Turwash Notes

Date - 7th Jan 2023

# CSS Basics - I

- appearance
- Styling
- Formatting

} How the content will appear?

CSS :- Cascading Style Sheet.

→ To Style the web page.

→ we add the properties

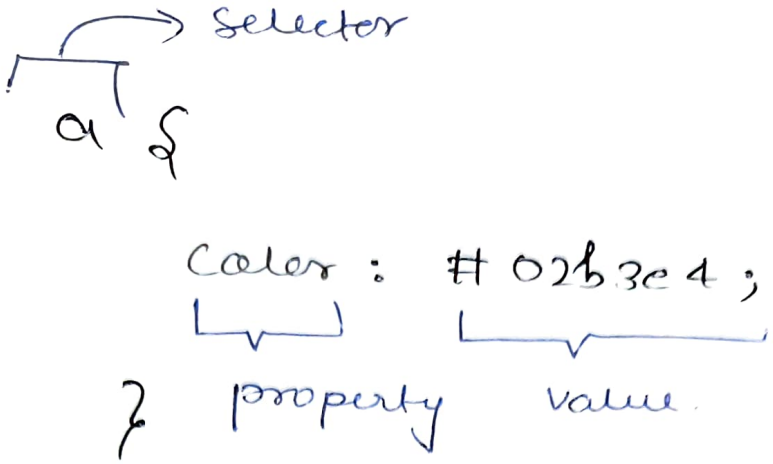
→ all about appearance (visual)

**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.




## 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 {
    color : red;
}
```

## (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 applid in Tag ✓

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

inline CSS in opening tag.

## ② Internal CSS :-

We use <style> tag in 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 Seperate 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">
```

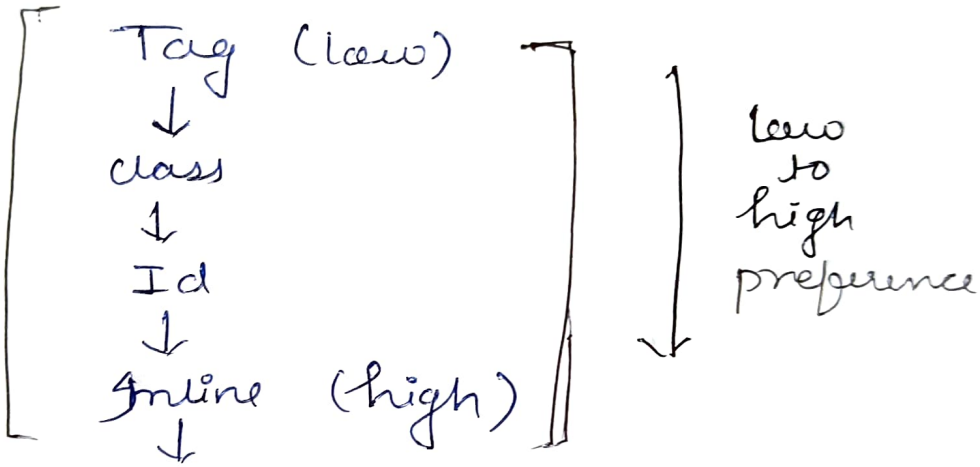

style.css


index.html.

# Specificity :-

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

To know whose Specifity is this:




---

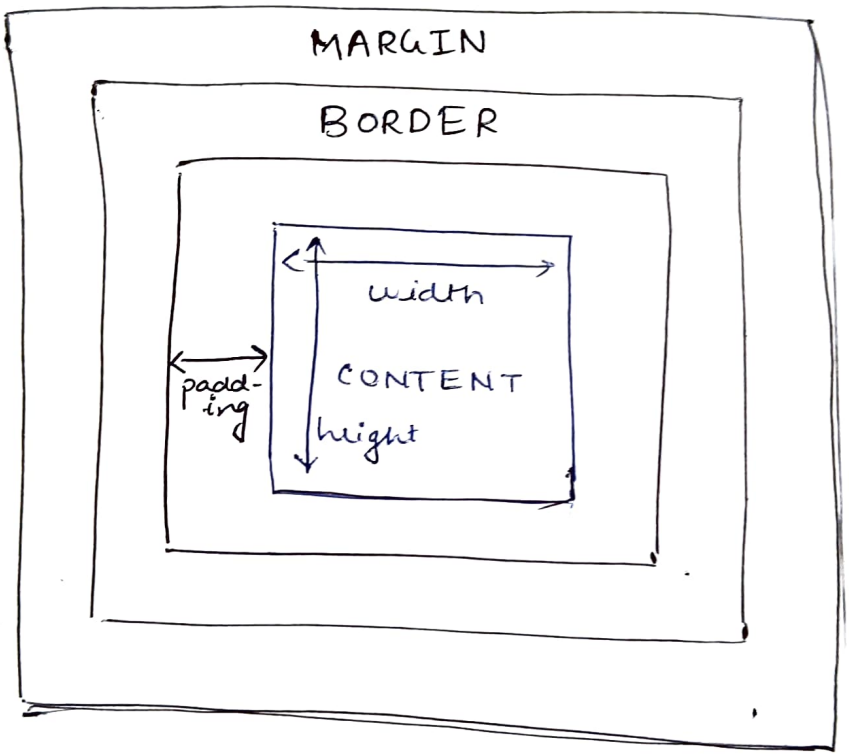
```
<div class="abc" id="def"> kya haal hai </div>

<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




(default margin → 8px)

✓ Space b/w Border & content → padding

✓ Outer Side of Border Spacing → margin

margin = 10px
(includes all left, right, top bottom)

margin-left
margin-right
margin-top
margin-bottom
} Specific

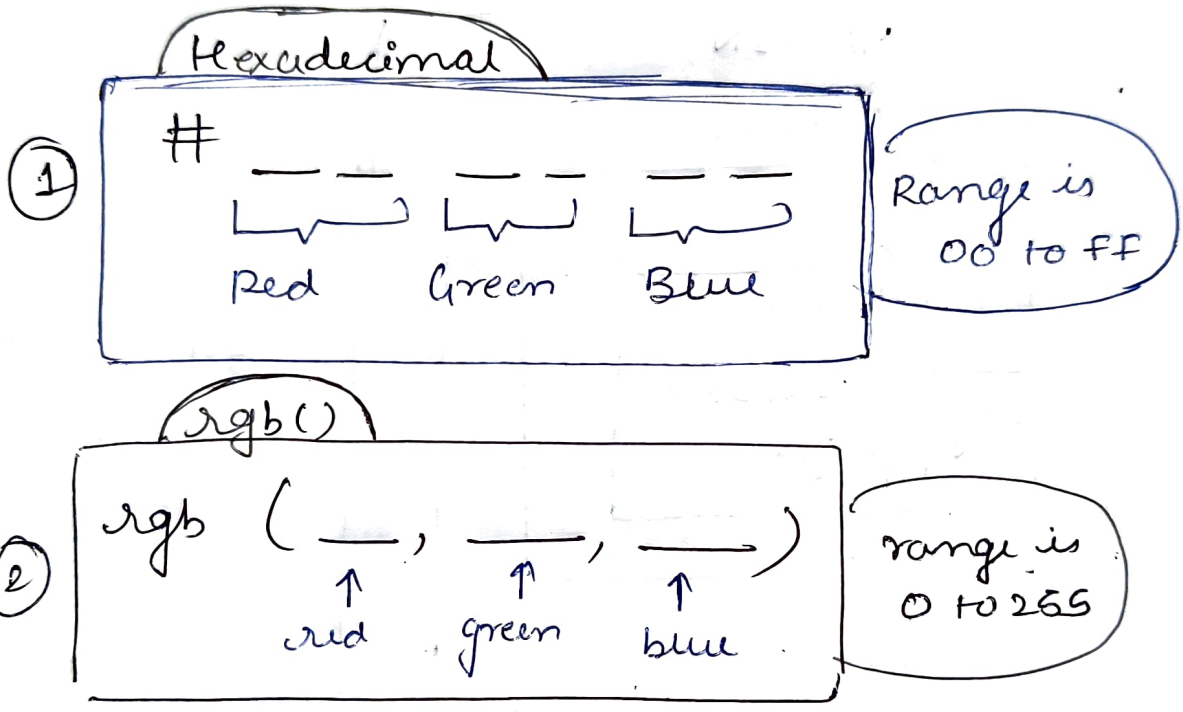
Same goes for padding.

# 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 = $\frac{1}{96}$ inch

(1) Absolute unit
- mm
- cm
- in
- px (fixed) For all devices

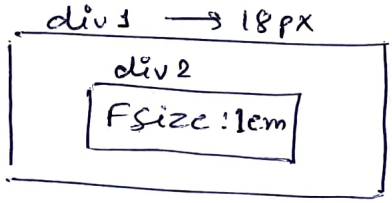
(2) Percentage unit → 10% (percentage of parent element)

(3) Relative unit
- ① Relative to font size
  - em
  - rem
- ② Relative to Document
  - vw
  - vh

✓ em → relative to parent element font size.

1 em = 1 X parent size.

eg:-




1 em = 1 X 18 ✓

2 em = 2 X 18 = 36 ✓

✓ rem → relative to root (root is html)

1 rem = 1 X 16 = 16px

✓ vw = $\frac{1}{100}$ x width of viewport

✓ vh = $\frac{1}{100}$ x height of viewport (display area)

Turvashi's Notes

Date :- 10th Jan 2023

# CSS-Basics-II

(Gradients, Shadows, Positioning, transforms && FlexBox)

## (1) Gradients :-

✓ Smooth transitions b/w the colors.

Types :-

- Linear (down/up/left/right/digonally)
- 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

Digonally { (to bottom Right, Red, Blue) also left
(to top Right, Red, Blue) also left

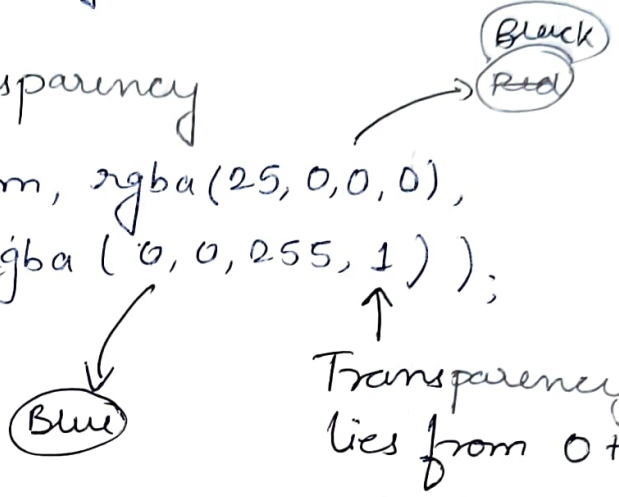
## Directions in Linear gradient :-

(1) Default Direction ( top to Bottom)

(2) Specific Direction ( to right, to left etc. etc)

(3) Angle ( 90deg, Red, Blue)

(4) Using transparency




## (ii) Radial Gradient :-

background-image: radial-gradient
( Red, Blue, orange );

Starting. last.

✓ Shape changing :-

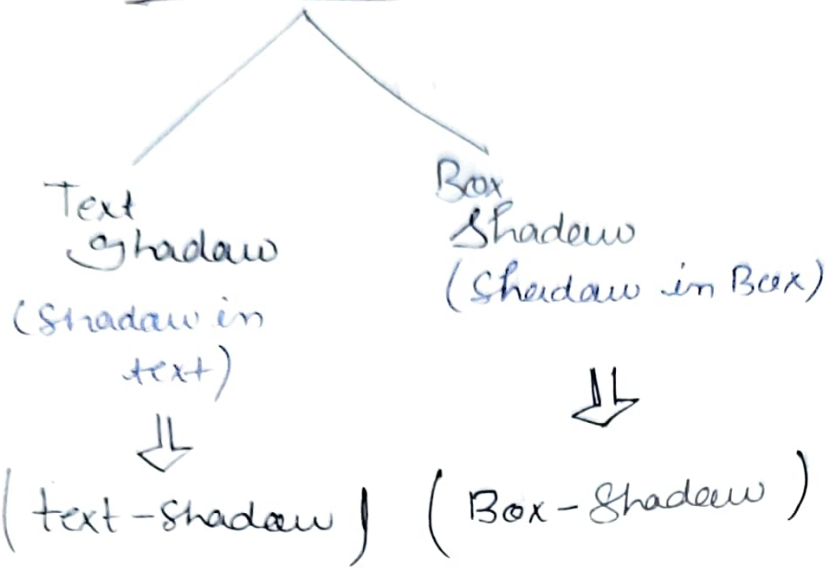
( circle, Red, Blue, orange )

Shape change.

✓ percentage :-

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

## 2 Shadows :-




### Text Shadow

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

- 3px → horizontal distance from text
- 3px → vertical distance from text
- 3px → Blur effect
- red → color of Shadow

✓ default Shadow = text color.

### Box Shadow

```
Box-shadow : 10px 10px 10px ;
```

- 10px → Horizontal
- 10px → vertical
- 10px → Blur

✓ By default Shadow = text color

else,

```
Box-shadow : 10px 10px 10px red;
```

- red → color of Shadow

✓ we can also add multiple text / Box Shadow

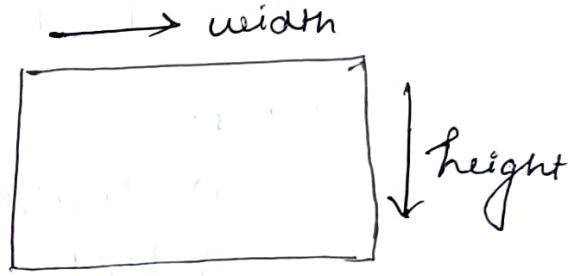
✓ By giving -ve values we can change the direction of Shadow.

* 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 ~~override~~ overflow & will get out of the Div.

Till 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

## (#) 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 used for an element)

1) Static (default)
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 s 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 Scaling ↑ Vertical Scaling

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

* **Skew() :-** (Tilt)

transform : Skew (20deg)

## (6) 3D Transform :-

Same operation w.r.t 3D ✓

3 axis

✓ Same properties will 3 axis :-

Turwash Notes

Date :- 12th Jan 2023

# CSS - BASICS - III

## # FlexBox :-

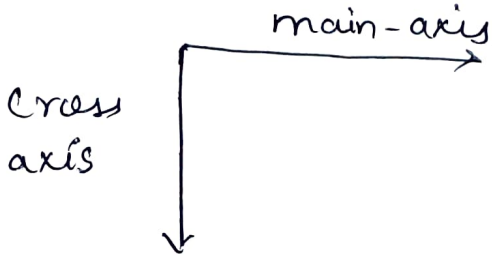
way to get more Flexibility in your layouts & to simplify responsive layout design.

→ It makes it easy to align elements on a 2D plane.

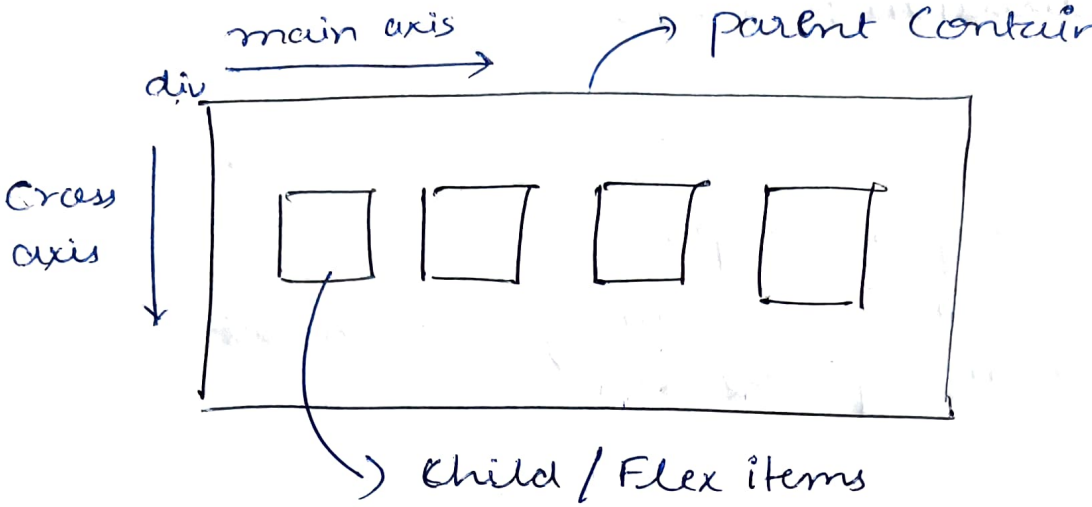
→ It is a 1D layout model where we can do space distribution b/w elements, with powerful alignment capabilities.


Space Distribution

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;   (vertical)
Flex-direction : row;      (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, wrap-reverse.
3) Flex-flow
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 equal spacing)
- justify content : space-evenly (equal spacing)

no space equal from left & right

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;

---

✓ Gapping b/w elements / boxes.

use.

gap: 10px;

gap → Shorthand notation

- row-gap
- column gap.

---

6) Align-content :-

↳ handeling 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~~ 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;
```

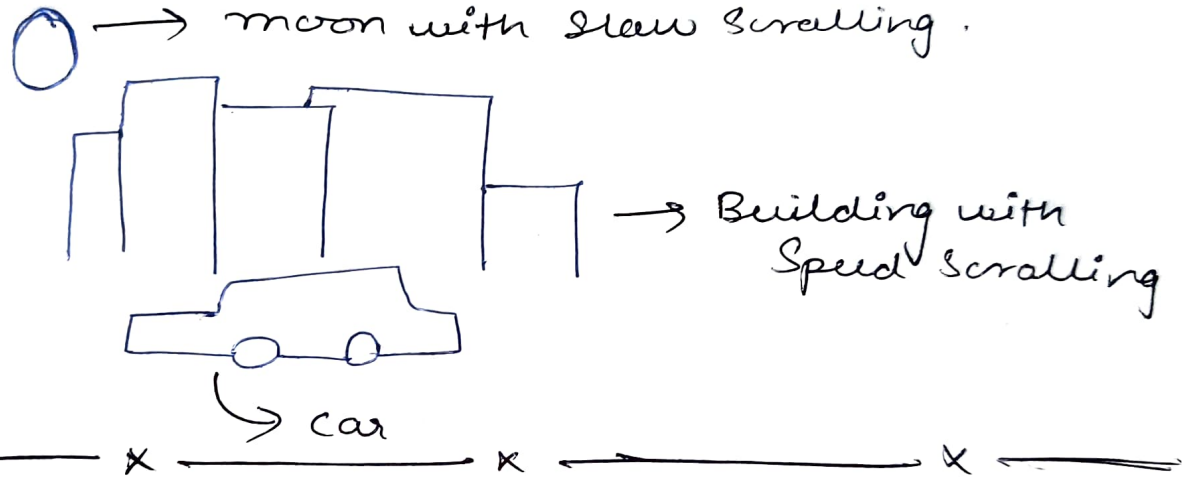
For each & every element;

# #parallax – website :-

parallax Effect?

↓

In a single frame when speed of different objects is different, that effect is parallax 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.

Tirwash Akotiy

Date :- 14th Jan 2023

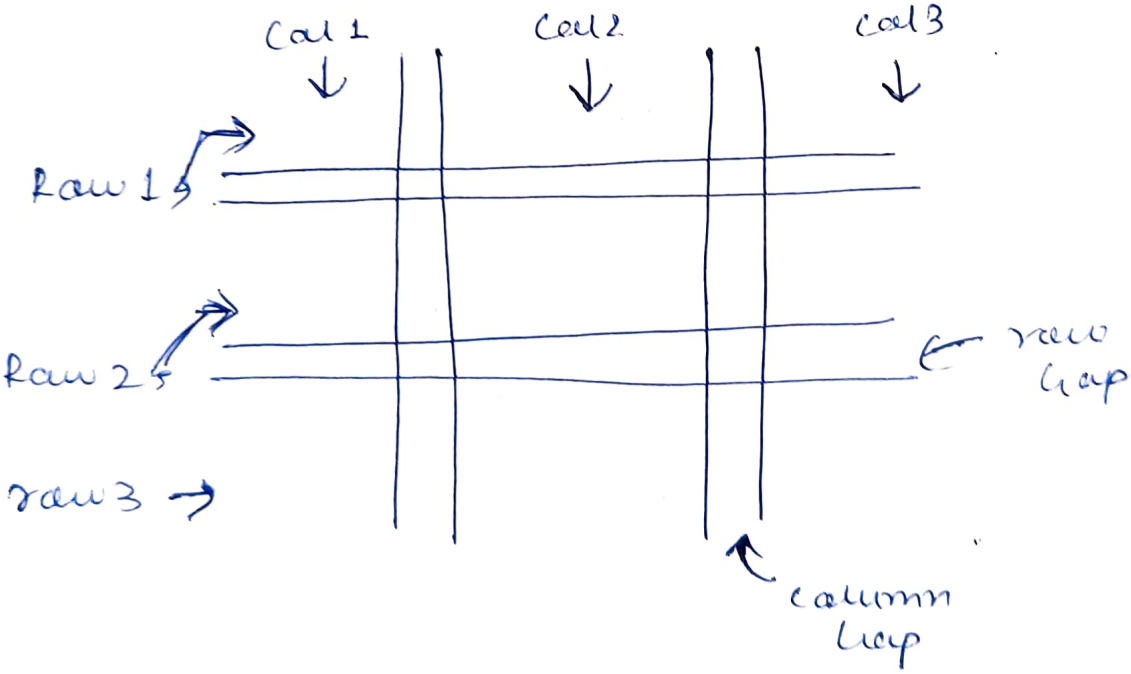
# CSS - Basics - 4

## 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 Raw :-

grid-template-raws : 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-raw-start
4) grid-raw-end
5) Shorthand properties : grid-raw, grid-column, grid-area.

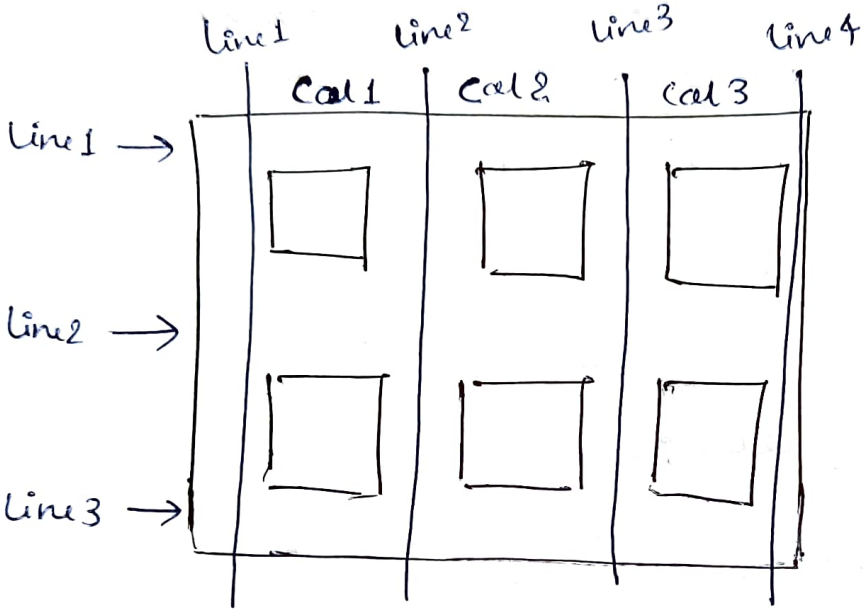
# Some more Properties

- justify-content
  - start
  - end
  - center
  - space-between
  - space-evenly
  - space-around
- align-content
- justify-self
- justify-items
- align-items
- align-self
- place-items
- place-self

Turwash Notes

Date :- 17th Jan 2023

# Grid & Responsiveness




* grid-area: _ / _ / _ / _

↑ grid row Start — ↑ grid column Start — ↑ grid row end — ↑ grid column 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 eg:-

```
.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 hd"
  "sd sd sd main main main main main"
  "ft ft ft ft ft ft ft ft";
}
```

naming & adding.

```
.header {
  grid-area: hd;
}
```

for adding eg:-

```
# header {
    grid-area: hd;
}
```

# grid-template-area working

| Raw → | Col 1 | Col 2 |
|---|---|---|
| raw 1 | hd | hd |
| raw 2 | side | side |
| raw 3 | ft | ft |

↑ this is how naming is done & when we give value that occupies the space named ✓

eg:- #header {
grid-area: hd;
}

↑ so this will occupy hd space ✓

Q)

If we have raws & 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

# Advanced Grid Concepts :-

1) Fr unit → box are equally divided using fr
eg: 1fr 1fr 1fr

2) Repeat Function → repeat (3, 1fr)
↑ 3 times ← equally divided

3) Grid-auto-rows : minmax()
↑ when there are unknown number of rows we use this property
← minimum & maximum size of the rows

## Grid properties :-

For parent
- ✓ justify-content
- ✓ Align-content
- ✓ justify-items
- ✓ Align-items

For grid-items (child)
- ✓ justify-self → horizontal
- ✓ Align-self
- ✓ place-items → vertically
- ✓ place-self

① justify-content
↳ content placed in main axis ←→
(Start, end, center, s-b, s-a, 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: 600px) 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
}
```

Turvash's Notes

Date:- 19th Jan 2023

# Animations In CSS


✓ CSS Style 1 to CSS Style 2 gradual change b/w them is called Animations.

## Functions in CSS :-


piece of code where we get output when we give Input to it

eg:- rgb ( )

repeat ( ) ← i/p parameters

↑ name

Scale(), translate(), gradiant

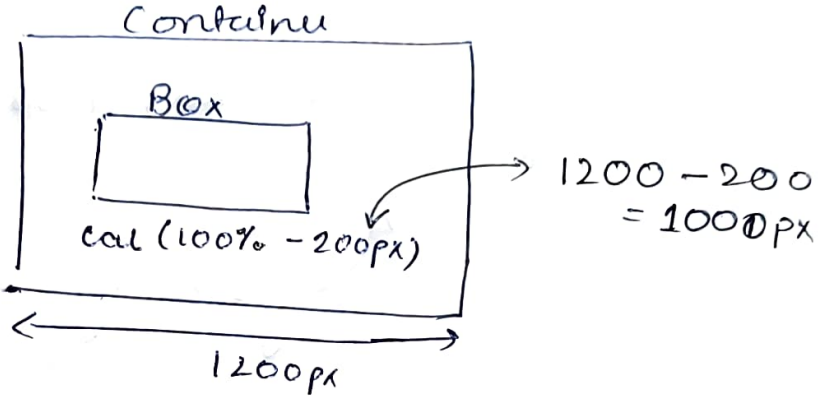
rgba(), url(), minmax() etc etc.

Now, :-

# Math functions in CSS :-

① calc() (for width)

↖ To solve expression (+, -, *, / etc) & use the value




cal ( parentvalue expression value )
↳ cal ( 100% — 200px )

② min (-,-) (for width)

③ max (-,-) (for width)

④ minmax (—, —) provides range b/w min & maximum
↑ min to ↑ max

# Variables in CSS :-

When you have a long blocky code (layout) & you want to add styles, but when you want to change style then from variables you can change all value from single place.

✓ variables works only inside the Blocks. when declared locally.

✓ When variable is declared in the :root then also it will work

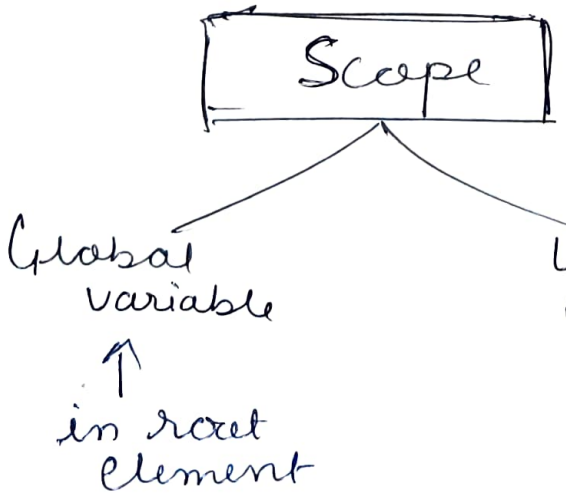
```
:root {
    --dark-red: #981a2C;
}
```

globally defined variable can be used anywhere.

but.

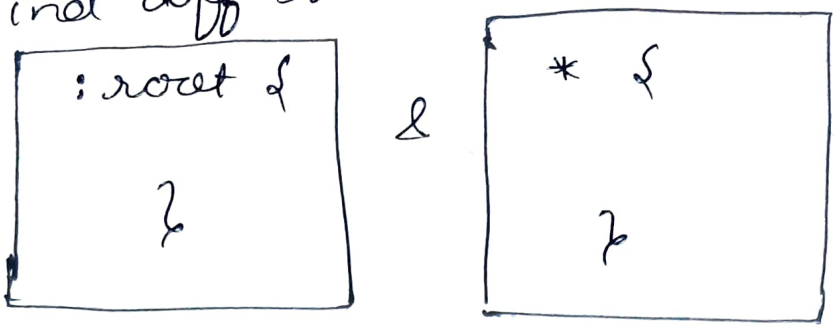
```
.container {
    --dark-red: #981a2C;
}
```

local variable only be used within blocks.



✓ Naw.

Find diff b/w



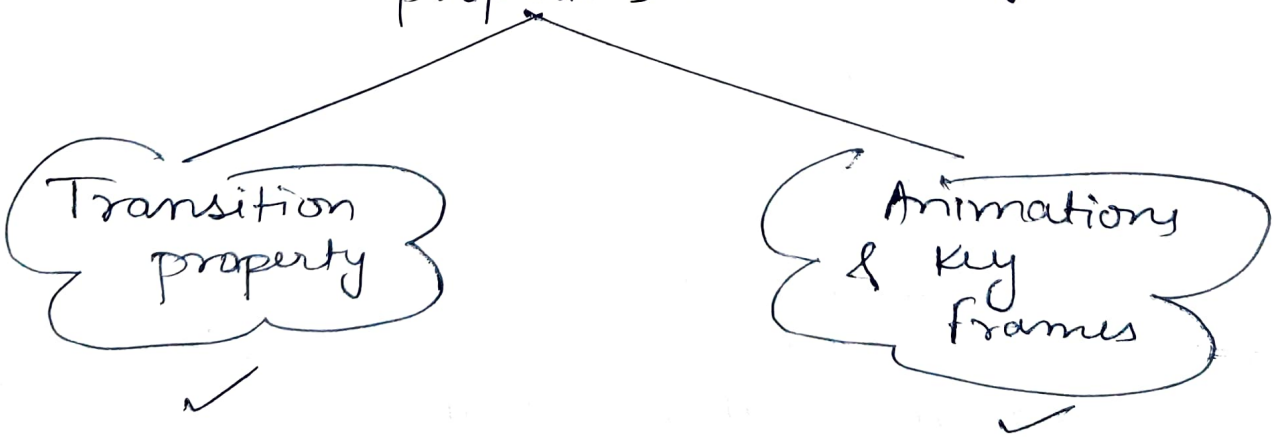

## Animations :-

## Transitions in CSS :-

1) Transition Timing (duration)
2) Transition function
3) Transition Delay (time after which transition occurs)
4) Transition property, (in which property we apply transition)

---

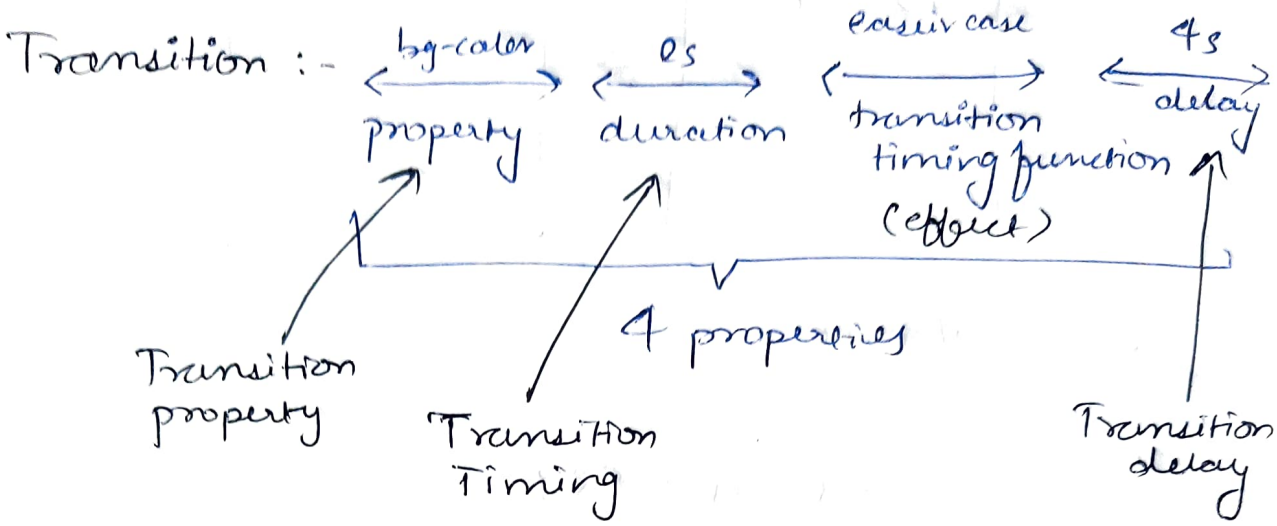
We can animate our objects using two properties.




✓ ✓

## ① Transition property :-

## Transition shorthand property :-




## ② Animation property :- & Keyframes :-

give properties inside the class you want to have animation

Then to run animation

@keyframes Name {

}

**properties :-**

animation-name
animation-duration
animation-iteration-count
animation-timing-function
animation-delay
animation-direction
animation-fill mode

} we write these properties inside the class css where we want animation to run.