CSS

↓

cascading style sheet

→ use to describe the layout of the page and style of page

↓

every Html has own default style

→ using CSS, we can override that style.

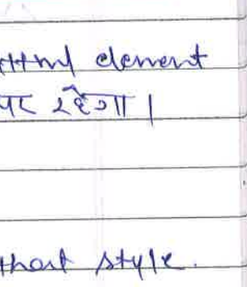
→ current version of CSS is 3 (CSS3)

→ 3 ways of using CSS →

- External CSS
- Internal CSS
- Inline CSS

* External CSS :- Extension of CSS file is ".css"

To use this CSS file in our Html




```
<html>
  <head>
    <link rel = "stylesheet"
          type = "text/css"
          href = "myStyle.css">
  </head>
</html>
```

⇒ Advantages :-

(1) Separation of concern → Html & CSS should be different.

(2) Reusability

→ Internal CSS :-

```
<html>
  <head>
    <style>
      h1 {
        color : blue;
        margin-left : 40px;
      }
    </style>
  </head>
  <body>
    =
    =
  </body>
</html>
```

In case of internal css, we place the css in style tag under head element

→ Inline css :-

```
<h1 style = "color : red ; background-color : "blue">
    This is h1 tag.
</h1>
```

Style = " prop1 : value1 ; prop 2 : value 2 "

* Inline css has highest priority.

⇒ CSS Selectors :-

- id selector
- class selector
- pseudo class selector
- attribute selector
- tag selector
- pseudo element selector
- All selector (*)

→ Cascading :-

- multiple styles can be applied on an element.

* If we applying multiple styles on a element, so which one will get priority :-

suppose, I have a 'h1' element

on this 'h1' tag Inline css लगे
Internal css लगे
external css लगे
multiple css rules हैं
using different selectors

अगर css external file में है
+
Internal css है
||
दोनों combine होकर एक css file है।

priority →

- Inline css → 1000
- id selector → 100
- class, pseudo class, attribute → 10
- tag, pseudo element → 1
- *, pseudo class : not → 0
- Inheritance
- default

whichever has greater calculation that will be applied

→ So, In order to know which css will be applied - you have to calculate specificity.

→ If you have - 2 different rules with same specificity then the rule that comes at last will be applied.

→ combinators does not play any role in specificity calculation.

High priority

Style
Date:
Page No:

→ Targeted → इस उस element को target कर रहे हैं।
↓ using selectors

→ Inherited

→ default
low priority

In css, a child element inherit css from parent.

Ex.

```
<body>
  <div>
    <p>
      <h1> ——— </h1>
    </p>
  </div>
</body>
```

here, h1 will inherit the css from body, div, p.

Ex2. <body> → color: red, background-color: blue

 → color: blue

<h1> ——— </h1>

 ↓ here h1 tag -

 • color: blue inherit करेगा p tag से
</body> • background-color: blue inherit करेगा body tag से

Immediate Parent

~~Pseudo classes~~:-

→ Selecting multiple elements:-

```
p, h1, li {
    color: Red
}
```

→ एक element में multiple classes हो सकते हैं।
<h1 class = "class1 class2"> This is ... </h1>

→ we can combine multiple selectors to target a particular element -

a.active
⇓
select anchor element having active class

→	class	id (unique)
	target multiple element	single element

!important
⇓
It will override the specificity

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

:not (p) {
    ...
}

<q> — <q> <p> — <p>

→ Select every element that is not p.

Style
Date:
Page No:

:not ( )
⇓
pseudo class

p will not be reflected
:not (p)
⇑
a:not(.active) ⇒ :not (selector)
⇓
select anchor tag that doesnot have active class on it

→ element selector :-

```
p {
    ...
}
```

select <p>

→ class - selector → Select the element with the specified class

.my-class
selects
<p class = "my-class"> and
<a class = "my-class">

→ id - selector

#my-id
Selects
<p id = "my-id">
or
<a id = "my-id">

Syntax:
① selector [attribute]
② selector [attribute = "value"]

*= → select or contain value
!= → select start with value
^= →
$=

Attribute selector :-

Selects element on the page with the specified attribute.

Ex-1.
```
img [src] {
    - -
}
```
Selects <img src = "myimg.png" > but not <img>

Ex-2.
① —— disney.com
② —— ——
③ —— ——

Style

applied to ② <a>
```
a [target] {
    background-color : yellow;
}
```
→ more specific to ① <a>
```
a [target = "_blank" ] {
    background-color : red;
}
```
applied to ① <a>

this style will be applicable to ① <a> tag because this is more specific.

③ <a> tag ~~ने~~ ~~कोई~~ style apply ~~नहीं~~ ~~होगा~~ ~~क्योंकि~~ ~~उसमें~~ ~~पर~~ ~~कोई~~ target attribute ~~नहीं~~ ~~है~~

Pseudo-class names are not case-sensitive

Style
Date:
Page No:

Pseudo-class selector :-

→ selects specified elements, but only when in the specified state, e.g. being hovered over.

```
a:hover {
    color : Red;
}
```
Selects <a>, but only when the mouse pointer is hovering over the link.

Syntax :-
```
element
   ↑
selector : pseudo-class {
    property : value
}
```

→ pseudo-classes let us apply a style to an element not only in relation to the content in Dom tree -
but also to external factors like the
- history of the navigator (:visited)
- the status of its content
  (like : checked)

or the position of the mouse
(like : hover, which lets you know if the mouse is over an element or not).

→ There are lot of pseudo classes.
→ Some important are here,

pseudo-classes and css classes:-

```
a.highlight:hover {
    color: Red;
}
```

```
<a class = "highlight"> ——— </a> —(1)
<a href"——"> ——— </a> —(2)
```

This style will be applied to <a> tag
(1) because this tag is anchor tag
as well as has "highlight" class also
but (2) <a> tag doesnot have highlight style.

Some imp pseudo classes:-

:empty - Selects elements which contain no text and no child elements like.

→ Anchor pseudo-classes:-

a:link { } // unvisited link

a:visited { } // visited link

a:hover { } // mouse-over

a:active { } // selected link

Some imp pseudo classes:
- :checked
- :disabled
- :enabled
- :focus
- :invalid → selects all <input> with an invalid value
- :target
- :required - select inputs with the required attribute
- :optional - ——— that not have ———

Positions.

* we can change the position of any Html element using property

* ~~position~~ → default for all html element

position: static

* position: fixed

① → Take the element out of the document flow.

⇓

समझेंगे कि this element is not here

⇓

and other element can take its position

② positional context will be viewport

⇓

webpage का वह हिस्सा जो हमेशा visible है।

③ we can adjust the position of this element relative to viewport by using top, bottom, left & right.

⇓

these 4 are not applicable to static

top: 0px
left: 0px } ⇒ element will be fixed at top-left corner.

④ Block / Inline element will become Inline-block element ✓

position : absolute

① Takes the element out of the document flow
⇓
दूसरों के लिए element वहाँ पर exist ही नहीं।
⇓
other element can take its position

② Block / Inline element will become Inline-block element ✓

③ suppose there is an element

This element has no Ancestor with position property as fixed, absolute or relative.	This element has Ancestor with position property as fixed, absolute or relative
⇓	⇓
positional context will be html page itself	positional context will be the nearest Ancestor with property as fixed, absolute or relative ✓
⇓	
<html>	
⇓	
You can use top, bottom, left, right	

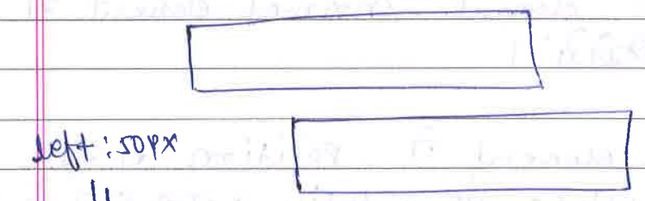
position : relative

Style
Date:
Page No:

* element is not taken out of document flow.
⇓
No other element will allow to take its place

* Inline, ~~inline~~ inline ही रहेगा
* block level, block level ही रहेगा

* positional context is the element itself.



⇓
element will move right by 50px from its actual position.

z-index

→ It is used to position an element along z-axis.

+ve z-index — ऊपर

0

-ve z-index — नीचे

* जब हम किसी element के position fixed, absolute या relative लगाते हैं तो वे ~~element~~ element normal element के ऊपर दिखेंगे।

** उपर 2 element के position fixed, absolute या relative लगा हुआ है, तो जो element बाद में होगा वह ऊपर दिखेगा।

* position: static → z-index doesnot work

default — z-index : auto ⇓ 0

Display property:

block → Inline
→ Inline-block
→ none

block-level → Takes full width

Inline → Doesnot take full width

↳ <a> tag, , <img>

Inline — display: block; → block

Inline → none.

display: none

display: none; // remove the element, other element can take place

display: block; → Inline to block

display: inline-block;

visibility: hidden → // element will be the same place but other element can not take place, still holds the space

Ex of block level elements:-

→ div, <h1>-<h6>, <p>, <form>, <header>, <footer>, <section>

Display: none → commonly used with JS to

* hide & show elements without deleting and recreating them.

* The element will be hidden and the page will be displayed as if the element is not there

→ Visibility: hidden → also hide an element

* but, the element will still take-up the same space as before.

display: none	Visibility: hidden
• element will be in Dom But taken out of document flow ⇓ यहाँ पर दूसरे element आ जाएँगे।	• Element will not be taken out of document flow ⇓ यहाँ पर दूसरे element नहीं आएँगे ⇓ सिर्फ element वहीं दिखेगा, लेकिन अपना जगह वहीं पर होगा

* Suppose, we have an ~~line~~ inline element & we are using display: block

→ so on screen it would be shown as block element

→ but in reality it is still an inline element

→ so it is still not allowed to have other block element inside it.

Inline
- → will take only require width
- → will not start in a new line
- → cannot set height, width
  - margin top, padding top
  - margin bottom, padding bottom

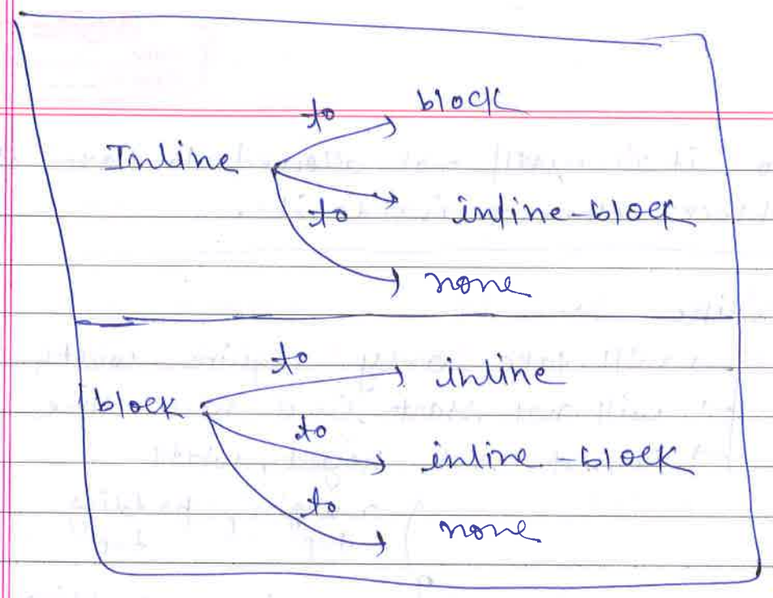
Block
- → will take full width
- → will start in a new line
- → can set height, width, margin-top, padding-top, margin-bottom, padding-bottom

Inline-block

Inline →
- → will take only require width
- → will not start in a new line

Block →
- → can set height, width, margin-top, padding-top, margin-bottom, padding-bottom

so, it is called → inline-block



Size & units

Style
Date:
Page No:

(1) **Absolute length** – It is independent of user setting.

px

cm, mm → Not used generally

```
<h1 style = "font-size : 20px" > This is h1 tag </h1>
```

↓

Its size will be 20px and it will not change on changing browser setting.

(2) **Viewport length**:- It depends on current viewport.

vh
vw
vmin
vmax

(3) **Font relative lengths**:- It depends on user setting.

em
rem

(4) percentage (%) →

Percentage

① In case of fixed positioning, containing block is viewport.

so, 10% width of element
⇓
10% of viewport

---

② In case of Absolute positining, Containing block is either <html> tag
⇓
webpage

or

containing block is ancestor with position as absolute or relative or fixed.

⇒ 10% of width
⇓
10% of nearest ancestor content that has position as absolute or relative/absolute
+
padding of this ancestor

---

③ In case of relative/static positioning :-

Containing element is a block level element that is the nearest ancestor
{ 10% of width }
⇓
10% of nearest ancestor block element's content

---

* If we are not defining font-size for a text, then this text will take size from user setting in browser

* If we provide the font-size of an element in px, then it

Working with em and rem

→ In browser setting, we can change the font-size.

By default it is Medium ⇒ 16 px

→ This settings change has no impact if we are using px.

→
```
<div style = "font-size : 2 rem">
    <h1 style = "font-size : 3 rem">
        This is h1 tag.
        <span style = "font-size : 2 rem">
            Hello </span>
    </h1>
</div>
```

Suppose in browser setting, if it 16 px
then 2 rem ⇒ 2*16 px ⇒ 32 px
3 rem ⇒ 3*16 px ⇒ 48 px

This is h1 tag Hello

⇒ So if we change the browser settings, rem value will change accordingly.

⇒ So rem is considered as relative to font-size of root element

⇓

font-size of <html> element

⇓

font size in browser setting

⇓

relative to font size in browser setting

em :—

① —
<h1 style = "font-size : 3 em "> ②
This is h1 tag
 ③
Hello
</h1>

→ suppose in browser setting font-size = 12 px

so in ①, font-size = 2em ⇒ 2 * 12 px
= 24 px

in ②, font-size = 3em ⇒ 3 * 24 px
= 72 px

in ③, font-size = 2em ⇒ 2 * 72 px
= 144 px

→ so em is considered as relative to font-size of parent.

→ So due to inheritance, its behaviour is difficult.

Working with vw & vh

↓ viewport width — ↓ viewport height

⇒ viewport → part of the webpage that is visible to the user.

If viewport width is 1000 px

then 1 vw = 1 % of viewport width

⇓

1 vw = $\frac{1}{100}$ * 1000 px = 10 px

<h1 style = "font-size : 2vw "> This is h1 tag
</h1>

⇓

20 px

So, as we adjust the viewport width, font size will change accordingly.

⇒ vh is similar to vw.

1 vh = 1 % of viewport height

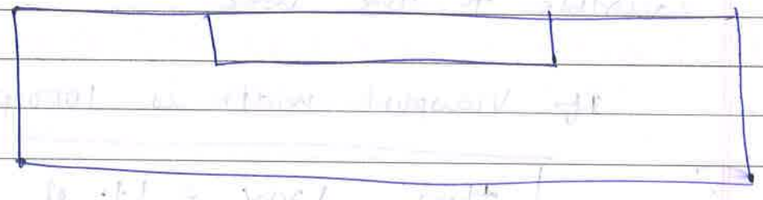
using auto to center element

↓

margin : auto

① This property is used to horizontally center an element.

② 'auto' value works only on the block elements that have explicitly set width. otherwise it will take complete available width.



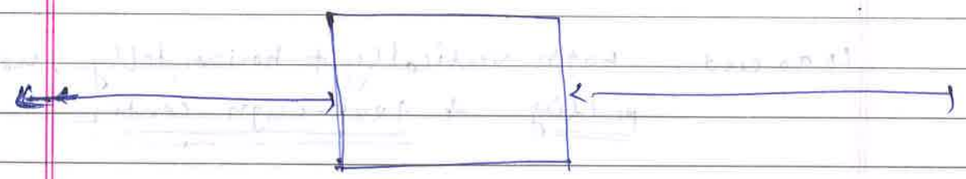
```
<div class = "div1"> <
  <div class = "div2" > </div>
</div>

.div1 {
    border : 2px solid green;
    height : 200px;
}

.div2 {
    border : 2px solid blue;
    height : 20%;
    width : 50%
    margin : auto
}
```

⇒ center-align Text → text-align : center;

This text is centered

```
.center {
    text-align : center;
    border : 3px solid green;
}
```

⇒ Center an Image :-



↳ set left & right margin to auto

↳ make it into a block element :-

```
img {
    display : block;
    margin-left : auto;
    margin-right : auto;
    width : 40%;
}
```

⇒ left-right align - using position

can do with position: absolute

⇒ Left & Right align → using float

⇒ Center vertically - using Padding

↳ There are many ways in CSS. One is to use padding (top, bottom)

↳ To center both vertically & horizontally, use padding & text-align-center

Sass

Syntatically Awesome Stylesheet

Sass —compiled→ css

Sass	css
does not run in browser	run in browser

Sass
- more readable
- Less code
- Easily maintainable

Sass has 2 syntax :-

.sass	.scss
• Old syntax	• New syntax
• Syntax without semicolon & curly-braces	• Syntax similar to css
• Based on indentation	• Every valid css code is also a valid scss code

Style
Date:
Page No:

Sass to css compilation :-

1. npm init
2. npm install node-sass
3. package.json

```
{
  script: {
    'test': ...
    'sasscompile': 'node-sass scssfolder cssfolder'
  }
}
```

- Any Name
- package
- folder in which scss file is there
- folder in which css file will be stored after compilation

→ Features of SASS

- Nested selector
- Nested properties
- Variables
  - → variables ~~start~~ storing value
  - → variables storing list.
  - → variables storing map
- built in functions
- Arithmatic
- Partials
- Inheritance
- mixing

① Nested Selector

```
.container {
    width: 100%;
}

.container h1 {
    color: red;
}
```

↑ CSS

```
.container {
    width: 100%;
    h1 {
        color: red;
    }
}
```

↑ SCSS

So, using SCSS we can nest selectors like this

Nested Properties

```
h1 {
    font-size: 30px;
    font: weight: bold;
    font: style : italic;
    color : blue;
}
```

↓

So, we can nest property like this

```
h1 {
    font: {
        size: 30px;
        weight: bold;
        style : italic;
    }
    color: blue;
}
```

↑ SCSS

Variables

- Suppose we have a SCSS file and we are using color value as #FA9325 at many places within this file.

↓

So, in future if we decide to change the color value to #FBB273 then we will have to make changes at each place.

↓

So instead of this we will use Variable in SCSS.

$variablename : variable_value

Ex: $colorname : #FA9325; ⇒

```
h1 {
    color : $colorname;
}
```

Nested selector Example

```
<div class = " class1 class2 ">
    <p class = " class3 class4 ">
        This is paragraph tag
    </p>
    <span> This is span tag </span>
</div>
```

CSS

```
div.class1.class2 {
    color: red;
}

div.class1.class2 > .class3.class4 {
    color: blue;
}
```

Here we are trying to select an element that has class3 & class 4 and this element should be direct child of a div element that has class1 & class2.

SCSS

```
$colorname1 : red;
$colorname2 : blue;

div.class1.class2 {
    color: $colorname1;
    > .class3.class4 {
        color: $colorname2;
    }
}
```

storing list & map in a variable

CSS

```
h1 {
    border : 2px solid red;
}
```

In SCSS

```
$colorname: red;
$borderval : 2px solid $colorname;

h1 {
    border : $borderval;
}
```

This is like storing list in a variable

- Suppose we are using three different colors in our project's css file :-

so,

```
$colorname1 : red;
$colorname2 : blue;
$colorname3 : green;
```

```
h1{
    color: $colorname1;
}

h2{
    color: $colorname2;
}

h3{
    color: $colorname3;
}
```

so, instead of writing like this we can create a map

```
$colorname : (main: red, secondary: blue, tertiary: green)
```

↓

This is map

to extract from this map, we will use

map-get ($colorname, main)

↓

built-in function

we are targeting to extract 'main' from $colorname it will give 'red' value

```
h1{
    color: map-get ( $colorname, main );
}

h2{
    color: map-get ( $colorname, secondary );
}

h3{
    color: map-get ( $colorname, tertiary );
}
```

→ Built-in-functions :- map-get() is a built fn

↓

- use to extract values from a map
- Like map-get (), there are other ton of built in functions.

→ Doing simple arithmatic

```
$default-size : 1em;

h1{
    font-size : $default-size * 2;
}
```

11

2em

or

$default-size / 2

11

Brilliant

Partials :-

→ Before discussing about partials, we need to discuss about import —

- using @import we can include the content of one style-sheet to other style-sheet

  @ import "abc.css"

  or

  @ import url("abc.css")

* @ import should be placed at the top.

partials —

You can create a file like —

—anyname.scss ⟹

↓

- Here -(underscore) tells that this is a partial ⇓

we will use the content of this file into other file by using @import.

—anyname.scss will not be compeled to css.

* @ import "_anyname"

↳ .scss is not required

* ~~Not~~ Now only 1 http call will be there for main stylesheet.

Inheritance

Style
Date:
Page No:

→ using @ extend, you can share a set of css properties from one selector to another selector.

html

```
<h1> This is h1 tag </h1>
<h2> This is h2 tag </h2>
```

.scss

```
h1{
    font-size : 1em;
    color : red;
    border : 2px solid red;
}
h2 {
    font-size : 1em;
    color : red;
    background-color : blue;
}
```

any selector

```
.class {
    font-size : 1em;
    color : red;
}
h1 {
    @ extend .class;
    border : 2px solid red;
}
h2 {
    @ extend .class;
    background-color : blue;
}
```

mixin

A mixin lets you write the CSS rules that you may want to use throughout the site.

@mixin anyname (param) {
(param → If any)

```
prop1 : val1;
prop2 : val2;
prop3 : val3;
```

}

h1 {
@include anyname ();
}

* All the properties mentioned in the mixin will be placed inside h1

So, reusable codes can be placed in a mixin & this mixin can be used anywhere in the stylesheet using

@include mixinname ( ) ;

using the ampersand (&) operator

Style Date: Page No:

→ Ampersand (&) operator is used to replace the parent selector

Ex1.

```
h1 {
    color: blue;
}
&{
    color: red;
}
```

⇒

```
h1 {
    color: blue red;
}
h1 {
    color: red;
}
```

Ex2.

```
h1 {
    color: blue;
    & span {
        color: red;
    }
}
```

⇒

```
h1 {
    color: blue;
}
h1 span {
    color: red;
}
```

Ex3.

```
h1 {
    color: blue;
}
& span {
    color: red;
}
```

⇒

```
h1 {
    color: blue;
}
h1 span {
    color: red;
}
```