

Animations & Responsiveness - I

Advanced Grids, Media Queries & Breakpoints.

Grid-Area for layout creation

```
.item {
  grid-area: 1/2/3/3 ;
}
```

1 → grid-row-start.	(grs) ✓
2 → grid-column-start.	(gcs) ✓
3 → grid-row-end.	(gre) ✓
4 → grid-column-end.	(gce) ✓

Initialize Grid :-

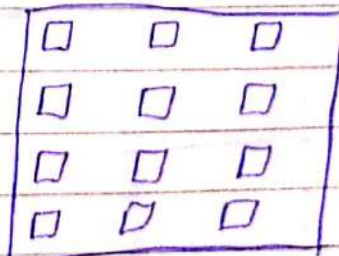
display: grid;

Creating Rows & Columns :-

```
grid-template-rows: repeat(4, 1fr);
grid-template-columns: repeat(3, 1fr);
```

Gap

grid-gap: 10px;



`/* width: 100px; */`

comment this width

box1 {

grid-column-start: 1;
grid-column-end: 4;

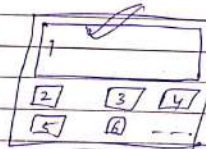
}



box2 {

grid-column-start: 1;
grid-column-end: 4;
grid-row-start: 1;
grid-row-end: 3;

}



~~for~~
~~***~~

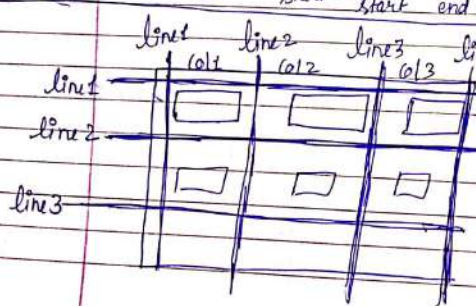
Short hand Notation :-
Span of Row & Col

Same behaviour

box1 {

grid-area: 1 / 1 / 3 / 4;

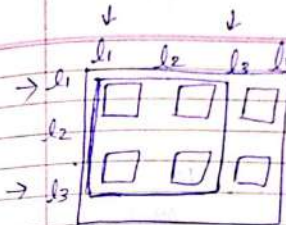
Row start Col start Row end Col end



These are the line number which we input into grid-area.

CLASSTIME Page No.
Date

CLASSTIME Page No.
Date

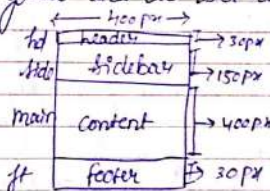


ges → 1
gee → 3
gse → 1
gie → 3

grid-area: 1 / 1 / 3 / 3;

Grid-template-area: (gta) ^{short cut}

It is the property used to name the rows and columns of a grid and to set its layout.



ex-1

display: grid;
grid-template-rows: 30px 150px 400px 30px;
grid-template-columns: 400px;

grid-template-areas: "hd"
"hdd"
"main"
"ft";

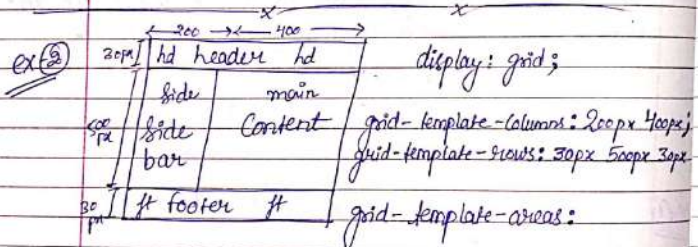
naming


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

```
#footer {
  grid-area: ft;
}
```

```
#sidebar {
  grid-area: side;
}
```

```
#content {
  grid-area: main;
}
```



```
#header { grid-area: hd; }
#footer { " ft; }
#sidebar { " side; }
#content { " main; }
```

"hd hd"

"side main"

"ft ft";

Note: 300px एका grid में 2 row defines करेंगे, और grid template area में width naming entry row में करेंगे, और grid में Rows Squeeze करेंगे -- की width और height (width)।

example:-

```
g.t.c: 300px 300px 300px;
g.t.r: 250px 60px;
```

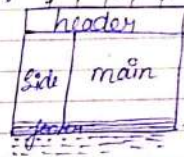
2 rows defined.

g.t.a: "hd hd hd hd hd hd"

"side side main main main main"

"ft ft ft ft ft ft"

3 rows here
Or can be more



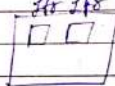
→ Squeeze है width।

Advanced Grid Concepts:-

* Fraction unit ✓

Fraction unit when we create equal size of rows and columns.

ex:- gtc: 1fr 1fr;



* Repeat function:-

gtc: repeat(2, 1fr);
repeat(110, 1fr);

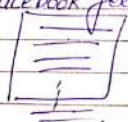
* Grid-auto-rows: minmax()

It adds automatically rows.

→ when we don't know number of rows.

→ Unknown no. of rows.

→ Then, we use Grid-auto-rows ex:- facebook feed.



→ minmax() → refers to size of row.

- minimum size (px). (width)
- maximum size (px). (width)

* Grid-auto-columns: minmax(a, b);

min width max width

It sets size of each column depends on the min width & max width (max).

Grid Properties

justify-content

align-content

justify-items

align-items

justify-self

align-self

place-items

place-self

→ Parent container use justify-content

→ child use align-items (items)

(a) justify-content: start;

: end;

: center;

: space-between;

: space-around;

: space-evenly;

(Main axis horizontal)
(Alignment vertical)

Cross-axis or Vertical axis (or) alignment or Handle Spring b/w Boxes

(b) align-content: space-between;
 space-around;
 space-start;

(c) justify-items: start; [] (first children or container property)
 : end; []
 : center; []

(d) align-items: start; Content → box or start
 end; Vertically alignment on box
 center; []
 stretch; (default)
 baseline; []

(e) justify-self: start; [] (particular box)
 : end; []
 : center; []
 justify-self items or start of box or start horizontally item or alignment on first

(f) align-self: start; []
 : end; []
 : center; []
 align-self all items or start of box or start vertically item or align on first

place-items / place-content first container or start

(g) place-items: shorthand property for justify-items & align-items.
 place-items: center;

[E] It aligns vertically & horizontally both aligned
 Box/items or center →

(h) place-content: shorthand property for justify-content and align-content.
 place-content: center;

Container or center or align or Box/items

Media Queries:-

- "Viewport" :- the area of the window in which web content can be seen.
- We use the dimensions of the viewport (width, height) as the basis of our media queries.
- We use breakpoints to set the condition of a media query.
- The logic is: $@media (\text{feature} : \text{value})$.
- Media queries are used to set different style rules for different devices or sized screens.

Ex: ① $@media (\text{max-width} : 400px) \{$
 .container {
 grid-gap: 50px;
 }

Multiple Breakpoints

② $@media (\text{min-width} : 300px) \text{ and } (\text{max-width} : 500px) \{$
 #box-2 {
 display: block;
 background-color: red;
 }

Nested Grid:-

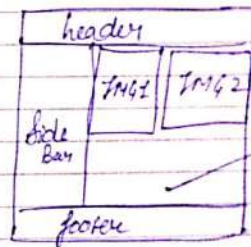
Nested Grid is simple and can be done by using the display: grid rule for both a parent and child element.
Grid inside grid.

```
<div class="container">  
  <div class="box" id="header">Header </div>  
  <div class="box" id="side">Side Bar </div>  
  <div class="box" id="main">  
    <div class="childgrid">  
      <div class="element">IMG1 </div>  
      <div class="element">IMG2 </div>  
    </div>  
  </div>
```

```
<div class="box" id="footer">  
</div>
```

```
•childgrid {  
  height: 100%;  
  width: 100%;  
  display: grid;  
  grid-template-columns: 1fr 1fr;  
  grid-template-rows: 1fr 1fr;  
  grid-gap: 5px;  
}
```

```
•element {  
  border: 2px solid orange;  
  background-color: lightblue;  
}
```



Animations & Responsiveness - II

CSS Functions, Variables, Transitions and Animations.

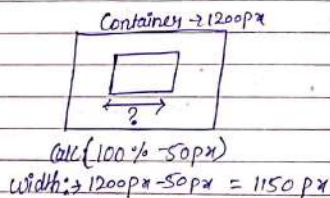
CSS Functions

It invokes special data processing or calculations to return a CSS value for a CSS property.

→ Math Functions

* calc() :- performs basic calculations Arithmetic or numerical values.

ex:- `width: calc(100% - 50px);`



* min() :- calculates the smallest of a list of values.

ex:- `min(50%, 400px)`

`Height: min(50%, 400px);`

* max() :- calculates the largest of a list of values.

ex:- `width: max(60%, 120px);`

* minmax() :- used with grid-auto-rows in grid when unknown no. of rows.

ex:- ~~height~~
`grid-auto-rows: minmax(100px, 400px);`

→ Trigonometric functions :-

* `sin()` * `asin()`
* `cos()` * `acos()`
* `tan()` * `atan()`

→ Exponential functions :-

* `pow()` * `sqrt()`
* `log()` * `exp()`

→ Filter functions :-

`blur()`
`brightness()`
`contrast()`
`drop-shadow()`
`grayscale()`
`opacity()`
`invert()`

→ Color functions:-

rgb(), hsl(),
rgba()

→ Gradient functions:-

* linear-gradient()
* radial-gradient()
* conic-gradient()

→ Grid function:-

* minmax()
* repeat()

→ Transition functions:-

→ Transform functions:-

translate(),
translateX(),
translateY(),
translate3D(),
translateZ()

→ Rotation functions:-

rotateX(), rotate(),
rotateY(), rotate3D(),
rotateZ()

→ Scaling functions:-

scale(),
scaleX(),
scaleY()

→ Skew functions:-

skewX(),
skewY(),
skew()

→ Perspective function

perspective()

CSS Variables

The var() function is used to insert the value of a CSS variable.

A good way to use CSS variables is when it comes to the colors of your design. Instead of copy and paste the same colors over and over again, you can place them in variables.

Syntax of the var () function.

var(--name, value)

-- By using double dashed, we can access.

example:-

(1)

```
:root {
  --darkRed: #981a2c;
}
```

Global Scope

We can access in container and child item both

. container {

```
  border: 1px solid black;
  height: 300px;
  width: 300px;
  background-color: var(--darkRed);
}
```

. box {

```
  border: 2px solid green;
  background-color: aqua;
```

```
  color: var(--darkRed);
}
```

(2)

. box {

```
  border: 2px solid green;
  background-color: aqua;
```

```
  --darkRed: #981a2c;
  color: var(--darkRed);
}
```

Local Scope

We can only access it inside child items only.

Variables

Local Variable:-

If we create inside div/class/id, then we can access that variable only locally inside div/class/id.

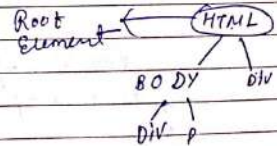
Global Variable:-

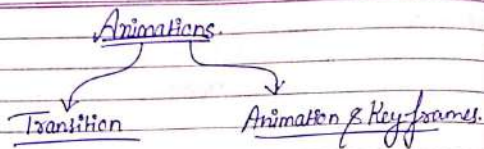
We can access it globally to both parent and child.

```
:root {
  --varname: value;
}
```

Accessing → var(--varname);

Note:- :root represents to Root Element in a DOM





Transitions in CSS :-

CSS transitions allow you to change property values smoothly, over a given duration.

- transition - property.
- transition - duration.
- transition - timing - function.
- transition - delay.

→ transition. (Shorthand Property for above all).

* transition - property :- At which property, transition effect will apply.

example:-

```

.box {
  background-color: aqua;

```

```

  transition-property: background-color;
  transition-property: width;
}

```



```

.box: hover {
  background-color: brown;
  width: 300px;
}

```

Note:- transition-property: all; It is a bad practice. It consumes more time to effect.

* transition - duration :- how much time, the transition will run.

example:-

```

.box {
  transition: width 2s;
}

```

```

.box: hover {
  width: 300px;
}

```

Increasing width of box on hover with duration of 2 seconds.

Note:-

We can give like:-

```

transition-property: width;
transition-duration: 2s;

```


* transition-timing-function :-
Specifies the speed of the transition effect.

Syntax:- transition-timing-function: values;

Values can be -

- (i) ease → slow start, then fast, end slowly.
↳ (default)
- (ii) linear → Same speed from start to end.
- (iii) ease-in → a slow start.
- (iv) ease-out → a slow end.
- (v) ease-in-out → a slow start and end.

*** (vi) cubic-bezier(n,n,n,n) → (Right click + Inspect + Cubic-Bezier)

Example:-

```

    .box {
      transition: background-color 2s ease;
    }
    .box :hover {
      background-color: brown;
    }
  
```

↑ property
 transition: background-color 2s ease;
 ↓ duration ↓ timing function

* transition-delay :- Specifies a delay for the transition effect.
After how much time delay, transition start.

ex:- transition-delay: 1s;

* Shortcut property :-


transition: transform 5s ease 1s;

↓ ↓ ↓ ↓
 property duration timing function delay

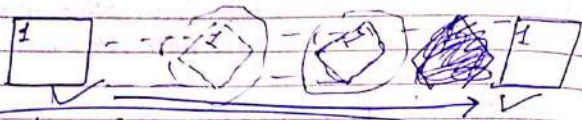
```

    .box: hover {
      transform: rotate(45deg);
    }
  
```

* If we want to create own transition (Right click + Inspect + Cubic-Bezier) :-
Cubic-bezier :-
transition: transform 3s cubic-bezier(1, -0.09, 0, 1.68);

Maza aa Agga 

Box Rotation in along X direction



```

• box: hover {
  transform: translateX(500px) rotate(360deg);
}

```

With grow Scale

```

• box: hover {
  transform: scale(2) translateX(500px)
            rotate(360deg);
}

```



Animation & Keyframes :-

properties:-

- @keyframes.
- animation-name. → To give name of Animation
- animation-duration. → duration
- animation-delay. → delay time
- animation-iteration-count. → No. of times of animation / Infinite
- animation-direction. → Normal / Reverse / Alternate etc.
- animation-timing-function. → ease / ease-in-out / ease-out / etc.
- animation-fill-mode. →

Shorthand property for above all.

→ animation.

CSS Animations

Gradually change from one style to another.

To use CSS animation, you must first specify some keyframes for the animation.

Keyframes hold what styles to element will have at certain times.

The @keyframes Rule :-

When you specify CSS styles inside the @keyframes rule, the animation will gradually change from the current style to the new style at certain times.

Syntax:- @keyframes animationName {
 from {
 property: value;
 }
 to {
 property: value;
 }
}

```
div {  
  animation-name: _____;  
  |           |           |  
  |           |           |  
}
```

Note:- We can use percent for animation duration instead of "from" "to".

Syntax:- @keyframes name {
 0% {
 property: value;
 }
 25% {
 property: value;
 }
 50% {
 property: value;
 }
 100% {
 property: value;
 }
}

① animation-name :-

ex:- animation-name: Ankit;

② animation-duration :-

ex:- animation-duration: 5s;

③ animation-delay :-

ex:- animation-delay: 1s;

④ animation-iteration-count:-

Set How many Times an Animation should Run.

ex:- `animation-iteration-count: 3;`
`animation-iteration-count: infinite;`

⑤ animation-direction:-

`animation-direction: normal;` → forward direction
" `: reverse;` ← backward
" `: alternate;` → ← Run forward then Backward
" `: alternate-reverse;` ← → Backward first, then Forward

⑥ animation-timing-function:-

`animation-timing-function: ease;` Slow, fast, slow
" `: linear;` Same speed
" `: ease-in;` Slow start
" `: ease-out;` Slow end
" `: ease-in-out;` Slow start & end
" `: cubic-bezier(n,n,n,n);`

⑦ animation-fill-mode:-

`animation-fill-mode: forwards;`
" `: backwards;`
" `: both;`
" `: none;` (default)

→ animation-fill-mode: forwards; (get styles from LAST Keyframe)

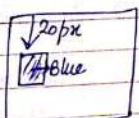
<div> (box) element retain the style values from the LAST Keyframe when Animation ends.

example:-

```
div {
  width: 100px;
  height: 100px;
  position: relative;
  animation-name: example;
  animation-duration: 3s;
  animation-fill-mode: forwards;
}
```

@keyframes example {

first keyframe ← from {
top: 0px;
to {
top: 20px;
background-color: blue;
LAST keyframe ↓




→ animation-fill-mode: backwards;


Element get the styles values set by the FIRST keyframe before the animation starts (during the animation delay).

ex:- animation-fill-mode: backwards;

@keyframes example {

from { top: 0px; 
 background-color: yellow;
 }

to {
 top: 200px;
 }

} 

→ animation-fill-mode: both;

get styles set by first frame before animation starts and then last frame after animation ends.

→ animation-fill-mode: none; (Default) Nothing

* Shorthand property:-

animation: name duration timing function delay iteration direction;
 ↓ ↓ ↓ ↓ ↓ ↓
 animation: example 5s linear 1s infinite alternate;

Using PostCSS Method Tailwind - Installation / Setup

1. Create a folder for project and Open with VS Code.
2. Open Terminal in VS Code.

3. Type "`npm init`" Press Enter.

3. Type in terminal & Run Commands.

(i) `npm i vite`

(ii) `npm install -D tailwindcss postcss autoprefixer`

(iii) `npm run tailwindcss:init`

4. Tailwind CSS intelligent extension ✓.

5. Add `content: ["*"],` in file `'tailwind.config.js'`.

6. Add Script in `'package.json'`

, "scripts": {

"start": "vite",

}

7. Create a file `'postcss.config.js'` and add:- from Tailwind website

module.exports = {

plugins: [

tailwindcss: {},

autoprefixer: {}

],

}

OR `styles.css`

8. create a css file `"main.css"` and add:- 3 lines from website.

@tailwind base;

@tailwind components;

@tailwind utilities;

9. Now, html file `"index.html"` and link css file.

End:

For Running file in browser.

Now in terminal.

1. Type → `npm start`

2. click + ctrl → `http://localhost:5173/`