# CSS

The key to understanding how ***CSS*** works is to imagine that there is an invisible box around every HTML element.

## THREE CSS DEFINITION LOCATIONS

**Inline**: the "style" attribute

```
<p style="font-color:red;font-size:10px;">Content</p>
```

Note, the selector for inline CSS is the tag which contains the style attribute.

**Internal**: the <style> markup tag

```
<head>
  <title>Using Internal CSS</title>
  <style type="text/css">
    body {
        font-family: arial;
        background-color: rgb(185,179,175);}
    h1 {
        color: rgb(255,255,255);}
  </style>
</head>
```

**External**: the .css stylesheet file

```
<head>
  <title>Using External CSS</title>
  <link href="css/styles.css" type="text/css"
    rel="stylesheet" />
</head>
```

## Why a separate style sheet?

All of your web pages can share the same style sheet. This is achieved by using the <link> element on each HTML page of your site to link to the same CSS document. This means that the same code does not need to be repeated in every page (which results in less code and smaller HTML pages). Therefore, once the user has downloaded the CSS stylesheet, the rest of the site will load faster. If you want to make a change to how your site appears, you only need to edit the one CSS file and all of your pages will be updated. For example, you can change the style of every <h1> element by altering the one CSS style sheet, rather than changing the CSS rules on every page. The HTML code will be easier to read and edit because it does not have lots of CSS rules in the same document. It is generally considered good practice to have the content of the site separated from the rules that determine how it appears.

## Multiple css files in html

To include multiple external CSS files in an HTML document, you use multiple `<link>` tags within the `<head>` section of your HTML.

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Multiple CSS Files</title>
    <link rel="stylesheet" href="styles/common.css">
    <link rel="stylesheet" href="styles/header.css">
    <link rel="stylesheet" href="styles/main-content.css">
    <link rel="stylesheet" href="styles/footer.css">
</head>
```

## Important considerations:

- **Order of linking:** The order in which you link your CSS files matters. Styles defined in later linked files will override conflicting styles defined in earlier linked files if they have the same specificity.
- **Performance:** While using multiple CSS files can help with organization and modularity during development, it can lead to more HTTP requests, potentially impacting page load performance. For production environments, it is common

practice to concatenate and minify multiple CSS files into a single file to reduce HTTP requests.

- **@import rule:** Alternatively, you can use the `@import` rule within one CSS file to import other CSS files. For example, in `styles/common.css`, you could have `@import url("header.css");`. However, using `@import` can also have performance implications as it can cause stylesheets to be loaded in sequence rather than in parallel. Direct `<link>` tags are generally preferred for linking external stylesheets

# Selectors


→ This rule indicates that all <p> elements should be shown in the Arial typeface.

**Selectors** indicate which element the rule applies to. The same rule can apply to more than one element if you separate the element names with commas.
**Declarations** indicate how the elements referred to in the selector should be styled. Declarations are split into two parts (a **property** and a **value**), and are separated by a colon.

CSS selectors are **case sensitive,** so they must match element names and attribute values exactly.

| Name / Type | Selector | Where It Is Used | When to Use | Example |
|---|---|---|---|---|
| Universal Selector | `*` | Applies to all elements in the document | | * {<br>  font-size: 16px;<br>} |
| Type Selector | `p, th` | To style **multiple different elements** with the same rules | When several tags share identical styling | p, th {<br>  font-size: 16px;<br>} |
| Class Selector | `.dept` | To style **any number of elements** that should look similar | When styles must be reused across elements | CSE <br><br> EEE |
| | `p.note` | Targets only  elements whose class attribute has a value of note | | |
| ID Selector | `#username` | To style **a unique element** (appears once per page) | When an element needs unique styling or is targeted by JS | <input id="username"> |
| Child Selector | `div>p` | Targets any  elements that are children of an  element (but not other elements in the page) | | <br>  Styled <br><!-- YES --><br>  <br>  Not styled <br><!-- NO (too deep) --><br>  <br> |

| | | | | |
|---|---|---|---|---|
| Descendant Selector | `div h3` | To style elements **only when they appear inside a specific parent** | When context matters (e.g., only h3 inside div) | <br><h3>Styled</h3> <!-- gets the style --><br> <br><br><h3>Not styled</h3> <!-- unaffected --> |
| Pseudo-class Selector | `th:hover` | To apply styles during **user actions** (hover, focus, active, etc.) | When you want interactive or dynamic UI behavior | th:hover {<br>background: yellow;<br>} |

Some example: CSS Selector Examples.html

| | |
|---|---|
| If the two selectors are identical, the latter of the two will take precedence. | If one selector is more specific than the others, the more specific rule will take precedence over more general ones. |

## CSS Combinators

A combinator is something that defines the relationship between two or more selectors. A CSS selector can contain more than one selector. Between the selectors, we can include a combinator, to create a more specific selection.

There are four different combinators in CSS:

- ☐ Descendant combinator (space)
- ☐ Child combinator (>)
- ☐ Next sibling combinator (+)
- ☐ Subsequent-sibling combinator (~)

### Next Sibling Combinator (+)

The next sibling combinator is used to select an element that is directly after a specific element.
Sibling elements must have the same parent element.

The following example selects the first <p> element that immediately follows a , and share the same parent:

```html
<!DOCTYPE html>
<html>
  <head>
    <style>
      div + p {
        background-color: yellow;
      }
    </style>
  </head>


  <body>

    <div>
      <p>Paragraph 1 in the div.</p>
      <p>Paragraph 2 in the div.</p>
    </div>

    <p>Paragraph 3. After a div.</p>
    <p>Paragraph 4. After a div.</p>

    <div>
      <p>Paragraph 5 in the div.</p>
      <p>Paragraph 6 in the div.</p>
    </div>

    <p>Paragraph 7. After a div.</p>
    <p>Paragraph 8. After a div.</p>

  </body>
</html>
```

Paragraph 1 in the div.

Paragraph 2 in the div.

Paragraph 3. After a div.

Paragraph 4. After a div.

Paragraph 5 in the div.

Paragraph 6 in the div.

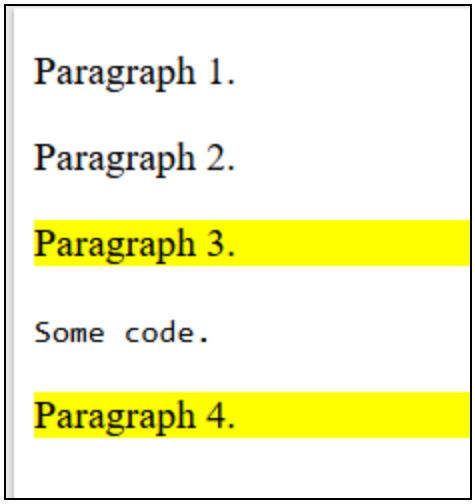
Paragraph 7. After a div.

Paragraph 8. After a div.

Subsequent-sibling Combinator (~)

The subsequent-sibling combinator selects all elements that are next siblings of a specified element.

```html
<!DOCTYPE html>
<html>
  <head>
    <style>
      div ~ p {
        background-color: yellow;
      }
    </style>
  </head>

  <body>

    <p>Paragraph 1.</p>

    <div>
      <p>Paragraph 2.</p>
    </div>

    <p>Paragraph 3.</p>
    <code>Some code.</code>
    <p>Paragraph 4.</p>

  </body>
</html>
```




## Pseudo-class

A keyword added to a selector that specifies a special state of an element. Examples include when a link is hovered, clicked, visited, or when an input is focused.
Link: CSS Pseudo-class

# Some common CSS:

Here is a list of the most common CSS fields and an example:

```
.color: #FF0000;    red;    rgba(255,00,100,1.0); //different ways to specify colors
.background-color: red;
.background-image: url('file.png');
.font: 18px 'Tahoma';
.border: 2px solid black;
.border-top: 2px solid red;
.border-radius: 2px; //to remove corners and make them more round
.margin: 10px; //distance from the border to the outer elements
.padding: 2px; //distance from the border to the inner elements
.width: 100%;    300px;    1.3em;  //many different ways to specify distances
.height: 200px;
.text-align: center;
.box-shadow: 3px 3px 5px black;
.cursor: pointer;
.display: inline-block;
.overflow: hidden;
```

# Position

CSS Position

[CSS Position Ultimate Guide]

- **Static positioning** is the default behavior where elements flow normally in the document. Use this for most content when you don't need special positioning - it's perfect for paragraphs, basic layouts, and any content that should follow the natural document flow without any offset or special placement.
- **Relative positioning** is ideal when you need to nudge an element slightly from its normal position without affecting other elements around it. Think of it as fine-tuning - use it for small adjustments like overlapping elements slightly, creating subtle visual effects, or when you need to establish a positioning context for absolutely positioned child elements. The key advantage is that the element's original space in the layout is preserved.
- **Absolute positioning** is your go-to when you need to place an element at specific coordinates within a container. It's perfect for tooltips, dropdown menus, modal windows, or any element that needs precise placement relative to its parent. Remember that absolutely positioned elements are removed from the normal document flow, so other elements will act like they don't exist.
- **Fixed positioning** is designed for elements that should stay visible regardless of scrolling. Use this for persistent navigation bars, floating action buttons, chat widgets, or any UI element that needs to remain constantly accessible. Fixed

elements are positioned relative to the viewport, making them ideal for creating sticky headers or footers that provide constant navigation.

- **Sticky positioning** combines the best of relative and fixed. It's perfect for section headers in long lists, table headers, or any content that should stick temporarily during scrolling. Use sticky when you want an element to scroll normally until it reaches a threshold, then lock in place - think of category headers in product listings, column headers in tables, or navigation that becomes fixed after some scrolling.

Absolute positioning stays inside its parent - but only if the parent is "positioned".

An absolutely positioned element looks for its nearest positioned ancestor (any element with position: relative, absolute, fixed, or sticky) and positions itself relative to that parent's boundaries. If no positioned ancestor exists, it uses the document body as its reference point.

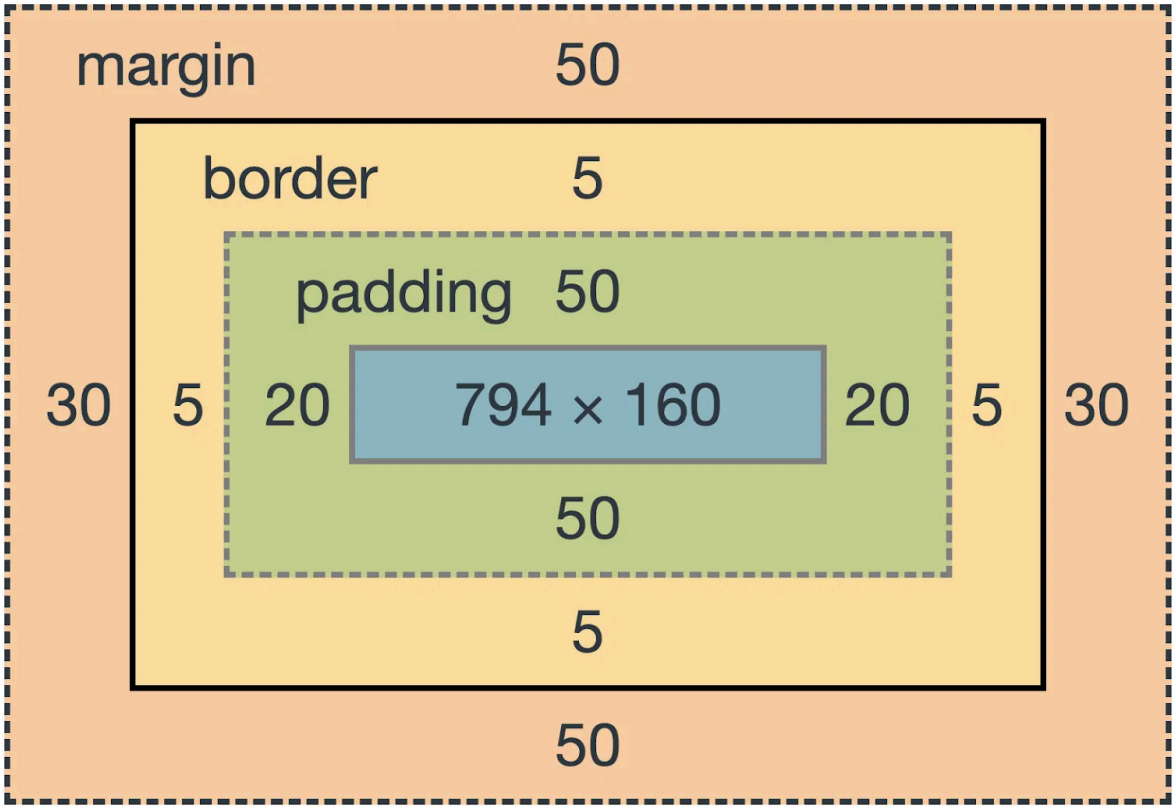
| Property | Scrolls With Page? | Positioned Relative To |
|---|---|---|
| **absolute** | Yes | nearest positioned parent |
| **fixed** | No | viewport (screen) |

| Position Type | What It Is | How It Behaves | When to Use It |
|---|---|---|---|
| **static** (default) | The default position for all elements | Ignores top/right/bottom/left; follows normal document flow | When you don't need special positioning; everyday layout with no offsets |
| **relative** | Positioned relative to its original position | Still takes up original space; offsets (top/left/…) move it visually without affecting surrounding elements | When you need a small nudge or want to create a positioning context for absolutely-positioned children |

| | | | |
|---|---|---|---|
| **absolute** | Removed from normal flow and positioned relative to the nearest positioned ancestor | Does NOT take up space; uses top/left/bottom/right relative to the closest ancestor with position other than static | When you need elements that overlap, tooltips, badges, dropdown menus, controlled overlays inside a container |
| **fixed** | Positioned relative to the viewport | Stays in the same spot even when scrolling; removed from flow | When you need UI elements that always stay visible—sticky headers, floating buttons, always-on-screen menus |
| **sticky** | Acts like relative until a scroll threshold, then sticks like fixed | Toggles between relative and fixed depending on scroll position; stays within its parent container's boundaries | When you want a header, sidebar, or label that sticks while scrolling through a specific section |

# Box model

Example link: CSS Box Model




1. Content (blue part)
    - The text or element inside the box.
    - If no styling is added, the box shrinks to fit this content.
    - You can control it using:
        - width and height.
2. Padding
    - Space inside the box, around the content.
    - Pushes the content inward.
    - Expands the background color.
    - Commonly used in styling buttons (bigger clickable area).
    - Use padding to add space inside an element (background expands).
    - Example: padding: 20px; adds 20px on all sides.

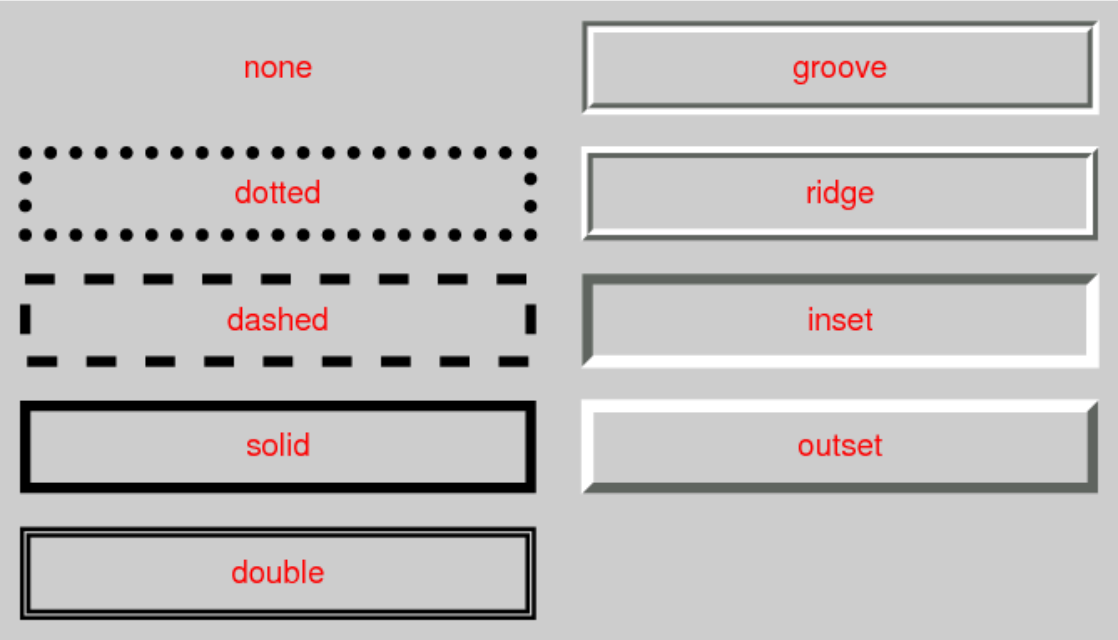
padding:25px — All Four Side

padding:25px 75px; — Top & bottom - left & right

padding:25px 50px 75px; — Top – right & left - bottom

padding:25px 50px 75px 100px; — Top  right bottom & left

3. Border
    - The visible outline around the padding.
    - Can be styled with size, color, style.
    - Example: border: 20px solid purple;
    - Border style:




    - Can also set border-width, but it won't work alone; it needs to be paired with border style.
4. Margin
    - Space outside the border.
    - Used to separate elements from each other.
    - Example: margin: 20px; creates space outside the box

# Margin Collapse

When **two vertical margins touch**, they collapse:
➞ **Only the larger margin is applied.**

Example:

- Box 1: `margin-bottom: 70px`
- Box 2: `margin-top: 60px`
  Result = **70px total spacing**, not 130px.

# Box Size Calculations (Default Behavior)

By default (`box-sizing: content-box`):

```
Total height = height + padding (top+bottom) + border
(top+bottom) (margins are not counted)
```

# `box-sizing: border-box` (Highly Recommended)

Makes the declared width/height include **padding + border**.

➞ Your element stays exactly the size you set.

# Z-index

- [ ] The z-index property specifies the stack order of positioned elements.
- [ ] The stack order defines which element should be placed in front or behind other elements.
- [ ] An element with greater stack order is always above an element with a lower stack order.

Example: CSS z-index.html

# Color

Example: CSS Color

It allows you to specify the color of **text** inside an element.

| Name | Explanation |
|---|---|
| rgb values | These express colors in terms of how much red, green and blue are used to make it up. For example: rgb(100,100,90) |
| RGBA Colors | rgba(red, green, blue, alpha)<br><br>Alpha: 0 (transparent) to 1 (opaque) |
| hex Codes (mostly used) | These are six-digit codes that represent the amount of red, green and blue in a color, preceded by a pound or hash # sign. For example: #ee3e80 |
| Color names | There are 147 predefined color names that are recognized by browsers. For example: DarkCyan |
| HSL Colors | Express colors using Hue (color wheel position), Saturation (intensity), and Lightness (brightness)<br>hsl(120, 100%, 50%) creates a pure green color |
| HSLA Colors | HSL colors with an additional Alpha channel for transparency control<br>hsla(240, 100%, 50%, 0.5) creates a semi-transparent blue color |