Module 1

Module 1: CSS Fundamentals

Welcome to the world of CSS! This module will introduce you to the core concepts of styling web pages. Think of CSS as the artist's brush that gives your website its unique look and feel.

1.1 Introduction to CSS

What is CSS and its role in web development?

CSS stands for **Cascading Style Sheets**. In simple terms, it's a language that tells web browsers how to display HTML elements. While HTML provides the structure (like the bones of a house), CSS provides the styling (like the paint, furniture, and decorations).

Its role is to make your web pages look good! This includes things like:

- Setting colors for text and backgrounds
- Controlling fonts and their sizes
- Arranging elements on the page
- Adding animations and transitions

Separation of Concerns (HTML for Structure, CSS for Presentation)

This is a really important concept! Imagine building a house:

- **HTML** is like the blueprint and the raw building materials (walls, doors, windows). It defines *what* content is on the page.
- **CSS** is like the interior designer and painter. It defines *how* that content looks.

Keeping HTML and CSS separate makes your code:

- **Easier to read and understand:** You know where to look for structure and where to look for styling.
- **Easier to maintain:** If you want to change the look of your entire website, you often only need to modify your CSS, not every single HTML file.
- **More efficient:** Your browser can load HTML quickly, and then apply styles.

Different ways to include CSS in an HTML document

There are three main ways to link your CSS to your HTML:

1. **Inline Styles:**
   - **How it works:** You write CSS directly inside an HTML tag using the `style` attribute.
   - **When to use:** Rarely! It mixes content and presentation, making your HTML messy and hard to manage. Best for very specific, one-off styles that won't be reused.
   - **Example:**

   HTML

   ```
   <p style="color: blue; font-size: 18px;">This text is blue and larger.</p>
   ```

2. **Internal (or Embedded) Styles:**

- **How it works:** You place CSS rules within a `<style>` tag in the `<head>` section of your HTML document.
- **When to use:** Useful for small projects or when you have styles specific to a single HTML page that won't be used anywhere else.
- **Example:**

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Internal CSS Example</title>
    <style>
        body {
            font-family: sans-serif;
            background-color: #f0f8ff; /* Alice Blue */
        }
        h1 {
            color: #8a2be2; /* Blue Violet */
        }
    </style>
</head>
<body>
    <h1>My Awesome Page</h1>
    <p>This page uses internal CSS.</p>
</body>
</html>
```

3. **External Styles (The Best Way!):**

- **How it works:** You write all your CSS in a separate file (e.g., `style.css`) and then link it to your HTML document using a `<link>` tag in the `<head>` section.
- **When to use:** Almost always! This is the professional and most maintainable way. It keeps your HTML clean and allows you to reuse the same CSS file across many different HTML pages.
- **Example (HTML - `index.html`):**

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>External CSS Example</title>
    <link rel="stylesheet" href="style.css"> </head>
<body>
    <h1>My External CSS Page</h1>
    <p>This page uses an external CSS file.</p>
    <button class="primary-button">Click Me</button>
</body>
</html>
```

Example (CSS - `style.css`):

CSS

```
/* style.css */
body {
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    background-color: #e0f2f7; /* Light Blue */
    margin: 20px;
}

h1 {
    color: #2c3e50; /* Dark Blue-Gray */
    text-align: center;
}

p {
    color: #34495e; /* Another Dark Blue-Gray */
    line-height: 1.6;
}

.primary-button {
    background-color: #007bff; /* Blue */
    color: white;
    padding: 10px 15px;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    transition: background-color 0.3s ease; /* Smooth hover effect */
}

.primary-button:hover {
    background-color: #0056b3; /* Darker Blue on hover */
}
```

Basic CSS syntax (rulesets, selectors, declarations, properties, values)

CSS is built on simple rules. Here's a breakdown of the parts that make up a CSS rule:

CSS

```
selector {
    property: value; /* This is a declaration */
    another-property: another-value;
}
```

- **Selector:** This tells the browser *which* HTML element(s) you want to style. In the example above, `selector` is the target.
- **Ruleset:** The entire block, including the selector and the curly braces `{}` containing the declarations.
- **Declarations:** Each line inside the curly braces. A declaration is a pair of a property and its value, separated by a colon (`:`), and ending with a semicolon (`;`).

- **Property:** The specific aspect of the element you want to change (e.g., `color`, `font-size`, `background-color`).
- **Value:** The setting you want to apply to that property (e.g., `blue`, `16px`, `#f0f0f0`).

Example:

CSS

```
p { /* 'p' is the selector */
    color: red; /* 'color' is the property, 'red' is the value */
    font-size: 16px; /* 'font-size' is the property, '16px' is the value */
}
```

This rule says: "Find all `<p>` (paragraph) elements, make their text color red, and set their font size to 16 pixels."

Mini-Project: "My First Styled Page"

1. Create a new folder called `my-styled-page`.
2. Inside, create an `index.html` file and a `style.css` file.
3. In `index.html`, add a basic HTML structure with:
   - A main heading (`<h1>`)
   - A paragraph (`<p>`)
   - An unordered list (`<ul>`) with a few list items (`<li>`)
   - Make sure to link your `style.css` file in the `<head>`!
4. In `style.css`, write some basic rules to:
   - Change the `body` background color.
   - Set the `<h1>` text color and center it.
   - Change the `p` (paragraph) font size and color.
   - Give the `ul` (unordered list) a different background color and remove its default bullet points (you'll need to search for `list-style-type`).
5. Open `index.html` in your browser to see your styled page!

1.2 CSS Selectors

Selectors are incredibly important in CSS because they determine *which* HTML elements your styles will apply to. Mastering selectors gives you precise control over your web page's appearance.

Type selectors, class selectors, ID selectors

These are the most fundamental selectors you'll use daily.

1. **Type (or Element) Selectors:**

   - **What they do:** Select all instances of a specific HTML element type.
   - **Syntax:** Just the element's tag name (e.g., `p`, `h1`, `div`).
   - **Example:**

   CSS

   ```
   /* Selects all paragraph elements */
   p {
   ```

```
    font-family: Georgia, serif;
    color: #333;
}

/* Selects all heading level 2 elements */
h2 {
    border-bottom: 2px solid #ccc;
    padding-bottom: 5px;
}
```

2. **Class Selectors:**

- **What they do:** Select all elements that have a specific `class` attribute. Classes are super flexible because you can apply the same class to multiple different HTML elements, and an element can have multiple classes.
- **Syntax:** A dot (`.`) followed by the class name (e.g., `.button`, `.card`).
- **HTML Example:** HTML

```
<p class="highlight">This text is highlighted.</p>
<div class="card highlight">This card is also highlighted.</div>
<button class="button primary">Submit</button>
```

- **CSS Example:**

CSS

```
/* Styles elements with the class "highlight" */
.highlight {
    background-color: yellow;
    font-weight: bold;
}

/* Styles elements with the class "button" */
.button {
    padding: 10px 20px;
    border-radius: 5px;
}

/* Styles elements with the class "primary" (often combined with other classes) */
.primary {
    background-color: dodgerblue;
    color: white;
}
```

3. **ID Selectors:**

- **What they do:** Select a *single, unique* element that has a specific `id` attribute. **An ID should only be used once per page!**
- **Syntax:** A hash (`#`) followed by the ID name (e.g., `#main-header`, `#footer`).
- **HTML Example:** HTML

```
<header id="main-header">
    <h1>Welcome</h1>
</header>
<footer id="page-footer">
    <p>&copy; 2025 My Website</p>
</footer>
```

- **CSS Example:**

CSS

```
/* Styles the element with the ID "main-header" */
#main-header {
    background-color: #28a745; /* Green */
    color: white;
    padding: 20px;
    text-align: center;
}

/* Styles the element with the ID "page-footer" */
#page-footer {
    background-color: #6c757d; /* Gray */
    color: white;
    padding: 15px;
    text-align: center;
}
```

Combinators

Combinators allow you to select elements based on their relationship to other elements.

1. **Descendant Combinator (Space):**

- **What it does:** Selects an element that is a descendant (child, grandchild, etc.) of another element.
- **Syntax:** `ancestor descendant` (space between selectors).
- **Example:**

CSS

```
/* Selects all 'a' (link) elements that are inside a 'nav' element */
nav a {
    color: #007bff;
    text-decoration: none;
}

/* Selects all 'li' (list item) elements that are inside a 'footer' element */
footer li {
    list-style: none; /* Removes bullet points */
    margin-bottom: 5px;
}
```

2. **Child Combinator (`>`):**

- **What it does:** Selects an element that is a *direct child* of another element.
- **Syntax:** `parent > child` (greater than sign).
- **Example:**

HTML

```
<div class="menu">
    <p>Menu Title</p>
    <ul>
        <li>Item 1</li>
        <li>Item 2</li>
    </ul>
</div>
```

CSS

```
/* Selects only the UL that is a direct child of an element with class "menu" */
.menu > ul {
    border: 1px solid red;
}

/* This would NOT select the <li> because it's a grandchild, not a direct child, of .menu
*/
/* .menu > li { ... } */
```

3. **Adjacent Sibling Combinator (`+`):**

- **What it does:** Selects an element that is the *immediately next sibling* of another element. They must share the same parent and the first element must come directly before the second in the HTML.
- **Syntax:** `element1 + element2` (plus sign).
- **Example:**

HTML

```
<h1>My Title</h1>
<p>This is the first paragraph.</p>
<p>This is the second paragraph.</p>
<div>Another element</div>
<p>This is the third paragraph.</p>
```

CSS

```
/* Selects the <p> element that immediately follows an <h1> element */
h1 + p {
    font-style: italic;
    margin-top: 30px; /* Add more space below the heading */
}
```

4. **General Sibling Combinator (`~`):**

- **What it does:** Selects all siblings of an element that appear *after* it in the HTML, regardless of whether they are immediately adjacent.
- **Syntax:** `element1 ~ element2` (tilde sign).
- **Example:**

HTML

```
<h2>Section Title</h2>
<p>Intro paragraph.</p>
<div>A div in between.</div>
<p>Another paragraph.</p>
<p>Last paragraph.</p>
```

CSS

```
/* Selects all <p> elements that come after an <h2> element */
h2 ~ p {
    color: green;
    border-left: 3px solid green;
    padding-left: 10px;
}
```

Attribute Selectors

Attribute selectors allow you to select elements based on the presence or value of their HTML attributes.

- `[attribute]` : Selects elements with the specified attribute.
- `[attribute="value"]` : Selects elements where the attribute has an exact value.
- `[attribute~="value"]` : Selects elements where the attribute's value is a space-separated list, and one of the words is "value".
- `[attribute|="value"]` : Selects elements where the attribute's value starts with "value" immediately followed by a hyphen or is exactly "value".
- `[attribute^="value"]` : Selects elements where the attribute's value starts with "value".
- `[attribute$="value"]` : Selects elements where the attribute's value ends with "value".
- `[attribute*="value"]` : Selects elements where the attribute's value contains "value" anywhere.

Example:

HTML

```
<a href="https://example.com" target="_blank">External Link</a>
<a href="/internal-page.html">Internal Link</a>
<input type="text" placeholder="Your name">
<input type="submit" value="Send">
<img src="image.jpg" alt="A beautiful landscape">
<div data-item-id="123">Item Info</div>
```

CSS

```
/* Selects all elements with a 'target' attribute */
[target] {
```

```
    border: 1px solid blue;
}

/* Selects all links that go to an external site */
a[href^="https://"] {
    color: purple;
    font-weight: bold;
}

/* Selects all images with an 'alt' attribute containing "beautiful" */
img[alt*="beautiful"] {
    box-shadow: 0 0 5px rgba(0,0,0,0.5);
}

/* Selects an input field specifically of type "text" */
input[type="text"] {
    border: 2px solid orange;
    padding: 5px;
}

/* Selects elements with a custom data attribute 'data-item-id' */
[data-item-id] {
    background-color: lightgray;
    padding: 10px;
}
```

Pseudo-classes

Pseudo-classes select elements based on a special state they are in, not based on their regular HTML attributes. They start with a single colon (`:`).

- `:hover`: When the mouse pointer is over an element.
- `:focus`: When an input field or interactive element is selected (e.g., clicked or tabbed to).
- `:active`: When an element is being activated (e.g., a button being clicked down).
- `:link`: An unvisited link.
- `:visited`: A visited link.
- `:first-child`: The first child element of its parent.
- `:last-child`: The last child element of its parent.
- `:nth-child(n)`: The nth child element of its parent (you can use numbers, `odd`, `even`, or formulas like `2n` for every second element).
- `:nth-of-type(n)`: Similar to `nth-child`, but selects the nth element of a *specific type* among its siblings.
- `:not(selector)`: Selects elements that do *not* match the given selector.
- `:empty`: Selects elements that have no children (including text nodes).
- `:has(selector)` (New in 2025, great for parent selection!): Selects an element if it contains at least one element that matches the given selector. This is super powerful for selecting a parent based on its child!

Example:

HTML

```
<nav>
    <a href="#home">Home</a>
    <a href="#about">About</a>
```

```
    <a href="#contact">Contact</a>
</nav>

<form>
    <input type="text" placeholder="Enter name">
    <button>Submit</button>
</form>

<ul>
    <li>First Item</li>
    <li>Second Item</li>
    <li>Third Item</li>
    <li>Fourth Item</li>
</ul>

<div class="card">
    <img src="thumbnail.jpg" alt="thumbnail">
    <p>Card content here</p>
</div>
<div class="card">
    <p>Card without image</p>
</div>
```

CSS

```
/* Change link color on hover */
a:hover {
    color: #ff5733; /* Orange-Red */
    text-decoration: underline;
}

/* Style input when it's focused */
input:focus {
    border-color: dodgerblue;
    box-shadow: 0 0 5px rgba(0, 123, 255, 0.5);
    outline: none; /* Remove default browser outline */
}

/* Make the first list item bold */
li:first-child {
    font-weight: bold;
}

/* Make every odd list item have a light background */
li:nth-child(odd) {
    background-color: #f9f9f9;
}

/* Style paragraphs that are NOT the first child */
p:not(:first-child) {
    margin-top: 15px;
}
```

```
/* Style a card if it contains an image (modern CSS :has()) */
.card:has(img) {
    border: 2px solid green;
    background-color: #e6ffe6; /* Light green background */
}
```

Pseudo-elements

Pseudo-elements style *parts* of an element, rather than the element itself. They start with a double colon (`::`).

- `::before`: Inserts content *before* the actual content of an element.
- `::after`: Inserts content *after* the actual content of an element.
- `::first-line`: Styles the first line of a block-level element.
- `::first-letter`: Styles the first letter of a block-level element.
- `::selection`: Styles the portion of an element that is highlighted by the user (e.g., when selecting text with the mouse).

Important: For `::before` and `::after`, you **must** include the `content` property, even if it's empty (`content: '';`).

Example:

HTML

```
<p class="quote">This is an important quote.</p>
<h3 class="decorated-heading">About Us</h3>
```

CSS

```
/* Add quotation marks before and after the quote */
.quote::before {
    content: "“";
    font-size: 2em;
    color: #555;
    vertical-align: middle;
    margin-right: 5px;
}

.quote::after {
    content: "”";
    font-size: 2em;
    color: #555;
    vertical-align: middle;
    margin-left: 5px;
}

/* Style the first line of a paragraph */
p::first-line {
    font-weight: bold;
    color: darkblue;
}

/* Style the first letter of an H3 with a custom icon */
```

```
.decorated-heading::before {
    content: "✨ "; /* An emoji or unicode character */
    color: gold;
    font-size: 1.2em;
}

/* Change the background color of selected text */
::selection {
    background-color: dodgerblue;
    color: white;
}
```

Specificity and the Cascade

These are two crucial concepts that determine which CSS rule wins when multiple rules try to style the same element.

1. **The Cascade:**

    - CSS stands for **Cascading** Style Sheets. This means styles flow down and can be overridden.
    - When multiple rules apply to an element, the browser follows a set of rules to decide which style to apply.
    - The general order of precedence (where later rules can override earlier ones):
        - Browser default styles
        - User custom styles (if any)
        - Author (your) styles:
            - External stylesheets
            - Internal stylesheets
            - Inline styles
        - `!important` (Use with caution! Avoid if possible).

    Example:

    CSS

    ```
    /* In style.css */
    p {
        color: blue;
    }

    /* Later in style.css */
    p {
        color: green; /* This will override 'blue' */
    }
    ```

    HTML

    ```
    <p style="color: purple;">This text will be purple.</p>
    ```

    In the example above, `purple` wins because inline styles have higher precedence than external/internal styles.
2. **Specificity:**

- Specificity is a score or weight given to a CSS selector based on how specific it is. A more specific selector wins over a less specific one, even if the less specific one comes later in the CSS file.
- Think of it like a scoring system:
  - **Inline styles:** 1000 points
  - **ID selectors:** 100 points
  - **Class selectors, attribute selectors, pseudo-classes:** 10 points
  - **Type selectors, pseudo-elements:** 1 point
  - **Universal selector (`*`):** 0 points

How to calculate: Sum up the points for each part of your selector.

- `p` (1 point)
- `.my-class` (10 points)
- `#my-id` (100 points)
- `div p` (1 + 1 = 2 points)
- `div.my-class` (1 + 10 = 11 points)
- `#main-nav li a:hover` (100 + 1 + 1 + 10 = 112 points)

Example:

CSS

```
/* Rule A */
p {
    color: red; /* Specificity: 1 */
}

/* Rule B */
.my-text {
    color: blue; /* Specificity: 10 */
}

/* Rule C */
#special-paragraph {
    color: green; /* Specificity: 100 */
}
```

HTML

```
<p class="my-text" id="special-paragraph">This text will be green.</p>
```

Even though `.my-text` comes later than `p` in the CSS, `#special-paragraph` wins because it has the highest specificity (100 points).

When specificity is equal, the rule declared later in the stylesheet wins.

`!important`:

- This is a special keyword that makes a declaration override all other declarations, regardless of specificity or order.
- **Use it as a last resort!** It breaks the cascade and makes CSS very hard to debug and maintain. Good practice is to avoid it if possible.

CSS

```
.my-button {
    background-color: blue !important; /* This will always win for background-color */
}
.my-button {
    background-color: red; /* This will be ignored due to !important */
}
```

Mini-Project: "Product Card Selectors"

Create an `index.html` and `style.css` file.

In `index.html`, create several "product cards" using `div` elements. Each card should contain:

- An `<h2>` for the product name.
- A `<p>` for a short description.
- A `<span>` for the price.
- A `<button>` for "Add to Cart".

Give some cards different classes (e.g., `featured-product`, `on-sale`). Give one card a unique ID (e.g., `best-seller`).

In `style.css`, use different selectors to style these cards:

- Style all `div` elements that are product cards.
- Style the `<h2>` inside `div` elements.
- Use `::before` or `::after` to add a small "⭐" emoji before the product name of `featured-product` cards.
- Change the background color of the `on-sale` cards.
- Make the `best-seller` card have a unique border.
- Use `:hover` to add a subtle shadow effect when hovering over any product card.
- Challenge: Use `:nth-child` to give alternating product cards a slightly different background color.
- Challenge: Use `:has()` to give a red border to any product card that contains a `span` with a specific class for "out-of-stock" (you'll need to add that class to some spans in your HTML).

1.3 Working with Color, Text, and Backgrounds

These are some of the most visible ways CSS allows you to customize your web page.

Understanding color values

Colors are fundamental to design. CSS offers several ways to define them:

1. **Keywords:** Simple, readable names for common colors. Limited but easy to remember.
   - `red`, `blue`, `green`, `black`, `white`, `orange`, `dodgerblue`, `lightgray`, `rebeccapurple`.
   - **Example:** `color: red;`
2. **Hexadecimal (Hex) Values:** A very common way to define colors using a six-digit code (or three-digit shorthand) preceded by a `#`. Each pair of digits (or single digit in shorthand) represents the amount of Red, Green, and Blue (RGB) from 00 (0) to FF (255).
   - `#RRGGBB`
   - `#FF0000` is red, `#00FF00` is green, `#0000FF` is blue.

- `#FFFFFF` is white, `#000000` is black.
- Shorthand: If each pair of digits is the same (e.g., `#FFCC00`), you can write `#FC0`.
- **Example:** `background-color: #3498db;` (a shade of blue)

3. **RGB Values:** `rgb(red, green, blue)` - Defines colors using specific values for Red, Green, and Blue, from 0 to 255.
   - `rgb(255, 0, 0)` is red.
   - `rgb(0, 128, 0)` is a shade of green.
   - **Example:** `color: rgb(75, 192, 192);`
4. **RGBA Values:** `rgba(red, green, blue, alpha)` - Same as RGB, but with an added "alpha" channel for **opacity**. The alpha value ranges from 0 (fully transparent) to 1 (fully opaque).
   - `rgba(255, 0, 0, 0.5)` is 50% opaque red.
   - **Example:** `background-color: rgba(0, 0, 0, 0.7);` (a semi-transparent black overlay)
5. **HSL Values:** `hsl(hue, saturation, lightness)` - Defines colors using Hue, Saturation, and Lightness. This model is often more intuitive for designers.
   - **Hue:** The color itself, a degree on a color wheel from 0 to 360 (e.g., 0/360 is red, 120 is green, 240 is blue).
   - **Saturation:** The intensity of the color, from 0% (grayscale) to 100% (full color).
   - **Lightness:** How light or dark the color is, from 0% (black) to 100% (white).
   - **Example:** `color: hsl(200, 80%, 50%);` (a vibrant blue)
6. **HSLA Values:** `hsla(hue, saturation, lightness, alpha)` - Same as HSL, but with an added "alpha" channel for opacity.
   - **Example:** `border-color: hsla(100, 70%, 40%, 0.6);` (a semi-transparent green border)

Best Practice for 2025: While all are valid, Hex and RGB/RGBA are widely used. HSL/HSLA is gaining popularity for its readability and ease of adjustment. CSS also supports `lab()` and `lch()` for more advanced color spaces, offering even greater precision for professional design, but for beginners, stick to the main ones.

Styling text

Text is a huge part of web content. CSS gives you immense control over its appearance.

- `font-family`: Sets the typeface. Always provide a "fallback" generic family (like `sans-serif`, `serif`, `monospace`) in case the user's system doesn't have your preferred font.
  - **Example:** `font-family: 'Arial', sans-serif;`
  - **Modern Note:** Use `@font-face` to embed custom fonts from services like Google Fonts.

  CSS

  ```
  /* Example with a Google Font imported (usually done at the top of your CSS file) */
  @import url('https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap');

  body {
      font-family: 'Roboto', sans-serif;
  }
  ```

- `font-size`: Sets the size of the text. Common units include `px` (pixels), `em` (relative to parent's font size), `rem` (relative to root `html` font size), `vw` (viewport width), `vh` (viewport height). `rem` is generally preferred for responsiveness.
  - **Example:** `font-size: 16px;` or `font-size: 1rem;` or `font-size: clamp(1rem, 2vw + 1rem, 2.5rem);` (modern, for responsive scaling)
- `font-weight`: Sets the thickness of the characters. Values often include `normal` (400), `bold` (700), `lighter`, `bolder`, or numeric values from 100-900 (if supported by the font).

    - **Example:** `font-weight: bold;` or `font-weight: 700;`
- `text-align`: Aligns text within its container. Values: `left`, `right`, `center`, `justify`.
    - **Example:** `text-align: center;`
- `line-height`: Sets the spacing between lines of text. Can be a number (multiplier of font-size, recommended for responsiveness), a length (e.g., `px`, `em`), or a percentage.
    - **Example:** `line-height: 1.6;` (1.6 times the font size)
- `color`: Sets the text color (as discussed above).
    - **Example:** `color: #333;`
- Other useful text properties:
    - `text-decoration`: `none` (to remove underlines from links), `underline`, `overline`, `line-through`.
    - `text-transform`: `uppercase`, `lowercase`, `capitalize`.
    - `letter-spacing`: Adjusts space between letters.
    - `word-spacing`: Adjusts space between words.

Applying background colors and images

Backgrounds are crucial for visual appeal.

1. **Background Color:**
    - Use the `background-color` property with any color value.
    - **Example:** `background-color: #f8f8f8;`
2. **Background Images:**
    - Use the `background-image` property with `url('path/to/image.jpg')`.
    - **Example:** `background-image: url('images/hero-bg.jpg');`
    - You can also use gradients:
        - `linear-gradient(direction, color1, color2, ...)`
        - `radial-gradient(shape, color1, color2, ...)`
        - **Example:** `background-image: linear-gradient(to right, #007bff, #33ccff);`

Background properties (repeat, position, size)

When using background images, these properties give you fine control:

- `background-repeat`: How the background image repeats.
    - `no-repeat`: Displays the image once.
    - `repeat`: Repeats both horizontally and vertically (default).
    - `repeat-x`: Repeats horizontally only.
    - `repeat-y`: Repeats vertically only.
    - **Example:** `background-repeat: no-repeat;`
- `background-position`: Where the background image is placed.
    - Keywords: `top`, `bottom`, `left`, `right`, `center`.
    - Percentages: `50% 50%` (center).
    - Lengths: `20px 30px` (x-offset y-offset).
    - **Example:** `background-position: center center;` or `background-position: 50% 50%;`
- `background-size`: How the background image is scaled.
    - `auto`: Default size.
    - `cover`: Scales the image to cover the entire background area, even if it has to crop parts of the image. **Very common for hero sections.**
    - `contain`: Scales the image to be as large as possible without cropping or stretching, fitting entirely within the background area (might leave empty space).
    - Percentages: `100% 100%` (stretches to fill).

    - Lengths: `500px auto`.
    - **Example:** `background-size: cover;`
- `background-attachment`: How the background image scrolls.
    - `scroll`: The background image scrolls with the content (default).
    - `fixed`: The background image stays in a fixed position relative to the viewport, creating a parallax effect.
    - `local`: The background scrolls with the content of the element itself.
    - **Example:** `background-attachment: fixed;`

Shorthand Property: `background` You can combine many background properties into a single `background` shorthand property for conciseness:

CSS

```
selector {
    background: [color] [image] [repeat] [attachment] [position] / [size];
}
/* Note: Size comes after position, separated by a slash (/) */
```

Example:

CSS

```
body {
    background: #f0f8ff url('images/pattern.png') repeat-x top center / 100% auto fixed;
}
```

This sets:

- `background-color`: `#f0f8ff`
- `background-image`: `url('images/pattern.png')`
- `background-repeat`: `repeat-x`
- `background-position`: `top center`
- `background-size`: `100% auto`
- `background-attachment`: `fixed`

Mini-Project: "Styling a Blog Post"

Create an `index.html` and `style.css` file.

In `index.html`, create a simple blog post structure:

- A `<div>` with the class `blog-post`.
- Inside the div:
    - An `<h1>` for the post title.
    - A `<p>` for a short intro.
    - Another `<p>` for the main content.
    - A `<footer>` with a `<p>` for the author and date.

In `style.css`:

- Set a pleasant `font-family` and a comfortable `font-size` for the `body`.
- Apply a `background-color` to the `body` and possibly a subtle `background-image` (you can find free patterns online or use a simple linear gradient). Ensure the image repeats nicely if it's a pattern.

- Style the `.blog-post` div:
  - Give it a `background-color` (e.g., white or light gray).
  - Add `padding` to give space around the content.
  - Give it a subtle `box-shadow` to make it stand out.
- Style the `<h1>`:
  - Set its `color`.
  - `text-align: center;`
  - Increase `font-size` and `font-weight`.
- Style the main content `<p>`:
  - Adjust `line-height` for readability.
  - Set its `color`.
  - Use `text-align: justify;`
- Style the footer text:
  - Make it smaller `font-size`.
  - Make it `italic`.
  - Use `text-align: right;`
- Challenge: Add a small, decorative `::before` pseudo-element to the `<h1>` with a content character or emoji.
- Challenge: Use `background-image` with `linear-gradient` on the `<footer>` of the blog post.