

Topic 8: Understanding the Functionality of CSS

1. Writing CSS (Cascading Style Sheet)

A. Purpose of CSS

- **Control Layout:** CSS separates content (HTML) from presentation, letting you position elements (headers, sidebars, footers) exactly where you want them.
 - **Consistent Design:** Define fonts, colors, spacing, and other styles in one place so your entire site shares the same “look and feel.”
-

B. Referencing CSS in Your Pages

1. Inline Styles

```
<p style="color: red; font-size: 16px;">Hello!</p>
```

– Quick but hard to maintain; overrides other rules.

2. **<style> Block in <head>**

```
<head>
  <style>
    p { color: red; }
  </style>
</head>
```

– Good for page-specific styles; keeps CSS separate from content.

3. **External .css File**

```
html
CopyEdit
<head>
  <link rel="stylesheet" href="css/styles.css">
</head>
```

– Best practice for large sites; one file can style hundreds of pages.

Best Practices:

- Organize CSS in a `/css/` folder.
- Use clear naming (e.g., `styles.css`, `layout.css`).
- Reference with **relative paths** (`href="css/styles.css"`), not absolute URLs, so links don't break when you move your site.

C. CSS Rule Syntax

```
selector {  
  property: value;  
  /* multiple declarations separated by semicolons */  
}
```

- **Selectors:** Target the HTML elements you want to style.
- **Properties:** The aspect you're changing (e.g., `color`, `margin`).
- **Values:** The setting for that property (e.g., `#333`, `20px`).

D. Selectors

1. Type Selector:

```
p { ... }      /* all <p> elements */
```

2. Class Selector:

```
.highlight { ... } /* any element with class="highlight" */
```

3. ID Selector:

```
#main-header { ... } /* element with id="main-header" */
```

4. Universal Selector:

```
* { margin: 0; padding: 0; } /* every element */
```

5. Attribute Selector:

```
a[target="_blank"] { ... }
```

6. Combinators:

◦ Child (>):

```
ul > li { ... } /* only direct <li> children of <ul> */
```

- **Descendant (space):**

```
nav a { ... } /* any <a> inside <nav> */
```

- **Adjacent Sibling (+):**

```
h2 + p { ... } /* the <p> immediately after an <h2> */
```

- **General Sibling (~):**

```
h2 ~ p { ... } /* any <p> after an <h2> on the same level */
```

7. Multiple Selectors:

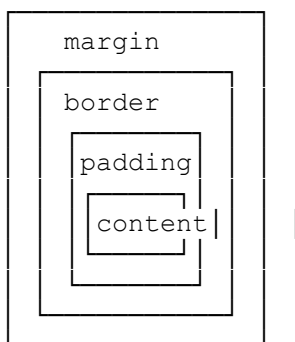
```
h1, h2, h3 { ... } /* apply to all listed elements */
```

E. Efficiency: Cascade & Inheritance

- **Cascade:** When multiple rules apply, the one with **higher specificity** (ID > class > type) wins. Later rules override earlier ones if they have equal specificity.
- **Inheritance:** Some properties (like color and font-family) are passed down from parent elements to children, reducing the need to redeclare them on every element.

F. The Box Model

Every element is a rectangular box composed of:



- **Content:** Your text or images.
- **Padding:** Space inside the box around the content.
- **Border:** The line around padding and content.
- **Margin:** Space outside the border, separating elements.

G. Attribute Selectors

- **Existence:**

```
[disabled] { opacity: 0.5; }
```

- **Equality:**

```
[type="text"] { width: 200px; }
```

- **Space-separated (~=):**

```
[class~="featured"] { font-weight: bold; }
```

- **Prefix (^=):**

```
[href^="https"] { color: green; }
```

- **Substring (*=):**

```
[title*="chapter"] { text-decoration: underline; }
```

- **Suffix (\$=):**

```
[src$=".png"] { border: 1px solid #ccc; }
```

Wrap-Up

- **CSS** is your tool for making websites look great and behave consistently.
- Master **selectors**, the **cascade**, and the **box model** to build flexible, maintainable styles.
- Use **external stylesheets** and **best practices** in organization to scale your designs across entire sites.

2. Styling Web Pages

A. Colours

1. Named Colours

```
color: red;
background-color: lightblue;
```

2. Hex Codes

```
color: #ff0000;    /* red */
background-color: #add8e6; /* lightblue */
```

3. RGB

```
color: rgb(255, 0, 0);
background-color: rgb(173, 216, 230);
```

4. HSL

```
color: hsl(0, 100%, 50%);    /* red */
background-color: hsl(195, 53%, 79%); /* lightblue */
```

B. Colour Manipulation

1. Opacity

```
/* RGBA adds alpha (0.0-1.0) */
background-color: rgba(255, 0, 0, 0.5); /* semi-transparent red */
opacity: 0.8; /* applies to entire element */
```

2. Gradients

```
/* Linear gradient */
background-image: linear-gradient(to right, #ff7e5f, #feb47b);

/* Radial gradient */
background-image: radial-gradient(circle, #ff7e5f, #feb47b);
```

C. Lengths

1. Absolute Units

- px (pixels): precise control, not scalable.
- cm, mm, in: physical measurements—rarely used on screen.

2. Relative Units

- %: percentage of parent element.
- em: relative to the font-size of the element (2em = 2 × current font-size).
- rem: relative to the root (html) font-size—easier for consistent scaling.

```
font-size: 1.2rem; /* 1.2 × root font-size */
width: 50%;        /* half the width of its container */
```

D. Element Styles

1. Text Layout & Fonts

```
body {
  font-family: Arial, sans-serif;
  line-height: 1.6;
  text-align: justify;
}
```

2. Links

```
a:link { color: blue; }
a:visited { color: purple; }
a:hover { color: red; text-decoration: underline; }
a:active { color: darkred; }
```

3. Lists

```
ul { list-style-type: disc; margin-left: 20px; }
ol { list-style-type: decimal; }
li { margin-bottom: 0.5em; }
```

4. Tables

```
table {
  border-collapse: collapse;
  width: 100%;
}
th, td {
  border: 1px solid #ccc;
  padding: 8px;
  text-align: left;
}
```

5. Forms

```
input, textarea, select {
```

```

    font-size: 1rem;
    padding: 0.5em;
    border: 1px solid #888;
    border-radius: 4px;
}
button {
    background-color: #28a745;
    color: white;
    padding: 0.6em 1.2em;
    border: none;
    border-radius: 4px;
    cursor: pointer;
}
button:hover { background-color: #218838; }

```

6. Images

```

img {
    max-width: 100%; /* responsive */
    height: auto;
    display: block;
    margin: 0 auto;
}

```

E. Box Model Enhancements

1. Background Images & CSS Sprites

```

.hero {
    background-image: url('hero.jpg');
    background-size: cover;
    background-position: center;
}
.icon {
    background: url('sprite.png') no-repeat;
    /* use background-position to show the right icon */
}

```

2. Gradients (revisited)

```

.banner {
    background: linear-gradient(45deg, #6a11cb, #2575fc);
}

```

3. Borders

```

.card {
    border: 2px solid #333;
    border-radius: 8px;
    padding: 1em;
    box-shadow: 0 2px 5px rgba(0,0,0,0.2);
}

```

Tip: Practice combining these properties to create visually appealing, responsive layouts. Understanding these fundamentals will let you build consistent, maintainable designs across your entire site.

3. Positioning Content

A. Positioning Methods

1. Normal Flow

- Default document layout: block-level elements stack vertically; inline elements flow within lines.
- **No CSS positioning** required.

2. Relative Positioning

```
.relative-box {  
  position: relative;  
  top: 10px;    /* moves down 10px from its normal spot */  
  left: 20px;   /* moves right 20px */  
}
```

- Offsets an element **relative to its normal position** without affecting document flow for other elements.

3. Absolute Positioning

```
.absolute-box {  
  position: absolute;  
  top: 50px;    /* relative to nearest positioned ancestor */  
  right: 10px;  
}
```

- Removes the element from normal flow; placed **relative to the nearest ancestor** with `position: relative|absolute|fixed`.

4. Fixed Positioning

```
.fixed-header {  
  position: fixed;  
  top: 0;  
  left: 0;  
  width: 100%;  
}
```

- Sticks the element to the **viewport**; it doesn't move when scrolling.

5. Floats

```
.float-img {  
  float: left;  
  margin: 0 1em 1em 0;  
}
```

- Allows text to **wrap** around the element. Remember to **clear** floats on parent containers.

6. Overlapping Elements (z-index)

```
.layer1 { position: relative; z-index: 1; }  
.layer2 { position: relative; z-index: 2; } /* sits above layer1 */
```

- z-index controls stacking order for positioned elements.

B. Box Model Control

1. Display Types

- `display: block;` (fills width, starts on new line)
- `display: inline;` (flows within text)
- `display: inline-block;` (inline but accepts width/height)

2. Sizing & Spacing

```
.box {  
  width: 200px;  
  height: 150px;  
  margin: 20px;      /* outside space */  
  padding: 10px;     /* inside space */  
  border: 2px solid #333;  
}
```

C. Responsive Layouts

1. Fixed-Width vs. Liquid Layouts

- **Fixed-Width:** Uses absolute units (e.g., `width: 960px;`), looks consistent but may overflow small screens.
- **Liquid (Fluid):** Uses relative units (e.g., `width: 80%;`), adapts to viewport width.

2. Layout Grids & CSS Frameworks

- **CSS Grid / Flexbox:** Native layout modules for responsive columns and rows.

- **Frameworks (Bootstrap):** Predefined grid classes (e.g., `.col-md-6`) that adjust at breakpoints.
-

D. Responsive Techniques

1. Media Queries

```
/* Mobile-first: base styles for small screens */
.sidebar { display: none; }

/* At 768px and up, show sidebar */
@media (min-width: 768px) {
  .sidebar { display: block; width: 250px; }
}
```

2. Flexible Units

- % for width/height relative to parent.
- em/rem for font-size and spacing relative to element or root font-size.
- vw/vh for viewport width/height.

3. Mobile-First Design

- Start with **small-screen styles**, then add media queries for larger viewports.
- Ensures performance and usability on mobile devices before enhancing for desktops.

Tip: Mix positioning methods and box-model adjustments with responsive techniques to build layouts that work across devices—float images on desktop but stack them on mobile, fix headers for easy navigation, and use media queries to reflow content fluidly.

C. Keyframe Animations

Defining Keyframes:

```
@keyframes slide-in {  
  0%   { transform: translateX(-100%); opacity: 0; }  
  50%  { opacity: 1; }  
  100% { transform: translateX(0); }  
}
```

Applying Animations:

```
.element {  
  animation-name: slide-in;  
  animation-duration: 2s;  
  animation-timing-function: ease-out;  
  animation-delay: 0.5s;  
  animation-iteration-count: infinite; /* or a number */  
  animation-direction: alternate; /* normal | reverse | alternate */  
  animation-fill-mode: forwards; /* none | forwards | backwards | both */  
}
```

5. Summary and Review (5 minutes)

Recap Key Concepts

1. **CSS Referencing:** Inline, `<style>` blocks, external files.
2. **Selectors & Specificity:** Type, class, ID, combinators; cascade & inheritance.
3. **Box Model:** Content → padding → border → margin.
4. **Styling Elements:** Colours, fonts, lists, tables, forms, images.
5. **Positioning & Layout:** Static flow, relative/absolute/fixed, floats, responsive grids, media queries.
6. **Animations:** Transitions, transforms, keyframes.

Quick Q&A

- “How does the cascade affect CSS rules?”

Later rules and more specific selectors override earlier or less specific ones.

- **“When would you use `position: absolute;`?”**

To remove an element from normal flow and place it at a precise location relative to its nearest positioned ancestor.

Additional Resources

Videos

1. **CSS Crash Course**

A fast-paced introduction to core CSS concepts and techniques.

 <https://www.youtube.com/watch?v=yfoY53QXEnI>

2. **Responsive Web Design with CSS Grid and Flexbox**

Covers modern layout methods for building responsive sites.

 <https://www.youtube.com/watch?v=0xMQfnTU6oo>

3. **CSS Animations Tutorial**

Step-by-step guide to creating smooth, performant animations.

 https://www.youtube.com/watch?v=Edsxf_NBFrw

Websites

- **MDN Web Docs (CSS)**

Comprehensive reference and guides on every CSS feature.

 <https://developer.mozilla.org/en-US/docs/Web/CSS>

- **CSS-Tricks**

Articles, “almanac” reference, and real-world examples.

 <https://css-tricks.com>

- **W3Schools CSS Tutorial**

Interactive examples and “Try It Yourself” editor.

 <https://www.w3schools.com/css/>

Reading Material

1. “CSS3: The Definitive Guide” by Eric A. Meyer & Estelle Weyl

In-depth coverage of CSS3 features and best practices.

 <https://www.oreilly.com/library/view/css3-the-definitive/9781449325053/>

2. MDN Guides on Layout & Animations

- Flexbox: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Basic_Concepts_of_Flexbox
- Grid: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout
- Animations: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Animations/Using_CSS_animations

3. CSS-Tricks “Complete Guides”

- Flexbox: <https://css-tricks.com/snippets/css/a-guide-to-flexbox/>
- Grid: <https://css-tricks.com/snippets/css/complete-guide-grid/>
- Animations: <https://css-tricks.com/almanac/properties/a/animation/>

4. Online Tutorials

- freeCodeCamp’s CSS curriculum: <https://www.freecodecamp.org/learn/responsive-web-design/>
- Codecademy’s CSS courses: <https://www.codecademy.com/learn/learn-css>