**🎨 2. CSS Properties**

**📌 2.1 Introduction to CSS Properties**

👉 In our last class, we learned that a **CSS rule** consists of two main parts:

1. **Selector** → tells the browser *which HTML element(s)* to style
2. **Declaration block** → contains the actual styling instructions inside { }

✅ Example:

h1 {

color: blue;

}

* h1 → **selector**
* { color: blue; } → **declaration block**
* color → **property**
* blue → **value**

✨ Notes:

* The **selector** chooses which elements to style.
* The **property** defines *what aspect* of the element you want to change (color, size, margin, etc.).
* The **value** defines *how* that aspect should look.
* ⚠️ If there’s any **syntax error** in a rule, the **entire rule is ignored** by the browser.

📚 Reference: [CSS Properties List (MDN)](https://developer.mozilla.org/en-US/docs/Web/CSS/Reference)

**📌 2.2 Introduction to the Most Common CSS Properties**

* Think of **CSS properties as words in a language**.
  + You don’t need to know every word in the dictionary to speak.
  + The more you know, the better.
  + The key is to **effectively use the ones you do know**.

✨ The most important CSS properties to start with:

* display
* width
* height
* margin
* padding
* color
* background
* font

📚 Learn more: [9 Important CSS Properties You Must Know](https://zellwk.com/blog/9-important-css-properties-you-must-know/)

**📌 2.3 Display Property**

The **display property** determines **how an element is shown in the browser**.

* It sets whether an element behaves as a **block** or **inline** element by default.
* Most elements are **block-level** unless specified otherwise.

**🔑 Common display values:**

* **block**
  + Makes an element behave like a block-level element.
  + Starts on a **new line** and takes up the **full width** available.  
    ✅ Example:
* span {
* display: block;
* }

This makes <span> elements act like blocks, starting on a new line and stretching horizontally.

* **inline**
  + Makes an element behave like an inline element.
  + Appears **within a line** and only takes up as much space as needed.  
    ✅ Example:
* p {
* display: inline;
* }

This forces all <p> elements to appear on the same line.

* **inline-block**
  + Behaves like an inline element **but** allows you to set width and height.
  + Very useful for styling buttons or small layout elements.
* **none**
  + The element is **completely removed** from the page (not just hidden).
  + The browser acts like it doesn’t exist.
* **flex**
  + Turns the element into a **flex container**.
  + Automatically arranges its children (flex items).
  + Items can **expand** to fill space or **shrink** to prevent overflow.
  + Perfect for building **responsive layouts**.

**🎨 2.4 Position Property**

👉 The **position property** sets **how an element is positioned** in a document.

* Once the position is set, you can use **top**, **bottom**, **left**, and **right** properties to move the element.
* ⚠️ Important: These offset properties **do not work unless position is defined**.

There are **6 main types of positioning** in CSS:

**📍 static**

* **Default value**.
* The element is positioned in the **normal flow of the page**.
* No special positioning is applied.

**📍 sticky**

* The element is positioned based on the **user’s scroll position**.
* It behaves like **relative** until a certain point, then **“sticks”** like **fixed**.
* Commonly used for **sticky headers or menus**.

**📍 fixed**

* The element is positioned **relative to the viewport (browser window)**.
* It **stays in the same place** even when the page is scrolled.
* Uses **top / right / bottom / left** to fix its location.
* Great for **navbars or floating buttons**.

**📍 relative**

* The element is positioned **relative to its normal position**.
* Offsets (top, left, etc.) move it **without affecting other elements’ flow**.

**📍 absolute**

* The element is positioned **relative to its nearest positioned ancestor**.
* If no ancestor has a position defined, it defaults to the **viewport**.

📚 Learn more & visualize: [W3Schools – CSS Positioning](https://www.w3schools.com/css/css_positioning.asp)

**📏 2.5 Width and Height**

The **width** and **height** properties are used to set the **size of an element**.

* They **do not include padding, borders, or margins**.
* They only define the **content area** inside the box model.

✅ Example:

div {

height: 200px;

width: 50%;

}

**🔑 Accepted Values:**

* **auto** → (default) browser calculates size automatically.
* **length** → specific size (px, cm, em, rem, etc.).
* **% (percentage)** → relative to the **containing block**.
* **initial** → resets to the default value.
* **inherit** → takes the value from its parent.

**⚖️ max-width vs width**

* **width**
  + Sets a **fixed width** for an element.
  + Can cause **horizontal scrolling** if the element is wider than the viewport.
* **max-width**
  + Sets the **maximum possible width** an element can have.
  + More flexible than width.
  + Prevents content from overflowing the viewport.
  + Recommended for **responsive design**.

📚 Learn more & see examples: [W3Schools – CSS Dimensions](https://www.w3schools.com/css/css_dimension.asp)

**📦 2.6 Margin and Padding**

When working with CSS, you’ll often hear the term **Box Model**.  
👉 The box model is how CSS thinks about **layout and spacing**.

Every HTML element is basically a **box** made up of:

1. **Content** → the actual text or image
2. **Padding** → the space *inside* the element, around the content
3. **Border** → wraps around the padding
4. **Margin** → the space *outside* the border

**🔲 Margin**

* **Margin = the space outside an element**.
* Controlled by the margin property.
* Always sits **outside the border**.

**Margin values:**

* margin-top
* margin-right
* margin-bottom
* margin-left

✅ Example:

div {

margin-top: 20px;

margin-bottom: 30px;

}

💡 **Pro tip:**  
You can **center an element horizontally** with:

margin: 0 auto;

This means → 0 margin top & bottom, auto left & right.

**🟦 Padding**

* **Padding = the space inside an element**, around the content.
* Controlled by the padding property.
* Always sits **inside the border**.

**Padding values:**

* padding-top
* padding-right
* padding-bottom
* padding-left

✅ Example:

p {

padding-left: 15px;

padding-right: 15px;

}

**✨ Shorthand for Margin & Padding**

Instead of writing each side separately, you can use **shorthand**:

1. **Four values** → margin: top right bottom left;
2. margin: 25px 50px 75px 100px;
   * Top = 25px
   * Right = 50px
   * Bottom = 75px
   * Left = 100px
3. **Three values** → margin: top right&left bottom;
4. margin: 25px 50px 75px;
   * Top = 25px
   * Right & Left = 50px
   * Bottom = 75px
5. **Two values** → margin: top&bottom right&left;
6. margin: 25px 50px;
   * Top & Bottom = 25px
   * Right & Left = 50px
7. **One value** → margin: all;
8. margin: 25px;
   * All sides = 25px

✨ Remember:

* **Margin** = pushes *other elements away* (outside spacing).
* **Padding** = creates *space inside the element* (breathing room for content).

**🐶 2.7 Applying Margin and Padding to car Lover’s Page**

To better understand how **margin** and **padding** work in a real project, check out this demo:  
▶️ [Watch Class Demo Video](https://youtu.be/o63_SjND94Q)

You’ll see how margins and padding are applied step by step in the **car Lover’s page** project.

**🎨 2.8 Borders**

The **CSS border properties** let you control the **style, width, and color** of an element’s border.

👉 You can use the **border shorthand property** to define all three at once.

**🔹 Border Sub-properties:**

* **border-width**
  + Sets the **thickness** of the border.
  + Values: thin, medium, thick, or exact values like 5px.
* **border-style**
  + Defines the **look of the border**.
  + Common values: solid, dotted, dashed, etc.
  + ⚠️ You **must always specify** a border style, or the border won’t show!
* **border-color**
  + Sets the **color** of the border.
  + If omitted, the border will automatically take the **text color** of the element.

✅ Example:

div {

border: 1px solid black;

}

👉 This creates a **1px solid black border**.

**🌈 2.9 Background**

The **background property** allows you to style the **background of an element**.  
👉 A background can be either a **color** or an **image**.

It’s a **shorthand property**, meaning you can combine multiple background properties into one line.

**🔹 Background Sub-properties:**

* **background-color** → Sets the background **color**
* **background-image** → Sets one or more background **images**
* **background-position** → Sets the **position** of the background image
* **background-size** → Controls the **size** of the background image

✅ Example (shorthand):

body {

background: green url("img\_tree.gif") center cover;

}

This means:

* Background color → green
* Background image → "img\_tree.gif"
* Image position → center
* Image size → cover

✨ Quick Recap:

* **Margin & Padding** → Space control (outside vs inside)
* **Borders** → Add style, width, and color around elements
* **Backgrounds** → Add color or images behind elements

**✍️ 2.10 Fonts**

The **CSS font property** is used to change the **look and feel of text** on a webpage.  
👉 The font property can be written as a **shorthand** for multiple font-related sub-properties.

**🔹 Font Sub-properties**

* **font-style** → Controls the *style* of the font:
  + Values: normal, italic, oblique
* **font-weight** → Controls the *thickness* or **boldness** of the font.
  + Example values: normal, bold, 100–900 (depending on font family)
* **font-size** → Sets the *size* of the font.
  + Example: 16px, 1.2em, 120%
* **font-family** → Specifies the *font family* to be used.
  + Example: "Arial", sans-serif

**🔹 Using Custom Fonts with @font-face**

You can **import custom fonts** into your project with the @font-face rule.  
✅ Example:

@font-face {

font-family: "Open Sans";

src: url("/fonts/OpenSans-Regular-webfont.woff2") format("woff2"),

url("/fonts/OpenSans-Regular-webfont.woff") format("woff");

}

👉 This allows you to use "Open Sans" in your CSS like this:

body {

font-family: "Open Sans", sans-serif;

}

📌 Helpful resources:

* [How to add custom fonts locally](https://www.pagecloud.com/blog/how-to-add-custom-fonts-to-any-website)
* [Using Google Fonts in your website](https://www.freecodecamp.org/news/how-to-use-google-fonts-in-your-next-web-design-project-e1ad48f1adfa/)

**🎨 2.11 Color**

The **color property** in CSS sets the **text color** of an element.  
👉 This value may also be used **indirectly** in other properties (like border-color) if not explicitly defined.

**🔹 Ways to Define Colors**

* **Hexadecimal (#RRGGBB)**
* body { color: #92a8d1; }

Example: #92a8d1

* **RGB values (rgb(red, green, blue))**
* body { color: rgb(201, 76, 76); }
* **Color names**
* body { color: red; }

👉 Browsers understand all these formats, so you can use whichever feels easiest!

**⚖️ 2.12 Priority Order in CSS**

When writing CSS, sometimes **rules conflict** with each other.  
👉 CSS follows a **priority order** (or “specificity”) to decide which style to apply.

**🔹 How Priority Works**

* **More specific rules** override more general ones.
* The **location of the rule** also matters → if two rules have the same specificity, the one that comes **last** wins.
* Using !important will **force** that rule to override everything else.

**🔹 Priority Levels (highest → lowest)**

1. **!important**
2. **Inline CSS** (styles written directly in the HTML element)
3. **Embedded styles** (internal <style> block in HTML)
4. **ID selectors** (#id)
5. **Class selectors** (.class)
6. **Tag selectors** (p, h1, div)

✅ Quick Expression:

!important > inline CSS > embedded styles > ID nesting > ID > class nesting > class > tag nesting > tag

✨ **Quick Recap:**

* **Fonts** 🎶 → Use font-family, font-size, font-style, font-weight + custom fonts with @font-face
* **Color** 🌈 → Define text colors with hex, RGB, or names
* **Priority** ⚖️ → More specific rules override general ones; !important is the boss!