

A Guide to CSS Concepts

This document covers the fundamental concepts of Cascading Style Sheets (CSS), from basic syntax to modern layout techniques.

1. CSS Basics: Syntax and Inclusion

CSS is a language used to describe the presentation of a document written in HTML.

Syntax

A CSS rule consists of a **selector** and a **declaration block**.

- The **selector** points to the HTML element you want to style.
- The **declaration block** contains one or more declarations separated by semicolons.
- Each **declaration** includes a CSS **property** name and a **value**, separated by a colon.

```
/* This is a CSS comment */
```

```
selector {  
    property: value;  
    another-property: another-value;  
}
```

Example:

```
/* Selects all <p> (paragraph) elements */  
p {  
    color: blue;  
    font-size: 16px;  
}
```

How to Add CSS to HTML

There are three ways to include CSS:

1. External Stylesheet (Best Practice)

A separate .css file is linked in the HTML <head>. This is the most efficient method.

HTML (<head> section):

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

CSS (styles.css):

```
body {
```

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

2. Internal Stylesheet

CSS is placed inside a `<style>` tag within the HTML `<head>`. Useful for single-page styles.

HTML (`<head>` section):

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

```
h1 {  
    color: navy;  
}  
</style>
```

3. Inline Styles

CSS is applied directly to an HTML element using the `style` attribute. This is generally discouraged as it's hard to maintain.

HTML (`<body>` section):

```
<h1 style="color: navy; font-size: 24px;">This is a heading</h1>
```

2. Selectors

Selectors are patterns used to target (select) the elements you want to style.

Basic Selectors

- **Element Selector:** Selects all elements of a specific type.

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

- **Class Selector:** Selects all elements with a specific class attribute. Uses a period (.)

```
/* HTML: <p class="highlight">...</p> */  
.highlight {  
    background-color: yellow;  
}
```

- **ID Selector:** Selects a single element with a specific id attribute. Uses a hash (#). IDs must be unique.

```
/* HTML: <div id="container">...</div> */  
#container {  
    width: 960px;  
}
```

- **Universal Selector:** Selects all elements. Often used for resets.

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

- **Grouping Selector:** Applies the same styles to multiple selectors.

```
h1, h2, h3 {  
    color: #333;  
}
```

Combinators (Combining Selectors)

- **Descendant Selector (space):** Selects elements that are descendants of a specified element.

```
/* Selects any <a> inside an element with class .nav */  
.nav a {  
    color: red;  
}
```

- **Child Selector (>):** Selects elements that are a *direct child* of a specified element.

```
/* Selects only <li> that are direct children of a <ul> */  
ul > li {  
    list-style-type: circle;  
}
```

- **Adjacent Sibling Selector (+):** Selects an element that is *immediately after* another specified element.

```
/* Selects the first <p> immediately after an <h1> */  
h1 + p {  
    font-weight: bold;  
}
```

- **General Sibling Selector (~):** Selects all elements that are *siblings after* a specified element.

```
/* Selects all <p> that come after an <h1> */  
h1 ~ p {  
    margin-left: 10px;
```

```
}
```

Pseudo-classes

A pseudo-class is used to define a special state of an element.

```
/* Style a link when the mouse is over it */  
a:hover {  
    text-decoration: underline;  
}  
  
/* Style a link that has already been visited */  
a:visited {  
    color: purple;  
}  
  
/* Style an input element when it is focused */  
input:focus {  
    border-color: blue;  
}  
  
/* Selects the first <p> element among its siblings */  
p:first-child {  
    font-style: italic;  
}  
  
/* Selects the third <li> element */  
li:nth-child(3) {  
    color: green;  
}
```

Pseudo-elements

A pseudo-element is used to style specified parts of an element.

```
/* Add content before every <p> element */  
p::before {  
    content: "Note: ";  
    font-weight: bold;  
}  
  
/* Add content after every <p> element */  
p::after {
```

```
content: " - Read more";
}

/* Style the first line of every <p> element */
p::first-line {
    font-size: 1.2em;
}
```

3. The Box Model

Every element in CSS is a rectangular box. The box model describes how this box is structured.

- **Content:** The text, image, or other content.
- **Padding:** The transparent space *inside* the border, between the content and the border.
- **Border:** The line that goes *around* the padding and content.
- **Margin:** The transparent space *outside* the border, separating it from other elements.

```
div {
    /* Size of the content box */
    width: 300px;
    height: 150px;

    /* Padding (shorthand: top, right, bottom, left) */
    padding: 20px;
    /* padding-top: 20px; (can be set individually) */

    /* Border (shorthand: width, style, color) */
    border: 5px solid black;
    /* border-style: solid; (can be set individually) */

    /* Margin (shorthand) */
    margin: 10px;
    /* margin-bottom: 15px; (can be set individually) */
}
```

box-sizing

By default, width and height apply *only to the content*. Padding and border are added *on top* of that, making the box bigger than you specified.

box-sizing: border-box; changes this. It makes width and height apply to the *total* box,

including padding and border. This is much more intuitive.

```
/* A modern, common reset */
* {
  box-sizing: border-box;
}

.box {
  /* This box will be 100px wide in total,
     NOT 100px + 20px padding + 2px border */
  width: 100px;
  padding: 10px;
  border: 1px solid red;
}
```

4. Typography (Text Styling)

```
p {
  /* Sets the font (with fallbacks) */
  font-family: 'Helvetica Neue', Arial, sans-serif;

  /* Sets the text size */
  font-size: 16px;

  /* Sets the font thickness */
  font-weight: normal; /* 400, bold (700), etc. */

  /* Sets the text color */
  color: #333; /* Dark gray */

  /* Sets the horizontal alignment */
  text-align: left; /* center, right, justify */

  /* Sets the space between lines of text */
  line-height: 1.5; /* 1.5 times the font size */

  /* Adds or removes underlines, etc. */
  text-decoration: none; /* underline, overline */

  /* Changes the case of the text */
  text-transform: none; /* uppercase, lowercase, capitalize */
}
```

Units: px, em, rem, %

- **px (Pixels)**: An absolute, fixed-size unit.
- **% (Percent)**: Relative to the parent element's property.
- **em**: Relative to the font-size of the *parent element*.
- **rem (Root Em)**: Relative to the font-size of the *root element* (<html>). **This is the most recommended unit for sizing text and spacing** because it makes scaling the entire page (for accessibility) easy.

```
html {  
  /* This is the root font size */  
  font-size: 16px;  
}
```

```
h1 {  
  /* 2 * 16px = 32px */  
  font-size: 2rem;  
}
```

```
p {  
  /* 1 * 16px = 16px */  
  font-size: 1rem;  
  /* 1.5 * 16px = 24px */  
  margin-bottom: 1.5rem;  
}
```

5. Colors and Backgrounds

Colors

Colors can be specified in several ways:

```
.box {  
  /* Keyword */  
  color: red;  
  
  /* HEX (RRGGBB) */  
  color: #FF0000;  
  
  /* Short HEX (RGB) */  
  color: #FO0;
```

```
/* RGB (Red, Green, Blue) */
color: rgb(255, 0, 0);

/* RGBA (Red, Green, Blue, Alpha/Opacity)
/* 0.5 is 50% transparent */
color: rgba(255, 0, 0, 0.5);
}
```

Backgrounds

```
.hero-section {
background-color: #eee;

/* Use an image as a background */
background-image: url('images/hero-bg.jpg');

/* Prevent the image from repeating */
background-repeat: no-repeat;

/* Center the image */
background-position: center center;

/* Scale the image to cover the entire element */
background-size: cover;
/* 'contain' is another common value */
}
```

6. Layout

Controlling the position and flow of elements.

display Property

This is the most fundamental layout property. It determines how an element behaves.

- display: block;
 - Starts on a new line.
 - Takes up the full width available.
 - width, height, margin, padding all work.
 - Examples: <div>, <p>, <h1>,
- display: inline;

- Flows with text, does not start on a new line.
 - Takes up only as much width as its content.
 - width, height, and vertical margin/padding are **ignored**.
 - Examples: <a>, ,
- display: inline-block;
 - The best of both: Flows with text (inline) but...
 - You can set width, height, margin, and padding (block).
- display: none;
 - Hides the element completely. It's removed from the page flow.
- display: flex; (See Flexbox section)
- display: grid; (See Grid section)

position Property

Allows you to take elements out of the normal page flow.

- position: static; (Default)
 - The element is in the normal page flow. top, right, bottom, left have no effect.
- position: relative;
 - The element is positioned *relative to its normal static position*.
 - You can now use top, left, etc., to "nudge" it.
 - **Crucially, it creates a new positioning context** for its absolute children.
- position: absolute;
 - The element is *removed* from the normal flow.
 - It is positioned relative to its *nearest positioned ancestor* (i.e., the closest parent that has position set to anything *other than* static).
 - If no positioned ancestor is found, it's positioned relative to the <body>.
- position: fixed;
 - The element is *removed* from the normal flow.
 - It is positioned relative to the **browser viewport** (the window).
 - It stays in the same place even when you scroll. (Example: "sticky" headers).
- position: sticky;
 - A hybrid. It behaves like relative until it hits a specified threshold (e.g., top: 0), at which point it "sticks" and behaves like fixed.

Example:

```
.parent {
  position: relative;
  width: 300px;
  height: 300px;
  background-color: #eee;
}
```

```
.child {
```

```
position: absolute;  
top: 10px;  
right: 10px;  
width: 50px;  
height: 50px;  
background-color: red;  
}
```

*In this example, the red child box will be 10px from the top and 10px from the right **of its gray parent box**.*

7. Modern Layout: Flexbox

Flexbox is a 1-dimensional layout model for arranging items in rows or columns. To use it, you create a **flex container**.

```
<div class="flex-container">  
  <div class="item">1</div>  
  <div class="item">2</div>  
  <div class="item">3</div>  
</div>
```

Properties for the Container (.flex-container)

- `display: flex;` (Turns it on)
- `flex-direction: row;` (Default) | column;
 - Sets the main axis (how items are laid out).
- `justify-content: flex-start;` (Default)
 - Aligns items along the **main axis**.
 - Values: center, flex-end, space-between, space-around, space-evenly.
- `align-items: stretch;` (Default)
 - Aligns items along the **cross axis** (the one perpendicular to flex-direction).
 - Values: flex-start, flex-end, center, baseline.
- `flex-wrap: nowrap;` (Default) | wrap;
 - Allows items to wrap to the next line if they run out of space.

Properties for the Items (.item)

- `flex-grow: 0;` (Default)
 - A number defining how much an item can "grow" to fill extra space.
- `flex-shrink: 1;` (Default)
 - A number defining how much an item can "shrink" if there isn't enough space.
- `flex-basis: auto;` (Default)

- The "ideal" starting size of the item before growing or shrinking.
- flex: 0 1 auto; (Shorthand for flex-grow, flex-shrink, and flex-basis)
- order: 0;
 - Changes the visual order of items without changing the HTML.

Example: A perfectly centered item

```
.parent {
  display: flex;
  justify-content: center; /* Center horizontally */
  align-items: center;    /* Center vertically */
  height: 100vh;         /* Make parent fill the viewport height */
}

.child {
  /* This item will be centered */
}
```

8. Modern Layout: CSS Grid

Grid is a 2-dimensional layout model for rows and columns simultaneously.

```
<div class="grid-container">
  <div class="item">1</div>
  <div class="item">2</div>
  <div class="item">3</div>
  <div class="item">4</div>
</div>
```

Properties for the Container (.grid-container)

- display: grid; (Turns it on)
- grid-template-columns: 1fr 1fr 1fr;
 - Defines the columns. 1fr is one "fractional unit." This creates 3 equal-width columns.
 - repeat(3, 1fr) is the same as above.
 - 100px 1fr 2fr creates one 100px column, a second 1-fraction column, and a third 2-fraction column (twice as big as the second).
- grid-template-rows: 100px auto;
 - Defines the rows. First row is 100px tall, second is automatic.
- gap: 20px;
 - Shorthand for row-gap and column-gap. Sets the space between grid items.

Properties for the Items (.item)

You can "place" items onto the grid you've defined.

```
.item-1 {  
  /* Start on column line 1, end before column line 3 (spans 2) */  
  grid-column: 1 / 3;  
  
  /* Start on row line 1, end before row line 2 (spans 1) */  
  grid-row: 1 / 2;  
  
  /* A shorthand for the above */  
  grid-area: 1 / 1 / 2 / 3; /* (row-start / col-start / row-end / col-end) */  
}
```

9. Transitions and Animations

transition

Smoothly animates a property change when an element's state changes (like on :hover).

```
.button {  
  background-color: blue;  
  padding: 1rem 2rem;  
  color: white;  
  
  /* Apply a transition to the background-color property.  
   It will take 0.3 seconds and use an 'ease-out' timing. */  
  transition: background-color 0.3s ease-out;  
}  
  
.button:hover {  
  /* This change will be animated */  
  background-color: darkblue;  
}
```

animation and @keyframes

For more complex, multi-step animations.

1. **Define the animation** using @keyframes.
2. **Apply the animation** to an element.

```
/* 1. Define the animation */
```

```

@keyframes fadeIn {
  from {
    opacity: 0;
    transform: translateY(20px);
  }
  to {
    opacity: 1;
    transform: translateY(0);
  }
}

/* 2. Apply the animation */
.box-to-fade-in {
  /* name duration timing-function delay iteration-count direction */
  animation: fadeIn 1s ease-in-out;
}

```

10. Responsive Design (Media Queries)

Media queries allow you to apply CSS rules *only* if certain conditions are met, such as the screen width. This is the key to responsive design.

The "mobile-first" approach is common: style for mobile by default, then use min-width media queries to add styles for larger screens.

```

/* --- Default styles (Mobile-First) --- */
.container {
  width: 100%;
  padding: 10px;
}

.sidebar {
  display: none; /* Hide sidebar on mobile */
}

/* --- Tablet styles --- */
/* This block applies ONLY if the viewport is 768px wide or WIDER */
@media (min-width: 768px) {
  .container {
    width: 750px;
    margin: 0 auto; /* Center the container */
  }
}

```

```
.sidebar {  
    display: block; /* Show the sidebar */  
}  
}  
  
/* --- Desktop styles --- */  
/* This block applies ONLY if the viewport is 1024px wide or WIDER */  
@media (min-width: 1024px) {  
    .container {  
        width: 980px;  
    }  
}
```

11. Other Important Concepts

CSS Variables (Custom Properties)

Define reusable values. This is extremely powerful for themeing and maintenance.

```
/* 1. Define variables on the :root (<html>) element */  
:root {  
    --primary-color: #3498db;  
    --main-font: Arial, sans-serif;  
    --spacing-medium: 1.5rem;  
}  
  
/* 2. Use the variables */  
body {  
    font-family: var(--main-font);  
}  
  
.button {  
    background-color: var(--primary-color);  
    padding: var(--spacing-medium);  
}  
  
.button:hover {  
    /* You can even change variables inside other rules */  
    --primary-color: #2980b9;  
}
```

Specificity

How does a browser decide which rule "wins" if multiple rules target the same element? This is **specificity**.

The most specific selector wins. Here is the hierarchy from most to least specific:

1. **Inline styles** (style="...")
 2. **ID selectors** (#id)
 3. **Class selectors** (.class), **pseudo-classes** (:hover), **attribute selectors** ([type="text"])
 4. **Element selectors** (p), **pseudo-elements** (::before)
- !important can be added to a declaration (color: red !important;) to override *all* other rules, but this is **bad practice** and should be avoided.

```
/* This rule will win, even though the ID rule comes later */
```

```
p#my-paragraph {  
    color: red; /* (1 ID + 1 Element) */  
}
```

```
#my-paragraph {  
    color: blue; /* (1 ID) */  
}
```

```
p.highlight {  
    color: green; /* (1 Class + 1 Element) */  
}
```

overflow

Controls what happens when content is too big for its box.

- overflow: visible; (Default) Content spills out.
- overflow: hidden; Content is clipped and hidden.
- overflow: scroll; Content is clipped, and scrollbars are *always* added.
- overflow: auto; Content is clipped, and scrollbars are added *only if needed*.