

CSS (Cascading Style Sheets)

CSS, or Cascading Style Sheets, is a style sheet language used for describing the presentation of a document written in HTML. It controls the layout, formatting, and appearance of multiple web pages all at once, providing a consistent and centralized way to manage the visual aspects of a website.

Why Use CSS?

1. Separation of Concerns:

CSS allows the separation of the structure (HTML) and presentation (CSS) of a web page. This makes code more modular, maintainable, and easier to understand.

2. Consistency:

CSS enables the consistent styling of multiple web pages. Changes made in a single stylesheet can be applied across the entire website.

3. Reusability:

Styles can be reused across different pages or elements. By defining styles in a stylesheet, you can apply them to multiple elements without duplicating code.

4. Ease of Maintenance:

With CSS, you can make global changes to the look and feel of a website by modifying a single stylesheet. This makes maintenance and updates more efficient.

5. Responsive Design:

CSS is essential for creating responsive designs. It allows you to adapt the layout and styling based on the device's screen size, making websites accessible on various devices.

6. Load Time:

Separating styling from HTML reduces page load times. Browser caching can be leveraged, and users can reuse the same stylesheet across multiple pages.

7. Accessibility:

CSS supports accessibility features, making it possible to create web content that is more user-friendly for people with disabilities.

How is CSS Used?

CSS is used by defining styles and rules that target HTML elements. Here's a brief overview of how CSS is used:

Selectors:

Selectors are patterns that select and style HTML elements. They can be based on element names, classes, IDs, attributes, and more.

`/* Element Selector */`

```
p {  
    color: blue;  
}
```

`/* Class Selector */`

```
.highlight {  
    background-color: yellow;  
}
```

`/* ID Selector */`

```
#header {
```

```
font-size: 20px;  
}
```

Properties and Values:

CSS properties are the aspects of an element you want to style (e.g., color, font-size), and values are the settings for those properties.

/* Property and Value */

```
p {  
    color: red;  
    font-size: 16px;  
}
```

Linking CSS to HTML:

CSS can be linked to HTML documents using the `<link>` tag in the `<head>` section or by using inline styles within HTML elements.

<!-- Linking External Stylesheet -->

```
<link rel="stylesheet" type="text/css" href="styles.css">
```

<!-- Inline Styles -->

```
<p style="color: green;">This is a green paragraph.</p>
```

Internal Styles:

Styles can also be defined within the HTML document using the `<style>` tag in the `<head>` section.

```
<style>
  body {
    background-color: #f0f0f0;
  }
</style>
```

****Selector:****

A selector in CSS is a pattern that matches one or more HTML elements. It defines the set of elements to which a particular style will be applied. Selectors can target elements based on their element type, class, ID, attributes, and more.

****Types of Selectors:****

1. **Universal Selector (*)**

Selects all elements on a page.

```
* {
  margin: 0;
  padding: 0;
}
```

2. **Tag/Element Selector:**

Selects all instances of a specific HTML element.

```
p {  
    font-size: 16px;  
}
```

3. ****ID Selector (#):****

Selects a specific element with a matching ID attribute.

```
#header {  
    background-color: #333;  
}
```

4. ****Class Selector (.):****

Selects elements with a specific class attribute.

```
.highlight {  
    color: yellow;  
}
```

5. ****Attribute Selector:****

Selects elements based on their attributes.

```
input[type="text"] {  
    border: 1px solid #ccc;  
}
```

1. ****Attribute Exists Selector ([attribute]):****

- Selects elements that have the specified attribute, regardless of its value.

```
[target] {  
    /* Styles for elements with a 'target' attribute */  
}
```

2. ****Attribute Equals Selector ([attribute=value]):****

- Selects elements with the specified attribute and value.

```
[type="text"] {  
    /* Styles for elements with 'type' attribute set to  
    "text" */  
}
```

3. ****Attribute Contains Selector ([attribute*=value]):****

- Selects elements with the specified attribute containing a certain value anywhere within its attribute value.

```
[href*="example"] {  
    /* Styles for elements with 'href' containing  
    "example" */  
}
```

4. **Attribute Starts With Selector
([attribute^=value]):**

- Selects elements with the specified attribute and value at the beginning of its attribute value.

```
[class^="btn"] {  
    /* Styles for elements with 'class' starting with "btn"  
    */  
}
```

5. **Attribute Ends With Selector
([attribute\$=value]):**

- Selects elements with the specified attribute and value at the end of its attribute value.

```
[src$=".png"] {  
    /* Styles for elements with 'src' ending with ".png"  
    */  
}
```

6. **Attribute Value Starts With Selector
([attribute|=value]):**

- Selects elements with the specified attribute and a value equal to or starting with the specified value, followed by a hyphen.

```
[lang|="en"] {  
    /* Styles for elements with 'lang' starting with "en"  
    */  
}
```

}

7. ****Attribute Not Equal Selector** (**[attribute!=value]**):**

- Selects elements with the specified attribute but not equal to the specified value.

```
[type!="checkbox"] {  
    /* Styles for elements with 'type' not equal to  
    "checkbox" */  
}
```

6. ****Relationship Selector:****

Selects elements based on their relationship in the HTML structure (descendant, child, adjacent sibling, general sibling).

```
div p {  
    margin-bottom: 10px;  
}
```

7. ****Pseudo Selector (:):****

Selects elements based on their state or position.

```
a:hover {  
    color: red;  
}
```


6. ****Group Selector:****

Selectors can be grouped together to apply the same styles to multiple selectors.

```
h1, h2, h3 {  
    font-family: 'Arial', sans-serif;  
}
```

****Specificity & When To Use Selectors:****

Specificity refers to the rules that browsers use to determine which style is applied to an element when conflicting styles exist. In general, it's recommended to use more specific selectors when needed but avoid overly complex selectors for better maintainability.

****Properties:****

Properties in CSS are the characteristics or attributes that you want to apply to the selected elements. They define how the elements should be styled.

```
p {  
    color: blue;  
    font-size: 14px;  
    margin-top: 10px;  
}
```

Sizes and Units in CSS :

In CSS (Cascading Style Sheets), various units and sizes are used to define the dimensions and spacing of elements. Here are some common units and sizes in CSS:

Units:

1. *Pixels (px):*****

- Represents a single dot on a screen.
- Absolute unit.
- Commonly used for fixed-size elements.

Example :

```
font-size: 16px;  
width: 200px;
```

2. *Em (em):*****

- Relative to the font-size of the nearest parent element.
- Can be used for responsive designs.

Example :

```
font-size: 1.5em; /* 1.5 times the parent element's font-size  
*/
```

3. *Rem (rem):*****

- Relative to the font-size of the root element (`html`).
- Useful for consistent scaling across the entire page.

Example

```
font-size: 1.2rem; /* 1.2 times the root element's font-size */
```

4. ***Percentage (%)***

- Relative to the parent element's size.
- Commonly used for width and height.

Example :

```
width: 50%; /* 50% of the parent element's width */
```

5. ***Viewport Height (vh) and Viewport Width (vw)***

- Relative to the viewport's height or width.
- Useful for responsive layouts.

Example :

```
height: 50vh; /* 50% of the viewport's height */
```

```
width: 30vw; /* 30% of the viewport's width */
```

Sizes:

1. ***Absolute Sizes***

- Fixed sizes, not affected by the size of the parent or viewport.
- Examples: `px`, `cm`, `mm`, `in`.

2. ***Relative Sizes***

- Sizes that are relative to the size of the parent element or viewport.
- Examples: `em`, `rem`, `%`, `vw`, `vh`.

3. *****Auto:*****

- Allows the browser to automatically calculate the size.

Example :

width: auto;

4. *****Fit Content:*****

- Sizes the element based on its content.

Example :

width: fit-content;

5. *****Max and Min Sizes:*****

- Sets maximum and minimum sizes for an element.

Example :

max-width: 500px;

min-height: 100px;

*****CSS General Rule:*****

The CSS general rule consists of a selector followed by a declaration block enclosed in curly braces. The declaration block contains one or more property-value pairs.

```
selector {  
    property: value;
```

}

Types of Colors:

1. ****Named Colors:****

CSS supports a set of predefined color names, such as `red`, `blue`, `green`, etc.

Example :

```
color: red;
```

2. ****Hexadecimal Colors:****

Hex codes represent colors using a combination of six alphanumeric characters.

Example :

```
color: #ff9900;
```

3. ****RGB Colors:****

RGB values define colors in terms of their Red, Green, and Blue components.

Example :

```
color: rgb(255, 0, 0);
```

4. ****RGBA Colors:****

RGBA is similar to RGB but includes an alpha channel for transparency.

Example

```
background-color: rgba(255, 0, 0, 0.5);
```

5. **HSL Colors:**

HSL stands for Hue, Saturation, and Lightness.

Example :

```
color: hsl(120, 100%, 50%);
```

6. **HSLA Colors:**

Similar to HSL, HSLA includes an alpha channel for transparency.

Example :

```
background-color: hsla(120, 100%, 50%, 0.5);
```

Coloring Text:

Example :

```
p {  
  color: blue;  
}
```

Background Colors:

Example :

```
body {  
    background-color: #f0f0f0;  
}
```

Images/URLs in CSS:

Example :

```
div {  
    background-image: url('image.jpg');  
    background-repeat: no-repeat;  
    background-size: cover;  
}
```

Other Background Properties:

Example :

```
div {  
    background-color: #f0f0f0;  
    background-image: linear-gradient(to right, #ff9900,  
    #ffcc00);  
    background-position: center;  
    background-size: cover;  
    background-attachment: fixed;  
}
```

Opacity / Transparency:

Example :

```
div {  
    background-color: rgba(255, 0, 0, 0.5);  
}
```

Gradients in CSS:

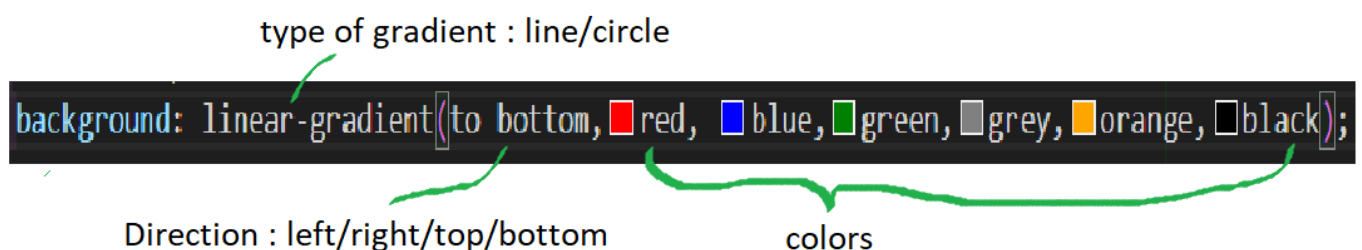
To apply gradient we need to set the type of gradient as we are having two types of gradients.

1. Linear-gradient
2. Radical-gradient

1.Linear-gradient :

Will give the straight lines gradient as background.

Ex :



The diagram shows the CSS `linear-gradient` function with annotations. The function is `background: linear-gradient(to bottom, red, blue, green, grey, orange, black);`. Annotations include: "type of gradient : line/circle" pointing to `linear-gradient`; "Direction : left/right/top/bottom" pointing to `to bottom`; and "colors" pointing to the list of color names `red, blue, green, grey, orange, black`. Each color name is preceded by a small colored square icon.

The output for the above gradient will look like below :



Here the directions can be : top bottom left right, top right, top left, bottom right, bottom left

2.Radical-Gradient :

Will give the circular lines gradient as background.

Ex

:

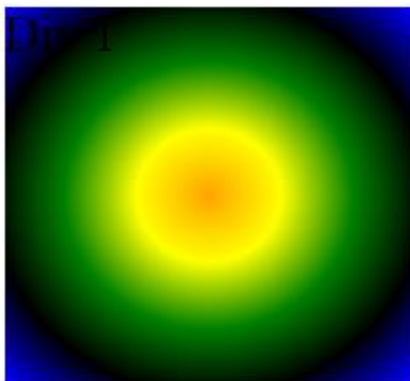
```
background: radial-gradient(position, orange, yellow, green, black, blue);
```

position

gradient type

colors

The output for the above gradient will look as below :



Here the directions can be : two types – ellipse, circle, at width(%) height (%)

- a. They both follows the below properties
 - a. Ellipse/circle at top
 - b. Ellipse/circle at bottom
 - c. Ellipse/circle at left
 - d. Ellipse/circle at right
 - e. Ellipse/circle at width (%) height (%)

NOTE : However, the practical limit may vary depending on the browser and the device's processing capabilities.

```
div {  
    background: linear-gradient(to right, #ff9900, #ffcc00);  
}
```

Box Model :

The CSS Box Model is a fundamental concept that describes how elements in a webpage are structured and how their dimensions and spacing are calculated. It consists of four main components:

1. *****Content:*****

- The actual content of the element, such as text, images, or other media.
- It is defined by the width and height properties.

2. *****Padding:*****

- The space between the content and the element's border.
- It is set using the `padding` property.
- Padding can be specified for each side (top, right, bottom, left).

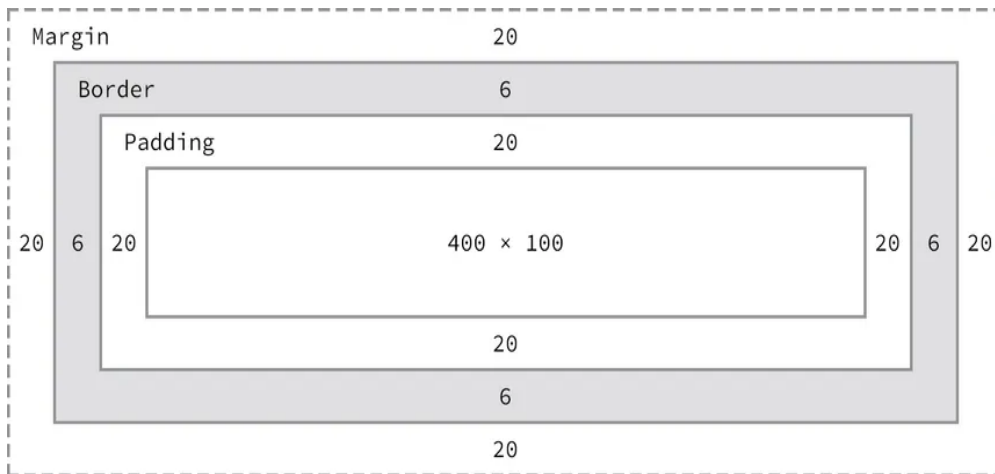
3. *****Border:*****

- A border surrounding the padding (and content).
- It is defined by the `border` property.
- Like padding, the border can have individual properties for each side.

4. *****Margin:*****

- The space between the element's border and surrounding elements.
- It is set using the `margin` property.
- Similar to padding and border, margin can have individual properties for each side.

Here's a visual representation of the CSS Box Model:



When you set the width and height of an element, you are setting the dimensions of the content area. The total space an element occupies on the page is then calculated by adding the content area, padding, border, and margin.

For example:

```
.box {  
  width: 200px;    /* Content width */  
  height: 100px;   /* Content height */  
  padding: 20px;    /* Padding */  
  border: 2px solid black; /* Border */  
  margin: 10px;     /* Margin */  
}
```

Text Manipulation :

CSS Typography refers to the styling and formatting of text on a webpage using Cascading Style Sheets (CSS). It allows web developers to control the appearance of text elements, including fonts, sizes, colors, spacing, and other typographic properties.

1. *****Font Properties:*****

- *****font-family:***** Specifies the typeface or font for the text.

Example :

```
body {  
    font-family: "Arial", sans-serif;  
}
```

- *****font-size:***** Sets the size of the text.

Example :

```
h1 {  
    font-size: 24px;  
}
```

2. *****Font Style and Weight:*****

- *****font-style:***** Sets the style of the font (normal, italic, or oblique).
- *****font-weight:***** Specifies the thickness of the characters (normal, bold, bolder, lighter, or a numeric value).

Example :

```
p {  
    font-style: italic;  
    font-weight: bold;  
}
```

3. *Text Color and Decoration:*****

- *****color:***** Defines the color of the text.
- *****text-decoration:***** Specifies the decoration applied to text (underline, overline, line-through, none).

Example :

```
a {  
    color: #3366cc;  
    text-decoration: none;  
}
```

4. *Line Height and Letter Spacing:*****

- *****line-height:***** Sets the height of a line of text.
- *****letter-spacing:***** Adjusts the spacing between characters.

Example :

```
p {  
    line-height: 1.5;  
    letter-spacing: 1px;
```

}

5. *****Text Alignment and Transformation:*****

- *****text-align:***** Aligns the text (left, right, center, justify).
- *****text-transform:***** Controls the capitalization of text.
(uppercase, lowercase, capitalize)

Example :

```
h2 {  
    text-align: center;  
    text-transform: uppercase;  
}
```

6. *****Text Shadow:*****

- *****text-shadow:***** Adds a shadow effect to the text.

Example :

```
h1 {  
    text-shadow: 2px 2px 4px #333;  
}
```

7. *****Overflow and White Space:*****

- *****white-space:***** Defines how white space inside an element is handled.
- *****overflow:***** Specifies how text should behave when it overflows its container.

Example :

```
p {  
    white-space: nowrap;  
    overflow: hidden;  
}
```

Other Styles :

- *Text-indent* : Used to give indentation for the text.

Example :

```
p{  
    text-indent : value px/em/rem;  
}
```

- *Line-height* : Used to give a line height properties to the context.

Example :

```
p{  
    line-height : 1.5;  
}
```

- *Letter-spacing* : Used to add the space between the letters.

Example :

```
p{  
    letter-spacing : 0.4em;  
}
```

- *word-spacing* : Used to add the space between the words.

Example :

```
p{  
    word-spacing : 0.4em;  
}
```

Pseudo Classes :

1. Visited :

:visited pseudo-class is used to select and style links that have been visited by the user. This pseudo-class helps in differentiating between visited and unvisited links. When a user clicks on a link and visits the associated page, the styling defined for the **:visited** pseudo-class is applied to that link.

Syntax :

```
.className/ a :visited{  
    color : value;  
    background-color: value;  
}
```

Example :

```
a:visited{  
    color : purple;  
    background-color : yellow;  
}
```

2. Hover :

:hover pseudo-class is used to select and style an element when the user hovers over it with the mouse cursor. This pseudo-class allows you to apply different styles to an element in response to user interaction, creating interactive and dynamic user interfaces.

Syntax :

```
.className/ a :hover{
```

```
color : value;
background-color: value;
}
```

Example :

```
button {
    background-color: #4CAF50;
    color: white;
    transition: background-color 0.3s ease;
}
/* hover effect*/
button:hover {
    background-color: #45a049;
}
```

3. Active :

:active pseudo-class is used to select and style an element when it is being activated or clicked. This pseudo-class is commonly used to provide visual feedback to users when they click on an interactive element, such as a button or a link.

Syntax :

```
.className/ a :active{
    color : value;
    background-color: value;
}
```

Example :

```
button {  
    background-color: #4CAF50;  
    color: white;  
    transition: background-color 0.3s ease;  
}  
/* hover effect*/  
Button:active{  
    background-color: #45a049;  
}
```

4. Focus :

:focus pseudo-class is used to select and style an element that is currently focused, typically as a result of user interaction, such as clicking on an input field or navigating through a webpage using the keyboard.

Syntax :

```
ClassName/ tag :focus{  
    //properties needs to apply  
}
```

Example :

```
input {  
    border: 1px solid #ccc;  
}
```

```
/* Styling for focus state */  
input:focus {  
    border: 2px solid #4CAF50;  
    outline: none; /* Optional: Remove the default  
focus outline */  
}
```

5. *Nth-child* :

In CSS, the `:nth-child` pseudo-class is used to select and style elements that match a specific position within their parent container. This pseudo-class allows you to apply styles to elements based on their index or position in the parent container.

syntax

```
selector:nth-child(an + b) {  
    /* styles */  
}
```

- ``selector``: The type of elements you want to select.
- ``an + b``: A formula that represents the position of the selected elements. ``n`` is a counter that starts at 0, and ``a`` and ``b`` are integers.

6. *is* :

It is used when you are trying to apply multiple pseudo properties together instead of repeating the same tag/ selector then use the `is`.

Syntax :

```
Selector:is( tag:hover, tag:active){  
    //properties;  
}
```

7. any-link :

This pseudo-class is used to select any link-like element, which includes both unvisited (:link) and visited (:visited) links. It was introduced to simplify the styling of links regardless of their visited status.

Syntax :

```
:any-link {  
    //properties  
}  
:any-link:visited {  
    //properties  
}
```

8. *Target :*

The :target pseudo-class is used to style the target element of the current URL's fragment identifier. In other words, it selects and styles an element that has an ID matching the fragment identifier in the URL.

Syntax :

```
element :target {  
    //properties  
}
```

Example :

1. **Select Every Even List Item:**

```
li:nth-child(even) {  
    background-color: #f2f2f2;  
}
```

This selects every even (2nd, 4th, 6th, etc.) ```` element and applies a background color.

2. **Select the Third Paragraph in a Container:**

```
.container p:nth-child(3) {  
    color: red;  
}
```

This selects the third ``<p>`` element inside an element with the class ``.container`` and changes its text color to red.

3. **Select Every 4th Element Starting from the 2nd:**

```
div:nth-child(4n + 2) {  
    font-weight: bold;  
}
```

This selects every 4th ``<div>`` element starting from the 2nd one and makes the text bold.

4. **Select Odd Rows in a Table:**

```
tr:nth-child(odd) {
```

```
background-color: #e0e0e0;
}
```

This selects odd rows in a table and applies a background color.

5. *****Select Every 3rd List Item Starting from the 1st:*****

```
ul li:nth-child(3n - 2) {
    text-decoration: underline;
}
```

This selects every 3rd `- ` element starting from the 1st one in an unordered list and underlines the text.

The `:nth-child` pseudo-class is powerful and flexible, allowing you to target specific elements within a container based on their position.

6. ***Is :***

```
Nav:is(a: hover, a: active){
//code
}
```

7. ***:any-link :***

```
a:any-link {
    color: purple;
}
```

8. ***Target :***

```
:target {
```



```
background-color: yellow;
}
```

9. Marker :

The `::marker` pseudo-element in CSS is used to style the marker box of a list item. It allows you to apply styles specifically to the marker of a list item, such as an unordered list (``) or an ordered list (``).

Example:

```
ul::marker {
  color: blue;
  content: "•"; /* Unicode character for a bullet point */
}
```

In this example:

- `ul::marker` targets the marker of each list item in an unordered list (``).
- `color: blue;` sets the color of the marker to blue.
- `content: "•";` replaces the default marker with a bullet point (•).

You can use `::marker` to customize the appearance of the list item markers, including changing their color, size, shape, or even replacing them with custom content.

```
ol::marker {
  color: #ff5733; /* Custom color for ordered list markers */
  font-weight: bold; /* Make the marker text bold */
}
```

```
content: counter(item) ". "; /* Use a counter for ordered list markers */  
}
```

In this example:

- ``ol::marker`` targets the marker of each list item in an ordered list (``).
- ``color: #ff5733;`` sets a custom color for the ordered list markers.
- ``font-weight: bold;`` makes the marker text bold.
- ``content: counter(item) ". ";`` replaces the default marker with a counter followed by a period for ordered lists.

NOTE : Keep in mind that browser support for `::marker`` may vary, and not all styling properties are supported universally.

Manipulating the list styles :

In CSS, you can style lists using the ``list-style`` property. This property has several values that allow you to control the appearance of list items. Here are the common values for the ``list-style`` property:

1. **`**list-style-type:**`**

- Specifies the type of marker used for the list items. Common values include:

- ``disc``: Filled circle (default).
- ``circle``: Hollow circle.
- ``square``: Filled square.
- ``decimal``: Decimal numbers.
- ``decimal-leading-zero``: Decimal numbers with leading zeros.
- ``lower-roman``: Lowercase Roman numerals.
- ``upper-roman``: Uppercase Roman numerals.
- ``lower-alpha``: Lowercase letters (alphabetic).
- ``upper-alpha``: Uppercase letters (alphabetic).

Example :

```
ul {  
    list-style-type: square;  
}
```

2. `list-style-position:**`**

- Specifies the position of the list item marker relative to the text content.

- ``inside``: Marker inside the content flow (default).
- ``outside``: Marker outside the content flow.

Example :

```
ul {  
    list-style-position: outside;  
}
```

3. *****list-style-image:*****

- Specifies an image as the marker for list items.

Example

```
ul {  
    list-style-image: url('bullet.png');  
}
```

4. *****list-style:*****

- Shorthand property to set all the list style properties in one declaration.

Example :

```
ul {  
    list-style: square outside url('bullet.png');  
}
```

Float properties in CSS :

The `float` property in CSS is used to specify whether an element should float to the left or right within its containing element. When an element is floated, it is taken out of the normal document flow and moved to the left or right until it reaches the edge of its containing element or another floated element.

Example :

```
.float-left {  
    float: left;  
}
```

```
.float-right {  
    float: right;  
}
```

In this example:

- The class `.float-left` makes an element float to the left.
- The class `.float-right` makes an element float to the right.

Note : When elements are floated, they can affect the layout of surrounding elements. Common use cases for the `float` property include creating multi-column layouts, wrapping text around images, and positioning elements side by side.

Warning : It's important to note that floating elements can cause issues with clearing and stacking, and it's often recommended to use more modern layout techniques like Flexbox or CSS Grid for complex layouts.

Additionally, the `clear` property is often used in conjunction with `float` to control how elements should behave around a floated element. For example:

Example :

```
.clear-both {  
    clear: both;  
}
```

This class would be applied to an element to ensure that it appears below any floated elements, clearing both the left and right sides.

Example :

```
<div class="float-left">Float Left</div>  
<div class="float-right">Float Right</div>  
<div class="clear-both">Clear Both</div>
```

In this example, the "Clear Both" element will appear below the floated elements and won't be affected by their positioning.

Columns in CSS :

In CSS, the `columns` property is used to create a multi-column layout for block-level elements. It allows content to flow into multiple columns, similar to a newspaper layout. The `columns` property provides control over the number of columns, their width, and the space between them.

Example :

```
.column-container {  
  columns: 3; /* Number of columns */  
  column-gap: 20px; /* Gap between columns */  
}
```

In this example:

- The `.column-container` element is set to have three columns.
- The `column-gap` property sets the space between the columns to 20 pixels.

You can also use the **shorthand** `column` property to specify the number of columns, width, and gap in one declaration:

Example :

```
.column-container {  
    column: 3 150px 20px; /* columns, width, gap */  
}
```

In this example:

- `3` represents the number of columns.
- `150px` represents the width of each column.
- `20px` represents the gap between columns.

Additionally, the `column-rule` property allows you to set the style, width, and color of the rule between columns:

Example :

```
.column-container {  
    column-rule: 2px solid #ccc; /* column rule: width style color */  
}
```

In this example:

- `2px` is the width of the column rule.
- `solid` is the style of the rule.
- `#ccc` is the color of the rule.

Example :

```
<div class="column-container">  
  <p>Column 1 - Lorem ipsum dolor sit amet...</p>  
  <p>Column 2 - Consectetur adipiscing elit...</p>  
  <p>Column 3 - Sed do eiusmod tempor incididunt...</p>  
  <!-- Add more content as needed -->  
</div>
```

The **`break-inside`** CSS property is used to control page/column breaks within an element. It specifies whether a page or column break is allowed inside a particular element, such as a block-level container. This property is often used in the context of multi-column layouts or paged media.

The **`break-inside`** property can take the following values:

- *****auto:***** The default value. The element is allowed to break inside it, based on the other break properties and the available space.

- *****avoid:***** The element prefers to not be broken inside. The browser will attempt to find a suitable break opportunity outside the element.

- *****avoid-page:***** The element prefers to not be broken inside a page. This is particularly relevant for paged media.

- *****avoid-column:***** The element prefers to not be broken inside a column. This is relevant in multi-column layouts.

Example :

```
.multi-column-container {  
    columns: 2;  
}  
  
.no-break-inside {  
    break-inside: avoid;  
}
```

In this example, the `.multi-column-container`` has two columns, and the `.no-break-inside`` class is applied to an element, indicating that it should preferably not be broken across columns.

Example

```
<div class="multi-column-container">  
    <p>Column 1 - Lorem ipsum dolor sit amet...</p>  
    <p class="no-break-inside">Column 2 - Consectetur adipiscing  
elit...</p>  
    <p>Column 2 - Sed do eiusmod tempor incididunt...</p>  
    <!-- Add more content as needed -->  
</div>
```

The `column-span`` property in CSS is used to control whether an element should span across multiple columns in a multi-column layout. This property is applied to block-level elements and determines whether the element should break across column boundaries or span the entire width of the columns.

The `column-span` property can take the following values:

- ***none***: The default value. The element does not span multiple columns.
- ***all***: The element should span across all columns.

Example :

```
.multi-column-container {  
  columns: 2;  
}
```

```
.span-all-columns {  
  column-span: all;  
}
```

In this example, the `.multi-column-container` has two columns, and the `.span-all-columns` class is applied to an element, indicating that it should span across both columns.

```
<div class="multi-column-container">  
  <p>Column 1 - Lorem ipsum dolor sit amet...</p>  
  <p class="span-all-columns">Spanning All Columns -  
  Consectetur adipiscing elit...</p>  
  <p>Column 2 - Sed do eiusmod tempor incididunt...</p>  
  <!-- Add more content as needed -->  
</div>
```

This can be useful when you have a specific element, such as a heading or a block of content, that you want to visually stand out by spanning across multiple columns in a multi-column layout.

Position in CSS :

In CSS, the `position` property is used to specify the positioning method for an element. It determines how the element is positioned within its containing element or the entire document. The `position` property can take several values:

1. *static (default)*: The element is positioned according to the normal flow of the document. The `top`, `right`, `bottom`, and `left` properties have no effect.

Example :

```
.static-example {  
    position: static;  
}
```

2. *relative*: The element is positioned relative to its normal position in the document flow. You can use the `top`, `right`, `bottom`, and `left` properties to offset it from its normal position.

Example :

```
.relative-example {  
    position: relative;  
    top: 10px;  
    left: 20px;  
}
```

3. *absolute*: The element is taken out of the normal document flow and positioned relative to its nearest positioned ancestor or the initial containing block. If there is no positioned ancestor, it is positioned relative to the initial containing block.

Example :

```
.absolute-example {  
    position: absolute;  
    top: 30px;  
    right: 10px;  
}
```

4. *fixed*: The element is taken out of the normal document flow and positioned relative to the browser window. It remains fixed even when the page is scrolled.

Example :

```
.fixed-example {  
    position: fixed;  
    bottom: 0;
```

```
right: 0;
}
```

5. *sticky***** : The element is treated as `relative` positioned until it crosses a specified point while scrolling, at which point it is treated as `fixed` positioned.

Example :

```
.sticky-example {
    position: sticky;
    top: 20px;
}
```

Z-index :

In CSS, the `z-index` property is used to control the stacking order of positioned elements on the z-axis (the axis perpendicular to the screen). Elements with a higher `z-index` value will be displayed in front of elements with a lower `z-index` value. The `z-index` property only applies to elements that have a specified `position` value other than `static` (the default).

Here's a basic example:

```
.element1 {
    position: relative;
    z-index: 2;
}
```

```
.element2 {  
  position: relative;  
  z-index: 1;  
}
```

In this example, `element1` will appear above `element2` on the z-axis because it has a higher `z-index` value.

Display in CSS :

In CSS, the `display` property is used to control the layout behavior of an element. It determines how an element is rendered in the document flow. The `display` property can take various values, each affecting the element's visibility and interaction with other elements. Here are some common values for the `display` property:

1. *****block:*****

- The element generates a block-level container. It starts on a new line and takes up the full width available.

Example :

```
div {  
  display: block;  
}
```

2. *****inline:*****

- The element generates an inline-level container. It does not start on a new line and only takes up as much width as necessary.

Example :

```
span {  
    display: inline;  
}
```

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

- The element generates an inline-level container, but it allows the use of block-level properties and can have a width and height.

Example :

```
img {  
    display: inline-block;  
}
```

4. *****none:*****

- The element is removed from the normal document flow, making it invisible and not taking up any space. Child elements are also hidden.

Example :

```
.hidden-element {  
    display: none;  
}
```

5. *****flex:*****

- The element becomes a flex container, and its children become flex items. It enables a flex context for layout.

Example :

```
.flex-container {  
  display: flex;  
}
```

We can use the below properties to set layout for the flex box.

i. flex-direction :

Used to set the direction of the flex

- a. Column – arranges the content in a multiple rows
- b. Row – will arrange the content in a single row
- c. Row-reverse – will reverse the row
- d. column-reverse – will reverse the column

Syntax :

```
.className {  
  display : flex;  
  flex-direction : row/col;  
}
```

ii. justify-content :

Will set the content inside the flex box horizontally.

- a. Start
- b. Center
- c. End
- e. Space-between
- f. Space-around

g. Space-evenly

Syntax :

```
.className{  
display : flex;  
flex-direction : row/col;  
justify-content : start/end/center/space-around;  
}
```

iii. align-items :

Will set the content inside the flex box horizontally.

a. Start

b. Center

c. End

Syntax :

```
.className{  
        align-items : start/end/center  
}
```

Iv. Flex-wrap :

The `flex-wrap` property in CSS is used within a flex container to specify how flex items should behave when the container's size is insufficient to accommodate them on a single line.

****Syntax:****

```
.container {  
    display: flex;  
    flex-wrap: nowrap | wrap | wrap-reverse;
```

```
}
```

- ``nowrap``: Default value. All flex items are forced onto one line, causing overflow if the container is too small.
- ``wrap``: Flex items wrap onto multiple lines as necessary.
- ``wrap-reverse``: Flex items wrap onto multiple lines in reverse order.

Example :

```
.container {  
  display: flex;  
  flex-wrap: wrap;  
}
```

```
.item {  
  flex: 1;  
  /* Additional styling for the flex items */  
}
```

v. Flex-flow :

Shorthand property for flex direction and wrap.

Syntax

```
.className{  
flex-flow : direction wrap;  
}
```

vi. Align-content :

The `align-content` property in CSS is used to align lines of flex items along the cross-axis in a flex container with multiple lines.

- **Syntax:**

```
.container {  
    align-content: flex-start | flex-end | center | space-between |  
    space-around | stretch;  
}
```

- **Values:**

- `flex-start`: Lines packed at the start.
- `flex-end`: Lines packed at the end.
- `center`: Lines centered within the container.
- `space-between`: Lines evenly distributed with space between them.
- `space-around`: Lines evenly distributed with space around them.
- `stretch`: Lines stretched to fill the container.

- **Example:**

```
.container {  
    display: flex;  
    flex-wrap: wrap;  
    align-content: space-between;  
}
```

6. *****grid:*****

- The element becomes a grid container, and its children become grid items. It enables a grid context for layout.

Example :

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

We can use the below properties to set layout for the flex box.

i. grid-auto-flow :

The `grid-auto-flow` property in CSS is used to control the automatic placement of grid items in a CSS Grid Layout when they are not explicitly placed using the `grid-area`, `grid-row`, or `grid-column` properties.

*****Syntax:*****

```
.container {  
    grid-auto-flow: row | column | dense | row dense | column  
    dense;  
}
```

- `row`: Grid items are automatically placed by filling each row in turn.
- `column`: Grid items are automatically placed by filling each column in turn.
- `dense`: Enables a more compact packing algorithm, attempting to fill in any gaps earlier in the grid if smaller items come up later.

*****Example:*****

```
.container {  
    display: grid;  
    grid-template-columns: repeat(3, 100px);  
    grid-auto-rows: 100px;  
    grid-auto-flow: row dense;  
}
```

Suggestion : Using `grid-auto-flow` is useful when dealing with dynamic content or unknown numbers of items, providing flexibility in how the grid layout adapts to various scenarios.

ii. Grid-template-columns :

Used to define the number of columns that we are using inside a grid.

Syntax :

```
.container{  
    grid-template-columns: repeat(3, 100px);  
}
```

- 'repeat' : s used to mention no.of columns we need to declare.

iii. Row-gap :

Used to give a gap row wise for the grid.

Syntax :

Row-gap: value;

iv. Column-gap :

Used to give a gap column wise for the grid.

Syntax :

Column-gap : value;

v. grid-column-start & grid-column-end :

Will specify the space that needs to occupy by the container in column wise.

Syntax :

```
.container{  
grid-column-start : value;  
grid-column-end : value;  
}
```

vi. grid-row-start & grid-row-end :

Will specify the space that needs to occupy by the container in row.

Syntax :

```
.container{  
grid-row-start : value;  
grid-row-end : value;  
}
```

vi. Place-content :

Used to place the content in desired position.

NOTE : There is short hand property for the grid-column and grid-row which accepts two values –start and end.

Grid-column : start/end;

Grid-row : start/end

7. *****table, table-row, table-cell:*****

- These values create table-like structures. They are used to control the layout behavior similar to table elements.

Example :

```
.table {  
    display: table;  
}
```

```
.table-row {  
    display: table-row;  
}
```

```
.table-cell {  
    display: table-cell;  
}
```

8. *****inherit:*****

- The element inherits the `display` value from its parent.

Example :

```
.inherited-display {  
    display: inherit;  
}
```


Images in CSS :

In css we are having the a property to deal with the images.

i. Background-image :

Background-image is a property which allows you to add an image as background

Syntax :

```
.container{  
    Background-image : url ( 'path');  
    Background-repeat : repeat/no repeat/ repeat-x/  
                        repeat-y  
}
```

The background-repeat will help us to fill the image in background.

ii. Background-size :

will helps us to align the background picture to that container.

Syntax :

```
.container{  
background-size : cover/contain/ length/ percentage  
}
```

iii. Background-position :

Used to position the background image.

Syntax :

Background-position : top/ bottom/ right/ left/ top right /top left/ center

TIP : we can combine the linear gradient with the background png to make it look cool.

Iv. Background-clip :

It is used to add the image to the text.

Syntax :

-webkit-background-clip : text;

or

Background-clip : text;

Color : transparent;

Note : not all browsers support the –webkit- method

v. background :

It is the shorthand property for the background-repeat, position, image, background-color.

Syntax :

Background : background-repeat, position, image, background-color;

Media Queries :

Used to set the view of the page on viewport unit. To set the content as per the device screen and height we use the media Queries.

Syntax :

```
@media mediaType and (condition : breakpoint){  
    //css properties  
}
```

Example :

```
@media screen and (min-width : value px)/(max - width : valuepx){  
background-color : red;  
}
```

We use the below breakpoints in media queries :

1. Width and height
2. Orientation
3. Resolution
4. Aspect ratio

1. width and height :

- A. Min-width /min - height
- B. Max-width/ Max-height

2. Orientation :

- A. Landscape
- B. portrait

3. Resolution :

The resolution media feature in CSS is used to query the resolution of the output device, typically a screen or printer. It allows you to apply styles based on the pixel density of the display. The resolution feature takes values in dots per inch (dpi) or dots per centimeter (dpcm).

A.

4. Aspect-ratio :

We use the device ratios

A. min-aspect-ratio : value/ value;

B. max-aspect-ratio : value/ value;

Pseudo – Elements :

Pseudo-elements are used to style a specific part of an element, creating effects or adding content without modifying the HTML.

They are denoted by double colons (::) in CSS.

Syntax :

```
Element :: pseudoElement{  
    //code  
}
```

1. ::after :

The ::after pseudo-element in CSS is used to insert content after the content of an element. It is often used for decorative purposes or to generate additional visual elements without modifying the HTML structure.

Syntax :

```
p::after{  
    Content : “content”;  
}
```

2. ::before:

The ::before pseudo-element in CSS is used to insert content before the content of an element. It is often used for decorative purposes or to generate additional visual elements without modifying the HTML structure.

Syntax :

```
p::before{  
    Content : “content”;
```

```
}
```

3. *::first-letter* :

Will target the first-letter of the text.

Syntax :

```
class::first-letter {  
    //code  
}
```

Variables :

Will help us to store the values inside a variable which can be used anywhere in the css file. Also we only need to change the value inside the variable to change the properties that are based on the variable.

- To write the variable we need to declare them inside :root pseudo class.

Syntax :

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

- To access the variable then we need to use the var().

Example :

```
:root{  
    --BgColor : red;  
}
```

```
div{  
  Background-color : var( --BgColor );  
}
```

- In general we can use the same values to store inside a variable.
- But if we use the duplicate variable name then the value will change in the original variable as well.
- So that we can use the media queries to sort this things out of it.

○ ***Syntax :***

```
@media (prefers-color-scheme : dark/light){  
:root{  
    --variableName : value;  
  }  
}
```

Functions :

In CSS (Cascading Style Sheets), functions are used to perform various tasks or calculations within style rules. Here are some common functions in CSS:

1. *****rgb() and rgba():*****

- ``rgb()`` is used to define colors using the red, green, and blue components.
- ``rgba()`` is an extension of ``rgb()`` and includes an additional parameter for opacity.

Example:

```
color: rgb(255, 0, 0);      /* Red */  
background-color: rgba(0, 128, 255, 0.5); /* Semi-transparent  
blue
```

2. *****hsl() and hsla():*****

- `hsl()` is used to define colors using the hue, saturation, and lightness components.

- `hsla()` is similar to `hsl()` but includes an additional parameter for opacity.

Example:

```
color: hsl(120, 100%, 50%);    /* Green */  
background-color: hsla(240, 100%, 50%, 0.5); /* Semi-transparent  
purple */
```

3. *****calc():*****

- `calc()` is used for performing calculations in CSS, allowing you to mix units and perform basic arithmetic.

Example:

```
width: calc(50% - 20px); /* Calculate width minus 20 pixels */
```

4. *****var():*****

- `var()` is used for working with CSS custom properties (variables). It allows you to define and reuse values throughout your stylesheets.

Example:

```
:root {
```



```
--main-color: #3498db;
}

a {
  color: var(--main-color);
}
```

5. ***url()***:

- `url()` is used to specify a path to a resource, such as an image.

Example:

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

Animations :

In css we can use this animations to make the webpage more interactive.

- To make it happen we can use the below properties in transform.
 - *Translate* – will move the container horizontally /vertically
 - TranslateX() - will move along x-axis
 - TranslateY() - will move along y-axis
 - Translate - will move along X & Y axis
 - *Rotate* – will rotate the content
 - Rotate() - will take the deg as inputs
 - *Scale* – will increase or decrease the content
 - ScaleX() - will increase the content along x axis
 - ScaleY() - will increase the content along y axis

- Scale – will increase /decrease the content along x & y axis
- *Skew – will tilt the content*
 - Skew() - will tilt in both x and y axis
 - SkewX() - will tilt in x axis
 - SkewY() - will tilt in y axis

We can use the animation name/ properties to activate the animation. To do that we have to use the following properties.

1. ***Animation-name*** : will give a name to the animation, so that we can target
2. ***Animation-duration*** : Used to set the duration of the animation.
3. ***Timing-function*** : how the animation effect will look like.
4. ***Delay*** : used to delay the animation effects
5. ***Iteration-count*** : will let us know how many no. of times we need to repeat the animation.
6. ***Direction*** : which direction will the animation need to take place.
7. ***Fill-mode*** : Defines what styles are applied to the element before and after the animation. Values include "none", "forwards", "backwards", and "both".

There is a shorthand property for all the above which is “**animation**”

Syntax :

```
selector {
    animation: name duration timing-function delay
              iteration-count direction fill-mode;
}
```

Keyframes :

Keyframes are used all the animations and create a style of animation at certain position.

Syntax :

```
@keyframe animationName{  
  From{  
    //animation  
  }  
  To{  
    //animation  
  }  
}
```

Example :

```
@keyframes bounce {  
  0%, 20%, 50%, 80%, 100% {  
    transform: translateY(0);  
  }  
  40% {  
    transform: translateY(-30px);  
  }  
  60% {  
    transform: translateY(-15px);  
  }  
}
```

```
}
```

```
/* Apply the animation shorthand to an element */
```

```
.element {
```

```
    animation: bounce 2s ease-in-out 0.5s infinite alternate;
```

```
}
```

Hall of Fame properties :

Scroll Behavior :

The `scroll-behavior` property in CSS is used to control the smoothness of scrolling behavior within an element. It determines whether scrolling should be animated or instantaneous.

The two possible values for the `scroll-behavior` property are:

1. ***auto***: This is the default value. The scrolling is instant and does not have a smooth animation.

Syntax :

```
html {
```

```
    scroll-behavior: auto;
```

```
}
```

2. ***smooth***: This value enables smooth scrolling with an animated effect. When a user triggers a scroll action, the scroll will be animated rather than immediate.

Syntax :

```
html {  
    scroll-behavior: smooth;  
}
```

Here's a simple example of how you might use it in a real-world scenario:

```
html {  
    scroll-behavior: smooth;  
}  
  
/* Other styles for your page */  
...
```

Note : With `scroll-behavior: smooth`, when users click on anchor links that navigate to different sections of the same page, the scrolling will be animated rather than a sudden jump.