Introduction to CSS

CSS stands for Cascading Style Sheets.

Prior to CSS the way to add styling like color, alignment or background image was to add them as attributes to the elements. But this made the html code full of attributes making the code look dirty. So the styling part was moved to CSS.

So CSS is the language for styling the HTML elements.

Adding styling to tags

So how do we style the elements?

Let's take a simple example where we will add a red color to a font inside a paragraph element.

To achieve this, we have to write a some code in CSS as shown below in the head section inside a **style** tag

```
<html>
    <head>
        <style type = “text/css”>
        p{
        color :red;
        }
        </style>
    </head>
    <body>
        <p> Hi..I am text inside a paragraph</p>
    </body>
</html>
```

CSS example Explained

First we have added <style> tag in the head section. Reason is that head section loads before the body section of HTML. Note that we specify the type attribute as type = “text/css” to tell browsers that the code is CSS.

Inside CSS code we mention the tag name we want to format and then add the properties inside curly braces.

Structure of a style

What we created in the previous example is a style definition

Each style definition contains:

A property

A colon

A value

A semicolon to separate two or more values

A style definition can include one or more values for styling

p{font-size:12px; color:red}

Tag Selector

A tag selector uses html tags to select the content to be styled.

Ex: The CSS code below styles a <h1> tag font to red color and font size of 12px.

```
h1{
   color:red;
   font-size:12px;

   }
```

CSS Selectors

CSS selectors are used for selecting the content we want to style in HTML.

Like in the previous example we used a tag selector to select a tag to apply the styling.

CSS selectors are broadly classified into following categories

1)Tag selector

2)ID selector

3)Class selector

4)Mixed selectors which are combination of the above three.

Tag Selector

A tag selector uses html tags to select the content to be styled.

Ex: The css code below styles a <h1> tag font to red color and font size of 12px.

```
h1{
  color:red;
  font-size:12px;
 }
```

ID Selector

All html tags can be given an ID to identify a specific element.

Ex: Hello

ID Selector is a part the CSS rule where an element can be styled based on the Id. So for styling the paragraph above we can use the following code. Just add **#** symbol before the id and add the styling. See the code below:

```
#para{
   color:red;
   font-size: 20px;
   }
```

Class Selector

ID can be used to identify a specific element. To classify a group of elements we use a class. See the example below. We have used the class **red-font** to two elements namely <p> and <h1>.

```
<p class = “red-font”>hello</p>
<h1 class = “red-font”>I am a heading</h1>
```

To add a class selector just add a dot before the class name and write the CSS rule. See the example below

```
.red-font{
color:red
}
```

Using a class selector we can apply the same style to a group of html elements

Types of Styles

Style are classified into three types

1)Embedded style: Embedded style has CSS rules kept inside the style tag in the same html file. We have already covered the embedded example before.

2)Inline style: If style is written within the tag, it is called inline style

3)External style has CSS rules linked in a separate external file

Let's explore them now.

Embedded Style

```
<html>
    <head>
        <title>Getting Started</title>
        <style type = "text/css">
        p {font-family: sans-serif; color: orange}
        </style>
    </head>
    <body>
        <p> Hi..I am text inside a paragraph </p>
    </body>
</html>
```

Inline Style

Below is the example of inline style. Note that style is written as attribute of tag.

```
<html>
    <head>
        <title>Getting Started</title>
    </head>
    <body>
        <p style = “font-family: sans-serif; color: orange;”> Hi..i am text inside a
        paragraph </p>
    </body>
</html>
```

External Style

The example below is an example of external style sheet. All the styles are kept in style.css

```
<html>
    <head>
        <style type = “text/css” rel = “stylesheet”  href = “style.css”></style>
    </head>
    <body>
        <h1>hello</h1>
    </body>
</html>
```

External style sheet is better

It's important to note that in development always use external style sheet because that helps to maintain style sheets. Having external style sheets help maintain the DRY principle(Don't repeat yourself).

We should avoid using embedded style sheets.

Using inline style sheets makes the html code messy and hence we should avoid it.

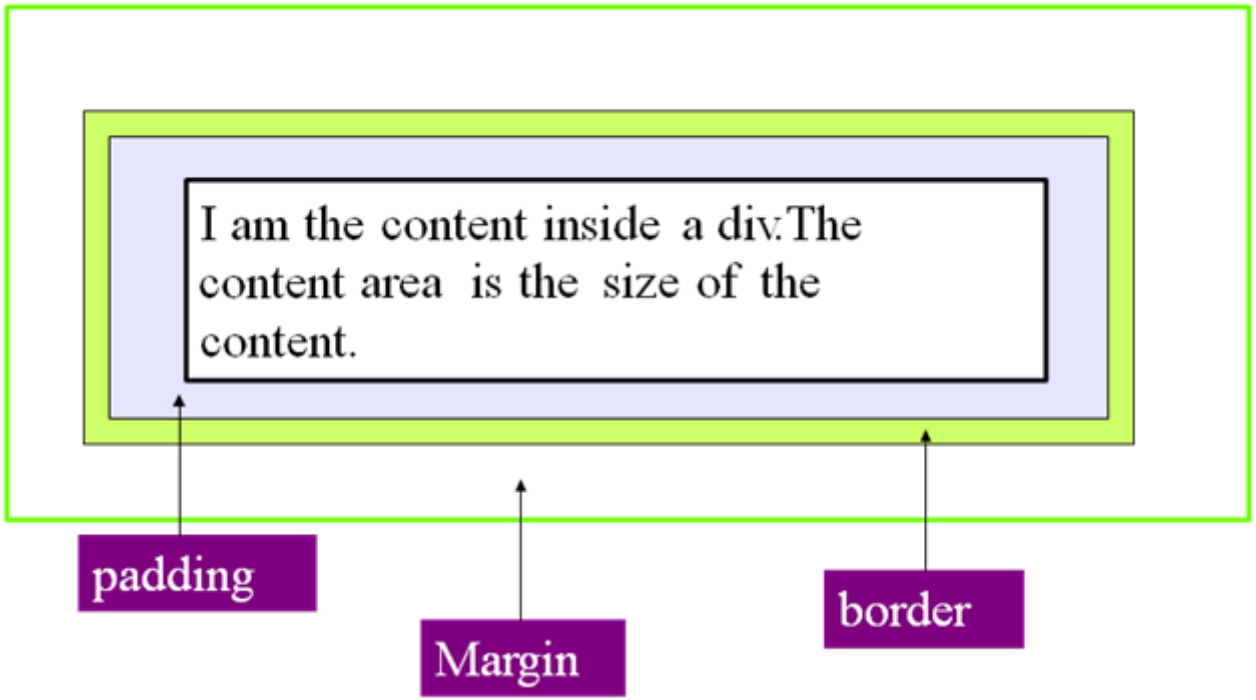
Box Model

Every HTML Element has padding, margin and border. Padding is the spacing around the content

The border is the border line surrounding the padding. Margin is surrounding space around the border.

Have a look at the image shown in the next slide

Margins, border and padding



Setting padding, margin

margin property is used to set the margin for an element. Use

margin:<top><top-right><top-bottom><top-left> Similarly: padding:<top><top-right><top-bottom><top-left> Ex:

```
h1{
    margin: 5px 2px 3px 5px;
}
```

Alternatively we can use the following

```
h1{
  margin-top:5px; margin-
  right:2px; margin-
  bottom:3px;
  margin-left:5px
}
```

Setting borders

Border can be set with three properties

- **border-width**
- **border-style**
- **Border-color**

For setting border sides we can use

- **border-top**
- **border-bottom**
- **border-left**
- **Border-right**

We can use the shorthand

border:<border-width><border-style><border-color>

```
h1{
   border: 5px solid red
}
```

CSS Box Model

CSS Box model uses divs, margins and padding to create a complete web page layout. The method of creating page layout using div is called table less design.

Prior to this, designer used tables to create web page layout. It is not recommended now to use this approach because it gives a non uniform layout.

Inbuilt Browser styles

Every browser provides a set of inbuilt CSS styles for html elements.

For example, Chrome gives margin for paragraph element. This helps to give a better layout when no CSS is used. However the problem is that every browser has different inbuilt CSS. This makes the website look different in different browsers.

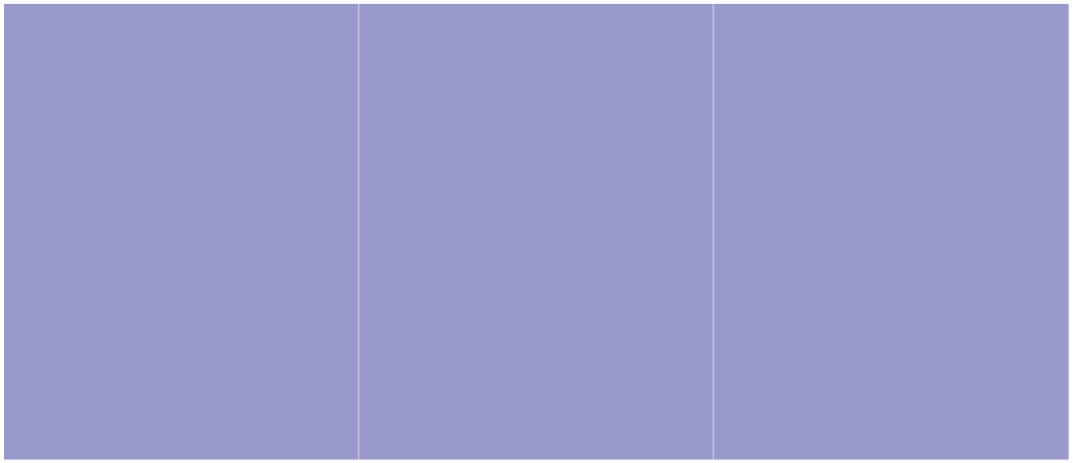
One way to solve this problem is by resetting all the values like padding, margin etc to 0 and then giving them values as required.

A better alternative is using CSS resets libraries. One example is normalize.css library which should be included in all your web pages to reset the default browser styles.

Float Property

The float property removes the element from the document flow and moves them to the edge specified in the float property.

float: left / right / none



Float property

The float property removes the element from the document flow and moves them to the edge specified in the float property

As we have discussed before, div is a block level element and block level elements adds a line break. But using float property we can let them float to the right or left and inline within the container.

float: left / right / none

Below shows div1 and div 2 floating in the left and div3 floating in the right

Problems with float

Float has one major problem. If we float any elements inside a container, then the container doesn't expand as per the size of the inner elements.

There are 2 fixes for it. One fix is to keep clear the float of the last element. If the last element also has to float, then we add a new empty div and set its property to float : none.

Have a look at the code used to float 2 images. One image floats to left and other floats to right.

```
<head>
  <style type = "text/css">
  img{
    height: 200px;
    width: 200px;
    margin: 0;
    border:3px solid green;
  }
  .float-left{
    float: left;
  }
  .float-right{
    float:right;
  }
  #outer{
    margin-top:100px;
    margin-left: 40px;
    width:670px;
    border:2px solid red;
    padding:10px;
  }
  #last-element{
    clear:both;
  }
  </style>
</head>
<body>
  <div id = "outer">
    The content is part of outer div
    <img class = "float-left" src = "jewel.jpg">
    <img class = "float-right" src = "jewel.jpg">
    <div id = "last-element"></div>
  </div>
</body>
```

Clear-fix

We have a better solution for the float problem. Instead of just using a div, we can use a clear fix class and add it to the container.

Have a look at the next code making use of clearfix.

Clear-fix code

```
<head>
  <style type = "text/css">
  img{
    height: 200px;
    width: 200px;
    margin: 0;
    border:3px solid green;
  }
  .clear-fix:before,.clear-fix:after{
    content: "";
    display:table;
  }
  .clear-fix:after{
    clear:both;
  }
  .clear-fix{
    zoom:1; /* IE 6 and 7 fix*/
  }
  .float-left{
    float: left;
  }
  .float-right{
    float:right;
  }
  #outer{
    margin-top:100px;
    margin-left: 40px;
    width:670px;
    border:2px solid red;
    padding:10px;
  }
  </style>
</head>
<body>
  <div id = "outer" class = "clear-fix">
    The content is part of outer div
    <img class = "float-left" src = "jewel.jpg">
    <img class = "float-right" src = "jewel.jpg">
  </div>
</body>
```

Div, Span

Every element has a default display value depending on what type of element it is.

The default for most elements is usually block or inline.

div is the standard block-level element. A block-level element starts on a new line and stretches out to the width of the container.

Example of other block-level elements are p and form,h1 etc

span is the inline element. An inline element helps to wrap some text inside a paragraph like this without disrupting the flow of that paragraph. The link <a> element is one example

"Display" Property Block/Inline

CSS gives us the flexibility to change the behavior of element using display property

For a block level element we can set **display:inline** which will force the element to behave as inline element.

Similarly we can set the property of an inline element to **display:block** if required.

“display" Property None

Setting display to none will render the page as though the element is not present in the flow.

It is commonly used with JavaScript to hide or show elements without deleting and recreating them

Note: This is different from visibility.

Setting display to none will render the page as though the element does not exist . visibility: hidden hides the element, but the element takes up the space as it would if it was fully visible.

margin: auto

```
#outer {
  width: 600px;
  margin: 0 auto;
}
```

Setting the margin property to auto automatically aligns it to the center.

Note: We have given two values. **0 auto** which means the top and bottom margins are 0 and left and right are auto. This is shorthand notation

max-width

```
#outer {
  max-width: 600px;
  margin: 0 auto;
}
```

Setting **max-width** property limits the stretch of the element when the screen resolution is higher.

max-width instead of width improves the browser's handling of small windows. It’s used for making a site usable on mobile.

Width box model issue

If padding and border property is set, the element appears bigger than what we set.

The element's border and padding stretches out the element beyond the specified width.

Consider the example

```
.section { width:
  500px; margin:
  20px auto;
  padding: 50px;
  border-width: 10px;
}
```

In the code above, the actual width of the element is width+padding+border-width. In this case the actual element width is 500+2*50+2*10 = 620px

Solution: Developers have always just written a smaller width value than what they wanted, subtracting out the padding and border. We will get a better solution when we move to CSS3 .

"display" property block

Every element has a default display value depending on what type of element it is.

The default for most elements is usually block or inline. A block element is often called a block-level element. An inline element is always just called an inline element.

div is the standard block-level element. A block-level element starts on a new line and stretches out to the left and right as far as it can.

Example of other block-level elements are p and form, and new in HTML5 are header, footer, section etc

Combining Selectors

Selectors can be combined together.

```
<div class="burger">
        <p>...</p>
        <p>...</p>
        <p class="cheese">...</p>
</div>
.burger p {
  background: brown;
}
p.cheese {
  background: yellow;
}
```

.burger p: Note the space between .burger and p. This selects all the p elements inside an element with the class burger.

p.cheese : Note that there is no space between p and .cheese. This selects

the paragraph with a class cheese.

Multiple Classes

- HTML elements can have more than one class attribute value so long as each value is space separated.
- This way we can add additional styles from another class.

```
<button class="btn notice">...</button>
<button class="btn error">...</button>
.btn {
  font-size: 16px;
}
.notice {
  background: red;
}
.error {
  background: green;
}
```

In the code above the button element gets properties of both btn and notice

class.

Keyword Color

Keyword color values are names (such as red, green) that map to a given color

Keyword color values are simple in nature, however they provide limited options

```
.main {
  background: maroon;
}
.nav {
  background: yellow;
}
```

RGB Color

RGB color value is stated using the rgb() function, which stands for red, green, and blue.

This function accepts three comma-separated values. Each value is an integer from 0 to 255.

A value of 0 is pure black; a value of 255 is pure white.

```
.main {
  background: rgb(128, 0, 0);
}
.nav {
  background: rgb(255, 255, 0);
}
```

RGBa Colors

RGBa color values may include an alpha also known as transparency by using the rgba() function.

This function uses a fourth value, which is a number between 0 and 1, including decimals.zero value creates a fully transparent colormaking it invisible, and a value of 1 creates a fully opaque color.

Values between 0 and 1 would create a semi-transparent color.

For shade of orange to appear 70% opaque, we would use an RGBa color value of rgba(255, 102, 0, .7).

```
.container {
  background: rgba(128, 0, 0,.7);
}
```

HSL & HSLa Colors

HSL color values use the hsl() function, which stands for hue, saturation, and lightness.

Within the parentheses, the function accepts three comma-separated values, much like rgb().

Returning to our shade of orange, as an HSL color value it would be written as hsl(24, 100%, 100%).

HSL color values, like RGBa, may also include an alpha, or transparency, channel with the use of the hsla() function

```
.wrapper {
  background: hsla(0, 100%, 50%,.7);
}
.container {
  background: hsl(60, 100%, 100%);
}
```

Absolute Length

Absolute length values are the simplest length values, as they are fixed to a physical measurement, such as inches, centimeters, or millimeters.

The most popular absolute unit of measurement is known as the pixel and is represented by the px unit notation.

The pixel is equal to 1/96th of an inch; thus there are 96 pixels in an inch. The exact measurement of a pixel, however, may vary slightly between high-density and low-density depending on viewing devices.

```
div {
  font-size: 14px;
}
```

Relative Length

Relative Lengths is not fixed units of measurement; they rely on the length of another measurement.

We have **percentages and ems** for relative length

Percentage length is defined in relation to length of another object. For example, to set the width of an element to 40%, we have to know the width of its parent element, the element it is nested within, and then identify 40% of the parent element's width

```
.main {
  width: 40%;
}
```

Relative Lengths Percentage

The **em unit** is also a very popular value for relative measurements.

The length is calculated based on an element's font size.

A single em unit is equivalent to an element's font size. So, for example, if an element's font size is 24 pixels and has width set to 3em, the calculated width would equal 72 pixels (24 pixels multiplied by 3.

Ex

```
.menu {
  font-size: 24px;
  width: 3em;
}
```

Relative Lengths Percentage

The em unit is also a very popular relative value. The em unit is represented by the em unit notation, and its length is calculated based on an element's font size.

A single em unit is equivalent to an element's font size. So, for example, if an element has a font size of 14 pixels and a width set to 5em, the width would equal 70 pixels (14 pixels multiplied by 5).

Ex

```
.banner {
 font-size: 14px;
 width: 5em;
}
```

Responsive Web Design

Responsive web design is a set of css/html practices to make website accessible and readable in mobiles, desktops, tablets and other device(if required).

The technique is to make the design flexible to make content readable in different resolution and view modes.

Measurements in responsive design

Following are the points to note for making responsive websites:

1)Lengths should be specified in relative units like percentages and ems.

2)we should use media queries to change the CSS for different devices

3)Layout should be fluid and responsive.

Inheritance

Inheritance is the process by which some CSS properties applied to one tag are passed on to nested tags. For example, a <p> tag is always nested in- side of the <body> tag, so properties applied to the <body> tag get inherited by the <p> tag. Say you created a CSS tag style

Many CSS properties don't pass down to descendant tags at all. For example, the border property (which lets you draw a box around an element) isn't inherited, and with good reason.

A general rule is that properties that affect the placement of elements on the page or the margins, background colors, and borders of elements aren't inherited.

Cascading of Styles

Cascade is a set of rules for determining which style properties get applied to an element. It specifies how a web browser should handle multiple styles that apply to the same tag and what to do when CSS properties conflict.

Inherited Styles Accumulate

Ex: Imagine a font family applied to the <body> tag, a font size applied to a <p> tag, and a font color applied to an <a> tag. Any <a> tag inside of a paragraph would inherit the font from the body and the size from the paragraph.

Ex: For a strong tag that is inside p for the following style

```
body { font-family: Verdana, Arial, Helvetica, sans-serif; }
p { color: grey; }
strong { font-size: 14px; }
```

Cascading for Ancestors

What happens when inherited CSS properties conflict?

Nearest Ancestor Wins

Consider a page where We have set the font color for the body tag to
red and the paragraph tag to green. Now imagine that within one paragraph, there's
 tag.
The tag inherits from both the body and p tag styles, so
is the text inside the tag red or green? T**he green** from the paragraph style.

That's because the style that's closest to the tag.

Specificity!

Try the following CSS and html code:

```
p.color-red{
        color:red;
}
p{
        color:green;
}
```

Below is html code(html, head and doctype tags should be added to the actual code)

```
<body>
<p class= “red”> What is the color of this text</p>
</body>
```

What is the color of the font. As per the rule, the later style should override the first one. Means the color of the font should be red?

The answer is No. Because the specificity of first style is higher.

Specificity Score

So Here is the rule.
The pattern goes like that
<ID selector count>:<class selectors count>:< tags selector count>

Let's see with examples below

```
p.color-red{
color:red;
 ------- Specificity score is 0:1:1 because there are 0 id selectors,1 class and 1 tag selector
}

p{
color:green;
 ------- Specificity score is 0:0:1 because there are 0 id selectors,0 class and 1 tag selector
}

p #data div.color-red {
color:blue;
------- Specificity score is 1:1:2 because there are 1 id selectors,1 class and 2 tag selector
}
```

Score Preference Rules

So rule is id score takes the highest preference. Second is class score and last is id score.
So a selector with score of 1:0:0 is higher than score of 0:4:1. Similarly

A selector with score 0:1:0 is higher than 0:0:4

A selector with a greater specificity score overwrites the other selector with lesser specificity irrespective of order where the rules are written

Media Queries

Using media queries, we can target CSS rules based on – for instance – screen size, device-orientation or display - density. See the example below. The example below sets the CSS property for elements only if the screen resolution min-width

```
@media (min-width: 481px) and (max-width: 768px){
.sidebar {
        float:none;
  border-top: 5px solid red;
  padding-top: 10px;
}
```