Who has heard of Saas?

Who has used Saas?

Introduction to Saas

- Saas stands for Syntactically Awesome Stylesheets which was developed in 2007.
- It is an extension of CSS .
- Saas is a CSS preprocessor.
- Saas is completely compatible with all versions of CSS.
- Saas reduces repetition and therefore saves time.
- Sass was designed by Hampton Catlin and developed by Natalie Weizenbaum in 2006.

How it Works

- Sass compiles into CSS files.
- Two formatting conventions

    1. .sass
    2. .scss

Features

Sass has 5 primary features:-

1. Variables
2. Nesting
3. Mixins
4. Partials
5. Import

Variables

Variables are a way to store information which you can use later.

Saas uses a $ symbol, followed by a name, to declare a variable.

```
$textColor: white;
$fontSize:24px;

body{
  font-size:$fontSize;
 color: $textColor;
}
```

Nesting

In CSS, the rules are defined one by one, but Saas allows you to define nested properties.

Many css properties have the same prefix like font-family,font-size,font-weight.

With saas you can write them as nested properties.

scss > main.scss > ...

```scss
$textColor: white;
$fontSize:24px;

.div1{
  .h1{
    color: $textColor;
  }
  .p{
    font-size: $fontSize;
  }
}
```

> css > # main.css > ...

```css
.div1 .h1 {
  color: white;
}

.div1 .p {
  font-size: 24px;
}
```

Mixins

@mixin directive allows you to create css code that be reusable throughout the website.

@include directive allows you to use the mixin.

x.html | main.scss × | # main.css

scss > main.scss > ...

```scss
@mixin commonStyle(){
  padding-top: 20px;
  background-color: green;
  text-align: center;

}


.h1{
  @include commonStyle();
  border-width: 2px;
  border-color: red;
}
```

ndex.html | main.scss | # main.css

> css > # main.css > ...

```
1
2
3   .h1 {
4     padding-top: 20px;
5     background-color: green;
6     text-align: center;
7     border-width: 2px;
8     border-color: red;
9   }
0
1
2
3
4
5
```

Partials

By default Sass transpiles all the .scss files directly. However, if you want to import a file you do not need to transpiled directly.

Sass has a mechanism for this. If you start the filename with an underscore, Saas will not compile it. Files named this way are called partials in Sass.

> scss > _abc.scss > ...

```scss
@mixin fun2(){
    background-color: red;
    color: white;
    width:100px;
    height:100px;
}
```

scss > main1.scss > ...

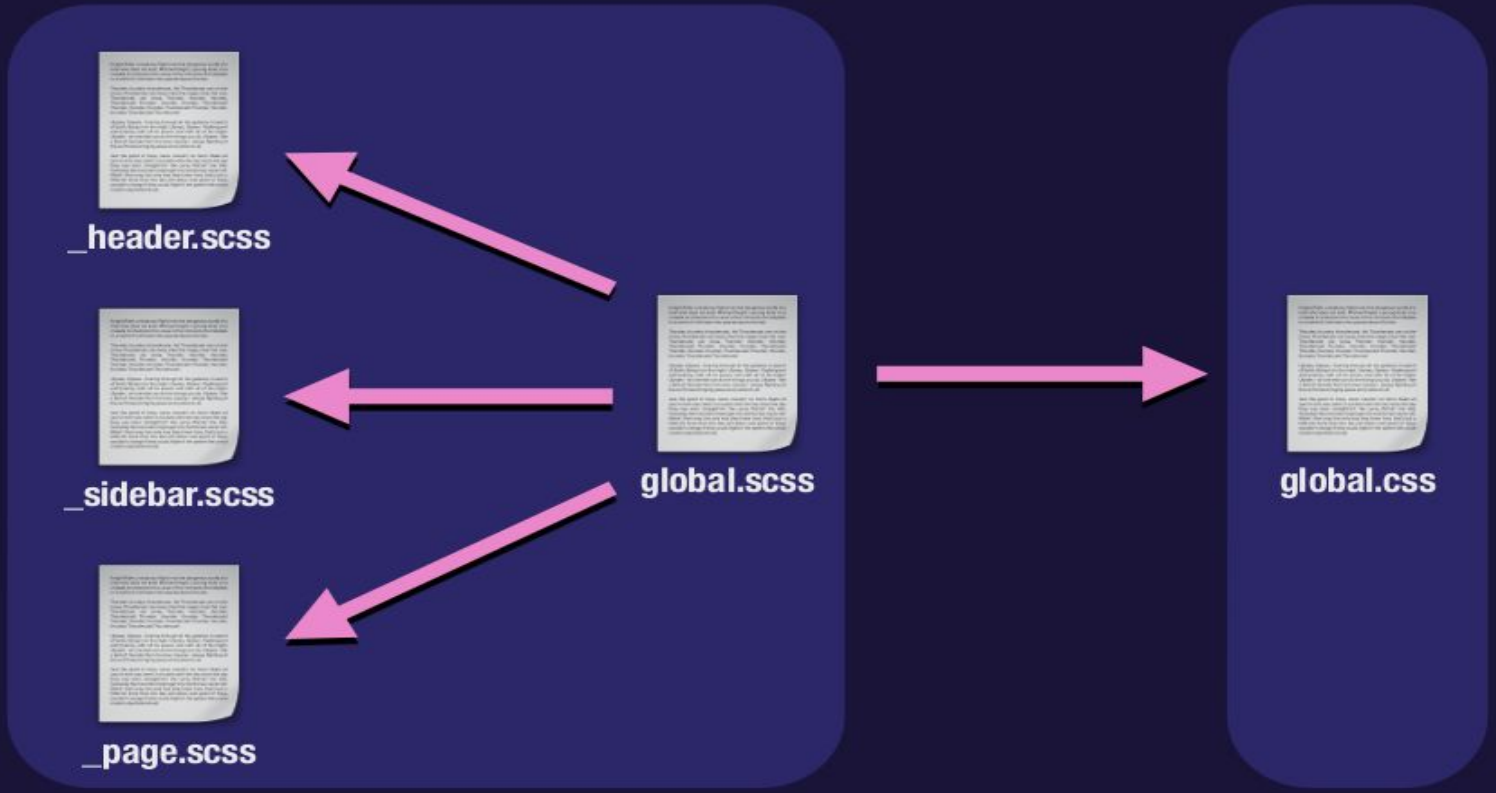
```scss
@import "abc";

.h1{
    @include fun2();
    text-align: center;
}
```

ex.html main.scss _abc.scs

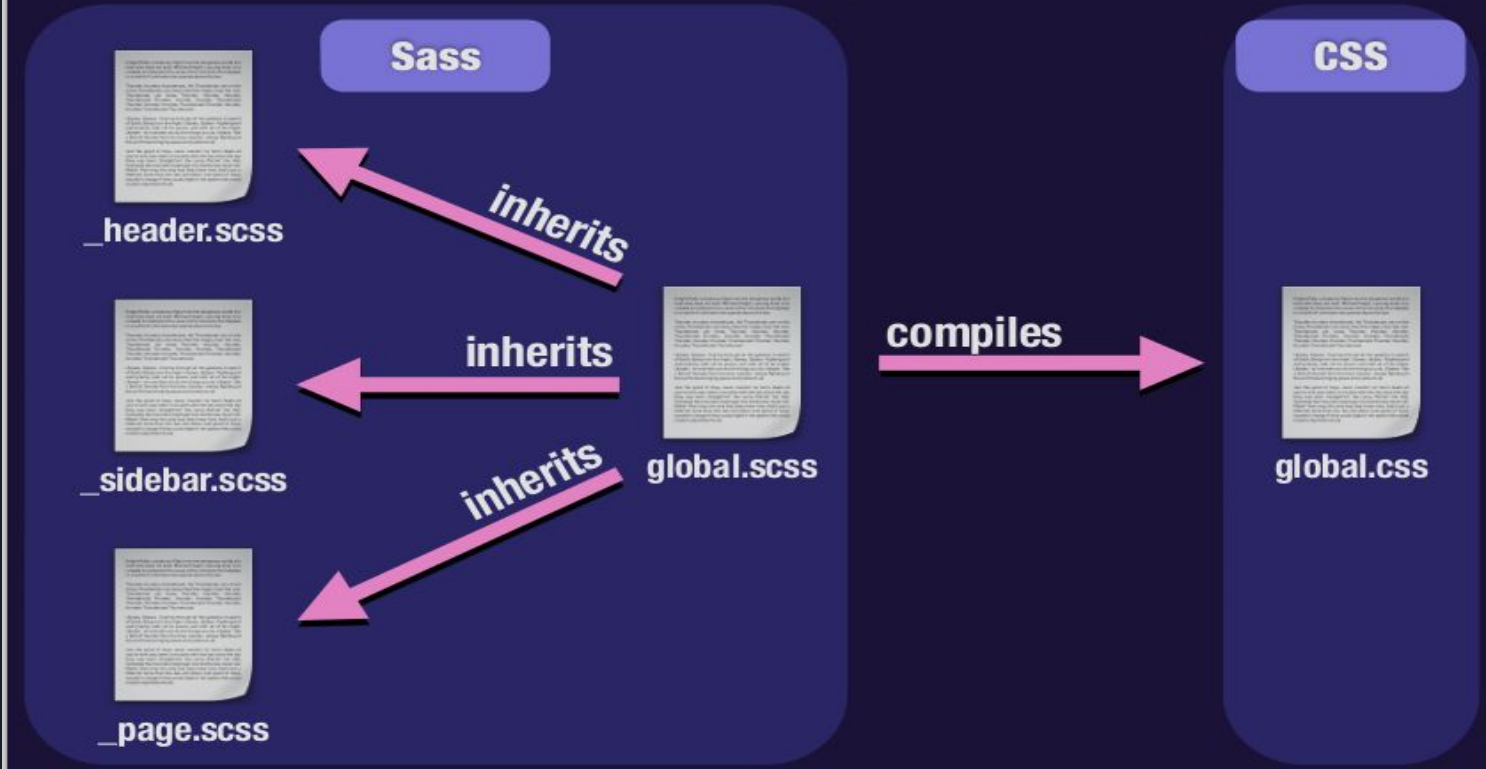
css > # main1.css > ...

```css
.h1 {
    background-color: red;
    color: white;
    width: 100px;
    height: 100px;
    text-align: center;
}
```

Visualizing Partials



Visualizing Partials



Operators

1. Arithmetic Operator :- Sass provide arithmetic operation over styling

   1. Addition :- We can add value . Just to make sure given value are in same format

```
p {
    font-size: 10px + 2em; // *error: incompatible unit
    font-size: 10px + 6px; // 16px
    font-size: 10px + 6;   // 16px
}
```

Operators

2. Subtraction : - We can subtract different values

```
div {
    height: 12% - 2%;
    margin: 4rem - 1;
}
```

Operators

3. Multiplication : - We can multiply value

```
p {
    width: 10px * 10px;              // *error
    width: 10px * 10;                // 100px
    width: 1px * 5 + 5px;            // 10px
    width: 5 * (5px + 5px);          // 50px
    width: 5px + (10px / 2) * 3;     // 20px
}
```

Operators

4. Division :- We can divide values

```
p {
    top: 16px / 24px
    top: (16px / 24px)
    top: #{$var1} / #{$var2};
    top: $var1 / $var2;
    top: random(4) / 5;
    top: 2px / 4px + 3px
}
```

Comparison Operators

These are the comparison operator which are used :-

Operator	Example	Description
==	x == y	returns true if x and y are equal
!=	x != y	returns true if x and y are not equal
>	x > y	returns true if x is greater than y
<	x < y	returns true if x is less than y
>=	x >= y	returns true if x is greater than or equal to y
<=	x <= y	returns true if x is less than or equal to y

Comparison Operator

We can try to write @mixins which will choose padding size if it is greater

```
@mixin spacing($padding, $margin) {
    @if ($padding > $margin) {
        padding: $padding;
    } @else {
        padding: $margin;
    }
}

.container {
    @include spacing(10px, 20px);
}
```

Logical Operator

Logical Operator which are used for simple Logical Operations .

Operator	Example	Description
and	x and y	returns true if x and y are true
or	x or y	returns true if x or y is true
not	not x	returns true if x is not true

Logical Operator

We can use Sass Logical Operator to create button color class that changes its background color according to its width.

```
@mixin button-color($height, $width) {
    @if(($height < $width) and ($width >= 35px)) {
        background-color: blue;
    } @else {
        background-color: green;
    }
}

.button {
    @include button-color(20px, 30px)
}
```

Control Flow Statement

SCSS has function() and @directive (also known as Rules) .

1. Function :- A function usually has parenthesis appended to the of function name.
2. Directive :- A directive or rule starts with @.

Type of function() and @directiive are given below :-

1. If() :- If() is a function .This function will return one of the two values according to the conditions.

Control Flow Statement

IF function usage :- Here , if function will return one value that is 1px among these two values according to the conditions given.

```
/* Using if() function */
if(true, 1px, 2px) => 1px
if(false, 1px, 2px) => 2px
```

Control Flow Statement

2. @IF directive :- This directive is used to branch out based on the conditions.

   @IF directive usage :-

```
/* Using @if directive */
p {
  @if 1 + 1 == 2 { border: 1px solid;  }
  @if 7 < 5      { border: 2px dotted; }
  @if null       { border: 3px double; }
}
```

Control Flow Statement

3. @FOR directive :- @FOR directive is used for repeating CSS definition multiple times in a row .

@FOR directive Usage :- Here , for directive defines i from 1 to 5 .

```
@for $i from 1 through 5 {
    .definition-#{$i} { width: 10px * $i; }
}
```

Control Flow Statement

Output for FOR directive :-

```
.definition-1 { width: 10px; }
.definition-2 { width: 20px; }
.definition-3 { width: 30px; }
.definition-4 { width: 40px; }
.definition-5 { width: 50px; }
```

Control Flow Statement

4. @While directive :- @While directive is used for repeating CSS definition multiple times .

@While directive usage :-

```
$index: 5;
@while $index > 0 {
    .element-#{$index} { width: 10px * $index; }
    $index: $index - 1;
}
```

Control Flow Statement

While Directive Output : -

```
.element-5 { width: 50px; }
.element-4 { width: 40px; }
.element-3 { width: 30px; }
.element-2 { width: 20px; }
.element-1 { width: 10px; }
```

SASS Function

SASS Function :- Using SASS we can define simple function as we define in Simple Programming Language.

SASS Function :- Here three-hundred-px function will return 300 px .

```
@function three-hundred-px()  {
      @return 300px;
}

.name  {
      width: three-hundred-px();
      border: 1px solid gray;
      display: block;
      position: absolute;
}
```

SASS Directive

@Extend directive :- @extend directive lets you share a set of CSS properties from one selector to another selector .

@extend selector usage : -

```
.button-basic  {
  border: none;
  padding: 15px 30px;
  text-align: center;
  font-size: 16px;
  cursor: pointer;
}

.button-report  {
  @extend .button-basic;
  background-color: red;
}
```