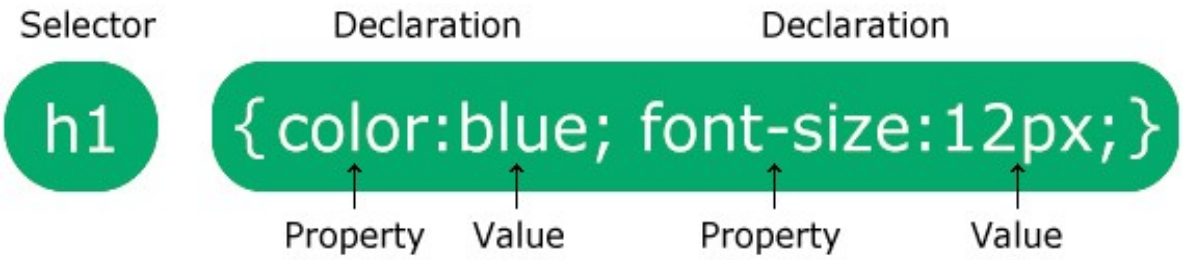
UST

CSS (Cascading Style Sheets)

Introduction to CSS

- Cascading Style Sheets
- CSS describes how HTML elements are to be displayed on screen, paper, or in other media
- It can control the layout of multiple web pages all at once
- External stylesheets are stored in CSS files
- CSS is the language we use to style an HTML document.
- CSS describes how HTML elements should be displayed.

SYNTAX OF CSS



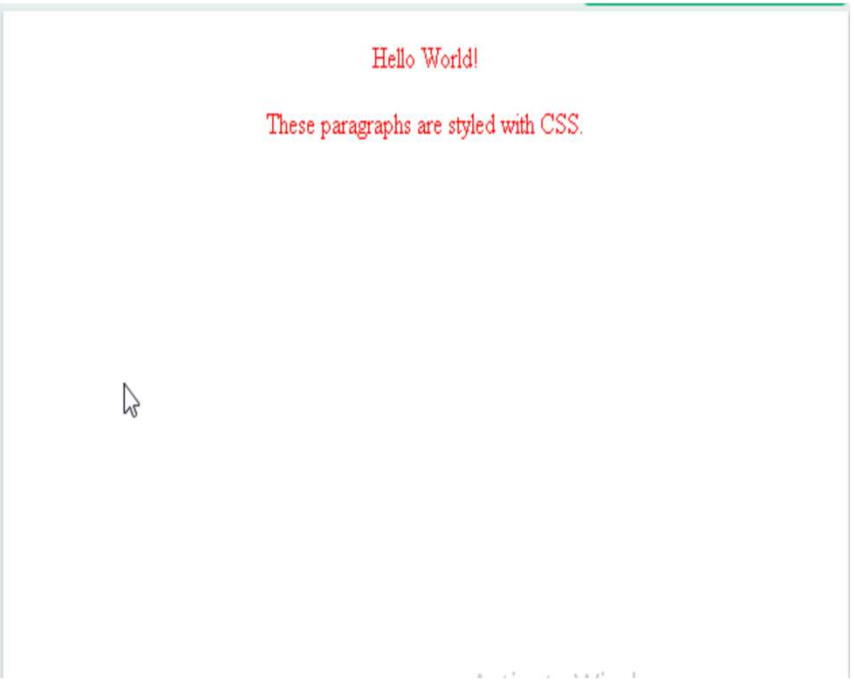
- The selector points to the HTML element you want to style.
- The declaration block contains one or more declarations separated by semicolons.
- Each declaration includes a CSS property name and a value, separated by a colon.
- Multiple CSS declarations are separated with semicolons, and declaration blocks are surrounded by curly braces.

Example

```
<!DOCTYPE html>
<html>
<head>
<style>
p {
  color: red;
  text-align: center;
}
</style>
</head>
<body>

<p>Hello World!</p>
<p>These paragraphs are styled with CSS.</p>

</body>
</html>
```



CSS SELECTORS

- A CSS selector selects the HTML element(s) you want to style.
- We can divide CSS selectors into five categories:

  - Simple selectors (select elements based on name, id, class)
  - Combinator selectors (select elements based on a specific relationship between them)
  - Pseudo-class selectors (select elements based on a certain state)
  - Pseudo-elements selectors (select and style a part of an element)
  - Attribute selectors (select elements based on an attribute or attribute value)

CSS ELEMENT SELECTOR

- The element selector selects HTML elements based on the element name

```
<!DOCTYPE html>
<html>
<head>
<style>
p {
  text-align: left;
  color: green;
}
</style>
</head>
<body>

<p>Welcome to programming.</p>
<p id="para1">Hello World!</p>
<p>!!</p>

</body>
</html>
```

Welcome to programming.

Hello World!

!!

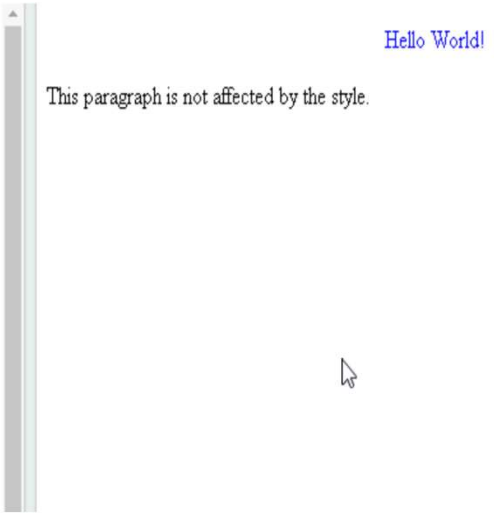
CSS ID SELECTOR

- The id selector uses the id attribute of an HTML element to select a specific element.
- To select an element with a specific id, write a hash (#) character, followed by the id of the element.

```
<!DOCTYPE html>
<html>
<head>
<style>
#para1 {
  text-align: center;
  color: blue;
}
</style>
</head>
<body>

<p id="para1">Hello World!</p>
<p>This paragraph is not affected by the style.</p>

</body>
</html>
```



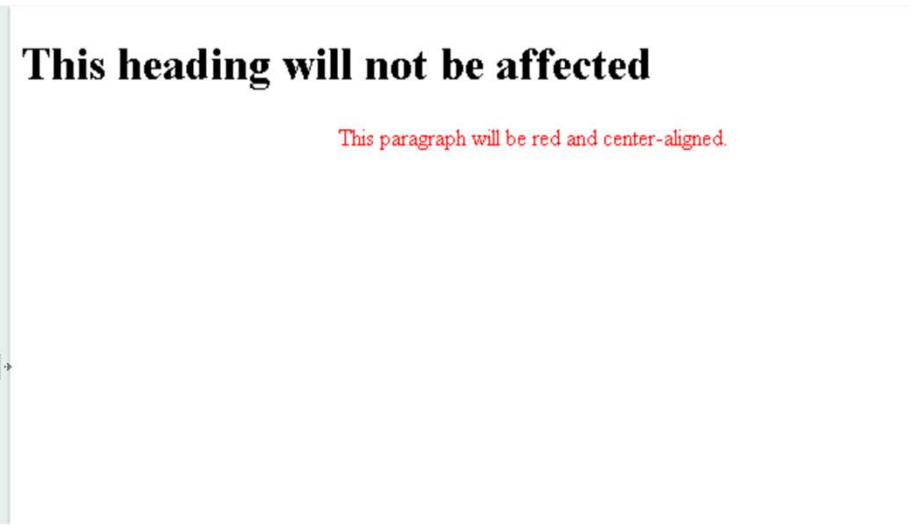
CSS CLASS SELECTOR

- The class selector selects HTML elements with a specific class attribute.
- To select elements with a specific class, write a period (.) character, followed by the class name.

```
<!DOCTYPE html>
<html>
<head>
<style>
p.center {
  text-align: center;
  color: red;
}
</style>
</head>
<body>

<h1 class="center">This heading will not be affected</h1>
<p class="center">This paragraph will be red and center-
aligned.</p>

</body>
</html>
```



CSS UNIVERSAL SELECTOR

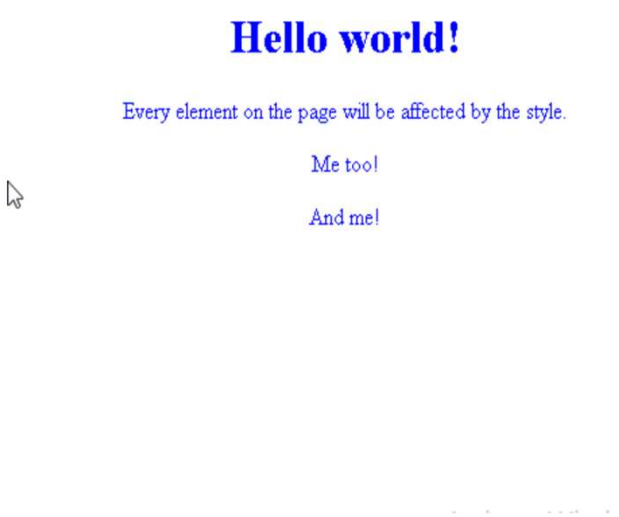
- The universal selector (*) selects all HTML elements on the page

```
<!DOCTYPE html>
<html>
<head>
<style>
* {
  text-align: center;
  color: blue;
}
</style>
</head>
<body>

<h1>Hello world!</h1>

<p>Every element on the page will be affected by the style.</p>
<p id="para1">Me too!</p>
<p>And me!</p>

</body>
</html>
```



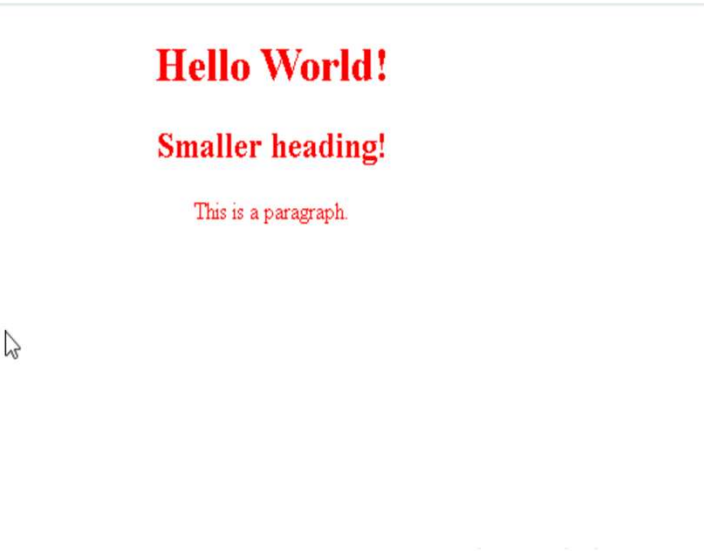
CSS GROUP SLECTOR

- The grouping selector selects all the HTML elements with the same style definitions.

```
<!DOCTYPE html>
<html>
<head>
<style>
h1, h2, p {
  text-align: center;
  color: red;
}
</style>
</head>
<body>

<h1>Hello World!</h1>
<h2>Smaller heading!</h2>
<p>This is a paragraph.</p>

</body>
</html>
```



HOW TO ADD CSS

- **Three Ways to Insert CSS**
  - External CSS
  - Internal CSS
  - Inline CSS

External CSS

- Each HTML page must include a reference to the external style sheet file inside the <link> element, inside the head section.
- The external .css file should not contain any HTML tags.

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

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

Internal CSS

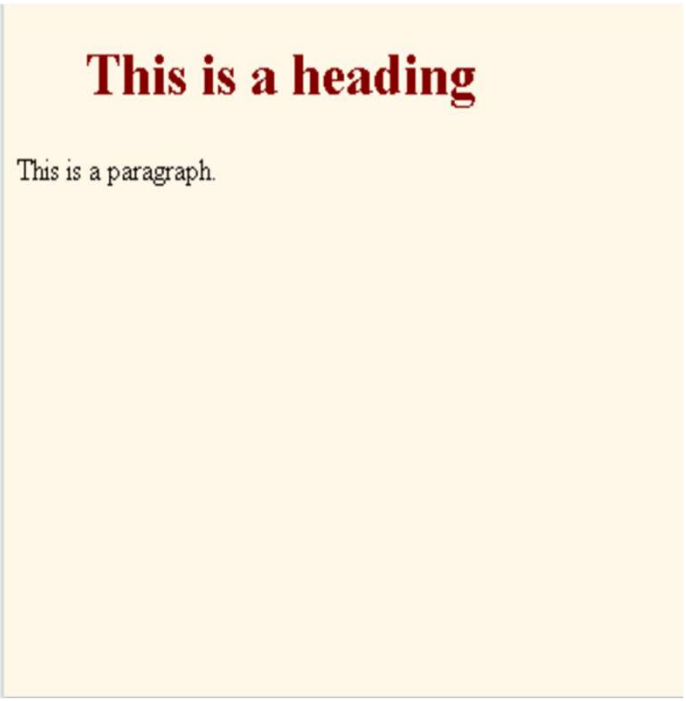
- An internal style sheet may be used if one single HTML page has a unique style.
- The internal style is defined inside the <style> element, inside the head section.

```
<!DOCTYPE html>
<html>
<head>
<style>
body {
  background-color: linen;
}

h1 {
  color: maroon;
  margin-left: 40px;
}
</style>
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
```



Inline CSS

- An inline style may be used to apply a unique style for a single element.
- To use inline styles, add the style attribute to the relevant element. The style attribute can contain any CSS property.

```
<!DOCTYPE html>
<html>
<body>

<h1 style="color:blue;text-align:center;">This is a heading</h1>
<p style="color:red;">This is a paragraph.</p>

</body>
</html>
```

CSS COMMENTS

- CSS comments are generally written to explain your code. It is very helpful for the users who reads your code so that they can easily understand the code.
- Comments are ignored by browsers.
- Comments are single or multiple lines statement and written within /*.....*/ .

CSS COLORS

- Colors are specified using predefined color names, or RGB, HEX, HSL, RGBA, HSLA values.
- RGB Value
- In CSS, a color can be specified as an RGB value, using this formula:
- **RGB(*red, green, blue*)**
- Each parameter (red, green, and blue) defines the intensity of the color between 0 and 255.
- For example, RGB(255, 0, 0) is displayed as red, because red is set to its highest value (255) and the others are set to 0.
- To display black, set all color parameters to 0, like this: RGB(0, 0, 0).
- To display white, set all color parameters to 255, like this: RGB(255, 255, 255).

- HEX Value
- In CSS, a color can be specified using a hexadecimal value in the form:
- ***#rrggbb***
- Where rr (red), gg (green) and bb (blue) are hexadecimal values between 00 and ff (same as decimal 0-255).
- For example, #ff0000 is displayed as red, because red is set to its highest value (ff) and the others are set to the lowest value (00).
- To display black, set all values to 00, like this: #000000.
- To display white, set all values to ff, like this: #ffffff.
- HSL Value
- In CSS, a color can be specified using hue, saturation, and lightness (HSL) in the form:
- **hsl(*hue, saturation, lightness*)**
- Hue is a degree on the color wheel from 0 to 360. 0 is red, 120 is green, and 240 is blue.
- Saturation is a percentage value. 0% means a shade of gray, and 100% is the full color.

•CSS BACKGROUND

- The `background-color` property specifies the background color of an element.
- With CSS, a color is most often specified by:
- a valid color name - like "red"
- a HEX value - like "#ff0000"
- an RGB value - like "rgb(255,0,0)".

CSS background-image

The `background-image` property specifies an image to use as the background of an element.

- CSS background-repeat
- By default, the `background-image` property repeats an image both horizontally and vertically.

- CSS background-attachment
- The `background-attachment` property specifies whether the background image should scroll or be fixed

- CSS background - Shorthand property
- To shorten the code, it is also possible to specify all the background properties in one single property. This is called a shorthand property.

CSS BORDERS

- The CSS border properties allow you to specify the style, width, and color of an element's border.
- CSS Border Style
- The `border-style` property specifies what kind of border to display.
- The following values are allowed:
- `dotted` - Defines a dotted border
- `dashed` - Defines a dashed border
- `solid` - Defines a solid border
- `double` - Defines a double border
- `groove` - Defines a 3D grooved border. The effect depends on the border-color value
- `ridge` - Defines a 3D ridged border. The effect depends on the border-color

- CSS Border Width
- The `border-width` property specifies the width of the four borders.
- The width can be set as a specific size (in px, pt, cm, em, etc) or by using one of the three pre-defined values: thin, medium, or thick:
- CSS Border Color
- The `border-color` property is used to set the color of the four borders.
- The color can be set by:
- name - specify a color name, like "red"
- HEX - specify a HEX value, like "#ff0000"
- RGB - specify a RGB value, like "rgb(255,0,0)"
- HSL - specify a HSL value, like "hsl(0, 100%, 50%)"
- transparent

- CSS Border - Individual Sides
- From the examples on the previous pages, you have seen that it is possible to specify a different border for each side.
- In CSS, there are also properties for specifying each of the borders (top, right, bottom, and left):
- CSS Border - Shorthand Property
- Like you saw in the previous page, there are many properties to consider when dealing with borders.
- To shorten the code, it is also possible to specify all the individual border properties in one property.
- The `border` property is a shorthand property for the following individual border properties:
- `border-width`
- `border-style` (required)
- `border-color`

- CSS Rounded Borders
- The `border-radius` property is used to add rounded borders to an element:
- Normal border
- Round border
- Rounder border
- Roundest border

CSS Margins

- The CSS `margin` properties are used to create space around elements, outside of any defined borders.

- With CSS, you have full control over the margins. There are properties for setting the margin for each side of an element (top, right, bottom, and left).
-

Margin - Individual Sides

- CSS has properties for specifying the margin for each side of an element:
- `margin-top`
- `margin-right`
- `margin-bottom`
- `margin-left`
- All the margin properties can have the following values:
- auto - the browser calculates the margin
- *length* - specifies a margin in px, pt, cm, etc.

- % - specifies a margin in % of the width of the containing element
- inherit - specifies that the margin should be inherited from the parent element.

Margin Collapse

- 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 left and right margins! Only top and bottom margins!

•CSS Padding

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

- All the padding properties can have the following values:
- *length* - specifies a padding in px, pt, cm, etc.
- *%* - specifies a padding in % of the width of the containing element
- inherit - specifies that the padding should be inherited from the parent element

- Padding - Shorthand Property

- To shorten the code, it is possible to specify all the padding properties in one property.
- The `padding` property is a shorthand property for the following individual padding properties:

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

- Padding and Element Width

- The CSS `width` property specifies the width of the element's content area. The content area is the portion inside the padding, border, and margin of an element (the box model).

- So, if an element has a specified width, the padding added to that element will be added to the total width of the element. This is often an undesirable result.

CSS Height, Width and Max-width

- The CSS `height` and `width` properties are used to set the height and width of an element.
- The CSS `max-width` property is used to set the maximum width of an element.

- CSS Setting height and width
- The `height` and `width` properties are used to set the height and width of an element.
- The height and width properties do not include padding, borders, or margins. It sets the height/width of the area inside the padding, border, and margin of the element.

- CSS height and width Values

- The `height` and `width` properties may have the following values:
- `auto` - This is default. The browser calculates the height and width
- `length` - Defines the height/width in px, cm, etc.
- `%` - Defines the height/width in percent of the containing block
- `initial` - Sets the height/width to its default value
- `inherit` - The height/width will be inherited from its parent value

- Setting max-width

- The `max-width` property is used to set the maximum width of an element.
- The `max-width` can be specified in *length values*, like px, cm, etc., or in percent (%) of the containing block, or set to none (this is default. Means that there is no maximum width).
- The problem with the `<div>` above occurs when the browser window is smaller than the width of the element (500px). The browser then adds a horizontal scrollbar to the page.
- Using `max-width` instead, in this situation, will improve the browser's handling of small windows.

CSS FONTS

- CSS font properties allow you to define various aspects of the typography, such as font family, size, style, weight, and more.
- Here's a description of some commonly used CSS font properties:

    - font-family:
        - Specifies the font family or a list of font families for text rendering.
        - Examples:
            - "Arial", "Helvetica", "Times New Roman", "Verdana", sans-serif.

CSS FONTS(cont)

- font-size:
  - Sets the size of the font.
  - Commonly used units include pixels (px), points (pt), em, and percentages (%).
  - Example:
    - font-size: 16px;
- font-style:
  - Defines the style of the font.
  - Values can be normal, italic, or oblique.
  - Example:
    - font-style: italic;
- font-weight:
  - Sets the weight or thickness of the font.
  - Values can be normal, bold, bolder, lighter, or numeric values (e.g., 400, 700).
  - Example:
    - font-weight: bold;

CSS FONTS(cont)

- font-variant:
  - Specifies whether the text should be displayed in small caps.
  - Values can be normal or small-caps.
  - Example:
    - font-variant: small-caps;
- font-stretch:
  - Defines the width or condensedness of the font.
  - Values can be normal, condensed, or expanded.
  - Example:
    - font-stretch: condensed;
- line-height:
  - Sets the height of each line of text.
  - It can be specified as a number, unit value, or percentage.
  - Example:
    - line-height: 1.5;

CSS FONTS(cont)

- letter-spacing:
  - Determines the spacing between characters in text.
  - It can be set as a length value or normal.
  - Example:
    - letter-spacing: 1px;
- text-decoration:
  - Adds decorative styles to text, such as underlining, overlining, and striking through.
  - Values can be none, underline, overline, line-through, or a combination.
  - Example:
    - text-decoration: underline;
- text-transform:
  - Specifies the capitalization of text.
  - Values can be uppercase, lowercase, capitalize, or none.
  - Example:
    - text-transform: uppercase;

CSS ICONS

- CSS icons are visual representations of symbols, objects, or actions that can be styled and displayed using CSS.
- They are commonly used to enhance the user interface of websites or applications. There are several ways to implement CSS icons, including using icon fonts, SVG icons, or icon libraries
- . Here are a few examples:
  - Icon Fonts:
    - Icon fonts are font files that contain vector icons as glyphs.
    - You can use CSS to style and display these icons by assigning the appropriate class or Unicode value to an HTML element.
    - Popular icon font libraries include Font Awesome, Material Design Icons, and Ionicons.
- Example using Font Awesome:
    - <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.3/css/all.min.css">
    -

CSS ICONS(cont)

- SVG Icons:
  - Scalable Vector Graphics (SVG) icons are XML-based vector images that can be styled and animated using CSS.
  - You can embed SVG icons directly in your HTML or use external SVG files.
- Example using inline SVG:
  - <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24" class="icon">
  - <path d="M12 2L1 21h22L12 2zm0 18l8.5-12H3.5L12 20z"/>
  - </svg>

CSS ICONS(cont)

- Icon Libraries:
  - There are various icon libraries available that provide a collection of ready-to-use CSS icons.
  - Offer a range of customizable icons that can be easily implemented in your project. Some popular icon libraries include Bootstrap Icons, Feather Icons, and Material Icons.
- Example using Bootstrap Icons:
  - <link rel="stylesheet" , href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.5.0/font/bootstrap-icons.css">
  -

CSS ICONS(cont)

- Custom CSS Icons:
  - You can create your own custom CSS icons using CSS properties like `border`, `border-radius`, `background`, or `::before` and `::after` pseudo-elements.
  - This allows you to design and style icons according to your specific needs.
- Example of a custom CSS icon:
  -

CSS LINKS

- CSS links in HTML are used to link external CSS files to an HTML document, allowing you to apply styles and formatting to the HTML elements.
- The CSS link is defined using the `<link>` tag within the `<head>` section of an HTML document.
- Here's a description of the various attributes used in the CSS link:
  - href:
    - The `href` attribute specifies the path or URL of the CSS file that you want to link.
    - It can be a relative or absolute path to the CSS file.
- For example:
    - <link rel="stylesheet" href="styles.css">

CSS LINKS(cont)

- rel:
  - The `rel` attribute defines the relationship between the HTML file and the linked CSS file.
  - For CSS links, the value should be set to "stylesheet".
  - It informs the browser that the linked file is a stylesheet.
- For example:
  - <link rel="stylesheet" href="styles.css">

CSS LINKS(cont)

- type:
  - The `type` attribute specifies the MIME type of the linked file.
  - For CSS files, the default value is "text/css", so you don't need to include this attribute in most cases.
  - However, if you are using a different type, you can specify it using the `type` attribute.
- For example:
  - <link rel="stylesheet" href="styles.css" type="text/css">

CSS List

Two main types of lists:

- unordered lists (<ul>)

  Example:

```
ul.a {
    list-style-type: circle;
  }
```

- ordered lists (<ol>)

  Example:

```
ol.c {
    list-style-type: upper-roman;
  }
```

CSS LIST-Properties

1.**`list-style-type`**: Sets the type of bullet or numbering style for the list items. Common values include:

- **`none`**: No bullets or numbering.
- **`disc`**: Default filled circle bullets.
- **`circle`**: Hollow circle bullets.
- **`square`**: Square bullets.
- **`decimal`**: Numeric decimal numbering.

U · S T

Properties

2. **`list-style-image`**: Specifies a custom image to be used as the bullet for list items. You can provide a URL to an image file.

Example:

```
ul {
    list-style-image: url('bullet-image.png');
}
```

Properties

3.**list-style-position**: Defines the position of the list item marker (bullet or number) relative to the text content. Common values include:

inside: The marker is inside the list item.

outside: The marker is outside the list item (default).

Example:

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

Properties

4. **list-style: <type> <position> <image>**: A shorthand property to set all the list-style properties in one declaration. The values are specified in the order: type, position, and image.

Example:

```
ul {
    list-style: square inside url('bullet-image.png');
}
```

CSS TABLE

- Properties:

1.**Table-layout**: Specifies the algorithm used to layout the table cells. Values can be auto (default), fixed, or inherit. The auto value allows the table to adjust its layout based on content, while fixed keeps the table width fixed regardless of content.

Properties

2. **Borders and Cell Spacing:**

- border: Sets the border properties for the table. For example, border: 1px solid #000; will set a 1 pixel solid black border around the table.
- border-collapse: Specifies whether table borders should be collapsed into a single border or separated. Values can be collapse or separate. The collapse value merges adjacent borders, while separate keeps them distinct.
- border-spacing: Sets the spacing between cells when border-collapse is set to separate. For example, border-spacing: 5px; will create a 5-pixel gap between cells.

Properties

3. **Cell Padding:**

- padding: Sets the padding within table cells. You can use shorthand notation like padding: 10px; or specify individual paddings for each side (e.g., padding-left, padding-right, padding-top, padding-bottom).

Properties

4. **Background and Text Colors:**

- background-color: Sets the background color of the table or individual cells. For example, background-color: #f1f1f1; will set a light gray background color.
- color: Sets the text color of the table or cells. For example, color: #333; will set a dark gray text color.

Example:

```
table {
  width: 100%;
  border-collapse: collapse;
}
th, td {
  border: 1px solid #000;
  padding: 8px;
  text-align: center;
}
```

CSS DISPLAY

- 'Display' property is used to control how an element is rendered on a webpage.
- It specifies the type of box or layout model an element should use.
- The display property can take various values, each affecting the element's behavior and appearance.

Block-level Elements

- A block-level element always starts on a new line and takes up the full width available
- The element is a block-level element.
- Examples of block-level elements:

1.
2. <h1> - <h6>
3. <p>
4. <form>
5. <header>
6. <footer>
7. <section>

Inline Elements

- An inline element does not start on a new line and only takes up as much width as necessary.
- This is an inline element inside a paragraph.
- Examples of inline elements:

1.
2. <a>
3. <img>

CSS Max-Width

- A block-level element always takes up the full width available (stretches out to the left and right as far as it can).
- Setting the width of a block-level element will prevent it from stretching out to the edges of its container.
- Then, you can set the margins to auto, to horizontally center the element within its container.
- The element will take up the specified width, and the remaining space will be split equally between the two margins.

CSS Max-Width

- When using width css property, problem occurs when the browser window is smaller than the width of the element. The browser then adds a horizontal scrollbar to the page.
- But when using max-width css property, will improve the browser's handling of small windows. This is important when making a site usable on small devices.

- Example:
  - div.ex1 {
    width: 500px;
    margin: auto;
    border: 3px solid #73AD21;
    }

```
div.ex2 {
  max-width: 500px;
  margin: auto;
  border: 3px solid #73AD21;
}
```

CSS Position

- The position property specifies the type of positioning method used for an element.
- There are five different position values:
  - Static
  - Relative
  - Fixed
  - Absolute
  - Sticky
- Elements are then positioned using the top, bottom, left, and right properties. However, these properties will not work unless the position property is set first. They also work differently depending on the position value.

CSS Position: static

- HTML elements are positioned static by default.
- Static positioned elements are not affected by the top, bottom, left, and right properties.
- An element with position: static; is not positioned in any special way; it is always positioned according to the normal flow of the page.

- Example:
  - div.static {
    position: static;
    border: 3px solid #73AD21;
    }

CSS Position: relative

- An element with position: relative; is positioned relative to its normal position.
- Setting the top, right, bottom, and left properties of a relatively-positioned element will cause it to be adjusted away from its normal position.
- Other content will not be adjusted to fit into any gap left by the element.

- Example:
  - div.relative {
    position: relative;
    left: 30px;
    border: 3px solid #73AD21;
    }

CSS Position: fixed

- An element with position: fixed; is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled.
- The top, right, bottom, and left properties are used to position the element.
- A fixed element does not leave a gap in the page where it would normally have been located.
- Notice the fixed element in the lower-right corner of the page. Here is the CSS that is used:

- Example:
  - div.fixed {
    position: fixed;
    bottom: 0;
    right: 0;
    width: 300px;
    border: 3px solid #73AD21;
    }

CSS Position: absolute

- An element with position: absolute; is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed).
- However; if an absolute positioned element has no positioned ancestors, it uses the document body, and moves along with page scrolling.
- Absolute positioned elements are removed from the normal flow, and can overlap elements.

- Example:
  - div.relative {
    position: relative;
    width: 400px;
    height: 200px;
    border: 3px solid #73AD21;
    }

```
div.absolute {
  position: absolute;
  top: 80px;
  right: 0;
  width: 200px;
  height: 100px;
  border: 3px solid #73AD21;
}
```

CSS Position: sticky

- An element with position: sticky; is positioned based on the user's scroll position.
- A sticky element toggles between relative and fixed, depending on the scroll position.
- It is positioned relative until a given offset position is met in the viewport - then it "sticks" in place (like position:fixed).

- Example:
  - div.sticky {
    position: -webkit-sticky; /* Safari */
    position: sticky;
    top: 0;
    background-color: green;
    border: 2px solid #4CAF50;
    }

CSS Z-Index

- The z-index property specifies the stack order of an element.
- When elements are positioned, they can overlap other elements.
- The z-index property specifies the stack order of an element (which element should be placed in front of, or behind, the others).
- An element can have a positive or negative stack order.
- Z-Index works only on positioned elements and flex items.
- If two positioned elements overlap each other without a z-index specified, the element defined last in the HTML code will be shown on top.

- Example
  - img {
    position: absolute;
    left: 0px;
    top: 0px;
    z-index: -1;
    }

CSS Overflow

The overflow property specifies whether to clip the content or to add scrollbars when the content of an element is too big to fit in the specified area.

The overflow property has the following values:

- visible - Default. The overflow is not clipped. The content renders outside the element's box
- hidden - The overflow is clipped, and the rest of the content will be invisible
- scroll - The overflow is clipped, and a scrollbar is added to see the rest of the content
- auto - Similar to scroll, but it adds scrollbars only when necessary

overflow: visible

By default, the overflow is visible, meaning that it is not clipped, and it renders outside the element's box:
You can use the overflow property when you want to have better control of the layout.
The overflow property specifies what happens if content overflows an element's box.

Example:

```
div { width: 200px; height: 65px;
background-color: coral;
overflow: visible; }
```

overflow: hidden

With the hidden value, the overflow is clipped, and the rest of the content is hidden: You can use the overflow property when you want to have better control of the layout. The overflow property specifies what happens if content overflows an element's box.

Example:

```
div { overflow: hidden; }
```

overflow: scroll

Setting the value to scroll, the overflow is clipped and a scrollbar is added to scroll inside the box.
Note that this will add a scrollbar both horizontally and vertically (even if you do not need it):
You can use the overflow property when you want to have better control of the layout.
The overflow property specifies what happens if content overflows an element's box.

Example :

```
div { overflow: scroll; }
```

overflow: auto

The auto value is similar to scroll, but it adds scrollbars only when necessary:
You can use the overflow property when you want to have better control of the layout.
The overflow property specifies what happens if content overflows an element's box.

Example:

```
div { overflow: auto; }
```

overflow-x and overflow-y

- The overflow-x and overflow-y properties specifies whether to change the overflow of content just horizontally or vertically (or both): overflow-x specifies what to do with the left/right edges of the content.
- The overflow-y specifies what to do with the top/bottom edges of the content. You can use the overflow property when you want to have better control of the layout.
- The overflow property specifies what happens if content overflows an element's box.

Example :

```
div { overflow-x: hidden;
/* Hide horizontal scrollbar */
overflow-y: scroll;
 /* Add vertical scrollbar */ }
```

CSS Layout - float and clear

- The CSS float property specifies how an element should float.
- The CSS clear property specifies what elements can float beside the cleared element and on which side.

The float Property

- The float property is used for positioning and formatting content

E.g. let an image float left to the text in a container.

- The float property can have one of the following values:
- left - The element floats to the left of its container
- right - The element floats to the right of its container
- none - The element does not float (will be displayed just where it occurs in the text). This is default
- inherit - The element inherits the float value of its parent In its simplest use, the float property can be used to wrap text around images.
- Example:

```
img { float: right; }
```

- Example :
- This example clears the float to the left. Here, it means that the <div2> element is pushed below the left floated <div1> element:

```
div1 { float: left;
 }
div2 { clear: left;
}
```

The clear Property

- When we use the float property, and we want the next element below (not on right or left), we will have to use the clear property.
- The clear property specifies what should happen with the element that is next to a floating element.
- The clear property can have one of the following values:
- none - The element is not pushed below left or right floated elements. This is default
- left - The element is pushed below left floated elements
- right - The element is pushed below right floated elements
- both - The element is pushed below both left and right floated elements
- inherit - The element inherits the clear value from its parent

CSS Layout - display: inline-block

- Compared to display: inline, the major difference is that display: inline-block allows to set a width and height on the element.
- Also, with display: inline-block, the top and bottom margins/paddings are respected, but with display: inline they are not.
- Compared to display: block, the major difference is that display: inline-block does not add a line-break after the element, so the element can sit next to other elements.

Example:

```
.container {
  text-align: center; /* Align the boxes in the center */
}
.box {
  display: inline-block;
  width: 100px;
  height: 100px;
  background-color: #f1f1f1;
  margin: 10px;
}
```

CSS ALIGN

- The **align** in CSS is used for positioning the items along with setting the distribution of space between and around content items. We can align the items either horizontally or vertically.

1. **text-align**: Use this property to align the text within the points.

```
Eg:  ul {
        text-align: left; /* or center, right, justify */
     }
```

2. **margin** or **padding**: Adjust the spacing around the points

```
Eg:  ul {
        margin: 0;
        padding-left: 20px; /* or any other value */
     }
```

3. **list-style-position**: This property controls the position of the bullet points.

```
Eg:        ul {
                list-style-position: inside; /* or outside */
             }
```

4.**display: flex**: Use flexbox to align presentation points horizontally or vertically.

```
Eg:      .container {
                display: flex;
                 flex-direction: column; /* or row for horizontal alignment */
                 justify-content: center; /* for vertical alignment */
                 align-items: center; /* for horizontal alignment */
}
```

5.**text-indent**: This property allows to control the indentation of the text within the points.

```
Eg:      ul {
             text-indent: 20px; /* or any other value */
            }
```

CSS Combinators

A combinator is something that explains the relationship between the selectors.

Descendant Selector

- The descendant selector matches all elements that are descendants of a specified element.

```
Eg :    div p {
        background-color: yellow;
         }
```

Child Selector (>)

- The child selector selects all elements that are the children of a specified element.

```
Eg:   div > p {
        background-color: yellow;
         }
```

Adjacent Sibling Selector (+)

•The adjacent sibling selector is used to select an element that is directly after another specific element.
•Sibling elements must have the same parent element, and "adjacent" means "immediately following".

```
Eg:    div + p {
        background-color: yellow;
         }
```

General Sibling Selector (~)

- The general sibling selector selects all elements that are next siblings of a specified element.

```
Eg: div ~ p {
            background-color: yellow;

             }
```

CSS Pseudo-classes

- A pseudo-class is used to select and style elements based on specific states or conditions.

Syntax:

```
selector:pseudo-class {
property: value;
 }
```

1.:first-child: This pseudo-class selects the first child element of its parent. It can be used to target the first presentation point within a container element.

Eg: .container li:first-child {

```
/* CSS styles for the first presentation point */
}
```

2. :last-child: This pseudo-class selects the last child element of its parent. It can be used to target the last presentation point within a container element.

Eg: .container li:last-child {
  /* CSS styles for the last presentation point */
}

3. :nth-child(): This pseudo-class allows to target specific presentation points based on their position within a container element.

Eg: .container li:nth-child(2) {
  /* CSS styles for the second presentation point */
}

.container li:nth-child(odd) {
  /* CSS styles for odd-numbered presentation points */
}

```
.container li:nth-child(even) {
  /* CSS styles for even-numbered presentation points */
}
```

4. :hover: This pseudo-class selects an element when it is being hovered over by the mouse pointer. It can be used to apply styles to a presentation point when it is being hovered.

```
Eg:   .container li:hover {
  /* CSS styles for the presentation point when hovered */
}
```

CSS Pseudo-elements

- A CSS pseudo-element is used to style specified parts of an element.
- For example, it can be used to:
- Style the first letter, or line, of an element
- Insert content before, or after, the content of an element

The syntax of pseudo-elements:

```
selector::pseudo-element {
    property: value;
  }
```

CSS Opacity

- The opacity property specifies the opacity/transparency of an element.
- The opacity property can take a value from 0.0 - 1.0. The lower the value, the more transparent

- Ex.
- ```
  img {
    opacity: 0.5;
  }
  ```
  ```

CSS Navigation Bar

- Navigation bars play a crucial role in web design by providing a means for users to navigate through a website or application.
- They serve as a visual roadmap, guiding users to different sections, pages, or features.
- Two types: vertical and horizontal

➢ Vertical Navigation Bar

- Use the following CSS styles to create a vertical navigation bar:
  - Set <a> elements inside the <li> elements to display as block elements.
  - Specify the width for the <a> elements to control their size.
- Use the "display: block;" property to make the entire link area clickable.
- Customize the width of the <ul> element to determine the overall width of the navigation bar.
- Apply appropriate styles to the <li> and <a> elements for consistent appearance and spacing.
- To create a hover effect, modify the background color and text color of the <a> elements on hover.
- To highlight the active/current page, add an "active" class to the corresponding <li> element.
- Optionally, center the links by adding "text-align: center;" to <li> or <a>.

Ex.

```
ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
  width: 200px;
  background-color: #f1f1f1;
}

li a {
  display: block;
  color: #000;
  padding: 8px 16px;
  text-decoration: none;
}
```

Horizontal Navigation Bar

- There are two ways to create a horizontal navigation bar: using inline or floating list items.
- Inline List Items:
  - Set the <li> elements to display as inline to have them appear on one line.
- Floating List Items:
  - Float the <li> elements to the left to position them next to each other.
  - Apply styles to the <a> elements within the <li> elements to customize their appearance.
- For a fixed navigation bar, use the "position: fixed;" property to keep it at the top or bottom of the page.
- Consider using sticky positioning with "position: sticky;" for a sticky navbar that toggles between relative and fixed positions based on scroll position.

Ex.

```
ul {
  list-style-
type: none;
  margin: 0;
  padding: 0;
  overflow: hidden;
  background-color: #333;
}

li {
  float: left;
}
```

```
li a {
  display: block;
  color: white;
  text-align: center;
  padding: 14px 16px;
  text-decoration: none;
}
```

CSS Dropdowns

- CSS dropdown menus are a common navigation technique used in web design to create hierarchical menus that expand and collapse when the user interacts with them.
- Dropdown menus are typically implemented using HTML unordered lists (**<ul>**) and list items (**<li>**), combined with CSS for styling and JavaScript for interactivity.
- The CSS **display** property is commonly used to control the visibility and positioning of dropdown menus. By default, dropdown menus are hidden (**display: none**) and are shown (**display: block** or **display: flex**) when a trigger element is hovered over or clicked.
- CSS pseudo-classes, such as **:hover** and **:focus**, can be used to apply specific styles to the dropdown menu when the user interacts with it. This allows for visual feedback and enhances the user experience.

CSS Dropdowns Cont...

- CSS properties like **position**, **top**, **left**, and **z-index** are often utilized to control the positioning and stacking order of dropdown menus, ensuring they appear in the correct location relative to their parent or trigger element.
- CSS transitions and animations can be applied to dropdown menus to create smooth and visually appealing effects when they are shown or hidden. This adds a level of interactivity and engagement to the user interface.
- Dropdown menus can be customized extensively using CSS, including the background color, text color, font styles, borders, padding, and spacing. This allows for seamless integration with the overall design and branding of a website or application.
- CSS frameworks, such as Bootstrap and Foundation, provide pre-built styles and components, including dropdown menus, that can be easily incorporated into projects. These frameworks often include additional JavaScript functionality for enhanced dropdown menu behavior

CSS Dropdowns Example

```
<style>
.dropdown {
  position: relative;
  display: inline-block;
}

.dropdown-content {
  display: none;
  position: absolute;
  background-color: #f9f9f9;
  min-width: 160px;
  box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2);
  padding: 12px 16px;
  z-index: 1;
}

dropdown:hover .dropdown-content {
  display: block;
}
</style>

<div class="dropdown">
  <span>Mouse over me</span>
  <div class="dropdown-content">
    <p>Hello World!</p>
  </div>
</div>
```

CSS Image Gallery

- CSS image galleries are a popular way to showcase multiple images in an interactive and visually appealing manner on a web page.
- Image galleries can be created using HTML and CSS, with optional JavaScript for additional functionality.
- CSS grid or CSS flexbox layouts are often used to arrange the images in a grid or a row/column structure within the gallery.
- CSS properties like **display**, **float**, **grid-template-columns**, and **flex** are commonly used to control the layout and positioning of the gallery images.
- CSS transitions and animations can be applied to create smooth image transitions, such as sliding, fading, or zooming effects, when the user interacts with the gallery.

CSS Image Gallery Cont...

- CSS pseudo-classes, like **:hover** or **:focus**, can be utilized to change the appearance of the images when the user hovers over or clicks on them, providing visual feedback and enhancing the user experience.
- CSS properties such as **border**, **box-shadow**, and **border-radius** can be used to add borders, drop shadows, and rounded corners to the images, enhancing their visual presentation.
- CSS media queries can be applied to make the image gallery responsive, adjusting its layout and appearance based on the screen size or device orientation.
- CSS techniques like CSS sprites or lazy loading can be employed to optimize the performance of the image gallery, ensuring faster loading times and better user experience.
- CSS frameworks, such as Bootstrap or Foundation, provide pre-built styles and components for image galleries that can be easily integrated into web projects, saving development time and effort.

CSS Image Gallery Example

```
<head>
<style>
div.gallery {
  margin: 5px;
  border: 1px solid #ccc;
  float: left;
  width: 180px;
}

div.gallery:hover {
  border: 1px solid #777;
}

div.gallery img {
  width: 100%;
  height: auto;
}

div.desc {
  padding: 15px;
  text-align: center;
}
</style>
</head>
```

CSS Image Sprites

- CSS image sprites are a technique used to optimize web page performance by combining multiple images into a single sprite sheet.
- By reducing the number of individual image requests, CSS sprites help minimize server round trips and improve page load times.
- Sprite sheets are created by merging multiple images into a single image file, usually in a grid-like arrangement.
- To display a specific image from the sprite sheet, CSS properties like **background-image** and **background-position** are used.
- The **background-position** property allows you to specify the coordinates within the sprite sheet where the desired image is located.

CSS Image Sprites Cont...

- By adjusting the **background-position** values, different images from the sprite sheet can be displayed for different states or interactions, such as hover or active effects.
- CSS image sprites are commonly used for icons, buttons, or small graphics that appear multiple times on a webpage.
- Using image sprites can significantly reduce the number of HTTP requests, leading to faster page rendering and improved user experience.
- CSS frameworks like Bootstrap and Foundation often provide built-in support for using image sprites, making it easier to incorporate them into your web projects.
- It's essential to consider the dimensions and positioning of each image within the sprite sheet to ensure accurate rendering and alignment.

CSS Image Sprites Example

```
#navlist {
  position: relative;
}

#navlist li {
  margin: 0;
  padding: 0;
  list-style: none;
  position: absolute;
  top: 0;
}

#navlist li, #navlist a {
  height: 44px;
  display: block;
}
```

```
#home {
  left: 0px;
  width: 46px;
  background: url('img_navsprites.gif') 0 0;
}

#prev {
  left: 63px;
  width: 43px;
  background: url('img_navsprites.gif') -47px 0;
}

#next {
  left: 129px;
  width: 43px;
  background: url('img_navsprites.gif') -91px 0;
}
```

CSS Box Model

The CSS box model is a fundamental concept in web design and layout that defines how elements are displayed and how their dimensions are calculated. It describes the structure of an element by dividing it into different components or "boxes."

The CSS box model consists of four main parts:-

- Content: This is the actual content of the element, such as text, images, or other HTML elements. It is defined by the width and height properties.
- Padding: The padding is the space between the content and the border of the element. It can be set using the padding property and affects the size of the element.
- Border: The border surrounds the padding and content of the element. It can be styled using properties like border-width, border-style, and border-color.
- Margin: The margin is the space outside the border, creating a gap between adjacent elements. It can be set using the margin property and affects the positioning of the element.

- Margin: The margin is the space outside the border, creating a gap between adjacent elements. It can be set using the margin property and affects the positioning of the element.

Size Of Box

Height	height + padding-top + padding-bottom + border-top + border-bottom + margin-top + margin-bottom
Width	width + padding-left + padding-right + border-left + border-right + margin-left + margin-right

Here is an example of CSS box model:-

```
.box {
width: 200px;
height: 100px;
padding: 20px;
border: 2px solid black;
margin: 10px;
}
```

Total width = content width + padding + border + margin

= 200px + 20px + 2px + 10px + 10px = 242px

Total height = content height + padding + border + margin

= 100px + 20px + 2px + 10px + 10px = 142px

CSS Text

CSS provides a range of properties to control the appearance and styling of text within HTML elements. Here are some commonly used CSS text properties:

- **`color`**: Sets the color of the text. You can specify it using named colors (e.g., **`red`**, **`blue`**) or using hexadecimal (**`#ff0000`**) or RGB (**`rgb(255, 0, 0)`**) color values.
- **`font-family`**: Defines the font or list of fonts to be used for the text. You can specify multiple fonts separated by commas, allowing the browser to use the first available font on the user's system. For example, **`font-family: Arial, sans-serif;`**.
- **`font-size`**: Sets the size of the text. You can use absolute units (e.g., **`px`**, **`pt`**) or relative units (e.g., **`em`**, **`rem`**) to define the font size.
- **`font-weight`**: Specifies the thickness or boldness of the text. Values can be **`normal`**, **`bold`**, **`lighter`**, **`bolder`**, or a numeric value between 100 and 900.

- **font-style**: Sets the style of the font. Values can be **normal**, **italic**, or **oblique**.
- **text-align**: Aligns the text within its containing element horizontally. Values can be **left**, **right**, **center**, or **justify**.
- **text-decoration**: Adds visual decorations to the text, such as underlines, overlines, or strikethroughs. Values can be **none**, **underline**, **overline**, **line-through**, or a combination separated by spaces (e.g., **underline overline**).
- **text-transform**: Controls the capitalization of the text. Values can be **none**, **capitalize**, **uppercase**, or **lowercase**.
- **line-height**: Sets the height of each line of text. It can be defined as a unitless number, a percentage, or a specific length.
- **letter-spacing**: Adjusts the spacing between characters. It can be set using a length value or the **normal** keyword.

Code example:-

```
p {
font-size: 16px;
Color:red;
}
```

CSS Outline

In CSS, the outline property allows you to add a visible outline around an element, typically used to highlight or emphasize the element. It is similar to the border property, but it does not affect the layout of the element or take up space. The syntax Is like:-

```
selector {
  outline: [outline-width] [outline-style] [outline-color];
}
```

Here's a breakdown of the individual components:

- **outline-width**: Specifies the width or thickness of the outline. It can be set using a length value (e.g., **2px**, **3px**), a relative value (e.g., **0.5em**, **1rem**), or the **thin**, **medium**, or **thick** keywords.
- **outline-style**: Defines the style of the outline. It can be set to **none** (no outline), **dotted**, **dashed**, **solid**, **double**, **groove**, **ridge**, **inset**, or **outset**. Each style produces a different visual effect.
- **outline-color**: Sets the color of the outline. It can be specified using named colors (e.g., **red**, **blue**), hexadecimal (**#ff0000**), or RGB (**rgb(255, 0, 0)**) color values.

Code example:

```
button
{
  outline: 2px dotted blue;
}
```

Difference Between Border and Outline:

Border and outline look similar, but there are some very important differences between them:

- It is not possible to apply a different outline width, style and color for the four sides of an element while in border; you can apply the provided value for all four sides of an element.
- The border is a part of element's dimension while the outline is not the part of element's dimension. Means, it doesn't matter how thick an outline you apply to the element, the dimensions of it won't change.

The outline property is a shorthand property. It can be divided into outline-width, outline-style and outline-color properties. It facilitates you to use any of the property alone, if you need it.

CSS Attribute Selectors

- The **[attribute]** selector is used to select elements with a specified attribute.

Example:

```
a[href] {
  /* Styles for anchor elements with href attribute */
}
```

- The **[attribute="value"]** selector is used to select elements with a specified attribute and value.

Example:

```
input[type="text"] {
  /* Styles for input elements with type="text" */
}
```

- The [attribute~="value"] selector is used to select elements with an attribute value containing a specified word.

```
div[class~="highlight"] {
  /* Styles for div elements with class containing "highlight" as one of the space-separated values */
}
```

- The [attribute|="value"] selector is used to select elements with the specified attribute, whose value can be exactly the specified value, or the specified value followed by a hyphen (-).

Attribute Selectors

- The [attribute^="value"] selector is used to select elements with the specified attribute, whose value starts with the specified value.
- The [attribute$="value"] selector is used to select elements whose attribute value ends with a specified value.
- The [attribute*="value"] selector is used to select elements whose attribute value contains a specified value.

CSS Forms

1. **Basic Structure**: Start by creating an HTML form using the **<form>** tag and add input fields, labels, buttons, and other form elements as needed.

Example:

```
<form>
  <label for="name">Name:</label>
  <input type="text" id="name" name="name">
  <label for="email">Email:</label>
  <input type="email" id="email" name="email">
  <input type="submit" value="Submit">
</form>
```

CSS Forms

2. Targeting Form Elements: Use CSS selectors to target specific form elements. You can use element selectors, class selectors, or ID selectors to apply styles. For example, to target all input fields, you can use the **input** selector:

Example:

```
input {
  /* CSS properties here */
}
```

CSS Forms

3. Styling Input Fields: Customize the appearance of input fields using CSS properties. You can modify properties like width, padding, margin, background-color, border, color, font-size, and more.

Example:

```
input[type="text"],
input[type="email"] {
  width: 100%;
  padding: 10px;
  margin-bottom: 10px;
  border: 1px solid #ccc;
  border-radius: 4px;
}
```

CSS Forms

4. Styling Labels: Style the labels associated with form fields using CSS. You can adjust properties like font-weight, margin, display, color, etc.

Example:

```
label {
  display: block;
  margin-bottom: 5px;
  font-weight: bold;
}
```

CSS Forms

5. Styling Buttons: Customize the appearance of buttons using CSS. You can modify properties like background-color, color, padding, border, cursor, etc.

Example:

```
input[type="submit"] {
  background-color: #4CAF50;
  color: white;
  padding: 10px 20px;
  border: none;
  cursor: pointer;
}
```

CSS Forms

6. Layout and Positioning: You can control the layout and positioning of form elements using CSS properties like display, float, position, width, height, margin, padding, and others.

Counters

- CSS counters are "variables" maintained by CSS whose values can be incremented by CSS rules.
- To work with CSS counters we will use the following properties:
  - counter-reset - Creates or resets a counter
  - counter-increment - Increments a counter value
  - content - Inserts generated content
  - counter() or counters() function - Adds the value of a counter to an element
- To use a CSS counter, it must first be created with counter-reset.

- Example

```
body {
  counter-reset: section;
}

h2::before {
  counter-increment: section;
  content: "Section " counter(section) ": ";
}
```

- This example creates a counter for the page, then increments the counter value for each <h2> element and adds "Section <*value of the counter*>:" to the beginning of each <h2> element

Nesting Counters

- In CSS, nesting counters refer to the ability to create hierarchical counters for numbered elements within nested structures, such as lists or sections.

- They allow you to automatically generate sequential numbers or labels for different levels of nesting.

- The counter-reset and counter-increment properties are used to define and manipulate counters in CSS.

- **Example**

```
body {
  counter-reset: section;
}

h1 {
  counter-reset: subsection;
}

h1::before {
  counter-increment: section;
  content: "Section " counter(section) ". ";
}

h2::before {
  counter-increment: subsection;
  content: counter(section) "." counter(subsection) " ";
}
```

CSS Website Layout

- A website is often divided into headers, menus, content and a footer.

Header

A header is usually located at the top of the website (or right below a top navigation menu). It often contains a logo or the website name.

Navigation Bar

A navigation bar contains a list of links to help visitors navigating through your website.

Content

The layout in this section, often depends on the target users. The most common layout is one (or combining them) of the following:

- **1-column** (often used for mobile browsers)
- **2-column** (often used for tablets and laptops)
- **3-column layout** (only used for desktops)

Footer

The footer is placed at the bottom of your page. It often contains information like copyright and contact info.

CSS Units

- CSS has several different units for expressing a length.
- Many CSS properties take "length" values, such as width, margin, padding, font-size, etc.
- Length is a number followed by a length unit, such as 10px, 2em, etc.
- There are two types of length units:

➢Absolute

➢Relative.

Example

- Set different length values, using px (pixels):
- h1 {
  font-size: 60px;
  }

  p {
  font-size: 25px;
  line-height: 50px;
  }

Absolute Lengths

- The absolute length units are fixed and a length expressed in any of these will appear as exactly that size.
- Absolute length units are not recommended for use on screen, because screen sizes vary so much.

Relative Lengths

- Relative length units specify a length relative to another length property.
- Relative length units scale better between different rendering mediums.
- Examples:

➢Em

➢Ex

➢Ch

➢rem

CSS Specificity

- Every CSS selector has its place in the specificity hierarchy.
- There are four categories which define the specificity level of a selector:

1. Inline styles - Example: <h1 style="color: pink;">
2. IDs - Example: #navbar
3. Classes, pseudo-classes, attribute selectors - Example: .test, :hover, [href]
4. Elements and pseudo-elements - Example: h1, ::before

Example

```
<html>
<head>
  <style>
    .test {color: green;}
    p {color: red;}
  </style>
</head>
<body>

<p class="test">Hello World!</p>

</body>
</html>
```

- In this example, we have added a class selector (named "test"), and specified a green color for this class. The text will now be green (even though we have specified a red color for the element selector "p").

!important

- Used to add more importance to a property/value than normal
- Overrides all the previous styling rules for that specific property on that element

```
p {
    background: blue !important;
}
```

The background color (blue) of 'p' tag won't change even if we try to change it by selectors

Math Functions

Allows mathematical expressions to be used as values

- calc() : performs a calculation to be used as the property value
- max() : uses the largest value, from a comma-separated list of values, as the property value
- min() : uses the smallest value, from a comma-separated list of values, as the property value

Math Functions

```
#div1 {
  width: calc(100% - 100px);
  border: 1px solid black;
}

#div2 {
  width: max(50%, 300px);
  border: 1px solid black;
}
```

```
#div3 {
  width: min(100%, 100px);
  border: 1px solid black;
}
```

UST

THANK YOU

ust.com