SASS
compile css into css, without installing on your environment
http://sass.js.org/

```
@mixin transform-style($style){
   transform-style: $style;
  -moz-transform-style: $style;
  -o-transform-style: $style;
  -ms-transform-style: $style;
  -webkit-transform-style: $style;
}
@include transform-style(preserve-3d);
```

```
@mixin transition($time){
  transition: $time;
  -webkit-transition: $time;
  -moz-transition: $time;
  -o-transition: $time;
}
@include transition(0.4s);
```

```
@mixin stripes($direction, $width-percent, $stripe-color, $stripe-background:
#FFF) {
  background: repeating-linear-gradient(
    $direction,
    $stripe-background,
    $stripe-background ($width-percent - 1),
    $stripe-color 1%,
    $stripe-background $width-percent
  );
}
$college-ruled-style: (
    direction: to bottom,
    width-percent: 15%,
    stripe-color: blue,
    stripe-background: white
);

 @include stripes($college-ruled-style...);
```

```
@mixin photo-content($file) {
  content: url(#{$file}.jpg);
  object-fit: cover;
}

.photo {
        @include photo-content('titanosaur');
        width: 60%;
        margin: 0px auto;
 }

________________________-

@mixin stripes($direction, $width-percent, $stripe-color, $stripe-background:
#FFF) {
  background: repeating-linear-gradient(
    $direction,
    $stripe-background,
    $stripe-background ($width-percent - 1),
    $stripe-color 1%,
    $stripe-background $width-percent
  );
}
$college-ruled-style: (
    direction: to bottom,
    width-percent: 15%,
    stripe-color: blue,
    stripe-background: white
);
$stripe-properties: to bottom, 15%, blue, white;
@mixin transform($transformation) {
  transform: $transformation;
  -webkit-transform: $transformation;
  -moz-transform: $transformation;
  -ms-transform: $transformation;
  -o-transform: $transformation;
}

@mixin photo-content($file) {
  content: url(#{$file}.jpg);
  object-fit: cover;
}

//Add your own mixins here
@mixin backface-visibility($visibility: hidden) { //Add an argument
  backface-visibility: $visibility;
  -webkit-backface-visibility: $visibility;
  -moz-backface-visibility: $visibility;
  -ms-backface-visibility: $visibility;
  -o-backface-visibility: $visibility;
}
@mixin transform-style($style){
   transform-style: $style;
  -moz-transform-style: $style;
  -o-transform-style: $style;
  -ms-transform-style: $style;
  -webkit-transform-style: $style;
}
@mixin transition($time){
  transition: $time;
  -webkit-transition: $time;
  -moz-transition: $time;
  -o-transition: $time;
}
.notecard {
  width: 300px;
  height: 180px;
  border: 1px solid black;
  display: inline-block;
  margin: 20px;
  font-family: Roboto, sans-serif;
  box-shadow: 1px 1px 2px 2px rgba(0,0,0,.2);
  &:hover{
  @include transform (rotatey(-180deg));
  }
   @include transform-style(preserve-3d);
      @include transition(0.4s);
  .front, .back {
    width: 100%;
    height: 100%;
    position: absolute;
    @include backface-visibility(hidden);
  }

  .front {
  z-index: 3;
  font-size: 20px;

    .word {
      display: block;
      text-align: center;
      position: relative;
      top: 40%;
    }
  }


  .back {
  z-index: 1;
  word-wrap: break-word;
  line-height: 1.6;

    .definition {
      width: 100%;
      height: 100%;
         @include stripes($college-ruled-style...);
       @include stripes($stripe-properties...);
      .photo {
         @include photo-content('titanosaur');
        width: 60%;
        margin: 0px auto;

      }
    }
    @include transform(rotatey(-180deg));
  }
}
```

- The & selector gets assigned the value of the parent at the point where the mixin is included.

- If there is no parent selector, then the value is null and Sass will throw an error.

```
@mixin text-hover($color){
  &:hover {
      color: $color;
  }
}
```

In the above, the value of the parent selector will be assigned based on the parent at the point where it is invoked.

```
  .word { //SCSS:
    display: block;
    text-align: center;
    position: relative;
    top: 40%;
    @include text-hover(red);
  }
```

The above will compile to the following CSS:

```
 .word{
   display: block;
   text-align: center;
   position: relative;
   top: 40%;
 }
 .word:hover{
   color: red;
 }
```

- *Mixins* are a powerful tool that allow you to keep your code DRY. Their ability to take in arguments, assign default values to those arguments, and accept said arguments in whatever format is most readable and convenient for you makes the mixin Sass's most popular directive.

- The & selector* is a Sass construct that allows for expressive flexibility by referencing the parent selector when working with CSS psuedo elements and classes.

- *String interpolation* is the glue that allows you to insert a string in the middle of another when it is in a variable format. Its applications vary, but the ability to interpolate is especially useful for passing in file names.

- In Sass, the function fade-out makes a color more transparent by taking a number between 0 and 1 and **decreasing opacity**, or the alpha channel, by that amount:

$color: rgba(39, 39, 39, 0.5);
$amount: 0.1;
$color2: **fade-out**($color, $amount);//rgba(39, 39, 39, 0.4)
$color2: **fade-in**($color, $amount); //rgba(55,7,56, 0.6)

he function adjust-hue($color, $degrees) changes the hue of a color by taking color and a number of degrees (usually between -360 degrees and 360 degrees), and rotate the color wheel by that amount.

Sass also allows us to perform mathematical functions to compute measurements— including colors.
Here is how Sass computes colors:

- The operation is performed on the red, green, and blue components.
- It computes the answer by operating on every two digits.

$color: #010203 + #040506;
The above would compute piece-wise as follows:
01 + 04 = 05
02 + 05 = 07
03 + 06 = 09
and compile to:
color: #050709;

```
font-size: $width/6/2;
$list: (orange, purple, teal);

//Add your each-loop here
@each $item in $list {
  .#{$item} {
    background: $item;
  }
}
```

---

main.scss

```
$total: 10; //Number of .ray divs in our html
$step: 360deg / $total; //Used to compute the hue based on color-wheel


.ray {
  height: 30px;
}

//Add your for-loop here:
@for $i from 1 to $total {
  .ray:nth-child(#{$i}) {
    background:adjust-hue(blue, $i * $step);;
   }
}
```

main.css

```
.ray {
  height: 30px;
}

.ray:nth-child(1) {
  background: #9900ff;
}

.ray:nth-child(2) {
  background: #ff00cc;
}

.ray:nth-child(3) {
  background: #ff0033;
}

.ray:nth-child(4) {
  background: #ff6600;
}

.ray:nth-child(5) {
  background: yellow;
}

.ray:nth-child(6) {
  background: #66ff00;
}

.ray:nth-child(7) {
  background: #00ff33;
}

.ray:nth-child(8) {
  background: #00ffcc;
}

.ray:nth-child(9) {
  background: #0099ff;
}

.ray:nth-child(10) {
  background: blue;
}
```

In Sass, if() is a function that can only branch one of two ways based on a condition. You can use it inline, to assign the value of a property:

```
width: if( $condition, $value-if-true, $value-if-false);
```

For cases with more than two outcomes, the @if, @else-if, and @else directives allow for more flexibility.

```
@mixin deck($suit) {
 @if($suit == hearts || $suit == spades){
   color: blue;
 }
 @else-if($suit == clovers || $suit == diamonds){
   color: red;
 }
 @else{
   //some rule
 }
}

$total: 10; //Number of .ray divs in our html
$step: 360deg / $total; //Used to compute the hue based on color-wheel
```

```
.ray {
  height: 30px;
}

//Add your for-loop here:
@for $i from 1 through $total {
  .ray:nth-child(#{$i}) {
    background: adjust-hue(blue, $i * $step);
    width: if($i % 2 == 0, 300px, 350px);
  margin-left: if($i % 2 == 0, 0px, 50px);
   }

}
```

---

to import a file named **_variables.scss**, add the following line of code:

```
 @import "variables";
```

```
.lemonade {
  border: 1px yellow;
  background-color: #fdd;
}
.strawberry {
  @extend .lemonade;
  border-color: pink;
}
```

***Placeholders* prevent rules from being rendered to CSS on their own and only become active once they are extended anywhere an id or class could be extended.**

```
 a%drink {
    font-size: 2em;
    background-color: $lemon-yellow;
 }

 .lemonade {
  @extend %drink;
  //more rules
 }
```

would translate to

```
  a.lemonade {
    font-size: 2em;
    background-color: $lemon-yellow;
 }

.lemonade {
  //more rules
}
```

```
@mixin no-variable {
  font-size: 12px;
  color: #FFF;
  opacity: .9;
}

%placeholder {
  font-size: 12px;
  color: #FFF;
  opacity: .9;
}

span {
  @extend %placeholder;
}

div {
  @extend %placeholder;
}

p {
  @include no-variable;
}

h1 {
  @include no-variable;
}
```

would compile to:

```
span, div{
  font-size: 12px;
  color: #FFF;
  opacity: .9;
}

p {
  font-size: 12px;
  color: #FFF;
  opacity: .9;
  //rules specific to ps
}

h1 {
  font-size: 12px;
  color: #FFF;
  opacity: .9;
  //rules specific to ps
}
```

- Try to only create mixins that take in an argument, otherwise you should extend.

- Always look at your CSS output to make sure your extend is behaving as you intended.