Cascading style sheets (CSS)

Falling of styles (m is silent) — adding of style/design — doc/file

⇒ 1994, "Hakon wium lie" is Published css.

⇒ It is used to design/style web-Pages.

⇒ It is Re-Published 1996 co-author "bert bos".

ways of writing css:-

There are 3 ways are there:- In-line, Internal and external.

In-line css:- It should be written in a same-line inside the opening tag.

```
Eg:- <body>
       <h1> Hello world </h1>
         ↓
       style = "border : 2px solid
                color : Red ;
                text-align : center ;"
```

⇒ In In-line css the drawback is creates confusion and makes complexity.

Internal css:- It should be written in a same Page, inside style tag (Paired tag).

Eg:-

⇒ Priority wise first In-line css, second Internal css and third External css.

Eg for Internal css :-

```
<html>
  <head>
    Here also we can write style tag.
  </head>
  <body>
    <h1> Hello </h1>
        id = "demo"
    <h2> Hii </h2>
    class = "demo1"
    <style>
        # demo {
        }
        .demo1 {
        }
    </style>
  </body>
</head>
</html>
```

External css :- It have 2 steps :-

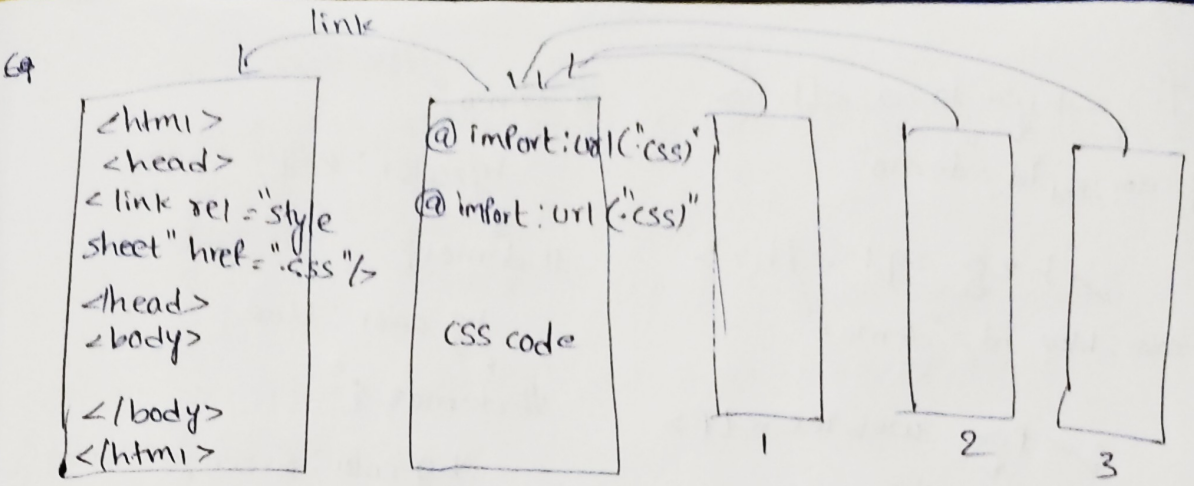
=> step 1 :- create a .css file.

=> step 2 :- link .css file to .html file with the help of link tag (unpaired tag).

* To Pair one .css file to multiple .css files we have 1 a Property called @import:url("Path")

* link tag have two attributes :-

relation → rel = "stylesheet" href = ".css"



Selectors in CSS :-

⇒ It is used to target an element to Provide css Property.

Syntax :- selector-name {

css Properties --

}

Types of selectors :-

1. Simple selector
2. combinator selector
3. Pseudo Element
4. Pseudo class
5. Attribute selector

1) Simple selector → ID, class, Element, Grouping, universal

i) ID selector :-

→ It should be always unique names.

→ Represented using # (hash).

Eg:-

<h1> java </h1>
bg-color: Red id = "demo"

<h2> sql </h2>
bg-color: blue id = "demo1"

 web tec
bg-color: Green id = "demo2"

```
#demo{
    bg-color: Red;
}
#demo1{
    bg-color: blue;
}
#demo2{
    bg-color: Green;
}
```

ii) Class selector :-

→ It always takes same-name.

→ Represented using . (dot).

Note :- Class selector will accept more than one class name

Eg:-

<h1> React </h1>
bg-color: Red class = "demo"

<h1> JS </h1>
bg-color: Red class = "demo demo1"
text-align: center

<h1> node.js </h1>
bg-color: Red class = "demo demo2"
text-decoration: underline

```
.demo{
    bg-color: Red;
}
.demo1{
    text-align: center;
}
.demo2{
    text-decoration: underline
}
```

iii) Element selector :-

→ Used to target element using tag-name.

→ Represented using tag-name.

Eg: used in ordered & unordered list.

Eg:-

```
<h1> Java </h1>
<h1> Python </h1>
<div> sql </div>
```

```
h1{
  bg-color : blue;
}
div {
  text-align : center;
}
```

iv) Grouping Selector :-

→ when different selectors are taking same CSS Property.

→ Represented by using , (comma).

Eg:-

```
<h1> Java </h1>
  id = "demo" → bg-color : Green;
<h1> Python </h1>  text-align : center;
  class = "demo1" → bg-color : Green
                    font-size : 20px;
<div> sql </div>
```

```
#demo{
  text-align : center;
}
.demo1{
  font-size : 20px;
}
```

Grouping selector

```
#Demo, .demo1, Div
{
  bg-color : Green;
}
```

v) Universal selector :-

→ Universal selector will works similar likes a body tag.

→ Represend by using * (astrick)

Eg:-

```
body{
  bg-color : black;
  color : white;
}
```

```
*{
  bg-color : black;
  color : white;
}
```

Combinator selector :-

→ It is a combination of more than one selector.

→ It is used to avoid simple selector (id, class).

→ There are four types are there :-

* Descendent selector (space)
* child " (>)
* adjacent " (+)
* General " (~)

1. Descendent selector :-

It is used to target both direct and In-Direct child.

* Represented by space (tab space)

Eg :-

```
<ol>
    <li> Home</li>
    <li> login </li>
    <li> Reg </li>
    <span><li> support</li></span>
</ol>
 <span> Hii </span>
```

```
ol li {
    bg-color : Red;
}
```

2) child selector :-

→ It will targets only Direct child.

→ Represented using ' > ' (greater than)

```
ol > li {
    bg-color : blue;
}
ol span > li {
    bg-color : blue;
}
ol > span > li {
    bg-color : blue;
}
```

3) Adjacent selector:- works like id.

→ It will target right after next-sibling.

→ Represented by '+' (plus).

Eg:-
```
<body>
  <ol>
    <li> Home </li>
    <li> login </li>
    <li> Reg </li>
    <span> <li> support </li> </span>
  </ol>
  <span> contact </span>
  <span> Help </span>
</body>
```

① bg-color: ~~Green~~ brown;

② bg-color: yellow

③ bg-color: Green

```
ol + span {
    bg-color : yellow;
}

ol + span + span {
    bg-color : Green;
}

ol li + li + li {
    bg-color : Brown;
}
```

4) General selector:-

→ It will target all the siblings.

→ Represented using '~' (tilet)

Eg:-
```
<ol>
  <li> Home </li>
  <li> login </li>
  <li> Reg </li>
  <span> <li> support </li> </span>
</ol>
<b>
  <span> contact </span>
  <span> Help </span>
</b>
```

```
ol ~ span {
    bg-color : Red
}
```

bg-color : Red

Pseudo Element (::)

→ It is used to style / Design specific Part of an element.

→ Represented using :: (double colon).

- ✓ :: first-letter
- ✓ :: first-line
- ✓ :: Before
- ✓ :: After
- ✓ :: Marker
- ✓ :: selection

1) :: first-letter :-

→ It is used to design / style first-letter in a element.

Eg:-
Lorem ----
---- ---
---- ---

```
p::first-letter{
    Bg-color : Brown;
    color : white;
    font-size : 25Px;
}
```

2) :: first-line :-

→ It is used to design / style first-line in a element.

Eg:-

```
p::first-line{
    font-size : 25Px;
    text-decoration : underline;
}
```

3) :: Before :-

→ It is used to Place an content / image before the element.

eg:- ~~<p>~~ P:: Before{

img ← content : url ("Path")

content : "Hello world"

4) :: After :-

→ It is used to Place image/content after the element.

eg:- P:: After {

content : url ("Path")

}

5) :: Marker :-

→ It is used to style/design items in a list.

eg:-

 React

 java

 sql

li:: marker {

color : "Red";

content : "☺"

}

li+li ::marker {

content : "☹"

}

6) :: selection :-

→ It is used to design/style when element will be selected.

eg:- P :: selection {

Color : white;

bg-color : Black;

}

Pseudo-class (:) :-

→ used to style specific state of an element.

→ Represented using : (colon).

1. Dynamic - Pseudo class

:link	:focus
:active	:checked
:visited	& :hover

Balue - un-visited state
Red - Active state
Purple - visited state

2. Structural - Pseudo-class

:first-child	:first-of-type
:last-child	:last-of-type
:nth-child( )	

1) :link :- Used to style state of un-visited link.

Ex:- click for flipkart

```
a : link {
    Bg-color : yellow;
    color : Red;
    }
```

2) :active :- used to style state of an active link.

Ex:-

```
a : active {
    Bg-color : black;
    color : white
    }
```

3) :visited :- used to style state of visited link.

Ex:-
```
a :visited {
        Bg-color : crimson;
        color : Blue;
        }
```

4) :focus :- It is used to style Particular input field when end-user focus on it.

Ex: <input type = "text" id = "" name = "auto focus" />

```
input : focus {
        Height : 100Px;
        width : 400Px;
        bg-color : yellow
        }
```

5): checked :- Used to style when end-user selects / checked Particular checkbox (or) Radio.

Ex: <input type = "checkbox" />

→ without using any selector we can do like this also.

```
input [type : "checkbox"] : checked {
                Height : 50Px;
                width : 50Px;

< input type = "Radio" />

input [type : "Radio"] : checked {
                Height : 100Px;
                width : 100Px;
                }
```

8

6) :hover ;- It is used to style particular element when end-user hover on it.

Ex:-
```
<button>
    click
</button>
```

```
button : hover {
    bg-color : blue;
}
```

Structural - Pseudo - class

- ✓ : first-child
- ✓ : last-child
- ✓ : nth-child
- ✓ : first-of-type
- ✓ : last-of-type

Eg:-
```
<body> <h1> Hi </h1>
    <p> Lorem </p>
    <h1> first-child </h1>
    <Div>
        <h1> first-child </h1>
        <h1> second-child </h1>
        <h1> Third-child </h1>
        <h1> fourth-child </h1>
    </Div>
    <h1> third-child </h1>
    <p> Lorem </p>
    <h1> Hello </h1>
</body>
```

```
h1 : first-child {
    bg-color : Red;
}

h1 : last-child {
    bg-color : blue;
}

h1 : nth-child (odd) {
    bg-color : crimson;
}

h1 : first-of-type {
    bg-color : Green;
}

h1 : last-of-type {
    bg-color : Yellow;
}
```

Text Property :-

color :- color-name, RGB, RGBA, Hexadecimal, HSL, HSLA.
(6 ways) (3)

Text-indent :- values
- Pixels
- %

Text-align :- Right, centre, left, justify

Text-decoration :- Underline
over-line
line-through
none

Text shadow :- x-axis, y-axis, blurness, color-name
10Px 2Px 1Px Red

letter-spacing :- values
- Px
- em

word-spacing :- values
- Px
- em

line-height :- values
- Px
- em

Text-decoration-style :- underline, over-line, line-through.

Text-decoration-color :- color-name

Text-decoration-line :- dotted, dashed, doubled, wavey.

use in anchor tag

Text-transform :- Uppercase, lowercase, Capatalize

css units

rem → root-element

em → element

px → Pixels

% .

vh (viewport height)

vw (viewport width)

By using text-overflow we help for this Property

- ✓ width
- ✓ border
- ✓ text-wrap : no wrap
- ✓ overflow : hidden

Text-overflow :- clip, eclipse. eg:- Lorem....

overflow :- visible, hidden, scroll.

overflow-x :- visible, hidden, scroll

overflow-Y :- visible, hidden, scroll

font-Property :-

font-size : values (small, medium, large, x-large, x-small)

- → Pixels
- → em
- → %

font-weight : 100-900, bold, bolder, lighter.

font-family : serif, san-serif, monospace, fantasy, curvesy.

font-varient : small-caps

font-style : static, obilique

Background-Property :-

It has 2 types :-

1. Background-Image.
2. Background-color : 6-ways we have to write.

Background-image : url("Path")

Background-Repeat : Repeat ; → Here default repeat.

Repeat-x ;

Repeat-y ;

no-Repeat ;

```
h1 {
  bg-image : url("Path");
}
```

Background-size : cover ;

contain ;

100%.

50%.

10%.

Background-Position : top, bottom, center, left, Right

bottom Right, top left

background-attachment : fixed ;

scroll ;

II Box-Model :-

→ It is a container that has 3-Properties :-

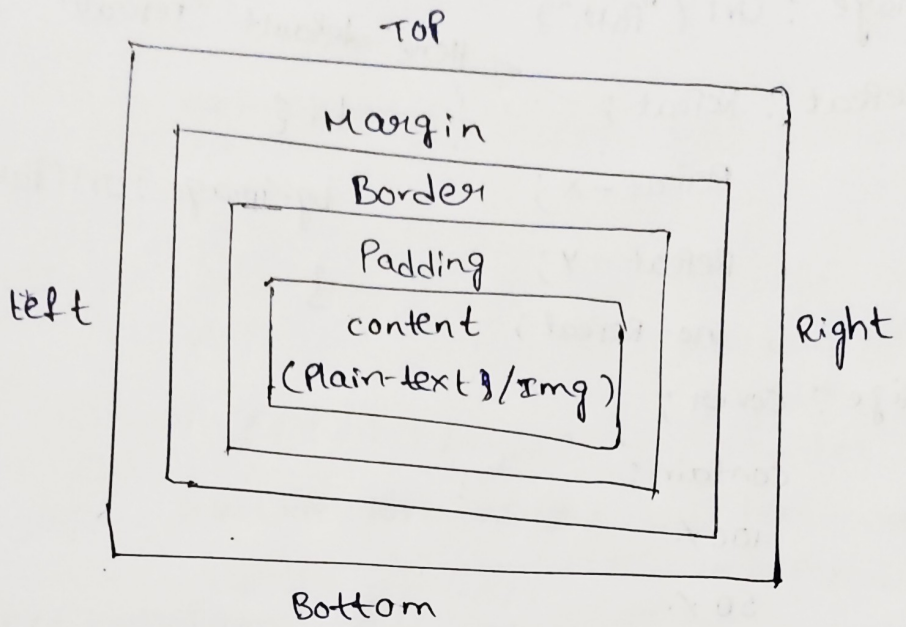
1. Padding (content)
2. Border
3. Margin

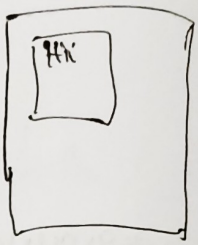
→ It is used to allign the content / elements.

Padding :- It will Provides space b/w content and border.

Border :- It Provides space b/w Padding & margin.

Margin :- It Provides space b/w border & web Page.





1) Padding : 100Px;
Padding-TOP : 150Px;
Padding - Right : 200Px;
Padding - Bottom : 250Px;
Padding - left : 300Px

Padding : 50Px 100Px 150Px 200Px
TOP R B L
Padding : 100Px 100Px
T-B L-R

2) border : 5Px solid red;
border-top : 10Px dotted Green;
border-Right : 15Px dashed crimson;
border-Bottom : 20Px doubled Yellow;
border-left : 25Px groove Blue;

3) Margin : 50Px;
Margin-top : 100Px;
Margin-right : 150Px;
Margin-bottom : 200Px;
Margin-left : 250Px;

Margin : 100Px 150Px 200Px 250Px
T R B L
Margin : 50Px 100Px
T-B L-R

Note:- user agent style sheet is the default style sheet of the browser.

→ 8Px is the default moorgin size.

Position - Property

- ✓ static Position
- ✓ fixed Position
- ✓ Relative Position
- ✓ sticky Position
- ✓ Absolute Position

1) Static Position:- It is a Default Position of Browser.

⇒ It doesn't accept top, right, bottom and left Property.

Ex:-
```
<div id='demo'>

</div>
```

```
#demo{
border:2Px solid;
Height : 200Px;
width : 200Px;
Position : static;
}
```

2) Relative Position:-

⇒ It is a Normal Position, which accepts top, Right, Bottom, left Property.

Note:- One disadvantage for relative Position i.e it Provides white space.

Ex:-
```
<Div id='demo'>
</Div>
<Div id='demo1'>
</Div>
```

```
#demo{
height : 200Px;
width : 200Px;
border : 2Px solid;
Position : Relative
TOP : 100Px;
left : 200Px;
}
```

```
#demo1{
border : 200Px solid;
height : 200Px;
width : 200Px;
```

3) Absolute Position:- It is a Parent to the realative Position.

=> It is consume white space Provided by relative Position.

=> It will also accepts top, right, bottom, left Property.

Ex:-

```
#demo1 {
    border: 200Px solid;
    height: 100Px
    width: 100Px
    Position: absolute
    TOP : 100Px
    left : 100Px
}
```

4) fixed Position:- Particular element fixed on the web-Page.

=> It will accepts top, right, bottom, left.

Ex:-

```
<div> id='demo'
</div>
```

```
#demo{
    border: 2Px solid;
    height: 200Px;
    width: 200Px;
    Position : fixed;
    TOP : 200Px;
}
```

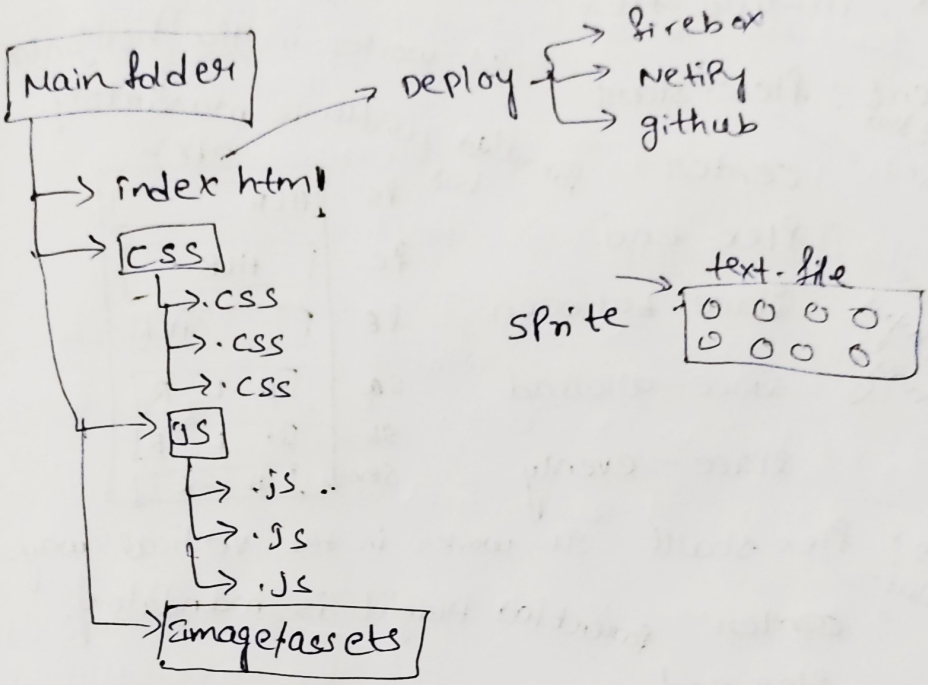
5) sticky Position:- It is a combination of fixed and relative Position.

=> whenever we take Position as a sticky top should be zero.

ex:- id=demo

#demo{
border : 2px solid;
height : 200px;
width : 100%;
position : sticky;
top : 0;
}

web-Development Project Outlook :-



z-index :-

⇒ It is use to overlap one element on the another element.

⇒ z-index will accept both Positive and negative value but default value of z-index is auto.

Opacity :-

⇒ It is used to Provide blurness for an content (or) image.

⇒ It will accept value from 0 to 1.

⇒ 0 means full Opacity. (It means can't see the content (or) image).

→ Default Opacity is 1. And 1 means no Opacity.

Flex-Box-Model :-

→ CSS layout Model.

→ Responsive web-Page.

→ To avoid using Position, float Property.

Note :- flex-Property should be given for Parent Element.

Display : flex, in-line flex

justify-content : flex-start → It works in the horizontal way.

↓ for this use display flex is mandatory. It is default one.

center

flex-end

space-between

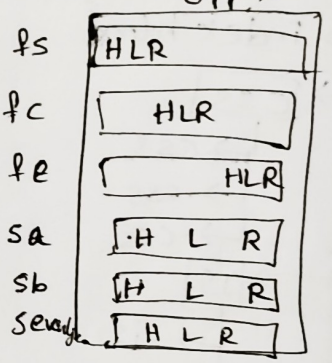
space-around

space-evenly

for this also display:flex is mandatory.

Parent has width is mandatory.

o/p :-



align-items : flex start → It works in the vertical way.

It is a default value.

center Parent has Height is mandatory.

flex end

Ex :-

```
<ol>
<li> Home </li>
<li> login </li>
<li> Reg </li>
</ol>
```

```
ol{
border : 2px solid ;
Padding : 0% .
list-style : none ;
Display : flex .
Display : in-line flex ;
width : 100% .
}
```

ol {

border : 2px solid;

Padding: 0%

list-style : none;

Display : flex;

justify-content : flex start;

Height : 700px;

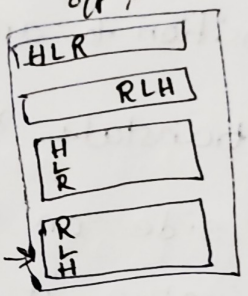
align-items : flex-start;

: center;

: flex-end;

}

o/p :-



flex-direction : row.

it is default

row-reverse

Column

Column reverse

flex-wrap : no-wrap

it is default

wrap

wrap reverse

align-content : flex-started

it is default

center

flex-end

Ex:-

 Home

 Login

 Reg

 1

 2

 3

ol { border : 2px solid;

Display : flex;

Padding: 0%

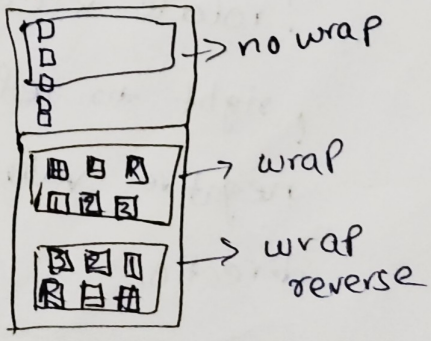
Height : 300px;

flex wrap : no-wrap;

: wrap;

: wrap-reverse;

o/p :-



=> Align-content works similar like align-items.

=> It also works is vertical way & height is mandatory.

=> The difference is whenever using align-content flex-wrap is mandatory. otherwise align-content not goona work.

2D Transition and Transform ;-> changes

Transform : translate (x-axis, y-axis) // Transform Property only used in hover.
Pixels Pixels

: It is used to move element from one Position to another Pasition.

: translate, x (Pixels) -> It move right side on x-axis. If you give minus value it will move left side.

: translate Y (Pixels) -> It move bottom side on Y-axis. If you give minus (or) negative value it will move top side

transform ; rotate (degree)

: It is used to rotate an element either in a clock-wise or Anti-clock wise.

; rotate x (deg) -> It rotates on x-axis right to left in clock-wise. If give negative values rotates anti-clockwise direction.

: rotate Y (deg) → It rotates on Y-axis top to bottom in clock-wise. If give negative values rotates anti-clockwise direction.

```
ex:  <Div> → id="demo"
        <Div> → id="demo1"
        </Div>
     </Div>

#demo{
    border : 2px solid;
    Height : 300px;
    width : 300px;

#demo:Hover{
    transform : translate (100px, 100px);
    transform : rotate (60 deg)
    }

#demo1 : Hover{
    transform : scale (5,5).
    transform : skew (8deg, 8deg)
    }
```

transform : scale (value , value)
x-axis Y-axis

→ It is used to increase / decrease size of an element.

: scale X (value)
: scale Y (value)

transform : skew (deg, deg)
x-axis y-axis

→ It is used to provide skewness for an element.

: skew X (deg)

: skew Y (deg)

transform : matrix (skew X, scale x, scale Y,
skewYx, translate x, translate Y).

→ By using this we can translate all ~~values~~ operations at a time.

transition :- for this we have to give Property name and duration.

transition - Duration : sec/ms

transition - Delay : sec/ms

Ex:- → id = 'demo'

#demo{
bg-color : Red;
border : 2px solid;
Height : 200px;
width : 200px;
}

#demo : hover{
/* transition-duration ; 5 sec */

```
/* transition : height 5 sec, width 5 sec; */
    Height : 400Px;
    width : 400 Px ;
    bg-color : Yellow;
transition-Delay : 5 sec ;
}
```

transition-Property; Here we mentioned Property-name. we can use this Property with the help of transition -duration (or) transition-delay.

→ It will not work Properly without transition-delay (or) transition-duration.

→ Previous #demo is also there here.

Ex:

```
#demo :hover {
    transition -duration : 5 sec;
        height : 400 Px ;
        width : 400 Px ;
        bg-color : Yellow;
    transition-Property : height width;
```

transition-timing-function : linear (Normal speed) all time
: ease (start slow Middle end slow)
: ease-In (start slow Middle end slow)
: ease-out (s M endslow)
: ease-In-out (start slow M endslow).

Ex:- id = demo

#demo {

bg-color : Red;

border : 2px sdid;

width : 200px;

}

#demo : hover {

transition-duration : 5 sec;

width : 1000px;

transition-timing-function : linear

Animation- Property :-

→ @key frames

⇒ animation-Duration :

⇒ animation-Delay :

⇒ animation-timing-function : linear, ease, ease-In, ease-out, ease-In-out

⇒ animation-iteration-count : infinite (or) we can give values also.

⇒ animation-fill-mode : backward
: forward
: both

Ex:- id = "demo"

#demo {

border : 2px solid;

```
{Height : 200Px
 width : 200Px

[animation-name : demo1]
 animation-duration : 5s;

@keyframes demo1{
    from{
        bg-color- Red;
    }
    TO{
        bg-color- Yellow;
    }
}
```

(or)

```
@keyframes demo1{
    0%{
        bg-color: Red;
    }
    25%{
        bg-color: blue;
    }
    50%{
        bg-color: crimson;
    }
    100%{
        bg-color: Yellow;
    }
}
```

Color-Property :- there are 6 ways.

1. color : color-name (Red)
2. color : RGB (0Red Green Blue)
   0-255 0-255 0-255
3. color : Hexadecimal
   : #RR GG BB
   : #00 00 00 → Black
   : #ff ff ff → while
   (or)
   : ~~000~~ fff
4. color : RGBA (0.Red, Green, blue, Alpha)
   It is like opacity
   0-255, 0-255, 0-255, 0-1

5. Color : HSL (Hue, Saturation, lightness).

↓ ↓ ↓

deg Greyness 0% - 100%

↓

0% - 100%

0 - 125 deg → Red

125 - 255 deg → Green

255 - 360 deg → Blue

6. Color : HSLA (Hue, Saturation, lightness, Alpha)

↓ ↓ ↓ ↓

Degree, %, %, 0 - 1

Background - color : 6-ways. same as above.

Gradients in CSS :-

→ It Provides smooth transition blw two (or) more colors.

3-types :-

1. linear - gradient :- It will travels in a straight line.

syntax :-

Background - Image : linear - gradient (direction, start-color, end color)

(or)

background ↓

: linear - gradient (to right, red, blue, green)

: " (to left)

: " (to bottom)

: " (to top)

↙

It is default one

⇒ Here we can Provide n number of colors.

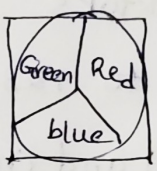
2. Radial-gradient :- It starts from center.

syntax :-

background-image : Radial-gradient (shape size at position)

shape: ellipse (it is default one), circle
size: farthest-corner, closet-corner, farthest-side, closest-side

3. conic-gradient :- It starts from circle.



syntax :-

background-image : conic-gradient (Green, Yellow, blue)

CSS- units :-

→ rem (root element) → 1rem = 16 Pixels

→ em (element) → 1em = 1% (svg, line-height)

→ Px (Pixels) → (font size, top, left, bottom, right)

→ % (Percentage) → (height, width)

(Here 1% means 1px) when we want to use more than 1000px. We can go for Percentage.

→ vh (viewPort height) → 1vh = 1%

→ vw (viewPort width) → 1vw = 1%

Media Queries :- Grid :- It is used in the form of rows & colums. v.2 grid-template rows, grid template columns and gap Property. Auto is default value. column-gap & row-gap } 2 more Properties in grid.

Media Queries:

Media Queries are a key part of responsive web design, as they allow you to create different layouts depending on the size of the viewport.

① ## Device = Desktops (2R resolution desktops)
Screen = 1281 Px to higher resolution desktops.

@ media (min-width : 1281 Px) {

② ## Device = Laptops, Desktops
Screen = B/w 1025 Px to 1280 Px

@ media (min-width : 1025 Px) and (max-width : 1280 Px)

③ ## Device = Tablets, IPads (Portrait)
Screen = B/w 768 Px to 1024 Px

@ media (min-width : 768 Px) and (max-width : 1024 Px)

④ ## Device = Tablets, IPads (landscape)
Screen = B/w 768 Px to 1024 Px

@ media (min-width : 768 Px) and (max-width : 1024 Px) and (orientation : landscape)

⑤ ## Device = Low Resolution Tablets, Mobiles (Landscape)
Screen = B/w 481 Px to 767 Px

@ media (min-width : 481 Px) and (max-width : 767 Px)

⑥ ## Device = Most of the smartphones Mobiles (Portrait)
Screen = B/w 320 Px to 479 Px

@ media (min-width : 320 Px) and (max-width : 480 Px)

where Property :-

CSS
```
where (h1, p) {
    color : red ;
}
```

html
```
<div>
    <h1> thi </h1>
    <h1> Hello </h1>
    <p> Good </p>
```