

CSS – Cascading Style Sheet

<iframe>

The iframe tag specifies an inline frame. An inline frame is used to embed another document within the current HTML document or render one web page into another page.

Tip: Use CSS to style the <iframe> (see example below).

Tip: It is a good practice to always include a title attribute for the <iframe>. This is used by screen readers to read out what the content of the <iframe> is.

Input:

```
<body>  
  <iframe src="https://attainu.com" frameborder="1"></iframe>  
</body>
```

Output:



Although we have only provided one tag; it is rendering the whole AttainU web page into our HTML document. The website will have its own content running. It was difficult to share different web page content into another page. Hence, using iframe we can show some information from another page so that the user doesn't have to leave our page.

Frameborder → the value 1 (the default) draws a border around the frame. The value 0 removes the border around this frame, but you should instead use the CSS property **border** {} to control <iframe> borders.

CSS – Introduction:

Cascading Style Sheets (CSS) is a stylesheet language used to describe the presentation of a document written in HTML or XHTML (including XML dialects such as SVG, MathML or XHTML). CSS describes how elements should be rendered on screen, on paper, in speech or on other media.

Selectors, Properties and Values:

```
selector {  
    property-name: value;  
}
```

Using selectors (h1-h6, p, div) we target that particular element in our html document, so that we can write some CSS rules. There are multiple properties and for each property we will be defining a particular value.

Ex:

```
<style type="text/css">  
    h1 {  
        background-color: green;  
        color: pink;  
        text-align: center;  
    }
```

The selector in the above example is '**h1**',

The property-name in the above example are '**background-color**', '**color**' & '**text-align**'

If you have more than one <h1> tag, this property would be applied to all the <h1> tags.

Ex:1

Input:

```
<head>
  <style type="text/css">
    h1 {
      background-color: green;
      color: yellow;
      text-align: center;
    }
    span {
      background-color: green;
      color: white;
    }
  </style>
</head>
<body>
<h1>this h1 tag has yellow color font and green background with its text at the center of the page.</h1>
<span>This is a span</span>
</body>
```

Output:

h1, yellow font-color, green background text at the center of the page.

This is a span

Applying Styles:

There is a certain order to how styles are applied into html.

There are two things (priority & specificity) of which in priority we have 3 ways

- Inline
- Internal/Same
- External

```
9      <style>
10
11      h1{
12        background-color: green;
13      }
14
15      span{
16        background-color: hotpink;
17      }
18
19    </style>
20  </head>
21  <body>
22
23    <h1 style="background-color: blue;">This is my H1</h1>
24
25    <span>This is a span</span>
26
27  </body>
```

For the code in the picture, the output shows as

This is my H1

This is a span

The **background color** for the h1 tag in inline is given as **blue**, whereas in the internal/same it is given as **green** and the output is **blue**. The reason is because of the priority level.



If the color is given in the inline as well as internal, then the inline value would be taken and not the internal value.

The last way of applying styles is via **external CSS file**. You create a external **.css file** and then link it to the html document using `<link>` tag.

`<link rel="stylesheet" href="">`

In the href attribute, you need to give the **.css file** name. This will connect the html and css files. Now, when you change the attribute value it would reflect in the html document.

The background color can be applied via inline, internal or external style sheets.

CSS-Properties – Length:

The length property has a different units. A css file can have 100's of lines of code and having it in the same html document will make it look very clumsy.

Let's add a <div> to our html document with style (internal)


```
div {  
  width: 150px;  
  height: 150px;  
  background-color: crimson;  
}
```

Output:

This is my H1

This is a span

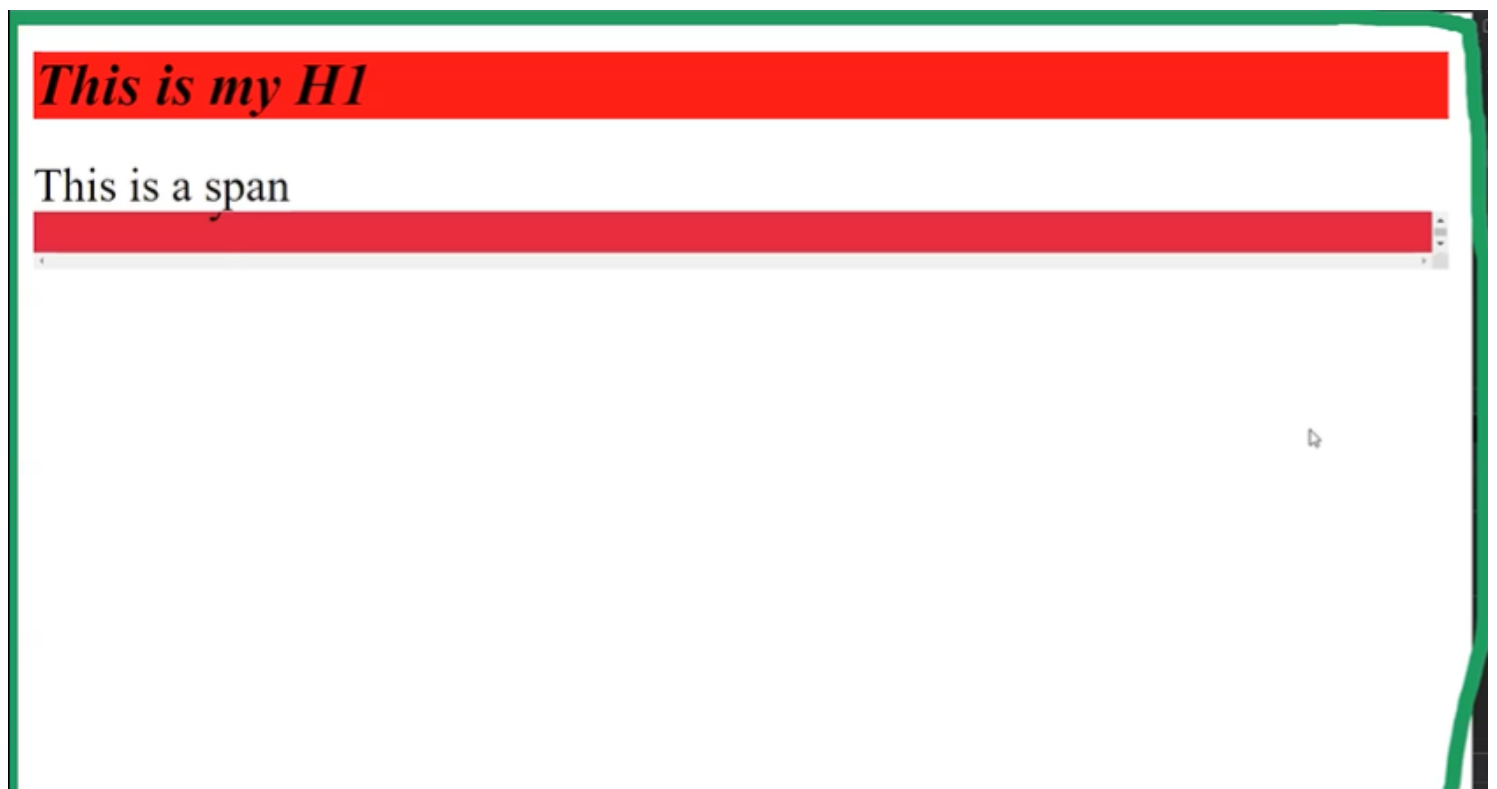
This is my div



All the elements in the div would have the same parameters as that of the <div>. but if my content takes the same size on every screen, then the image would be going out of the webpage content. So, instead of giving fixed values, if we give the size values in percentage, then the elements would automatically adjust to the screen size.



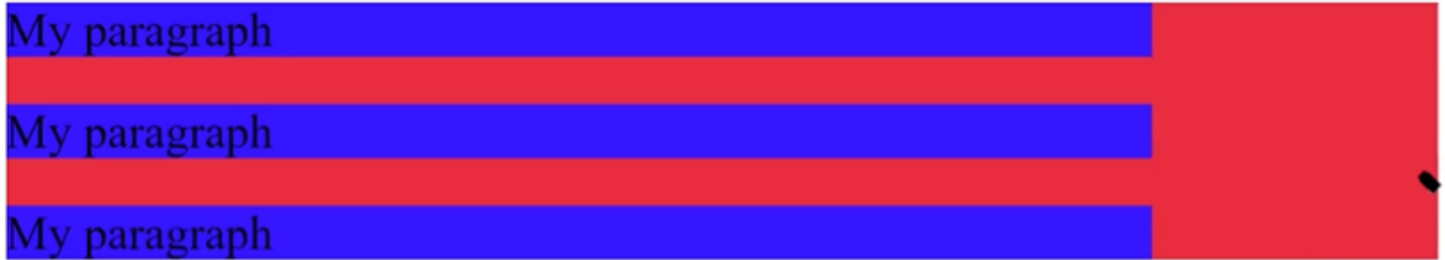
If we don't want content in our `<div>` to be shown, then we can use the attribute `overflow: hidden` and it will hide the remaining content. If use attribute: `scroll`, then we can scroll through the div content.



We use percentage in responsive designs. Now, `<div>` is a block level element so it would take complete width of the web page. Inside there is a paragraph which is also a block level element and so it would also take the whole width; at the same time increasing the height of the `<div>` element.

If we add a `background-color: blue` to the paragraphs, and specified `<p>` to take 80% of the web page, then the output would be,

This is a span



As we decrease the size of the web page, the `<p>` element would still be covering the 80% of the decreased width. If the width is 50%, then it would only take 50% width of the web page. The percentage value is always relative to the parent container.

em is also a relative value. Based on the parents font-size, the value of `em` is decided. In the above example, `<div>` is a parent element and `<p>` is a child element. Now, if font-size in `div` = 20px, then the child also will have the same font-size is font-size of `<p>` = 1 *em*.

`<div>` = 20px → `<p>` = 1 *em* which means the font-size of `<p>` is 20 px
`<div>` = 20px → `<p>` = 0.5 *em* which means the font-size of `<p>` is 10 px

Colors:



There are few colors which are already predefined. But if you want to get a particular color, then we need to use rgb() function. rgb → Red, Green, Blue

For Red – rgb(255,0,0)

For Green – rgb(0,255,0)

For Blue – rgb(0,0,255)

For VS Code, install an extension of “HTML CSS Support”.

Gradient:

Syntax

```
background-image: linear-gradient(direction, color-stop1, color-stop2, ...);
```

Direction – Top to Bottom (default) – Example

```
#grad {  
  background-image: linear-gradient(red, yellow);  
}
```

Direction – Left to Right – Example

```
#grad {  
  background-image: linear-gradient(to right, red, yellow);  
}
```

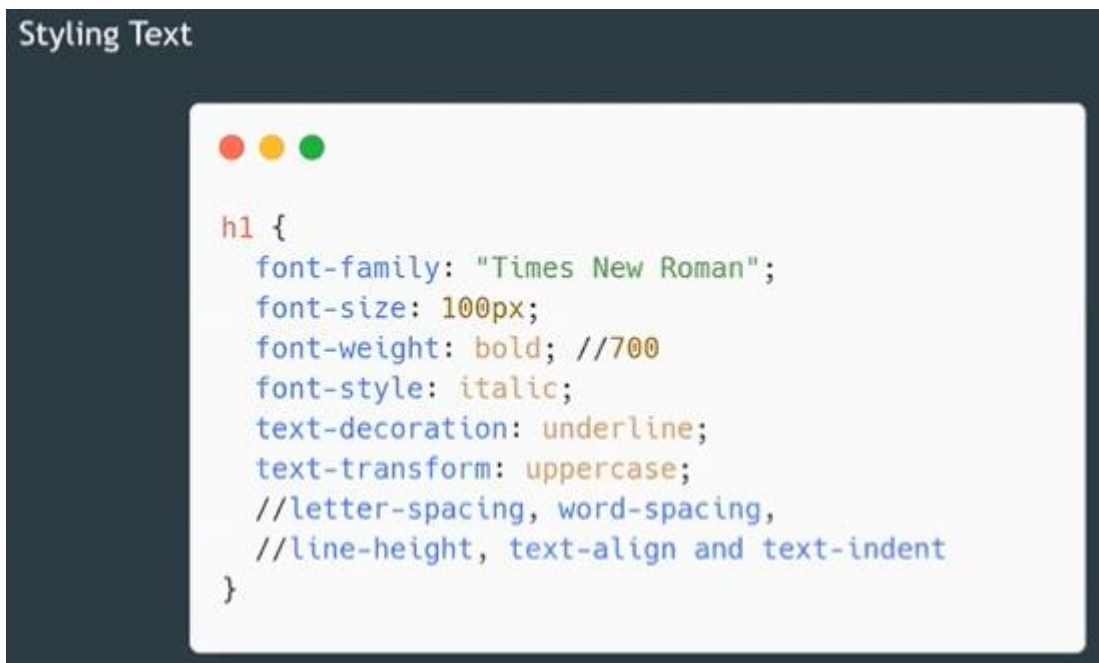

Direction - Diagonal

You can make a gradient diagonally by specifying both the horizontal and vertical starting positions.

Example

```
#grad {  
  background-image: linear-gradient(to bottom right, red, yellow);  
}
```

Styling Text:



font-weight:

```
{  
  font-weight: bold;  
}
```

It is similar to the tag. But in this we have a range of font thickness from 100 to 900 wherein 900 font-weight would be equal to using the tag and 100 font-weight would be equal to not using the tag.

font-transform:

```
{  
  Font-transform: uppercase  
}
```

All the alphabets in the particular element would be capitalized.

Cascading:

Cascading" means that styles can fall (or **cascade**) from one style sheet to another, enabling multiple style sheets to be used on one HTML document.

```
<style type="text/css">  
  h1 {color: green;}  
  
  h1 {color: blue;}  
</style>
```

The output would be in blue color, because the browser reads from the top and the last or recent value given to a particular attribute will be displayed.

‘*’ is a global selector. Once you use this, all the elements in the html document would have the same attributes given to the ‘*’ global selector.

```
<div>  
  <h1>This is to show you how cascading works</h1>  
  <p>this is to show you how cascading works</p>  
</div>
```

Output:

This is div

This is to show you how cascading works

this is to show you how cascading works