CSS Basics – Quick Reference

This README covers the foundational ways to include CSS in a webpage and common selectors for targeting elements.

---

3 Ways To Include CSS

Internal CSS

CSS is written inside a `<style>` tag within the `<head>` of an HTML file.

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      p {
        color: blue;
      }
    </style>
  </head>
  <body>
    <p>This is a blue paragraph using internal CSS.</p>
  </body>
</html>
```

Inline CSS

CSS is applied directly to HTML elements using the `style` attribute.

```
<p style="color: blue;">This is a blue paragraph using inline CSS.</p>
```

External CSS

CSS is written in a separate file and linked to the HTML file.

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
```

```
    <p>This is a blue paragraph using external CSS.</p>
  </body>
</html>
```

style.css

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

CSS Selectors

◇ Class Selector

Targets elements with a specific class name.

```
.myClass {
  color: green;
}
```

```
<p class="myClass">This text is green using a class selector.</p>
```

◇ ID Selector

Targets a unique element with a specific ID.

```
#myId {
  font-size: 20px;
}
```

```
<p id="myId">This text uses an ID selector.</p>
```

Child Selector (>)

Targets **direct child** elements only.

```css
div > p {
  color: red;
}
```

```html
<div>
  <p>This paragraph is a direct child and will be red.</p>
  <section>
    <p>This paragraph is not a direct child and won't be red.</p>
  </section>
</div>
```

Descendant Selector (space)

Targets **all nested** elements inside a parent, regardless of depth.

```css
div p {
  color: orange;
}
```

```html
<div>
  <p>This paragraph is orange.</p>
  <section>
    <p>This nested paragraph is also orange.</p>
  </section>
</div>
```

Universal Selector (*)

Applies styles to **all elements** on the page.

```css
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}
```

Pseudo Selectors

Used to define special states or parts of elements.

```css
/* Style when mouse hovers over link */
a:hover {
  color: red;
}

/* Style the first paragraph inside an element */
p:first-child {
  font-weight: bold;
}

/* Style visited links */
a:visited {
  color: purple;
}

/* Style input fields when focused */
input:focus {
  border-color: blue;
}
```

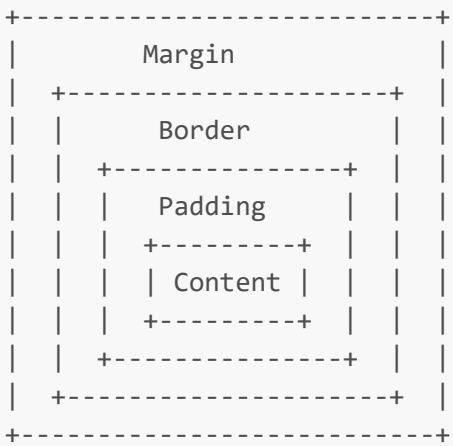
📦 CSS Box Model & Typography – Quick Reference

This section includes the CSS Box Model and important typography-related properties in CSS.

📦 CSS Box Model

The Box Model describes how every HTML element is represented as a rectangular box.

📊 Diagram of Box Model



🔠 Fonts in CSS

Font Style

```
font-style: italic; /* Text leans to the right */
font-style: normal; /* Default style */
font-style: oblique; /* Similar to italic with less tilt */
```

Font Weight

```
font-weight: normal; /* Default */
font-weight: bold; /* Bold text */
font-weight: 100 - 900; /* Numeric values for precision */
```

Font Family

```
font-family: 'Arial', sans-serif;
font-family: 'Times New Roman', serif;
```

✍️ Text Decoration

```
text-decoration: overline; /* Line above the text */
text-decoration: underline; /* Line below the text */
text-decoration: line-through; /* Line through the text */
text-decoration: none; /* No decoration */
```

📐 Text Alignment

```
text-align: left; /* Align to the left */
text-align: center; /* Align to the center */
text-align: right; /* Align to the right */
text-align: justify; /* Justify text across the line */
```

📊 Example

```
Left:     This is text
Center:       This is text
```

```
Right:           This is text
Justify: This   is   spaced   out   text
```

🔤 Text Transformation

```css
text-transform: uppercase; /* ALL CAPS */
text-transform: lowercase; /* all lowercase */
text-transform: capitalize; /* First Letter Capitalized */
text-transform: none; /* No transformation */
```

📊 Example

```
UPPERCASE -> HELLO WORLD
lowercase -> hello world
Capitalize -> Hello World
None -> Hello world
```

✨ Additional Typography Properties

Letter Spacing

```css
letter-spacing: 2px;
```

Example

```
Text: H  E  L  L  O
```

Line Height

```css
line-height: 1.6;
```

Example

```
Line 1: Hello

Line 2: World  <- More space between lines
```

Text Shadow

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

(Visual effect can't be shown in)

Text Overflow

```
text-overflow: ellipsis;
text-overflow: clip;
overflow: hidden;
white-space: nowrap;
```

📊 Example

```
Ellipsis: This is a long text...
Clip:     This is a long text
```

!Important > Inline Style > ID Selector > Class or Attribute Selector > Element Selector > Universal Selector

px em rem vh vw vmin vmax %

🧮 CSS Specificity & Units – Quick Reference

This section includes the **specificity hierarchy** in CSS and the **various units** used for measurements and sizing.

📊 CSS Specificity Hierarchy

When multiple CSS rules apply to the same element, the browser uses specificity to decide which one to apply. The order of precedence (from highest to lowest) is:

```
!important > Inline Style > ID Selector > Class/Attribute Selector > Element
Selector > Universal Selector
```

📏 CSS Units

CSS uses various units to measure length, spacing, and sizes. They can be either **absolute** or **relative**.

◇ Absolute Unit

- `px` – Pixels: Fixed unit, not responsive.

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

◇ Relative Units

- `em` – Relative to the **font-size of the parent**.

```
font-size: 2em; /* 2x parent size */
```

- `rem` – Relative to the **root element's font-size** (`html` tag).

```
font-size: 1.5rem;
```

- `%` – Relative to the **parent element**.

```
width: 50%; /* half the width of the parent */
```

- `vh` – Relative to **1% of the viewport height**.

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

- `vw` – Relative to **1% of the viewport width**.

```
width: 100vw; /* Full width of the viewport */
```

- `vmin` – Relative to the **smaller dimension** (height or width) of the viewport.

```
font-size: 10vmin;
```

- `vmax` – Relative to the **larger dimension** of the viewport.

```

```

```
padding: 5vmax;
```

🎨 Advanced CSS Styling – Quick Reference

This guide covers advanced CSS properties like `display`, `box-shadow`, `text-shadow`, `outline`, styling lists, and controlling overflow.

🖼️ Display Property

The `display` property defines how an element is displayed in the document. It is one of the most important properties in CSS.

Common Values:

- `block`: The element takes up the full width and starts on a new line.
- `inline`: The element does not start on a new line and only takes up as much width as necessary.
- `inline-block`: Behaves like `inline` but allows setting width and height.
- `none`: Hides the element from the layout (it's not rendered).
- `flex`: Enables Flexbox layout for flexible layouts.
- `grid`: Enables CSS Grid layout.
- `inline-flex` / `inline-grid`: Same as above but behave like inline elements.
- `contents`: Makes the container disappear, making the child elements act as if they're not wrapped.
- `table`: Makes the element behave like a `<table>`.
- `list-item`: Makes the element behave like a `<li>`.

```
.container {
  display: flex;
  justify-content: center;
}
```

🌫️ Shadows & Outlines

■ Box Shadow

Adds shadow effects around an element's frame.

```
box-shadow: h-offset v-offset blur spread color inset;
```

- **h-offset**: Horizontal distance
- **v-offset**: Vertical distance
- **blur**: Blur radius

- **spread**: Size of the shadow
- **color**: Shadow color
- **inset**: Makes the shadow inner

```
box-shadow: 4px 4px 10px 2px rgba(0, 0, 0, 0.2);
```

✍️ Text Shadow

Adds shadow to text for visual depth or clarity.

```
text-shadow: h-offset v-offset blur color;
```

```
text-shadow: 2px 2px 4px #888888;
```

🖉 Outline

Outlines are lines drawn outside the border edge. Commonly used for highlighting focusable elements.

```
outline: width style color;
```

- **width**: e.g., `2px`
- **style**: `solid`, `dotted`, `dashed`, `double`, etc.
- **color**: e.g., `red`

```
outline: 2px dashed red;
```

> 🤓 Unlike `border`, outlines do not take up space in layout and can overlap content.

☑ Styling Lists

You can use `list-style-type`, `list-style-position`, and `list-style-image` for customizing lists.

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

- `list-style-type`: `disc`, `circle`, `square`, `decimal`, etc.
- `list-style-position`: `inside`, `outside`
- `list-style-image`: Use an image as the bullet.

📜 CSS Overflow

Controls what happens when content overflows an element's box.

```
overflow: visible | hidden | scroll | auto;
```

- `visible`: Default. Content is not clipped.
- `hidden`: Overflowing content is hidden.
- `scroll`: Always adds a scrollbar.
- `auto`: Scrollbar only added when needed.

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

📍 CSS Position Property

The `position` property specifies how an element is positioned in the document. It works in conjunction with the `top`, `right`, `bottom`, and `left` properties.

◇ static (default)

- Elements are positioned according to the normal document flow.
- `top`, `right`, `bottom`, `left`, and `z-index` have **no effect**.

```
.element {
  position: static;
}
```

◇ relative

- Element is positioned relative to its **normal** position.
- `top`, `right`, `bottom`, `left` move the element **from its original location**.

```
.element {
  position: relative;
  top: 10px;
```

```
  left: 5px;
}
```

◇ absolute

- Positioned relative to the **nearest positioned ancestor** (non-static).
- If no such ancestor, it is positioned relative to the `<html>` element.
- It is removed from normal document flow.

```
.element {
  position: absolute;
  top: 0;
  left: 0;
}
```

◇ fixed

- Positioned relative to the **viewport**.
- Stays fixed when the page is scrolled.

```
.element {
  position: fixed;
  bottom: 0;
  right: 0;
}
```

◇ sticky

- A hybrid of `relative` and `fixed`.
- Element toggles between `relative` and `fixed` depending on scroll position.

```
.element {
  position: sticky;
  top: 20px;
}
```

✨ z-index

- Controls the **stacking order** of positioned elements.
- Higher value = closer to the front.

```css
.element {
  position: absolute;
  z-index: 10;
}
```

🎯 CSS Variables (Custom Properties)

CSS variables allow you to define reusable values across your CSS stylesheets.

◇ Syntax

Define a variable:

```css
:root {
  --main-color: #3498db;
  --padding: 20px;
}
```

Use a variable:

```css
.box {
  background-color: var(--main-color);
  padding: var(--padding);
}
```

- Variables are case-sensitive.
- They cascade like normal CSS properties and can be overridden within specific scopes.

📲 Media Queries

Media queries are used to apply CSS rules based on device characteristics such as screen width, height, orientation, and more.

◇ Media Query Syntax

```css
@media not|only mediatype and (expression) {
  /* CSS code */
}
```

☑ Example

```
@media only screen and (min-width: 444px) {
  .boxes {
    background-color: purple;
  }
}
```

📏 min-width

```
@media (min-width: 500px) {
  /* styles apply */
}
```

- ☑ Applied when screen width is **500px or more**
- ❌ Ignored if screen is **less than 500px**

📐 max-width

```
@media (max-width: 500px) {
  /* styles apply */
}
```

- ☑ Applied when screen width is **500px or less**
- ❌ Ignored if screen is **more than 500px**

🖥️ Common Media Types

- `all` – Suitable for all devices.
- `print` – For printed material or print preview.
- `screen` – For computer screens, tablets, smartphones.

📦 Float and Clear

◇ float

The `float` property places an element to the left or right of its container, allowing text and inline elements to wrap around it.

```
img {
  float: left;
}
```

Values:

- `left` – Floats the element to the left
- `right` – Floats the element to the right
- `none` – Default. Element does not float
- `inherit` – Inherits float value from parent

◇ clear

The `clear` property prevents an element from wrapping around floating elements.

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

Values:

- `left` – No floating elements allowed on the left
- `right` – No floating elements allowed on the right
- `both` – No floating elements allowed on either side
- `none` – Allows floating elements on both sides

☑ Use a clearfix to contain floats inside a container:

```
.clearfix::after {
  content: '';
  display: table;
  clear: both;
}
```

🎯 Advanced CSS Selectors – Cheat Sheet

🧠 Pseudo-Classes

`:first-child`

Selects the first child of a parent element.

```
.box:first-child {
  background-color: red;
}
```

`:nth-child(odd)`

Selects every odd child element (1st, 3rd, 5th, ...).

```
.box:nth-child(odd) {
  background-color: blue;
}
```

`:nth-last-child(2)`

Selects the second element from the end.

```
.box:nth-last-child(2) {
  background-color: red;
}
```

🎭 Pseudo-Elements

`::first-line`

Targets the first line of text in an element.

```
.box::first-line {
  color: yellow;
}
```

`::first-letter`

Targets the first letter of the text in an element.

```
.box::first-letter {
  color: peru;
  font-size: 45px;
}
```

`::before` and `::after`

Used to insert content before or after the content of an element.

```
.boxes::before {
  content: 'Harry is good';
  color: blue;
}

.boxes::after {
  content: 'Sigma course is also good';
  color: red;
}
```

`::selection`

Applies styles to the portion of an element that is selected by the user.

```
::selection {
  background-color: black;
  color: aqua;
}
```

`::placeholder`

Styles placeholder text in `<input>` and `<textarea>`.

```
input::placeholder {
  color: rgb(84, 84, 88);
}
```

Group Selectors and Universal Selectors

Universal Selector `*`

Targets all elements inside `.boxes`.

```
.boxes * {
  color: blue;
  border: 2px solid black;
}
```

Group Selector `p, a, .box, [data-color="primary"]`

Applies styles to multiple elements and attributes.

```
p,
a,
.box,
[data-color='primary'] {
  padding-top: 45px;
}
```

The **Flexbox Layout (Flexible Box)** module makes it easier to design flexible responsive layout structures without using float or positioning.

🔧 Flex Container Properties

☑ `display: flex;`

Defines a flex container.

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

☑ `flex-direction`

Defines the direction of the **main axis**.

- `row` – Default, left to right.
- `row-reverse` – Right to left.
- `column` – Top to bottom.
- `column-reverse` – Bottom to top.

```
.container {
  flex-direction: column;
}
```

☑ `flex-wrap`

Controls whether flex items wrap or not.

- `nowrap` – Default, all items on one line.
- `wrap` – Items wrap onto multiple lines.
- `wrap-reverse` – Wrap in reverse order.

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

☑ flex-flow

Shorthand for `flex-direction` and `flex-wrap`.

```
.container {
  flex-flow: row wrap;
}
```

☑ justify-content

Aligns items **horizontally** along the **main axis**.

- `flex-start` | `flex-end`
- `center`
- `space-between`
- `space-around`
- `space-evenly`

```
.container {
  justify-content: space-between;
}
```

☑ align-items

Aligns items **vertically** along the **cross axis**.

- `stretch` (default)
- `flex-start` | `flex-end`
- `center`
- `baseline`

```
.container {
  align-items: center;
}
```

☑ align-content

Aligns **rows of items** when there's extra space on the **cross axis**.

- `flex-start` | `flex-end`
- `center`
- `space-between`
- `space-around`
- `stretch`

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

🎯 Flex Item Properties

◇ `order`

Defines the order of the item (default: 0). Lower values appear first.

```
.item {
  order: 2;
}
```

◇ `flex-grow`

Defines how much a flex item will grow relative to the rest.

```
.item {
  flex-grow: 1;
}
```

◇ `flex-shrink`

Defines how a flex item shrinks when space is tight.

```
.item {
  flex-shrink: 1;
}
```

◇ `flex-basis`

Defines the initial size of a flex item **before** remaining space is distributed.

```
.item {
  flex-basis: 200px;
}
```

◇ `flex`

Shorthand for `flex-grow`, `flex-shrink`, and `flex-basis`.

```
.item {
  flex: 1 1 100px;
}
```

◇ `align-self`

Overrides `align-items` for individual items.

- `auto` (default)
- `flex-start` | `flex-end`
- `center`
- `baseline`
- `stretch`

```
.item {
  align-self: flex-end;
}
```

📊 CSS Grid – Complete Cheat Sheet

CSS Grid Layout is a two-dimensional layout system for the web. It lets you layout items in rows and columns easily.

🔧 Grid Container Properties

☑ `display: grid;` / `inline-grid`

Turns an element into a grid container.

```
```

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

☑ `grid-template-columns` / `grid-template-rows`

Defines column/row sizes and counts.

```
.container {
  grid-template-columns: 200px 1fr auto;
  grid-template-rows: 100px 50px;
}
```

☑ `gap`, `row-gap`, `column-gap`

Sets spacing between rows/columns.

```
.container {
  gap: 10px;
}
```

☑ `grid-template-areas`

Creates named layout areas for easy positioning.

```
.container {
  grid-template-areas:
    'header header'
    'sidebar main'
    'footer footer';
}
```

☑ `grid-auto-rows`, `grid-auto-columns`

Sets size of automatically created tracks.

```
.container {
  grid-auto-rows: 100px;
}
```

☑ `grid-auto-flow`

Controls auto-placement direction.

```
.container {
  grid-auto-flow: row; /* or column */
}
```

🎯 Grid Item Properties

◇ `grid-column` / `grid-row`

Defines start and end lines.

```
.item {
  grid-column: 1 / 3;
  grid-row: 2 / 4;
}
```

◇ `grid-column-start`, `grid-column-end`

Control which grid lines the item spans.

◇ `grid-area`

Assigns an item to a named grid area.

```
.item {
  grid-area: header;
}
```

◇ `justify-self`, `align-self`, `place-self`

Align a single item in the cell:

- `start`, `end`, `center`, `stretch`

🔄 Grid Container Alignment

- `justify-items`: Aligns all items horizontally.

- `align-items`: Aligns all items vertically.
- `place-items`: Shorthand of both.
- `justify-content`: Aligns the grid inside the container horizontally.
- `align-content`: Aligns vertically.
- `place-content`: Shorthand for both.

Repeat Syntax

```
grid-template-columns: repeat(3, 1fr);
```

Unit: `fr`

Represents a fraction of the available space.

```
grid-template-columns: 1fr 2fr;
```

Example

```
.container {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  gap: 20px;
}

.item {
  grid-column: span 2;
}
```

CSS Transform Notes

CSS Transform allows you to visually manipulate elements by skewing, rotating, translating, or scaling.

Syntax

```
transform: function(value);
```

Multiple transform functions can be chained using space:

```
transform: rotate(30deg) scale(1.5);
```

◇ 2D Transform Functions

Function	Description
`scaleX(n)`	Scales the element along the X-axis
`scaleY(n)`	Scales the element along the Y-axis
`scale(x, y)`	Scales on both X and Y axes
`translateX(n)`	Moves the element along the X-axis
`translateY(n)`	Moves along Y-axis
`translate(x, y)`	Moves along both X and Y
`rotate(angle)`	Rotates the element clockwise by given degree around the Z-axis
`skew(x-angle, y-angle)`	Skews along X and Y
`skewX(angle)`	Skews only along X-axis
`skewY(angle)`	Skews only along Y-axis
`matrix(a, b, c, d, e, f)`	Applies a 2D matrix transformation

◇ 3D Transform Functions

Function	Description
`translate3d(x, y, z)`	Moves along all three axes
`translateZ(z)`	Moves along Z-axis
`rotateX(angle)`	Rotates around X-axis
`rotateY(angle)`	Rotates around Y-axis
`rotate3d(x, y, z, angle)`	Rotates in 3D space
`scaleZ(z)`	Scales along Z-axis
`matrix3d(...)`	3D transformation matrix (16 values)

Additional Notes

- Use `transition` to animate transformations smoothly.

```
transition: transform 0.5s ease;
```

- To enable 3D effects, set `perspective` on the parent:

```
.parent {
  perspective: 500px;
}
```

📌 Example Visual Description

ScaleX:

Expands the element width-wise on hover.

TranslateY:

Moves the element downward when hovered.

Rotate:

Spins the element clockwise (e.g., 270 degrees).

Matrix3D:

A complex transformation combining scale, rotate, skew, and translate in 3D.

CSS Transitions

CSS Transitions enable you to change property values smoothly (over a given duration).

Syntax

```
selector {
  transition-property: property_name;
  transition-duration: time;
  transition-timing-function: easing_function;
  transition-delay: time;
}
```

Properties

- **transition-property**: Specifies the name of the CSS property the transition effect is for. Use `all` to apply to all.
- **transition-duration**: Defines how long the transition takes to complete.

- **transition-timing-function**: Describes how the intermediate values of the transition are calculated.
- **transition-delay**: Defines a delay before the transition starts.

Shorthand Syntax

```
transition: property duration timing-function delay;
```

Example:

```
div {
  transition: all 1s ease-in 1s;
}
```

Common Timing Functions

- `ease`: Starts slow, speeds up, then slows down (default).
- `linear`: Constant speed.
- `ease-in`: Starts slow and speeds up.
- `ease-out`: Starts fast and slows down.
- `ease-in-out`: Starts and ends slowly.
- `cubic-bezier(n,n,n,n)`: Custom timing.

CSS Animations

CSS Animations make it possible to animate transitions from one CSS style configuration to another.

Required Properties

- **@keyframes**: Define the animation.
- **animation-name**: Name of the @keyframes animation.
- **animation-duration**: How long the animation takes.

Other Properties

- **animation-timing-function**
- **animation-delay**
- **animation-iteration-count**
- **animation-direction**
- **animation-fill-mode**
- **animation-play-state**

Syntax

```
@keyframes example {
  from {
```

```
    background-color: red;
  }
  to {
    background-color: yellow;
  }
}

div {
  animation-name: example;
  animation-duration: 4s;
}
```

Animation Shorthand

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

Example:

```
animation: example 4s ease-in-out 1s 3 alternate both running;
```

Common Values

- **animation-iteration-count**: Number of repetitions (or `infinite`).
- **animation-direction**: `normal`, `reverse`, `alternate`, `alternate-reverse`.
- **animation-fill-mode**: Controls styles applied before/after animation (`none`, `forwards`, `backwards`, `both`).
- **animation-play-state**: `running` or `paused`.

CSS: Object Fit, Object Position & Background Properties

`object-fit`

- Defines how a replaced element's content (like `<img>`) should be resized to fit its container.

Values:

- `fill` (default): Stretch to fill the content box, may be distorted.
- `contain`: Scales to maintain aspect ratio while fitting within the content box.
- `cover`: Scales to maintain aspect ratio while covering the entire content box.
- `none`: Keeps original size.
- `scale-down`: Compares `none` and `contain` and applies the smaller result.

Example:

```
img {
  object-fit: cover;
}
```

---

object-position

- Specifies the alignment of the content within a replaced element when `object-fit` is used.

Syntax:

```
object-position: x y;
```

- Values like `top`, `left`, `center`, `right`, `bottom`, or percentages.

Example:

```
img {
  object-fit: cover;
  object-position: center center;
}
```

Background Properties

background-image

Sets one or more background images.

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

background-position

Sets the starting position of a background image.

```
background-position: center center;
```

- Can also use values like `top`, `bottom`, `left`, `right`, percentages or lengths.

background-repeat

Defines how background images repeat.

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

Values:

- `repeat` (default)
- `repeat-x`
- `repeat-y`
- `no-repeat`

`background-clip`

Defines how far the background extends within an element.

```
background-clip: border-box;
```

Values:

- `border-box` (default)
- `padding-box`
- `content-box`

Example Combining All:

```
.box {
  background-image: url('bg.jpg');
  background-position: center center;
  background-repeat: no-repeat;
  background-clip: content-box;
}
```

Visual Tip: Use a box with fixed width and height to easily demonstrate how `object-fit` and background properties behave visually with different content.

🎨 CSS Filters Overview

CSS Filters allow you to apply graphical effects such as blurring, color shifting, and more to elements. These effects can enhance the visual presentation of images, backgrounds, borders, and other elements.

🔧 `filter` Property

The `filter` property applies effects directly to the element, including its contents, borders, and padding.

Syntax:

```
filter: none | <filter-function> [<filter-function>] * | url();
```

Example:

```
img {
  filter: blur(5px) brightness(1.2);
}
```

`backdrop-filter` Property

The `backdrop-filter` property applies effects to the area behind an element. It's commonly used with semi-transparent backgrounds to create a frosted-glass effect.

Syntax:

```
backdrop-filter: none | <filter-function> [<filter-function>] * | url();
```

Example:

```
.modal {
  backdrop-filter: blur(10px) brightness(0.8);
}
```

CSS Filter Functions

Below is a list of available filter functions:

- **`blur(radius)`**: Applies a Gaussian blur.
  Example: `blur(5px)`
- **`brightness(amount)`**: Adjusts brightness. Values >1 increase brightness, values <1 decrease it.
  Example: `brightness(1.5)`
- **`contrast(amount)`**: Adjusts contrast. Values >1 increase contrast, values <1 decrease it.
  Example: `contrast(120%)`
- **`drop-shadow(offset-x offset-y blur-radius color)`**: Applies a drop shadow.
  Example: `drop-shadow(4px 4px 10px rgba(0,0,0,0.5))`
- **`grayscale(amount)`**: Converts the image to grayscale.
  Example: `grayscale(50%)`

- `hue-rotate(angle)`: Rotates the hue of the image.
  Example: `hue-rotate(90deg)`
- `invert(amount)`: Inverts the colors.
  Example: `invert(100%)`
- `opacity(amount)`: Adjusts the opacity.
  Example: `opacity(75%)`
- `saturate(amount)`: Saturates the colors.
  Example: `saturate(2)`
- `sepia(amount)`: Applies a sepia filter.
  Example: `sepia(60%)`

🧩 Combining Multiple Filters

Multiple filter functions can be combined by separating them with spaces. They are applied in the order they are listed.

Example:

```
img {
  filter: grayscale(50%) blur(2px) brightness(1.2);
}
```

📓 Additional Notes

- **SVG Filters**: You can reference SVG filters using the `url()` function.
  Example: `filter: url('filters.svg#myFilter');`
- **Performance**: Using filters can impact performance, especially on large elements or complex filter chains.
- **Browser Support**: Most modern browsers support CSS filters, but always check compatibility, especially for `backdrop-filter`, which may have limited support or require vendor prefixes.
