CSS

- CSS stands for Cascading Style Sheets
- CSS describes how HTML elements are to be displayed on screen.
- It is used to add styling and aligning the property.

We can add CSS in 3 ways.

1. Inline CSS
2. Internal CSS
3. External CSS

Inline CSS

- It is the way of writing the css in same line.
- for this we need style attribute .

Internal CSS

it is the way of writing the css in same html file by using <style> tag.

```
<!--? syntax of selector -->

 selectorName {
propertyName : value ;
propertyName : value;
   .
   .
   .
  propertyName:value
 }
```

External CSS

- here we are writing our css code in separate css file.
- for creating css file we need .css ext.
- then we have to link our css file with our html file.
- for that we need <link> tag.
- in <link> tag we have two attributes. (rel, href)

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

<!--! Notes -->

- if we apply inline, internal and external css on the same element. the inline css will be applyed.

- internal and external css has same priority. which one will be at the end , that will be applyed.

Selectors

Type

1. Simple Selector
2. Combinator Selector
3. Pseudo Class Selector
4. Pseudo Element Selector
5. Attribute Selector

Simple Selectors

- TagName
- Id Name (#)
- Class Name (.)
- Universal (*)
- Grouping (,)

<!-- ? TagName: -->

○ To target the element based on tagname itself we have to use tagName selector.
○ The symbol was the tagname itself.

<!--? Universal: -->

○ It will target all the elements in the document including body tag too.
○ The symbol used is asterisk (*).

<!-- ? Id Name: -->

○ To target the elements uniquely we have to use an id name.
○ In CSS id name can be duplicated also there is no problem,

<!--! But once we moved to advanced languages, Repetition of id name will not work. So it is highly recommended not to use it from now on. -->

- The Symbol used is hash (#).

<!-- ? Class Name: -->

- when we want to give same css to the more than one elements then we have to use class.
- The symbol used is dot (.)

<!--? Grouping: -->

- To target multiple elements at a time we have to use a grouping selector.
- Whenever we need to pass similar properties for multiple elements we can use a grouping selector.
- The symbol used to combine all elements is comma (,)

Combinator Selector

It is a combination of 2 simple selectors.
Based on the relation b/w 2 elements the css will target the elements.
They are:

1. Descendent Selector ( )
2. Direct Child Selector (>)

<!-- ? Descendent Selector: -->

- It will target all the children, grandchildren , great grandchildren and so on.
- The symbol used is space ( ).

<!-- ? Direct Child Selector: -->

- It will target only the children but not grandchildren , great grandchildren and so on.

- The symbol used is greater than (>).

Selectors

Type

1. Simple Selector
2. Combinator Selector
3. Pseudo Class Selector
4. Pseudo Element Selector
5. Attribute Selector

Pseudo-Classes Selector

<!--? : hover -->

The `:hover` pseudo-class applies when the user keeps mouse cursor on the element.

<!--? :active -->

The `:active` pseudo-class applies during the period in which the element is being activated (e.g., between the times the user presses the mouse button and releases it).

<!--? :nth-child() -->

The `:nth-child()` pseudo-class matches elements based on their position in a group of siblings. It takes a single argument that can be a number, keyword, or formula. (an+b)

Pseudo-elements Selector

Pseudo-elements in CSS are used to style specific parts of an element.

before

The `::before` pseudo-element can be used to insert content before the content of an element.

```
<!--? syntax -->

selector::before {
   content: "content";      <!--! Required property  -->

    <!-- Other CSS properties  -->
}
```

::after

The `::after` pseudo-element can be used to insert content after the content of an element.

```
<!-- ? syntax -->

selector::after {
   content: "content";      <!--! Required property  -->

    <!-- Other CSS properties  -->
}
```

::first-line

The `::first-line` pseudo-element applies styles to the first line of a block-level element.

::first-letter

The `::first-letter` pseudo-element applies styles to the first letter of a blocklevel element.

::selection

The `::selection` pseudo-element applies styles to the portion of a document that has been highlighted by the user (e.g., selected with the mouse).

::marker

The `::marker` pseudo-element is used to style the marker box of a list item (i.e., the bullet points or numbers).

Attribute Selectors

It will target the elements based on attribute name.
Instead of creating new id names and class names we can use existing attributes.

1. [attr]: represents elements with an attribute name of attr.

2.[attr=value]:Represents element with an attribute name of attr whose value is exactly value.

Font Property

Font Properties:

1. **font-style**:

- Specifies the style of the font.
- Values: normal, italic, oblique.

- Example: font-style: italic;.

2. **font-weight**:

- Specifies the weight (or boldness) of the font.
- Values: normal, bold, bolder, lighter, or numeric values (100 to 900).

- Example: font-weight: bold;.

3. **font-size**:

- Specifies the size of the font.

- Example: font-size: 16px;.

4. **font-family**:

- Specifies a prioritized list of font family names and/or generic family names.

- Example: font-family: "Arial", sans-serif;.

Note: we can add google fonts also.

Text Property

Text properties in CSS are used to control the appearance and formatting of text on a webpage.

text-align:

- Aligns the text within an element.
- Values: left, right, center, justify, start, end.

- Example: text-align: center;.

text-decoration:

- Specifies the decoration added to text.
- Values: none, underline, overline, line-through.

- Example: text-decoration: underline;.

text-indent:

- Specifies the indentation of the first line in a text block.

- Example: text-indent: 20px;.

text-transform:
- Controls the capitalization of text.
- Values: none, capitalize, uppercase, lowercase.

- Example: text-transform: uppercase;.

text-shadow: -
Adds shadow to text.
- Values: none or a combination of h-shadow, v-shadow, blur-radius, and color.

- Example: text-shadow: 2px 2px 5px gray;.

letter-spacing:
- Sets the spacing between characters.
- Values: length values (px, em, etc.).

- Example: letter-spacing: 2px;.

word-spacing:
- Sets the spacing between words.
- Values: length values (px, em, etc.).

- Example: word-spacing: 4px;.

line-height:
- The line-height property is used to specify the space between lines:
- A line height of 1.5 means the space between lines is 1.5 times the font size.

- Example: line-height: 0.8;

Box Model

Definition:

- The CSS box model represents the structure of an element in terms of its content, padding, border, and margin.

Components of the Box Model:

1. **Content**:

   - The actual content of the element, such as text, images, or other media.
   - Example: width and height properties define the size of the content area.

2. **Padding**:

   - The space between the content and the border.
   - Example: padding: 20px; (applies padding equally to all sides).

3. **Border**:

- A line surrounding the padding (if any) and the content.
- Can be styled with different widths, colors, and styles.
- Example: border: 1px solid black; (sets a 1px solid black border).

4. **Margin**:

- The space outside the border, separating the element from adjacent elements.
- Example: margin: 10px; (applies margin equally to all sides).

Box-Sizing Property:

- **box-sizing**:

- Defines how the total width and height of an element are calculated.
- content-box (default): width and height include only the content.
- border-box: width and height include content, padding, and border.
- Example: box-sizing: border-box;.

Border Property

Definition:

- The border property in CSS is used to define the border around an element.

Key Border Properties:

1. **border-width**:

- Specifies the thickness of the border.

2. **border-style**:

- Specifies the style or type of the border.
- Common styles include:
- none, solid, dashed, dotted, double

3. **border-color**:

- Specifies the color of the border.

4. **border-radius**:

- Rounds the corners of the border, creating a curved effect.

5. **border** (Shorthand):

- Combines border-width, border-style, and border-color into a single property.
- Syntax: border: [width] [style] [color]; - Example:

```
css
border: 2px solid black;
```

```
<!-- !           CSS Units            -->
```

<!-- ! what is units ? -->

A unit is a standard for the measurement of physical quantities of the same kind.

<!-- ! Types Of Units -->

<!--? 1. Absolute Units : -->

The absolute length units are fixed and a length expressed in any of these will appear as exactly that size.

<!--? 2. Relative Units -->

Relative length units specify a length relative to another length property.

<!-- ! Types Of Absolute Units -->

1. MM(milli meter)
2. CM(Centi-meter)
3.Px(Pixel)

<!-- ! Types Of Relative Units -->

<!--? 1. %   -->

Relative to the parent element

<!--? 2. vw -->

Viewport Width: One vw is equal to 1% of the width of the viewport (the browser window size).

<!--? 3. vh -->

Viewport Height: One vh is equal to 1% of the height of the viewport.

<!--? 4. em -->

Relative to the font-size of the element: One em is equal to the current font size of the element. If the font size of the element is 16px, then 1em equals 16px. The size is relative to the parent element's font size if used in other properties.

<!--? 5. rem -->

Relative to the font-size of the root element (typically <html>): One rem is equal to the font size of the root element. This provides a consistent measurement across the entire document.

<!-- ! Display Property -->

Definition:

- The `display` property in CSS specifies the display behavior (the type of rendering box) of an element.
  It determines how an element is rendered on the web page.

`display` Values:

1. **`block`**:

- The element is rendered as a block-level element.
- Takes up the full width available.
- Examples: `<div>`, `<p>`, `<h1>` by default.

- Example: `display: block;`.

2. **`inline`**:

- The element is rendered as an inline element.
- Takes up only as much width as necessary.
- Examples: `<span>`, `<a>` by default.

- Example: `display: inline;`.

3. **`inline-block`**:

- Combines features of both `block` and `inline` elements.
- The element takes up only as much width as necessary, but you can set width and height.
    - Example: `display: inline-block;`.

4. **`none`**:

- The element is not displayed and does not take up any space in the layout.
- Example: `display: none;`.

<!--TODO: Practical Use Cases: -->

- **Creating Layouts**: Use `flex` or `grid` for modern, responsive layouts. - **Hiding Elements**: Use `display: none;` to remove elements from the document flow without deleting them from the HTML.
- **Combining Elements**: Use `inline-block` for items that need to appear side by side but still allow control over their size.

<!-- ! Color -->

RGB:- (Red, Green, Blue)
RGBA:- (Red, Green, Blue ,Alpha)
HSL:- (Hue, Saturation, Lightness)
HSLA:- (Hue, Saturation, Lightness,Alpha)
HEXADECIMAL (#00ff00)

<!--? RGB -->

RGB color specified with the rgb() function.
Syntax: rgb(red, green, blue).
Each parameter (red, green, blue) defines the, intensity of the color and can be an integer between 0 and 255 or a percentage value (from 0% to 100%).

<!--? RGBA -->

RGBA color values are an extension of RGB color values with an alpha, channel which specifics the opacity of the object.
An RGBA color is specified with rgba() function.
Syntax: rgba(red, green, blue, alpha).
The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (fully opaque).

<!--? HSL -->

HSL stands for hue, saturation, and lightness and represents a cylindrical-coordinate representation of colors.
An HSL color value is specified with hsl() function.

Syntax: hsl(hue, saturation, lightness).
Hue is a degree on the color wheel (from 0 to 360). 0 is red, 120 is green, 240 is blue.
Saturation is percentage value. 0% means a shade of gray and 100% is the full color.
Lightness is also a percentage, 0% is black, 100% is white.

<!--? HSLA -->
HSLA color values are an extension of HSL color values with an alpha, channel which specifics the opacity of the object.
An HSLA color value is specified with hsla() function.
Syntax: hsla(hue, saturation, lightness, alpha).
The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (fully opaque).

<!--? HEXADECIMAL COLOR -->

A hexadecimal color is specified with: #RRGGBB, where the RR (red), GG (green), and BB (blue) hexadecimal integers specify the components of the color. All values must be between 00 and FF.

<!--? HEXADECIMAL COLOR WITH TRANSPARENCY -->

A hexadecimal color is specified with: #RRGGBB. To add transparency, add two additional digit between 00 and FF.

<!-- ! Flex -->

Definition:
- Flexbox is a CSS layout model that allows you to design a flexible and responsive layout structure without using positioning.

Key Concepts:
- **Flex Container**: The parent element where display: flex; is applied.
- **Flex Items**: The direct children of a flex container.

Flex Container Properties:
1. **display: flex;**:
    - Defines a flex container, enabling flex context for all its direct children.

    - Example: display: flex;

2. **flex-direction**:

- Specifies the direction of the flex items in the flex container.
- Values:
- row: Default, items arranged horizontally (left to right).
- row-reverse: Items arranged horizontally (right to left).
- column: Items arranged vertically (top to bottom). - column-reverse: Items arranged vertically (bottom to top).

- Example: flex-direction: row;.

3. **flex-wrap**:

    - Controls whether flex items should wrap onto multiple lines.
    - Values:
    - nowrap: Default, all flex items will be on one line.
    - wrap: Flex items will wrap onto multiple lines, from top to bottom.
    - wrap-reverse: Flex items will wrap onto multiple lines, from bottom to top.

    - Example: flex-wrap: wrap;.
4. **justify-content**:

    - Aligns flex items along the main axis (horizontally if flex-direction: row;).
    - Values:
    - flex-start: Items align to the start of the container.
    - flex-end: Items align to the end of the container.
    - center: Items align at the center.
    - space-between: Items are evenly distributed, with the first item at the start and last at the end.
    - space-around: Items are evenly distributed with equal space around them. - space-evenly: Items are distributed with equal space between them.

    - Example: justify-content: space-between;.

5. **align-items**:
    - Aligns flex items along the cross axis (vertically if flex-direction: row;).
    - Values:
    - stretch: Default, items stretch to fill the container.
    - flex-start: Items align to the start of the cross axis.
    - flex-end: Items align to the end of the cross axis.
    - center: Items align at the center of the cross axis.

    - Example: align-items: center;.

Flex Item Properties:

1. **flex-basis**:

- Defines the default size of an element before the remaining space is distributed.
- Can be set in length units (px, % , etc.) or auto. - Example: flex-basis: 30%;.

-

<!-- ! background Property -->

Background Property in CSS

The background property in CSS is a shorthand property used to set all background-related properties in one declaration. It can include the following individual properties:

1. **background-color**:

    - Sets the background color of an element.
    - You can specify colors using names (red), hex codes (#ff0000), RGB (rgb(255, 0, 0)), RGBA (rgba(255, 0, 0, 0.5)), HSL (hsl(0, 100%, 50%)), or HSLA (hsla(0, 100%, 50%, 0.5)).

```
background-color: lightblue;
```

2. **background-image**:

    - Sets an image as the background of an element.
    - You can use a URL to specify an image, or use none to remove a background image.

```
background-image:
url('image.png');
```

3. **background-repeat**:

    - Defines how the background image is repeated.
    - Values:
    - repeat: Repeats the image both horizontally and vertically.
    - repeat-x: Repeats the image horizontally only.
    - repeat-y: Repeats the image vertically only. - no-repeat: Does not repeat the image.

```
background-repeat: no-
repeat;
```

4. **background-position**:

   - Specifies the initial position of the background image.
   - Values can be keywords (top, bottom, left, right, center), or coordinates (50% 50% or 10px 20px).

```
 background-position: 100%
100%;
```

5. **background-size**:

   - Specifies the size of the background image.
   - Values:
   - auto: Default. Keeps the original size.
   - cover: Scales the image to cover the entire element (image may be cropped).
   - contain: Scales the image to fit within the element (image won't be cropped). - Specific sizes (100px 50px, 50% 50%).

```
 background-size:
cover;
```

6. **background-attachment**:

   - Determines whether the background image is fixed relative to the viewport or scrolls with the content.
   - Values:
   - scroll: Background image scrolls with the element's content.
   - fixed: Background image stays fixed relative to the viewport.
   - local: Background scrolls with the element’s content, but not the entire page.

```
background-attachment: fixed;
```

<!-- ! overflow -->

CSS overflow Property

The overflow property in CSS controls how content is handled when it overflows its containing element's box.
This is particularly important when dealing with elements that have a fixed size, such as divs, where the content may extend beyond the boundaries of the element. **# Syntax** overflow: visible | hidden | scroll | auto ;

Values

1. **visible** (default)

- **Description**: Content is not clipped and is rendered outside the element's box. This is the default behavior for most elements. - **Use Case**: When we want the overflowing content to be fully displayed without any clipping.

- **Example**:
```
.box {
    overflow: visible;
}
```

2. **hidden**

- **Description**: The overflow is clipped, and the rest of the content is hidden. No scrollbars are provided to view the hidden content.
- **Use Case**: Useful when we want to ensure that the content does not spill over outside the element's boundaries.

- **Example**:
```
.box {
    overflow: hidden;
}
```

3. **scroll**

- **Description**: The overflow is clipped, but a scrollbar is provided to scroll and view the hidden content. The scrollbar appears whether or not the content actually overflows.
- **Use Case**: Use this when you want to ensure that the user can always scroll to see all the content.

- **Example**:
```
.box {
overflow: scroll;
}
```

4. **auto**

- **Description**: Similar to scroll, but the scrollbar is only added if the content overflows. If there is no overflow, no scrollbar is visible. - **Use Case**: Commonly used for responsive designs where scrollbars are only needed when the content is too large for its container.

- **Example**:

```
.box {
    overflow: auto;
}
```

<!-- ! transition -->

The transition property in CSS is used to create smooth animations when changing the properties of an element.
It allows you to control the speed of these property changes over a specified duration, rather than having them occur instantaneously.

Syntax transition: property duration timing-function delay;

Components of a Transition

1. **property**

- **Description**: Specifies the CSS property that the transition effect is applied to (e.g., width, height, background-color, etc.).
- **Special Value**: all - Applies the transition to all changeable properties.

- **Example**: transition: background-color 0.5s ease;

2. **duration**

- **Description**: Specifies the length of time the transition takes to complete. The duration is defined in seconds (s) or milliseconds (ms).

- **Example**: transition: background-color 1s;

3. **timing-function**

- **Description**: Defines the speed curve of the transition. It controls how the intermediate states of the transition are calculated.

- **Common Values**:
- ease: Starts slow, then fast, then ends slow (default value).
- linear: Constant speed from start to end.
- ease-in: Starts slow, then fast.
- ease-out: Starts fast, then slow.
- ease-in-out: Starts slow, speeds up, then slows down.
- **Example**: transition: width 2s ease-in;

4. **delay**

- **Description**: Specifies a delay before the transition starts. This can be in seconds (s) or milliseconds (ms).
- **Example**: transition: height 0.5s ease 0.3s;

Shorthand Property
The transition property is often written in shorthand to include all the above components. You can omit any component, and it will use the default value.

Example: transition: all 0.3s
ease-in-out;

<!-- ! transform -->

Definition:

- The transform property in CSS allows you to apply various transformations to an element, such as moving, rotating, scaling, or skewing it.

Transform Functions:

1. **translate()**:

- Moves the element from its current position.
- translate(x, y) moves the element horizontally by x and vertically by y.

- Example: transform: translate(50px, 100px); (moves the element 50px to the right and 100px down).

2. **rotate()**:

- Rotates the element around a fixed point (the center by default).
- rotate(angle) rotates the element by the specified angle in degrees.

- Example: transform: rotate(45deg); (rotates the element 45 degrees clockwise).

3. **scale()**:

- Resizes the element.
- scale(x, y) scales the element by x horizontally and y vertically.

- Example: transform: scale(2, 1.5); (doubles the width and increases the height by 50%).

4. **skew()**:

- Skews the element along the X and Y axes.

- skew(x-angle, y-angle) skews the element by the specified angles. - Example: transform: skew(30deg, 10deg); (skews the element 30 degrees along the X-axis and 10 degrees along the Y-axis).

Transform Origin:

- **transform-origin**:
- Defines the point around which the transformation occurs.
- Can be set using values like center, top, bottom, left, right, or specific coordinates.
- Example: transform-origin: top;

Position Property

Position Property in

It determines how an element is positioned in the document flow and how it interacts with other elements.

Position Values:

1. **static** (default):
   - The default positioning for all HTML elements.
   - Elements are positioned according to the normal document flow - top, right, bottom, left, and z-index properties do not apply.

```
Example:  div {
position: static;
}
```

2. **relative**:

   - The element is positioned relative to its normal position in the document flow.
   - The top, right, bottom, and left properties can be used to adjust the element's position relative to its original position. - Other elements' positions are not affected.

```
Example:       div    {
position:     relative;
top: 20px; left: 10px;
}
```

   In this example, the div is shifted 20px down and 10px to the right from where it would normally be positioned.

3. **absolute**:

- The element is removed from the normal document flow and positioned relative to its nearest positioned ancestor (i.e., an ancestor with a relative, absolute, or fixed position).
- If no such ancestor exists, it is positioned relative to the initial containing block (usually the viewport).
- top, right, bottom, and left properties are used to position the element.

```
Example:        div    {
position:      absolute;
top: 50px; left: 30px;
}
```

In this example, the div is positioned 50px from the top and 30px from the left of its nearest positioned ancestor.

4. **fixed**:

- The element is removed from the normal document flow and positioned relative to the viewport.
- The element stays fixed in position even when the page is scrolled.
- top, right, bottom, and left properties are used to position the element.

```
Example:      div   {
position:      fixed;
top:   10px;   right:
10px;
}
```

In this example, the div remains fixed 10px from the top and right of the viewport, even when the page is scrolled.

5. **sticky**:

- The element is positioned based on the user's scroll position.
- It toggles between relative and fixed, depending on the scroll position.
- The element behaves like relative until a given offset position is met, then it behaves like fixed.
- The top, right, bottom, or left properties define the threshold where the element becomes sticky.

```
Example:      div   {
position:      sticky;
top: 100px;
}
```

<!-- ! z-index -->

- The z-index property controls the stacking order of positioned elements (elements with relative, absolute, fixed, or sticky position). - Higher z-index values stack elements in front of those with lower values.

```
Example:        div    {
position:  absolute;  z-
index: 1;
}
```

Difference Between Relative And Absolute Position

Feature	position: relative	position: absolute
Definition	Positions the element **relative to its normal/static position**.	Positions the element **relative to the nearest positioned ancestor** or the viewport if no ancestor is positioned.
Document Flow	Element **remains in the document flow**; its space is reserved.	Element is **removed from the document flow**, and does not occupy space.
Positioning Reference	Uses its **original position** as the reference for top, right, bottom, and left.	Uses the **nearest positioned ancestor** (with relative, absolute, or fixed) or the viewport as the reference.
Sibling Elements	Does not overlap or affect the positioning of sibling elements.	May overlap or affect sibling elements, as it's removed from the flow.
Example CSS	position: relative; top: 20px; left: 30px;	position: absolute; top: 10px; left: 15px;

<!-- ! animation -->

- CSS Animations allow you to create smooth transitions and dynamic effects by animating changes to CSS properties over time.

Key Components of CSS Animations:

1. **`@keyframes` Rule**:
- Defines the stages of the animation and the styles at each stage. - You can specify the styles at different percentages of the animation's duration or use keywords like `from` (equivalent to `0%`) and `to` (equivalent to `100%`).

- **Example**:
    ```css
    @keyframes slideIn {       from {
transform: translateX(-100%);
      }       to {
transform: translateX(0);
    ```

```
        }
}
```

2. **Animation Properties**:

    - These properties are used to apply and control the animation on an element.

    - **`animation-name`**: Specifies the name of the `@keyframes` animation to apply.

    Example: `animation-name: slideIn;`

    - **`animation-duration`**: Defines how long the animation takes to complete one cycle.
    - Example: `animation-duration: 2s;` (2 seconds)

    - **`animation-timing-function`**: Specifies the speed curve of the animation.
    - Common values:
    - `linear`: Constant speed from start to finish.
    - `ease`: Starts slow, speeds up, and then slows down at the end.
    - `ease-in`: Starts slow and then speeds up.
    - `ease-out`: Starts fast and then slows down.
    - `ease-in-out`: Slow start, fast middle, slow end.
    - **`animation-delay`**: Sets a delay before the animation starts.
    - Example: `animation-delay: 1s;` (starts after 1 second)

    - **`animation-iteration-count`**: Defines how many times the animation should repeat.
    - Common values:
    - `infinite`: Loops the animation indefinitely.
    - A specific number (e.g., `3`).
    - **Example**: `animation-iteration-count: infinite;`

    - **`animation-direction`**: Specifies whether the animation should play forward, backward, or alternate between the two. - Values:
    - `normal`: Plays the animation forwards.
    - `reverse`: Plays the animation backwards.
    - `alternate`: Alternates between forward and reverse on each cycle.
    - **Example**: `animation-direction: alternate;`

    - **`animation-fill-mode`**: Determines how the element should be styled before the animation starts and after it ends.
    - Values:
    - `none`: Default, no styles applied before/after the animation.
    - `forwards`: Retains the final styles after the animation ends.
    - `backwards`: Applies the starting styles before the animation begins.
    - `both`: Applies both `forwards` and `backwards` behaviors.
    - **Example**: `animation-fill-mode: forwards;`

3. **Shorthand Property**:
- The `animation` property can be used to set all the animation properties in one line.
- **Syntax**: `animation: name duration timing-function delay iterationcount direction fill-mode play-state;` - **Example**:
    ```css
    .box {
        animation: slideIn 2s ease-in-out 1s infinite alternate forwards;
    }
```

What is CSS Grid?

CSS Grid is a layout system in CSS that allows developers to design web pages by organizing content into rows and columns.
It is a two-dimensional system, meaning it handles layouts in both rows and columns simultaneously.

How to Enable CSS Grid?

To use the grid layout, set the `display` property of a container to `grid` .

```
.container {
display: grid;
}
```

Key Concepts in CSS Grid

1. **Grid Container**:
   The element with `display: grid` or `display: inline-grid`. It is the parent element of the grid items.

2. **Grid Items**:
   The direct child elements of the grid container. These items are automatically placed into the grid.

CSS Grid Container Properties

1. **grid-template-columns**
   Defines the number and size of columns in the grid.
   Example:
```
    grid-template-columns: 100px 200px 1fr;
```

2. **grid-template-rows**
   Defines the number and size of rows in the grid.
   Example:
```
    grid-template-rows: 50px auto 50px;
```
3. **gap** (or `row-gap` and `column-gap`)
   Specifies the spacing between grid rows and columns.

Example:

```
gap: 10px;
row-gap: 20px;
column-gap: 15px;
```

4. **grid-auto-rows**
   Defines the size of rows that are automatically generated.
   Example:

```
grid-auto-rows: 100px;
```

5. **justify-content**
   Aligns the entire grid horizontally within the container.
   Example:

```
justify-content: space-between;
```

6. **align-content**
   Aligns the entire grid vertically within the container.
   Example:

```
align-content: space-around;
```

CSS Grid Item Properties

1. **grid-column**
   Specifies the starting and ending column for a grid item.
   Example:

```
grid-column: span 2;
```

2. **grid-row**
   Specifies the starting and ending row for a grid item.
   Example:

```
grid-row: span 3;
```

Difference Between CSS Grid and Flexbox

Aspect	CSS Grid	Flexbox
Layout Type	Two-dimensional layout system (rows and columns).	One-dimensional layout system (either rows or columns).
Primary Purpose	Best for creating complex layouts involving both rows and columns.	Best for aligning and distributing items in a single row or column.
Child Alignment	Allows placement and alignment of items both vertically and horizontally.	Aligns items along the main or cross axis in one dimension.
Container vs. Items	Focuses on the entire grid container structure.	Focuses on aligning individual items.
Content Reordering	Items can span multiple rows and columns (`grid-row`, `grid-column`).	Reordering works within a single row or column (`order`).
Syntax Complexity	More complex and verbose syntax for setting up grids.	Simpler syntax for arranging items in one dimension.
Use Cases	Ideal for page layouts, dashboards, and complex UI designs.	Ideal for navbars, toolbars, and linear alignments.

<!-- ! media query -->

Definition:

- **Media queries** are a CSS technique that allows you to apply styles to elements based on the characteristics of the device or screen displaying the content. They are essential for creating responsive web designs that adapt to different screen sizes, orientations, and resolutions.

Basic Syntax:

- The basic structure of a media query involves the `@media` rule followed by a media type and one or more expressions to check for specific conditions.

```css
@media (condition) {
  /* CSS rules here */
}
```

Common Media Features (Conditions):

1. **Width and Height**:
- **`min-width`**: Target screens that are at least a certain width.
- **`max-width`**: Target screens that are no wider than a certain width. -
  `min-height`: Target screens that are at least a certain height. -
  `max-height`: Target screens that are no taller than a certain height.
- **Example**:
  ```css
  @media (min-width: 600px) {
    /* Styles for screens wider than 600px */
  }
  ```

2. **Orientation**:
- **`orientation: portrait`**: Targets devices with a portrait orientation (height is greater than width).
- **`orientation: landscape`**: Targets devices with a landscape orientation (width is greater than height).
- **Example**:
  ```css
  @media (orientation: landscape) {
    /* Styles for landscape mode */
  }
  ```

we can also give range for applying media query :

- **Example**:
  ```css
  @media (min-width: 600px) and (max-width: 1200px) {
    /* Styles for screens between 600px and 1200px wide */
  }
  ```

<!-- ! div And span Tag -->

Tag (Block-Level Element)

- The tag is a block-level container used to group elements together for styling and layout purposes. It doesn't inherently have any meaning or style.

- **Characteristics**:
- Block-level element: Occupies the full width available.
- Can contain any HTML elements such as text, images, other divs, forms, etc.
- Useful for dividing a webpage into sections.

  -
Usage:
- Ideal for creating large structural sections like headers, footers, content areas, and navigation menus.

- Helps in applying CSS styles and JavaScript functionality to groups of elements.

** Tag (Inline-Level Element)**

- The tag is an inline-level container used to group text or other inline elements. Like , it has no default styling or meaning but is useful for applying specific styles to inline content.

- **Characteristics**:
- Inline-level element: Occupies only the space it needs (doesn't start on a new line).
- Cannot contain block-level elements (like , <h1>, etc.).
- Mainly used for styling a specific part of the text or inline elements.
  -

Usage:
- Ideal for applying styles or JavaScript to a small portion of text or elements within a larger block of content.
- Commonly used within paragraphs (<p>) to style or modify part of the text.

When to Use:
- **Use **: When you need to group large sections of content or create page layout structures.
- **Use **: When you want to apply styles or modify small parts of inline content without affecting the overall layout.

<!-- ! semantic tags -->

- **Definition**:
Semantic tags are HTML elements that clearly describe their meaning both to the browser and to developers.
Unlike non-semantic tags (,), semantic tags provide meaning to the content enclosed within them.

- **Purpose**: The primary purpose of semantic tags is to improve the structure, readability, and accessibility of web pages, making it easier for search engines, screen readers, and developers to understand the layout and content.

Common Semantic Tags

1. **<header>**
- **Definition**: Represents the introductory section or a group of navigation links in a webpage.
- **Usage**: Typically contains logo, site name, and navigation elements.
- **Example**:
    <header>

```
        <h1>My Website</h1>
        <nav>
            <a href="#">Home</a>
            <a href="#">About</a>
            <a href="#">Contact</a>
        </nav>
    </header>
```

2. **<nav>**
- **Definition**: Represents a section of the page that links to other pages or to parts within the same page.
- **Usage**: Usually used for navigation bars or menus.
- **Example**:

```
    <nav>
        <ul>
            <li><a href="#">Home</a></li>
            <li><a href="#">Services</a></li>
            <li><a href="#">Contact</a></li>
        </ul>
    </nav>
```

3. **<main>**
- **Definition**: Represents the main content of the webpage that is unique to the document, excluding headers, footers, and sidebars.
- **Usage**: Used for the central content of the webpage.
- **Example**:

```
    <main>
        <h2>Welcome to Our Website</h2>
        <p>This is the primary content of the page.</p>
    </main>
```

4. **<article>**
- **Definition**: Represents an independent piece of content that could be independently distributed or reused, such as blog posts, news articles, or user comments.
- **Usage**: Used for self-contained, reusable content.
- **Example**:

```
    <article>
        <h3>Blog Post Title</h3>
        <p>This is a blog post description.</p>
    </article>
```

5. **<section>**
- **Definition**: Represents a generic section of a document or application.
Used to group related content together.

- **Usage**: Used for thematic grouping of content.
- **Example**:

```
<section>
    <h2>Our Services</h2>
    <p>We provide high-quality services to our customers.</p>
</section>
```

6. **<aside>**
- **Definition**: Represents content that is tangentially related to the content around it, such as sidebars or pull quotes.
- **Usage**: Typically used for supplementary content.
- **Example**:

```
<aside>
    <h4>Related Articles</h4>
    <p>Check out these articles for more information.</p>
</aside>
```

7. **<footer>**
- **Definition**: Represents the footer of a section or page, typically containing copyright information, links to privacy policies, or contact details.
- **Usage**: Found at the bottom of the page or section. - **Example**:

```
<footer>
    <p>&copy; 2024 My Website. All rights reserved.</p>
</footer>
```

8. **<figure>**
- **Definition**: Represents self-contained content, like images, diagrams, or illustrations, along with their caption.
- **Usage**: Used for media elements with captions.
- **Example**:

```
<figure>
    <img src="image.jpg" alt="A beautiful scenery">
    <figcaption>Beautiful mountain scenery at sunrise.</figcaption>
</figure>
```

9. **<figcaption>**
- **Definition**: Used to define a caption for a <figure> element.
- **Usage**: Describes or gives context to the content within the <figure>.
- **Example**: (See the example for <figure> above)

10. **<mark>**
- **Definition**: Represents text that has been highlighted for reference purposes.
- **Usage**: Used to emphasize or highlight parts of the text.
- **Example**:

```
<p>The most <mark>important</mark> part of this paragraph is marked.</p>
```

11. <details>

- **Definition**: The <details> tag is used to create a collapsible section that users can open and close to reveal or hide additional information.

- **Purpose**: Provides a way to display optional or hidden information, which can be toggled by the user.
- **Behavior**: By default, the content inside the <details> tag is hidden, but it can be revealed when the user clicks on it.

- **Syntax**:

```
<details>
    <summary>Click to view more details</summary>
    <p>This is additional information that can be toggled.</p>
</details>
```

12. <summary>

- **Definition**: The <summary> tag is used inside the <details> tag to provide a heading or summary of the content that can be toggled.
- **Purpose**: Acts as a label or title that is clickable, and clicking on it reveals or hides the associated content.
- **Behavior**: It is always visible and is used to control the visibility of the rest of the content within the <details> tag.

- **Syntax**:

```
<details>
    <summary>Click here for details</summary>
    <p>Hidden content goes here.</p>
</details>
```