AIMERZ.ai
Aim.Act.Achieve

NOTES

Deep dive into CSS

What is CSS selector

CSS (Cascading Style Sheets) selectors are used to target specific HTML elements or groups of elements in order to apply styles to them. CSS selectors allow us to specify which HTML elements we want to style, based on their attributes, classes, ids, and other characteristics.

Tag name Selector

A tag name selector in css targets HTML elements based on their tag name.It allows you to apply styles to all occurrences of a particular HTML element throughout your web page.

Tag Name Syntax:

```
JavaScript
tagname{
    /* styles properties */

}
```

Let us take an example of tagName to understand better:

index.html

```
JavaScript
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tag Name Selector Example</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <h1>This is a heading</h1>
    <p>This is a paragraph. The style of this text won't change.</p>
</body>
</html>
```

Style.css

```
JavaScript
h1 {
    color: blue;
    font-size: 36px;
    text-align: center;
    border-bottom: 2px solid black;

}
```

Output:

ID Selector

An ID selector in CSS targets a specific HTML element based on its unique ID attribute value. It allows you to apply styles to a single element with a unique ID. Hash(#) followed by the id name selects the particular HTML element.

To use an ID selector, one should know the ID of the element to be styled . The id attribute is added to an element in the HTML code.

ID selector Syntax

```
JavaScript
#idName {
 /* styles properties */

}
```

Example:

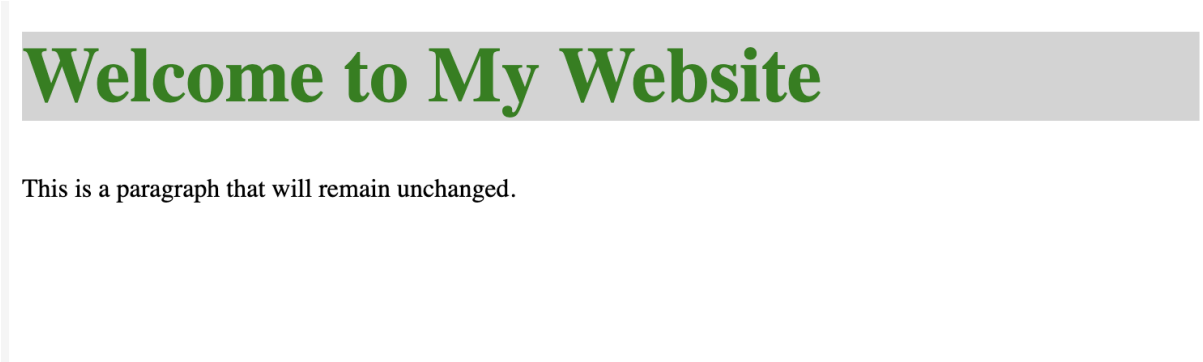
index.html

```
JavaScript
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>ID Selector Example</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <h1 id="main-heading">Welcome to My Website</h1>
    <p>This is a paragraph that will remain unchanged.</p>
</body>
</html>
```

Styles.css

```
JavaScript
/* ID Selector for #main-heading */
#main-heading {
    color: green;
    font-size: 48px;
    text-align: left;
    background-color: lightgray;

}
```

Output:



Class selector

A class selector in CSS targets HTML elements based on their class attribute value or class name. It allows you to apply styles to one or more elements that share the same class name.
To use a class selector, ones should know the class name of the element to be styled. The class name attribute is added to an element in the HTML code.

Dot (.) followed by the class name is used to select the particular HTML element.

Class selector syntax:

```
JavaScript
.className {
  /* styles properties */
}
```

Example:

index.html

```
JavaScript
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Class Selector Example</title>
    <link rel="stylesheet" href="styles.css">
</head>
```

```
<body>
    <h1 class="highlight-text">Welcome to My Website</h1>
    <p class="highlight-text">This is a paragraph with highlighted text.</p>
    <p>This is a normal paragraph without any special style.</p>
</body>
</html>
```

style.css

```
JavaScript
/* Class Selector for .highlight-text */

.highlight-text {
    color: purple;
    font-weight: bold;
    background-color: yellow;
    padding: 10px;
    border-radius: 5px;
}
```

Universal Selector

The universal selector in CSS is represented by an asterisk (*). It can be used to select all elements in a document.
To use a universal selection, simply place an asterisk before the element name.

Syntax:

```
JavaScript
* { /* styles properties */}
```

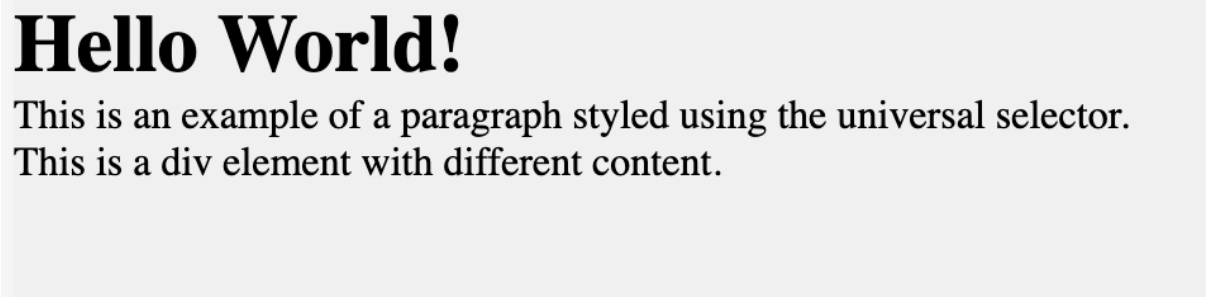
index.html

```
JavaScript
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Universal Selector Example</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <h1>Welcome to My Website</h1>
    <p>This is a paragraph with some sample text.</p>
    <div>This is a div element with some content.</div>
</body>
</html>
```

Style.css

```
JavaScript
/* Universal Selector */
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
    background-color: #f0f0f0;
}
```

Output:



Now let's take an example where we use all the selector in CSS:

Index.html

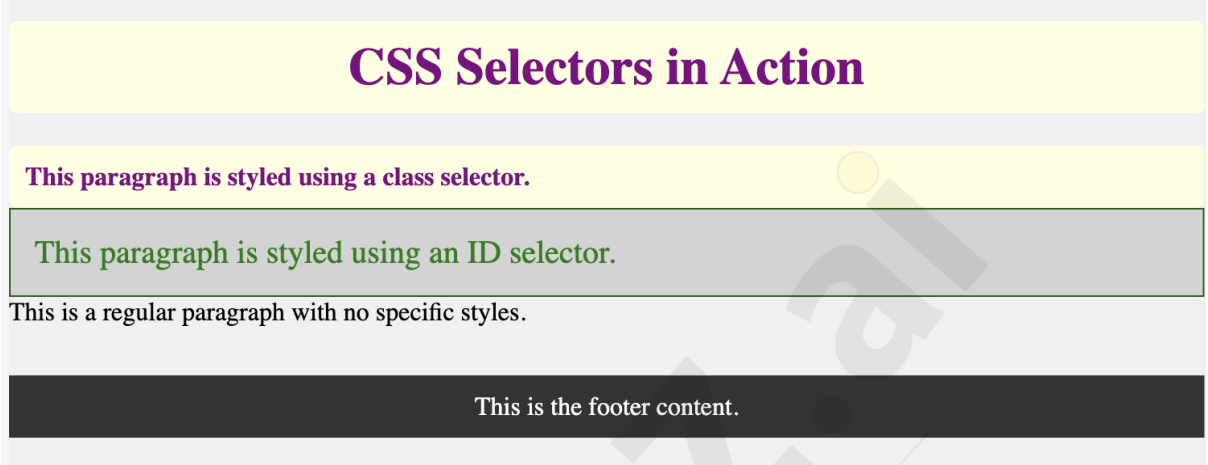
```JavaScript
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Comprehensive CSS Selector Example</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <header id="main-header">
        <h1 class="highlight">CSS Selectors in Action</h1>
    </header>
    <section>
        <p class="highlight">This paragraph is styled using a class
selector.</p>
        <p id="unique-paragraph">This paragraph is styled using an ID
selector.</p>
        <p>This is a regular paragraph with no specific styles.</p>
    </section>
    <footer>
        <div class="footer-text">This is the footer content.</div>
    </footer>
</body>
</html>
```

Style.css

Unset

```
/* Universal Selector */

* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
    background-color: #f0f0f0;
}

/* Tag Name Selector */

h1 {
    color: darkblue;
    text-align: center;
    margin: 20px 0;
}

/* Class Selector */

.highlight {
    color: purple;
    font-weight: bold;
    background-color: lightyellow;
    padding: 10px;
    border-radius: 5px;
}

/* ID Selector */

#unique-paragraph {
    color: green;
    font-size: 20px;
    text-align: justify;
    background-color: lightgray;
    padding: 15px;
    border: 1px solid darkgreen;
}

/* Footer Styling (Tag Name + Class) */

footer .footer-text {
    text-align: center;
    color: white;
    background-color: #333;
    padding: 10px 0;
    margin-top: 30px;
}
```

Output:



Note: We will discuss the above css properties in the upcoming lecture.

Grouping selectors

In CSS, grouping selectors allows you to apply styles to multiple elements with a single rule, reducing the amount of repetitive code. Grouping Selectors involves separating individual selectors with commas (","). Any styles specified within the rule will be applied to all elements that match any of the group selectors.

The different types of grouping selectors in CSS can be -

- Grouping by tag name
- Grouping by class name
- Grouping by ID
- Grouping by combining different selectors

Let's have a look in detail at all the different types of Grouping selectors in CSS

Grouping by Tag name

Selecting multiple tag names together to apply the same styles to all elements with the same tag name .

Let's have a look in detail at all the different types of Grouping selectors in css

Syntax:

```
JavaScript
tagname1, tagname2,tagname3 {
   /* styles properties */
}
```

Example:

index.html

```
JavaScript
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Blog Example</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <header>
        <h1>My Travel Blog</h1>
    </header>
    <article>
        <h2>Exploring the Mountains</h2>
        <p>The mountains are a wonderful escape from the hustle and bustle
of city life.</p>
        <p>On my recent trip, I discovered breathtaking views and hidden
trails.</p>
    </article>
    <article>
        <h2>Beach Getaways</h2>
        <p>There's nothing like the sound of waves crashing on the
shore.</p>
        <p>Relaxing on the beach is the perfect way to unwind.</p>
```

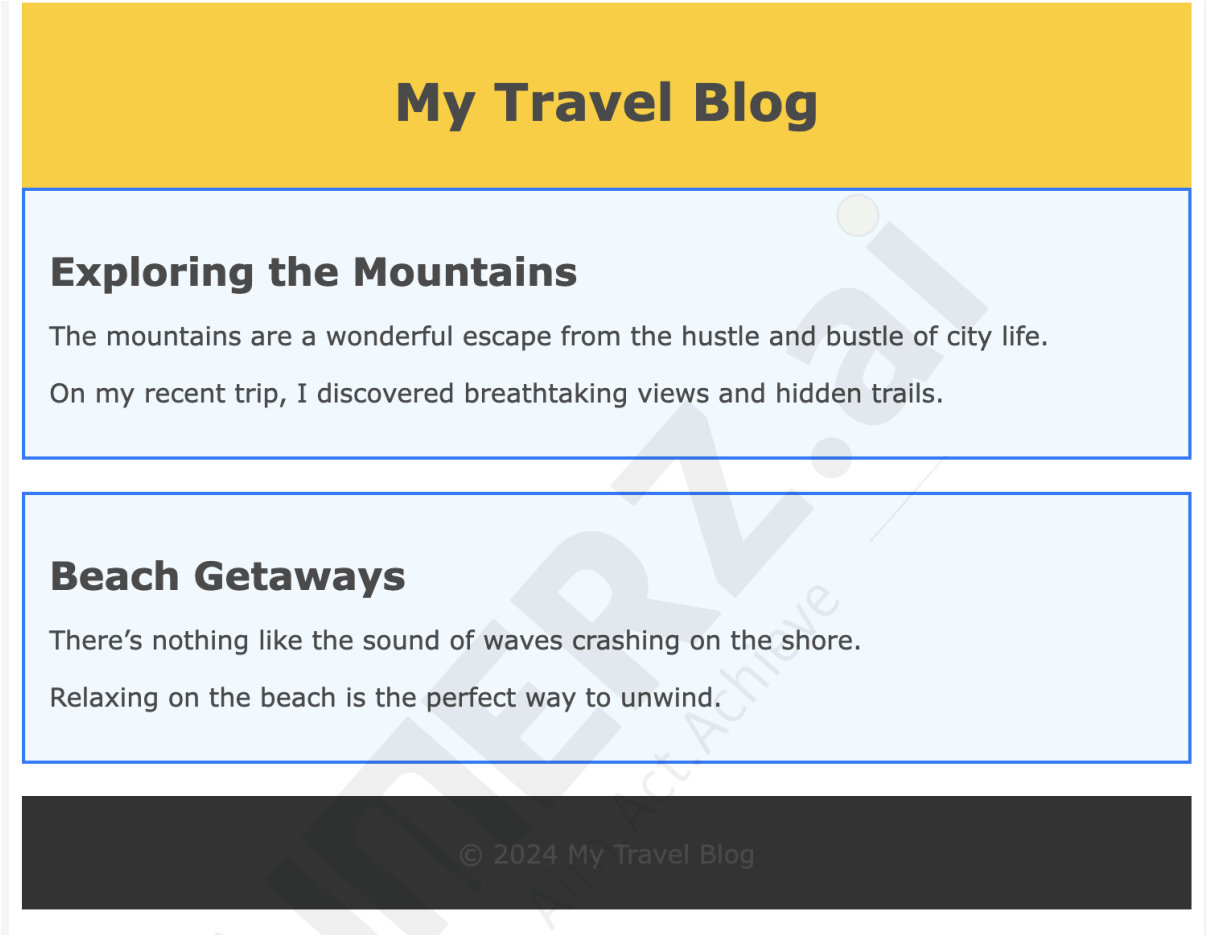
```
    </article>
    <footer>
        <p>© 2024 My Travel Blog</p>
    </footer>
</body>
</html>
```

Styele.css

JavaScript

```
/* Grouping by Tag Name */
h1, h2, p {
    color: #4A4A4A;
    font-family: 'Verdana', sans-serif;
    margin-bottom: 15px;
}

header {
    background-color: #FFCC00;
    padding: 20px;
    text-align: center;
    color: #FFFFFF;
}

article {
    border: 2px solid #007BFF;
    background-color: #F0F8FF;
    padding: 15px;
    margin-bottom: 20px;
}

footer {
    background-color: #333333;
    color: white;
    padding: 10px;
    text-align: center;
}
```

Output:



Grouping by class name

Selecting multiple class names together to apply the same styles to all HTML elements with the corresponding class name.

Syntax:

```javascript
.className, .className, .className {
   /* styles properties */
}
```

Ex:

index.html

```JavaScript
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Grouping Class Name Example</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="box important">This is an important box.</div>
    <div class="box secondary">This is a secondary box.</div>
    <div class="box highlight">This is a highlighted box.</div>
</body>
</html>
```

Style.css

```JavaScript
/* Grouping by Class Name */
.important, .secondary, .highlight {
    border-radius: 10px;
    padding: 20px;
    font-family: Arial, sans-serif;
    margin-bottom: 15px;
}

/* Specific Styles for Each Class */
.important {
    background-color: #FF5733;
    color: white;
}

.secondary {
    background-color: #3498DB;
    color: white;
}

.highlight {
    background-color: #F1C40F;
    color: black;
}
```

Output:

Grouping By ID

Although IDs should be unique in HTML , you can still group them in css selectors.

Syntax:

```
JavaScript
idName1, idName2, idName3 {
   /* styles properties */

}
```

Example;

index.html

```
JavaScript
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Grouping by ID Example</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div id="header">Header Section</div>
```

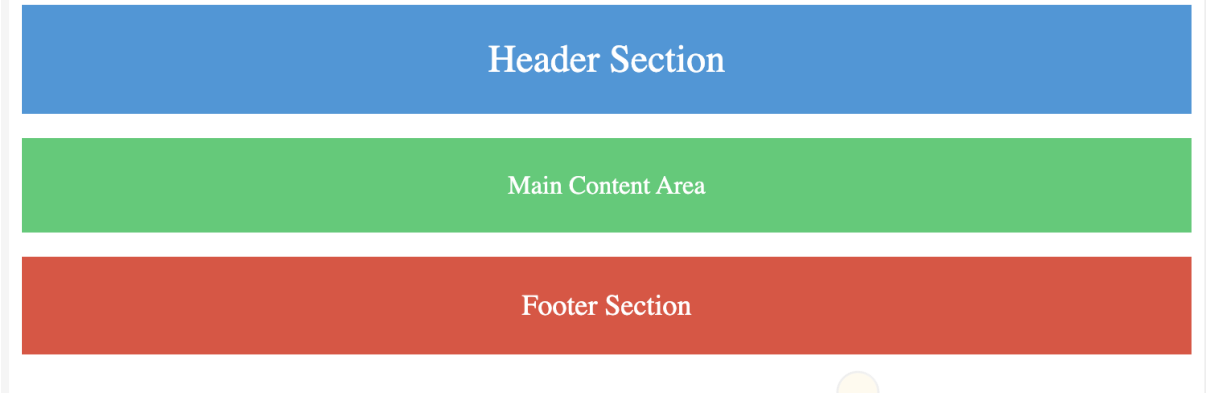
```
    <div id="main-content">Main Content Area</div>
    <div id="footer">Footer Section</div>
</body>
</html>
```

Style.css

JavaScript

```
/* Grouping by ID */
#header, #main-content, #footer {
    background-color: #3498db;
    color: white;
    padding: 20px;
    text-align: center;
    margin-bottom: 15px;
}

/* Specific styles for each ID */
#header {
    font-size: 24px;
}

#main-content {
    background-color: #2ecc71;
}

#footer {
    background-color: #e74c3c;
    font-size: 18px;
}
```

Output:



Grouping by combining different selectors

Selecting multiple different selectors' names together to apply the same styles to all the corresponding HTML elements respectively.

Syntax:

```
JavaScript
tagName, class, id {
   /* styles properties */
}
```

Example:
index.html

```
JavaScript
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Combining Different Selectors with Colors</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <header id="main-header">Welcome to My Colorful Website</header>
```

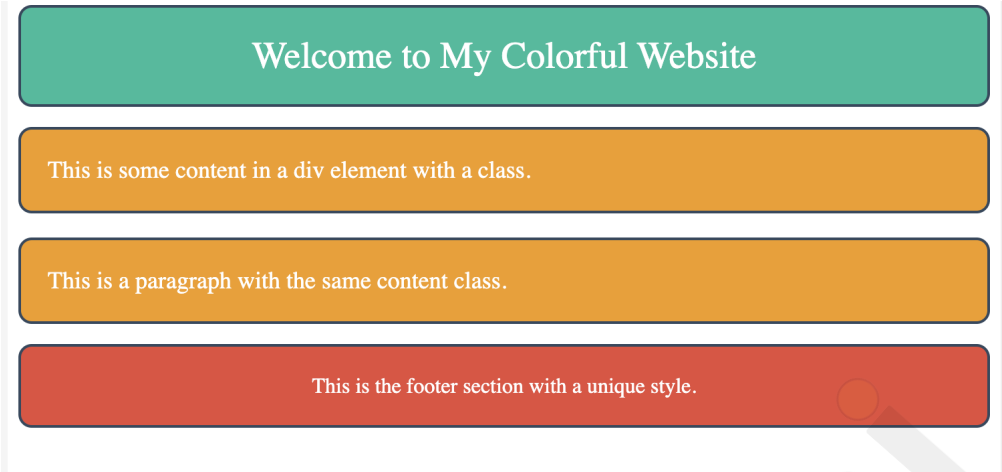
```
    <div class="content">This is some content in a div element with a
class.</div>
    <p class="content">This is a paragraph with the same content class.</p>
    <footer id="footer">This is the footer section with a unique
style.</footer>
</body>
</html>
```

style.css

JavaScript

```
/* Grouping different selectors: ID, Class, and Tag Name */
#main-header, .content, footer {
    background-color: #ffdd57;
    color: #2c3e50;
    padding: 20px;
    margin-bottom: 15px;
    border-radius: 10px;
    border: 2px solid #34495e;
}

/* Individual Styles for Each Selector */
#main-header {
    background-color: #1abc9c;
    color: white;
    font-size: 28px;
    text-align: center;
}

.content {
    background-color: #f39c12;
    color: white;
    font-size: 18px;
}

#footer {
    background-color: #e74c3c;
    color: white;
    font-size: 16px;
    text-align: center;
}
```

Output:



Three commonly used CSS Selectors

The **class selector,** the **id selector**, and the **tagname selector** are the three most commonly used CSS selectors.The choice between the element, id, and class selectors depends on the level of specificity and uniqueness required for our styling.

The **class selectors** are mostly preferred in many cases over the two selectors (id and element selectors) as it is versatile and suitable for styling multiple elements that share the same class, promote consistency, reusability, and maintainability in the styles, offer good balance of specificity and are an excellent choice for styling similar elements across a website.

The **tagname selector** is secondly preferred as the tagname should be used when you want to establish global styles for standard HTML elements. They target all instances of a particular HTML element type. Tagname selectors are well-suited for setting default styles for elements like paragraphs, headings, lists, and other standard HTML elements.

The **id selector** is the third preferred selector, as selectors should be used carefully and considered as a last resort and They offer high specificity and should only target unique elements on a page.

Introduction to a class naming convention

Class naming conventions are a set of guidelines that help web developers name CSS classes in a consistent, meaningful, and structured way. Using well-organized and descriptive class names makes your code more readable, maintainable, and scalable. It's crucial for any project, especially as it grows in complexity.

Why Use Class Naming Conventions?

1. **Consistency:** A unified approach to naming helps developers (or teams) understand and navigate the CSS code easily.
2. **Readability:** Clear, meaningful names make it easier to understand what each class does.
3. **Maintainability:** Organized class names reduce confusion and make the code easier to modify or update in the future.
4. **Collaboration:** When working in teams, following a naming convention ensures everyone can understand and work with the code without having to decipher another person's logic.

Importance of Consistent and Meaningful Class Names

1. **Avoid Confusion:** Unclear names like .box1, .red-bg, or .header-main-content-upper-left can be confusing, especially if the project evolves or someone else takes over.
2. **Predictability:** When naming is consistent, you can easily guess or search for the right class. For example, a .btn-primary class would likely be the main button style throughout the project.
3. **Reusability:** With proper names, you can reuse classes across multiple elements without having to write duplicate CSS.
4. **Long-Term Efficiency:** Properly named classes help you understand the structure of your page, making future updates faster and less error-prone.

Popular Naming Conventions for CSS

There are several popular naming conventions used in CSS, each designed to create structure in your styles. Let's look at some of the most popular ones:

1. BEM (Block, Element, Modifier)

BEM is one of the most widely-used class naming conventions. It stands for **Block, Element, Modifier** and helps you create reusable and modular code.

- **Block**: A standalone component that represents a distinct part of the interface (e.g., a navigation bar, button, or form).
- **Element**: A part of the block that has a specific function (e.g., a button within a form).
- **Modifier**: A variant of a block or element that changes its appearance or behavior (e.g., a primary button vs. a secondary button).

BEM Structure:

- **Block**: .block
- **Element**: .block__element
- **Modifier**: .block--modifier or .block__element--modifier

Example:

Let's say we have a button component with a few variations:

```
JavaScript
<button class="btn btn--primary">Submit</button>
<button class="btn btn--secondary">Cancel</button>
```

Here's the corresponding CSS:

```
Unset
/* Block */
.btn {
```

```
  padding: 10px 20px;
  font-size: 16px;
  border-radius: 5px;
}

/* Modifier for primary button */

.btn--primary {
  background-color: blue;
  color: white;
  border: none;
}

/* Modifier for secondary button */

.btn--secondary {
  background-color: grey;
  color: white;
  border: none;
}
```

In the above example, .btn is the **block**, representing a button. .btn--primary and .btn--secondary are **modifiers** that change the appearance of the block (different colors for different button types).

2. **SMACSS (Scalable and Modular Architecture for CSS)**

SMACSS is another popular approach that emphasizes modularity. It divides CSS into 5 categories: base, layout, module, state, and theme.

- **Base**: Default styles for elements (like resetting margins and paddings).
- **Layout**: Styles that define the structure (like header, footer, sidebar).
- **Module**: Reusable components (like buttons, cards, and forms).
- **State**: Styles that define how elements look in different states (like is-active, is-hidden).
- **Theme**: Styles related to theme changes (like color schemes).

```JavaScript
<div class="layout-header">
  <button class="btn is-active">Home</button>
  <button class="btn">Profile</button>
</div>
```

Style.css

```JavaScript
/* Layout for header */
.layout-header {

  background-color: #333;
  padding: 20px;
}

/* Module: button */

.btn {
  background-color: grey;
  color: white;
  border: none;
}

/* State: active button */

.is-active {
  background-color: blue;
}
```

3. **OOCSS (Object-Oriented CSS)**

OOCSS focuses on making CSS more modular and reusable by separating structure (layout) from the skin (visual appearance). The idea is to avoid repetition and write CSS that can be reused across multiple components.

Ex:

```JavaScript
<div class="card card--highlight">
  <h2 class="card__title">Article Title</h2>
```

```
  <p class="card__content">This is the content of the article.</p>
</div>
```

Style.css

```
Unset
/* Structure */

.card {
  padding: 20px;
  border: 1px solid #ddd;
}

/* Visual appearance (skin) */

.card--highlight {
  background-color: yellow;
}
```


Color

In CSS, colors refer to the visual aspects of an element, such as the **foreground, background, text, or border color.** They are a fundamental part of web design, helping to create visually appealing and attractive layouts.

CSS colors can be specified in various formats, including color names, **hexadecimal values, RGB, RGBA, HSL, and HSLA.** CSS also provides predefined color keywords, which are easier to use and remember compared to color codes. By defining the color of an element in CSS, designers ensure a consistent visual design across all pages of a website.

Now let's explore the different CSS color formats:

Color Name

CSS provides a set of predefined color keywords, such as **"red," "blue," "green," "yellow," "aqua," "black,"** and many more. These color keywords are **not case-sensitive** and can be used in any CSS property that accepts a color value.

```
JavaScript
p {
  color: red;
  background-color: yellow;
}
```

In the above example , the text colour of the <p> element is set to red. Also the background color of the <p> element is set to yellow.

hex color

In CSS, hexadecimal notation (hex color) is a method of specifying color using a **six-digit code** that represents the amount of **red, green, and blue (RGB) in a color.** Each pair of digits in the code represents the intensity of each color channel, with values ranging from **00 (0) to FF (255)** in hexadecimal notation.

To use hexadecimal notation in CSS, you can use the **pound sign (#)** followed by the six-digit code

```
JavaScript
h1 {
  color: #FF5733; /* A shade of orange */
  background-color: #1C1C1C; /* Dark gray background */
}
```

In the above example, each part of the code breaks down like this:

- #FF (red),
- 57 (green),
- 33 (blue)

RGB

In CSS, the rgb() function is used to specify colors by providing values for red, green, and blue. The RGB model is an additive color model, meaning colors are created by combining different intensities of red, green, and blue light. Each color channel can have a value between 0 (no light) and 255 (maximum intensity).

Ex:

```
JavaScript
h2 {
  color: rgb(255, 99, 71); /* A shade of tomato red */
  background-color: rgb(0, 128, 128); /* Teal background */
}
```

In the above example, **rgb(255, 99, 71**) sets the text color to a shade of tomato red. **rgb(0, 128, 128)** sets the background color to teal.

RGBA

Similar to the rgb() function, the rgba() function in CSS specifies colors with an additional alpha value for transparency. The alpha value controls the opacity of the color, with 0 being fully transparent and 1 being fully opaque.

Syntax:

```
JavaScript
rgba(red, green, blue, alpha)
```

- red: The red component (0 to 255)
- green: The green component (0 to 255)
- blue: The blue component (0 to 255)
- alpha: The opacity level (0 to 1)

Example:

```
JavaScript
div {
  color: rgba(255, 99, 71, 0.7); /* Semi-transparent tomato red text */
```

```
  background-color: rgba(0, 128, 128, 0.5); /* Semi-transparent teal
background */
}
```

HSL in CSS

In CSS, the **HSL** (Hue, Saturation, Lightness) color model represents colors based on three attributes: **hue**, **saturation**, and **lightness**. This model is designed to be more intuitive, allowing you to specify colors in terms of their perceived attributes.

- **Hue**: Represents the type of color and is specified as a degree on the color wheel (0 to 360°). For example, 0° is red, 120° is green, and 240° is blue.
- **Saturation**: Represents the intensity or purity of the color. A value of 100% is fully saturated (pure color), while 0% is completely gray.
- **Lightness**: Represents the brightness of the color. A value of 0% is black, 100% is white, and 50% is the pure color with no lightness or darkness added.

Syntax:

JavaScript
```
hsl(hue, saturation, lightness)
```

In the above example ,

- **hue**: A degree value (0 to 360)
- **saturation**: A percentage value (0% to 100%)
- **lightness**: A percentage value (0% to 100%)

Ex:

JavaScript
```
p {
  color: hsl(120, 100%, 50%); /* Pure green color */
  background-color: hsl(240, 100%, 50%); /* Pure blue background */
}
```

HSLA in CSS

The hsla() function extends the HSL color model by adding an **alpha** value for transparency. This allows you to specify colors with varying degrees of opacity.

- **Hue**: Represents the color on the color wheel (0 to 360°).
- **Saturation**: Indicates the intensity of the color (0% to 100%).
- **Lightness**: Determines the brightness of the color (0% to 100%).
- **Alpha**: Represents the opacity of the color, where 0 is fully transparent and 1 is fully opaque.

Syntax:

```javascript
hsla(hue, saturation, lightness, alpha)
```

- **hue**: A degree value (0 to 360)
- **saturation**: A percentage value (0% to 100%)
- **lightness**: A percentage value (0% to 100%)
- **alpha**: A value between 0 (fully transparent) and 1 (fully opaque)

Example:

```javascript
div {
  color: hsla(120, 100%, 50%, 0.7); /* Semi-transparent green text */
  background-color: hsla(240, 100%, 50%, 0.3); /* Semi-transparent blue
background */
}
```

Opacity in CSS

The opacity property in CSS controls the transparency of an element. It allows you to set how see-through an element is, with the following range of values:

- 0.0: Completely transparent (invisible)
- 1.0: Completely opaque (fully visible)

Syntax:

```JavaScript
element {
  opacity: value;
}
```

- value: A number between 0.0 (fully transparent) and 1.0 (fully opaque)

Ex:

```Java
div {
  opacity: 0.5; /* 50% transparent */
}
```

opacity: 0.5 makes the element 50% transparent. This means that the element will be partially see-through, allowing the content behind it to be visible through it.

Example: Consider below image

```JavaScript
<img src="photo.jpg" />
```

Let's alter image opacity with 0.1, 0.5 and 0.8.

img {
opacity: 0.1;
}
	img {
	opacity: 0.5;
	}
		img {
		opacity: 0.8;
		}

Background

In CSS, the background property is a shorthand that allows you to set multiple background properties for an element in a single declaration. It helps simplify the process of styling backgrounds without writing separate declarations for each property.

The following are the different properties you can use with the background shorthand to style the background of an HTML element:

1. **Background Color in CSS**

The **background-color** property in CSS is used to set the background color of an element on a webpage. This property allows you to choose a specific color that will fill the background area behind the content of the element.

Example:

```javascript
div {
  background-color: #f0f8ff; /* AliceBlue background color */
}
```

2. Background image

CSS **background-image** is a property used to set an image as the background of an element on a webpage. It allows you to display a picture or graphic behind the content of the element.

Syntax

```
Background-image : url(<image-url>);
```

<image-url> is the path to the image you want to use as the background. It can be a relative path (e.g., **"images/my-background.jpg"**) or an absolute URL (e.g., **"https://picsum.photos/200/300"**).

Example

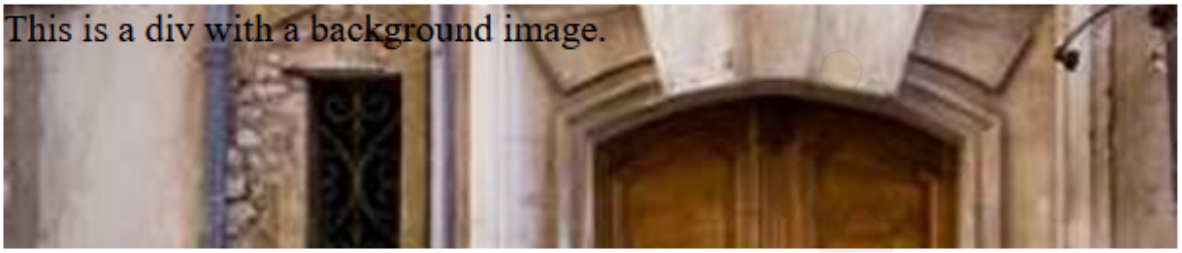
index.html

```
<div class="my-div">
       <p>
            This is a div with a background image.
      </p>
 </div>
```

style.css

```
Unset

.my-div {
     background-image: url("https://picsum.photos/500/300");
     }
```

Browser output



We apply the CSS background-image property to the div with the class "my-div" and set the value to the URL of the image we want to use as the background and also set the height 100px to “my-div” to see the background properly.

3. Background repeat

CSS **background-repeat** is a property used to control how a background image is repeated within an element when the image's size is smaller than the element's size. It allows you to decide whether the background image should repeat horizontally, vertically, both, or not at all.

Syntax

```
JavaScript
background-repeat: repeat | repeat-x | repeat-y | no-repeat;
```

repeat: This is the default value. It means the background image will be tiled and repeated both horizontally and vertically to cover the entire background area.
repeat-x: The background image will only repeat horizontally (left to right) to cover the width of the element.

repeat-y: The background image will only repeat vertically (top to bottom) to cover the height of the element.
no-repeat: The background image will not be repeated, and it will appear only once in the background.

Example

index.html

```
Unset
<div class="my-div">
      <p>
          This is a div with a background image.
     </p>
 </div>
```

Example for repeat-x

style.css

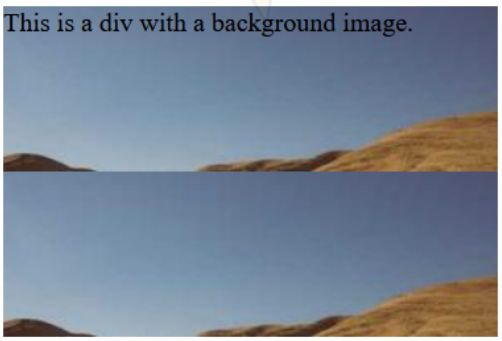
```
Unset

.my-div {
     background-image: url("https://picsum.photos/200/300");
     Background-repeat:repeat-x;
     height: 200px;
     }
```

Browser output

Example for repeat-y style.css

```
Unset
.my-div {
      background-image: url("https://picsum.photos/300/100");
      Background-repeat:repeat-y;
      height: 200px;
      }
```

Browser output

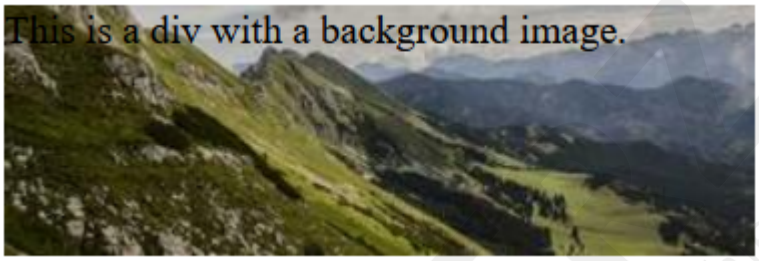


Example for no-repeat

style.css

```
Unset
.my-div {
      background-image: url("https://picsum.photos/300/100");
      Background-repeat:no-repeat;
      height: 200px;
      }
```

Browser output



4. Background attachment

CSS **background-attachment** is a property that controls whether a background image scrolls with the content or remains fixed in its position as the user scrolls the webpage. It determines whether the background image is attached to the viewport or to the element's content.

Syntax

```
Unset
Background-attachment: scroll | fixed;
```

scroll: This is the default value. The background image will scroll along with the content as the user scrolls down or up the webpage. It will move relative to the element and will stay positioned behind the content.

fixed: The background image will remain fixed in its position within the viewport, regardless of the user's scrolling. It will appear as if the background is attached to the browser window rather than the element.

Example;

index.html

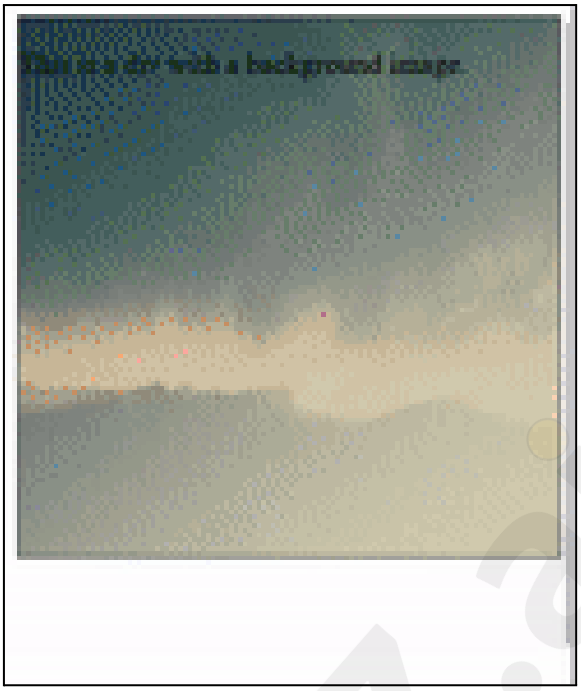
```
Unset
<div class="my-div">
  <p>This is a div with a fixed background image.</p>
</div>
```

Example for background-attachment: scroll;

style.css

```
Unset
.my-div {
  background-image: url("https://picsum.photos/300/300");
  background-repeat: no-repeat; /* The image will not repeat
*/
  background-attachment: scroll; /* The image will remain
fixed in the viewport */
 height: 300px;
 border: solid 1px;
}
```

Browser output

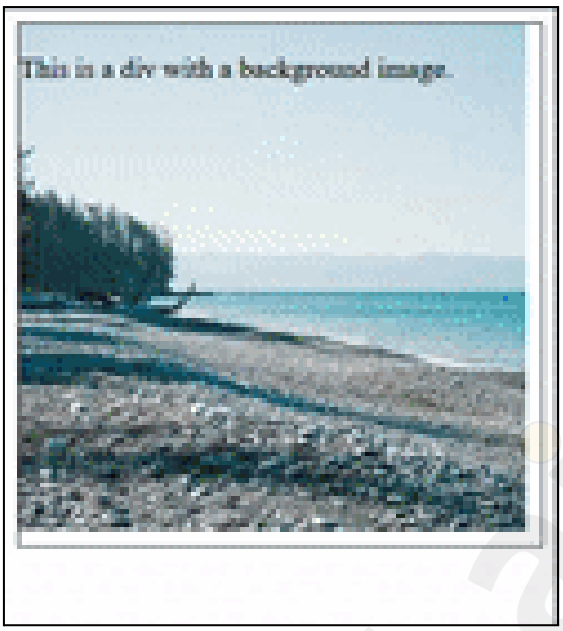


Example for background-attachment: fixed;

style.css

```
Unset
.my-div {
  background-image: url("https://picsum.photos/300/300");
  background-repeat: no-repeat; /* The image will not repeat
*/
  background-attachment: fixed; /* The image will remain fixed
in the viewport */
 height: 300px;
 border: solid 1px;
}
```

Browser output



we apply the CSS background-attachment property to the div with the class "**my-div**" and set the value to fixed. As a result, the background image will stay in a fixed position as the user scrolls the webpage. And we also added height and border to “my-div” to understand this example properly. **Note:** we will learn about css border property in a later section in more detail.

5. Background position

CSS **background-position** is a property used to set the starting position of a background image within an element. It allows you to specify where the background image should be placed relative to the element's background area.

Syntax

```
Unset
Background-position: <position>;
```

<position> can be defined using different units and values to position the background image. It can use keywords such as **top**, **bottom**, **left**, **right**, and **center**, or it can use **specific length** or **percentage** values.

Example

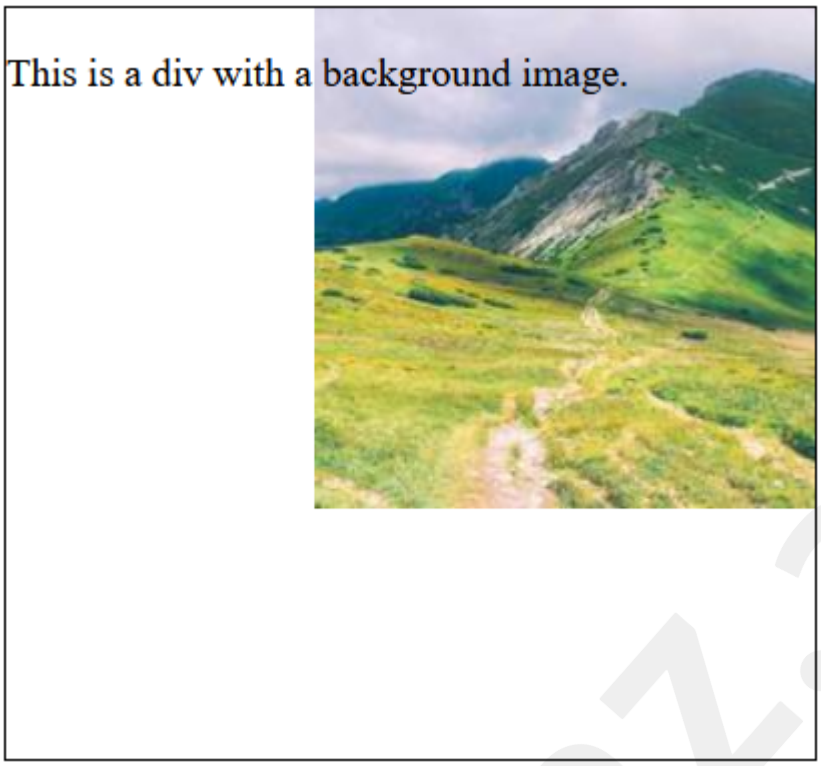
index.html

```
Unset
<div class="my-div">
  <p>This is a div with a positioned background image.</p>
</div>
```

Style.css

```
Unset
.my-div {
  background-image: url("https://picsum.photos/200/300");
  background-repeat: no-repeat; /* The image will not repeat
*/
  background-position: top right; /* The image will be
positioned at the top right corner */
  height: 300px;
  border: solid 1px;
}
```

Browser output:



we apply the CSS background-position property to the div with the class "**my-div**" and set the value to top right. This will position the background image at the top right corner of the element

Here are some examples of different background-position values:

1. Example for position bottom

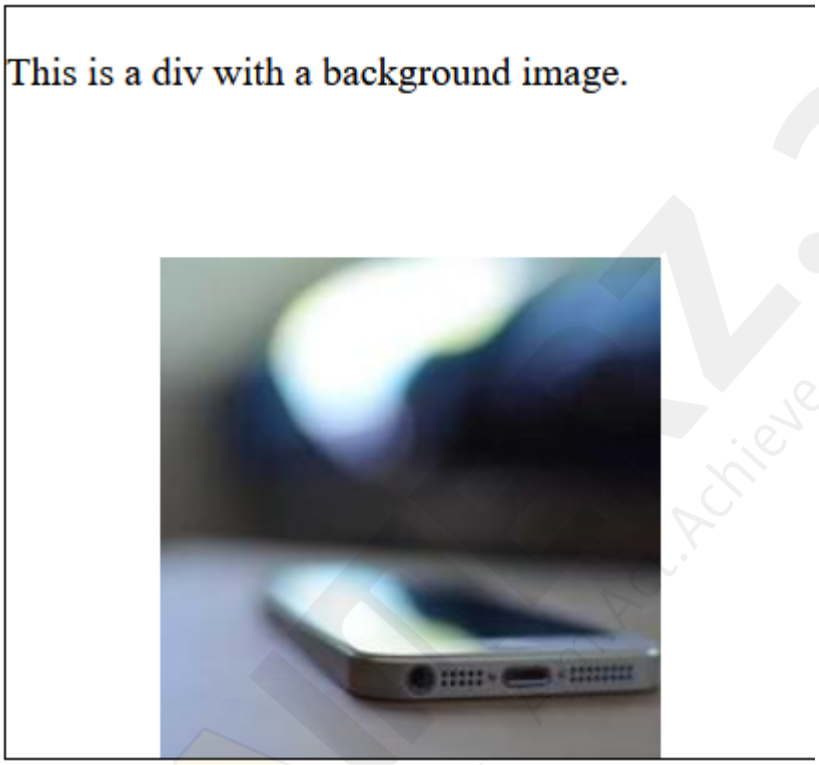
Style.css

Unset

```
.my-div {
  background-image: url("https://picsum.photos/200/300");
  background-repeat: no-repeat; /* The image will not repeat
*/
  background-position: bottom; /* The image will be positioned
at the top right corner */
```

```
  height: 300px;
  border: solid 1px;
}
```

Browser output



2. Example for percentage values

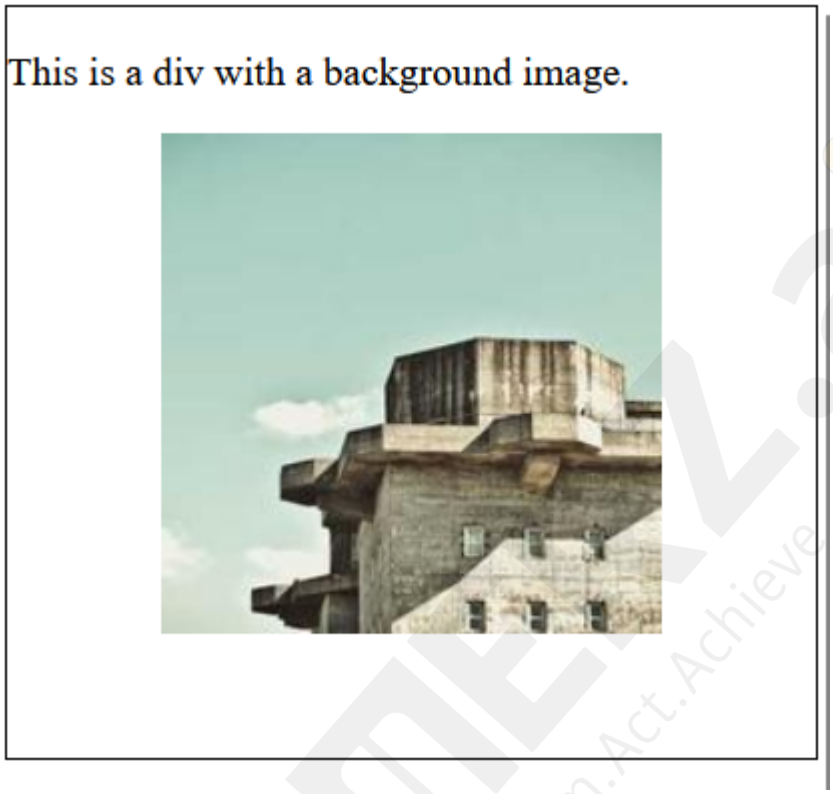
Style.css

Unset

```
.my-div {
  background-image: url("https://picsum.photos/200/300");
  background-repeat: no-repeat; /* The image will not repeat
*/
  background-position:50% 10%; /* The image will be positioned
at the top right corner */
  height: 300px;
  border: solid 1px;
```

```
}
```

Browser output



6. Background shorthand

CSS background shorthand is a way to set **multiple background-related properties in a single line of code**. Instead of writing separate lines for each background property (like background-image, background-repeat, background-position, etc.), you can combine them into one concise declaration.

Syntax

```
Unset
background: <background-color> <background-image>
<background-repeat> <background-attachment>
<background-position>;
```

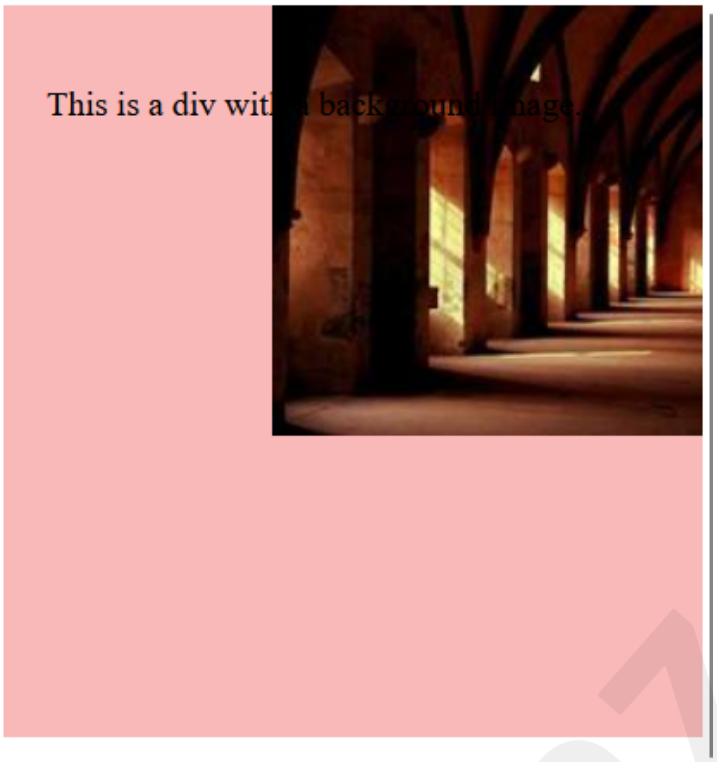
Example

index.html

```
Unset
<div class="my-div">
  <p>This is a div with a background image using
shorthand.</p>
</div>
```

```
Unset
.my-div {
  background: #f2f2f2 url("https://picsum.photos/200/300")
no-repeat top right;
  padding: 20px; /* Just to create some space around the
content for better visualization */
  color: black; /* Adding black text color to make it visible
on top of the background */
}
```

Browser output



we use the CSS background shorthand to set the background color to #f2f2f2, use an image as the background, set it to not repeat (no-repeat), and position it at the top right corner of the element.

Typography in CSS

Typography plays a crucial role in web design, affecting readability, aesthetics, and user experience.Lets understand one by one with example:

Text Color

Adding color to a text in css can be achieved by using **color:** property. The value to this property can be a from the following:

- Named colors or predefined color keywords

- RGB
- RGBa
- Hexadecimal notation
- HSL
- HSLa

index.html

```
JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="style.css" />
    <title>Typography</title>
  </head>
  <body>
    <p id="color1">Welcome to Aimerz!</p>
    <p id="color2">Welcome to Aimerz!</p>
    <p id="color3">Welcome to Aimerz!</p>
    <p id="color4">Welcome to Aimerz!</p>
    <p id="color5">Welcome to Aimerz!</p>
    <p id="color6">Welcome to Aimerz!</p>
  </body>
</html>
```

style.css

```
JavaScript
#color1 {
  color: red; /* Named color */
}

#color2 {
  color: #00ff00; /* Hexadecimal */
}

#color3 {
  color: rgb(55, 155, 255); /* RGB */
}

#color4 {
  color: rgba(255, 255, 100, 0.8); /* RGBA (with alpha) */
}
```

```
#color5 {
  color: hsl(16, 88%, 54%); /* HSL */
}

#color6 {
  color: hsla(12, 100%, 20%, 0.5); /* HSLA (with alpha) */
}
```

Output:

If we want to apply different colors within a single paragraph, wrap specific words in a ** tag:**

index.html

```
JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
   <link rel="stylesheet" href="styles.css">
    <title>Typography</title>
  </head>
  <body>
  <p>Welcome to <span class="highlight">Aimerz</span>. Let's learn
together!</p>
  </body>
</html>
```

styles.css

```
JavaScript
p {
  color: blue;
}
.highlight {
  color: red;
}
```

Output:

Text-decoration

In CSS, the text-decoration property is used to style the appearance of text, such as adding underlines, overlines, or striking through text.

Let's understand one by one with example:

1. **text-decoration: none;**

   It removes any decoration (like underlines) from the text.

```JavaScript
p {
  text-decoration: none;
}
```

2. **text-decoration: underline;**

   It adds an underline beneath the text.

```JavaScript
h1 {
  text-decoration: underline;
}
```

3. **text-decoration: overline;**

   It adds a line above the text

```JavaScript
h2 {
  text-decoration: overline;
}
```

4. text-decoration: line-through;

It adds a line through the middle of the text (strikethrough effect).

```JavaScript
span {
  text-decoration: line-through;
}
```

5. text-decoration: underline overline;

Adds both an underline and an overline to the text.

```JavaScript
h3 {
  text-decoration: underline overline;
}
```

6. text-decoration-color

Changes the color of the text decoration (underline, overline, etc.).

```JavaScript
p {
  text-decoration: underline;
  text-decoration-color: red;
}
```

7. text-decoration-style

Changes the style of the text decoration (solid, dotted, dashed, etc.).

```javascript
p {
  text-decoration: underline;
  text-decoration-style: dashed;
}
```

Now let's take an example to understand better:

Online Store Pricing Page

Index.html

```javascript
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Online Store - Special Offer</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <header>
        <h1>Welcome to Our Store</h1>
    </header>

    <section class="pricing">
        <h2>Special Offer</h2>

        <p class="old-price">Original Price: <span>$99.99</span></p>

        <p class="discount-price">Discounted Price: <span>$59.99</span></p>

        <p class="limited-offer">Limited Time Offer: <a href="#">Shop
Now</a></p>
    </section>

    <footer>
        <p>© 2024 Online Store. All rights reserved.</p>
    </footer>
</body>
</html>
```

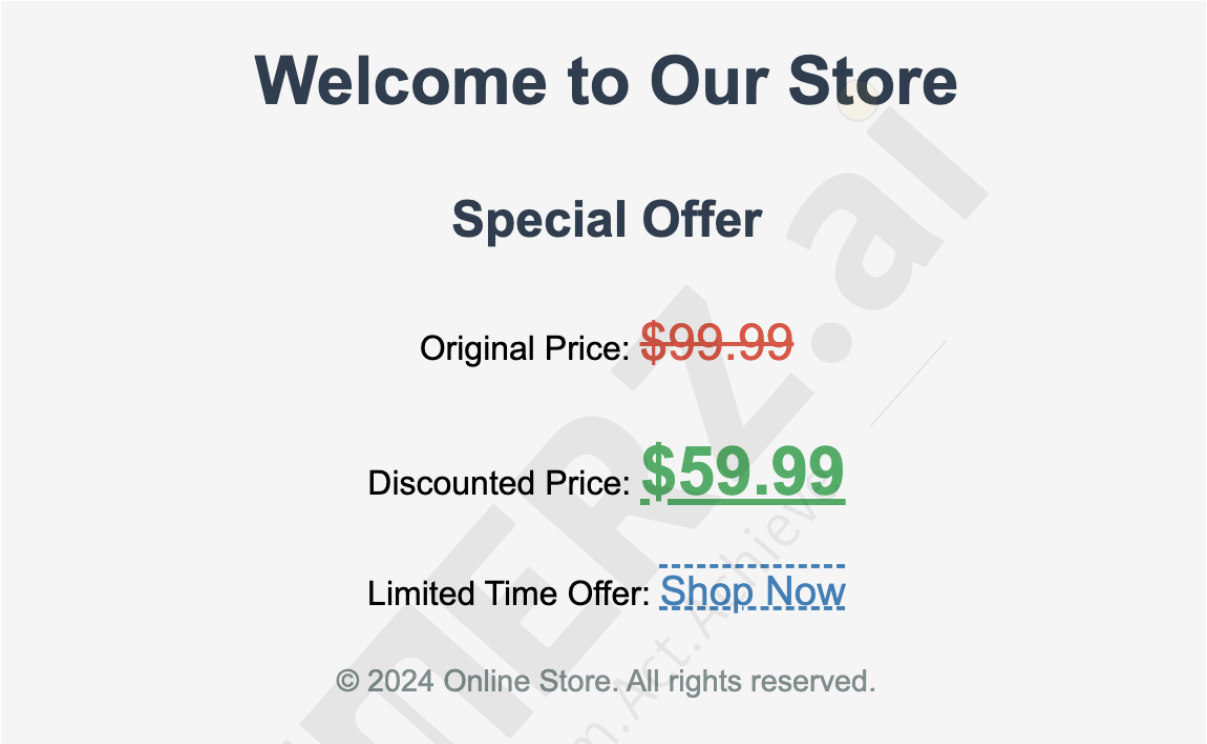
Styles.css

JavaScript

```
/* General Styles */

body {
    font-family: Arial, sans-serif;
    line-height: 1.6;
    background-color: #f5f5f5;
    padding: 20px;
    text-align: center;
}

h1, h2 {
    color: #2c3e50;
}

/* Old Price (strikethrough) */

.old-price span {
    text-decoration: line-through;
    color: #e74c3c;
    font-size: 1.5rem;
}

/* Discounted Price (underline) */

.discount-price span {
    text-decoration: underline;
    text-decoration-color: #27ae60;
    font-size: 2rem;
    color: #27ae60;
    font-weight: bold;
}

/* Limited Offer (underline overline) */

.limited-offer a {
    text-decoration: underline overline;
    text-decoration-color: #2980b9;
    text-decoration-style: dashed;
    font-size: 1.2rem;
    color: #2980b9;
}

/* Footer */
footer p {
    font-size: 0.9rem;
    color: #7f8c8d;
```

```
}
```

Output:



Text Alignment

The text-align property is used for alignment of text in css.

JavaScript

```
p {
  text-align: left;
  text-align: right;
  text-align: center;
  text-align: justify;
}
```

Let's take an example to understand better:

index.html

```JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <p class="left">
      Lorem ipsum dolor sit amet consectetur adipisicing elit. Enim
molestias
      nostrum neque aperiam ipsam numquam nobis perferendis dolores
voluptate
      officia.
    </p>
    <p class="right">
      Lorem ipsum dolor sit amet consectetur adipisicing elit. Molestias at
      sequi nostrum quod beatae ipsam architecto voluptatibus accusamus
libero
      quis.
    </p>
    <p class="center">
      Lorem ipsum dolor sit amet consectetur adipisicing elit. Perferendis
      maxime rerum, qui obcaecati numquam iste quia perspiciatis dolor
corporis
      optio?
    </p>
    <p class="justify">
      This is an example of justified text alignment. The text is stretched
so
      that it fits the width of the container, with both the left and right
      sides aligned.
    </p>
  </body>
</html>
```

Style.css

```Unset
/* Left aligned text */
```

```
p.left {
  text-align: left;
}

/* Right aligned text */

p.right {
  text-align: right;
}

/* Center aligned text */

p.center {
  text-align: center;
}

/* Justified text */

p.justify {
  text-align: justify;
}
```

Output:

Lorem ipsum dolor sit amet consectetur adipisicing elit. Enim molestias nostrum neque aperiam ipsam numquam nobis perferendis dolores voluptate officia.

Lorem ipsum dolor sit amet consectetur adipisicing elit. Molestias at sequi nostrum quod beatae ipsam architecto voluptatibus accusamus libero quis.

Lorem ipsum dolor sit amet consectetur adipisicing elit. Perferendis maxime rerum, qui obcaecati numquam iste quia perspiciatis dolor corporis optio?

This is an example of justified text alignment. The text is stretched so that it fits the width of the container, with both the left and right sides aligned.

Text Transformation

Text transformation is to make a text **uppercase or lowercase** or to **capitalize** using CSS. We have **text-transform** to achieve this in CSS. The value for these property are **lowercase, uppercase ,and capitalize.**

```
text-transform: uppercase;
text-transform: lowercase;
text-transform :capitalize;
```

Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <style>
    /* Uppercase transformation */

    p.uppercase {
      text-transform: uppercase;
    }

    /* Lowercase transformation */

    p.lowercase {
      text-transform: lowercase;
    }

    /* Capitalizes the first letter of each word */

    p.capitalize {
      text-transform: capitalize;
    }

    /* No transformation */
    p.normal {
      text-transform: none;
    }
  </style>
</head>
<body>
  <p class="uppercase">This text will be converted to uppercase.</p>
  <p class="lowercase">THIS TEXT WILL BE CONVERTED TO LOWERCASE.</p>
  <p class="capitalize">this text will be capitalized.</p>
  <p class="normal">This text has no transformation.</p>
```

```
</body>
</html>
```

Output:

THIS TEXT WILL BE CONVERTED TO UPPERCASE.

this text will be converted to lowercase.

This Text Will Be Capitalized.

This text has no transformation.

Text Spacing

In CSS, text spacing refers to the space between characters, words, or lines of text. Several properties control this spacing, allowing for more readable and visually appealing text. The main properties used to adjust text spacing are:

1. **letter-spacing:**
   The letter-spacing property controls the **space between individual letters or characters.**

Syntax:

```
JavaScript
letter-spacing: value;
```

You can set letter spacing using **values** like 2px to increase space, -1px to decrease it, or "normal" for the default spacing.

2. **word-spacing**

   The word-spacing property controls the **space between words** in text.

Syntax:

```
JavaScript
word-spacing: value;
```

You can adjust word spacing with a length value (like 5px to add space or -2px to reduce it) or use "normal" for default spacing.

3. **line-height**

   The line-height property controls the vertical space between lines of text within a block. It helps improve readability, especially in paragraphs with longer text.

Syntax:

```
JavaScript
line-height: value;
```

You can set line height with a **number** (e.g., 1.5 times the font size), a specific **length** (e.g., 20px), or a **percentage** (e.g., 150% of the font size). **"Normal"** uses the default line height, usually about 1.2 times the font size.

4. **text-indent**

   The text-indent property is used to control the indentation of the first line of a block of text.

Syntax:

```javascript
text-indent: value;
```

Value: Can be a length (e.g., `50px`, `2em`) or a percentage.

- **Positive values (e.g., `50px`):** Indent the first line to the right.
- **Negative values (e.g., `-20px`):** Indent the first line to the left.
- **Percentage**: Indents a percentage of the container's width.

5.White-space:

The white-space property controls how **white space (spaces, tabs, and line breaks) is handled in text.**

Syntax:

```javascript
white-space: value;
```

Values:

- **normal**: Collapses multiple spaces into one and wraps text when necessary.
- **nowrap**: Prevents the text from wrapping onto a new line.
- **pre**: Preserves spaces, tabs, and line breaks (acts like the pre HTML element).
- **pre-wrap**: Preserves spaces and line breaks, but allows text wrapping when necessary.
- **pre-line**: Collapses spaces, but preserves line breaks.

Let's take an example

```java
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <style>
```

```
    /* Controls the space between letters */

    p.letter-spacing {
      letter-spacing: 3px; /* Adds 3px space between each letter */
    }

    /* Controls the space between words */

    p.word-spacing {
      word-spacing: 10px; /* Adds 10px space between words */
    }

    /* Controls the space between lines */

    p.line-height {
      line-height: 2; /* Line height is 2 times the font size */
    }

    /* Indents the first line of the paragraph */

    p.text-indent {
      text-indent: 40px; /* Indents the first line by 40px */
    }

    /* Controls white space and prevents text wrapping */

    p.white-space {
      white-space: nowrap; /* Text will not wrap to the next line */
    }
  </style>
</head>
<body>
  <p class="letter-spacing">This text has increased letter spacing. Each
letter is farther apart.</p>

  <p class="word-spacing">This text has increased word spacing. Each word is
farther apart.</p>

  <p class="line-height">This text has a line height of 2. There is more
vertical space between lines, making it easier to read.</p>

  <p class="text-indent">This paragraph has a text indent of 40px. The first
line of the paragraph is indented, which helps indicate the start of a new
paragraph.</p>

  <p class="white-space">This text has white-space set to nowrap, so it will
stay on one line and not wrap even if it overflows the container.</p>
</body>
```

```
</html>
```

Output:

This text has increased letter spacing. Each letter is farther apart.

This text has increased word spacing. Each word is farther apart.

This text has a line height of 2. There is more vertical space between lines, making it easier to read.

This paragraph has a text indent of 40px. The first line of the paragraph is indented, which helps indicate the start of a new paragraph.

This text has white-space set to nowrap, so it will stay on one line and not wrap even if it overflows the container.

Text Shadow

The **text-shadow** property is used to add shadow to text. Each shadow is described by some combination of X-offsets & Y-offsets from the text ,blur-radius, color.

```Java
text-shadow: offset-x | offset-y | blur-radius | color
```

Example:

```JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <h1>Aimerz!</h1>
  </body>
```

```
</html>
```

style.css

```
JavaScript
h1 {
  text-shadow: 2px 2px 2px rgb(255, 247, 0);
}
```

Output:

direction

The direction property in CSS is a fundamental styling attribute used to control the text direction within an HTML element. It plays a crucial role in rendering text correctly, especially when dealing with languages and scripts that have different writing directions. This property offers two main values:

- **ltr (Left-to-Right):** This is the default value and is used for languages that are written from left to right, such as English, Spanish, or French.
- **rtl (Right-to-Left):** This value is employed for languages and scripts that are written from right to left, such as Arabic, Hebrew, or Persian.

Syntax:

```
JavaScript
direction : ltr | rtl
```

Example:

index.html

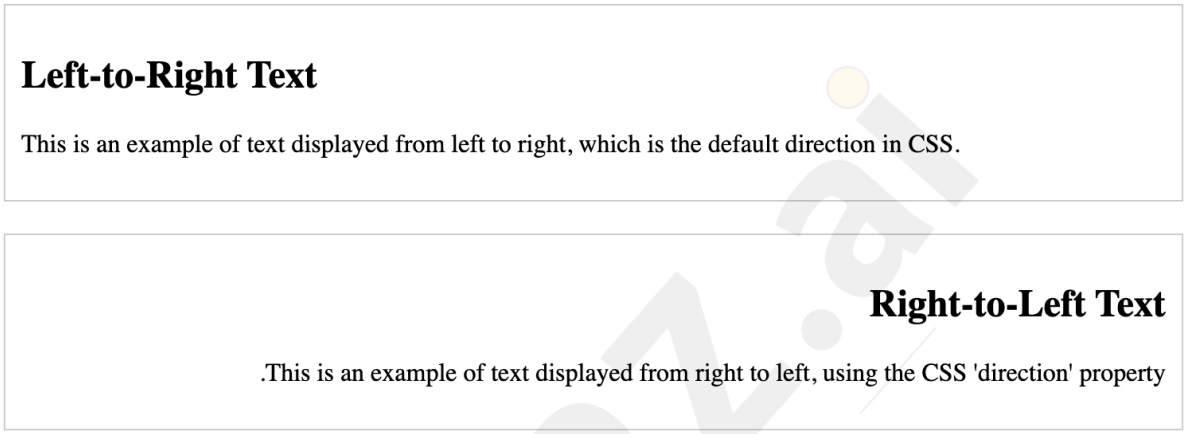
```
JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div class="ltr-text">
      <h2>Left-to-Right Text</h2>
      <p>
        This is an example of text displayed from left to right, which is
the
        default direction in CSS.
      </p>
    </div>
    <div class="rtl-text">
      <h2>Right-to-Left Text</h2>
      <p>
        This is an example of text displayed from right to left, using the
CSS
        'direction' property.
      </p>
    </div>
  </body>
</html>
```

style.css

```
JavaScript
.ltr-text {
  direction: ltr;
  border: 1px solid #ccc;
  padding: 10px;
  margin-bottom: 20px;
}

.rtl-text {
  direction: rtl;
  border: 1px solid #ccc;
  padding: 10px;
```

```
  margin-bottom: 20px;
}
```

Output:



Text Overflow

The **text-overflow** property in CSS is used to control how text that overflows the content area of an element is visually displayed. It is commonly used when you have text within an element that doesn't fit within its designated space, such as in a fixed-width container or a table cell.

text-overflow is especially useful for handling long strings of text or overflowing content in a user-friendly way. It can be clipped, display an ellipsis (...), or display a custom string.

Both of the following properties are required for text-overflow:

```JavaScript
white-space: nowrap;
overflow: hidden;
```

Syntax:

```
Unset
text-overflow: clip | ellipsis | string
```

index.html

```
JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <h2>text-overflow: clip (default):</h2>
    <div class="a">India's First Job Focused Learning Platform</div>
    <h2>text-overflow: ellipsis:</h2>
    <div class="b">India's First Job Focused Learning Platform</div>
    <h2>text-overflow: "---" (user defined string) :</h2>
    <div class="c">India's First Job Focused Learning Platform</div>
  </body>
</html>
```

style.css

```
JavaScript
div.a {
  white-space: nowrap;
  width: 150px;
  overflow: hidden;
  text-overflow: clip;
  border: 1px solid #000000;
}
div.b {
  white-space: nowrap;
  width: 150px;
  overflow: hidden;
  text-overflow: ellipsis;
  border: 1px solid #000000;
}
div.c {
  white-space: nowrap;
  width: 150px;
  overflow: hidden;
```

```
    text-overflow: "----";
    border: 1px solid #000000;
  }
```

Output:

text-overflow: clip (default):

India's First Job Focuse

text-overflow: ellipsis:

India's First Job Foc…

text-overflow: "---" (user defined string) :

India's First Job Focuse

Note: The text-overflow:"string" only works in firefox

Font Family

The font-family in CSS property is used to provide a comma-separated list of font families. It sets the font face for the text content of an element. This property can hold multiple font names as a fallback system, i.e., if one font is unsupported in the browser, then others can be used. The different font-family is used for making attractive web pages.

Note: If the font name is more than one word, it must be in quotation marks, like: **"Times New Roman".**

There are two types of font-family names in CSS, which are defined below

family-name: It is the name of the font-family such as Courier, Arial, Times, etc.

generic-family: It is the name of the generic family that includes five categories, which are **"serif", "sans-serif", "cursive", "fantasy",** and **"monospace"**. It should be placed at last in the list of the font family names.

Syntax:

```
Unset
font-family: family-name|generic-family|initial|inherit;
```

Let's see the values of the font-family property.

family-name/generic-family: It is the list of font-family names and generic family names.
Initial: It is used to set the property to its default value.
Inherit: It is used to inherit the property from its parent element.

Ex:

```
JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <h1>Aimerz!</h1>
  </body>
</html>
```

Style.css

```
JavaScript
h1 {
  font-family: Arial, sans-serif;
}
```

Aimerz!

font style

This property specifies the style of the font, such as italic or normal.

index.html

```
JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <h1>Aimerz!</h1>
    <p>Welcome to Aimerz!</p>
  </body>
</html>
```

Style.css

```
JavaScript
h1 {
  font-style: italic;
```

```
}

p {
  font-style: normal;
}
```

Output:

Font Variant

font-variant is a CSS property that allows you to control the appearance of text characters within an element. It mainly deals with Text Case and text size. It can modify the case of the text characters, particularly for lowercase letters. The primary value here is small-caps, which converts lowercase letters to small capital letters, making them stand out while leaving uppercase letters unchanged.

```
Unset
font-variant: normal|small-caps;
```

index.html

```
JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="style.css" />
```

```
  </head>
  <body>
    <h1>Aimerz!</h1>
    <p>Welcome to Aimerz!</p>
  </body>
</html>
```

Style.css

```
JavaScript
h1 {
  font-variant: normal;
}

p {
  font-variant: small-caps;
}
```

Output:

font size & font weight

The font-size property in CSS is used to specify the height and size of the font. It affects the size of the text of an element. Its default value is medium and can be applied to every element.

The value of the property includes "xx-small", "x-small", "small", "medium", "large", "x-large", and "xx-large".

The font-size can be relative or absolute.

```
JavaScript
font-size:
medium| xx-small|x-small| small|large|x-large|xx-large| smaller|larger|
length(px, em,rem)
```

We will see all the absolute and relative in the upcoming section called **“CSS unit".**

```
JavaScript
h1 {
  font-size: 30px;
}
```

Aimerz!

When we set the size of text with pixels, then it provides us full control over the size of the text.

The **font-weight** property is used for setting the thickness and boldness of the font. It is used to define the weight of the text. The available weight depends on the **font-family**, which is used by the browser.

The value of the property includes **normal, lighter, bolder, bold, number, inherit, initial, unset;**

```
JavaScript
font-weight:
medium| xx-small|x-small| small|large|x-large|xx-large| smaller|larger|
length(px, em,rem)
```

```
JavaScript
h1 {
  font-weight: lighter;
}
```

Output:

Aimerz!

font shorthand

We have a shorthand property font which includes **font-style, font-variant, font-weight, font-size/line-height, and font-family.**

Let's look at its syntax:

```
Unset
font: font-style font-variant font-weight font-size/line-height font-family;
```

Ex:

```JavaScript
body {
  font: italic normal bold 16px/1.5 "Helvetica Neue", Arial, sans-serif;
}
```

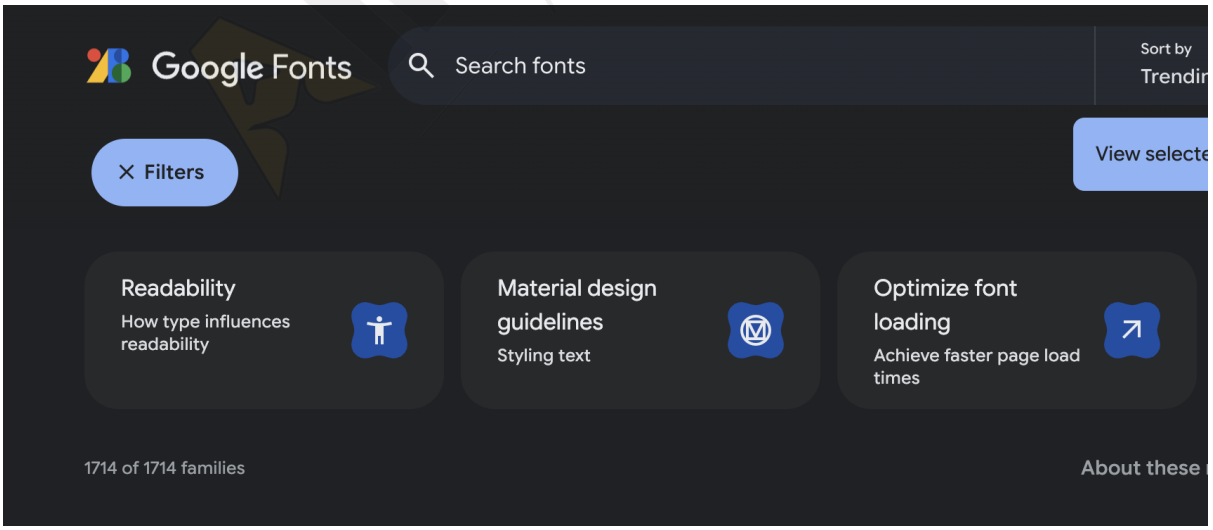
Welcome to Aimerz!

What is google font and how to use it?

Google Fonts is a free service offered by Google that provides a collection of web fonts that can be easily embedded into a website. These fonts are optimized for web use and can be easily incorporated into CSS code using a few lines of code.

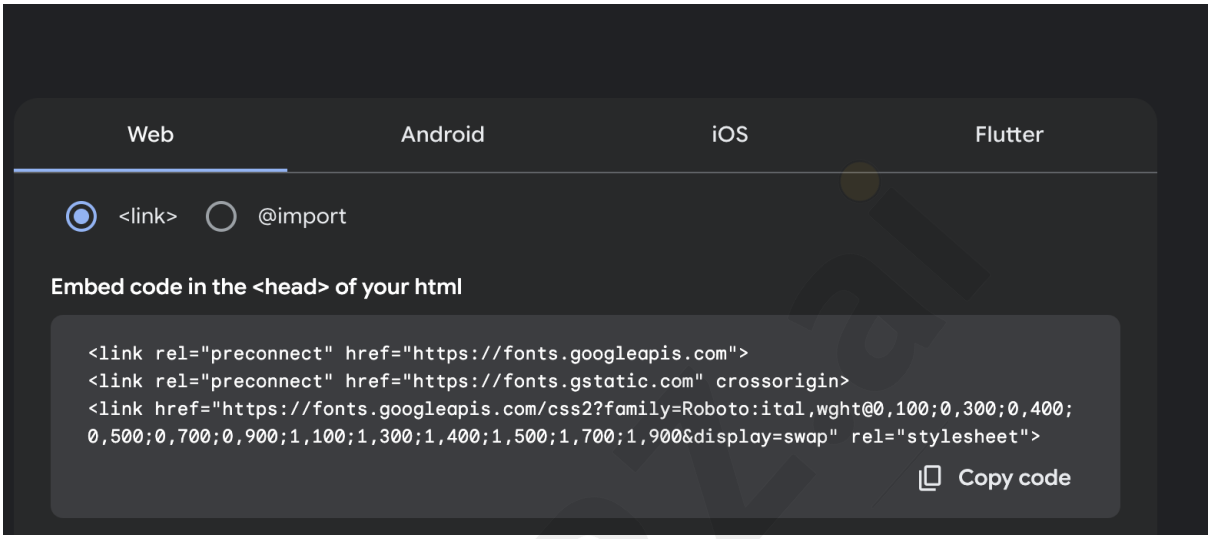
Here's how to use Google Fonts in your CSS code:

Go to the Google Fonts website Link and browse the available fonts. Once you've found a font you want to use, select it.



Once you've chosen a font, you will see a collection of the same fonts with different styles. Add the font to your collection, and choose how you want to embed it by adding a link in HTML or importing the fonts in a CSS file.

First way to apply Google Fonts:
Copy the link and add it to your HTML file in **<head>**, and add a CSS rule in style.css.



index.html

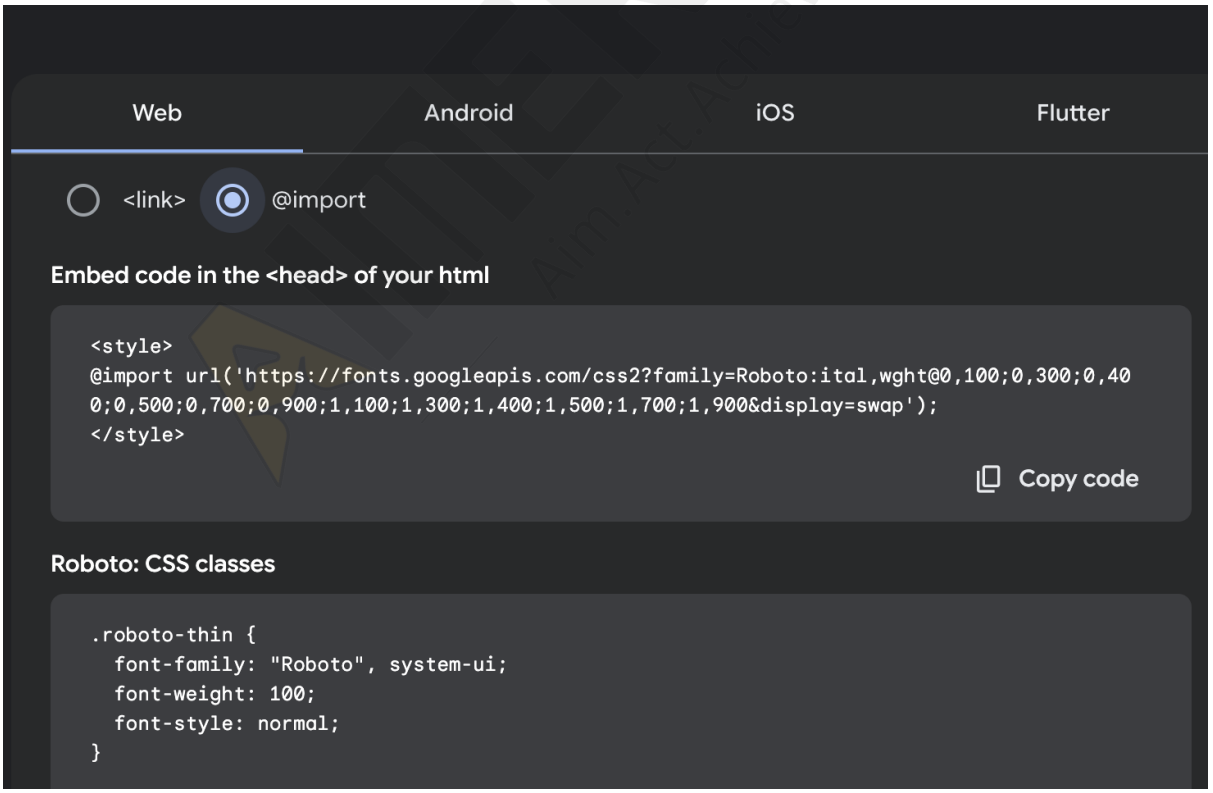
```
JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="style.css" />
    <link rel="preconnect" href="https://fonts.googleapis.com" />
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin />
    <link

href="https://fonts.googleapis.com/css2?family=Roboto:ital,wght@0,100;0,300;
0,400;0,500;0,700;0,900;1,100;1,300;1,400;1,500;1,700;1,900&display=swap"
      rel="stylesheet"
    />
  </head>
  <body>
    <h1>Welcome to Aimerz!</h1>
  </body>
</html>
```

Style.css

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

Welcome to Aimerz!

Second Way:
Applying Google Fonts by importing the fonts



index.html

```
JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="style.css" />
    <link rel="preconnect" href="https://fonts.googleapis.com" />
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin />
    <link

href="https://fonts.googleapis.com/css2?family=Roboto:ital,wght@0,100;0,300;
0,400;0,500;0,700;0,900;1,100;1,300;1,400;1,500;1,700;1,900&display=swap"
      rel="stylesheet"
    />
  </head>
  <body>
    <h1>Welcome to Aimerz!</h1>
  </body>
</html>
```

Style.css

```
JavaScript
//Importing Google fonts
@import
url('https://fonts.googleapis.com/css2?family=Roboto:ital,wght@0,100;0,300;0
,400;0,500;0,700;0,900;1,100;1,300;1,400;1,500;1,700;1,900&display=swap');

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

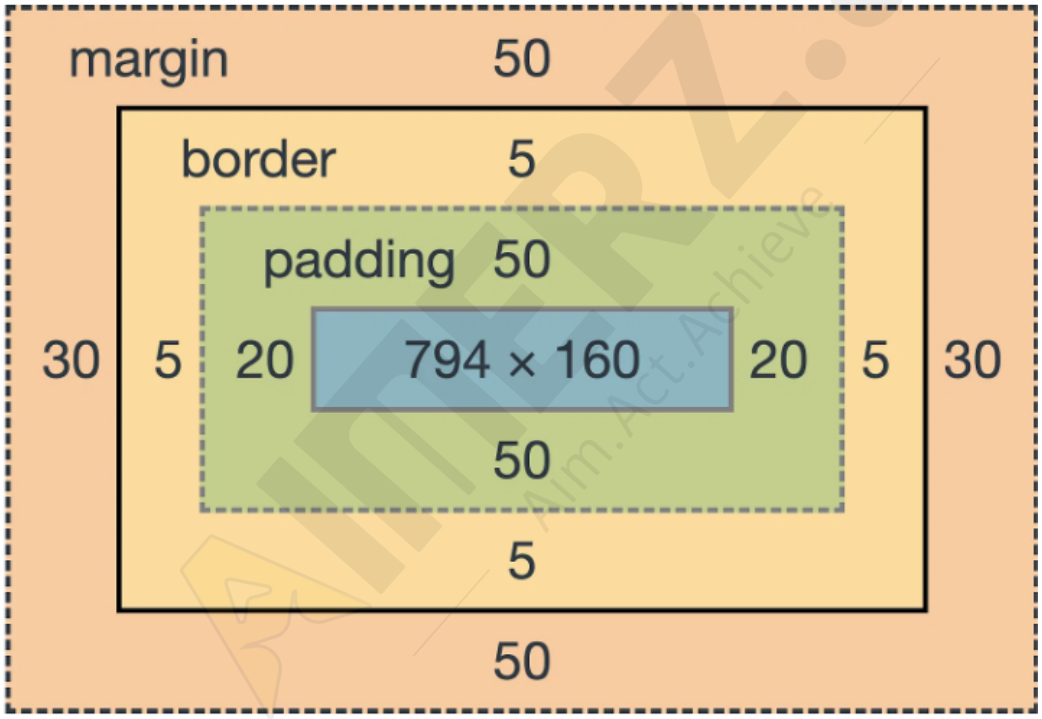
Output:

Welcome to Aimerz!

Introduction to box model

The CSS box model is a container that contains multiple properties, including **borders**, **margins**, **padding**, and the **content** itself. It is used to create the design and layout of web pages. According to the CSS box model, the web browser supplies each element as a square prism.

The following diagram illustrates the box model.



Properties of the box-model

1. **Content:** This is the area where the actual content, such as text, images, or videos, is displayed. The width and height properties control the size of this area.

2. **Padding:** Padding is the space between the content and the border. It adds extra space inside the box and can be adjusted for each side (top, right, bottom, and left).
3. **Border:** The border surrounds the padding and content. You can style it with different widths, styles, and colors.
4. **Margin:** Margin is the outermost part of the box and defines the space between the element's border and neighboring elements. It's transparent and allows for spacing between elements.

Padding

The padding property controls the space between the content and the border. Padding can be applied to all sides or to individual sides of an element.

Padding Properties:

- `padding-top`
- `padding-right`
- `padding-bottom`
- `padding-left`

You can set padding in different ways:

1. **Individual sides:**

```JavaScript
.box {
  padding-top: 10px;
  padding-right: 20px;
  padding-bottom: 30px;
  padding-left: 40px;
}
```

2. **Shorthand with four values:** This applies padding in clockwise order: top, right, bottom, and left.

```JavaScript
.box {
```

```
  padding: 10px 20px 30px 40px;
}
```

3. **Shorthand with three values:** The first value is for top padding, the second for left and right, and the third for bottom padding.

```JavaScript
.box {
  padding: 10px 20px 30px;
}
```

4. **Shorthand with two values:** The first value applies to top and bottom, while the second applies to left and right.

```JavaScript
.box {
  padding: 10px 20px;
}
```

5. **One value for all sides:** All sides have equal padding.

```JavaScript
.box {
  padding: 10px;
}
```


Let's take an example to understand better:

`index.html`

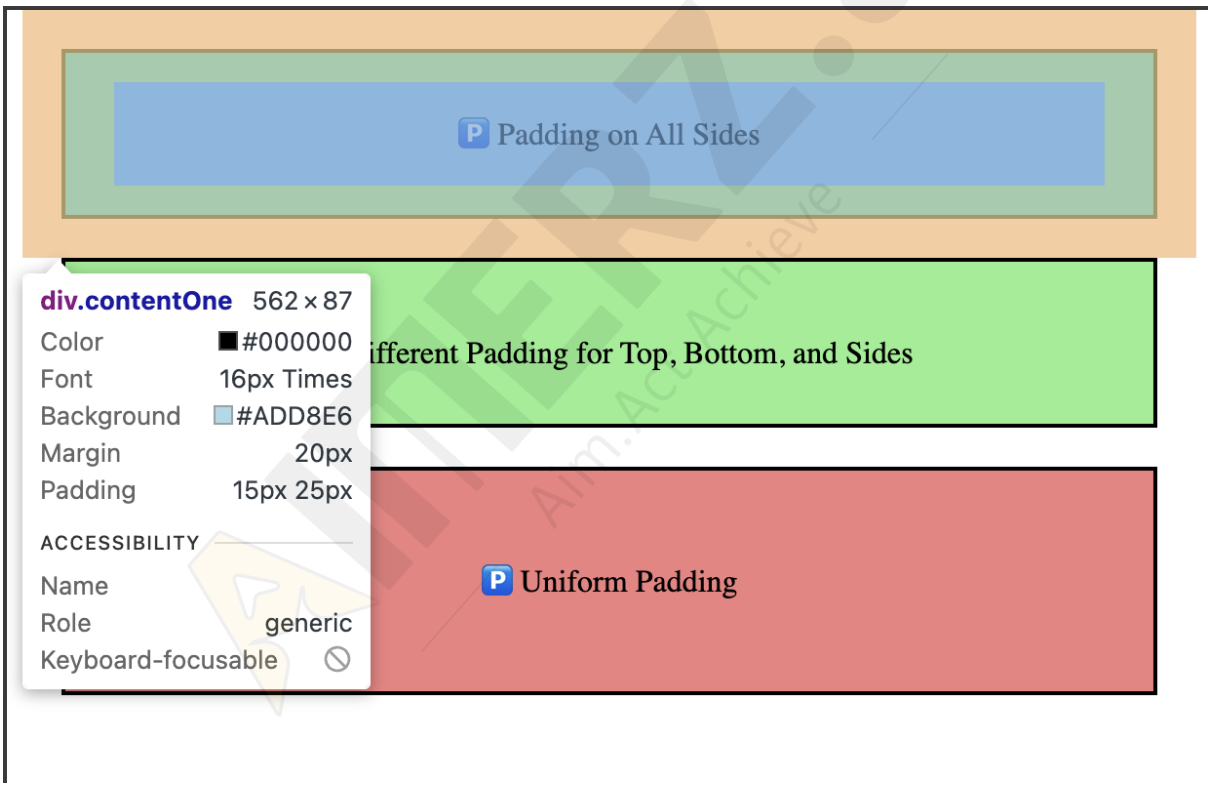
```
JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="/style.css" />
    <title>Padding Example</title>
  </head>
  <body>
    <div class="contentOne">
      <p> Padding on All Sides</p>
    </div>
    <div class="contentTwo">
      <p> Different Padding for Top, Bottom, and Sides</p>
    </div>
    <div class="contentThree">
      <p>Uniform Padding</p>
    </div>
  </body>
</html>
```

style.css

```
JavaScript
/* General styling for the divs */
div {
  margin: 20px;
  text-align: center;
  border: 2px solid black;
}

/* Box 1: Padding on all sides */

.contentOne {
  background-color: lightblue;
  padding-top: 15px;
  padding-right: 25px;
  padding-bottom: 15px;
  padding-left: 25px;
}

/* Box 2: Different padding for top/bottom and sides */
```

```
.contentTwo {
  background-color: lightgreen;
  padding: 20px 50px 10px;
}

/* Box 3: Uniform padding on all sides */

.contentThree {
  background-color: lightcoral;
  padding: 30px;
}
```

Output:



Border

The border property adds a visible line around an element. You can customize the width, style, and color of the border.

Properties of Border

1. **Border-style :** solid, dotted, dashed, double, groove, ridge, inset, outset

Note: We can also **individually change the border style** of the **left**, **right**, **bottom**, and **top** borders of an element by using **border-left-style**, **border-right-style**, **border-top-style**, and **border-bottom style**.

Ex:

index.html

```
JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="style.css" />
    <title>Document</title>
  </head>
  <body>

    <div class="none">none</div>

    <div class="solid">solid</div>

    <div class="dotted">dotted</div>

    <div class="dashed">dashed</div>

    <div class="double">double</div>

    <div class="groove">groove</div>

    <div class="ridge">ridge</div>

    <div class="inset">inset</div>
```

```
    <div class="outset">outset</div>

    <div class="mixup">dashed, dotted, solid,and inset</div>

  </body>
</html>
```

style.css

```JavaScript
.none {
  border-style: none;
}
.solid {
  border-style: solid;
}
.dotted {
  border-style: dotted;
}
.double {
  border-style: double;
}
.dashed {
  border-style: dashed;
}
.groove {
  border-style: groove;
}
.ridge {
  border-style: ridge;
}
.inset {
  border-style: inset;
}
.outset {
  border-style: outset;
}
.mixup{
    border-top-style: dashed;
    border-right-style: dotted;
    border-bottom-style: solid;
```

```
    border-left-style: inset
}
```

Output:

none
solid
dotted
dashed
double
groove
ridge
inset
outset
dashed, dotted, solid,and inset

2. Border-width:

The border-width **property allows you to set** the **width of an element's border**. The value of this property could be either a length in **px**, **pt**, or **cm**, or it should be set to **thin**, **medium**, or **thick**.

We can also **individually set the border width** of the **left**, **right**, **bottom**, and **top sides** of an element by using **border-left-width**, **border-right-width**, **border-top-width**, and **border-bottom-width**.

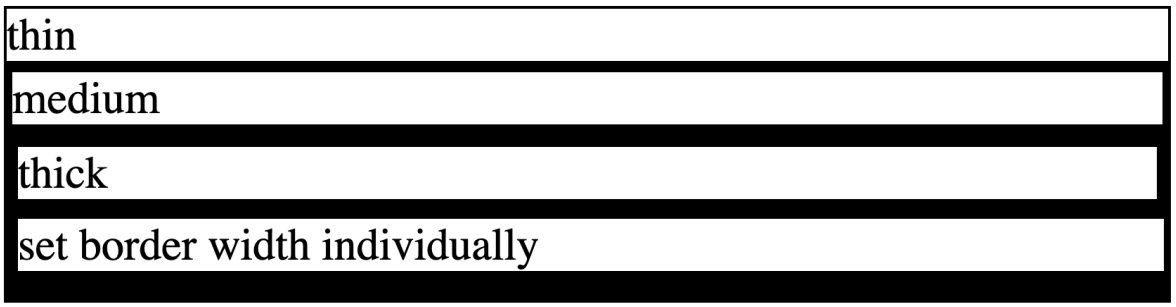
index.html

```
JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="style.css" />
    <title>Document</title>
  </head>
  <body>
    <div class="thin">thin</div>
    <div class="medium">medium</div>
    <div class="thick">thick</div>
    <div class="unitValue">set border width individually</div>
  </body>
</html>
```

style.css

```
JavaScript
.thin {
  border-style: solid;
  border-width: thin;
}

.medium {
  border-style: solid;
  border-width: medium;
}

.thick {
  border-style: solid;
  border-width: thick;
}

.unitValue {
  border-style: solid;
  border-top-width: 2px;
  border-right-width: 2pt;
  border-bottom-width: 0.3cm;
  border-left-width: 5px;
}
```

Output:



3. Border-color:

The border color property allows us to change the border color, we can also change the border-bottom-color, border-top color, border-right-color, and border-left color of an element individually.

index.html

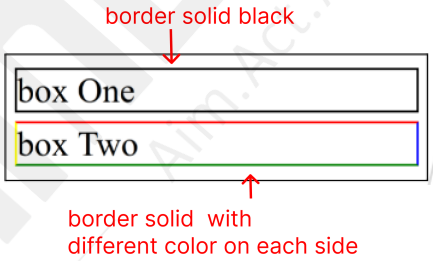
```
Unset
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width,
initial-scale=1.0" />
    <link rel="stylesheet" href="style.css" />
    <title>border color</title>
  </head>
  <body>
    <div class="boxOne">box One</div>
```

```
    <div class="boxTwo">box Two</div>
  </body>
</html>
```

Style.css

```
Unset
.boxOne {
  border-color: black;
  border-style: solid;
}
.boxTwo {
  border-style: solid;
  border-color: red blue green yellow;
}
```

Output:



4. Border-radius

Using the border-radius property, we can set the rounded borders and provide the rounded corners around an element, tag, or div. The border-radius defines the radius of the corners of an element.

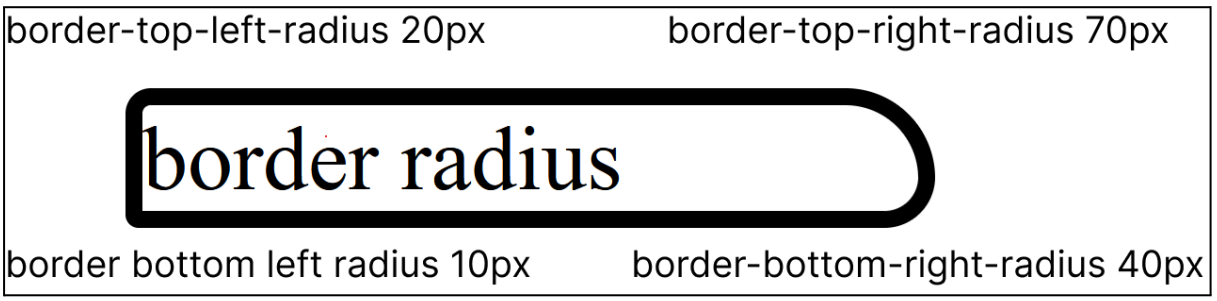
Index.html

```
Unset
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width,
initial-scale=1.0" />
    <link rel="stylesheet" href="style.css" />
    <title>Document</title>
  </head>
  <body>
    <div class="box">border radius</div>
  </body
</html>
```

Style.css

```
Unset
.box {
  border-style: solid;
  border-top-left-radius: 20px;
  border-top-right-radius: 70px;
  border-bottom-right-radius: 40px;
  border-bottom-left-radius: 10px;
}
```

output



Border radius - shorthand property

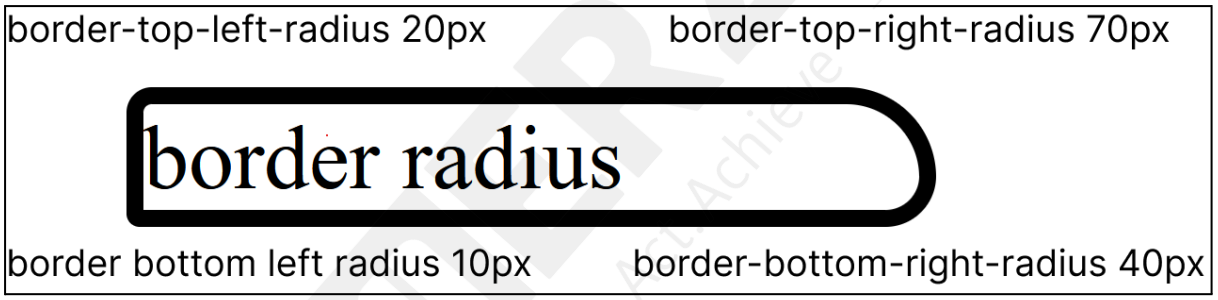
We can specify all the border-radius properties in one property.

I. Border radius with four values

style.css

```
Unset
.box {
  border-style: solid;
  border-radius: 20px 70px 40px  10px;
}
```

output:

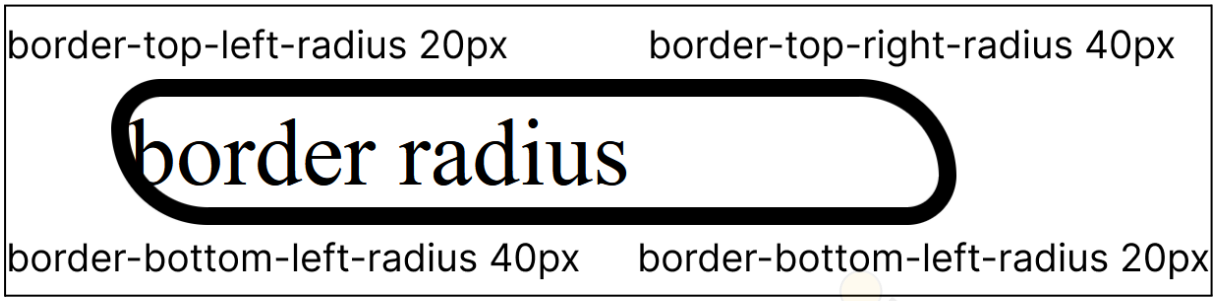


II. Border radius with two values

Style.css:

```
Unset
.box {
  border-style: solid;
  border-radius: 20px 40px ;
}
```

Output:

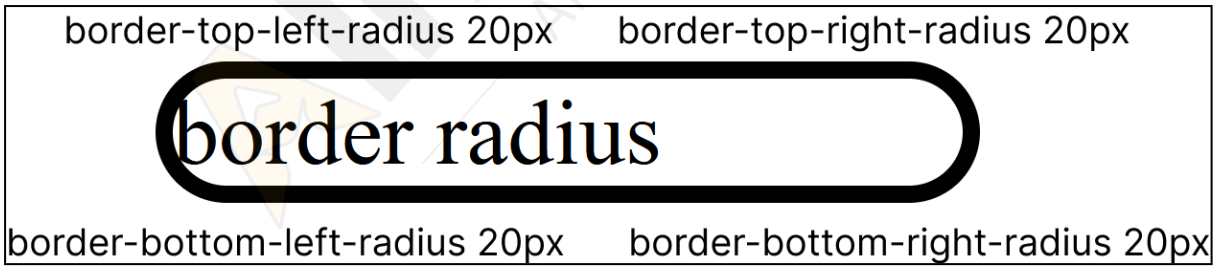


III. Border radius with one value

Style.css

```
Unset
.box {
  border-style: solid;
  border-radius: 20px;
}
```

Output:



Border - shorthand property

To shorten the code, it is also possible to specify all the individual border properties in one property **except border-radius**.

The border property is a shorthand property for the following individual border properties:

- border-width
- border-style (required)
- border-color

```
Unset
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width,
initial-scale=1.0" />
    <link rel="stylesheet" href="style.css" />
    <title>Document</title>
</head>

<body>
    <div class="box">border shorthand</div>
</body>

</html>
```

style.css:

```
Unset
.box {
  border: 5px solid purple;
}
```

output:

border shorthand

When the border's **non-required** properties are skipped, the **browser will use default values** for those properties, and the **output will be determined by these defaults**.

border-width: If you skip specifying the border width, it will typically **default to "medium,"** which is a browser-specific value, **equal to 1px**.

border-color: If you skip specifying the border color, it will **default to the current text color**, which is usually black.

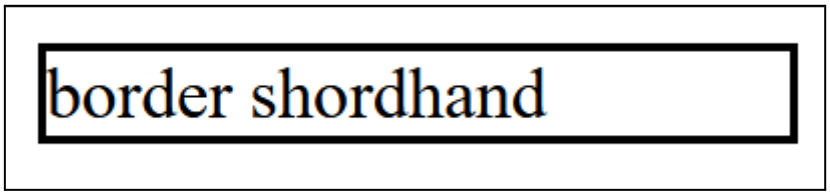
Let's see some examples where we skip specifying border shorthand non-required properties:

For the example, **let's use the same HTML code** that we had used in the above border shorthand property example.

style.css

```
Unset
.box {
  border: solid;
}
```

output:



As you can see in the above example, we have specified the border style as **'solid,'** resulting in a solid line style for the border. The default border width, which is **'medium,'** and the default border color, which is the current text color (usually black), are applied.

Now let's update the above example by adding text color.

style.css

```
Unset
.box {
  border: solid;
  color: purple;
}
```

output

In the above case, we set the text color to purple but didn't specify the **'border-color'** in the 'border' shorthand property. As a result, it takes the current text color **(purple)** as the **'border-color'.**

Margin

The margin property defines the space around an HTML element. It is possible to use negative values to overlap content.

We have the following properties to set an element margin:

Margin - Individual Sides

We can specify the margin for each side of an element.

index.html

```
Unset
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width,
initial-scale=1.0" />
    <link rel="stylesheet" href="/style.css" />
    <title>Document</title>
  </head>
  <body>
     <div class="elementOne">Element 1</div>
     <div class="elementTwo">Element 2</div>
     <div class="elementThree">Element 3</div>
  </body>
</html>
```

Note: *We will use the same HTML code for all the margin examples.*

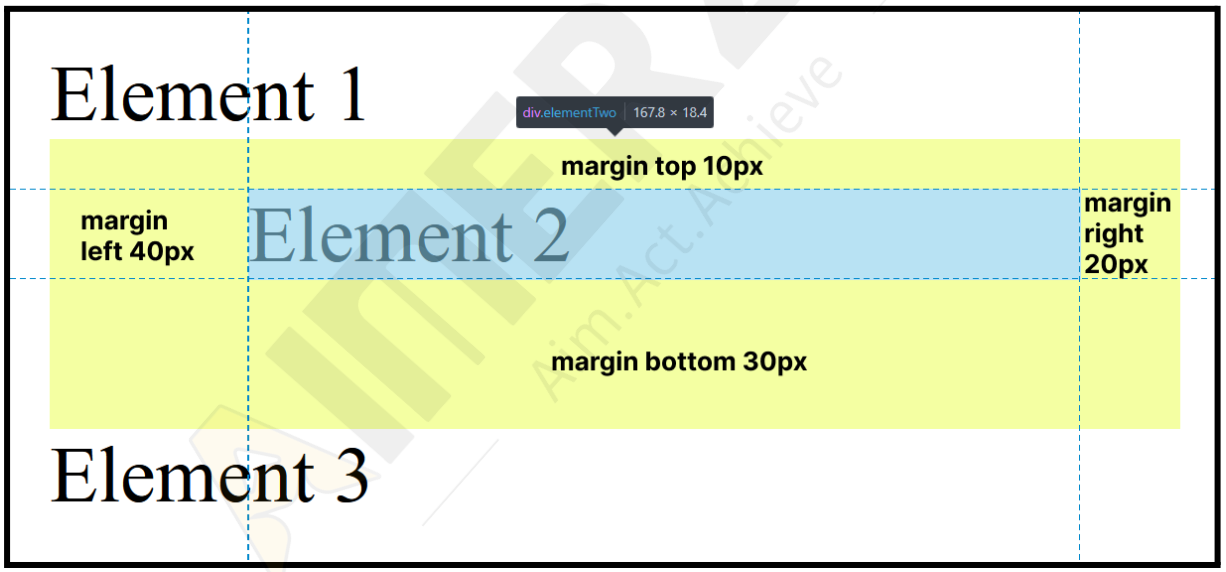
The **margin-top** specifies the top margin of an element.

The **margin-right** specifies the right margin of an element.
The **margin-bottom** specifies the bottom margin of an element.
The **margin-left** specifies the left margin of an element.

style.css

```
Unset
.elementTwo{
margin-top: 10px;
margin-right: 20px;
margin-bottom: 30px;
margin-left: 40px;
 }
```

output



Margin - shorthand property

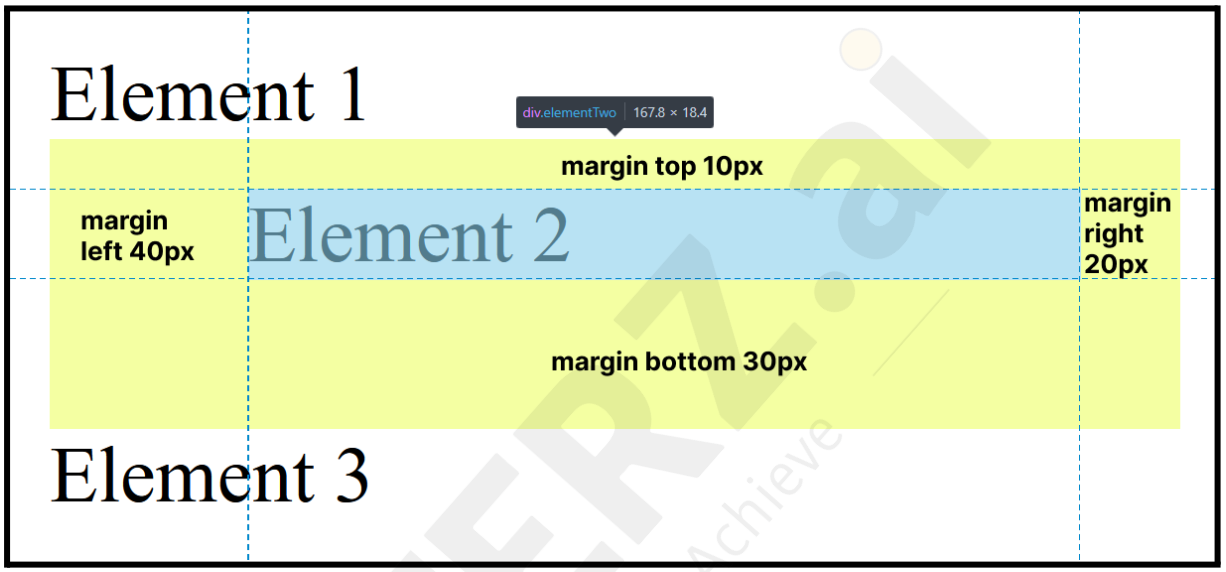
Using the margin, we can specify all the margin properties in one property.

1. Margin property with four values.

style.css:

```
Unset
.elementTwo {
  margin: 10px 20px 30px 40px;
}
```

output



The **first value** is for the **top margin**, the **second value** is for the **right margin**, the **third value** is for the **bottom margin,** and the **fourth value** is for the **left margin**. Observe the clockwise pattern.

2. Margin property with three values.

style.css:

```
Unset
.elementTwo {
margin: 20px 40px 30px
}
```

output:

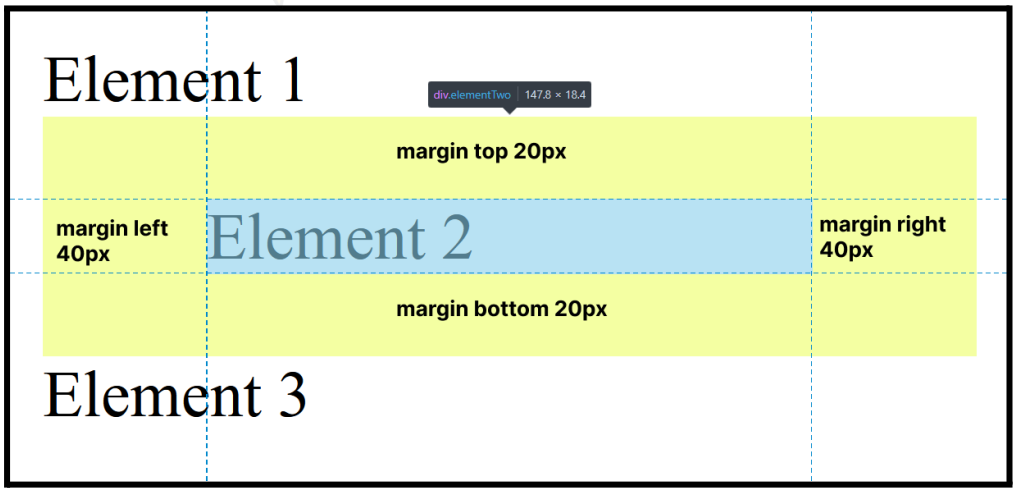
The **first value** is for the **top margin**, the **second value** is for the **right margin** and the **left margin**, and the **third value** is for the **bottom margin.**

3. Margin property with two values

style.css:

```
Unset
.elementTwo {
margin: 20px 40px
}
```

output:



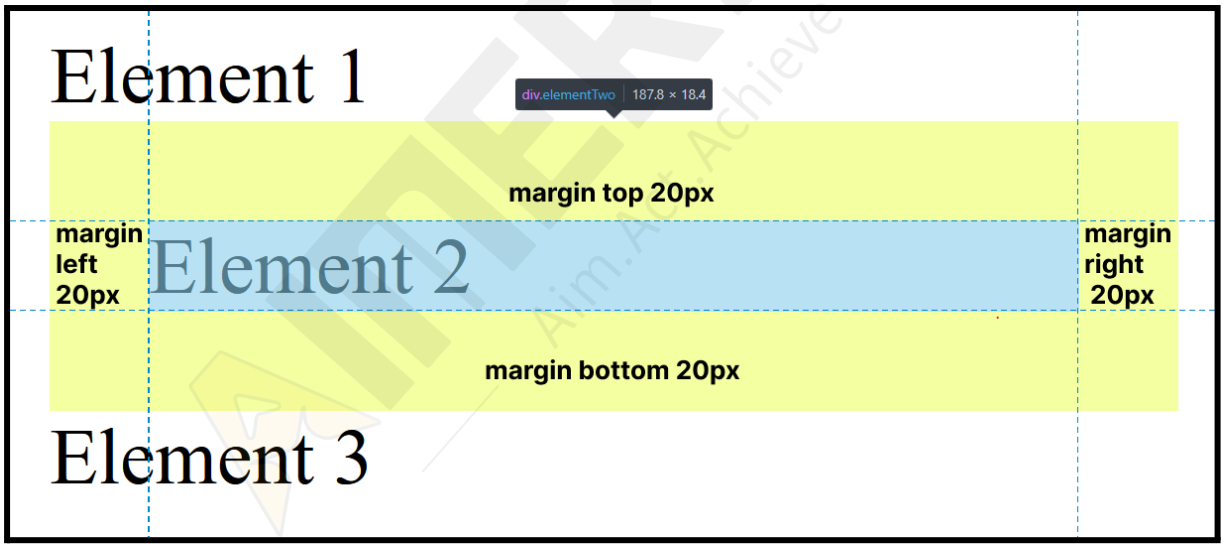
The first value is for the **top margin and bottom margin** and the second value is for the **right margin and left Margin**.

4. Margin property with only one value

Style.css

```
Unset
.elementOne{
margin: 10px;
}
```

output:



Margin Collapse

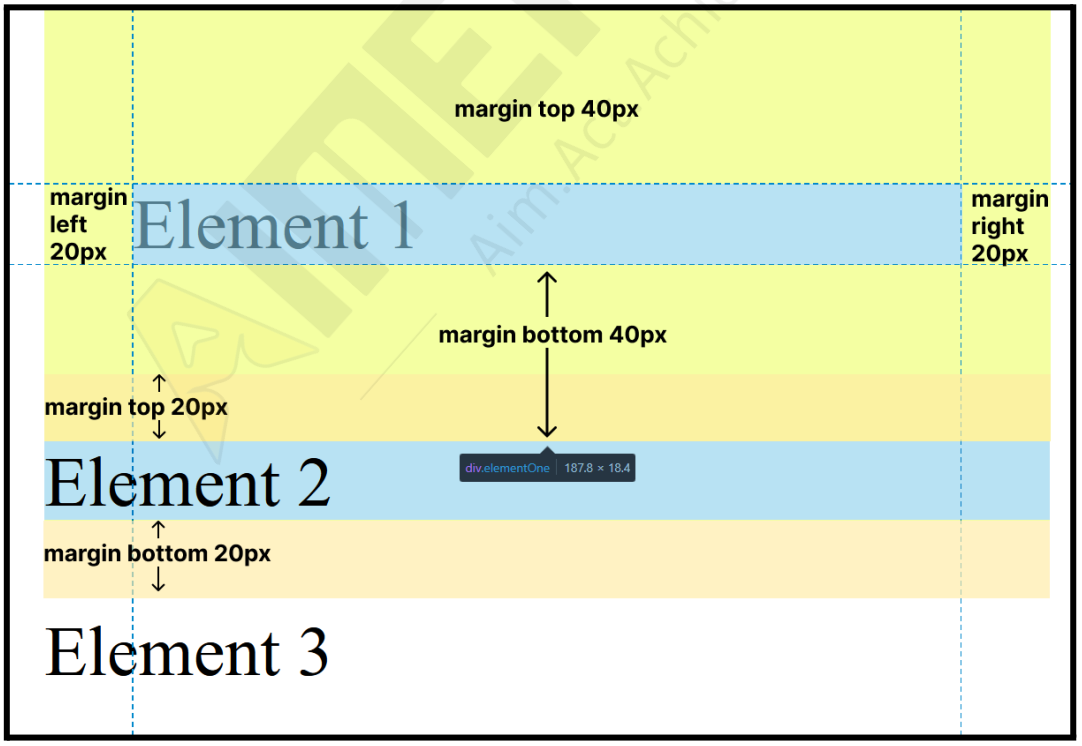
The top and bottom margins of elements are sometimes collapsed into a single margin that is equal to the largest of the two margins.

This does not happen on the left and right margins, only top and bottom margins which means only vertical margins.

style.css:

```
Unset
.elementOne {
margin: 40px 20px;
}

.elementTwo {
margin: 20px 0px;
}
```

Output:

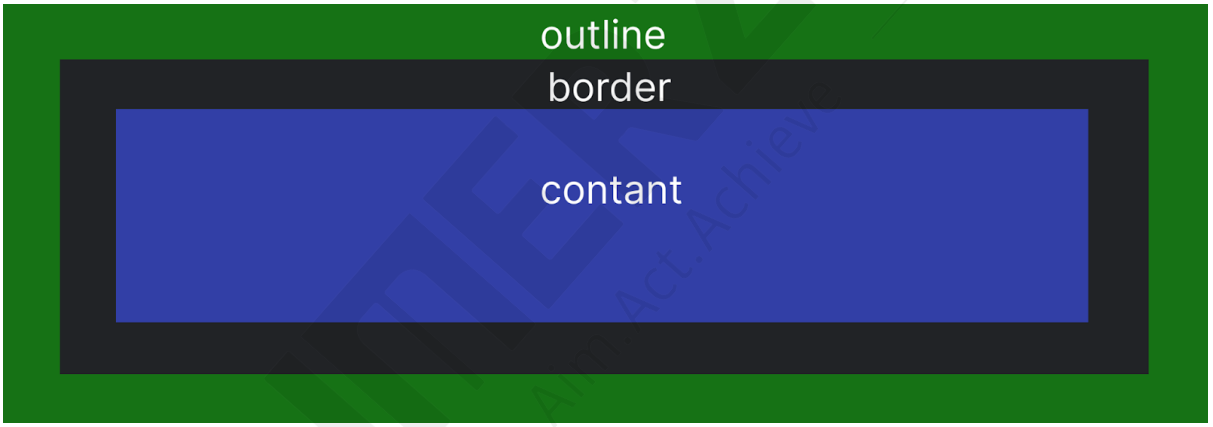


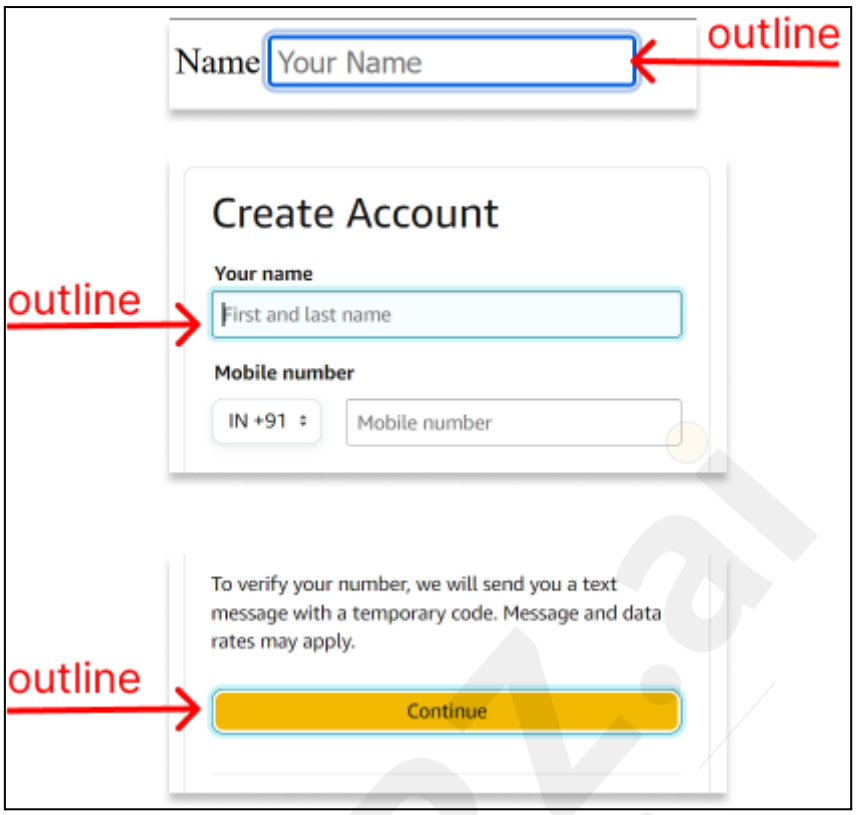
Outline

Outline Introduction

The CSS outline properties allow you to define an outline area around an element's box.

An outline is a line that is drawn just outside the border edge of the elements. Outlines are generally used to indicate the focus or active states of elements such as buttons, links, form fields, etc.





You can set the following outline properties using CSS.

- outline-style
- outline-color
- outline-width
- outline-offset
- outline

outline-style

You can set the style of an outline using the outline-style property. The outline-style property accepts a range of values that determine the style of the outline. Here are some common outline styles you can use in CSS:

- **“solid”** - creates a solid line around the element.
- **“dotted”** - creates a dotted line around the element.
- **“dashed”**- creates a dashed line around the element.
- **“double”** - creates a double line around the element.
- **“groove”**- creates a 3D groove effect around the element.
- **“ridge”** - creates a 3D ridge effect around the element.
- **“inset”** - creates a 3D inset effect around the element.
- **“outset”**- creates a 3D outset effect around the element.
- **“none”** - removes the outline from the element.

Here's an example of how to set the outline-style property:

Index.html

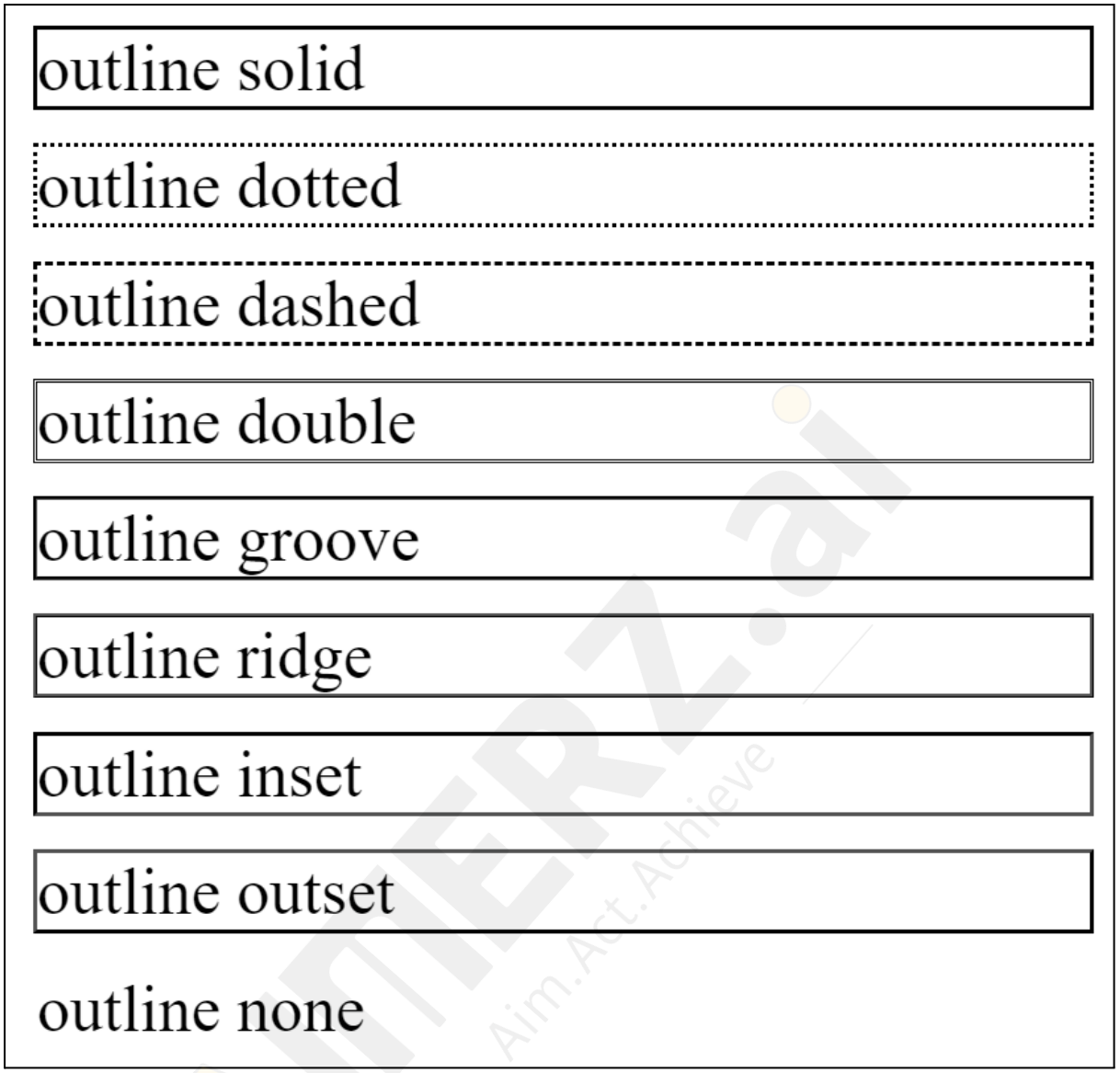
```
Unset
    <div class="solid">outline solid</div>
    <div class="dotted">outline dotted</div>
    <div class="dashed">outline dashed</div>
    <div class="double">outline double</div>
    <div class="groove">outline groove</div>
    <div class="ridge">outline ridge</div>
    <div class="inset">outline inset</div>
    <div class="outset">outline outset</div>
    <div class="none">outline none</div>
```

Style.css

```
Unset
.solid {
     outline-style: solid;
     }
```

```
.dotted {
     outline-style: dotted;
     }
.dashed {
     outline-style: dashed;
     }
.double{
     outline-style: double;
     }
.groove {
     outline-style: groove;
     }
.ridge {
     outline-style: ridge;
     }
.inset {
     outline-style: inset;
     }
.outset {outline-style: outset;}
.none {  outline-style: none; }
```

Browser output:



Outline color

You can set the color of an outline using the outline-color property.

Here's an example of how to set the outline-color property:

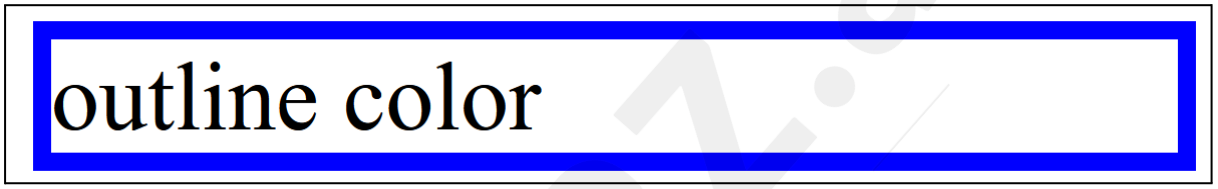
Index.html

```
Unset
<div class= "outlineColor> outline color</div>
```

style.css

```
Unset
.outlineColor {
  outline-style: solid;
  outline-color: blue;
}
```

Browser output



Outline-width

The outline-width property specifies the width of the outline and can have one of the following values:

- thin (typically 1px)
- medium (typically 3px)
- thick (typically 5px)
- A specific size (in px, pt, cm, em, etc.)

Here's an example of how to set the outline-width property:

index.html:

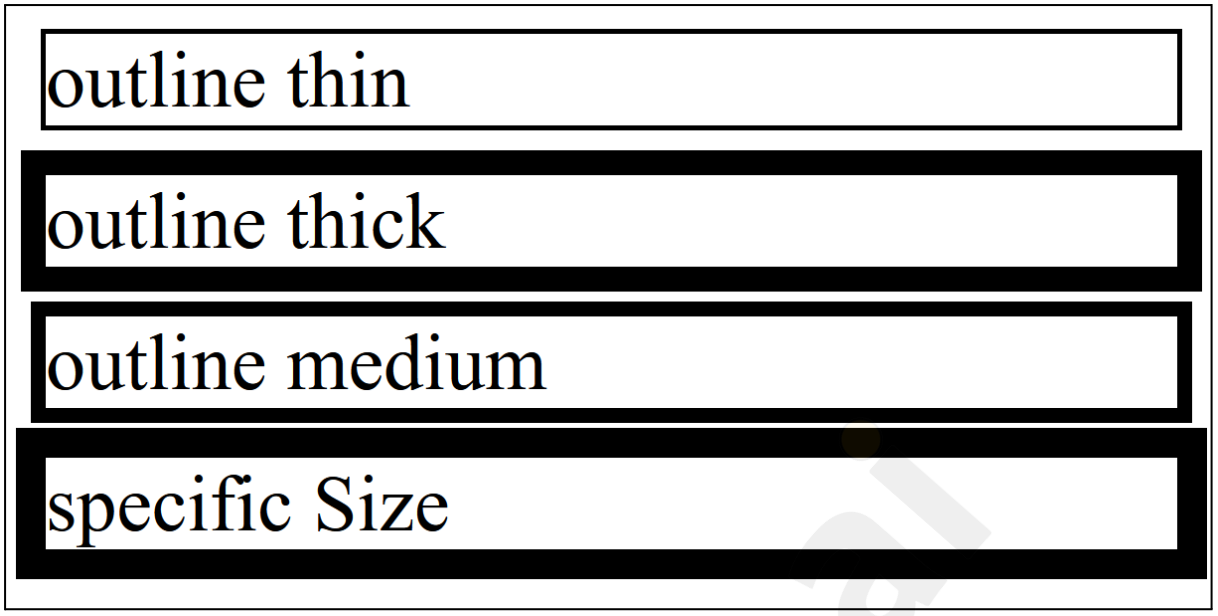
```
Unset
  <div class="thin">outline thin</div>
```

```
  <div class="thick">outline thick</div>
  <div class="medium">outline medium</div>
  <div class="specificSize">unit value</div>
```

style.css:

```
Unset
.thin {
  outline-style: solid;
  outline-width: thin;
}
.medium {
  outline-style: solid;
  outline-width: medium;
}
.thick {
  outline-style: solid;
  outline-width: thick;
}
.specificSize {
  outline-style: solid;
  outline-width: 6px;
}
```

Browser output:



Outline offset

The outline-offset property adds space between an outline and the edge/border of an element. The space between an element and its outline is transparent.

Here's an example of how to set the outline-offset property:

Index.html

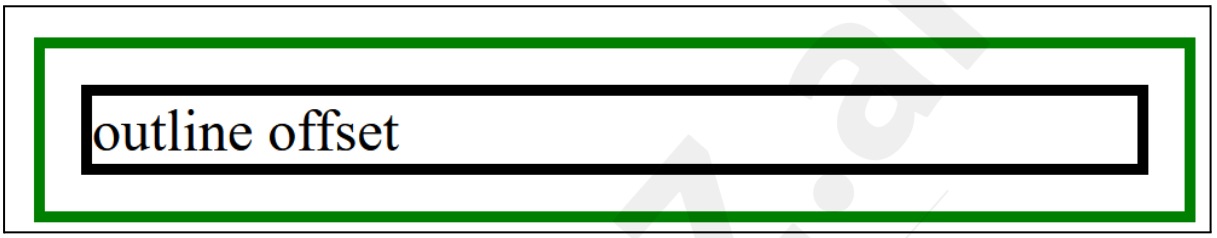
```
Unset
<div class="offset">outline offset</div>
```

style.css

```
Unset
.offset {
```

```
  margin: 3rem;
  outline-offset: 10px;
  outline-style: solid;
  outline-color: green;
  border-style: solid;
}
```

Browser output:



Outline shorthand property

The outline property is a shorthand property for setting the following individual outline properties:

- outline-width
- outline-style (required)
- outline-color

```
Unset
outline:[outline-width] [outline-style] [outline-color];
```

Where outline-width, outline-style, and outline-color are optional values that can be specified in any order. Here are some examples:

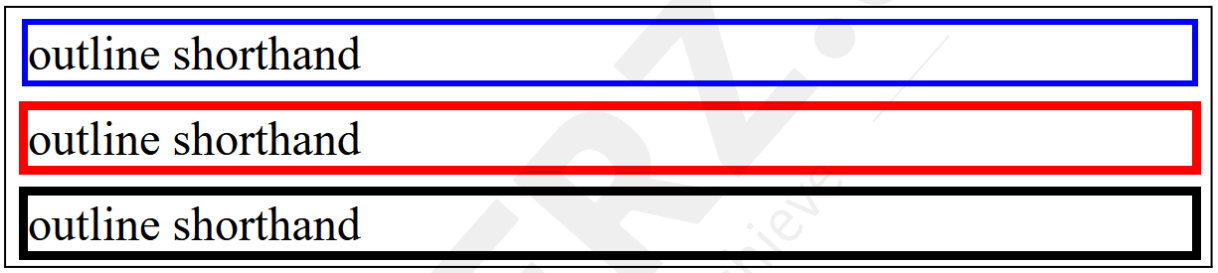
index.html

```
Unset
    <div class="three">outline shorthand</div>
    <div class="two">outline shorthand</div>
    <div class="one">outline shorthand</div>
```

style.css

```
Unset
.three{ outline: 2px solid blue; }
.two{ outline: solid red; }
.one{ outline: solid }
```

Browser output:



We can also use the outline shorthand property to set individual outline properties, by omitting one or more of the values.

Example:

index.html

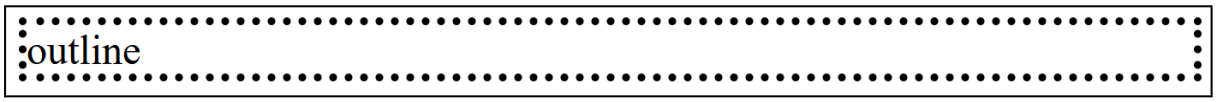
```
Unset
<div class="outline">outline</div>
```

style.css

```
Unset
.outline{
```

```
  outline: dotted;
}
```

Browser output:



In the above example, we use only one '**outline-style**' instead of all three properties. Let's see what happens when we skip the non-required outline properties.

When **outline** non-required properties are skipped in CSS, the browser will **use default values for those properties**, and the **output will be determined by these defaults**.

The default values of non-required outline properties in CSS are typically **set to "initial" values or default values specified by the browser's default styles.**

output-width: If you skip specifying the output width, it will typically default to "medium," which is a browser-specific value, often equivalent to 1px.

outline-color: If you skip specifying the outline color, it will default to the current text color, which is usually black.

Let's see some examples where we skip specifying border shorthand non-required properties:

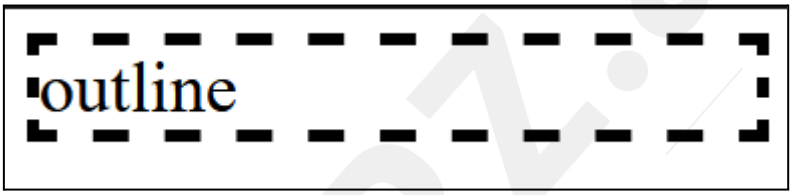
index.html

```
Unset
<div class="outline">outline</div>
```

style.css:

```
Unset
.outline{
    outline: dashed
}
```

Browser output:



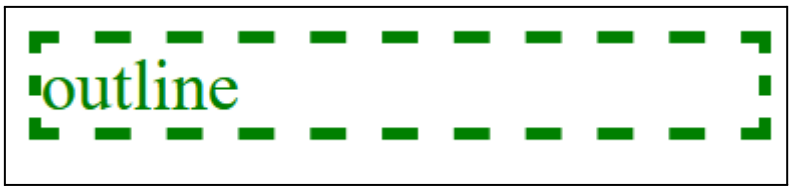
As you can see in the above example, we specified the outline style as **'dashed,'** resulting in a dashed line style for the outline. The default outline width, which is **'medium,'** and the default border color, which is the current text color (usually black), are applied.

Now let's update the above example by adding text color.

style.css

```
Unset
 .outline {
   outline: dashed;
   color : green;
 }
```

Browser output



In the above case, we set the text color but didn't specify the 'outline-color' in the 'outline' shorthand property. As a result, it takes the current text color as the **'outline-color'**.

***Note**: It's important to note that the outline property is different from the border property. While the border affects the element's layout and size, an outline does not.*

Width

CSS **width** is a property that helps us control the size of an element on a webpage. Think of it as how wide or narrow something should be. It does not include any padding, borders, or margins.

Imagine you have a box on your webpage, like a picture or a text block. The CSS width property allows you to decide how much horizontal space this box should take up on the screen.

Example

Index.html

```
Unset
<body>
```

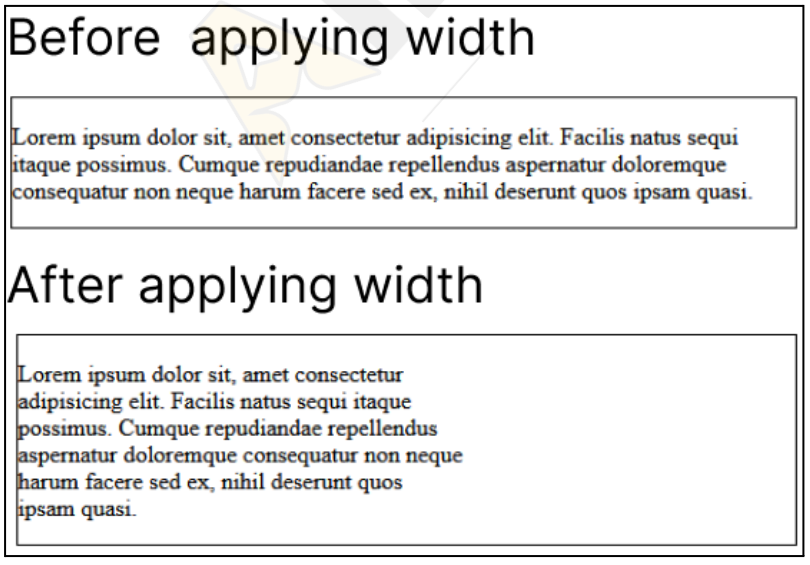
```
    <div class="container">
        <p class="text">Lorem ipsum dolor sit, amet
consectetur adipisicing elit. Facilis natus sequi itaque
            possimus. Cumque
            repudiandae repellendus aspernatur doloremque
consequatur non neque harum facere sed ex, nihil deserunt quos
            ipsam quasi.</p>
    </div>
</body>
```

style.css

Unset

```
    .container {
        border: solid 1px;
    }

    .text {
        width: 300px;
    }
```

Browser output



min and max-width:

min-width

The **"min-width"** property sets the minimum width that an element can have. If the content inside the element requires more width than the specified minimum, the element will expand to accommodate the content. If the content inside the element requires less width than the specified minimum, the element will still have the minimum width.

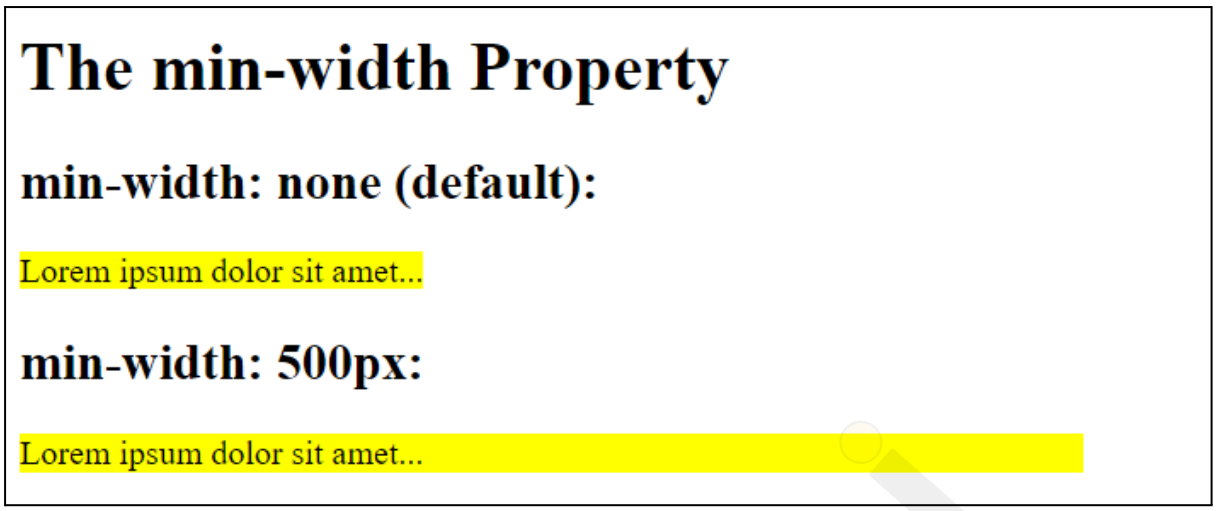
Index.html

```
Unset
<h1>The min-width Property</h1>
<h2>min-width: none (default):</h2>
<span>Lorem ipsum dolor sit amet...</span>

<h2>min-width: 500px:</h2>
<span class="minWidth">Lorem ipsum dolor sit amet...</span>
```

style.css

```
Unset
span {
  background-color: yellow;
}
.minWidth {
  min-width: 500px;
  display: inline-block;
}
```

Browser output



The min-width property defines the minimum width of an element.
If the content is smaller than the minimum width, the minimum width will be applied.
If the content is larger than the minimum width, the min-width property has no effect.

max-width

The **"max-width"** property sets the maximum width that an element can have. If the content inside the element requires less width than the specified maximum, the element will shrink to fit the content. If the content inside the element requires more width than the specified maximum, the element will still have the maximum width.

index.html

```
Unset
<h1>The max-width Property</h1>
<h2>max-width: none (default):</h2>
```

```
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Etiam semper diam at erat pulvinar, at pulvinar felis
blandit.</p>

<h2>max-width: 150px:</h2>
<p class="maxWidth">Lorem ipsum dolor sit amet, consectetur
adipiscing elit. Etiam semper diam at erat pulvinar, at
pulvinar felis blandit.</p>
```

style.css

Unset

```
.maxWidth {
  max-width: 150px;
}
```

Browser output:

Height

CSS height is a property that allows us to control the height of an element on a webpage. It's like how tall or short something should be.
It does not include any padding, borders, or margins

Imagine you have a rectangular box on your webpage, like an image or a div container. The CSS height property lets you decide how much vertical space this box should take up on the screen.

Example

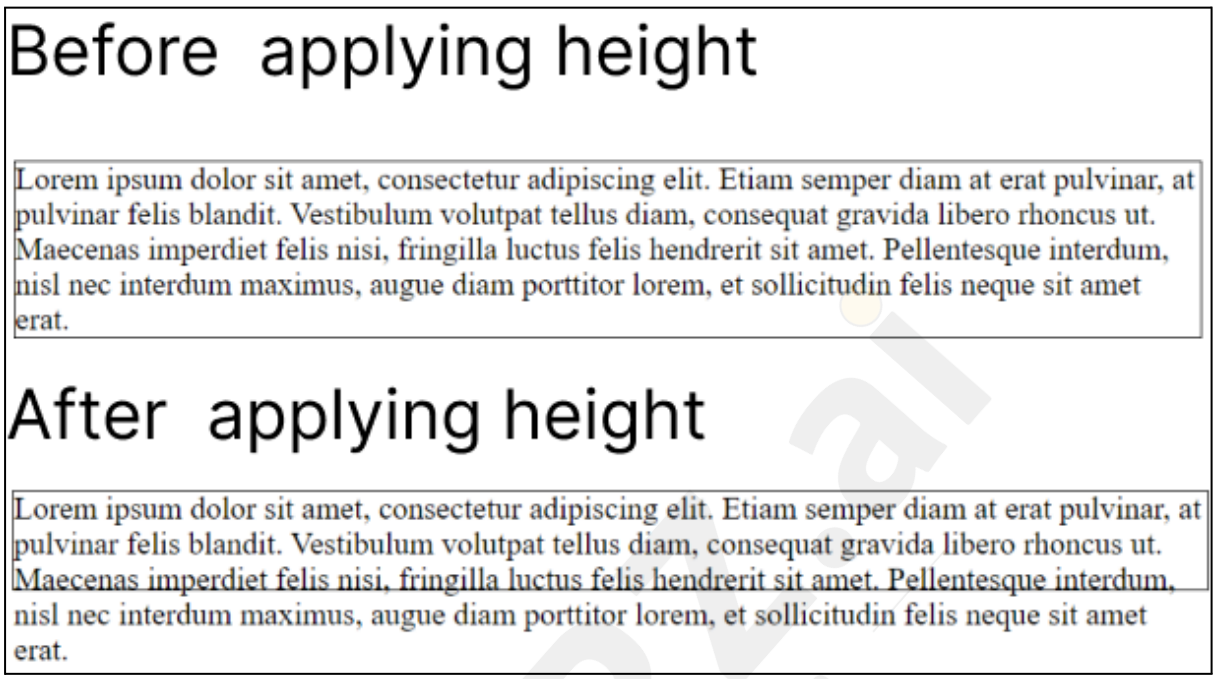
Index.html

```
Unset
<body>

    <p class="text">Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Etiam semper diam at erat pulvinar, at
        pulvinar felis blandit. Vestibulum volutpat tellus diam, consequat
gravida libero rhoncus ut. Maecenas imperdiet
        felis nisi, fringilla luctus felis hendrerit sit amet. Pellentesque
interdum, nisl nec interdum maximus, augue
        diam porttitor lorem, et sollicitudin felis neque sit amet erat.</p>

</body>
```

style.css

```
Unset
    .text {
        height: 50px;
        border: 1px solid black;
    }
```

Browser output



Min and max-height:

min-height:

The "**min-height**" property sets the minimum height that an element can have. If the content inside the element requires more height than the specified minimum, the element will expand to accommodate the content. If the content inside the element requires less height than the specified minimum, the element will still have the minimum height.

Example

index.html

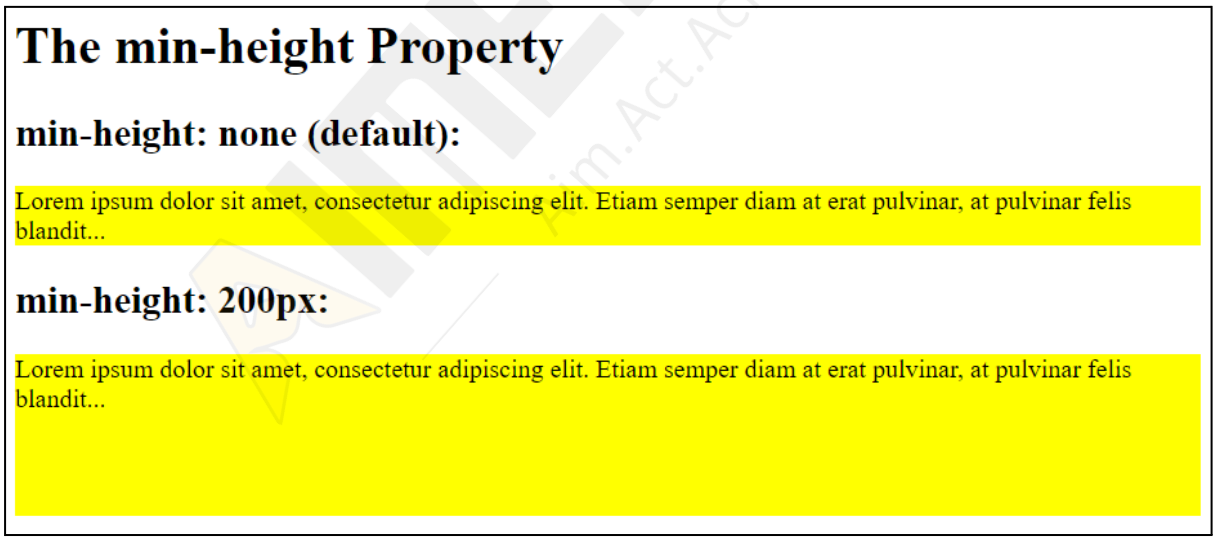
```
Unset
<h1>The min-height Property</h1>
<h2>min-height: none (default):</h2>
```

```
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Etiam semper diam at erat pulvinar, at pulvinar felis
blandit...</p>
<h2>min-height: 200px:</h2>
<p class="minHeight">Lorem ipsum dolor sit amet, consectetur
adipiscing elit. Etiam semper diam at erat pulvinar, at
pulvinar felis blandit...</p>
```

style.css:

```
Unset
.minHeight {
  min-height: 200px;
}
```

Browser output:



If the content is smaller than the minimum height, the minimum height will be applied.
If the content is larger than the minimum height, the min-height property has no effect.

max-height

The "**max-height**" property sets the maximum height that an element can have. If the content inside the element requires less height than the specified maximum, the element will shrink to fit the content. If the content inside the element requires more height than the specified maximum, the element will still have the maximum height.

Example

Index.html

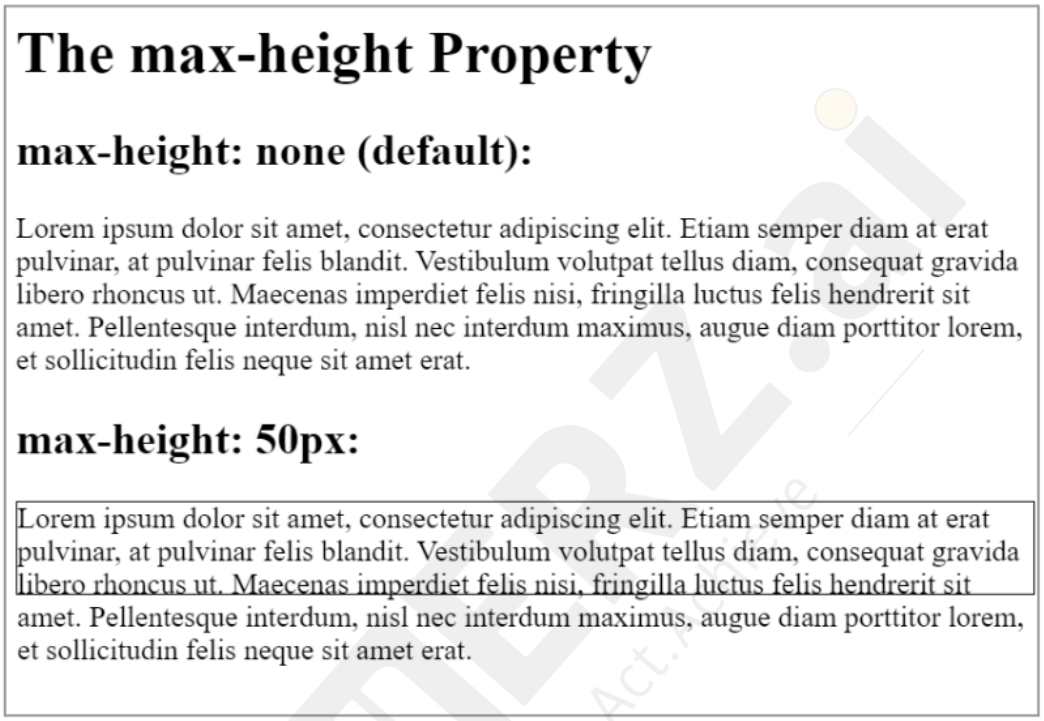
```
Unset
<h1>The max-height Property</h1>
<h2>max-height: none (default):</h2>
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Etiam semper diam at erat pulvinar, at pulvinar felis blandit.
Vestibulum volutpat tellus diam, consequat gravida libero
rhoncus ut. Maecenas imperdiet felis nisi, fringilla luctus
felis hendrerit sit amet. Pellentesque interdum, nisl nec
interdum maximus, augue diam porttitor lorem, et sollicitudin
felis neque sit amet erat.</p>

<h2>max-height: 50px:</h2>
<p class="maxHeight">Lorem ipsum dolor sit amet, consectetur
adipiscing elit. Etiam semper diam at erat pulvinar, at
pulvinar felis blandit. Vestibulum volutpat tellus diam,
consequat gravida libero rhoncus ut. Maecenas imperdiet felis
nisi, fringilla luctus felis hendrerit sit amet. Pellentesque
interdum, nisl nec interdum maximus, augue diam porttitor
lorem, et sollicitudin felis neque sit amet erat.</p>
```

style.css:

```
Unset
.maxHeight {
```

```
  max-height: 50px;
}
```

Browser output:



If the content is larger than the maximum height, it will overflow. If the content is smaller than the maximum height, the max-height property has no effect.

Box sizing

The box-sizing property sets how the total width and height of an element are calculated.

The syntax of the box-sizing property is given as

Unset

```
box-sizing: border-box;  // for border-box
```

```
box-sizing: content-box; // for content-box
```

By default box-sizing is set to “**content-box**”, When using this value, the element's width and height properties will only consider the content, and will not include the margin, padding, or border values.

For example, if we specify the width of an element as 200px, the content will be exactly 200px wide, while the border and padding will be added to the final display, making the element wider than 200px.

The formula for calculating box size in the case of **box-sizing: ”content-box”** is

```
Unset
Actual Width = Width + Horizontal Padding + Horizontal Border
Actual Height = Height + Vertical Padding + Vertical Border
```

It means if we create a box of some specific height and width and then add padding and a border to it, it will look wider than the actual width.

Example: Let's create two div elements with the same height and width, but different borders and padding.
Index.html

```
Unset
<div class="withoutPadding">without padding</div>
<div class="withPadding">with padding</div>
```

Style.css

```
Unset
.withoutPadding {
  height: 50px;
  width: 100px;
  border: solid brown;
}

.withPadding {
  height: 50px;
  width: 100px;
  border: solid purple;
  padding: 10px;
}
```

Browser output:

From the above output, it's clear that the two divs have different sizes, even though we specified the same size for both. The second div appears larger than the first one.

This issue can be resolved by changing box-sizing property to **“border-box”**

The "**border-box**" value for the **box-sizing** property is commonly used to tell the browser to **fit the element's border and padding within its specified width and height**.

For example, if we set an element's width to 200 px, the border and padding we specify will be included in that 200 px, and the content box will adjust to any extra width.

Example: Let's compare border-box and content-box with the below example.

In the below example, we have two containers, one with **box-sizing: ”border-box”** and another with default **box-sizing:”content-box”**(no need to set explicitly)

index.html

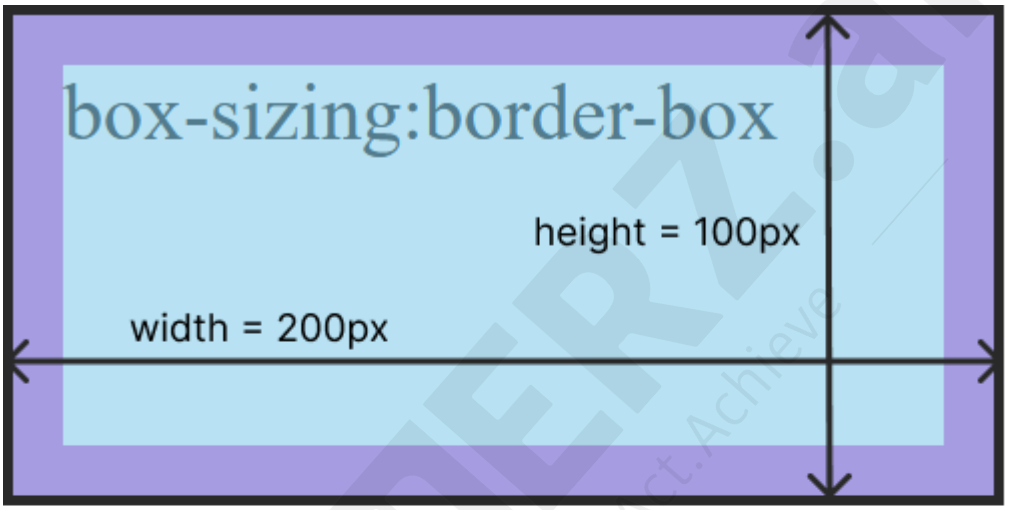
```
Unset
<div class="withBoxSizing">box-sizing:border-box</div>
<div class="withoutBoxSizing">without box-sizing</div>
```

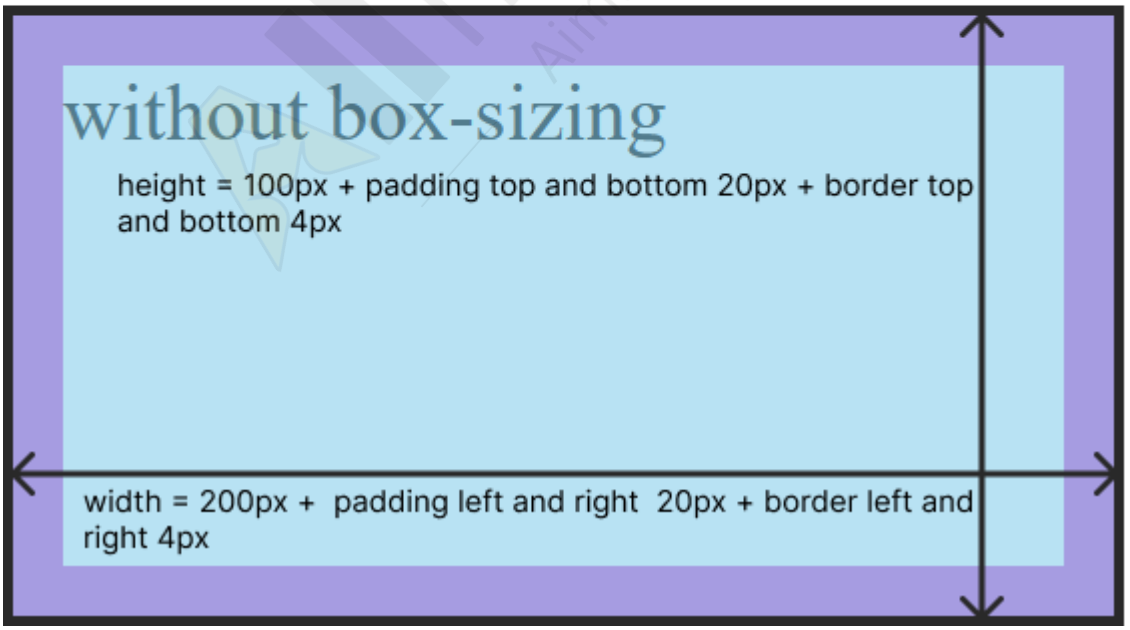
style.css

```
Unset
.withBoxSizing {
  height: 100px;
  width: 200px;
  border: solid 2px;
  padding: 10px;
  box-sizing: border-box;
}
```

```
.withoutBoxSizing {
  height: 100px;
  width: 200px;
  border: solid 2px;
  padding: 10px;
}
```

output:





CSS units and their types

CSS units are used to specify the size, length, and other measurements for various properties like font size, margin, padding, and more. There are several types of units available in CSS, and each has its own unique characteristics and uses.

For example, if you wanted to set the property margin of a paragraph, you would give it a specific value. This value includes the CSS unit.

Look at this simple example:

```
Unset
p {
    margin: 20px;
}
```

In this case, the **margin** is the **property**, **20px;** is the **value**, and **px** (or "pixel") is the CSS **unit**.

Even though it's common to see units like **px** used, the big question is often, **"What's the best unit to use here?"**

When considering all the options for which units to use, it's important to consider the two categories of units: **absolute and relative.**

1. **Absolute** - These are the **fixed-length units**, and the length expressed using the absolute units will appear as **exactly that size**. It is **not recommended to use on-screen**, because the size of the screen varies too much.

The Absolute unit types consist of **Pixels(px), centimeters (cm), Millimetres(mm), inches(in), Points(pt), and Picas(pc).**

2. **Relative** - Relative units are good for styling the responsive site because they scale relative to the window size or the parent (depending on the unit). They specify the length, which is relative to another length property.

The Relative unit types consist of **percentage(%), parent element's font size(em), view height(vh), viewport minimum(vmin), viewport maximum(vmax), character(ch), X-height(ex)**

Usage of different types

Absolute unit Usage

Absolute units can be useful when working on a project where responsiveness is not being considered. For example, desktop apps that can't be resized can be styled to the default dimensions. If the window doesn't scale, you don't need the content either.

Pixels (px): It is used to define the measurement in pixels and is one of the most common length units in CSS.

Example: "font-size: 16px" sets the font size to 16 pixels.

```
Unset
 html-element {
      font-size: 16px;
}
```

Centimeter (cm): It is used to define the measurement in centimeters.
Example: "width: 5 cm," 1 cm is roughly equivalent to 37.8 pixels

```
Unset
 html-element {
      font-size: 5cm;
}
```

Millimeters (mm): It is used to define the measurement in millimeters.
Example: “width: 5mm;” 1 mm is roughly equivalent to 3.78 pixels.

```
Unset
 html-element {
      font-size:5mm;
}
```

Inches (in): It is used to define the measurement in inches.
Example: “height: 1in;” 1 in(one inch) is roughly equivalent to 96 pixels or about 2.54 cm.

```
Unset
 html-element {
      font-size:1in;
}
```

Points (pt): A unit of length commonly used in typography to specify the size of the text. One point is equal to 1/72nd of an inch, and the size of a point is typically defined as the height of the letter "M" in a given font.

Example: “ font-size: 12pt;” 1pt is roughly equivalent to 1.3333 pixels, or 1/72th of an inch.

```
Unset
 html-element { font-size:  12pt; }
```

Picas (pc): It is commonly used in typography to specify the size of the text. One pica is equal to 12 points, or 1/6th of an inch. Example: “font-size: 1pc”. 1pc is roughly 16 pixels or ⅙ of an inch.

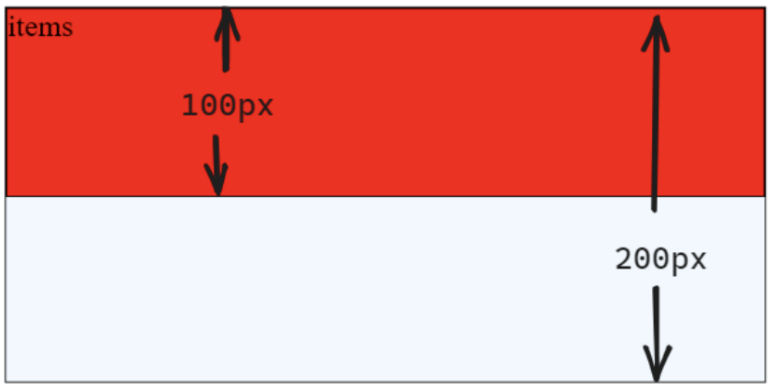
```
Unset
 html-element {
      font-size:1pc;}
```

Relative unit Usage

Relative units are widely used in web development because they provide a flexible and responsive way to size elements and create layouts that adapt to different screen sizes and devices. This unit can be a little more difficult to determine than absolute units, so let’s go through your options in detail.

percentage (%):

It is used to define the measurement as a percentage that is relative to the parent element’s value.
Example: If the height of the parent element is 200px, **“height: 50%”** sets the height of the child element to 100 pixels (50% of 200 pixels).



parent element's font size (em):
It is relative to the font size of the parent element.
if the font size of the parent element is 16 pixels, then 1em would be equal to 16 pixels. If the font size of the parent element is 20 pixels, then 1em would be equal to 20 pixels.

If the font size of a parent element is not explicitly set in CSS, the child elements using the **em** unit will inherit the font size from the nearest ancestor that has a font size explicitly defined. If no ancestor has a defined font size, then the browser's default font size will be used whereas the default font size in most web browsers is set to 16px.

Example: if the font size of the parent element is 16px, "**height: 2em"** sets the height to 32 pixels (2 times 16 pixels). "**width:4em**" sets the width to 64 pixels (4 times 16 pixels) of the child element.

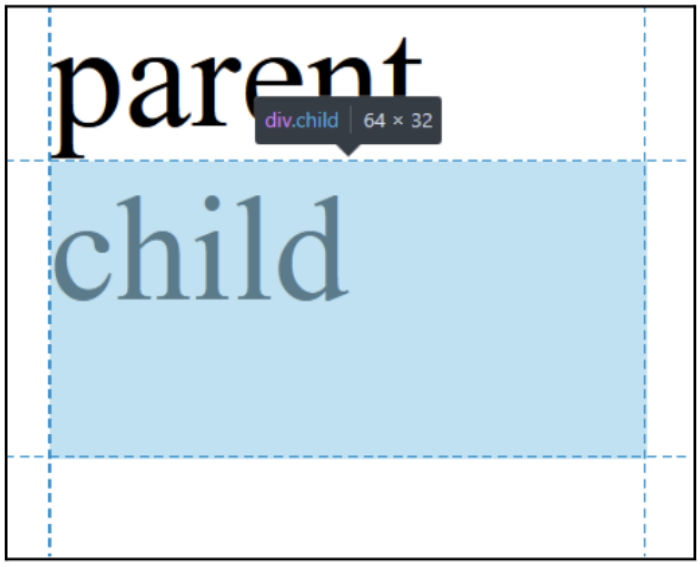
index.html

```
Unset
<div class="parent">parent<div class="child">child</div>
</div>
```

style.css

```
Unset
.parent { font-size: 16px; }
.child {
  height: 2em;  // 2 * 16 = 32
  width: 4em;   // 4 * 16 = 64 }
```

output:



The root element's font size (rem):
it is relative to the font- size of the root element. The **<html>** tag is the root element.
Example: If the font size of the root element is 16px,
“font-size: 1.5rem” sets the font size to 24 pixels. I.e. 1.5 * 16 = 24px
Index.html

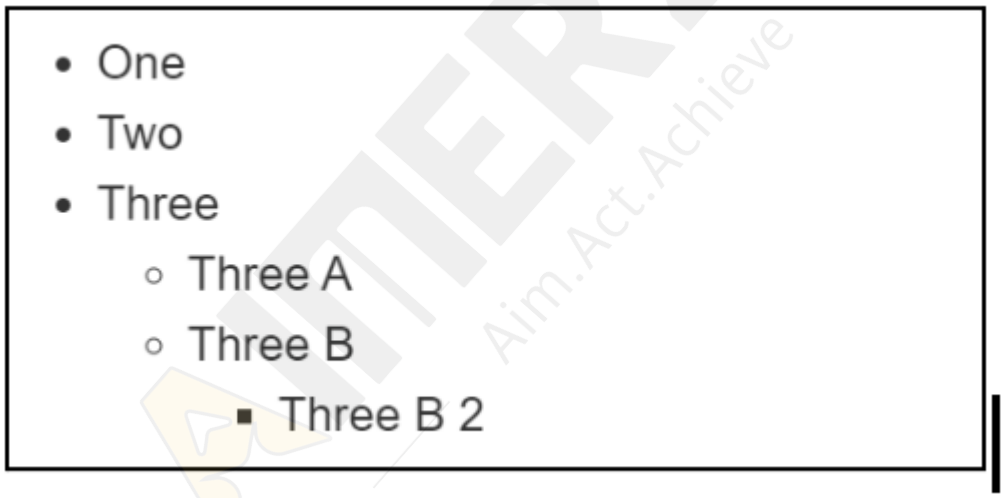
```
Unset
<ul class="rems">
  <li>One</li>
  <li>Two</li>
  <li>Three
    <ul>
      <li>Three A</li>
      <li>Three B
        <ul>
          <li>Three B 2</li>
        </ul>
      </li>
    </ul>
  </li>
</ul>
```

style.css

```
Unset
html {
  font-size: 16px;
}
.rems li {
  font-size: 1.5rem; // i.e 16 * 1.5 = 24px
}
```

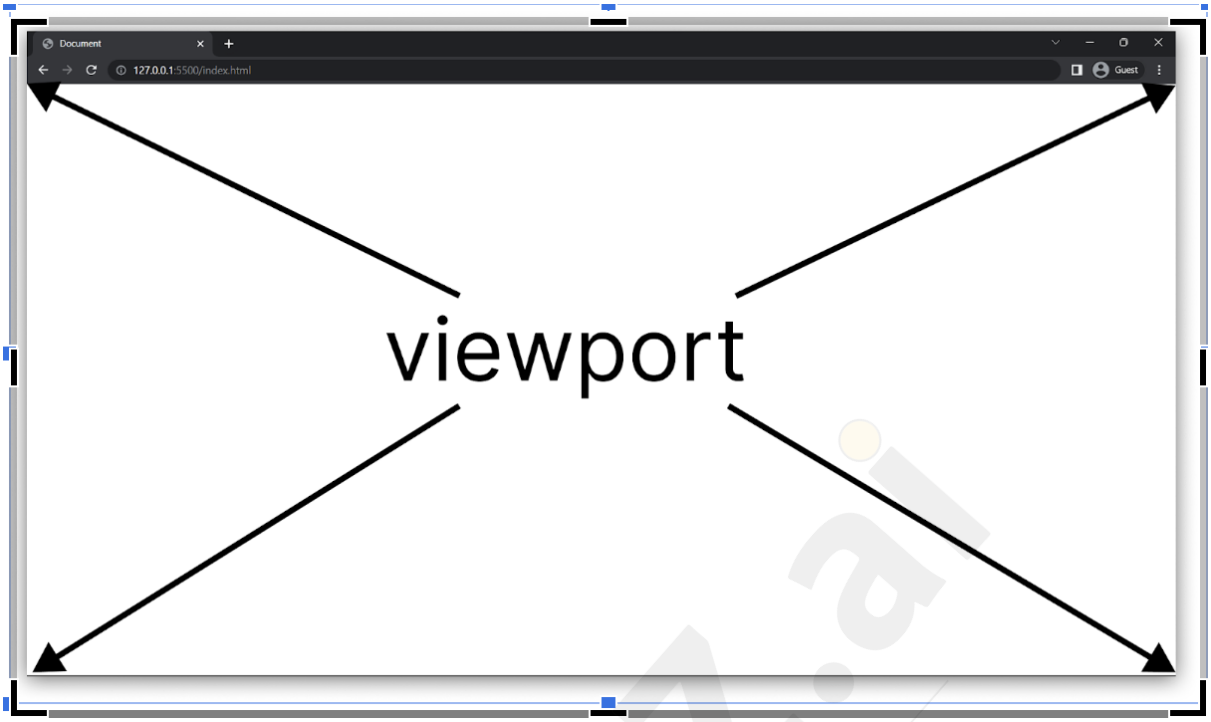
From the above example, the root element is set in the **<html> tag,** with **font size** of **16px**

Browser output:



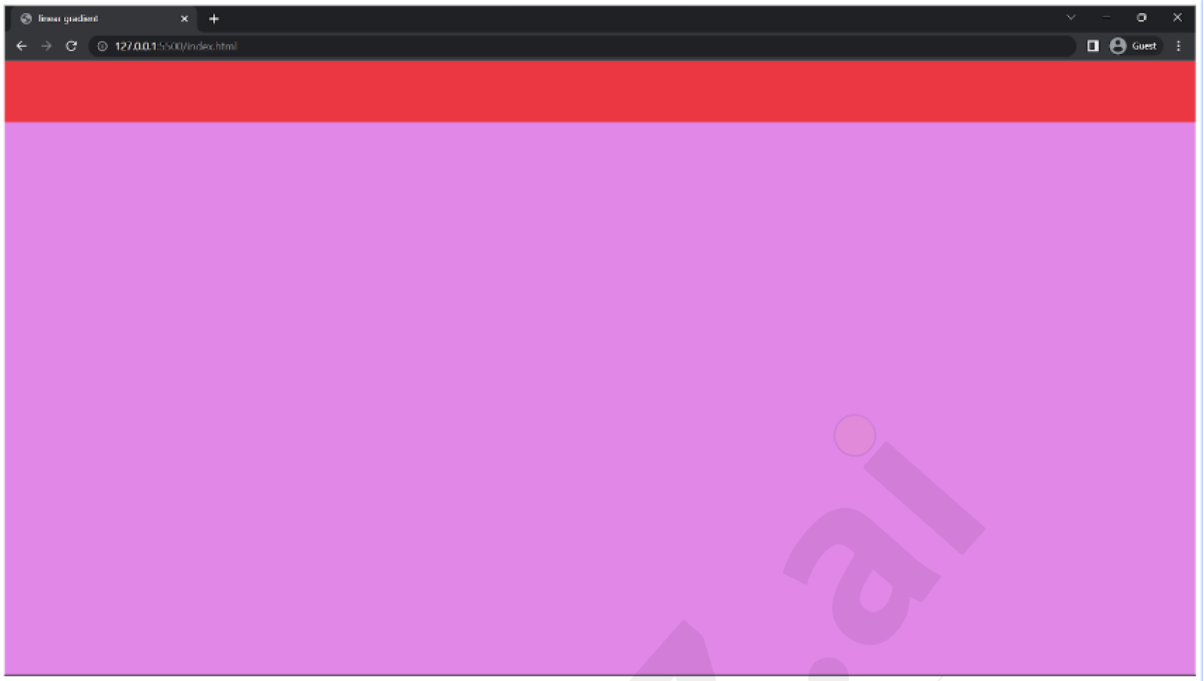
To understand CSS viewport properties like **vh**, **vw**, **vmin,** and **vmax**, let's first understand what the viewport is.

"Viewport" is a term used in web development that refers to the visible area of a web page on a device's screen. It's basically the portion of the screen that you can see when you open a website on your phone, tablet, or computer.

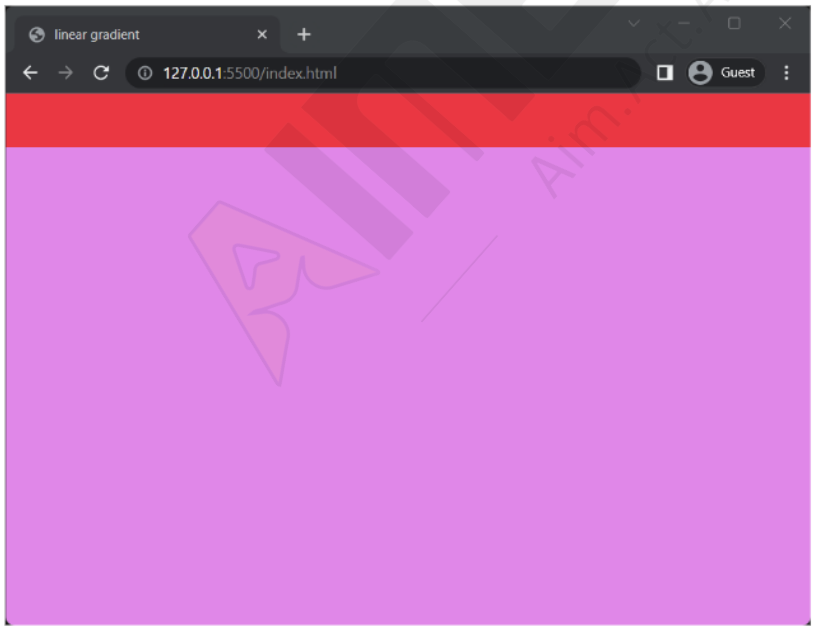


On a desktop device, the viewport size matches the **browser's window size**, **excluding the toolbar and other elements** that aren't part of the page. On a mobile device, the viewport is generally the size of the device's screen.

Let's say we have a webpage that has a header and a main section. The header is set to have a height of 10% of the viewport height, and the main section is set to have a height of 90% of the viewport height.



When we open the webpage in a maximized browser window, the viewport will be the full height and width of the scene. The header will take up 10% of the scene height, and the main section will take up the remaining 90%.



However, if we resize the browser window to be smaller, the viewport will also become smaller. As a result, the header and main section will also become smaller, since their heights are based on the viewport height.

CSS relative units based on the viewport

Viewport height (vh):
It is relative to the height of the viewport.
1 vh = 1%, or 1/100 of the height of the viewport.
Example: **“height: 50vh”** sets the height of the element to 50% of the viewport height.

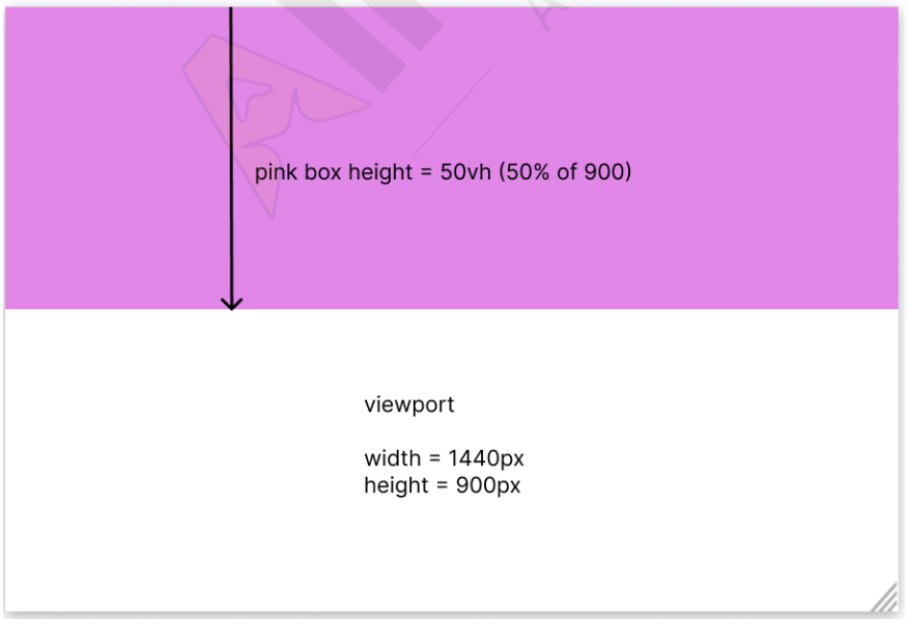
Index.html

```
Unset
<div></div>
```

style.css

```
Unset
div {
  background-color: violet;
  height: 50vh;
}
```

output



Viewport width (vw):
It is relative to the width of the viewport.
1 vw = 1%, or 1/100 of the width of the viewport.
Example: "width: 50vw" sets the width of the element to 50% of the viewport width.

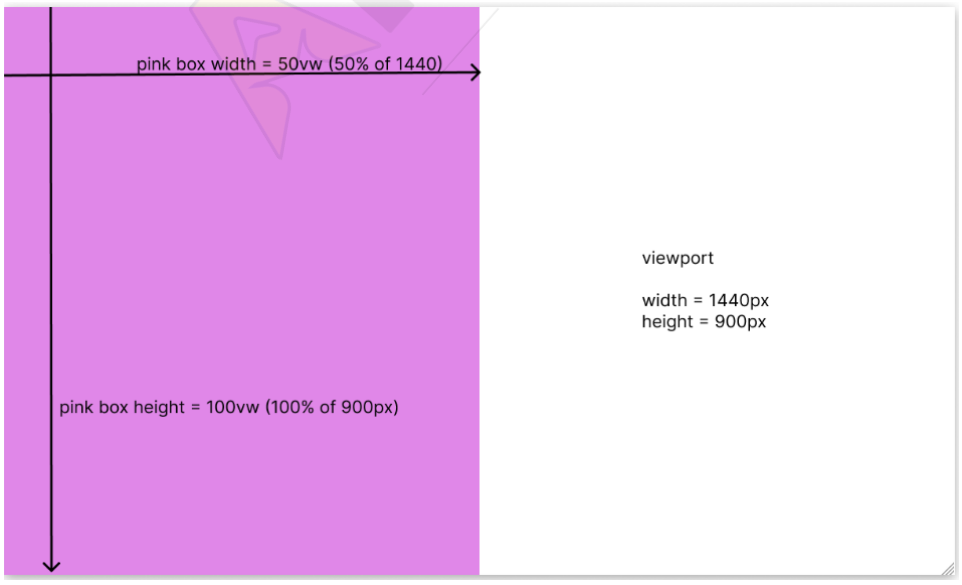
Index.html

```
Unset
<div></div>
```

style.css

```
Unset
div {
  background-color: violet;
  width: 50vw;
  height: 100vh;
}
```

output:



Viewport minimum (vmin):
it is relative to the smaller dimension of the viewport (either the height or width, whichever is smaller) of the viewport.

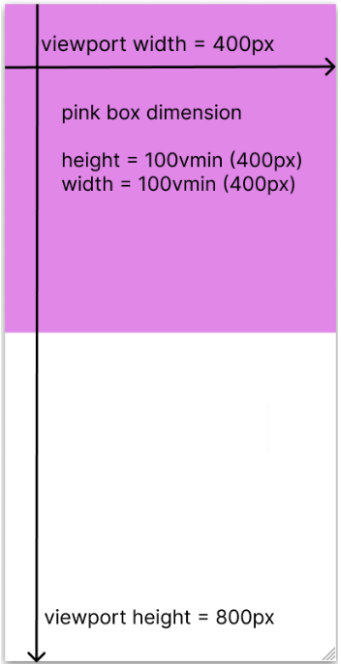
Vmin example in mobile:
If the viewport is 400 pixels wide and 800 pixels tall, the smaller dimension would be the width (400 pixels). Therefore, the width and height of the "div" element would be set to 400 pixels (100% of 400 pixels).
Index.html

```
Unset
<div><div>
```

style.css

```
Unset
div {
  background-color: pink;
  height: 100vmin;
  width: 100vmin;
}
```

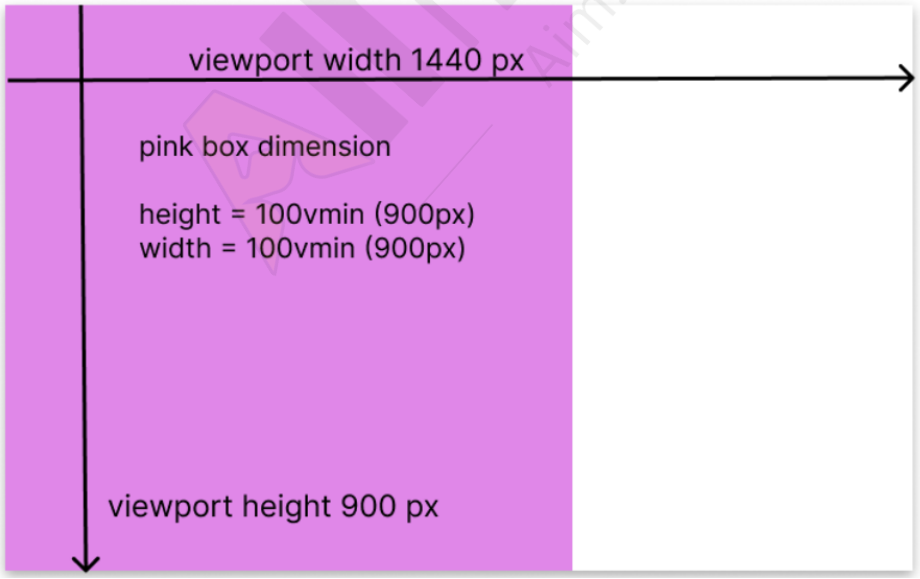
Output:



vmin example in desktop:

If the viewport is **1440 pixels wide** and **900 pixels tall**, the **smaller dimension would be the height** (900 pixels). Therefore, the **width and height of the "div"** element would be **set to 900 pixels** (100% of 900 pixels).

output:



Viewport maximum (vmax):
It is **relative to the larger dimension of the viewport** (either the height or width, whichever is larger) of the viewport.
1 vmax = 1% or 1/100 of the viewport's larger dimension

Vmax example in mobile:
If the viewport is **400 pixels wide** and **800 pixels tall**, the **larger dimension would be the height** (800 pixels). Therefore, the width and height of the "div" element would be set to 400 pixels (50% of 800 pixels).

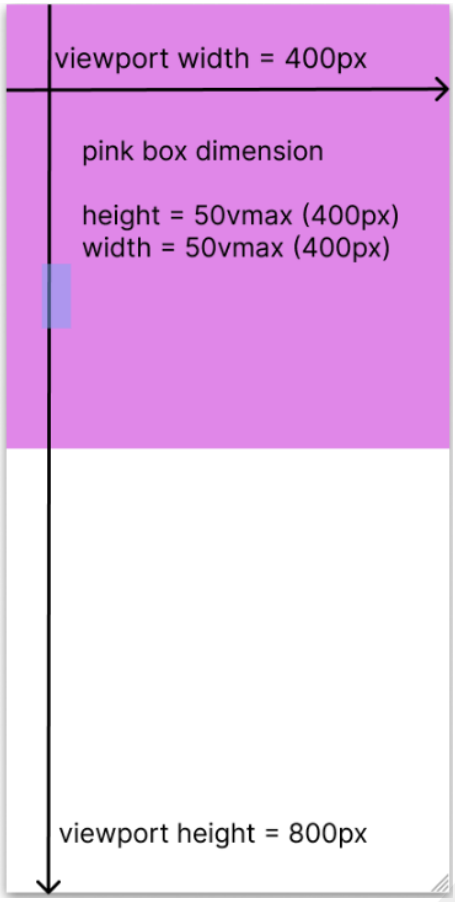
Index.html

```
Unset
<div><div>
```

style.css

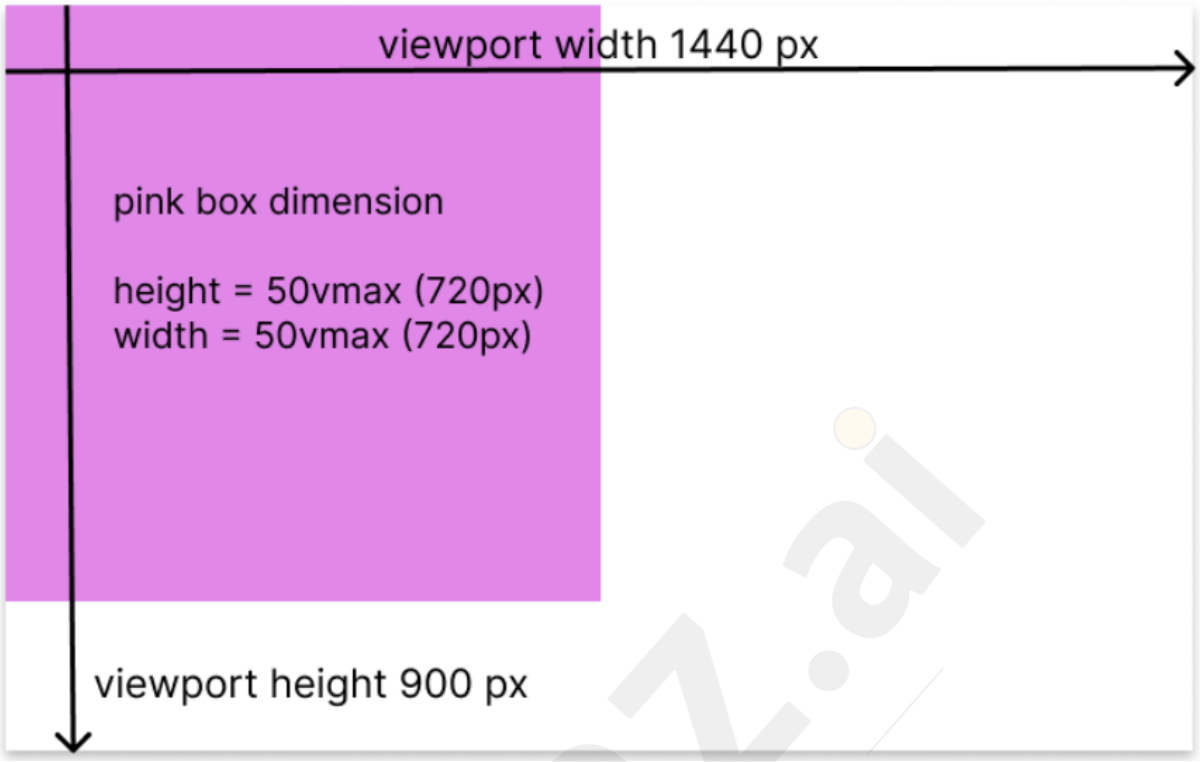
```
Unset
div {
  background-color: pink;
  height: 50vmax;
  width: 50vmax;
}
```

output:



vmax example in desktop: If the viewport is 1440 pixels wide and 900 pixels tall, the larger dimension would be the width (1440 pixels). Therefore, the width and height of the "div" element would be set to 720 pixels (50% of 1440 pixels).

output:



character (ch): This unit is based on the width of the "0" character in the font used for the element.
Example: **“width: 10ch”** sets the width of the element to the width of 10 "0" characters.

X-height (ex): It is relative to the x-height of the font of the element. It is rarely used, the x-height is determined by the height of the lowercase letter "x".
Example: **“font-size: 2ex”** sets the font size to twice the x-height of the font.

Preferred CSS unit and why

In CSS the **px** is a commonly and widely used CSS unit, as it provides a fixed measurement for specifying lengths and sizes

and is ideal for elements with dimensions like borders and margins

However, when it comes to preferred CSS units it depends on the **specific use cases** and **design goals** of the website being non-responsive and responsive.

For **Non-responsive webpages** - For designing or creating a web page that is not responsive a pixels unit (**px**) is mostly used as it has the advantage when there is a need for precise control over an element's sizes and positions, especially for designs that don't adapt to different screens

For **Responsive webpages** - For designing or creating a web page that is responsive, the **Relative** units are always preferred as it allow elements to scale proportionally with the viewport.

Some of the most frequently used relative units can be -

- **Percentage** - Percentage units are relative to the parent container's dimensions. They are commonly used for creating fluid layouts that adapt to different screen sizes.
- **Viewport Percentage (vw, vh, vmin, vmax)** - These units are relative to the viewport dimensions. They are excellent for responsive designs because they scale with the viewport size.
- **em** - The **em** unit is relative to the font size of the nearest parent element. It's useful for creating scalable typography and spacing. This is the most used/recommended out of all.
- **rem** - The **rem** unit is similar to **em,** but it's relative to the root element's font size. This makes it easier to create consistent scaling across the entire webpage.

Understanding CSS Specificity and the !important Rule

What is CSS Specificity?

CSS specificity is a way to determine which style rule applies to an HTML element when there are multiple conflicting styles. It gives a "weight" to each rule based on the type of selectors used. The rule with the higher weight, or specificity, wins.

How to Calculate Specificity

CSS specificity is calculated based on the types of selectors used in the rule. Here's how the different selectors rank in specificity:

1. **Inline styles:** These have the highest specificity because they are applied directly to an HTML element using the style attribute.

```
JavaScript
<div style="color: red;">Hello</div>
```

Specificity value: 1000

2. **ID selectors:** These target elements by their unique ID and have a high specificity.

```
JavaScript
#myId {
    color: red;
}
```

Specificity value: 100

3. **Class selectors:** These target elements by their class and have a lower specificity than ID selectors.

```JavaScript
.classname {
    color: black;
}
```

Specificity value: 10

4. **Element selectors:** These target elements based on their HTML tag and have the lowest specificity.

```JavaScript
h1 {
    color: purple;
}
```

Specificity value: 1

How to calculate Specificity order

CSS specificity is calculated based on the type of selector used and is represented by a four-digit value.

Here is how to calculate specificity -

- **Inline style** gets a specificity value of 1000 and is always given the **highest priority.**
- **Id selector**, for each id selector, 100 is added to the specificity value.
- **Class selector**, for each class selector, 10 is added to the specificity value.
- Element selector, for each type selector, 1 is added to the specificity value.
- Universal selector is ignored, i.e. 0 is added to the specificity value.

The table below shows some examples on how to calculate specificity values:

SELECTOR	SPECIFICITY VALUE	CALCULATION
p	1	1
p.test	11	1 + 10
p#demo	101	1 + 100
<p style="color:pink;">	1000	1000
#demo	100	100
.test	10	10
p.test1.test2	21	1 + 10 + 10
#navbar p#demo	201	100 + 1 + 100
*	0	0 (the universal selector is ignored

Note -

- The selector with the highest specificity value will win and take effect!
- There is one exception to this rule: if you use the **!important** rule, it will even override inline styles.

Let us take an example to understand better:

index.html

```
JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="style.css" />
    <title>Document</title>
  </head>
  <body>
    <h1 class="heading" style="color: yellowgreen">Aimerz!</h1>
  </body>
</html>
```

style.css

```
JavaScript
.heading {
  color: red;
}

h1#heading {
  color: yellow;
}
```

Output:

From the above code we get three specificities i.e -

- h1 (type selector) with specificity value of 1
- h1#heading (type selector with id) with specificity value of 100 + 1 =101

- Inline styling with the specificity of 1000
- Since the third rule with inline styling has the highest specificity value of 1000, this style declaration will be applied.

Understanding Cascading Order

It refers to the order in which style rules are applied when multiple rules target the same HTML element and have the same specificity. The order of the rules in the CSS file decides which rule gets applied.

The order of class names in the HTML does not matter. The CSS specificity depends on the selector, not the order of the class names in the HTML. Similarly, the cascading order is based on the specificity of the rules and the order in which they appear in the CSS file.

If two rules have the same specificity, the rule that appears later in the CSS file will take effect. This means the last rule in the file will be applied if both have the same weight.

Example:

index.html

```
JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="style.css" />
    <title>Document</title>
  </head>
  <body>
    <h1 class="heading1 heading2">Aimerz!</h1>
  </body>
</html>
```

style.css

```javascript
.heading1 {
  color: rgb(144, 228, 144);
  text-decoration: wavy;
}
/* According to the Cascading order rule - the below rule will be applied */
.heading2 {
  color: rgb(210, 235, 21);
  text-decoration: underline;
}
```

Output:

Introduction to !Important in CSS

The “!important” rule is used to give the style rule the highest priority, which means it overrides all other styles applied to the same element.

When the “!important” rule is added to a style property, it will take precedence over any other styles applied to the same element, regardless of where those styles are defined in the CSS file or in the HTML code. This can be useful in situations where a specific style needs to take precedence over all others, such as when debugging a layout issue or when working with third-party CSS libraries.

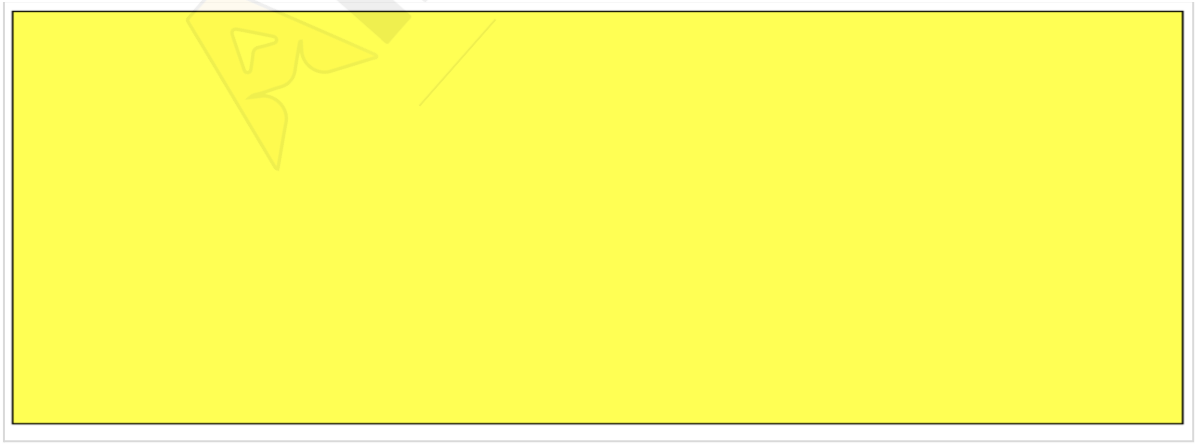
Example
index.html

```
Unset
<div class="container"></div>
```

style.css

```
JavaScript
.container {
  height: 50vh;
  border: 1px solid;
 background-color: yellow !important;

  }
/* This will override the background color*/
.container {
  background-color: green;
 }
```

output -



Drawbacks of using !Important

Some of the drawback point to be considered when using the **!important** CSS rule are as follows -

- It is more difficult to maintain and update styles in the future. Because it takes precedence over other styles.
- It can create specificity conflicts and make it harder to override styles in specific situations
- Debugging issues that arise from the use of "!important" can be more difficult, as styles can be overridden unexpectedly and cause layout issues.

When you have to use and avoid the "!important" CSS rule

When to use -

- **Overriding styles from third-party libraries:** Sometimes, you may need to override styles from a third-party library that you cannot modify directly. In this case, using **!important** can be a quick and easy way to override the styles without modifying the original library.
- **Debugging layout issues:** In some cases, using **!important** can help you quickly identify and fix layout issues that are caused by conflicting styles.

- **Applying accessibility styles:** Using **!important** can help ensure that accessibility styles are applied consistently across your site, even if other styles conflict with them.

When to avoid -

- **Maintenance and scalability:** Overuse of the **!important** rule can make it harder to maintain and scale your styles over time, especially when working on larger projects.
- **Specificity conflicts:** Overuse of the **!important** rule can lead to specificity conflicts, making it harder to override styles in specific situations.
- **Unpredictable behaviour:** Overuse of the !important rule can lead to unpredictable behaviour in your styles, especially when working with larger and more complex stylesheets.

THANK YOU

AIMERZ.ai
Aim.Act.Achieve