

CSS

Topics:

- CSS Reset
- Transform
- Transition
- Gradients
- SASS

CSS Reset:

A CSS Reset style sheet is a list of rules that reset all of the default browser style as they can interfere with the styles we actually want to apply. This provides a consistent base across all browsers. We reset the browser styles for two primary reasons:

- **Not all browsers apply the same default rules.** They may be difficult to provide the same designs in each browser if the basic styles are different.
- Once you start designing and coding all of the fine details of your site, you may discover that a lot of what you are doing is simply **overriding default browser styles**. The reset does this quickly so that you don't have to.
- We can use the reset style sheet as an external style sheet just like we do with our normal styles. **Just make sure to add it first**, since order matters.

```
<link rel="stylesheet" href="reset.css" media="screen" />
```

```
<link rel="stylesheet" href="styles.css" media="screen" />
```

WHY?

Two Reasons:

- This will save you a lot of time and frustration when you are creating complicated layouts with CSS.
- You're the designer. You shouldn't let the browser makers decide how any part of your web pages will look.

<https://gist.github.com/DavidWells/18e73022e723037a50d6>

Transform:

The transform property applies a transformation to an element. This property allows you to rotate, scale, move, skew, etc., elements.

In css

```
.box {  
  height: 50px;  
  width: 50px;  
  margin: 5px  
  background-color: yellow;  
}
```

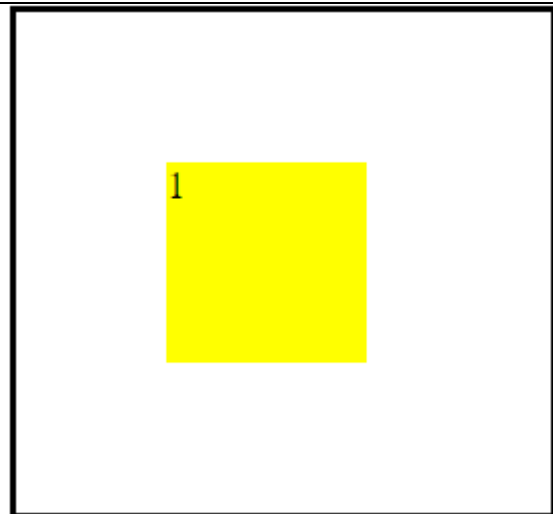
In html

```
<div class="box-container">  
  <div class="box box-1">1</div>  
</div>
```

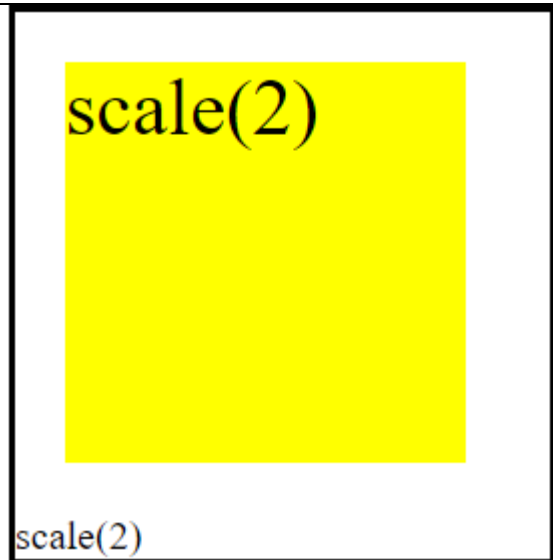
Transformation Attributes:

```
.box {  
  height: 50px;  
  width: 50px;  
  margin: 5px  
  background-color: yellow;  
}
```

No transformation attributes



```
.box-1 {  
  transform: scale(2);  
}
```



```
.box-1 {  
  transform: scaleX(2);  
}
```

scaleX(2)

scaleX(2)

```
.box-3 {  
  transform: scaleY(2);  
}
```

scaleY(2)

scaleY(2)

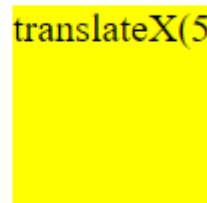
```
.box-4 {  
  transform: rotate(25deg);  
}
```

rotate(25deg)

rotate(25deg)

```
.box-5 {  
  transform: translateX(55px);  
}
```

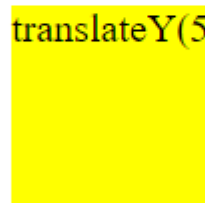
translateX(55px)



translateX(55px)

```
box-6 {  
  transform: translateY(55px);  
}
```

translateY(55px)



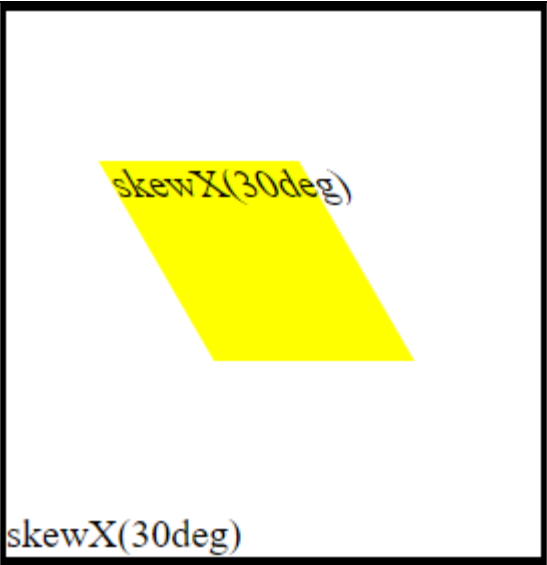
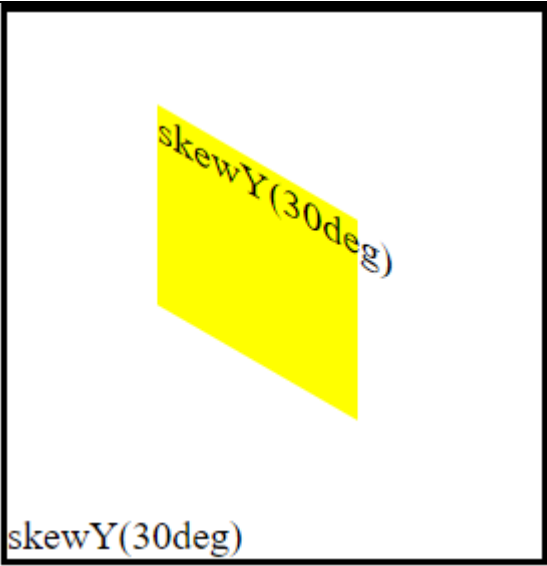
translateY(55px)

```
box-7 {  
  transform: skew(30deg,  
20deg);  
}
```

skew(30deg,
20deg)



skew(30deg, 20deg)

<pre>box-8 { transform: skewX(30deg); }</pre>	
<pre>box-9 { transform: skew(30deg); }</pre>	

Function	Description
<i>translate(x,y)</i>	Defines a 2D translation, moving the element along the X- and the Y-axis
<i>translateX(n)</i>	Defines a 2D translation, moving the element along the X-axis
<i>translateY(n)</i>	Defines a 2D translation, moving the element along the Y-axis
<i>scale(x,y)</i>	Defines a 2D scale transformation, changing the elements width and height
<i>scaleX(n)</i>	Defines a 2D scale transformation, changing the element's width
<i>scaleY(n)</i>	Defines a 2D scale transformation, changing the element's height
<i>rotate(angle)</i>	Defines a 2D rotation, the angle is specified in the parameter
<i>skew(x-angle,y-angle)</i>	Defines a 2D skew transformation along the X- and the Y-axis
<i>skewX(angle)</i>	Defines a 2D skew transformation along the X-axis
<i>skewY(angle)</i>	Defines a 2D skew transformation along the Y-axis

Transition:

CSS transitions allows you to change property values smoothly, over a given duration.

To create a transition effect, you must specify two things:

- the CSS property you want to add an effect to
- the duration of the effect

NOTE: if the duration part is not specified, the transition will have no effect, because the default value is 0.

Specify the Speed Curve of the Transition

The transition-timing-function property specifies the speed curve of the transition effect. The transition-timing-function property can have the following values:

Function	Description
ease	specifies a transition effect with a slow start, then speed increases and finally slows down (this is default)
linear	specifies a transition effect with same speed from start to end
ease-in	specifies a transition effect with a slow start
ease-out	specifies a transition effect with slow end
ease-in-out	specifies a transition effect with a slow start and end
cubic-bezier	lets' you define your own values in a cubic-bezier function

Ex: 01

In css:

```
div {  
  width: 100px;  
  height: 100px;  
  background: red;  
  transition: width 2s;  
}  
  
#div1 {transition-timing-  
function: linear;}  
#div2 {transition-timing-  
function: ease;}  
#div3 {transition-timing-  
function: ease-in;}
```

In html:

```
<div id="div1">linear</div><br>  
<div id="div2">ease</div><br>  
<div id="div3">ease-in</div><br>  
<div id="div4">ease-out</div><br>  
<div id="div5">ease-in-out</div><br>
```

```
#div4 {transition-timing-  
function: ease-out;}  
#div5 {transition-timing-  
function: ease-in-out;}  
  
div:hover {  
    width: 300px;  
}
```

Property	Description
transition	A shorthand property for setting the four transition properties into a single property
transition-delay	Specifies a delay (in seconds) for the transition effect
transition-duration	Specifies how many seconds or milliseconds a transition effect takes to complete
transition-property	Specifies the name of the CSS property the transition effect is for
transition-timing-function	Specifies the speed curve of the transition effect

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Transitions/Using_CSS_transitions

<https://css-tricks.com/almanac/properties/t/transition/>

<https://www.the-art-of-web.com/css/timing-function/>

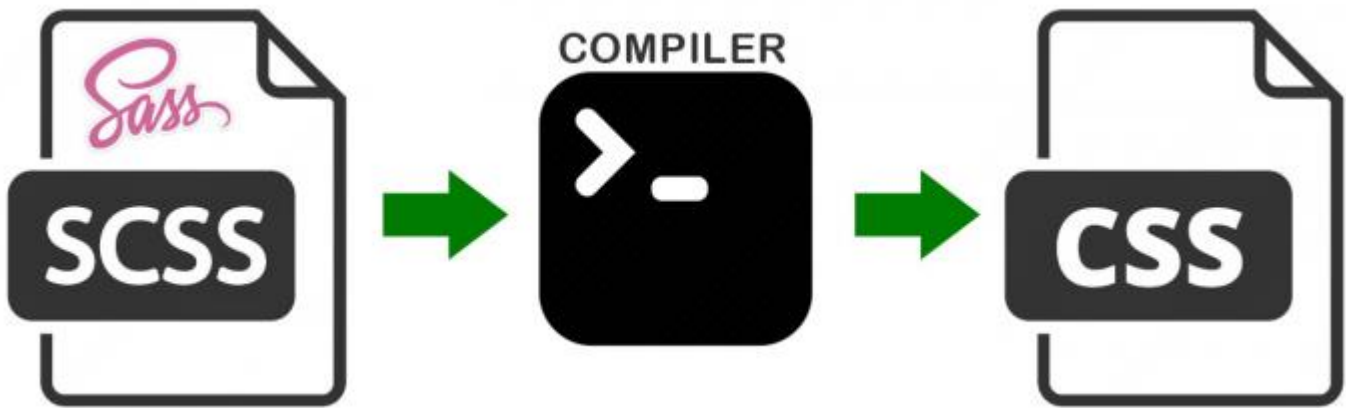
SASS:

Syntactically Awesome Style Sheet – is a pre-processor which takes some input and convert that into an understandable output.

In short, it is a CSS preprocessor, which adds special features such as variables, nested rules and mixins (sometimes referred to as syntactic sugar) into regular CSS. The aim is to make the coding process simpler and more efficient.

CSS Preprocessor:

A CSS preprocessor is a scripting language that extends CSS by allowing developers to write code in one language and then compile it into CSS. Sass is perhaps the most popular preprocessor around right now, but other well-known examples include Less and Stylus.



We need to install an extension, Live Sass Compiler (Ritwick Dey). Once this is installed, create a CSS file, mySassFile.scss and when you save it, in VS Code, it would create two additional files automatically;

- mySassFile.css
- mySassFile.css.map

mySassFile.css is the file created by the compiler which will have all the code written by you in **.sass** file properly into **.css** file.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
Live Sass Compile
Change Detected...
mySassFile.scss
-----
Generated :
h:\Projects\AttainU\au16-lecture-coding-material\week-16\day-3-transform-transition\mySassFile.css
h:\Projects\AttainU\au16-lecture-coding-material\week-16\day-3-transform-transition\mySassFile.css.map
-----
Watching...
-----
```


Variables:

Variables are a way to store information that you can re-use later. With Sass, you can store information in variables, like:

- strings
- numbers
- colors
- booleans
- lists
- nulls

Sass uses the \$ symbol, followed by a name, to declare variables:

<pre>\$font-stack: Helvetica, sans-serif; \$primary-color: #333; body { font: 100% \$font-stack; color: \$primary-color; }</pre>	<pre>body { font: 100% Helvetica, sans-serif; color: #333; }</pre>
--	--

When the Sass is processed, it takes the variables we define for the **\$font-stack** and **\$primary-color** and outputs normal CSS with our variable values placed in the CSS. This can be extremely powerful when working with brand colors and keeping them consistent throughout the site.

If we don't want to use Sass functionality, then we can use **var()** syntax.

```
:root {
  --blue: #6495ed;
  --white: #faf0e6;
}
h2 { border-bottom: 2px solid var(--blue); }

.container {
  color: var(--blue);
  background-color: var(--white);
  padding: 15px;
}
```

Nesting:

When writing HTML, you've probably noticed that it has a clear nested and visual hierarchy. CSS, on the other hand, doesn't.

Sass will let you nest your CSS selectors in a way that follows the same visual hierarchy of your HTML. Here's an example of some typical styles for a site's navigation:

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

You'll notice that the **ul**, **li**, and **a** selectors are nested inside the **nav** selector. This is a great way to organize your CSS and make it more readable.

Many CSS properties have the same prefix, like font-family, font-size and font-weight or text-align, text-transform and text-overflow.

<pre>font: { family: Helvetica, sans-serif; size: 18px; weight: bold; } text: { align: center; transform: lowercase; overflow: hidden; }</pre>	<pre>font-family: Helvetica, sans-serif; font-size: 18px; font-weight: bold; text-align: center; text-transform: lowercase; text-overflow: hidden;</pre>
--	---

Modules:

You don't have to write all your Sass in a single file. You can split it up however you want with the `@use` rule. This rule loads another Sass file as a *module*, which means you can refer to its variables, mixins, and functions in your Sass file with a namespace based on the filename. Using a file will also include the CSS it generates in your compiled output!

```
// _base.scss
$font-stack: Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}

// styles.scss
@use 'base';

.inverse {
  background-color: base.$primary-color;
  color: white;
}
```

```
body {
  font: 100% Helvetica, sans-serif;
  color: #333;
}

.inverse {
  background-color: #333;
  color: white;
}
```

Extend/Inheritance

This is one of the most useful features of Sass. Using **@extend** lets you share a set of CSS properties from one selector to another. It helps keep your Sass very DRY. In our example we're going to create a simple series of messaging for errors, warnings and successes using another feature which goes hand in hand with extend, placeholder classes. A placeholder class is a special type of class that only prints when it is extended, and can help keep your compiled CSS neat and clean.

We cannot use inheritance in CSS, but we can write it down in sass and later covert it into CSS.

Operators

Doing math in your CSS is very helpful. Sass has a handful of standard math operators like +, -, *, /, and %. In our example we're going to do some simple math to calculate widths for an **aside** & **article**

```
.container {  
  width: 100%;  
}  
  
article[role="main"] {  
  float: left;  
  width: 600px / 960px * 100%;  
}  
  
aside[role="complementary"] {  
  float: right;  
  width: 300px / 960px * 100%;  
}
```

```
.container {  
  width: 100%;  
}  
  
article[role="main"] {  
  float: left;  
  width: 62.5%;  
}  
  
aside[role="complementary"] {  
  float: right;  
  width: 31.25%;  
}
```

We've created a very simple fluid grid, based on 960px. Operations in Sass let us do something like take pixel values and convert them to percentages without much hassle.

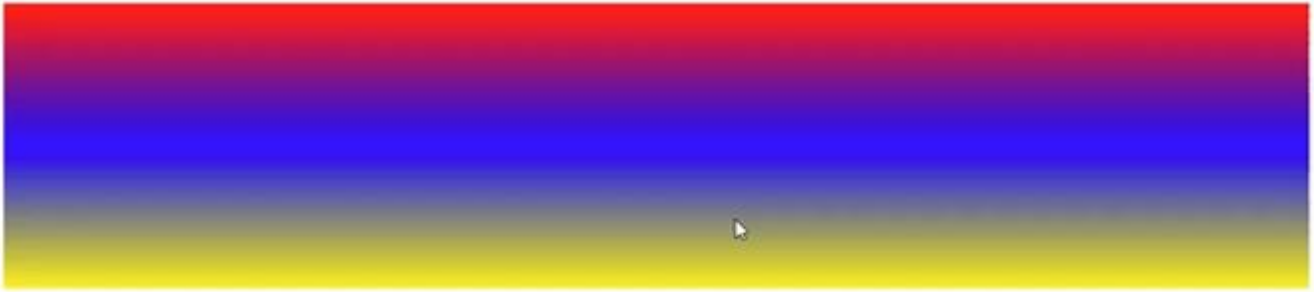
Gradient:

Linear Gradient is not a property, it is a function. It produces value.

```
.box {  
  height: 50px;  
  background-image: linear-gradient (red, blue)  
}
```



```
.box {  
  height: 50px;  
  background-image: linear-gradient (red, blue, yellow)  
}
```



Direction – *to right*:

```
.box {  
  height: 50px;  
  background-image: linear-gradient (to right, red, blue, yellow)  
}
```



Direction – *to left*:

```
.box {  
  height: 50px;  
  background-image: linear-gradient (to right, red, blue, yellow)  
}
```



We can provide a particular degree as well to the direction.

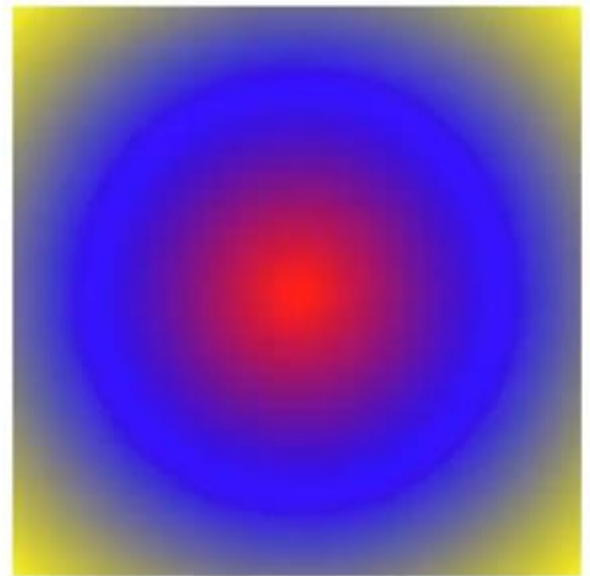
```
.box {  
  height: 50px;  
  background-image: linear-gradient (60deg, red, blue, yellow)  
}
```



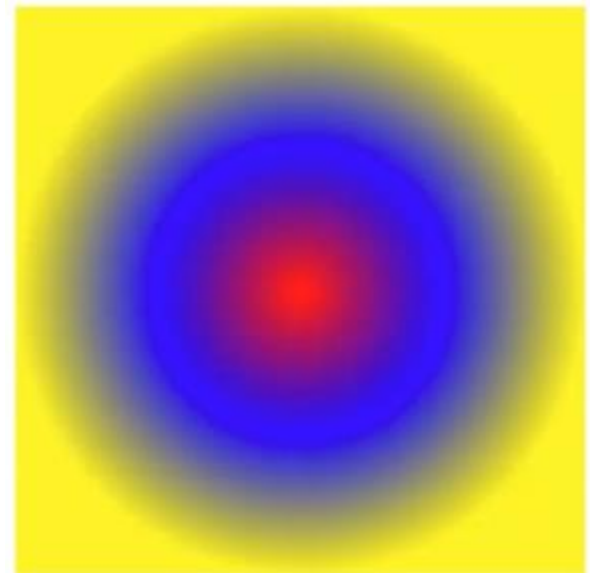
If we put angle as 90°, it would be **to right** direction

Radial Gradient:

```
.box {  
  height: 50px;  
  width: 50px;  
  background-image: radial-gradient  
(red, blue, yellow)  
}
```



```
.box {  
  height: 50px;  
  width: 50px;  
  background-image: radial-gradient  
(closest-side, red, blue, yellow)  
}
```



```
.box {  
    height: 50px;  
    width: 50px;  
    background-image: linear-gradient  
(90deg, red 80%, blue, yellow)  
}
```



In this the linear gradient has 80% and the remaining 20% is shared between blue and yellow.

```
.box {  
    height: 50px;  
    background: linear-gradient (135deg, orange 60%, cyan);  
}
```



Sharp or Stripped effect

```
.box {  
    height: 50px;  
    background: linear-gradient (to right,  
    red 20%, orange 20% 40%, yellow 40% 60%, green 60% 80%, blue 80%);  
}
```



[https://developer.mozilla.org/en-US/docs/Web/CSS/linear-gradient\(\)](https://developer.mozilla.org/en-US/docs/Web/CSS/linear-gradient())
https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Images/Using_CSS_gradients