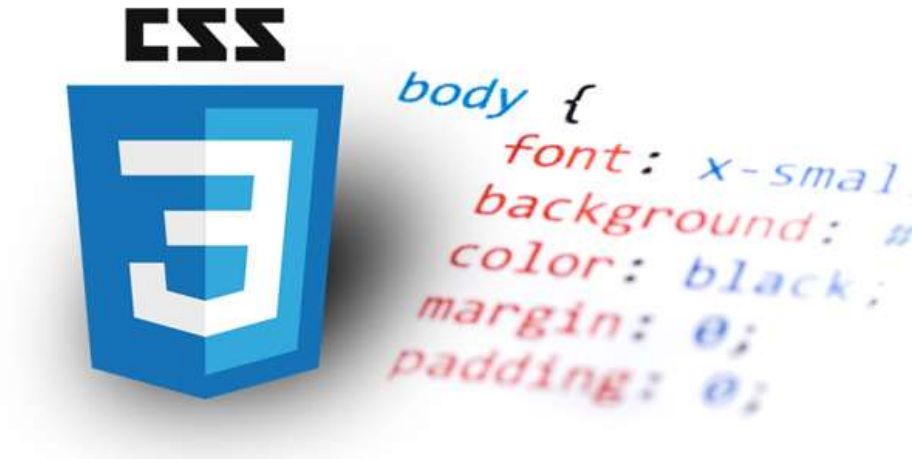
# Module-2
# Advanced CSS



## MODULE II: Advanced CSS

## [L-8hrs.,P- 8hrs.]

- **Advanced CSS**: Layout, Normal Flow, Positioning Elements, Floating Elements, Constructing Multicolumn Layouts, Approaches to CSS Layout, Responsive Design, CSS Frameworks
- **XML:** Basics, demonstration of applications using XML

# Cascading Style Sheet(CSS)

- Cascading Style Sheet(CSS) is used to set the style in web pages that contain HTML elements.
- It sets the background color, font-size, font-family, color, … etc property of elements on a web page

**There are three types of CSS which are given below:**

- **Inline CSS -** by using the style attribute inside HTML elements
- **Internal or Embedded CSS -**by using a <style> element in the <head> section
- **External CSS** - by using a <link> element to link to an external CSS file

# Cascading Style Sheet(CSS)

## Inline CSS :

- Inline CSS contains the CSS property in the body section attached with element is known as inline CSS.
- This kind of style is specified within an HTML tag using the style attribute.
- **<h1 style="color: green; text-decoration: underline;">Hello world!</h1>**
- ** I Love CSS **
- Inline styles are generally the safest way to ensure rendering compatibility across various email clients, programs and devices, but can be time-consuming to write and a bit challenging to manage.

# Cascading Style Sheet(CSS)

## Inline CSS :

```
<!DOCTYPE html>
<html>
   <head>       <title>Inline CSS</title>      </head>

   <body>
      <p style = "color:#009900; font-size:50px; font-style:italic; text-align:center;">
         Web Technology
      </p>
    </body>
</html>
```

# Cascading Style Sheet(CSS)

## Internal or Embedded CSS:

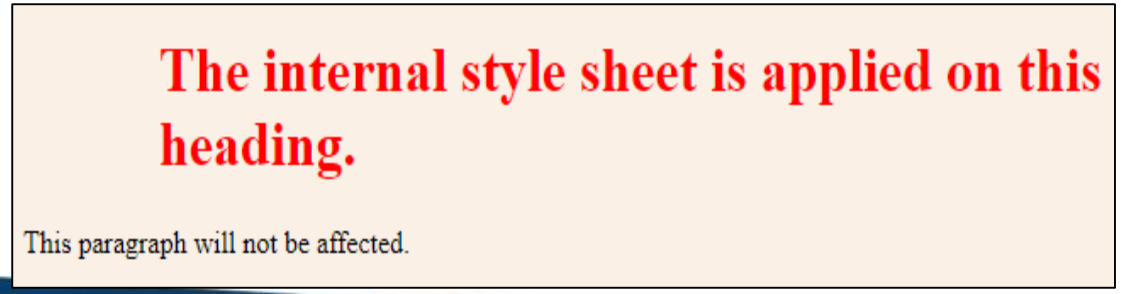
- This can be used when a single HTML document must be styled uniquely.
- It is defined in <head> section of the HTML page inside the <style> tag.

# Cascading Style Sheet(CSS)

## Internal or Embedded CSS:

**Example:**

```
<!DOCTYPE html>
<html>  <head>
<style>
body {
    background-color: linen;
}
h1 {
    color: red;
margin-left: 80px;
}
</style>  </head>
</head>
<body>
<h1>The internal style sheet is applied on this heading.</h1>
<p>This paragraph will not be affected.</p>
</body>
</html>
```



# Cascading Style Sheet(CSS)

## External CSS:

- The external style sheet is generally used when you want to make changes on multiple pages.
- It is ideal for this condition because it facilitates you to change the look of the entire web site by changing just one file.
- It uses the <link> tag on every pages and the <link> tag should be put inside the head section.

**Example:**

```
<head> <link rel="stylesheet" type="text/css" href="mystyle.css"> </head>
```

- The external style sheet may be written in any text editor but must be saved with a **.css extension**. This file should not contain HTML elements.

# Cascading Style Sheet(CSS)

## External CSS:

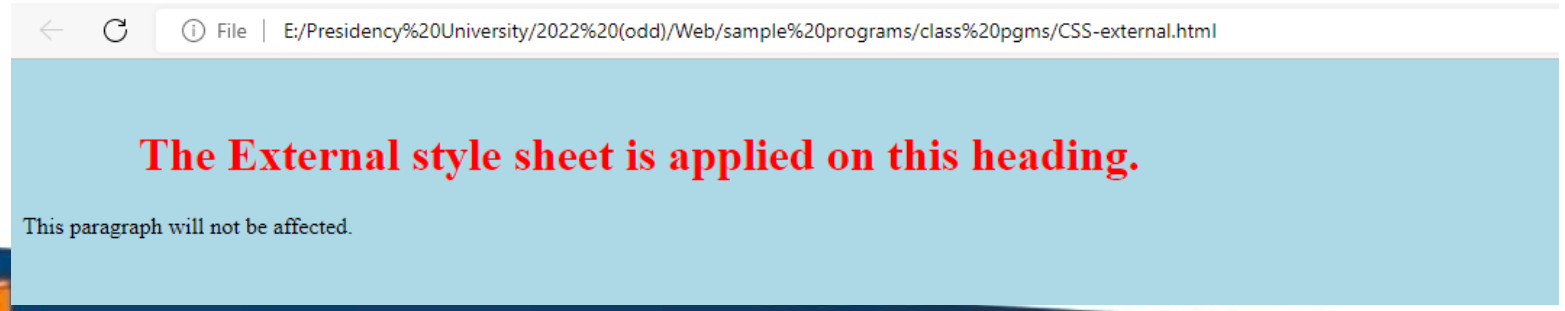
- Let's take an example of a style sheet file named "**mystyle.css**".

```
body {
        background-color: lightblue;
        }
h1 {
    color: navy;
    margin-left: 20px;
}
```

# Cascading Style Sheet(CSS)

## External CSS:

```
<!DOCTYPE html>
<html>
<head>
        <link rel="stylesheet" type="text/css“ href="mystyle.css">
</head>
<body>
<h1>The External style sheet is applied on this heading. </h1>
<p>This paragraph will not be affected.</p>
</body>
</html>
```



# Cascading Style Sheet(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 and multiple lines statement, written within // and /*.....*/ respectively.

# Cascading Style Sheet(CSS)

## Comments

```
<!DOCTYPE html>
<html>
<head>
<style>
p {
    color: blue;
    /* This is a single-line comment */
    text-align: center;
}
/* This is
a multi-line
comment */
</style>
</head>
<body>
<p>Hello Web Technology</p>
</html>
```

Hello Web Technology

# Selectors

The selector identifies which element or elements in the HTML document will be affected by the declarations in the rule. They are a pattern that is used by the browser to select the HTML elements that will receive the style.

- Element Selectors
- Class Selectors
- Id Selectors
- Attribute Selectors
- Pseudo-Element and Pseudo-Class Selectors
- Contextual Selectors

# Element Selectors

- **Element selectors** select an element or group of elements of the HTML document, and the properties are applied on it.
- **Group selector** - Group of elements are separated using commas is called group selector.
- **Universal element selector -** All elements of the document can be selected by using the * (asterisk) character.

Example :

```
p{ font-style:italic; font-weight:bold;}

h1,h2{font-weight:bold; color:red;}

*{ color:blue;}
```

## Example

```
<head>
        <title>Student details </title>
        <style>
        *{ color:blue;}
        h1{color: red;}
        </style>
</head>
<body>
        <h1 >Student Info</h1>
        <p >Amith</p>
        <p>Easy to learn.</p>
        <hr/>
        <p >Bhushan</p>
        <p>Very much special.</p>
        <hr/>
</body>
```

# Class Selectors

A **class selector** allows to simultaneously target different HTML elements. The HTML elements with the same class attribute value, can be styled by using a class selector.

Syntax:  period (.)classname{ styles;}

Example:

```
.first {
font-style: italic;
color: red;
}

.cen {text-align: center;}
```

## Example

```
<head>
<title>Student details </title>
<style>
.first {
font-style: italic;
color: red;
}
</style>
</head>
<body>
            <h1 class="first">Student Info</h1>
            <div>
            <p class="first">Amith</p>
            <p>Easy to learn.</p>
            </div>
          <hr/>
          <div>
          <p class="first">Bhushan</p>
          <p>Very much special.</p>
            </div>
            <hr/>
</body>
</html>
```

# Id Selectors

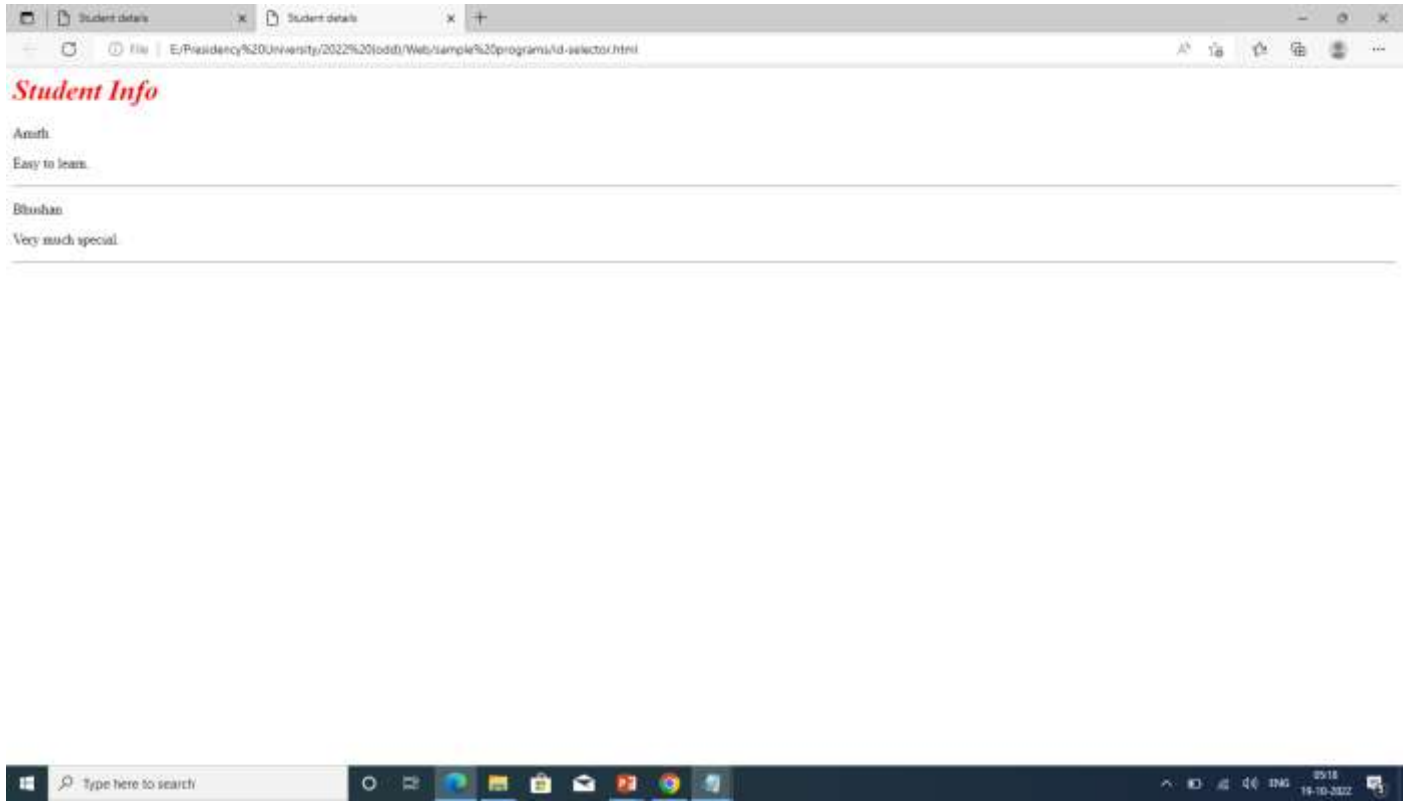
An **id selector** allows to assign style to a specific element by its id attribute.
Syntax: hash (#)id name

Eg:

```
<head>
<title>Student details </title>
<style>
#first {
font-style: italic;
color: red;
}
</style>
</head>
<body>
<h1 id="first">Student Info</h1>
<div>
```

# Example

```
<head>
<title>Student details </title>
<style>
#first {
font-style: italic;
color: red;
}
</style>
</head>
<body>
<h1 id="first">Student Info</h1>
<p> Amith</p>
<p>Easy to learn</p>
<hr/>
<p >Bhushan</p>
<p>Very much special.</p>
<hr/>
</body>
</html>
```



# Attribute Selectors

An **attribute selector** provides a way to select HTML elements either by the presence of an element attribute or by the value of an attribute.

Eg: [src], [src$=".jpg"] , a[href*="gala"] etc.

[src] – selects all the elements which have 'src' as an attribute
[src$=".jpg"] – selects all the elements with 'src' value ending with .jpg
a[href*="gala"] – selects <a> tag with 'href' value having text 'gala'.

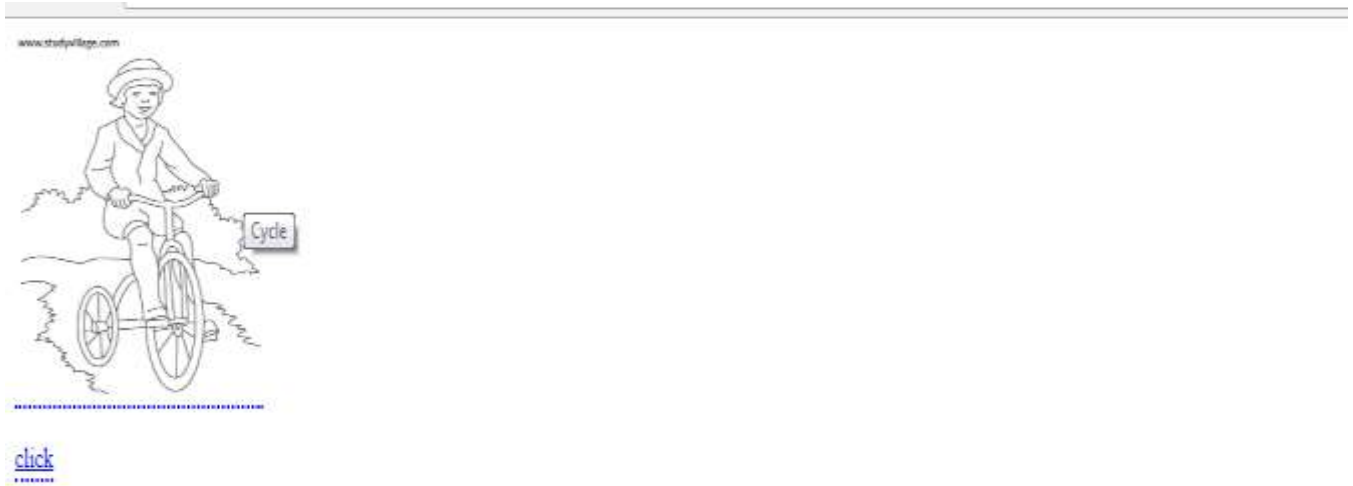
Suppose, we want special attention of user when a pop-up tooltip is available for a link or image. This can be done by using the following attribute selector: [title] { … }

Example -

```
[title] {
 cursor: help;
 padding-bottom: 3px;
 border-bottom: 2px dotted blue;
 }
```

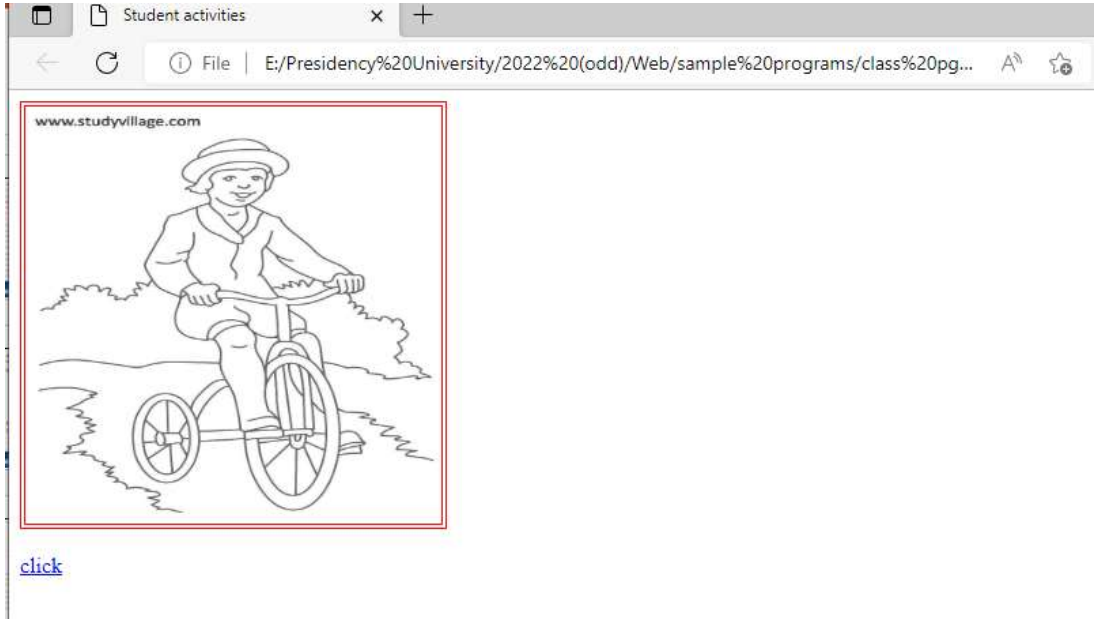
-

```
<head >
 <title>Student activities</title>
 <style>
 [title] {
 cursor: help;
 padding-bottom: 3px;
 border-bottom: 2px dotted blue;
 }
 </style>
</head>
<body>
<div>
<img src="cycling.jpg" title="Cycle" />
<a href = “s1.jpg” title= “link to photo”> click </a>
</div>
</body>
```



| Selector | Matches | Example |
|---|---|---|
| [] | A specific attribute. | `[title]`<br>Matches any element with a title attribute |
| [=] | A specific attribute with a specific value. | `a[title="posts from this country"]`<br>Matches any <a> element whose title attribute is exactly `"posts from this country"` |
| [~=] | A specific attribute whose value matches at least one of the words in a space-delimited list of words. | `[title~="Countries"]`<br>Matches any `title` attribute that contains the word `"Countries"` |
| [^=] | A specific attribute whose value begins with a specified value. | `a[href^="mailto"]`<br>Matches any <a> element whose `href` attribute begins with `"mailto"` |
| [*=] | A specific attribute whose value contains a substring. | `img[src*="flag"]`<br>Matches any <img> element whose `src` attribute contains somewhere within it the text `"flag"` |
| [$=] | A specific attribute whose value ends with a specified value. | `a[href$=".pdf"]`<br>Matches any <a> element whose `href` attribute ends with the text ".pdf" |

```
<title>Student activities</title>
<style>
[src$=".jpg"] {
cursor: help;
padding: 3px;
border: 4px double red;
}
</style>
</head>
<body>
<div>
<img src="cycle-riding.jpg" title="Cycle" width="300" height="300"/>
<br /><br/>
<a href = “s1.jpg” title= “link to photo”> click </a>
</div>
</body>
```



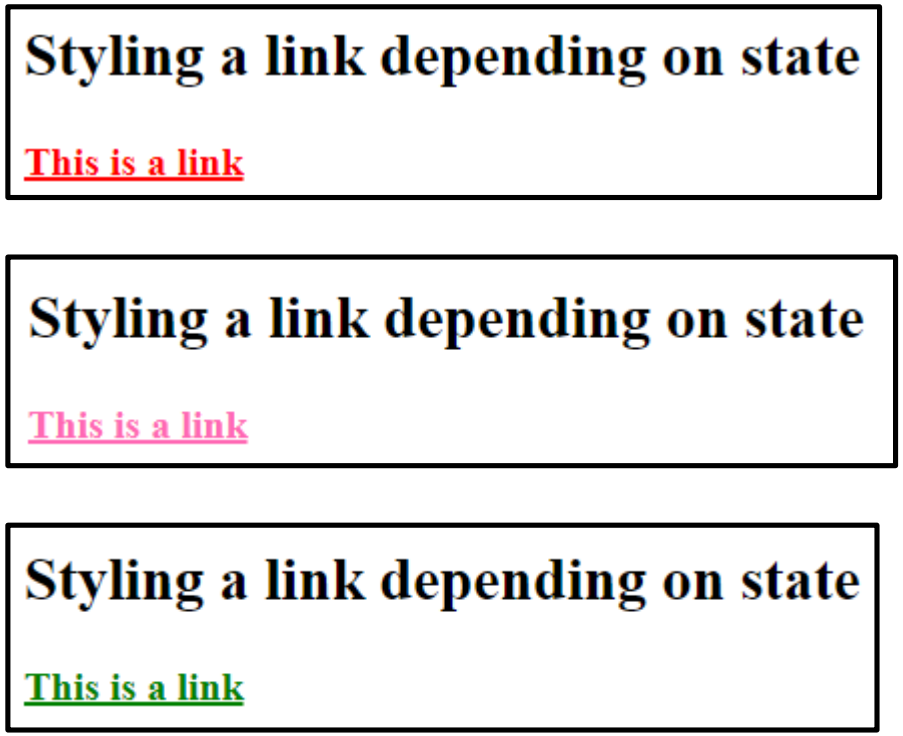
# Pseudo-classes

- A pseudo-class is used to define a special state of an element that is recognizable.
- For example, it can be used to:
  - Style an element when a user moves the mouse over it
  - Style visited and unvisited links differently
  - Style an element when it gets focus
  - Style the first letter of a paragraph etc.

| Selector | Type | Description |
|---|---|---|
| `a:link` | pseudo-class | Selects links that have not been visited |
| `a:visited` | pseudo-class | Selects links that have been visited |
| `:focus` | pseudo-class | Selects elements (such as text boxes or list boxes) that have the input focus. |
| `:hover` | pseudo-class | Selects elements that the mouse pointer is currently above. |
| `:active` | pseudo-class | Selects an element that is being activated by the user. A typical example is a link that is being clicked. |
| `:checked` | pseudo-class | Selects a form element that is currently checked. A typical example might be a radio button or a check box. |
| `:first-child` | pseudo-class | Selects an element that is the first child of its parent. A common use is to provide different styling to the first element in a list. |
| `:first-letter` | pseudo-element | Selects the first letter of an element. Useful for adding drop-caps to a paragraph. |
| `:first-line` | pseudo-element | Selects the first line of an element. |

# Pseudo class (examples)

```
<html>
<head>
<style>
a:link {      color: red;       }
a:visited {  color: green;    }
a:hover {   color: hotpink;  }
</style>
</head>
<body>
<h2>Styling a link depending on state</h2>
<p><b><a href="https:\\www.google.com" target="_blank">This is a link</a></b></p>
</body>
</html>
```

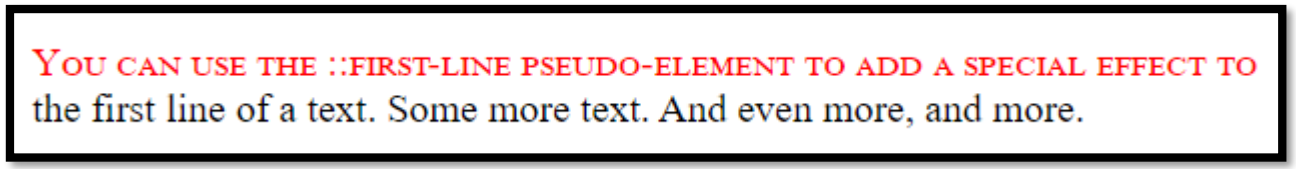


# Pseudo-Elements

- 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 ::first-line Pseudo-element

```
<html>
<head>
<style>
p::first-line {
  color: #ff0000;
  font-variant: small-caps;
}
</style>
</head>
<body>
<p>You can use the ::first-line pseudo-element to add a special effect to the first line of a text. Some more text. And even more, and more.</p>

</body>

</html>
```



# Pseudo element (examples)

```
<html>
<head>
<style>
    p:first-letter { font-size: 300%; color: red;}
    p:first-line {text-decoration: underline;}
</style>
</head>
<body>
<p>  This is a Demo to demonstrate the use of Pseudo-element selectors. Here the first letter and first line of paragraph tag are effected. The first letter appears larger and is red colored. the first line is underlined.... </p>
</body>
</html>
```

# The ::first-letter Pseudo-element

```
<style>
p::first-letter {
  color: #ff0000;
  font-size: xx-large;
}
```

You can use the ::first-letter pseudo-element to add a special effect to the first character of a text!

```
<style>
p.intro::first-letter {
  color: #ff0000;
  font-size: 200%;
}
<p class="intro">This is an introduction.</p>
```

This is an introduction.

# Contextual Selectors

A **contextual selector** (in CSS3 also called **combinators**) allows to select elements based on their ancestors, descendants, or siblings. It selects elements based on their context or relation to other elements in the document tree.

Eg – Descendant selector matches the specified element that is contained within another element.

div p – selects <p> tag that is contained within  tag.

| Format | Example | Description |
|---|---|---|
| ***Descendant Selector*** - *element element* | div p | Selects all <p> elements inside  elements |
| **Child Selector** - element > element | div > p | Selects all <p> elements where the parent is a  element |
| **Immediate Adjacent Sibling** Selector element +element | div + p | Selects the first <p> element that are placed immediately after  elements |
| **General Sibling Selector** element ~ element | div ~ p | Selects every <p> element that are placed after  element |

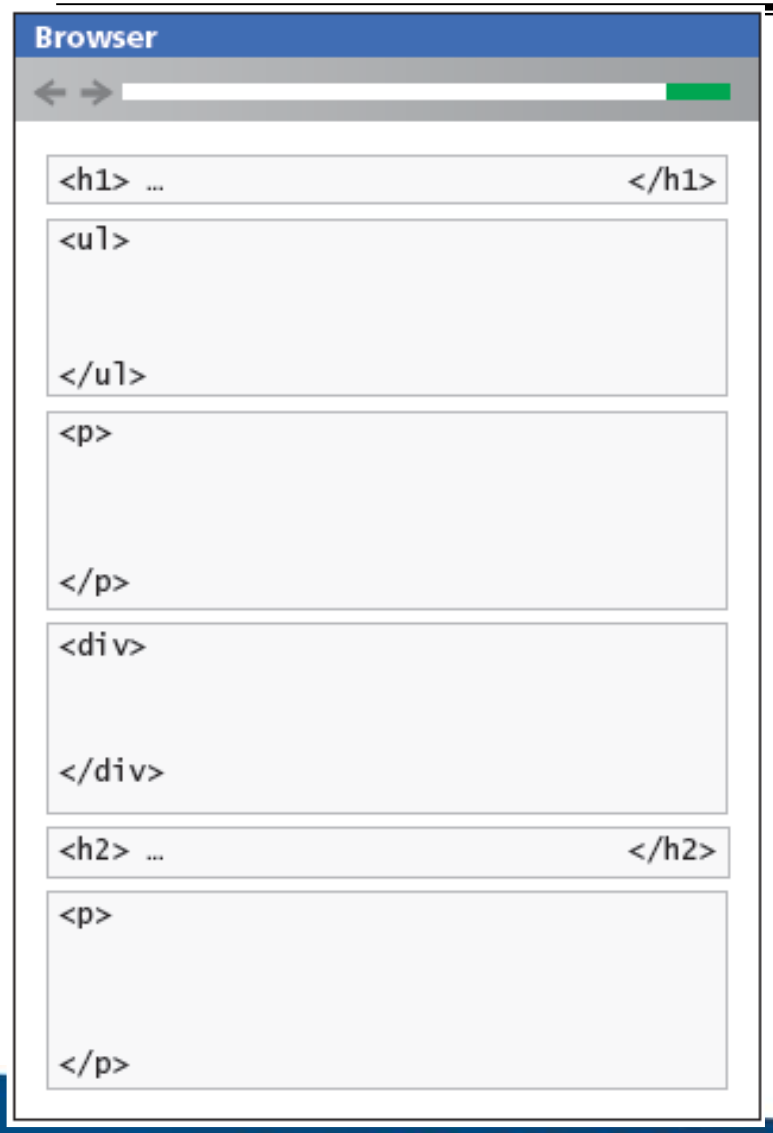
# Advanced CSS: Layout

# Normal Flow

The browser will normally display block-level elements and inline elements from left to right and from top to bottom.

- **Block-level elements** such as <p>, , <h2>, <ul>, and <table> are elements that are contained on their own line, because block-level elements begin with a line break (new line).
- Two block-level elements will not exist on the same line, without styling.
- **Inline elements** such as <b>, <u>, <sub>, <sup>, <img>, <i> etc. are displayed within the same line and do not form their own blocks.

# Block-level elements

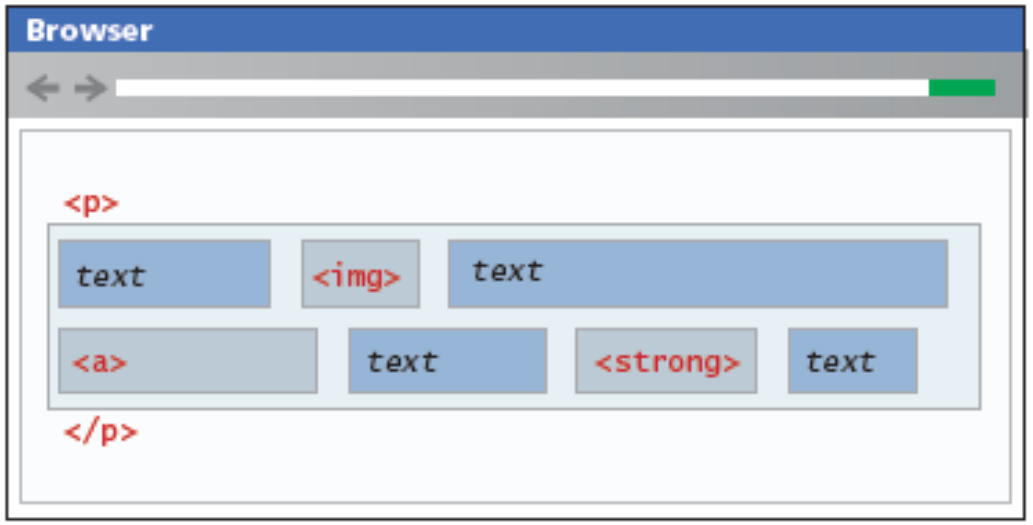



Properties of block-level elements -

- Each block exists on its own line.
- It is displayed in normal flow from the browser window's top to its bottom.
- By default each block level element fills up the entire width of its parent (browser window).
- CSS box model properties can be used to customize, for instance, the width of the box and the margin space between other block level elements.

# Inline elements

Inline elements line up next to one another horizontally from left to right on the same line, when there is no enough space on the line, the content moves to a new line. Example - <b>, <i>, <img>, <u>,<a> etc.




Properties of inline elements are –

- Inline element is displayed in normal flow from its container's left to right.
- When a line is filled with content, the next line will receive the remaining content, and so on.
- If the browser window resizes, then inline content will be "re-flowed" based on the new width.

# Types of inline elements

- **Replaced** and **Nonreplaced** inline elements
- **Replaced inline elements** are elements whose content and appearance is defined by some external resource, such as <img> and the various form elements.
- **Nonreplaced inline elements** are those elements whose content is defined within the document. Eg: <a>,<b>,<i>,.

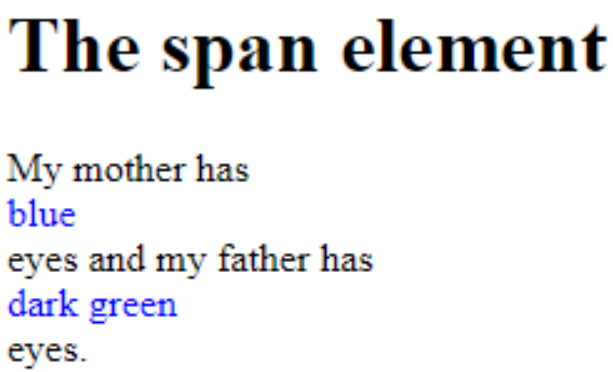
Note: A block-level or inline element is converted to another by using the CSS ‘display’ property.

span { display: block; }

li { display: inline; }

- These two rules will make all  elements behave like block-level elements and all <li> elements like inline (that is, each list item will be displayed on the same line).

```
<!DOCTYPE html>
<html>
<head>
<style>
span{color:blue; display:block;}
</style>
</head>
<body>

<h1>The span element</h1>
<div>
<p>My mother has <span >blue</span> eyes and
my father has <span >dark green</span> eyes.</p>
</div>
</body>
</html>
```



# Positioning Elements

The position property of CSS is used to move an item from its regular position in the normal flow. An element can also be fixed to a position, so that it is always visible while the rest of the content scrolls.

- The possible values for position property are-

| Type | Description |
|---|---|
| **relative** | The element is moved relative to where it would be in the normal flow. |
| **absolute** | The element is removed from normal flow and positioned in relation to its nearest positioned ancestor. |
| **fixed** | The element is fixed in a specific position in the window even when the document is scrolled |
| **static** | The element is positioned according to the normal flow. This is the default. |

- The left, right, top, and bottom properties are used to indicate the distance the element will move.

# Relative Positioning

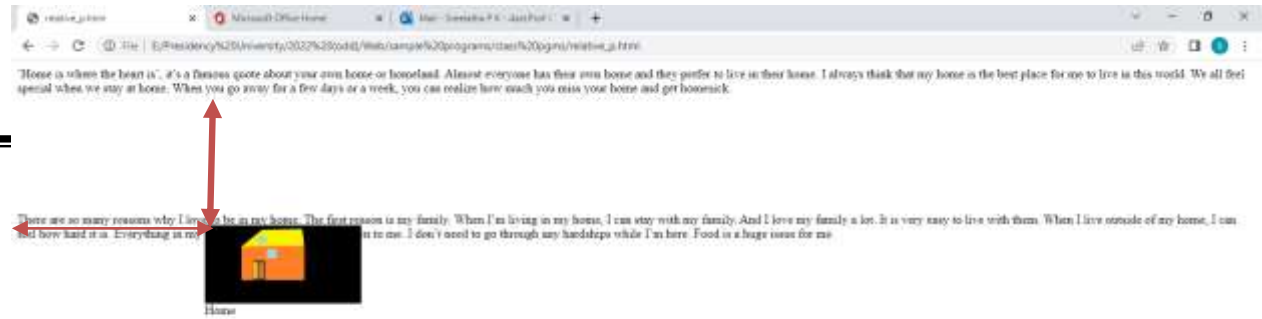
In **relative positioning** an element is displaced out of its normal flow position and moved relative to where it would have been placed normally. The other contents around the relatively positioned element remain in its old position in the flow. the space the element would have occupied is preserved.

Eg :

```
figure {
position: relative;
top: 150px;
left: 200px;
}
```

The contents of block tag (figure) has to be placed at 150px,200px from its actual position.

```
<html>
<head>
<style>
figure {
position: relative;
top: 150px;
left: 200px;
}
</style>
</head>
<body>
<p>‘Home is where the heart is’, it’s a famous quote about your own home or homeland. Almost everyone has their own home and they prefer to live in their home. I always think that my home is the best place for me to live in this world. We all feel special when we stay at home. When you go away for a few days or a week, you can realize how much you miss your home and get homesick. </p>
<figure>
<img src="HOUSE.png" width="200" height="100" />
<figcaption>Home</figcaption>
</figure>
<p>There are so many reasons why I love to be in my home. The first reason is my family. When I’m living in my home, I can stay with my family. And I love my family a lot. It is very easy to live with them. When I live outside of my home, I can feel how hard it is. Everything in my home is very familiar and known to me. I don’t need to go through any hardships while I’m here. Food is a huge issue for me.
</p>
 </body>
</html>
```



# Absolute Positioning

In **absolute positioning** an element is completely removed from normal flow. Here, space is not left for the moved element, as it is no longer in the normal flow. Its position is moved in relation to its container block.

Eg –

```
figure {
position: absolute;
top: 60px;
left: 200px;
}
```

With this positioning, the figure tag is placed at a distance of 60px from top and 200 px from left with respect to its container block. (Update this property in the above code, to view the difference).

# Fixed Positioning

The element is positioned in relation to the viewport (i.e., to the browser window). Elements with **fixed positioning** do not move when the user scrolls up or down the page.

The fixed position is used to ensure that navigation elements or **advertisements are always visible.**

Eg –

```
figure {
position: fixed;
top: 0px;
left: 0px;
}
```

With this positioning, the figure tag is placed at the top left most corner. (Update this property in the above code, to view the difference).

# Z-index

Each positioned element has a stacking order defined by the z-index property (named for the z-axis). Items closest to the viewer (and thus on the top) have a larger **z-index** value, as shown in the example below.

```
<head>
	<style>
figure {
position: absolute;
top: 60px;
left: 200px;
z-index:-1;
}

body{
z-index:1:
}
	</style>
</head>
```

# Floating Elements

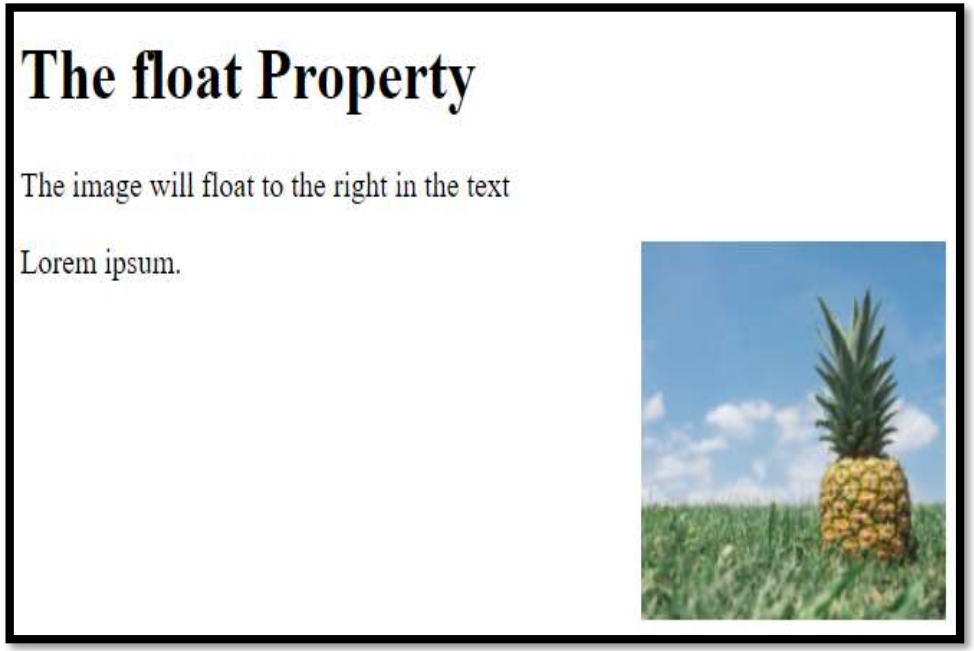
CSS float property is to displace an element out of its position in the normal flow.

When an item is floated, it is moved all the way to the far left or far right of its containing block and the rest of the content is “re-flowed” around the floated element.

Note: Absolutely positioned **elements** ignore the **float property**!

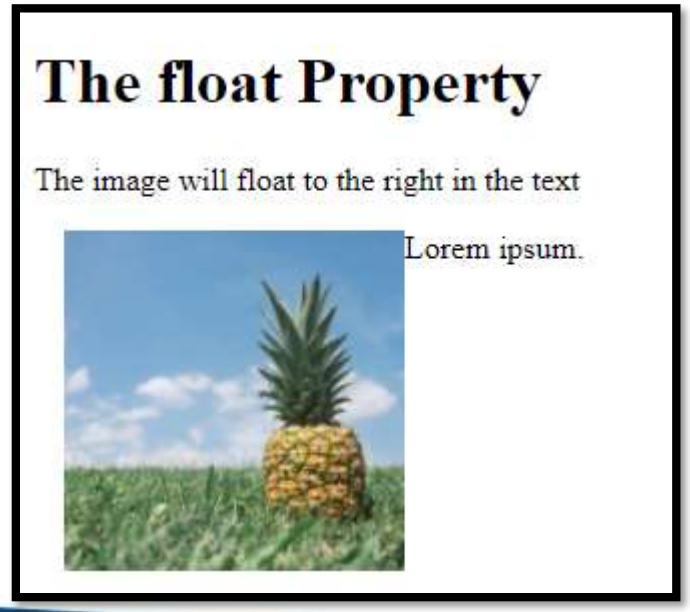
# Float

```
<head>
<style>
img {
  float: right;
}
</style>
</head>
<body>
<h1>The float Property</h1>
<p>The image will float to the right in the text</p>
<p><img src="pineapple.jpg" alt="Pineapple">
Lorem ipsum.</p>
</body>
</html>
```

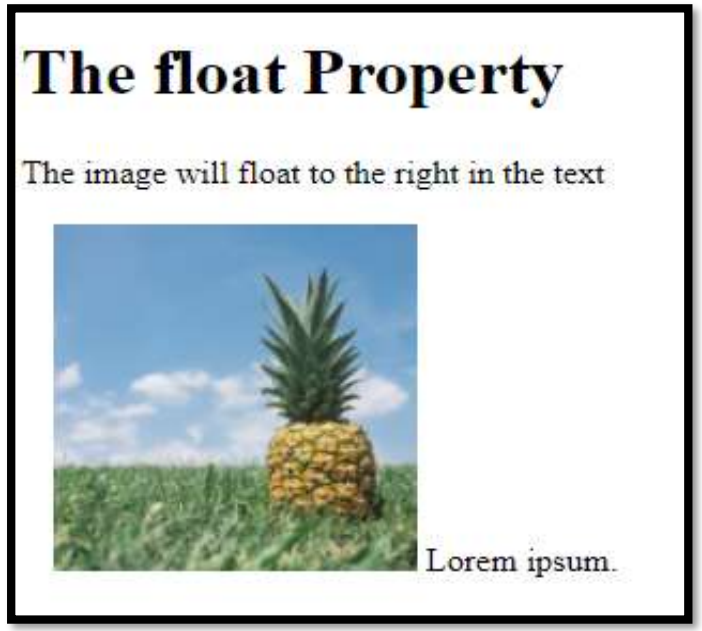


# Cont’d

```
<style>
img {
  float: left;
}
</style>
```



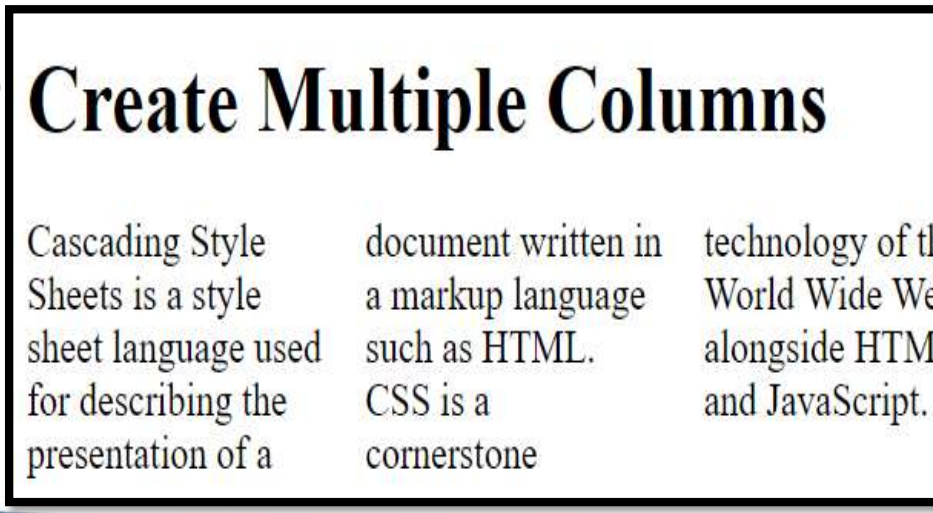

```
<style>
img {
  float: none;
}
</style>
```



# Constructing Multicolumn Layouts

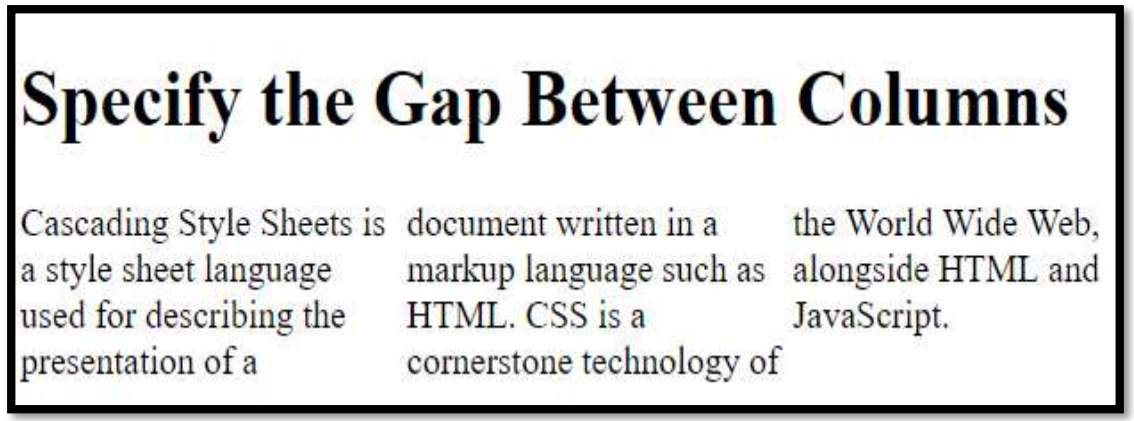
The space is divided into number of columns as mentioned by **column-count** property.

```
<html>
<head>
<style>
.newspaper {
  column-count: 3;
}
</style>
</head>
<body>
<h1>Create Multiple Columns</h1>
<div class="newspaper">
Cascading Style Sheets is a style sheet language used for describing the presentation of a document written in a markup language such as HTML. CSS is a cornerstone technology of the World Wide Web alongside HTML and JavaScript.
</div>
</body>
</html>
```

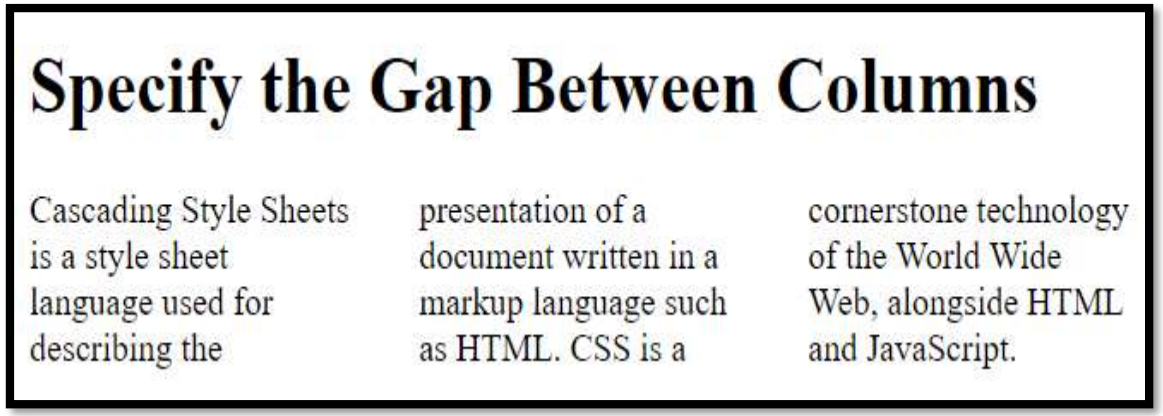


# Column-gap

```
<style>
.newspaper {
  column-count: 3;
  column-gap: 5px;
}
</style>
```

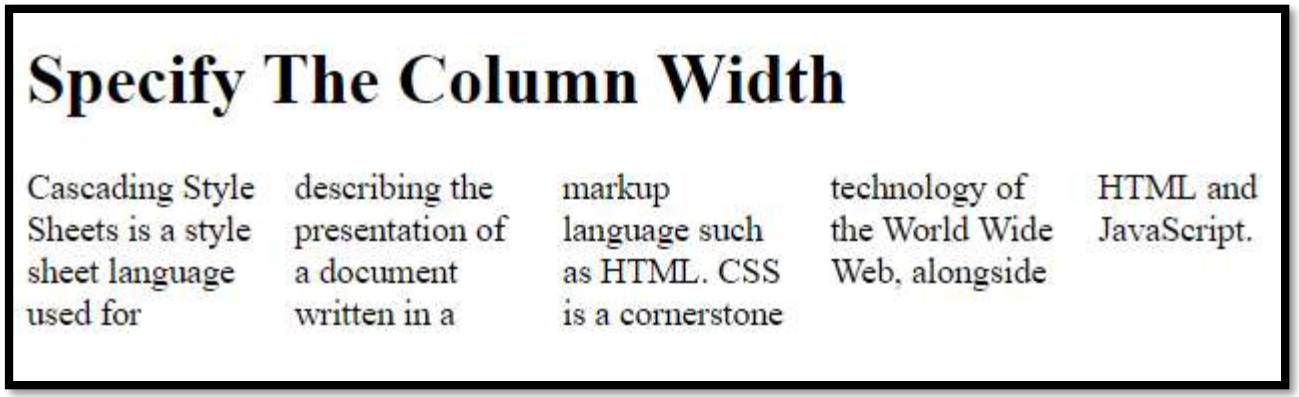



```
<style>
.newspaper {
  column-count: 3;
  column-gap: 30px;
}
</style>
```



# Column width

```
<style>
.newspaper {
  column-width: 100px;
}
```



# Approaches to CSS Layout

- The size of the screen used to view the page can vary
- Like some users will visit a website on a 21-inch wide screen, while few others on 120 inches screen.
- Users with the large monitor might expect a site to take advantage of the extra size; users with the small monitor will expect the site to scale to the smaller size and still be usable (clear).
- This problem can be dealt in two basic ways - **Fixed Layout and Liquid Layout.**

# Fixed Layout

Ideal width is fixed by the designer.

A common width, that fits normal desktop monitor (1024 × 768), is considered.

This content may be positioned on the left or the center of the monitor.



The advantage of a fixed layout –

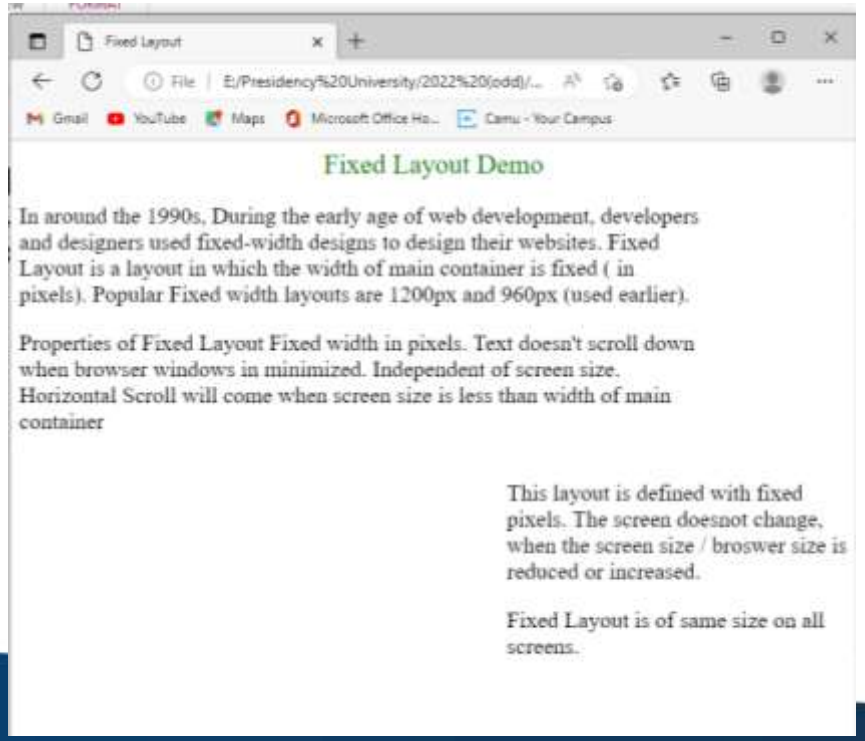
- easy to produce
- predictable visual result
- optimized for typical desktop monitors

The disadvantage of a fixed layout –

- For larger screens, there may be an excessive amount of blank space to the left and/or right of the content.
- When the browser window is less than the fixed width; the user will have to horizontally scroll to see all the content.
- If smaller mobile devices are used, more horizontal scrolling has to be done.

```
<!DOCTYPE html>
<head>
        <style>
                div#left{
                        width: 600px;
                        float: left;
                        font-size: 20px;
                }
                div#right{
                        width: 300px;
                        float: right;
                        font-size: 20px;
                }
        </style>
        <title>Fixed Layout</title>
</head>
<body>
        <div style="text-align: center; color:green; font-size:x-large">
                Fixed Layout Demo
        </div>

        <div id="left">
        <p>In around the 1990s, During the early age of web development, developers and designers used fixed-width designs to design their
        websites. Fixed Layout is a layout in which the width of main container is fixed ( in pixels). Popular Fixed width layouts are 1200px and
        960px (used earlier).
        </p>
        <p>Properties of Fixed Layout Fixed width in pixels. Text doesn't scroll down when browser windows in minimized. Independent of screen
        size. Horizontal Scroll will come when screen size is less than width of main container
        </p>
        </div>
```

```
<div id="right">
<p>This layout is defined with fixed pixels.
The screen doesnot change, when the
screen size / broswer size is reduced or increased.
</p>

<p> Fixed Layout is of same size on all screens.
</p>
</div>


</body>

</html>
```

# Liquid Layout

widths are not specified using pixels, but percentage values

As widths are expressed as percentages, the webpage will adapt to any browser size.

Eg: width: 50%;


Fluid layouts are based on the browser window.


However, elements can get too spread out as browser expands.

The advantage of a liquid layout –

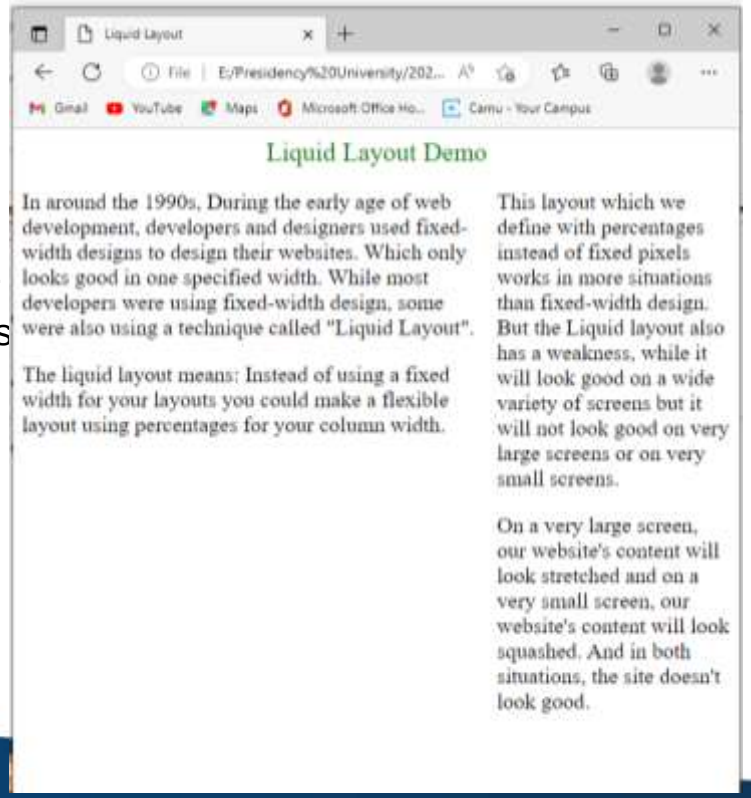
- Adapts to different browser sizes, so there is neither wasted white space nor any need for horizontal scrolling.

The disadvantage of a liquid layout –

- more difficult to create because some elements, such as images, have fixed pixel sizes.
- The screen may grow or shrink dramatically.

```
<!DOCTYPE html>
<head>
	<style>
		div#left{
			width: 66%;
			float: left;
			font-size: 20px;
		}
		div#right{
			width: 33%;
			float: right;
			font-size: 20px;
		}
	</style>
	<title>Liquid Layout</title>
</head>

<body>
	<div style="text-align: center; color:green; font-size:x-large">
		Liquid Layout Demo
	</div>

	<div id="left">
	<p>In around the 1990s, During the early age of web development, developers and designers used fixed-width designs
	to design their websites.  Which only looks good in one specified width. While most developers were using fixed-width
	design, some were also using a technique called "Liquid Layout".
	</p>
```

```
<p>The liquid layout means:
Instead of using a fixed width for your layouts
you could make a flexible layout using percentage
for your column width.
</p>
</div>

<div id="right">
<p>This layout which we define with percentages
 instead of fixed pixels works in more situations
 than fixed-width design. But the Liquid layout
also has a weakness, while it
will look good on a wide variety of screens but it
will not look good on very large screens or on very
small screens.
</p>

<p>On a very large screen, our website's content
 will look stretched and on a very small screen, our
 website's content will look squashed. And in both s
the site doesn't look good.
</p>
</div>

</body>

</html>
```

# Responsive Design

- Makes your web page look good on all devices, using HTML and CSS.
- Web pages should not leave out information to fit smaller devices, but rather adapt its content to fit any device:



The four key components that make responsive design work are -

- Liquid layouts
- Scaling images to the viewport size
- Setting viewports via the <meta> tag
- Customizing the CSS for different viewports using media queries

# Scaling images to the viewport size

- The viewport is the user's visible area of a web page.
- The viewport varies with the device, and will be smaller on a mobile phone than on a computer screen.

<meta name="viewport" content="width=device-width" />

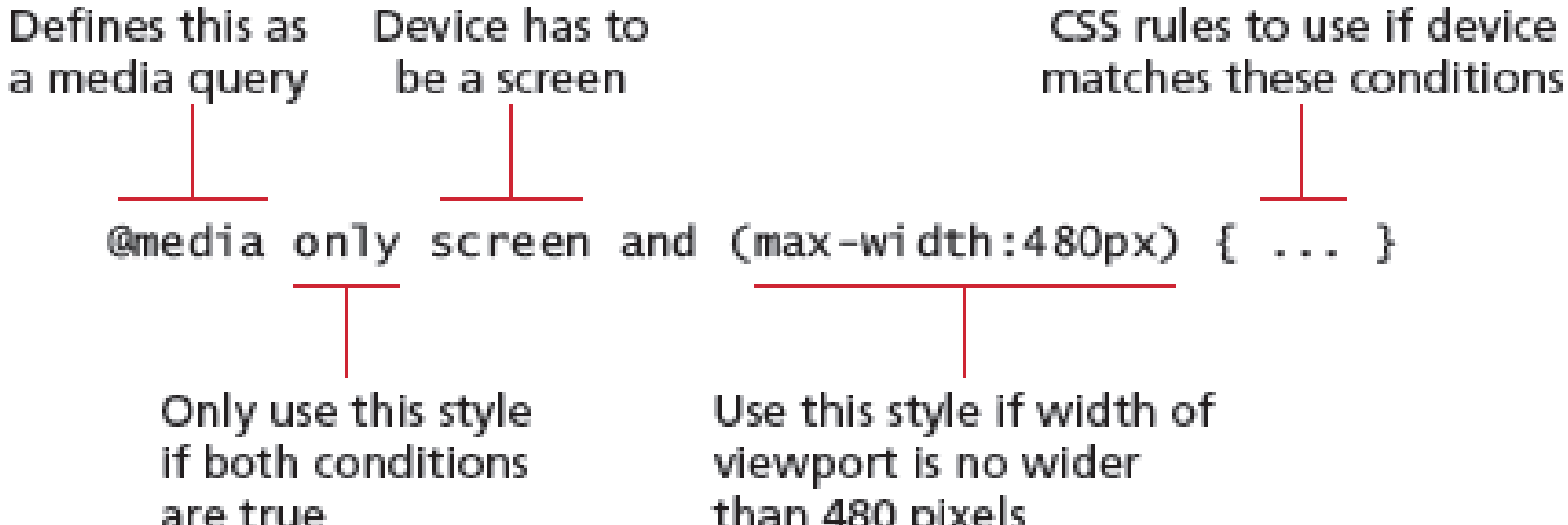
This makes the viewport as many pixels wide as the device screen width. This means that if the device has a screen that is 320 px wide, the viewport width will be 320 px; if the screen is 480 px, then the viewport width will be 480 px.

# Media Queries

- A media query is a way to apply style rules based on the medium that is displaying the file.
- It uses the @media rule to include a block of CSS properties only if a certain condition is true.
- Use these queries to look at the capabilities of the device, and then define CSS rules.

Example-

@media only screen and (max-width: 480px) {……} //This set of rule is applied when smaller screen is used. Like set font-size, left and right margin etc.



# CSS Frameworks

A **CSS framework** is a pre-created set of CSS classes or other software tools that make it easier to use and work with CSS.

They are two main types of CSS framework: grid systems and CSS preprocessors.

i) Grid Systems – Grids are used to achieve visual uniformity in a design. The screen is virtually divided into 5- or 7- or 12-column grid. Then, the text or graphics of the document is aligned and sized according to the grid.

ii) CSS Preprocessor – this is a tool that that takes code written in some type of preprocessed language and then converts that code into normal CSS. The preprocessed language uses programming identities such as variables, inheritance, calculations, and functions. Ex - LESS, SASS, and Stylus.