CSS Reference Guide

(Cascading Style Sheet(Css))

This document is a curated and organized reference for essential CSS concepts

GitHub: https://github.com/parichabd

Author: Parichehr Abedzadeh

CSS (Cascading Style Sheets) is a language used to style and layout HTML elements on a web page for example, setting colors, fonts, spacing, and positioning.

We have three type of Css:

**Inline CSS**:  
Written directly in the HTML element using the style attribute.

**Internal CSS**:  
Placed inside a <style> tag within the <head> of an HTML file.

**External CSS**:  
Written in a separate .css file and linked to the HTML with a <link> tag.

**CSS Priority (Specificity Order):**

1. **Inline CSS** – Highest priority
2. **Internal CSS** – Medium priority
3. **External CSS** – Lowest priority

**But we usually prefer using External CSS** because:

* It keeps the code clean and maintainable
* It allows reusability across multiple pages
* It improves **performance and loading speed**, especially with browser caching

So, although **inline CSS has more power**, **external CSS is better for performance and organization**.

Common CSS Selectors:

**1. Universal Selector (\*)**

Selects **all elements** on the page.

**2. Type Selector (Tag Selector)**

Selects elements by their **HTML tag name**.

**3. Class Selector (.classname)**

Selects all elements with a specific **class**.

**4. ID Selector (#idname)**

Selects a **single element** with a specific ID.

**5. Group Selector (A, B)**

Selects **multiple elements** at once.

**6. Descendant Selector (A B)**

Selects element **B inside A**, at any depth.

**7. Child Selector (A > B)**

Selects **direct children** of element A.

**8. Adjacent Sibling Selector (A + B)**

Selects element B that is the **next sibling** of A.

**9. General Sibling Selector (A ~ B)**

Selects **all siblings** B after element A.

**10. Attribute Selector ([attr=value])**

Selects elements with a specific attribute and value.

**11. Pseudo-class Selector (:pseudo-class)**

Selects elements in a specific state.  
Examples:

**12. Pseudo-element Selector (::pseudo-element)**

Selects part of an element.

**Here is difference:** Pseudo-classes style elements **based on conditions**, while pseudo-elements style **parts inside elements**.

* **Pseudo-class (:pseudo-class)** targets an element **based on its state or position**, like :*hover, :first-child, or :checked*.
* **Pseudo-element (::pseudo-element)** targets a **part of an element's content**, like *::before, ::after, or ::first-line.*

13. **<div>:** A block-level element used to group larger sections of content. It takes up the full width available.

**<span>:** An inline element used to style or group small pieces of text or other inline content without breaking the flow.

In a website, **color** is used to set the text **color**, and **background-color** is used to set the **background color**.

Both can accept color values in different formats, including:

**Color Formats:**

* **HEX**:  
  Example: #ff0000 (red)
* **RGB** (rgb(r, g, b)):  
  Example: rgb(255, 0, 0) (red)
* **RGBA** (rgba(r, g, b, a)):  
  Adds transparency  
  Example: rgba(255, 0, 0, 0.5) (semi-transparent red)

**Width** and **height and front Size** in a website are two properties that can accept four main types of units: **px**, **em & rem**, **vh & vw**, and **percent**.

**1. px (Pixels) — For fixed, precise sizes**

* Use when you need **exact control** over size.
* Good for icons, borders, spacing in non-responsive layouts.

**2. em — Relative to the parent’s font size**

* Ideal for **padding, margin**, or element sizing that should scale with **parent font size**.
* Useful inside text components or buttons.

**3. rem — Relative to the root (html) font size**

* More consistent than em.
* Great for **typography**, layout, and global scaling.

**4. % — Relative to the parent element**

* Use when sizing elements based on the **parent’s dimensions**.
* Helpful for **responsive design**.

**5. vh / vw — Relative to the viewport size**

* 1vh = 1% of viewport height
* 1vw = 1% of viewport width
* Best for **full-screen layouts** or sections that scale with screen size.

| **Unit** | **Scales With** | **Best For** |
| --- | --- | --- |
| px | Fixed size | Precise control |
| em | Parent font size | Inner spacing, text components |
| rem | Root font size | Typography, consistent scaling |
| % | Parent element | Responsive layout |
| vh/vw | Viewport | Full-page sections, flexible layouts |

**Text and Font Properties**

color, font-size, font-family, font-weight, line-height, text-align, text-decoration, letter-spacing, text-transform, etc.

**Font family has two main approaches:**

1. **Linking the font from online services** – This can be risky because if the external server goes down or the font is removed, the font will no longer load correctly on your website.
2. **Adding the font file to your project** – This is a safer option. You can include the font directly using @font-face, ensuring it always loads as expected regardless of internet resources.
3. **text-align**

**Purpose:** Controls the horizontal alignment of text within its container.

**Values:**

* 1. left – Aligns text to the left (default for LTR languages)
  2. right – Aligns text to the right
  3. center – Centers the text
  4. justify – Stretches lines so that each line has equal width

1. **text-indent**

**Purpose:**  
Controls the **indentation of the first line** in a block of text (like a paragraph).

**5.text-decoration**

**Purpose:**  
Adds or removes decorative lines to text.

**Common Values:**

* none – No decoration (default)
* underline – Underlines the text
* overline – Adds a line above the text
* line-through – Strikes through the text
* underline overline – Can combine multiple values

**6.text-transform**

**Purpose:**  
Controls the capitalization of text.

**Values:**

* none – No transformation (default)
* capitalize – Capitalizes the first letter of each word
* uppercase – Converts all letters to uppercase
* lowercase – Converts all letters to lowercase

**7.line-height**

**Purpose:**  
Sets the vertical space between lines of text (line spacing).

**Values:**

* Number (multiplier): line-height: 1.5;
* Length unit: line-height: 24px;
* Percentage: line-height: 150%;
* normal (default)

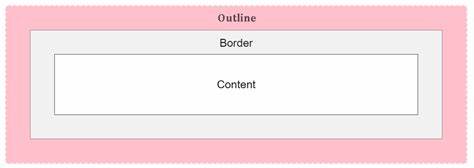
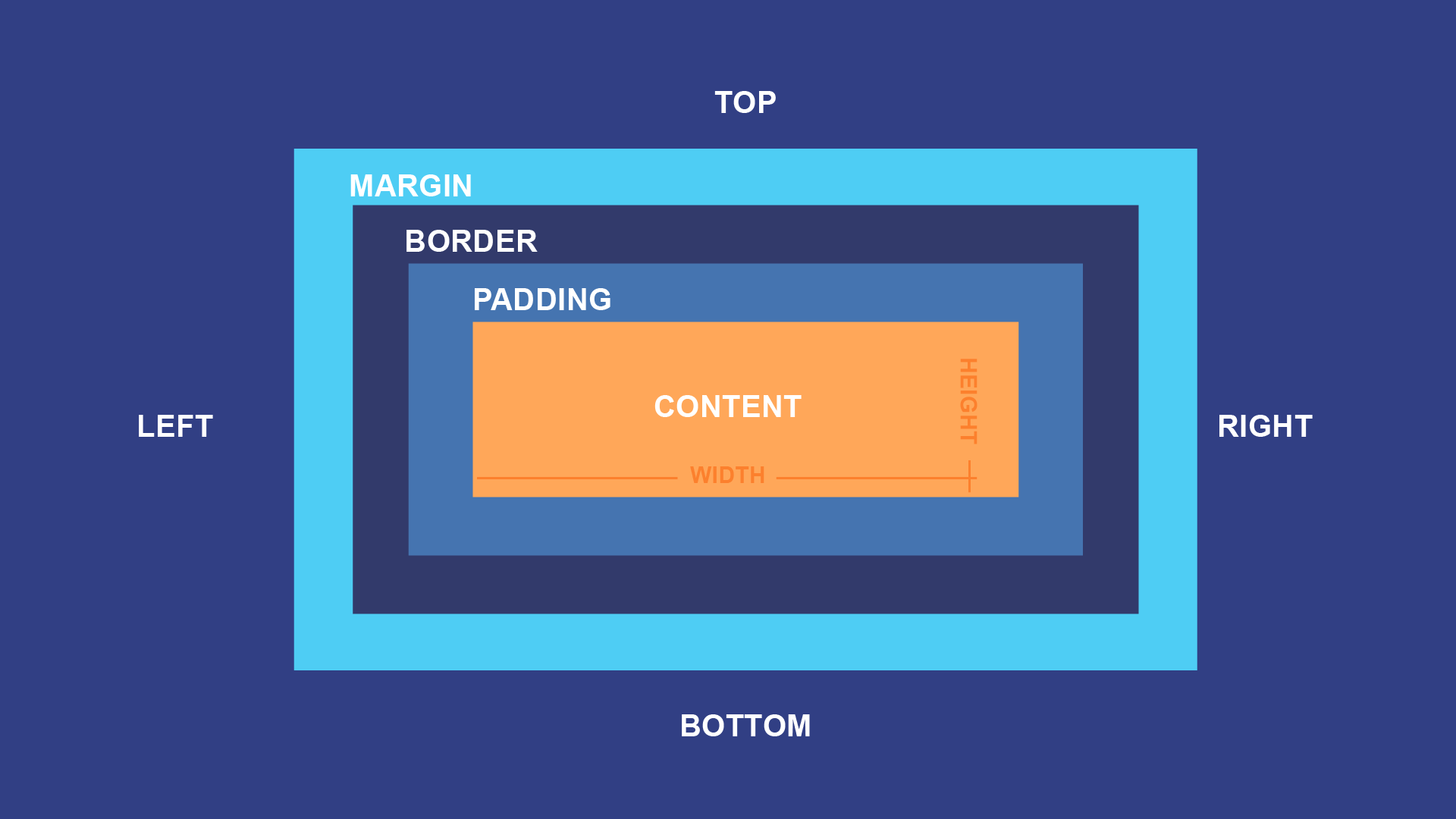
**8.letter-spacing**

**Purpose:**  
Controls the spacing between individual letters (tracking).

**Values:**

* Length values: letter-spacing: 2px;
* Can be negative: letter-spacing: -1px; (to tighten text)

Css Box Model



Outline-offset :

Display

1.inline : inline elements do **not** start on a new line. They only take up as much width as necessary and sit inline with surrounding content.

2.block : Block elements start on a new line and take up the full width of their parent container by default.

3.inline-block : its inline with block attribute

4. …

To center the content of an element on the page, (Horizontal Centring)

we use **text-align**. However, the issue is that text-align only centers the **inline content** inside block-level elements. If the width of the element is small, the content might be centered inside it, but the element itself won’t be centered on the page.

So, what is the solution for centering block elements?

* One way is to set the left and right margin to auto, and control the top and bottom margins as needed. This centers the entire block element horizontally.

For **inline elements**, margin (especially vertical) doesn't apply properly, and browsers often ignore it.

**Solution:** Change the element’s display to block.

**Problem:** A block-level element stretches to fill the full width of its parent.

**Fix:** Use display: inline-block instead — this allows proper margin/padding control without stretching the full width.

**Box-sizing:**  
Defines how the total width and height of an element are calculated — content-box excludes padding/border, border-box includes them.

**display: none;**  
Completely removes the element from the page and document flow — it behaves as if it doesn't exist.

**opacity:**  
Controls the transparency of an element (0 = fully transparent, 1 = fully visible), but the element still takes up space and responds to events.

**visibility: hidden;**  
Hides the element visually, but it still takes up space in the layout.

| **Feature** | **display: none** | **visibility: hidden** | **opacity: 0** |
| --- | --- | --- | --- |
| Visible on screen | ❌ No | ❌ No | ❌ No |
| Takes up space | ❌ No | ✅ Yes | ✅ Yes |
| Affects layout | ✅ No effect | ✅ Still affects layout | ✅ Still affects layout |
| Clickable / interactive | ❌ Not clickable | ❌ Not clickable | ✅ Still clickable (unless pointer-events is disabled) |
| Part of DOM | ❌ No (ignored by screen readers) | ✅ Yes | ✅ Yes |
| CSS transitions allowed | ❌ Not visible, not animatable | ✅ Some use, but not opacity | ✅ Yes, great for fade effects |

* opacity: 0 is often used for **animations and transitions** because it keeps the element fully in the DOM and interactive unless you disable pointer-events.
* If you want the element to be hidden **but still interactive**, use opacity: 0 **with care**.(Click)
* display: none is best for completely removing an element from view and layout.

Background:

Background: URL (“image path”)

### 🎨 **background-repeat**

Controls whether and how a background image repeats inside an element.

**Common values:**

* repeat (default): Repeats both horizontally and vertically.
* repeat-x: Repeats only horizontally.
* repeat-y: Repeats only vertically.
* no-repeat: Background image shows only once (no repetition).

### 📐 **background-size**

Defines the size of the background image.

**Common values:**

* auto: Keeps the image’s original size.
* cover: Scales the image to completely cover the element (might crop).
* contain: Scales the image to fit entirely inside the element (might leave empty space).
* Specific sizes: e.g., 100px 50px or 100% 100% to set width and height manually.

### 📍 **background-position**

Specifies the starting position of a background image inside the element.

**Common values:**

* Keywords: left, right, top, bottom, center (can be combined, e.g., center center, top right)
* Percentages: 50% 50% (center), 0% 0% (top-left)
* Length values: 10px 20px (x and y offset)

### 📎 **background-attachment**

Defines whether the background scrolls with the page or stays fixed.

**Values:**

* scroll (default): Background scrolls with content.
* fixed: Background stays fixed relative to the viewport.
* local: Background scrolls with the element's content (less commonly used).

CSS Gradients :

Gradients in CSS allow you to create smooth transitions between two or more colors. They can be used as background images, borders, or anywhere CSS accepts images.

### 1. **Types of Gradients**

There are two main types of gradients in CSS:

* **Linear Gradient:** Colors change along a straight line (horizontal, vertical, or angled).
* **Radial Gradient:** Colors radiate outward in a circular or elliptical shape from a center point.

### **Linear Gradient**

Syntax:

css

Copy code

background-image: linear-gradient(direction, color-stop1, color-stop2, ...);

* **direction:** Specifies the angle or direction (default is top to bottom).
  + Keywords: to right, to left, to bottom right, etc.
  + Angles: 45deg, 90deg, etc.
* **color-stop:** Colors and their optional stop positions (red 0%, blue 50%, etc.)

### **Radial Gradient**

Syntax:

css

Copy code

background-image: radial-gradient(shape size at position, start-color, ..., last-color);

* **shape:** circle (default) or ellipse
* **size:** closest-side, farthest-corner, etc.
* **position:** Where the gradient starts (default is center)
* **color stops:** As in linear gradients

### **Repeating Gradients**

CSS also supports repeating versions of gradients:

* repeating-linear-gradient()
* repeating-radial-gradient()

These repeat the gradient pattern indefinitely.

**CSS position**

The position property in CSS determines how an element is positioned in the document. There are five main values: static (default), relative, absolute, fixed, and sticky.

Here we focus on the four most commonly used ones.

### 1. **position: relative**

* The element remains in the normal document flow.
* You can shift it using top, right, bottom, or left **relative to its original position**.
* **Doesn't affect siblings' layout.**
* **Useful as a parent for absolute children.**

**Use cases:**

* Slightly nudging an element.
* Creating a positioned ancestor for absolutely positioned child elements.

### 2. **position: absolute**

* **Completely removed from the normal flow** (does not affect or respect siblings).
* Positioned relative to the **nearest positioned ancestor** (i.e., an ancestor with position: relative | absolute | fixed | sticky).
* If no such ancestor exists, it will be positioned relative to the <html> (viewport).

**Use cases:**

* Floating tooltips, dropdowns, modals, or precisely placed UI elements.

### 3. **position: fixed**

* **Removed from the flow** like absolute.
* Positioned **relative to the viewport**, not any ancestor.
* **Does not move** when you scroll the page.

**Use cases:**

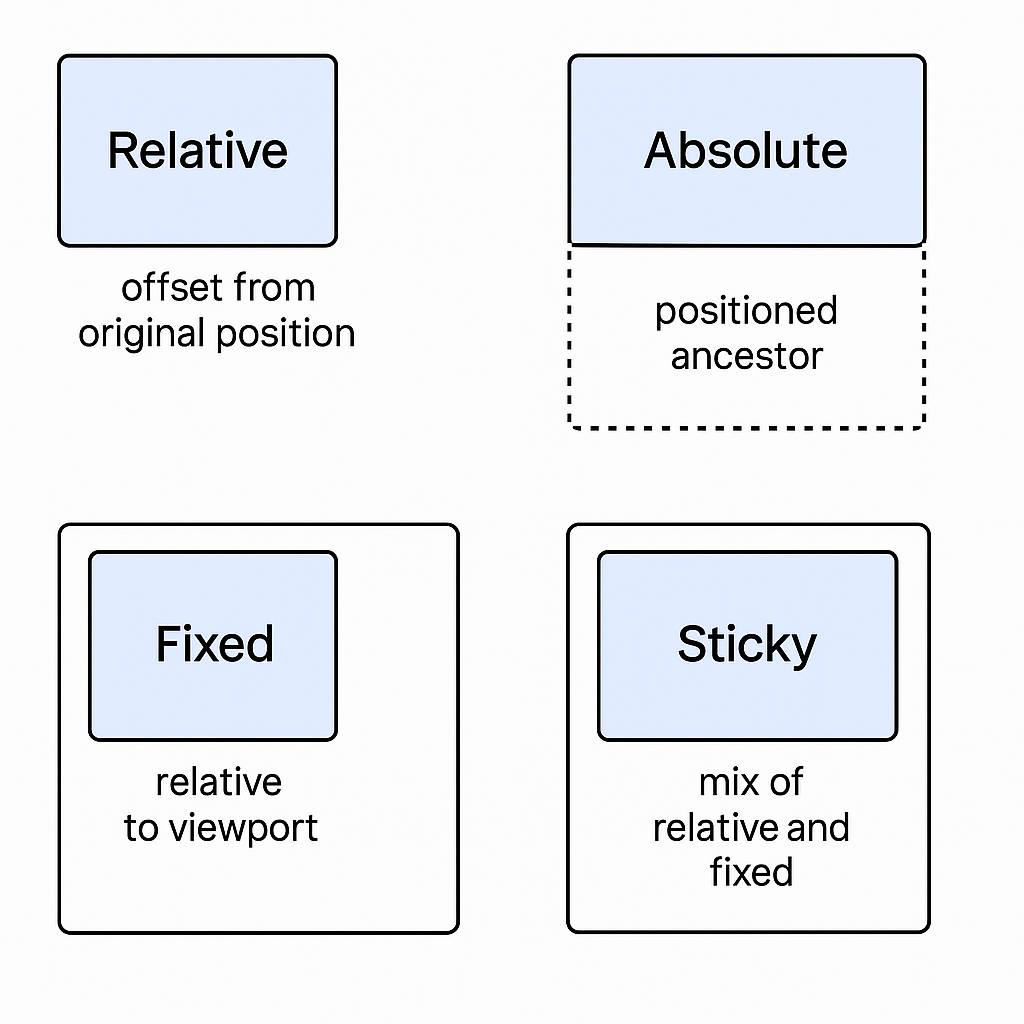
* Sticky headers/footers that always stay visible.
* Floating action buttons or fixed navigation bars.

### 4. **position: sticky**

* Acts like relative **until** a certain scroll position is reached, then sticks like fixed.
* It’s relative **within its scrollable container**.
* Needs a defined top, left, etc., and the container must have scrollable overflow.

**Use cases:**

* Sticky table headers or section headers that scroll with content and stick at the top.



### ✅ text-shadow

Used to add shadow effects to **text**.

**Syntax:**

css

CopyEdit

text-shadow: horizontal-offset vertical-offset blur-radius color;

**Example:**

css

CopyEdit

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

This adds a gray shadow 2px right, 2px down, with 5px blur.

### ✅ box-shadow

Used to add shadow effects to the **element’s box** (like div, button, etc).

**Syntax:**

css

CopyEdit

box-shadow: horizontal-offset vertical-offset blur-radius [spread-radius] color;

**Example:**

css

CopyEdit

box-shadow: 3px 3px 8px black;

Adds a black shadow 3px right, 3px down, with 8px blur.

Bottom of Form

**Flexbox (Flexible Box Layout):**

Flexbox helps us create layouts more easily and efficiently, especially when dealing with the arrangement of elements in a single dimension (either row or column).

To use Flexbox, the display: flex; property must be applied to a parent container. This container is referred to as the flex container, and the direct children inside it become flex items. It’s important to note that Flexbox works only with direct children, not nested elements deeper inside the tree.

By default, once an element is set to display: flex;, all of its children (regardless of whether they are inline, block, or inline-block) are laid out next to each other in a single row. This default behavior aligns the items in a horizontal direction (row) and prevents them from wrapping onto the next line unless you explicitly allow it using flex-wrap: wrap;.

### Key Properties:

**On the Flex Container:**

* display: flex; – Enables Flexbox layout on the container.
* flex-direction – Determines the main axis (row, column, row-reverse, column-reverse).
* justify-content – Controls alignment along the main axis (e.g., center, space-between).
* align-items – Controls alignment along the cross axis (e.g., stretch, center, flex-start).
* flex-wrap – Allows items to wrap onto multiple lines (nowrap, wrap, wrap-reverse).
* Align-content
* Flex-flow – **Flex-direction + Flex-wrap**

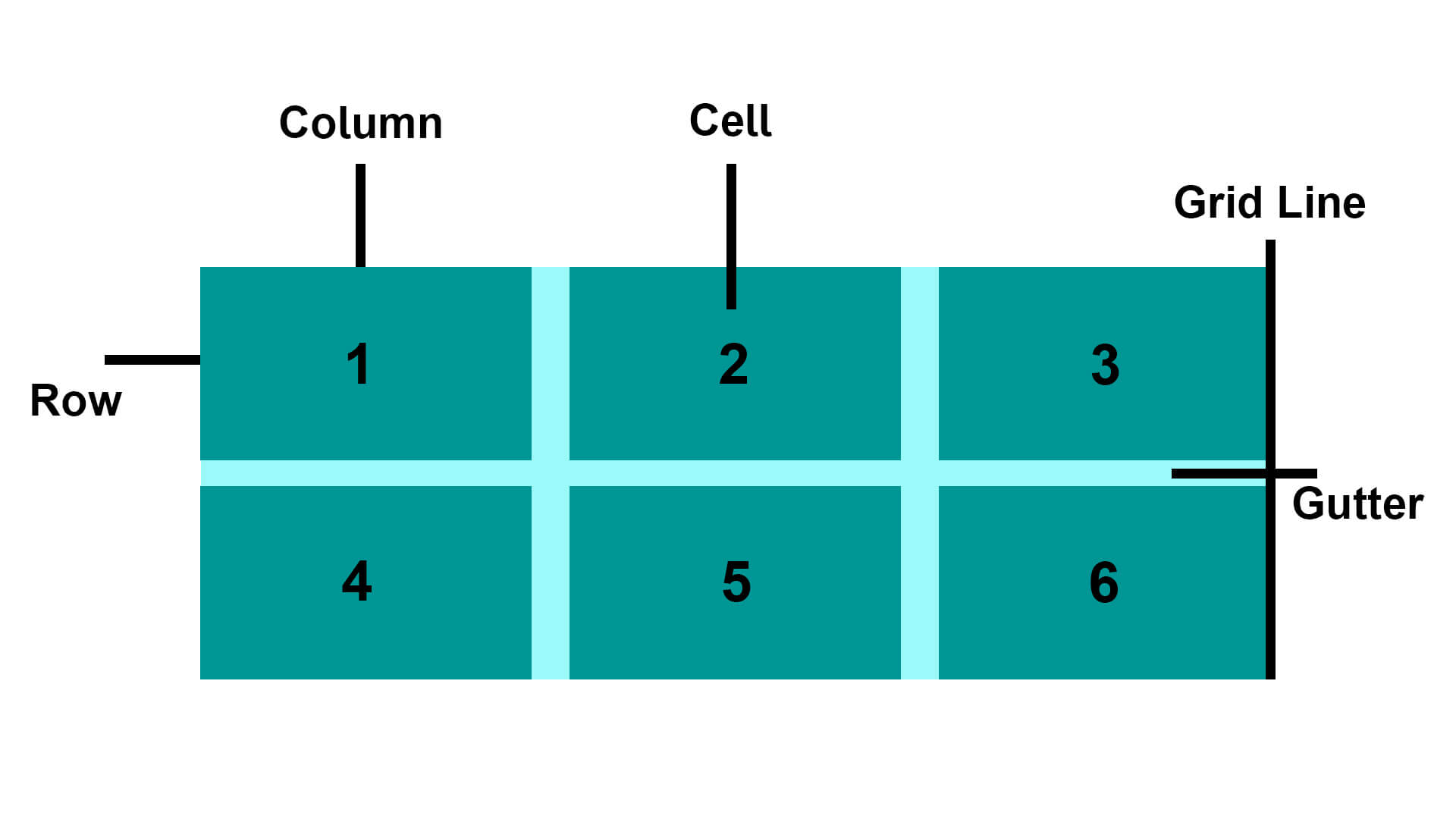
**On Flex Items:**

* flex – A shorthand to set flex-grow, flex-shrink, and flex-basis.
* align-self – Allows individual item alignment on the cross axis.
* order – Controls the visual order of items without changing the HTML structure.

**CSS Grid:**  
We know that **Flexbox** can arrange and align elements in only one dimension — either row-wise or column-wise. However, **CSS Grid** allows us to control layout in **two dimensions** (both rows and columns), making it more powerful for creating complex layouts.

Here are a few important terms in CSS Grid:

* **Gap:** Refers to the spacing between rows and columns. You can define row-gap and column-gap separately, or simply use gap to set both at once.
* **Cell:** Each box in the grid is called a cell. A cell can contain one or more elements.
* **Row:** The horizontal lines in the grid.
* **Column:** The vertical lines in the grid.



CSS Grid works similarly to Flexbox, but instead of working in one dimension, it operates in **two dimensions**: both rows and columns. Just like Flexbox, you need to apply display: grid to the **parent container** in order to control the layout of its children.

Here are some of the most important attributes in CSS Grid:

#### 1. gap or grid-gap

This property defines the space **between cells** in both directions (rows and columns). You can also use column-gap and row-gap separately if you need more control.

#### 2. grid-template-columns and grid-template-rows

These properties define the layout structure:

* grid-template-columns: sets the width of columns
* grid-template-rows: sets the height of rows

### 3. minmax(min, max)

Defines a range for sizing — between a minimum and a maximum.

Ex : grid-template-columns: minmax(100px, 1fr);

### 4. auto-fill vs auto-fit

Both are used with repeat() and useful in responsive design.

#### ✅ auto-fill

Fills the row with as many columns as possible — includes empty ones.

#### ✅ auto-fit

Fills the row but **collapses** empty columns and lets others expand.

| **Feature** |  |  |  |  |  |  |  | **auto-fill** |  |  |  |  |  | **auto-fit** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Empty slots |  |  |  |  |  |  |  | Remain visible |  |  |  |  |  | Removed |
| Layout |  |  |  |  |  |  |  | Fixed columns |  |  |  |  |  | Flexible columns |

### 5. grid-column-start, grid-column-end, etc.

Controls placement of a grid item.

✅ justify-content

Controls **horizontal alignment** of the **entire grid** inside the grid container.

| **Value** | **Description** |
| --- | --- |
| start | Aligns grid to the left |
| center | Centers grid horizontally |
| end | Aligns grid to the right |
| space-between | Equal space between columns, no outer gaps |
| space-around | Equal space around each column |
| space-evenly | Equal space before, between, and after columns |

### ✅ align-content

Controls **vertical alignment** of the entire grid inside the container.

### 🧠 Summary:

* justify-content: Aligns the grid horizontally inside the container
* align-content: Aligns the grid vertically inside the container

They only work **when the container is larger than the grid**.

### **Media Queries & Responsive Design (CSS)**

**Responsive Design** makes websites look good on all screen sizes (mobile, tablet, desktop).  
**Media Queries** are CSS rules that apply styles based on screen size, resolution, or device type.

#### **Basic Syntax:**

@media (max-width: 768px) {

/\* CSS rules for screens smaller than 768px \*/

}

#### **Common Breakpoints:**

* max-width: 1200px → large screens (laptops)
* max-width: 992px → medium screens (tablets)
* max-width: 768px → small screens (mobiles)
* max-width: 576px → extra small screens

#### **Example:**

@media (max-width: 768px) {

.main {

flex-direction: column;

}

}

This changes layout from row to column on smaller screens.

## 🧠 **CSS Transitions, Animations, and Transforms — Explained**

### ✅ 1. **CSS** transform

The transform property allows you to visually manipulate an element in 2D or 3D space. It doesn't affect layout — only how an element is rendered.

#### Common Transform Functions:

| **Function** | **Description** | **Example** |
| --- | --- | --- |
| translate(x, y) | Moves element along X and Y axes | transform: translate(50px, 20px); |
| scale(x, y) | Resizes the element | transform: scale(1.5, 1.2); |
| rotate(deg) | Rotates the element | transform: rotate(45deg); |
| skew(x, y) | Skews the element | transform: skew(10deg, 5deg); |

### ✅ 2. **CSS** transition

transition lets you smoothly animate property changes when they occur (e.g., on hover).

#### Syntax:

transition: property duration timing-function delay;

#### Common Timing Functions:

* ease – starts slow, speeds up, then slows down
* linear – same speed throughout
* ease-in – starts slow
* ease-out – ends slow
* ease-in-out – starts and ends slow

### ✅ 3. **CSS** animation

While transition only animates between two states (like hover on/off), animation uses **keyframes** to define complex, multi-step animations that can repeat.

#### Syntax:

animation: name duration timing-function delay iteration-count direction fill-mode;

#### Example:

@keyframes moveRight {

0% { transform: translateX(0); }

100% { transform: translateX(200px); }

}

.box {

animation: moveRight 2s ease-in-out infinite alternate;

}

* infinite → repeats forever
* alternate → goes back and forth
* fill-mode: forwards → keeps final state after animation ends

### ✅ 4. **Comparison Table**

| **Feature** |  | **Transition** |  | **Animation** |
| --- | --- | --- | --- | --- |
| Trigger |  | Requires event (e.g., hover, click) |  | Starts automatically or manually |
| Keyframes |  | Not used |  | Uses @keyframes |
| Complexity |  | Simple (start → end) |  | Complex (multi-step, loops, etc.) |
| Repeatable |  | No (hover only lasts while hovered) |  | Yes (infinite, alternate, etc.) |

### ✅ Summary

| **Property** |  |  |  |  |  |  |  |  |  |  |  |  | **Purpose** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| transform |  |  |  |  |  |  |  |  |  |  |  |  | Applies visual changes (rotate, scale, etc.) |
| transition |  |  |  |  |  |  |  |  |  |  |  |  | Smoothly animates between two states |
| animation |  |  |  |  |  |  |  |  |  |  |  |  | Runs complex sequences of changes via keyframes |

## 🎨 **CSS** filter **— Complete Explanation**

### ✅ What is filter in CSS?

The filter property applies **graphical effects** like blur, brightness, contrast, and more to an element — typically images, backgrounds, or any rendered content.

It acts similar to image filters in design software like Photoshop, but in CSS.

### ✅ Syntax:

filter: filter-function(value);

You can also chain multiple filters:

filter: brightness(1.2) contrast(110%) blur(2px);

### ✅ Common Filter Functions:

| **Function** | **Description** | **Example** |
| --- | --- | --- |
| blur(px) | Applies a Gaussian blur | filter: blur(4px); |
| brightness(%) | Adjusts brightness (1 is default) | filter: brightness(1.5); |
| contrast(%) | Adjusts contrast (1 is default) | filter: contrast(120%); |
| grayscale(%) | Converts to grayscale (1 = 100% grayscale) | filter: grayscale(1); |
| invert(%) | Inverts colors (useful for dark mode) | filter: invert(100%); |
| opacity(%) | Makes element transparent | filter: opacity(0.5); |
| saturate(%) | Increases or decreases color saturation | filter: saturate(200%); |
| sepia(%) | Applies a warm sepia tone | filter: sepia(100%); |
| drop-shadow() | Applies shadow like box-shadow, but as a filter | filter: drop-shadow(4px 4px 10px gray); |

### ✅ Example 1: Blur an Image

img {

### ✅ Notes:

* filter works on most HTML elements including <img>, <div>, even text.
* It is **GPU-accelerated**, so performance is generally good.
* Some filters (like drop-shadow) respect transparency, unlike box-shadow.

### ✅ Browser Support

Most modern browsers support filter, including:

| **Browser** | **Support** |
| --- | --- |
| Chrome | ✅ |
| Firefox | ✅ |
| Safari | ✅ |
| Edge | ✅ |
| Internet Explorer | ❌ (no support) |

## 🚨 **CSS** !important **— Complete Explanation**

### ✅ What is !important in CSS?

The !important keyword is used to **override all other declarations** of the same property — including inline styles, IDs, classes, and even specificity.

It gives **absolute priority** to a CSS rule.

### ✅ Syntax:

selector {

property: value !important;

}

**Why avoid overusing it?**

* Breaks natural CSS cascading and specificity
* Makes future overrides difficult
* Encourages bad practices if used as a shortcut

### ✅ When to use !important (Good Use Cases)

| **Scenario** |  |  |  |  | **Example** |
| --- | --- | --- | --- | --- | --- |
| Overriding inline styles |  |  |  |  | <div style="color: blue"> |
| Third-party libraries you can't control |  |  |  |  | Bootstrap, plugins, etc. |
| Quick fixes in temporary/debug code |  |  |  |  | During development |
| User stylesheets (e.g. accessibility) |  |  |  |  | Overriding website styles |

### ✅ Best Practice

"Use !important as a scalpel, not a hammer."

**What are CSS Variables?**

CSS variables (custom properties) let you:

* Reuse values (like colors, sizes)
* Update styles from one place
* Simplify theme switching (e.g. light/dark mode)
* Make your CSS cleaner and more maintainable

### ✅ Summary Table

| **Use** | **Syntax** |
| --- | --- |
| Define Variable | --name: value; |
| Use Variable | var(--name) |
| Global Scope | :root { --x: ... } |
| Fallback Value | var(--x, fallback) |
| Theme Switching | Override inside .dark etc. |

NOTE :

1./\* \*/ is how we can comment smt in Css

2.Site fro color Paletts : FlatUicolor.com & Colors.co

3.Site for Font : GoogleFont.com

4.Site for Background : Unsplash , Pxhle

5.Site for Gradiant : CssGradient.com

6.Site for Logo : Fontawesome.com

#Resources

- [CSS Tricks](https://css-tricks.com/)

- [MDN Web Docs - CSS](https://developer.mozilla.org/en-US/docs/Web/CSS)

- [Google Fonts](https://fonts.google.com/)

- [FlatUIColors](https://flatuicolors.com/)

- [CSS Gradient Generator](https://cssgradient.io/)