# UI DESIGN

## HTML5 Style Guide and Coding Convention

- There are some guidelines and tips to create a good HTML code.
- A consistent, clean, and tidy HTML code makes it easier for others to read and understand the code.

### Declaring Document Type

- Document type is declared as the first line of the document.
  Ex: `<!doctype html>`

### Use Lower Case Element Names

- HTML5 allows mixing uppercase and lowercase letters in element names, but as a common practice using lowercase element names is preferred.
- Mixing both uppercase and lowercase names is not considered as a good practice.
- Developers are used to using lowercase names.
- Lowercase look cleaner, and are easier to write
  Ex:

```
<section>
  <p>This is a paragraph.</p>
</section>
```

### Close All HTML Elements

- In HTML, there are some elements which need not be closed Ex: <p> element.
- But it is strongly recommend closing all HTML elements.

Ex:

| Instead of giving like this | The code can be written with proper closing tags |
|---|---|
| `<section>`<br>`  <p>This is a paragraph.`<br>`  <p>This is a paragraph.`<br>`</section>` | `<section>`<br>`  <p>This is a paragraph.</p>`<br>`  <p>This is a paragraph.</p>`<br>`</section>` |

### Use Lowercase Attribute Names

- HTML5 allows mixing uppercase and lowercase letters in element names, but as a common practice using lowercase element names is preferred.
- Mixing both uppercase and lowercase names is not considered as a good practice.
- Developers are used to using lowercase names.
- Lowercase look cleaner, and are easier to write

Instead of writing a code like this

```
<a HREF="https://www.w3schools.com/html/">Visit our HTML tutorial</a>
```

Its better to write the above code in lower case

```
<a href="https://www.w3schools.com/html/">Visit our HTML tutorial</a>
```

### Close Empty HTML Elements

- In HTML5, it is optional to close empty elements. We can write an empty tag like this

```
<meta charset="utf-8">
```

- But it is good practice to close an empty tag also. Closing tag will be helpful if our page is accessed by XML.
- The slash (/) is to close a empty tag, this is required in XHTML and XML.
- An empty tag can be closed as shown below

```
<meta charset="utf-8" />
```

## Quote Attribute Values

Though HTML5 allows attribute values without quotes, but its recommend quoting attribute values:

- We have to use quotes if the value contains spaces
- Quoted values are easier to read

This will not work, because the value contains spaces:

Ex: `<table class="striped">`

## Image Attributes

- Always specify the alt attribute for images. This attribute is important if the image for some reason cannot be displayed.
- It is also necessary to define the width and height of images. This reduces flickering, because the browser can reserve space for the image before loading.

Ex:

```
<img src="html5.gif" alt="HTML5" style="width:128px;height:128px">
```

## Spaces and Equal Signs

- HTML allows spaces around equal signs. But space-less is easier to read and groups entities better together.

Ex: `<link rel="stylesheet" href="styles.css">`

## Avoid Long Code Lines

- Long code line makes to scroll right and left to read, its not convenient to read.
- It is better to avoid too long code lines.

## Blank Lines and Indentation

- Do not add blank lines, spaces, or indentations without a reason.
- For readability, add blank lines to separate large or logical code blocks.
- Add two spaces of indentation. Do not use the tab key.

Ex:

```
<ul>
  <li>London</li>
  <li>Paris</li>
  <li>Tokyo</li>
</ul>
```

## The <title> Element

- The <title> element is required in HTML.
- The contents of a page title is very important for search engine optimization (SEO).
- The page title is used by search engine algorithms to decide the order when listing pages in search results.
- The <title> element:
  - defines a title in the browser toolbar
  - provides a title for the page when it is added to favorites
  - displays a title for the page in search-engine results

Ex: `<title>HTML Style Guide and Coding Conventions</title>`

## <html> and <body> Elements

- An HTML page will validate without the <html> and <body> tags
- Omitting <body> can produce errors in older browsers.
- Omitting <html> and <body> can also crash DOM and XML software.

Ex:

```
<!DOCTYPE html>
<head>
  <title>Page Title</title>
</head>
```

## <head> Element

- The HTML <head> tag can also be omitted, however, it is recommend using the <head> tag.
- If omitted Browsers will add all elements before <body>, to a default <head> element.

Ex:

```
<!DOCTYPE html>
<html>
<title>Page Title</title>
<body>

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

</body>
</html>
```

## The lang Attribute and Meta data

- We should always include the lang attribute inside the <html> tag, to declare the language of the Web page.
- This is meant to assist search engines and browsers.
- To ensure proper interpretation and correct search engine indexing, both the language and the character encoding should be defined.

Ex:

```
<!DOCTYPE html>
<html lang="en-us">
```

```
<head>
  <meta charset="UTF-8">
  <title>Page Title</title>
</head>
```

## HTML Comments

- Short comments should be written on one line, like this:

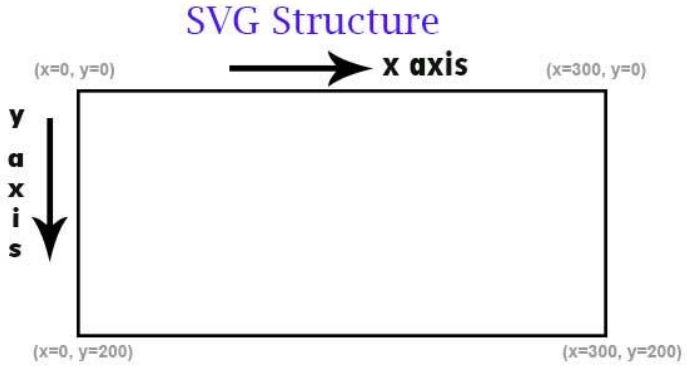
```
<!-- This is a comment -->
```

- Long comments are easier to observe if they are indented with two spaces.
- Comments that spans more than one line, should be written like this

```
<!--
  This is a long comment example. This is a long comment example.
  This is a long comment example. This is a long comment example.
-->
```

# Html SVG

- SVG stands for Scalable Vector Graphics
- It is a language for describing 2D-graphics and graphical applications in XML and the XML is then rendered by an SVG viewer.
- SVG is mostly useful for vector type diagrams like Pie charts, Two-dimensional graphs in an X,Y coordinate system etc.
- SVG can create Vector based drawing and objects like lines, rectangle, circle, polygons, text so on.




- Inside svg element, child tag of svg like rect, circle, polygon, text, g (group), ellipse are created.
- Default width of svg element is 300 and default height is 150.
- We can change width and height of svg element using width and height attributes.

### Embedding SVG in HTML5

- HTML5 allows embedding SVG directly using **<svg>...</svg>** tag which has following simple syntax
- SVG has several methods for drawing paths, boxes, circles, text, and graphic images.

## SVG Line

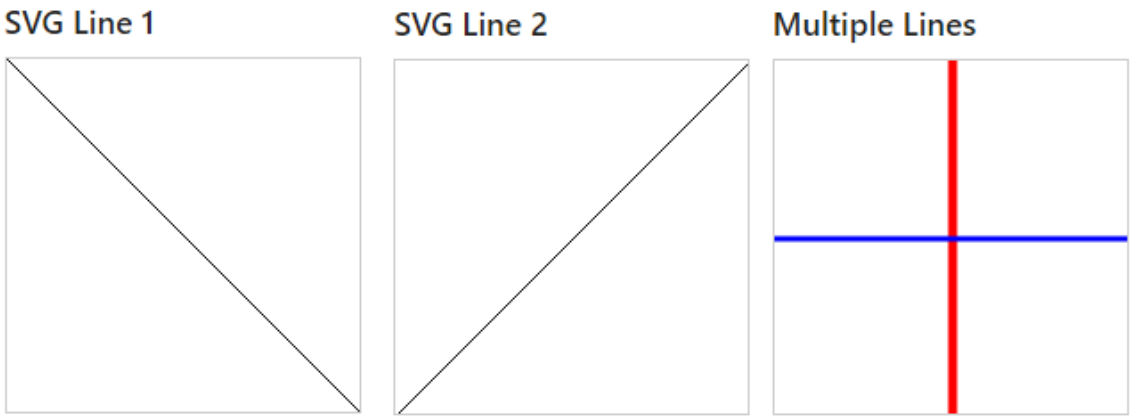
- A line can have stroke and stroke-width, but can't be filled as it is not a shape like rectangle and circle.

Ex:

```
<svg width="200" height="200" style="border:1px solid #ccc">
  <line x1="0" y1="0" x2="200" y2="200" > </line>
</svg>

<svg width="200" height="200" style="border:1px solid #ccc">
  <line x1="200" y1="0" x2="0" y2="200" stroke="black"> </line>
</svg>

<svg width="200" height="200" style="border:1px solid #ccc">
        <line x1="100" y1="0" x2="100" y2="200" stroke="red" stroke-width="5"> </line>
        <line x1="0" y1="100" x2="200" y2="100" stroke="blue" stroke-width="3"></line>
</svg>
```



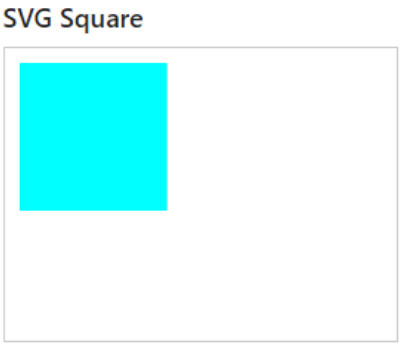

## SVG Rectangle

**SVG Rect** can create rectangle inside an SVG tag.

- A rectangle can have x ( horizontal distance from left-top corner ), y ( vertical distance from left-top corner ), width and height.
- Default fill ( *background-color*) of rectangle is black.

Ex:

```
<svg width="200" height="200" style="border:1px solid #ccc>
        <rect x="10" y="10" width="100" height="100" fill="aqua" ></rect>
</svg>
```




## HTML5 – SVG Circle

- SVG Circle can create circle inside an SVG tag. Circle tag can have cx ( center from x), cy ( center from y) and r ( radius).

Ex:

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

<svg width="100" height="100">
  <circle cx="50" cy="50" r="40" stroke="green" stroke-
width="4" fill="yellow" />
</svg>

</body>
</html>
```

## SVG Text

- SVG Text is used using <text> tag in svg. Default position for text is x="0" and y="0"

Ex:

```
<svg>
    <text x="100" y="120" fill="red" font-size="20px">This is text 2</text>
</svg>
```

**Differences between SVG and Canvas**

| SVG | Canvas |
|---|---|
| Vector based (composed of shapes) | Raster based (composed of pixel) |
| Multiple graphical elements, which become the part of the page's DOM tree | Single element similar to <img> in behavior. Canvas diagram can be saved to PNG or JPG format |
| Modified through script and CSS | Modified through script only |
| Good text rendering capabilities | Poor text rendering capabilities |

| | |
|---|---|
| Give better performance with smaller number of objects or larger surface, or both | Give better performance with larger number of objects or smaller surface, or both |
| Better scalability. Can be printed with high quality at any resolution. Pixelation does not occur | Poor scalability. Not suitable for printing on higher resolution. Pixelation may occur |

## Canvas

- The HTML <canvas> element is used to draw graphics, via JavaScript.
- The <canvas> element is only a container for graphics. We must use JavaScript to draw the graphics.
- Canvas has several methods for drawing paths, boxes, circles, text, and adding images.
- A canvas is a rectangular area on an HTML page. By default, a canvas has no border and no content.

The markup looks like this:

```
<canvas id="myCanvas" width="200" height="100"
      style="border:1px solid gray;">
</canvas>
```

## Canvas getContext

- To build content inside canvas we have to use JavaScript.
- We need to use getContext("2d") function of Canvas Element to put context put elements in the Canvas.

Ex:

```
var c = document.getElementById("myCanvas");
var cctx = c.getContext("2d");
```

## Drawing Line on Canvas

- To draw a straight line on the canvas, we can use the following two methods.
    - moveTo(x,y) - to define the starting point of the line.
    - lineTo(x,y) - to define the ending point of the line.

Ex:

```
<canvas id="myCanvasLine" width="200" height="100" style="border:1px solid gray">
</canvas>
<script>
        var c = document.getElementById("myCanvasLine");
        var ctx = c.getContext("2d");
        ctx.moveTo(0,0);
        ctx.lineTo(200,100);
        ctx.stroke();
</script>
```

## Drawing rectangle on Canvas

- To draw a rectangle we have to use fillRect() function.
- It uses four coordinates (0,0) first two defines the upper left corner on the canvas.
- The last two defines the lower right corner on the canvas.

Ex:

```
<canvas id="myCanvas" width="200" height="200" style="border:1px solid gray">
</canvas>
<script>
 var c = document.getElementById("myCanvas");
 var ctx = c.getContext("2d");
 ctx.fillStyle = "blue";
 ctx.fillRect(10,10,100,100);
</script>
```

## Drawing Circle on Canvas

- To draw a circle on the canvas, the arc() method is used

arc(x, y, r, start, stop)

Ex:

```
<canvas id="myCanvas" width="200" height="200" style="border:1px solid gray">
</canvas>
<script>
 var c = document.getElementById("myCanvas");
 var ctx = c.getContext("2d");
 ctx.beginPath();
 ctx.arc(100,100,50,0,2*Math.PI);
 ctx.stroke();
 ctx.fillStyle = "red";
 ctx.fill();
</script>
```

## Drawing text on canvas

- The following methods and properties are used to draw text on the canvas.
- font property - used to define the font property for the text.
- fillText(text,x,y) method - used to draw filled text on the canvas.
- strokeText(text,x,y) method - draw unfilled text on the canvas.

Ex:

```
<canvas id="myCanvas" width="200" height="200" style="border:1px solid gray">
</canvas>
<script>
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.font = "30px Arial";
ctx.fillStyle = "red";
ctx.fillText("Hello Hinduthan",10,50);
</script>
```

# Html API's

## What is Web API?

- API stands for **A**pplication **P**rogramming **I**nterface.
- A Web API is an application programming interface for the Web.
- A Browser API can extend the functionality of a web browser.
- A Server API can extend the functionality of a web server.
- In a typical HTML API, the calls and requests along with definitions and protocols are written and invoked with HTML itself.
- HTML API uses certain class or attribute patterns used in the HTML element to invoke the functions in the API.

## Browser APIs

- All browsers have a set of built-in Web APIs to support complex operations, and to help accessing data.
- For example, the Geolocation API can return the coordinates of where the browser is located.

## Third Party APIs

Third party APIs are not built into your browser.
To use these APIs, you will have to download the code from the Web.
Examples:

- YouTube API - Allows you to display videos on a web site.
- Twitter API - Allows you to display Tweets on a web site.
- Facebook API - Allows you to display Facebook info on a web site.

# Audio & Video

- HTML5 supports <audio> tag which is used to embed sound content in an HTML or XHTML document as follows.

# Drag/Drop

- The Drag and Drop Browser API enables to Drag an object on the webpage and Drop it at an another destination on the same webpage.
- This feature was first implemented on Internet Explorer in 1999, but now all browser supports this feature:
- A Simple Drag and Drop operation can be performed in four simple steps as follows.
    - Create a draggable element, by setting the draggable attribute to "true"
    - Handle the dragged object using drag events.
    - Receive the Dropped item at the Drop Zone.
    - Complete the Drag-Drop operation by adding the dragged items to destination's DOM.

## *Creating a Draggable element*

- We first let the browser know which elements can be dragged, by defining the draggable attribute. It can assume three values.
    - "true": Specifies that elements can be dragged.
    - "false": Specifies that the element cannot be dragged.

- "auto": Leaves it to the browser to decide if the element can be dragged. Its the **default** value.

**Ex:**

```
<body>
  <div id="Drag_me">
  <img draggable="true" id="facebook" src="facebook.png" alt="facebook"/>
  </div>
</body>
```

## *Handling the Dragged Element*

- The dragged elements need to be handled with a series of event handlers.
- We set the ondragstart event handler on the elements that are supposed to be dragged.
- The Drag-Drop API has many attributes and methods, the one useful in this case is setData(format,Data)
  - agrument format : Can take two values **Text** or **Url**.
  - Data :Denotes the data to be transferred.

## *Recieving the drop*

- The dragged element needs to find the destination where it can be dropped.
- For an element to act as a destination for drop, it must **cancel** the default action and capture the ondragOver event .

## *Complete the Drop Operation*

- Next, within the target element include the ondrop attribute with drop event.
- This function removes the dragged item from the source list and adds it to the destination element's DOM. Using the standard appendChild method.

```
<html>
<body>

<script>
function allowDrop(ev) {ev.preventDefault();}

function drag(ev) {ev.dataTransfer.setData("text", ev.target.id);}

function drop(ev) {
ev.preventDefault();
var data = ev.dataTransfer.getData("text");
ev.target.appendChild(document.getElementById(data));
}
</script>

<p>Drag the image into the rectangle:</p>

<div id="div1" style="width:350px;height:200px;padding:10px;border:1px solid #aaaaaa;"
```

```
ondrop="drop(event)" ondragover="allowDrop(event)">
<img id="drag1" src="facebook.png" alt="FaceBook image"
draggable="true" ondragstart="drag(event)"/>

</div>
<br>

<div id="div2" style="width:350px;height:200px;padding:10px;border:1px solid #aaaaaa;"
ondrop="drop(event)" ondragover="allowDrop(event)"></div>

</body>
</html>
```

## Local Storage

- This feature allows web applications to locally store data within the browser on the client side.
- It stores data in the form of key/value pair on the browser.
- This is also known as DOM storage.
- Storing data with the help of web storage is similar to cookies, but it is better and faster than cookies storage.

In compared to cookies Web Storage has Following Advantages:

- Web Storage can use storage space upto 5MB per domain.
- It will not send data to the server side, hence it is faster than cookies storage.
- The data stored by local Storage never expires, but cookies data expires after some time or session.
- Web Storage is more secure than cookies.

### Types of Web Storage

There are two types of web storage with different scope and lifetime.

- **Local Storage**
  - Uses Windows.localStorage object which stores data and available for every page.
  - Data is persist even if the browser is closed and reopened.
- **Session Storage**
  - Uses Windows.sessionStorage object which stores data for one session
  - Data will be lost if the window or browser tab will be closed.

### The localStorage Object

- The localStorage object stores data locally within the browser.
- The data stored by localStroage object does not have any expiration date.
- Hence the stored data will not be deleted if the browser is closed or reopened.
- Each piece of data is stored in simple key-value pairs.
- Can be accessed with **localStorage.getItem()** and **localStorage.setItem()** methods.

### The sessionStorage Object

- The sessionStorage object is same as the localStorage object, but the difference is that it stores data only for one session.

- If the browser is closed, then data will be lost or deleted.

## Remove Web Storage

- Local storage will remain in the browser even if we close it.
- Hence to delete the local storage data, two methods are used
  - **localStorage.removeItem('key'):** to delete the value on a particular key.
  - **localStorage.clear():** to delete or clear all settings with key/value pair.

```
<!DOCTYPE html>

<html>
<!DOCTYPE html>
<html>
<head>
 <title>Web Local Storage API</title>
 <style>
  body{
   color: green;
   text-align: center;
   font-size: 30px;
   margin-top: 30px;
   font-style: italic;
  }
 </style>
</head>
<body>

<script>
 if(typeof(Storage)!=="undefined") {
  localStorage.setItem("name","Kumar");
  localStorage.setItem("Country", "India");
   document.write("Hi"+" "+localStorage.name+" "+"from" +" "+ localStorage.Country);
}
 else{
  alert("Sorry! your browser is not supporting the browser")
 }
</script>
```

```
<!DOCTYPE HTML>

<html>
   <body>
      <script type = "text/javascript">

         if( localStorage.hits ) {
            localStorage.hits = Number(localStorage.hits) +1;
         } else {
```

```
            localStorage.hits = 1;
         }
         document.write("Total Hits :" + localStorage.hits );
      </script>

      <p>Refresh the page to increase number of hits.</p>
      <p>Close the window and open it again and check the result.</p>

   </body>
</html>
```

```
<!DOCTYPE html>
<html>
<head>
<title>Web Storage API</title>
<style>
body{
color: green;
text-align: center;
font-size: 30px;
margin-top: 30px;
font-style: italic;
}
</style>
</head>
<body>
<button onclick="remove();">Remove item</button>
<div id="output"></div>

<script>
if(typeof(Storage)!=="undefined") {
localStorage.setItem("name","Harshita");
localStorage.setItem("Country", "India");
docu-
ment.getElementById('output').innerHTML= "Hii, my name is"+ " "+ localStorage.name +" "+"and
 i belongs to"+" "+localStorage.Country;
}
else{
alert("Sorry! your browser is not supporting the browser")
}
function remove() {
localStorage.removeItem("name");
docu-
ment.getElementById('output').innerHTML= "Hii, my name is"+ " "+ localStorage.name +" "+"and
 i belongs to"+" "+localStorage.Country;
}
</script>
</body>
</html>
```

# Web Socket API

- WebSockets is a bidirectional communication technology for web applications can be used via a JavaScript interface in HTML 5.
- Once we get a Web Socket connection with the web server, we can send and receive data from browser to server
- **send()** method to send data to server.
- To receive data from server to browser is handled by **onmessage** event handler.

## WebSocket Attributes

**Socket.readyState**

- The readonly attribute readyState represents the state of the connection.
- It can have the following four values
  - **0** - Connection has not yet been established.
  - **1** - Connection is established and communication is possible.
  - **2** - Connection is going through the closing handshake.
  - **3** - Connection has been closed or could not be opened.

**Socket.bufferedAmount**

- The readonly attribute bufferedAmount represents the number of bytes of UTF-8 text that have been queued using send() method.

## WebSocket Events

| Event | Event Handler | Description |
|---|---|---|
| **open** | Socket.onopen | This event occurs when socket connection is established. |
| **message** | Socket.onmessage | This event occurs when client receives data from server. |
| **error** | Socket.onerror | This event occurs when there is any error in communication. |
| **close** | Socket.onclose | This event occurs when connection is closed. |

## WebSocket Methods

Following are the methods associated with WebSocket object.

**Socket.send()** The send(data) method transmits data using the connection.

**Socket.close()** The close() method would be used to terminate any existing connection.

- Following is the API which creates a new WebSocket object.

```
Syntax: var Socket = new WebSocket(url, [protocal] );
Example:
            var ws = new WebSocket("mca://localhost:9998/echo");
            ws.send("Message to send");
            ws.onmessage = function (evt) {
                    var rece_msg = evt.data;
                    alert("Message is received..." + rece_msg);
                };
```

## Debugging and Validating Html

There are many tools available, that allows us you to validate the code on our sites. The most common three validators are listed below

- Validator.nu - It goes through the entire document to check whether markup follows the doctype correctly.
- The W3C MarkUp Validator - This looks at the (X)HTML doctype
- The W3C Link Checker - check, and tests all the links inside that document.
- The W3C CSS Validator - check that the CSS follows the CSS specs properly.

Some of the common things these validators perform are

- Clean up code
- You can automatically remove empty tags, combine nested font tags, and otherwise improve messy or unreadable HTML or XHTML code.
- Remove Empty Container Tags
- Remove Redundant Nested Tags
- Remove Non-Dreamweaver HTML Comments
- Combine Nested <font> Tags When Possible
- Check for balanced tags
- Check for balanced parentheses, braces, or square brackets

## Cascading Style Sheet (CSS3): The Need for CSS

- It is the coding language that gives a website its look and layout.
- Along with HTML, CSS is fundamental to web design.
- It is a highly effective HTML tool that provides easy control over layout and presentation of website pages by separating content from design.
- Before the development of CSS in 1996 by the World Wide Web Consortium (W3C), Web pages were extremely limited in both form and function.

- CSS allowed several innovations to webpage layout, such as the ability to:
    - Specify fonts other than the default for the browser
    - Specify color and size of text and links
    - Apply colors to backgrounds
    - Contain webpage elements in boxes and float those boxes to specific positions on the page
    - They put the "style" in style sheets, and Web pages could be designed.

Benefits of CSS in Web Development

**Improves Website Presentation**

- The standout advantage of CSS is the added design flexibility and interactivity it brings to web development.
- Developers have greater control over the layout allowing them to make precise section-wise changes.

**Makes Updates Easier and Smoother**

- CSS works by creating rules.

- These rules are simultaneously applied to multiple elements within the site.
- Eliminating the repetitive coding style of HTML makes development work faster and less monotonous.
- Errors are also reduced considerably.
- Since the content is completely separated from the design, changes across the website can be implemented all at once.
- This reduces delivery times and costs of future edits.

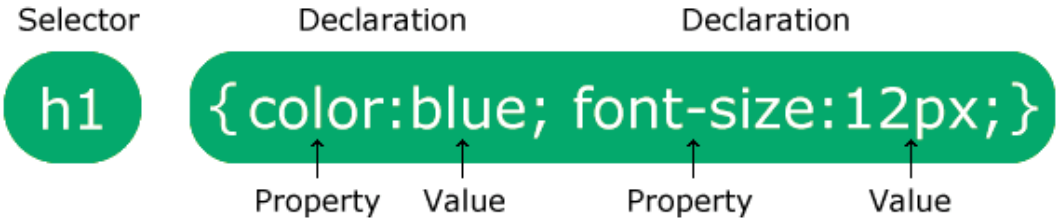
**Helps Web Pages Load Faster**

- Browsers download the CSS rules once and cache them for loading all the pages of a website.
- It makes browsing the website faster and enhances the overall user experience.
- This feature comes in handy in making websites work smoothly at lower internet speeds.
- Accessibility on low end devices also improves with better loading speeds.

## Basic Syntax and Structure Inline Styles

### *CSS Syntax*

A CSS rule consists of a selector and a declaration block.




- The selector points to the HTML element we 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**

In this example all <p> elements will be center-aligned, with a red text color

```
p {
  color: red;
  text-align: center;
}
```

- p is a selector in CSS.
- color is a property, and red is the property value
- text-align is a property, and center is the property value

## *CSS Selector*

- **CSS selectors** are used *to select the content we want to style.*
- Selectors are the part of CSS rule set.
- CSS selectors select HTML elements according to its id, class, type, attribute etc.

There are several different types of selectors in CSS.

1. CSS Element Selector
2. CSS Id Selector
3. CSS Class Selector
4. CSS Universal Selector
5. CSS Group Selector

### 1) CSS Element Selector

The element selector selects the HTML element by name.

```
<!DOCTYPE html>
<html>
<head>
<style>
        p{
           text-align: center;
           color: blue;
        }
</style>
</head>
<body>
  <p>This style will be applied on every paragraph.</p>
  <p id="para1">Me too!</p>
  <p>And me!</p>
</body>
</html>
```

### 2) CSS Id Selector

- The id selector selects the id attribute of an HTML element to select a specific element.
- An id is always unique within the page so it is chosen to select a single, unique element.
- It is written with the 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 Javatpoint.com</p>
     <p>This paragraph will not be affected.</p>
    </body>
    </html>
```

### 3) CSS Class Selector

- The class selector selects HTML elements with a specific class attribute. It is used with a period character . (full stop symbol) followed by the class name.

```
    <!DOCTYPE html>
    <html>
    <head>
    <style>
        .center {
            text-align: center;
            color: blue;
        }
    </style>
    </head>
    <body>
    <h1 class="center">This heading is blue and center-aligned.</h1>
    <p class="center">This paragraph is blue and center-aligned.</p>
    </body>
    </html>
```

### CSS Class Selector for specific element

- If you want to specify that only one specific HTML element should be affected then you should use the element name with class selector.

```
p.center {
  text-align: center;
  color: red;
}
```

## 4) CSS Universal Selector

The universal selector is used as a wildcard character. It selects all the elements on the pages.

```
<!DOCTYPE html>
<html>
<head>
<style>
        * {
          color: green;
          font-size: 20px;
        }
</style>
</head>
<body>
<h2>This is heading</h2>
<p>This style will be applied on every paragraph.</p>
<p id="para1">Me too!</p>
<p>And me!</p>
</body>
</html>
```

## 5) CSS Group Selector

- The grouping selector is used to select all the elements with the same style definitions.
- Grouping selector is used to minimize the code. Commas are used to separate each selector in grouping.
- Let's see the CSS code without group selector.

```
h1 {
  text-align: center;
  color: red;
}

h2 {
  text-align: center;
  color: red;
}

p {
  text-align: center;
  color: red;
}
```

- It will be better to group the selectors, to minimize the code.
- To group selectors, separate each selector with a comma.

- The above can be grouped and written as given below

```
h1, h2, p {
  text-align: center;
  color: red;
}
```

## *Using CSS*

CSS can be added to HTML documents in 3 ways:

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

## *Inline CSS*

- We can apply CSS in a single element by inline CSS technique.
- This method mitigates some advantages of style sheets so it is advised to use this method sparingly.
- If you want to use inline CSS, you should use the style attribute to the relevant tag.

**Syntax:**

```
<htmltag style="cssproperty1:value; cssproperty2:value;"> </htmltag>
```

Example:

```
<h2 style="color:red;margin-left:40px;">Inline CSS is applied on this heading.</h2>
<p>This paragraph is not affected.</p
```

### *Disadvantages*

- You cannot use quotations within inline CSS.
- Cannot be reused anywhere else.
- These styles are difficult to be edit because they are not stored at a single place.
- It is not possible to style pseudo-codes and pseudo-classes with inline CSS.
- Inline CSS does not provide browser cache advantages.

## Embedding Style Sheets

- An internal CSS is used to define a style for a single HTML page.
- An internal CSS is defined in the <head> section of an HTML page, within a <style> element.

Example

```
<!DOCTYPE html>
<html>
<head>
<style>
body {background-color:powderblue;}
h1   {color:blue;}
p    {color:red;}
</style>
</head>
```

```
<body>

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

</body>
</html>
```

## Linking External Style Sheets

- 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.
- The external style sheet must be saved with a .**css** extension. This file should not contain HTML elements.

**Example**

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

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

</body>
</html>
```

"styles.css" may contain code like given below

```
body {
  background-color: powderblue;
}
h1 {
  color: blue;
}
p {
  color: red;
}
```

# Introduction to CSS3

- CSS stands for Cascading Style Sheets
- CSS describes how HTML elements are to be displayed on screen, paper, or in other media
- CSS saves a lot of work. It can control the layout of multiple web pages all at once
- External stylesheets are stored in CSS files

# Backgrounds

- CSS background property is used to define the background effects on element.
- There are 5 CSS background properties that affects the HTML elements:
  - **background-color**
  - **background-image**
  - **background-repeat**
  - **background-attachment**
  - **background-position**

## 1) CSS background-color

The background-color property specifies the background color of an element.

**Example**

The background color of a page is set like this:

```
body {
  background-color: lightblue;
}
```

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)"

Colors in CSS can be specified by the following methods:

- Hexadecimal colors
- Hexadecimal colors with transparency
- RGB colors
- RGBA colors
- HSL colors
- HSLA colors
- Predefined/Cross-browser color names
- With the currentcolor keyword

## 2) CSS background-image

- The background-image property is used to set an image as a background of an element.
- By default the image covers the entire element.

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

```
<head>
<style>
body {
background-image: url("paper1.gif");
margin-left:100px;
}
</style>
</head>
<body>
<h1>Hello Javatpoint.com</h1>
</body>
</html>
```

The background image can also be set for specific elements, like the <p> element:

**Example**

```
p {
  background-image: url("paper.gif");
}
```

## 3) CSS background-repeat

- By default, the background-image property repeats an image both horizontally and vertically.
- The image can be repeated horizontally using this property (background-repeat: repeat-x;)

Example

```
body {
  background-image: url("gradient_bg.png");
  background-repeat: repeat-x;
}
```

## *CSS background-repeat: no-repeat*

- Showing the background image only once is also specified by the background-repeat property:

**Example**

Show the background image only once:

```
body {
  background-image: url("img_tree.png");
  background-repeat: no-repeat;
}
```

## 4) CSS background-position

The background-position property is used to specify the position of the background image.

**Example**

Position the background image in the top-right corner:

```
body {
  background-image: url("img_tree.png");
  background-repeat: no-repeat;
  background-position: right top;
}
```

## 5) CSS background-attachment

The background-attachment property is used to specify if the background image is fixed or scroll with the rest of the page in browser window. If you set fixed the background image then the image will not move during scrolling in the browser. Let?s take an example with fixed background image.

**Example**

Specify that the background image should be fixed:

```
body {
  background-image: url("img_tree.png");
  background-repeat: no-repeat;
  background-position: right top;
  background-attachment: fixed;
}
```

**Example**

Specify that the background image should scroll with the rest of the page:

```
body {
  background-image: url("img_tree.png");
  background-repeat: no-repeat;
  background-position: right top;
  background-attachment: scroll;
}
```

## 6) 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.
- Instead of writing:

```
body {
  background-color: #ffffff;
  background-image: url("img_tree.png");
  background-repeat: no-repeat;
  background-position: right top;
}
```

You can use the shorthand property background:

**Example**

Use the shorthand property to set the background properties in one declaration:

```
body {
  background: #ffffff url("img_tree.png") no-repeat right top;
}
```

When using the shorthand property the order of the property values is:

- background-color

- background-image
- background-repeat
- background-attachment
- background-position

It does not matter if one of the property values is missing, as long as the other ones are in this order.

# Manipulating text

## *Text Color*

- The color property is used to set the color of the text. The color is specified by:
- The default text color for a page is defined in the body selector.

**Example**

```
body {
  color: blue;
}

h1 {
  color: green;
}
```

## *Text Color and Background Color*

- In this example, we define both the background-color property and the color property:

**Example**

```
body {
  background-color: lightgrey;
  color: blue;
}

h1 {
  background-color: black;
  color: white;
}
```

## *Text Alignment*

- The text-align property is used to set the horizontal alignment of a text.
- A text can be left or right aligned, centered, or justified.
- Left alignment is default if text direction is left-to-right, and right alignment is default if text direction is right-to-left.
- When the text-align property is set to "justify", each line is stretched so that every line has equal width, and the left and right margins are straight (like in magazines and newspapers)

**Example**

```
h1 {
  text-align: center;
}
```

```
h2 {
  text-align: left;
}

h3 {
  text-align: right;
}

div {
  text-align: justify;
}
```

## *Text Direction*

- The direction and unicode-bidi properties can be used to change the text direction of an element:

**Example**

```
p {
  direction: rtl;
  unicode-bidi: bidi-override;
}
```

## *Text Decoration*

- The text-decoration property is used to set or remove decorations from text.
- The value text-decoration: none; is often used to remove underlines from links:

**Example**

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

h3 {
  text-decoration: line-through;
}

h4 {
  text-decoration: underline;
}
```

## *Text Transformation*

- The text-transform property is used to specify uppercase and lowercase letters in a text.
- It can be used to turn everything into uppercase or lowercase letters, or capitalize the first letter of each word:

**Example**

```
p.uppercase {
  text-transform: uppercase;
}

p.lowercase {
```

```
  text-transform: lowercase;
}

p.capitalize {
  text-transform: capitalize;
}
```

## *Text Indentation*

- The text-indent property is used to specify the indentation of the first line of a text:

**Example**

```
p {
  text-indent: 50px;
}
```

## *Letter Spacing*

- The letter-spacing property is used to specify the space between the characters in a text.
- The following example demonstrates how to increase or decrease the space between characters:

**Example**

```
h1 {
  letter-spacing: 5px;
}

h2 {
  letter-spacing: -2px;
}
```

## *Line Height*

- The line-height property is used to specify the space between lines:

**Example**

```
p.small {
  line-height: 0.8;
}

p.big {
  line-height: 1.8;
}
```

## *Word Spacing*

- The word-spacing property is used to specify the space between the words in a text.
- The following example demonstrates how to increase or decrease the space between words:

**Example**

```
p.one {
  word-spacing: 10px;
}

p.two {
  word-spacing: -2px;
}
```

### *Text Shadow*

- The text-shadow property adds shadow to text.
- In its simplest use, you only specify the horizontal shadow (2px) and the vertical shadow (2px):

**Example**

```
h1 {
  text-shadow: 2px 2px;
}
```

### *Text shadow effect!*

**Example**

```
h1 {
  text-shadow: 2px 2px red;
}
```

- Then, add a blur effect (5px) to the shadow:

**Example**

```
h1 {
  text-shadow: 2px 2px 5px red;
}
```

## Margins and Padding

- Margins are used to create space around elements, outside of any defined borders.
- 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

**Example**

Set different margins for all four sides of a <p> element:

```
p {
  margin-top: 100px;
  margin-bottom: 100px;
  margin-right: 150px;
  margin-left: 80px;
}
```

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

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

**Example**
If the margin has 4 values, it represents top, right, bottom and left margins in order

```
p {
  margin: 25px 50px 75px 100px;
}
```

If the margin has 3 values, it represents top, right & left, bottom margins in order

```
p {
  margin: 25px 50px 75px;
}
```

If the margin has 2 values, it represents top & bottom, right & left margins in order

```
p {
  margin: 25px 50px;
}
```

If the margin has only one values, it represents margin on all sides

```
p {
  margin: 25px;
}
```

## *The auto Value*

- You can set the margin property to auto to horizontally center the element within its container.
- The element will then take up the specified width, and the remaining space will be split equally between the left and right margins.

**Example**

```
div {
  margin: auto;
  }
```

## *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.
- Margin collapse will happen only on left and right margins.

**Example**

```
h1 {
  margin: 0 0 50px 0;
}
```

```
h2 {
  margin: 20px 0 0 0;
}
```

- In the above example the margin will not be 70px, but will be 50px due to margin collapse.

### *CSS Padding*

- The CSS padding properties are used to generate space around an element's content, inside of any defined borders.
- With CSS, you have full control over the padding. There are properties for setting the padding for each side of an element (top, right, bottom, and left).

### *Padding - Individual Sides*

CSS has properties for specifying the padding for each side of an element:

- 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 elemen

### *Padding - Shorthand Property*

Padding shorthand property is similar to margin shorthand property

### *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
- If an element has a specified width, the padding added to that element will be added to the total width of the element.
- To keep the width at 300px, no matter the amount of padding, we can use the box-sizing property.
- This causes the element to maintain its actual width; if we increase the padding, the available content space will decrease.

**Example**

Use the box-sizing property to keep the width at 300px, no matter the amount of padding:

```
div {
  width: 300px;
  padding: 25px;
  box-sizing: border-box;
}
```

# Positioning Using CSS

- CSS helps you to position the HTML element.
- We can put any HTML element at whatever location.
- The position property specifies the type of positioning method used for an element.

## *Relative Positioning*

- Relative positioning changes the position of the HTML element relative to where it normally appears.
- We can use two values ***top*** and ***left*** along with the *position* property to move an HTML element anywhere in the HTML document

**Example**

```
div.relative {
  position: relative;
  left: 30px;
}
```

## *Absolute Positioning*

- An element with position: absolute is positioned at the specified coordinates relative to the screen top-left corner.

**Example**

```
div.absolute {
  position: absolute;
  top: 80px;
  right: 0;
}
```

## *Fixed Positioning*

- Fixed positioning allows you to fix the position of an element to a particular spot on the page, regardless of scrolling.
- Specified coordinates will be relative to the browser window.

```
div.fixed {
  position: fixed;
  bottom: 0;
  right: 0;
  width: 300px;
  border: 3px solid orange
```

## *Sticky Positioning*

- 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.

**Example**

```
div.sticky {
  position: sticky;
  top: 0;
```

```
  background-color: green;
  border: 2px solid #4CAF50;
}
```

## Responsive Web Design

- Responsive web design, also called RWD design, describes a modern web design approach that allows websites and pages to render (or display) on all devices and screen sizes by automatically adapting to the screen, whether it's a desktop, laptop, tablet, or smartphone.

### *Responsive Images*

- Responsive images are images that scale nicely to fit any browser size, using the width Property
- If the CSS width property is set to 100%, the image will be responsive and scale up and down.

**Example**

```
<img src="img_girl.jpg" style="width:100%;">
```

### *Using the max-width Property*

- If the **max-width** property is set to 100%, the image will scale down if it has to, but never scale up to be larger than its original size.

**Example**

```
<img src="img_girl.jpg" style="max-width:100%;height:auto;">
```

### *Show Different Images Depending on Browser Width*

- The HTML **<picture>** element allows to define different images for different browser window sizes.
- The images change depending on the browser size.
- The <picture> tag contains **<source>** and **<img>** tags.
- The attribute value is set to load more appropriate image.
- The <img> element is used to provide backward compatibility for browsers that do not support the element, or if none of the source tags matched.

**Example**

```
<picture>
  <source srcset="smallflower.jpg" media="(max-width: 500px)">
  <source srcset="midflowers.jpg" media="(max-width: 1000px)">
  <source srcset="flowers.jpg">
  <img src="smallflower.jpg" alt="Flowers">
</picture>
```

### *Viewport*

- 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.

**Setting The Viewport**

- HTML5 introduced a method to let web designers take control over the viewport, through the **<meta>** tag.
- **<meta>** tag gives the browser instructions on how to control the page's dimensions and scaling.

**Example**

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

- The width=device-width part sets the width of the page to follow the screen-width of the device (which will vary depending on the device).
- The initial-scale=1.0 part sets the initial zoom level when the page is first loaded by the browser.

## *Responsive Font Size*

- The text size can be set with a vw unit, which means the "viewport width".
- That way the text size will follow the size of the browser window

**Example**

```
<h1 style="font-size:10vw">Hello World</h1>
```

## *Media Queries*

- In addition to resize text and images, it is also common to use media queries in responsive web pages.
- With media queries we can define completely different styles for different browser sizes.
- It uses the @media rule to include a block of CSS properties only if a certain condition is true.

**Example**

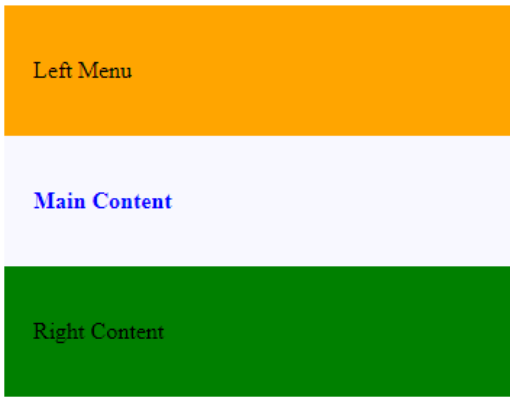
If the browser window is 600px or smaller, the background color will be lightblue:

```
@media screen and (max-width: 600px) {
  body {
    background-color: lightblue;
  }
}
```

- Elements can be displayed horizontally on large screens and stacked vertically on small screens

When viewed in Full browser size

When resized or viewed in mobile devices




## Introduction to LESS (Leaner CSS)

- LESS is a CSS pre-processor that enables customizable, manageable and reusable style sheet for website.
- LESS is a dynamic style sheet language that extends the capability of CSS.
- LESS supports creating cleaner, cross-browser friendly CSS faster and easier.
- LESS is designed in JavaScript and also created to be used in live, which compiles faster than other CSS pre-processors.
- LESS keeps your code in modular way which is really important by making it readable and easily changeable.
- Faster maintenance can be achieved by the use of LESS variables.

### *Variables*

- LESS allows variables to be defined with an @ symbol. The Variable assignment is done with a colon(:).
- Repetition of same value many times can be avoided by the use of variables.
- We have to create the style.less file.

```
@width: 10px;
@height: @width + 10px;

#header {
  width: @width;
  height: @height;
}
```

- Compile the *style.less* to *style.css* by using the following command −

```
lessc style.less style.css
```

Outputs:

```
#header {
  width: 10px;
  height: 20px;
}
```

| Index | Variable | Explanation |
|---|---|---|
| 1) | Overview | A variable can be used to avoid the repetition of same value occurred many times. |
| 2) | Variable Interpolation | You can use variables on other places like selector names, property names, URLs and @import statements. |
| 3) | Variable Names | A variable can be defined with a variable name containing a value. |
| 4) | Lazy Loading | Lazy Loading specifies that you can use the variables even though they are not declared yet. |
| 5) | Default Variables | Defaulr variable facilitates you to set a variable only when it?s not already set. It is not a necessary feature because variables can be easily overridden by defining them afterwards. |

## *Mixins*

- Mixins are a way of including ("mixing in") a bunch of properties from one rule-set into another rule-set.
- Mixins are similar to functions in programming languages.
- Mixins are a group of CSS properties that allow you to use properties of one class for another class and includes class name as its properties.
- In LESS, we can declare a mixin in the same way as CSS style using class or id selector. It can store multiple values and can be reused in the code whenever necessary.

**Example**

```
.bordered {
  border-top: dotted 1px black;
  border-bottom: solid 2px black;
}
```

- When we want to use these properties inside other rule-sets. We just have to use in the name of the class where we want the properties, like:
- The properties of the .bordered class will now appear in both #menu a and .post a.

```
#menu a {
  color: #111;
  .bordered();
}

.post a {
  color: red;
  .bordered();
}
```

### *Nesting*

- Less gives you the ability to use nesting instead of, or in combination with cascading.

```
#header {
  color: black;
}
#header .navigation {
  font-size: 12px;
}
#header .logo {
  width: 300px;
}
```

- In Less, we can also write it this way:

```
#header {
  color: black;

  .navigation {
    font-size: 12px;
  }

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

## SASS (Syntactically Awesome Style sheets)

- **Sass** stands for **S**yntactically **A**wesome **S**tylesheet
- Sass is an extension to CSS
- Sass is a CSS pre-processor
- Sass is completely compatible with all versions of CSS
- Sass reduces repetition of CSS and therefore saves time

- Sass was designed by Hampton Catlin and developed by Natalie Weizenbaum in 2006
- Sass is free to download and use

Why Use Sass?

- Stylesheets are larger, more complex, and harder to maintain.
- Sass lets you use features that do not exist in CSS, like variables, nested rules, mixins, imports, inheritance, built-in functions, and other stuff.
- In the following example, HEX values need not be repeated multiple times.
- Variations of the same colors can be implemented easily.

Sass Example

```
/* define variables for the primary colors */
$primary_1: #a2b9bc;
$primary_2: #b2ad7f;
$primary_3: #878f99;

/* use the variables */
.main-header {
  background-color: $primary_1;
}

.menu-left {
  background-color: $primary_2;
}

.menu-right {
  background-color: $primary_3;
}
```

## *Sass Variable Scope*

- Sass variables are only available at the level of nesting where they are defined.
- In the following example the color of the text inside a <p> tag will be red.
- Inside the <h1> the text color will be green.

**Example**:

```
$myColor: red;

h1 {
  $myColor: green;
  color: $myColor;
}

p {
  color: $myColor;
}
```

Corresponding CSS Output will be

```
h1 {
  color: green;
}

p {
  color: red;
}
```

## *Using Sass !global*

- The default behavior for variable scope can be overridden by using the !global switch.
- !global indicates that a variable is global, which means that it is accessible on all levels.

**Example**

```
$myColor: red;

h1 {
  $myColor: green !global;
  color: $myColor;
}

p {
  color: $myColor;
}
```

- Now the color of the text inside a <p> tag will be green.
- The CSS output will be

```
h1 {
  color: green;
}

p {
  color: green;
}
```

## *Sass Mixins*

- The @mixin directive lets you create CSS code that is to be reused throughout the web-site.
- The @include directive is created to let you use (include) the mixin.

**Defining a Mixin**

A mixin is defined with the @mixin directive.

SCSS Syntax:

```
@mixin important-text {
  color: red;
  font-size: 25px;
  font-weight: bold;
  border: 1px solid blue;
}
```

**Using a Mixin**

The `@include` directive is used to include a mixin.

SCSS Syntax:

```
.danger {
  @include important-text;
  background-color: green;
}
```

The Sass transpiler will convert the above to normal CSS:

CSS output:

```
.danger {
  color: red;
  font-size: 25px;
  font-weight: bold;
  border: 1px solid blue;
  background-color: green;
}
```

## *Sass Nested Rules*

- Sass lets you nest CSS selectors in the same way as HTML.
- In the below example in Sass, the ul, li, and a selectors are nested inside the nav selector.
- While in CSS, the rules are defined one by one (not nested):

- Example SCSS Syntax:

```
nav {
  ul {
    margin: 0;
    padding: 0;
    list-style: none;
  }
  li {
    display: inline-block;
  }
  a {
    display: block;
    padding: 6px 12px;
    text-decoration: none;
  }
}
```

CSS Syntax

```
nav ul {
  margin: 0;
  padding: 0;
  list-style: none;
}
nav li {
```

```
  display: inline-block;
}
nav a {
  display: block;
  padding: 6px 12px;
  text-decoration: none;
}
```