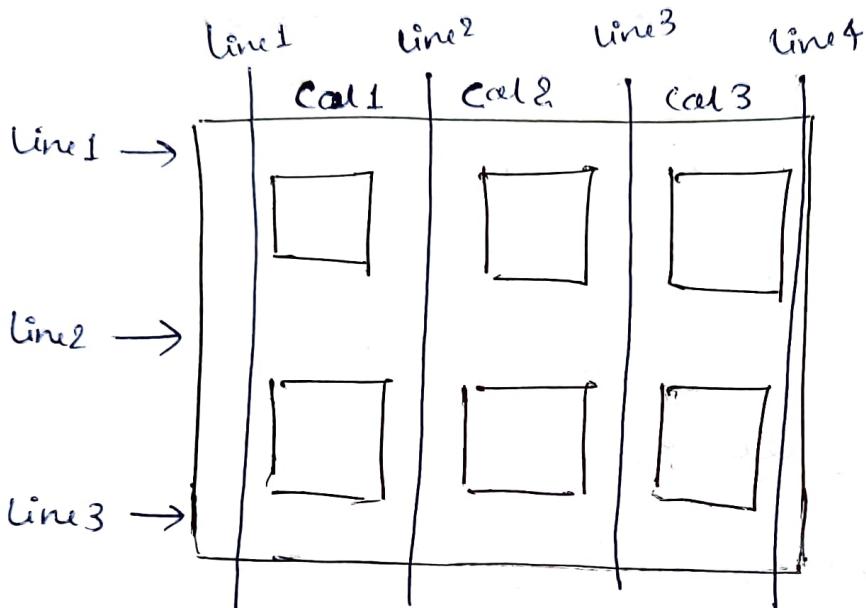
→ align-self

justify-items

→ place-items

→ place-self

Grid & Responsiveness



↗ grid-area: — / — / — / —
 ↑ ↑ ↑ ↑
 grid grid grid grid column
 row column row column
 Start Start end end

Single line giving
 values of rows &
 columns in grid ✓

Grid-area :-

The grid-area property specifies a particular area or set of rows & columns that a grid-item occupies. It is applied to the grid item itself with CSS e.g:-

- Item & grid-area : 1/2/3/3 }

Grid-template-area:

grid-template-area is the property used to name the rows & columns of a grid & to set its layout. It could look like this

• container &

display: grid;

grid-template-columns:

300px 300px 300px;

grid-template-rows:

250px 600px

grid-template-area:

"hd hd hd hd hd hd hd"

"sd sd sd main main main main main"

"ft ft ft ft ft ft ft";

• header &

grid-area: hd;

}

naming
& adding.

for adding e.g:-

header {

grid-area: hd;

}

~~grid-template
-area
Working~~

| Row → | Col 1 | Col 2 |
|-------|-------|-------|
| Row 1 | hd | hd |
| Row 2 | Side | Side |
| Row 3 | Ft | Ft |

↑ this is how naming is done & when we give value that occupies the space named ✓
eg:- #header { grid-area: hd; }

Q) If we have 3 rows & 2 columns defined in our grid & can name them with (grid-template-area) & then give more data (box) than defined what will happen to the extra boxes
→ They all will squeeze below the container

↑ so this will occupy hd space ✓

Advanced Grid Concepts :-

- 1) Fr unit → boxes are equally divided using fr
(eg: 1fr 1fr 1fr)
- 2) Repeat Function → repeat (3, 1fr)
 ↑ 3 times equally divided
- 3) Grid - auto-rows: minmax()
q
When there are unknown number of rows we use this property
- minimum & maximum size of the rows

Grid properties :-

- ✓ justify-content → for parent horizontal
- ✓ Align-content → for grid-items (child)
- ✓ justify-items
- ✓ Align-items → vertically
- ✓ place-items
- ✓ place-self

① justify-content

↳ content placed in main axis
(start, end, center, s-b, s-q, s-e)

② Align-~~content~~ items

↳ content placed in vertical axis



- ✓ justify-content ↗ changes position with value without change in width.
- ✓ justify-items ↗ changes ~~height~~ width according to content
- ✓ justify-self ↗ change in Specific box

Media Queries :- (Responsiveness)

css style in every device

- 1) "viewport" → The area of the window in which web content can be seen.
- 2) Media queries are used to set different style rules for different devices or sized screens. we use breakpoints to set the condition of a media query.
The logic is

$$@media (feature: value)$$
- 3) Essentially, media query breakpoints are pixel values that a developer/designer can define in CSS. When a responsive website reaches those pixel values, a transformation (such as the one detailed above) occurs so that the website offers an optimal user experience.

For Multiple break points

we use and keyword

@media (min-width: 800px) and
(max-width - 900px) {

• container {

// rules to change size.

}

}

Nested Grids

Nesting CSS grids is simple & can be done simply by using the display: grid rule for both a parent & child element

• container {

display: grid;

// . . .

}

one {

display: grid

}