tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

SASS (Syntactically Awesome Stylesheet) is a CSS pre-processor, which helps to reduce repetition with CSS and saves time. It is more stable and powerful CSS extension language that describes the style of document structurally. This tutorial covers the basics of SASS.

Audience

This tutorial will help both students as well as professionals, who want to make their websites or personal blogs more attractive.

Prerequisites

Before you proceed with this tutorial, we assume that you know:

- Basic word processing using any text editor.
- How to create directories and files.
- How to navigate through different directories.
- Internet browsing using popular browsers like Internet Explorer or Firefox.
- How to develop simple Web Pages using HTML or XHTML.

If you are new to HTML and XHTML, then we would suggest you to go through our HTML Tutorial or XHTML Tutorial first.

Copyright & Disclaimer

Table of Contents

1. SASS – Overview

What is SASS?

SASS (Syntactically Awesome Stylesheet) is a CSS pre-processor, which helps to reduce repetition with CSS and saves time. It is more stable and powerful CSS extension language that describes the style of a document cleanly and structurally.

It was initially designed by **Hampton Catlin** and developed by **Natalie Weizenbaum** in 2006. Later, **Weizenbaum** and **Chris Eppstein** used its initial version to extend the Sass with SassScript.

Why to Use SASS?

- It is a pre-processing language, which provides indented syntax (its own syntax) for CSS.
- It provides some features, which are used for creating stylesheets that allows writing code more efficiently and is easy to maintain.
- It is a super set of CSS, which means it contains all the features of CSS and is an open source pre-processor, coded in **Ruby**.
- It provides the document style in a good, structured format than flat CSS. It uses re-usable methods, logic statements and some of the built-in functions such as color manipulation, mathematics and parameter lists.

Features of SASS

- It is more stable, powerful, and compatible with versions of CSS.
- It is a super set of CSS and is based on JavaScript.
- It is known as syntactic sugar for CSS, which means it makes easier way for user to read or express the things more clearly.
- It uses its own syntax and compiles to readable CSS.
- You can easily write CSS in less code within less time.
- It is an open source pre-processor, which is interpreted into CSS.

Advantages of SASS

- It allows writing clean CSS in a programming construct.
- It helps in writing CSS quickly.
- It is a superset of CSS, which helps designers and developers work more efficiently and quickly.
- As Sass is compatible with all versions of CSS, we can use any available CSS libraries.
- It is possible to use nested syntax and useful functions such as color manipulation, mathematics and other values.

Disadvantages of SASS

- It takes time for a developer to learn new features present in this pre-processor.
- If many people are working on the same site, then should use the same preprocessor. Some people use Sass and some people use CSS to edit the files directly. Therefore, it becomes difficult to work on the site.
- There are chances of losing benefits of browser's built-in element inspector.

2. SASS – Installation

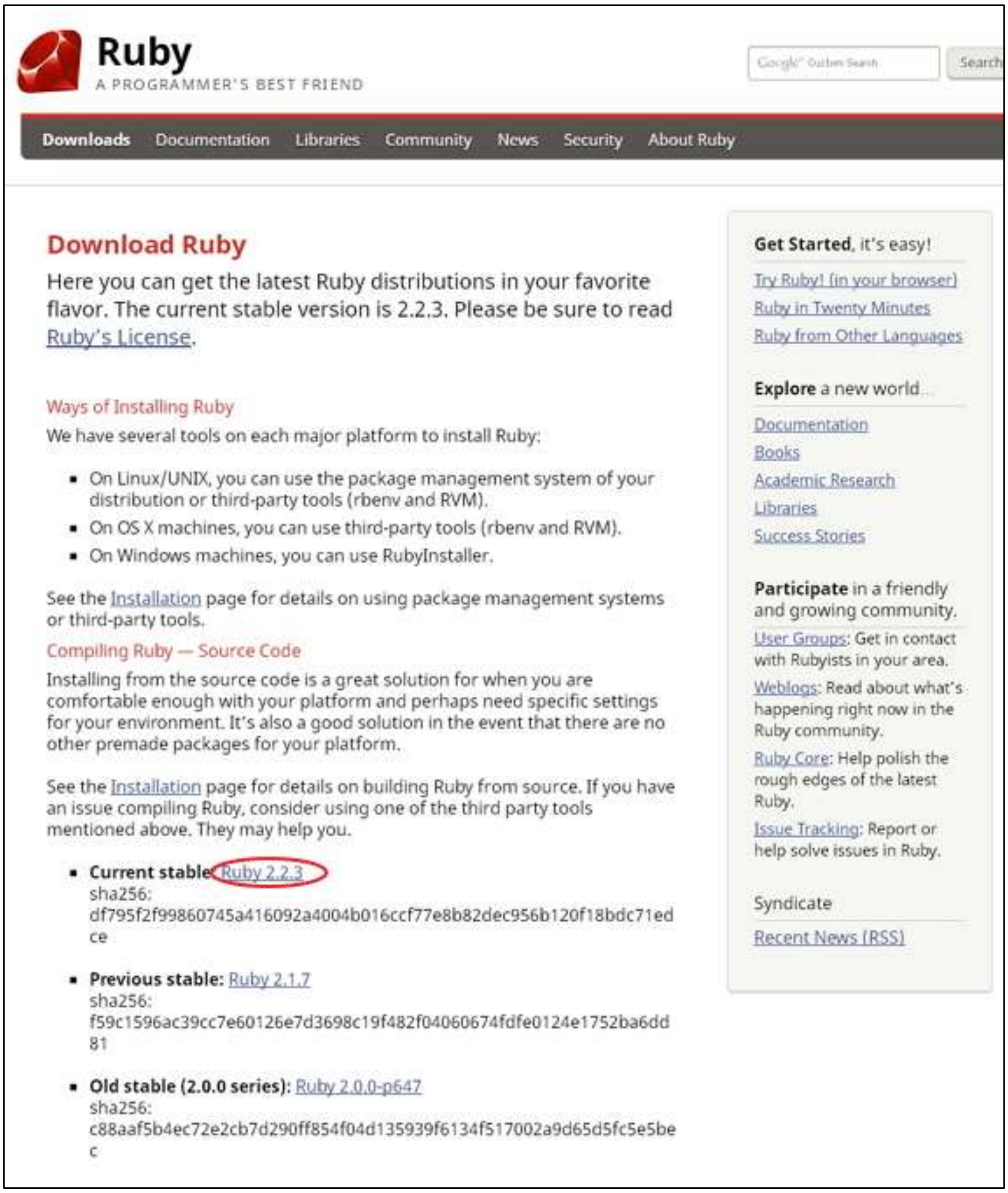
In this chapter, we will learn the step-by-step procedure to install Ruby, which is used for executing the SASS files.

System Requirements for SASS

- **Operating System :** Cross-platform
- **Browser Support :** IE (Internet Explorer 8+), Firefox, Google Chrome, Safari, Opera
- **Programming Language:** Ruby

Installation of Ruby

Step(1): Open the link https://www.ruby-lang.org/en/downloads/, you will see a screen as shown below:



Download Ruby

Here you can get the latest Ruby distributions in your favorite flavor. The current stable version is 2.2.3. Please be sure to read Ruby's License.

Ways of Installing Ruby

We have several tools on each major platform to install Ruby:

- On Linux/UNIX, you can use the package management system of your distribution or third-party tools (rbenv and RVM).
- On OS X machines, you can use third-party tools (rbenv and RVM).
- On Windows machines, you can use RubyInstaller.

See the Installation page for details on using package management systems or third-party tools.

Compiling Ruby — Source Code

Installing from the source code is a great solution for when you are comfortable enough with your platform and perhaps need specific settings for your environment. It's also a good solution in the event that there are no other premade packages for your platform.

See the Installation page for details on building Ruby from source. If you have an issue compiling Ruby, consider using one of the third party tools mentioned above. They may help you.

- **Current stable:** Ruby 2.2.3
  sha256:
  df795f2f99860745a416092a4004b016ccf77e8b82dec956b120f18bdc71ed ce
- **Previous stable:** Ruby 2.1.7
  sha256:
  f59c1596ac39cc7e60126e7d3698c19f482f04060674fdfe0124e1752ba6dd 81
- **Old stable (2.0.0 series):** Ruby 2.0.0-p647
  sha256:
  c88aaf5b4ec72e2cb7d290ff854f04d135939f6134f517002a9d65d5fc5e5be c

Get Started, it's easy!

Try Ruby! (in your browser)
Ruby in Twenty Minutes
Ruby from Other Languages

Explore a new world...

Documentation
Books
Academic Research
Libraries
Success Stories

Participate in a friendly and growing community.

User Groups: Get in contact with Rubyists in your area.
Weblogs: Read about what's happening right now in the Ruby community.
Ruby Core: Help polish the rough edges of the latest Ruby.
Issue Tracking: Report or help solve issues in Ruby.

Syndicate

Recent News (RSS)

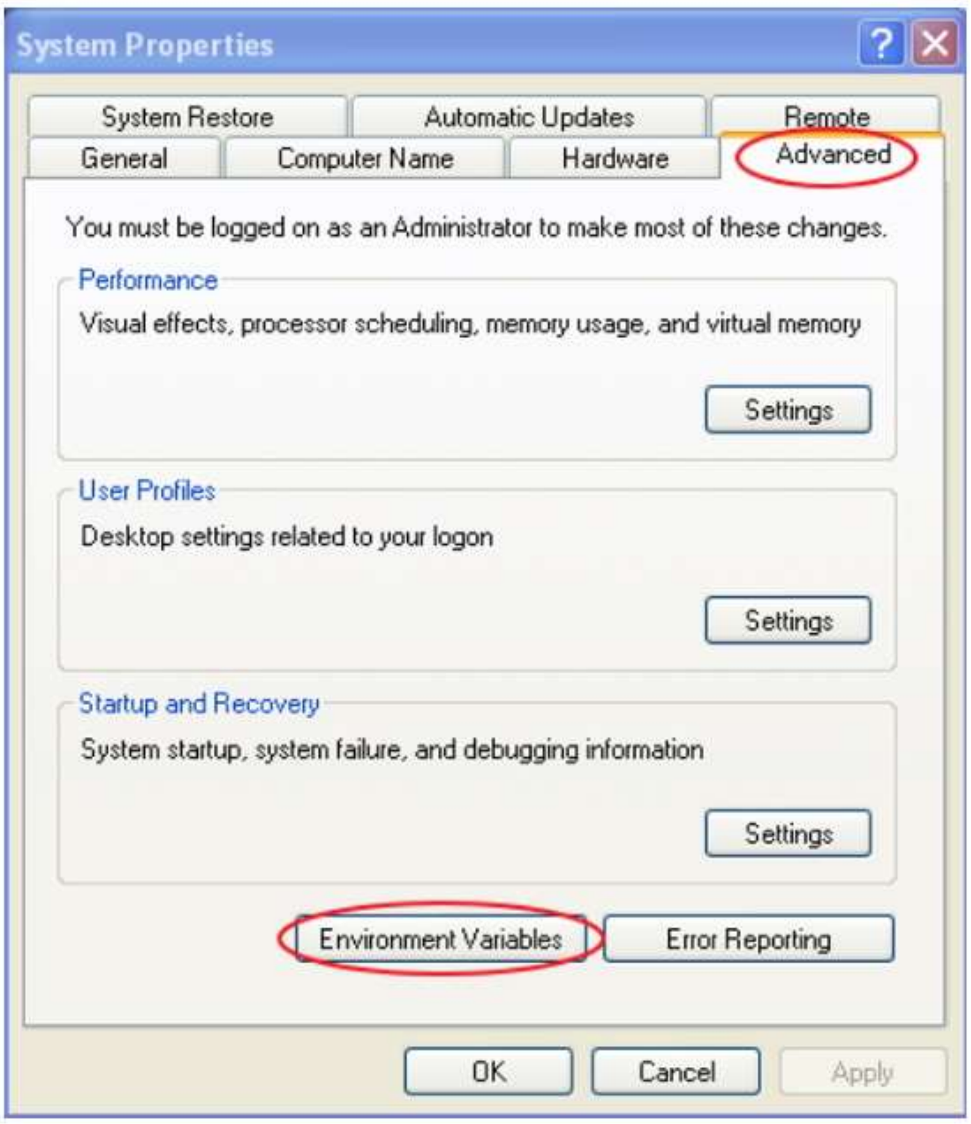
Download the *Current stable* version of the zip file.

Step(2): Next, run the setup to install **Ruby** on the System.

Step(3): Next, add Ruby bin folder to your *PATH User Variable* and *System Variable* to work with gem command.

Path User Variable:

- Right Click the **My Computer** icon.
- Select **Properties**.
- Next, click the **Advanced** tab and click **Environment Variables**.


System Properties

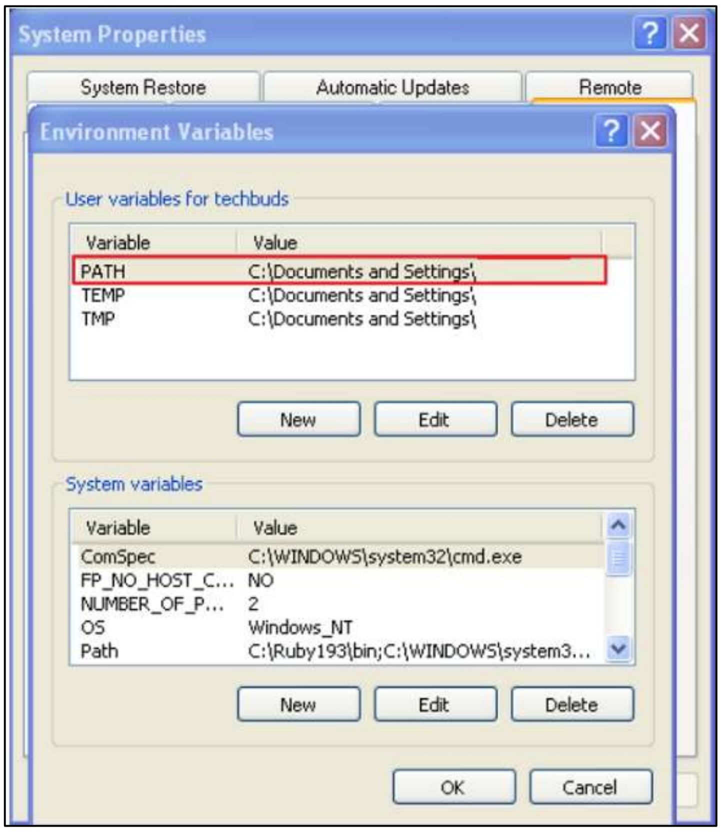
You must be logged on as an Administrator to make most of these changes.

Performance
Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles
Desktop settings related to your logon

Startup and Recovery
System startup, system failure, and debugging information

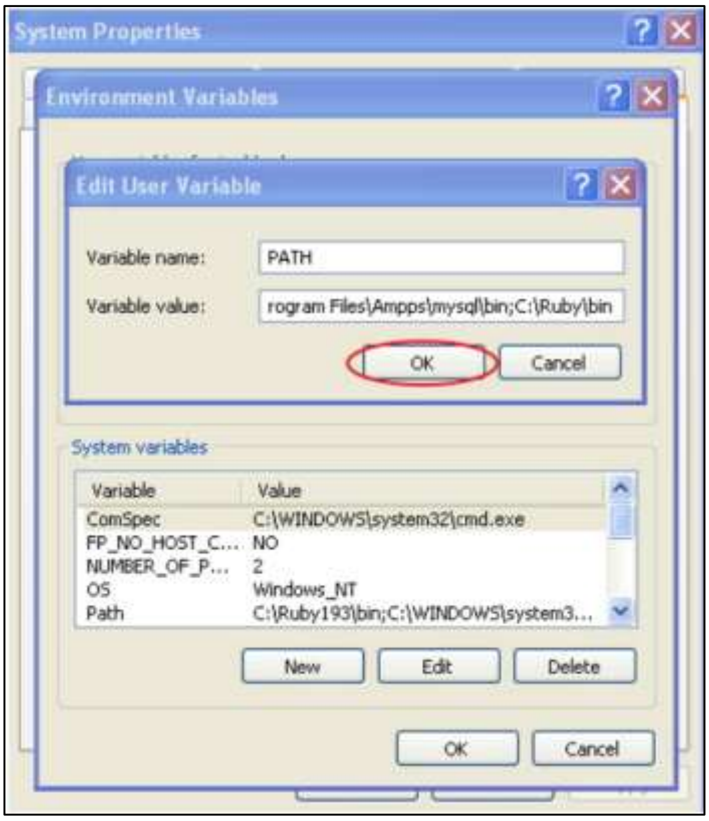
In the *Environment Variables* window, double click the *PATH* as shown in the screenshot given below-



Variable	Value
PATH	C:\Documents and Settings\
TEMP	C:\Documents and Settings\
TMP	C:\Documents and Settings\

Variable	Value
ComSpec	C:\WINDOWS\system32\cmd.exe
FP_NO_HOST_C...	NO
NUMBER_OF_P...	2
OS	Windows_NT
Path	C:\Ruby193\bin;C:\WINDOWS\system3...

You will get an *Edit User Variable* box as shown. Add ruby bin folder path in the *Variable value* field as **C:\Ruby\bin**. If path is already set for other files, then put semicolon after that and add the Ruby folder path as shown below.



Variable name: PATH

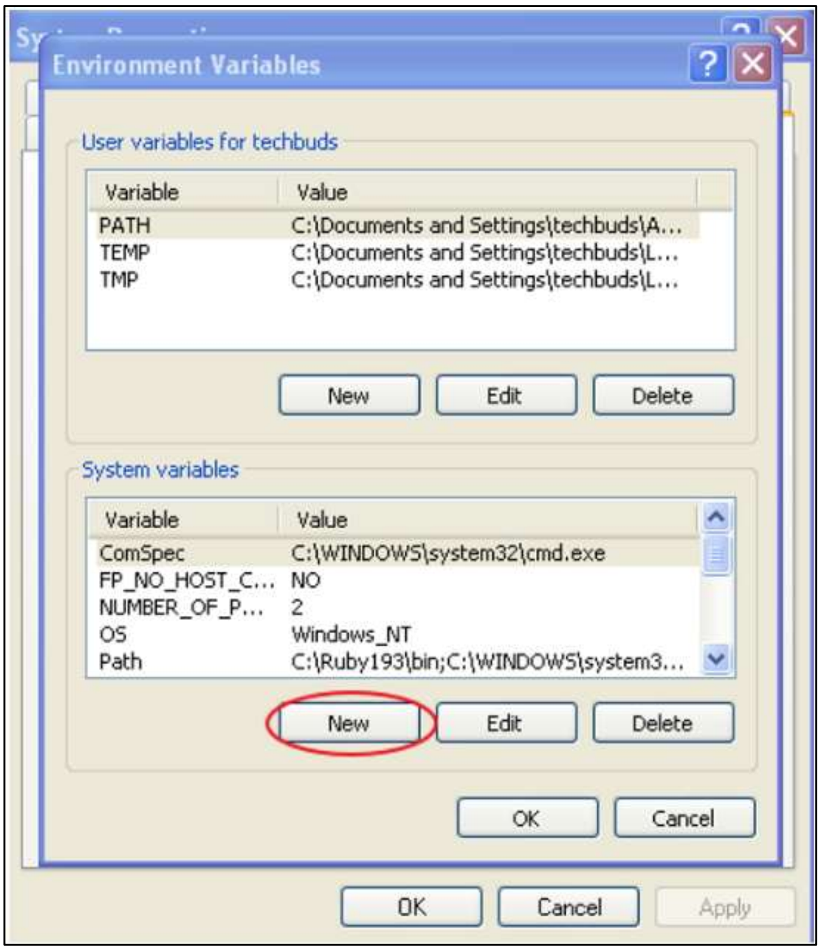
Variable value: rogram Files\Ampps\mysql\bin;C:\Ruby\bin

Variable	Value
ComSpec	C:\WINDOWS\system32\cmd.exe
FP_NO_HOST_C...	NO
NUMBER_OF_P...	2
OS	Windows_NT
Path	C:\Ruby193\bin;C:\WINDOWS\system3...

- Click the **OK** button.

System Variable:

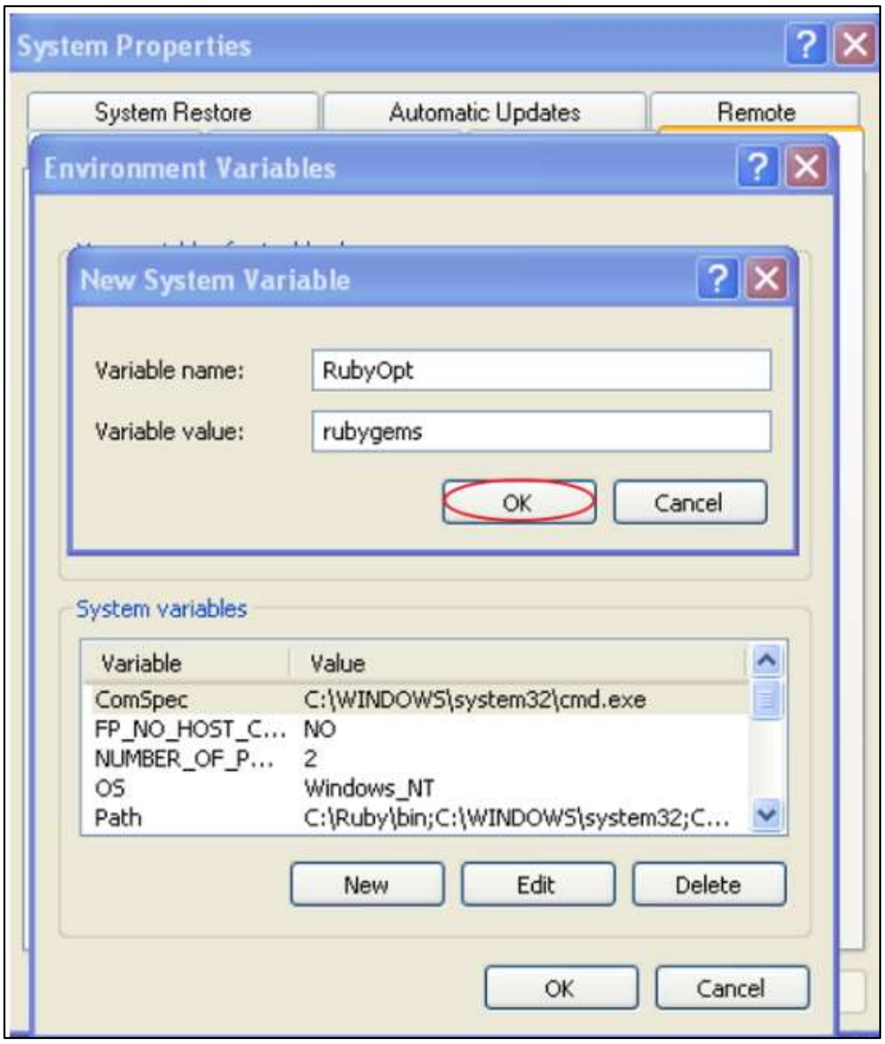
- Click the **New** button.



Variable	Value
PATH	C:\Documents and Settings\techbuds\A...
TEMP	C:\Documents and Settings\techbuds\L...
TMP	C:\Documents and Settings\techbuds\L...

Variable	Value
ComSpec	C:\WINDOWS\system32\cmd.exe
FP_NO_HOST_C...	NO
NUMBER_OF_P...	2
OS	Windows_NT
Path	C:\Ruby193\bin;C:\WINDOWS\system3...

Next, the **New System Variable** block is displayed as shown below.



Variable name: RubyOpt
Variable value: rubygems

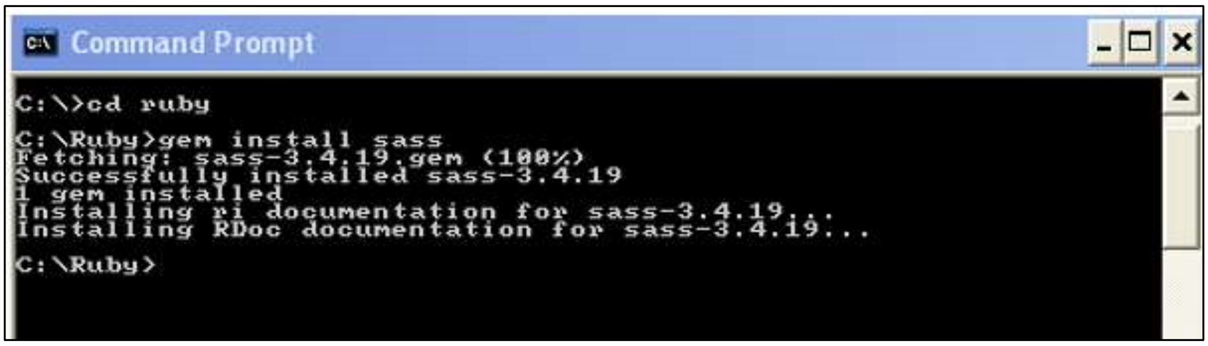
Variable	Value
ComSpec	C:\WINDOWS\system32\cmd.exe
FP_NO_HOST_C...	NO
NUMBER_OF_P...	2
OS	Windows_NT
Path	C:\Ruby\bin;C:\WINDOWS\system32;C...

- Enter **RubyOpt** in the *Variable name* field and **rubygems** in the *Variable value* field. After writing the *Variable name* and *value*, click the **OK** button.

Step(4): Open the command prompt in your system and enter the following line-

```
gem install sass
```

Step(5): Next, you will see the following screen after installing SASS successfully.



```
C:\>cd ruby

C:\Ruby>gem install sass
Fetching: sass-3.4.19.gem (100%)
Successfully installed sass-3.4.19
1 gem installed
Installing ri documentation for sass-3.4.19...
Installing RDoc documentation for sass-3.4.19...

C:\Ruby>
```

Example

Following is a simple example of SASS.

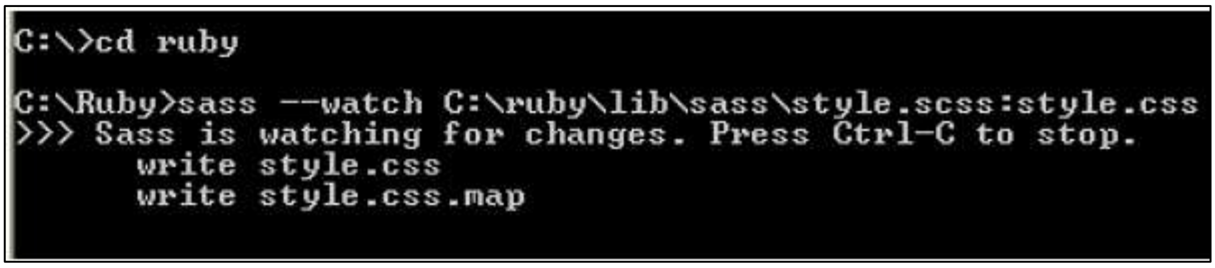
```
<html>
<head>
   <title> Import example of sass</title>
   <link rel="stylesheet" type="text/css" href="style.css"/>
</head>
<body>
   <h1>Simple Example</h1>
   <h3>Welcome to TutorialsPoint</h3>
</body>
</html>
```

Now, we will create file as *style.scss,* which is quite similar to CSS and the only one difference is that it will be saved with .scss extension. Both, .htm and .scss files should be created inside the folder **ruby**. You can save your .scss file in the folder **ruby\lib\sass\** (Before this process, create a folder as **sass** in lib directory).

```
h1{
   color: #AF80ED;
}
h3{
   color: #DE5E85;
}
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes by using the following command:

```
sass --watch C:\ruby\lib\sass\style.scss:style.css
```



```
C:\>cd ruby

C:\Ruby>sass --watch C:\ruby\lib\sass\style.scss:style.css
>>> Sass is watching for changes. Press Ctrl-C to stop.
      write style.css
      write style.css.map
```

When you run the above command, it will create the *style.css* file automatically. Whenever you change the SCSS file, the *style.css* file will be updated automatically.

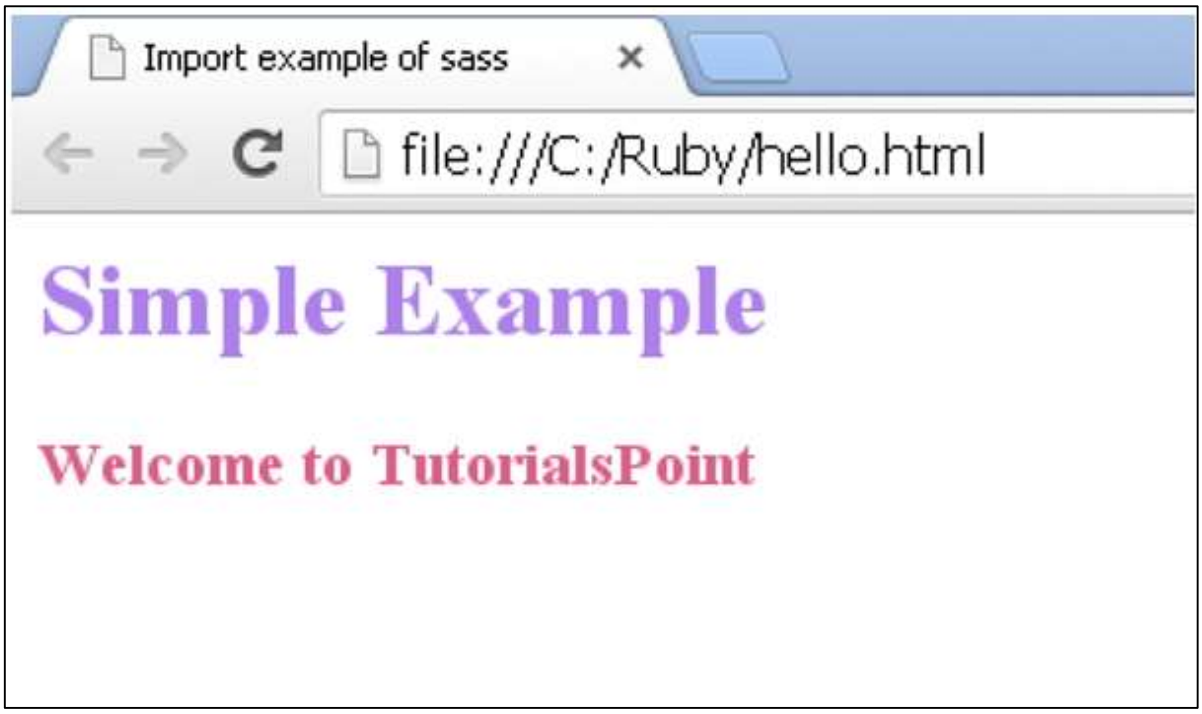
The *style.css* file will have the following code when you run the above given command:

style.css

```
h1 {
  color: #AF80ED;
  }
h3 {
  color: #DE5E85;
  }
```

Let us carry out the following steps to see how the above given code works:

- Save the above given code in **hello.html** file.
- Open this HTML file in a browser.



Simple Example

Welcome to TutorialsPoint

3. SASS – Syntax

In this chapter, we will study about SASS **Syntax**. SASS supports two syntaxes namely- **SCSS** and **Indented syntax**.

- The **SCSS (Sassy CSS)** is an extension of CSS syntax. This means, every valid CSS is a valid SCSS as well. SCSS makes, much easier to maintain, large stylesheets and can recognize vendor specific syntax. Many CSS and .SCSS files use the extension **.scss**.

- **Indented** - This is older syntax and sometimes just called as **SASS**. Using this form of syntax, CSS can be written concisely. SASS files use the extension **.sass**.

SASS Indented Syntax

SASS Indented syntax or just SASS is an alternative to CSS based SCSS syntax.

- It uses *indentation* rather than **{** and **}** to delimit blocks.

- To separate statements, it uses *newlines* instead of *semicolons(;)*.

- Property declaration and selectors must be placed on its *own line* and statements within **{** and **}** must be placed on *new line* and *indented*.

For instance, consider the following SCSS code:

```
.myclass {
  color= red;
  font-size= 0.2em;
}
```

The **indented** syntax is an older syntax, which is not recommended for use in new Sass files. If you use this file, it will display error in the CSS file as we have used **=** instead of **:** for setting properties and variables.

Compile the above given code using the following command:

```
sass --watch C:\ruby\lib\sass\style.scss:style.css
```

Next, run the above command; it will display an error in *style.css* file as shown below:

```
Error: Invalid CSS after "  color= red": expected "{", was ";"
        on line 2 of C:\ruby\lib\sass\style17.scss

1: .myclass {
2:   color= red;
3:   font-size= 0.2em;
4: }
```

Syntax Differences of SASS

Most CSS and SCSS syntaxes work perfectly in SASS. However, there are some differences, which are explained in the following sections:

Property Syntax

CSS properties can be declared in two ways:

- Properties can be declared similar to CSS but without **semicolon(;)**.
- **colon(:)** will be prefixed to every property name.

For instance, you can write as:

```
.myclass
  :color red
  :font-size 0.2em
```

Both the above ways (properties declaration without semicolon and colon prefixed to property name) can be used, by default. However, only one property syntax is allowed to specify when you use the :property_syntax option.

Multiline Selectors

In Indented syntax, selectors can be placed on a newline whenever they appear after **commas**.

Example

The following example describes the use of multiline selectors in the SCSS file:

```
<html>
<head>
   <title>Multiline Selectors</title>
   <link rel="stylesheet" type="text/css" href="style.css" />
   <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css">
   <script
src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>
   <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/js/bootstrap.min.js"></script>
</head>
<body>
   <h2>Example using Multiline Selectors</h2>
   <p class="class1">Welcome to Tutorialspoint!!!</p>
   <p class="class2">SASS stands for Syntactically Awesome Stylesheet...</p>
</body>
```

```
</html>
```

Next, create file *style.scss*. Note the *.scss* extension.

style.scss

```
.class1,
.class2{
   color:red;
}
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\style.scss:style.css
```

Next, execute the above given command, it will create *style.css* file automatically with the following code:

The generated *style.css* is as shown below:

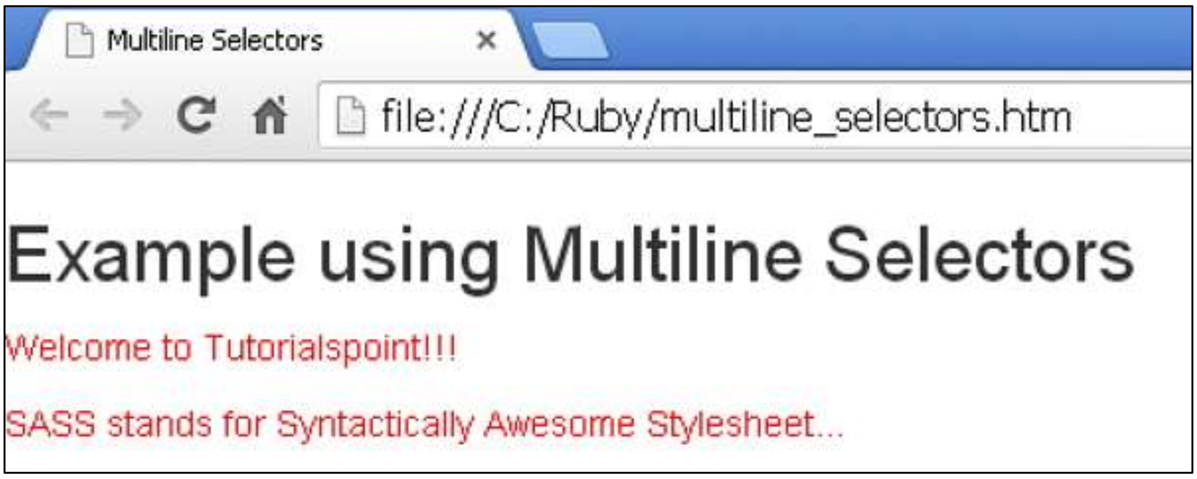
style.css

```
.class1,
.class2 {
  color: red;
}
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in *multiline_selectors.html* file.
- Open this HTML file in a browser, an output is displayed as shown below.



Example using Multiline Selectors

Welcome to Tutorialspoint!!!

SASS stands for Syntactically Awesome Stylesheet...

Comments

Comments take up an entire line and enclose all the text nested under them. They are line-based in indented syntax. For more information about comments, refer this link.

@import

In SASS the **@import** directive can be written with/without quotes. Unlike in SCSS, they must be used with quotes.

For instance, in SCSS the **@import** directive can be used as:

```
@import "themes/blackforest";

@import "style.sass";
```

This can be written in SASS as:

```
@import themes/blackforest

@import fontstyle.sass
```

Mixin Directives

SASS supports shorthand for directives like **@mixin** and **@include**. Instead of **@mixin** and **@include** you can use **=** and **+** characters, which require less typing and makes your code simpler, and easier to read.

For instance, you can write the mixin directives as:

```
=myclass

  font-size: 12px;

p

  +myclass
```

The above given code is the same as-

```
@mixin myclass

  font-size: 12px;

p

  @include myclass
```

Deprecated Syntax

SASS supports the use of some old syntax. However, using this syntax in SASS is **not recommended**. Warning will be displayed if you use this syntax and it is removed in later versions. Some of the old syntaxes are shown in the following table.

S. No.	Operator & Description
1	**=** It was used instead of : when setting variables and properties to values of SassScript.
2	**||=** It was used instead of : whenever you are assigning default value of a variable.
3	**!** Instead of $, ! was used as variable prefix. Functionality will not be changed when it is used instead of $.

4. Using SASS

SASS is more powerful and stable that provides power to the basic language by using extension of CSS. You can use SASS in three different ways:

- As a command line tool
- As a Ruby module
- As a plugin for Rack enable framework

If you are using SASS on windows, then you need to install **Ruby** first. For more information about installing Ruby, refer the SASS Installation chapter.

The following table shows the commands, which are used for executing the SASS code:

S. No.	Command & Description
1	**sass input.scss output.css** It is used to run the SASS code from the command line.
2	**sass --watch input.scss:output.css** It informs SASS to watch the file and update the CSS whenever SASS file changes.
3	**sass --watch app/sass:public/stylesheets** It is used to watch the entire directory, if SASS contains many files in a directory.

Rack/Rails/Merb Plugin

Rack is a web server interface, which is used for developing web applications in Ruby. For information about Rack, just visit this link.

You can enable the SASS in the **Rails 3** version using the **environment.rb** file present under the **config** folder. Enable the SASS for the Rails 3 using the following code:

```
config.gem "sass"
```

You can use the following line to the Gemfile for the Rails 3(and above version), as:

```
gem "sass"
```

Rails is an open-source web framework that uses web standards such as JSON, HTML, CSS and JavaScript for displaying user interface. To work with Rails, you need to have a basic knowledge of Ruby and object-oriented programming. Learn more about on Rails framework here.

If you want to enable the SASS in **Rack** application, add the following lines to the **config.ru** file, which is present in the app's root directory:

```
require 'sass/plugin/rack'

use Sass::Plugin::Rack
```

Merb is a web application framework, which provides speed and modularity to Rails. To know more about Merb, just open this link.

You can enable the SASS in **Merb** by adding the following line to the **config/dependencies.rb** file:

```
dependency "merb-haml"
```

Caching

SASS caches documents such as templates and partials, which can be reused without parsing them unless they have changed. It makes compilation of SASS files faster and works even better when the templates are divided into separate files, which are all imported into one large file. If you delete cached files, they will be generated again when you compile next time.

Options

You can set the options in the *environment.rb* file of Rails or *config.ru* file of Rack application by using the following line:

```
Sass::Plugin.options[:style] = :compact
```

You can also set options in the *init.rb* file of Merb by using the following line:

```
Merb::Plugin.config[:sass][:style] = :compact
```

There are some options available with *SASS* and *SCSS* as described in the table given below:

S. No.	Option & Description
1	**:style** It displays style of the output.
2	**:syntax** You can use indented syntax for *sass* and CSS extension syntax for *scss*.
3	**:property_syntax** It uses indented syntax to make use of properties. If it is not correct, then it will throw an error. For instance, consider "background: #F5F5F5" in which *background* is a property name and *#F5F5F5* is its property value. You must use colon after the property name.
4	**:cache** It speeds up compilation of SASS files. It is set to true by default.
5	**:read_cache** It read only SASS files if *cache* is not set and *read_cache* is set.

6	**:cache_store** It can be used to store and access the cached result by setting it to an instance of *Sass::CacheStores::Base*.
7	**:never_update** It should never update the CSS file if the template files changes. By default it is set to false.
8	**:always_update** It should update the CSS file whenever the template files changes.
9	**:always_check** It should check for the updates whenever the server starts. It will recompile and overwrite the CSS file, if there is an update in the SASS template file.
10	**:poll** It uses polling backend for *Sass::Plugin::Compiler#watch (which watches the template and updation of CSS files)* by setting it to true.
11	**:full_exception** It displays the error description whenever an exception occurs in SASS code within generated CSS file. It displays a line number where an error occurred along with source in the CSS file.
12	**:template_location** It provides the path for the template directory in the application.
13	**:css_location** It provides the path for the CSS stylesheets in the application.
14	**:unix_newlines** It provides Unix style newlines when writing files by setting it to true.
15	**:filename** It is name of the filename being displayed and used for reporting errors.
16	**:line** It specifies the first line of the SASS template and displays the line numbers for errors.
17	**:load_paths** It is used to load the paths for SASS template which are included using *@import* directive.
18	**:filesystem_importer** It is used to import files from file system that uses *Sass::Importers::Base* sub class to handle string load paths.
19	**:sourcemap** It generates source maps which instructs browser to find the SASS styles. It uses three values: • **:auto**: It contains relative URIs. If there is no relative URI, then uses "file:" URI.

	• **:file**: It uses "file:" URIs, which work locally, not on remote server. • **:inline**: It contains source text in the source map which is used to create large source map files.
20	**:line_numbers** It displays the line number for errors reported in the CSS file by setting it to true.
21	**:trace_selectors** It helps to trace the selectors of imports and mixins when it is set to true.
22	**:debug_info** It provides debug information of SASS file using line number and file when it is set to true.
23	**:custom** It makes data available to SASS functions in the separate applications.
24	**:quiet** It disables the warnings by setting it to true.

Syntax Selection

You can determine which syntax you are using in the SASS template by using the SASS command line tool. By default, SASS uses indented syntax, which is an alternative to CSS based SCSS syntax. You can use the SCSS command line program, which is similar to the SASS program, but by default, it considers the syntax to be SCSS.

Encodings

SASS uses the character encoding of stylesheets by specifying the following CSS specifications:

- First, it checks for Unicode byte, next *@charset* declaration and then Ruby string encoding.
- Next, if nothing is set, then it considers charset encoding as *UTF-8*.
- Determine character encoding explicitly by using *@charset* declaration. Just use "@charset encoding name" at the beginning of the stylesheet and SASS will assume that this is the given character encoding.
- If output file of SASS contains non-ASCII characters, then it will use the *@charset* declaration.

5. SASS – CSS Extensions

In this chapter, we will study about **CSS Extensions**. CSS Extensions can be used to enhance the functionality of the web pages. The following table lists down some of the CSS extensions used in SASS:

S. No.	CSS Extension & Description
1	Nested Rules It is a way of combining multiple CSS rules within one another.
2	Referencing Parent Selectors: & It is the process of selecting parent selector by using the **&** character.
3	Nested Properties It allows nesting of properties into other properties, which leads to grouping of another related code.
4	Placeholder Selectors Sass supports *placeholder selector* using class or id selector by making use of *@extend* directive.

SASS – Nested Rules

Description

Nesting is combining of different logic structures. Using SASS, we can combine multiple CSS rules within one another. If you are using multiple selectors, then you can use one selector inside another to create compound selectors.

Example

The following example describes the use of nested rules in the SCSS file:

```
<html>
<head>
   <title>Nested Rules</title>
   <link rel="stylesheet" type="text/css" href="style.css" />
   <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css">
   <script
src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>
   <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/js/bootstrap.min.js"></scr
ipt>
```

```
</head>
<body>
   <div class="container">
   <h1>My First Heading</h1>
   <p>It is a CSS pre-processor which helps to reduce repetition with CSS and save the time. </p>
   <p>It is more stable and powerful CSS extension language.</p>
   <div class="box">
   <h1>My Second Heading</h1>
   <p>It is initially designeld by Hampton Catlin and developed by Natalie Weizenbaum in 2006.</p>
   </div>
   </div>
</body>
</html>
```

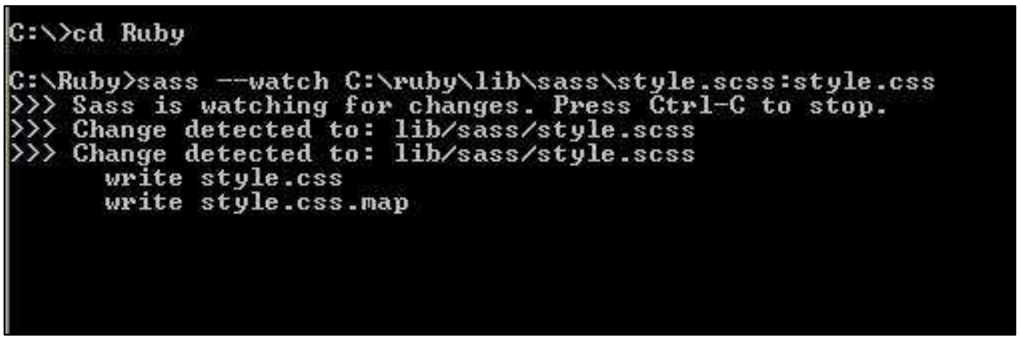
Next, create file *style.scss*. Note the *.scss* extension.

style.scss

```
.container{
  h1{
       font-size: 25px;
       color:#E45456;
 }
  p{
       font-size: 25px;
       color:#3C7949;
  }

 .box{
  h1{
        font-size: 25px;
        color:#E45456;
  }
  p{
       font-size: 25px;
       color:#3C7949;
  }
 }
}
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes by using the following command:

```
sass --watch C:\ruby\lib\sass\style.scss:style.css
```



```
C:\>cd Ruby

C:\Ruby>sass --watch C:\ruby\lib\sass\style.scss:style.css
>>> Sass is watching for changes. Press Ctrl-C to stop.
>>> Change detected to: lib/sass/style.scss
>>> Change detected to: lib/sass/style.scss
      write style.css
      write style.css.map
```

Next, execute the above command, it will create *style.css* file automatically with the following code:

The generated *style.css* is as shown below:

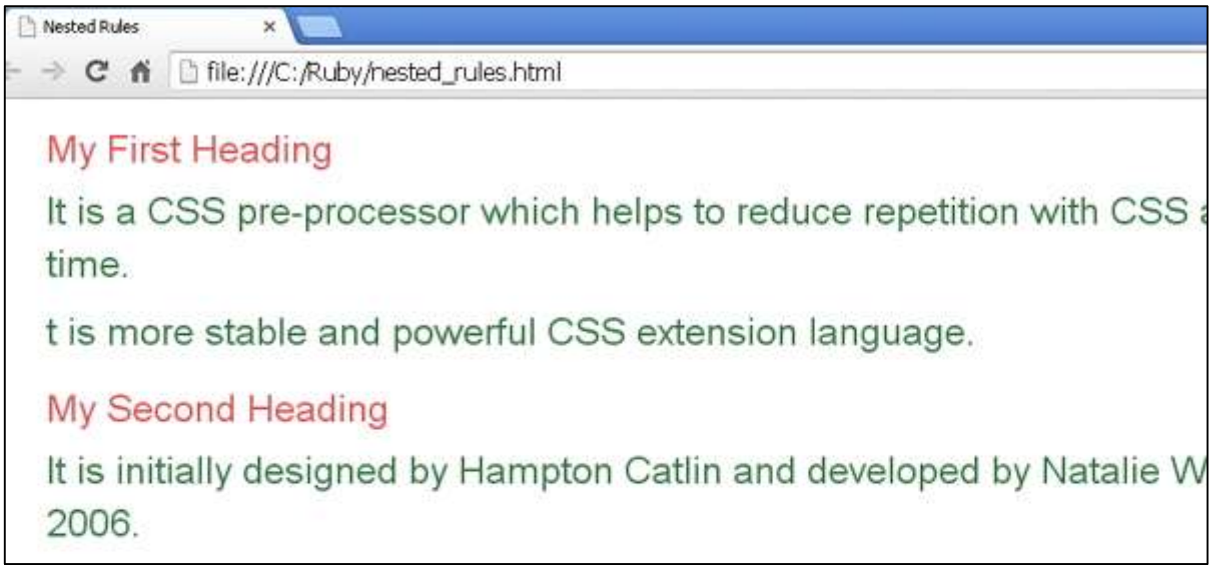
style.css

```
.container h1 {
  font-size: 25px;
  color: #E45456;
}
.container p {
  font-size: 25px;
  color: #3C7949;
}
.container .box h1 {
  font-size: 25px;
  color: #E45456;
}
.container .box p {
  font-size: 25px;
  color: #3C7949;
}
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in **nested_rules.html** file.
- Open this HTML file in a browser, an output is displayed as shown below.



My First Heading

It is a CSS pre-processor which helps to reduce repetition with CSS a
time.

t is more stable and powerful CSS extension language.

My Second Heading

It is initially designed by Hampton Catlin and developed by Natalie W
2006.

SASS – Referencing Parent Selectors

Description

You can select the parent selector by using the **&** character. It tells where the parent selector should be inserted.

Example

The following example describes the use of parent selectors in the SCSS file:

```
<html>
<head>
   <title>Referencing Parent Selectors</title>
   <link rel="stylesheet" type="text/css" href="style.css" />
   <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css">
   <script
src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>
   <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/js/bootstrap.min.js"></script>
</head>
<body>
   <div class="container">
   <h1>Example using Parent Selector</h1>
   <a href="http://www.tutorialspoint.com/"> www.tutorialspoint.com </a>
   </div>
</body>
</html>
```

Next, create the file *style.scss*. Note the use of **&** character, which specifies where the parent selector should be inserted.

style.scss

```
a {
    font-size: 20px;
    &:hover { background-color: yellow; }
}
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes by using the following command:

```
sass --watch C:\ruby\lib\sass\style.scss:style.css
```

Next, execute the above command; it will create the *style.css* file automatically with the following code:

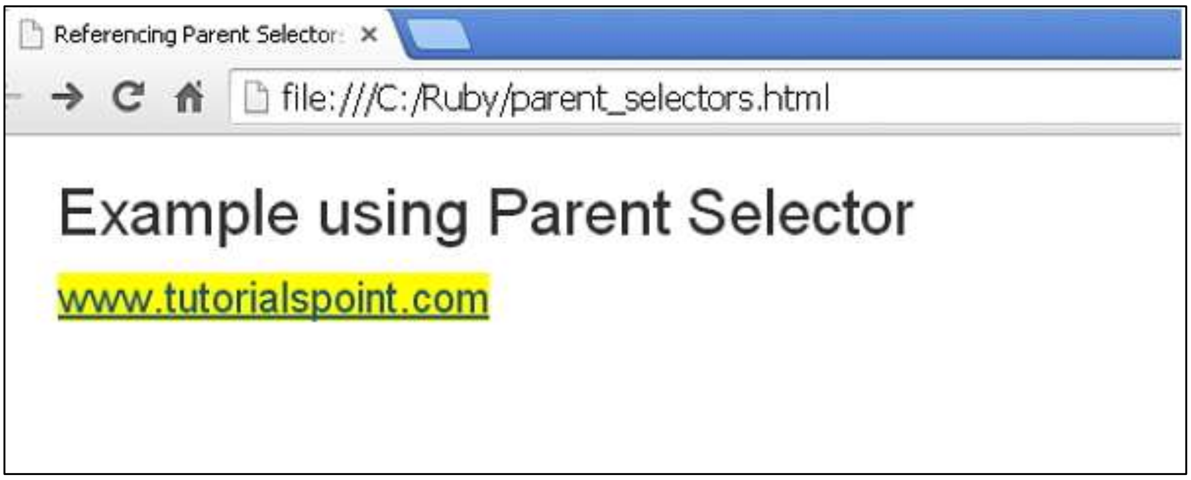
style.css

```
a {
  font-size: 20px;
}
a:hover {
    background-color: yellow;
}
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in **parent_selectors.html** file.
- Open this HTML file in a browser, an output is displayed as shown below.
- Here **&** will be replaced with the parent selector **a**. When you hover on the link, it will display background color as *yellow*.


Referencing Parent Selector
file:///C:/Ruby/parent_selectors.html

Example using Parent Selector

www.tutorialspoint.com

SASS – Nested Properties

Description

Using nested properties, you can avoid rewriting CSS multiple times. For instance, use *font* as namespace, which uses some properties such as font-family, font-size, font-weight and font-variant. In normal CSS, you need to write these properties every time with namespace. Using SASS, you can nest the properties by writing the namespace only once.

Example

The following example describes the use of nested properties in the SCSS file:

```
<html>
<head>
   <title>Nested Properties</title>
   <link rel="stylesheet" type="text/css" href="style.css" />
   <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css">
   <script
src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script
>
   <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/js/bootstrap.min.js"></scr
ipt>
</head>
<body>
   <div class="container">
   <h1>Example using Nested Properties</h1>
   <p class="line">SASS stands for Syntactically Awesome Stylesheet</p>
   </div>
</body>
</html>
```

Next, create file *style.scss*.

style.scss

```
.line {
  font: {
    family: Lucida Sans Unicode;
    size:20px;
    weight: bold;
    variant: small-caps;
  }
}
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\style.scss:style.css
```

Next, execute the above command; it will create the *style.css* file automatically with the following code:

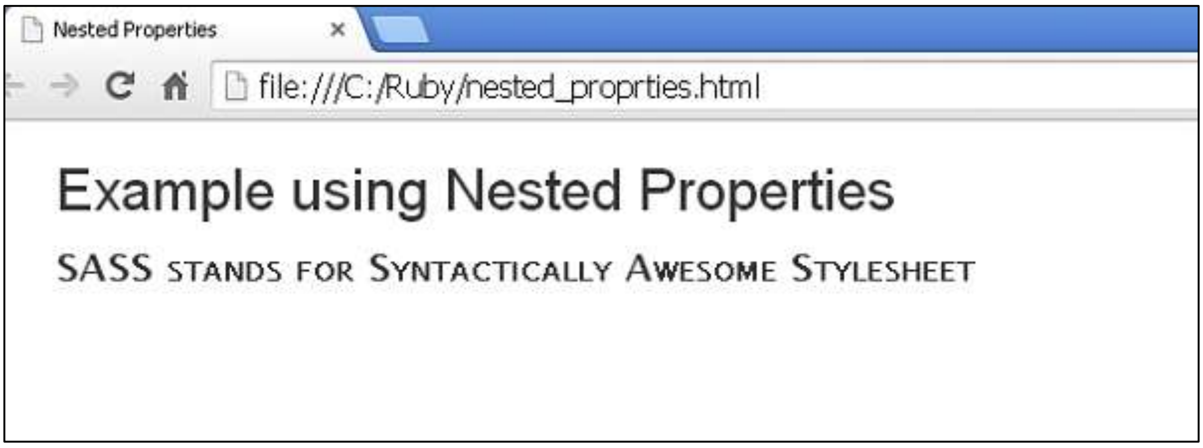
style.css

```
.line {
  font-family: Lucida Sans Unicode;
  font-size: 20px;
  font-weight: bold;
  font-variant: small-caps;
}
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in **nested_properties.html** file.
- Open this HTML file in a browser, an output is displayed as shown below.



Example using Nested Properties

SASS stands for Syntactically Awesome Stylesheet

SASS – Placeholder Selectors

Description

SASS supports *placeholder selector* using *class* or *id* selector. In normal CSS, these are specified with "**#**" or "**.**", but in SASS they are replaced with "**%**". To work with placeholder selector, they can be used with *@extend* directive. Without using *@extend* directive, you cannot display the result in CSS.

Example

The following example demonstrates the use of placeholder selectors in the SCSS file:

```
<html>
<head>
   <title>Placeholder Selectors</title>
   <link rel="stylesheet" type="text/css" href="style.css" />
   <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css">
   <script
src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>
   <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/js/bootstrap.min.js"></scr
ipt>l
</head>
<body>
   <h1>First Heading</h1>
   <p class="frst_para">It is a CSS pre-processor which helps to reduce
repetition with CSS and save the time. </p>
   <h1>Second Heading</h1>
   <p class="sec_para">It was initially designed by Hampton Catlin and
developed by Natalie Weizenbaum in 2006.</p>
</body>
</html>
```

Next, create file *style.scss*.

style.scss

```
.frst_para {
   color: green;
}
.sec_para {
   @extend .frst_para;
   font-size:20px;
}
```

Here, we have used the **@extend** directive, which allows one selector to inherit styles of another selector. You can tell SASS to watch the file and update the CSS whenever Sass file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\style.scss:style.css
```

Next, execute the above command; it will create the *style.css* file automatically with the following code:

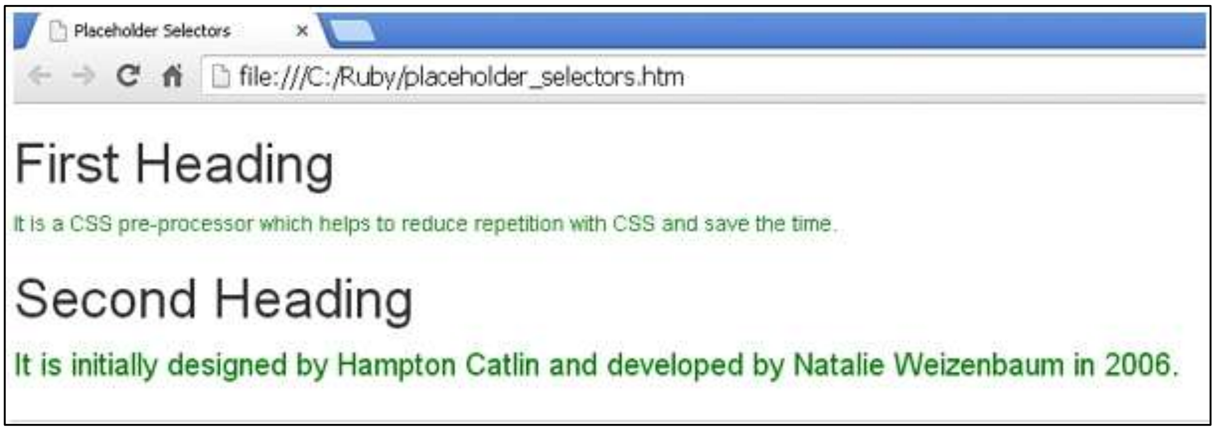
style.css

```
.frst_para, .sec_para {
  color: green;
}
.sec_para {
  font-size: 20px;
}
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in **placeholder_selectors.html** file.
- Open this HTML file in a browser, an output is displayed as shown below.



First Heading

It is a CSS pre-processor which helps to reduce repetition with CSS and save the time.

Second Heading

It is initially designed by Hampton Catlin and developed by Natalie Weizenbaum in 2006.

6. SASS – Comments

In this chapter, we will study about Sass **Comments**. Comments are non-executable statements, which are placed in source code. Comments make source code easier to understand. SASS supports two types of comments.

- **Multiline comments -** These are written using /* and */. Multiline comments are preserved in CSS output.

- **Single line comments -** These are written using **//** followed by comments. Single line comments are not preserved in CSS output.

Example

The following example demonstrates the use of comments in the SCSS file:

```
<html>
<head>
   <title>SASS comments</title>
   <link rel="stylesheet" type="text/css" href="style.css"/>
</head>
<body>
  <h1>Welcome to TutorialsPoint</h1>
  <a href="http://www.tutorialspoint.com/">TutorialsPoint</a>
</body>
</html>
```

Next, create file *style.scss*.

style.scss

```
/* This comment is
 * more than one line long
 * since it uses the CSS comment syntax,
 * it will appear in the CSS output. */
body { color: black; }

// These comments are in single line
// They will not appear in the CSS output,
// since they use the single-line comment syntax.
a { color: blue; }
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\style.scss:style.css
```

Next, execute the above command; it will create the *style.css* file automatically with the following code:

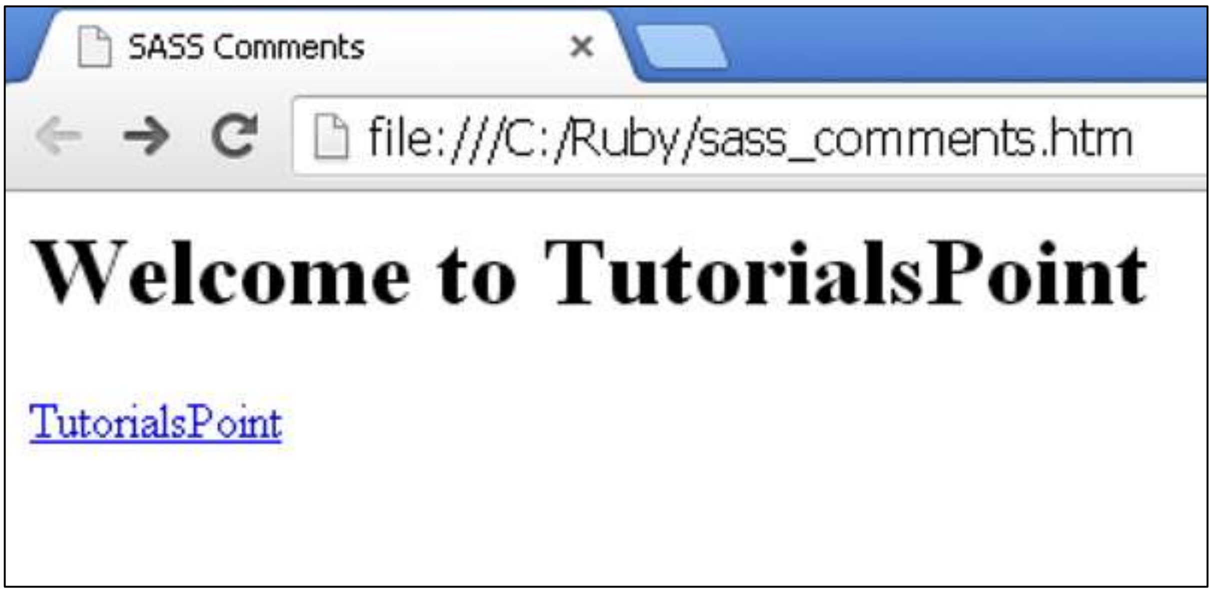
style.css

```
/* This comment is
 * more than one line long
 * since it uses the CSS comment syntax,
 * it will appear in the CSS output. */
body {
  color: black; }
a {
  color: blue; }
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in **sass_comments.html** file.
- Open this HTML file in a browser, an output is displayed as shown below.



Welcome to TutorialsPoint

TutorialsPoint

To study about interpolation within multiline comments, click this link.

Sass – Interpolation in Multiline Comments

Description

Interpolation within the multiline comments are resolved in the resulting CSS. You can specify variables or property names within the curly braces.

Syntax

```
$var : "value";
/* multiline comments #{$var} */
```

Example

The following example demonstrates the use of interpolation in multiline comments in the SCSS file:

```
<html>
<head>
   <title>SASS comments</title>
   <link rel="stylesheet" type="text/css" href="style.css"/>
</head>
<body>
  <h1>Welcome to TutorialsPoint</h1>
  <p>This is an example for Interpolation in SASS.</p>
</body>
</html>
```

Next, create file *style.scss*.

style.scss

```
$version: "7.8";
/* Framework version for the generated CSS is #{$version}. */
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\style.scss:style.css
```

Next, execute the above command; it will create the *style.css* file automatically with the following code:

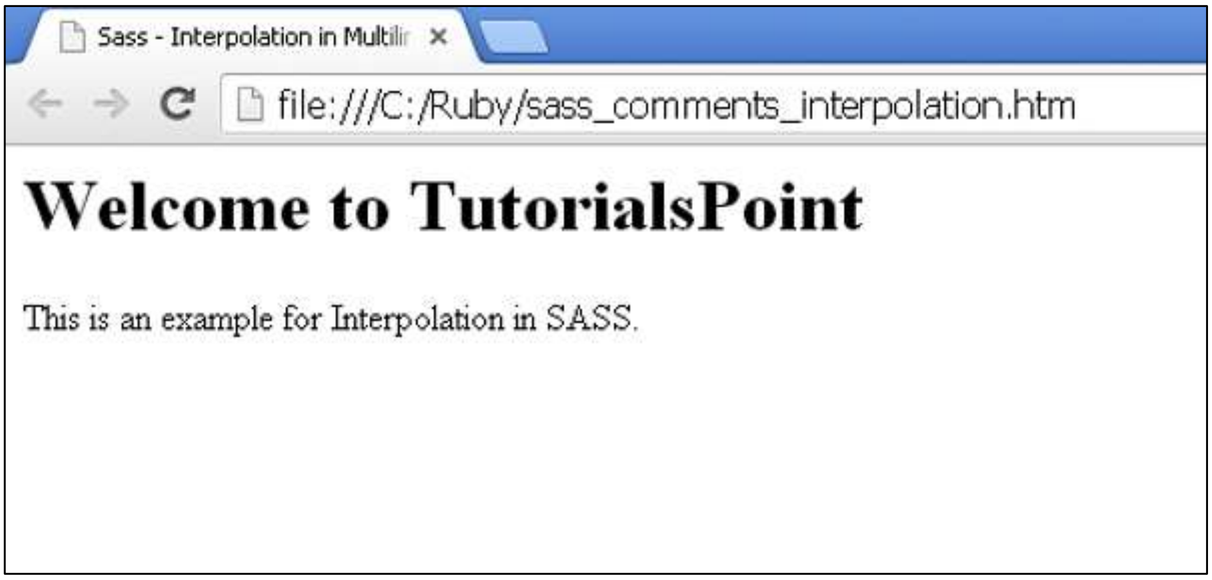
style.css

```
/* Framework version for the generated CSS is 7.8. */
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in **sass_comments_interpolation.htm** file.
- Open this HTML file in a browser, an output is displayed as shown below.


Sass - Interpolation in Multilir
file:///C:/Ruby/sass_comments_interpolation.htm

Welcome to TutorialsPoint

This is an example for Interpolation in SASS.

7. SASS – Script

SASS uses a small set of extensions known as SassScript, which can be included in the SASS documents to compute variables from property values and uses properties of variables, arithmetic, and other functions. SassScript can also be used with selectors and property names while using mixins (Mixins allows to re-use CSS styles throughout the stylesheet).

The following table lists some of the CSS extensions used in SASS:

S. No.	CSS Extension & Description
1	Interactive Shell It evaluates SassScript expression using command line.
2	Variables It represents the data such as numeric values, characters or memory addresses.
3	DataTypes It declares data type for every data object.
4	Operations It provides operations such as number, color, string, boolean and list operations.
5	Parentheses It is pair of signs which are usually marked off by round brackets ( ) or square brackets [].
6	Functions It supports for the use of functions by providing some keyword arguments.
7	Interpolation It provides SassScript variables and property names using **#{ }** syntax.
8	& in SassScript It allows nesting of properties into another properties which leads to group of another related code.
9	Variable Defaults It allows nesting of properties into another properties which leads to group of another related code.

SASS – Interactive Shell

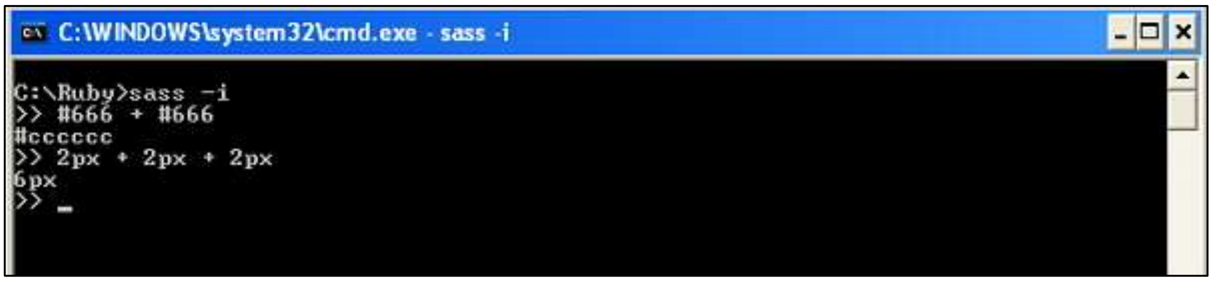
Description

You can work easily with SassScript by using the interactive shell. You can run the shell with the SASS command line along with the -i option.

Syntax

```
$ sass -i
```

Using the above command, let us write some expressions as shown below:



```
C:\Ruby>sass -i
>> #666 + #666
#cccccc
>> 2px + 2px + 2px
6px
>> _
```

As you can see in the above image, we have used color codes which produces another color code by adding them and adds the three numbers to display the total value of the given numbers.

SASS – Variables

Description

Programmers use variables to represent data, such as numeric values, characters or memory addresses. The importance of variables is, you can reuse the stored values in the variable throughout the stylesheet.

Syntax

```
$variable_name : some value;
```

Variables are defined with *dollar sign ($)* and ends with *semicolon (;)*.

Example

The following example demonstrates the use of variable in the SCSS file:

```
<html>
<head>
   <title>Variables</title>
   <link rel="stylesheet" type="text/css" href="style.css" />
   <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css">
```

```
   <script
src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>
   <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/js/bootstrap.min.js"></script>
</head>
<body>
   <div class="container">
   <h1>Example using Variables</h1>
   <p>Sass is an extension of CSS that adds power and elegance to the basic
language.</p>
   </div>
</body>
</html>
```

Next, create file *style.scss*.

style.scss

```
$txtcolor:#008000;
$fontSize: 20px;
p{
   color:$txtcolor;
   font-size:$fontSize;
}
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\style.scss:style.css
```

Next, execute the above command; it will create the *style.css* file automatically with the following code:

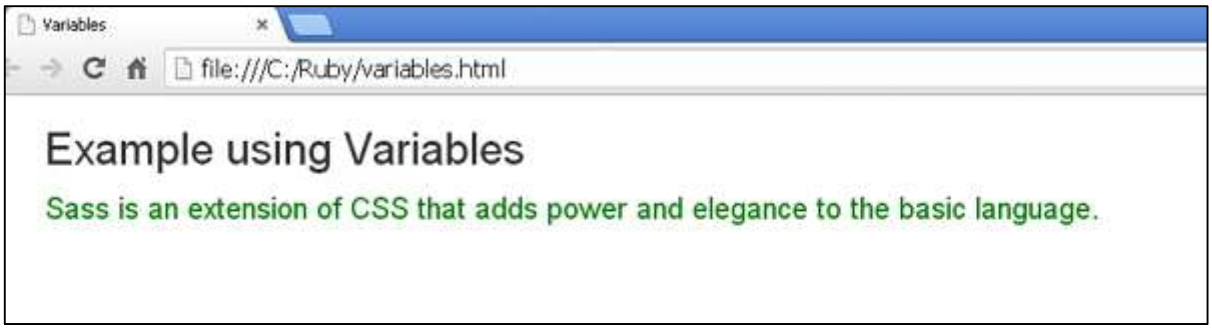
style.css

```
p {
  color: #008000;
  font-size: 20px;
}
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in **variables.html** file.
- Open this HTML file in a browser, an output is displayed as shown below.



Example using Variables

Sass is an extension of CSS that adds power and elegance to the basic language.

SASS – Data Types

Data Types

Data Type is a type of information, which requires declaring data type for every data object. The following table shows various data types supported by SassScript:

S. No.	Data Type & Description	Example
1	**Numbers** It represents integer types.	2, 10.5
2	**Strings** It is sequence of characters defined within single or double quotes.	'Tutorialspoint', "Tutorialspoint"
3	**Colors** It is used for defining color value.	red, #008000, rgb(25,255,204)
4	**Booleans** It returns true or false boolean types.	10 > 9 specifies true
5	**Nulls** It specifies null value, which is unknown data.	if(val==null) {//statements}
6	**Space and Comma** Represents the values, which are separated by spaces or commas.	1px solid #eeeeee, 0 0 0 1px
7	**Mapping** It maps from one value to another value.	FirsyKey: frstvalue, SecondKey: secvalue

Strings

Strings are series of characters, which are represented within single or double quotes. The strings which are defined with single quote or double quotes will be displayed as unquoted string value by using **#{ }** interpolation (it is a way of using variables in selectors).

Example

The following example demonstrates the use of strings in the SCSS file:

```
<html>
<head>
   <title>Strings</title>
   <link rel="stylesheet" type="text/css" href="style.css" />
   <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css">
   <script
src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script
>
   <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/js/bootstrap.min.js"></scr
ipt>
</head>
<body>
   <div class="container">
   <h2>Example using Strings</h2>
   <p class="tutorialspoint">Sass is an extension of CSS that adds power and
elegance to the basic language.</p>
   </div>
</body>
</html>
```

Next, create file *style.scss*.

style.scss

```
$name: "tutorialspoint";
p.#{$name} {
  color: blue;
}
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\style.scss:style.css
```

Next, execute the above command; it will create the style.css file automatically with the following code:

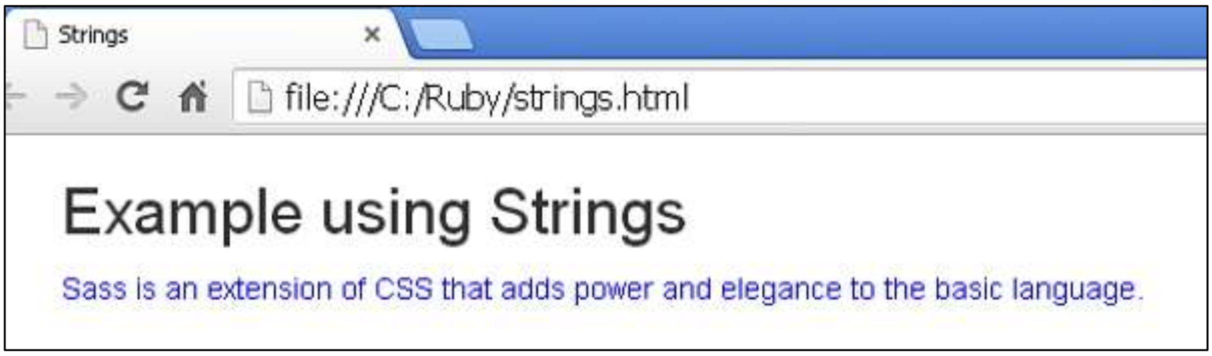
style.css

```
p.tutorialspoint {
  color: blue;
}
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in **strings.html** file.
- Open this HTML file in a browser, an output is displayed as shown below.



Example using Strings

Sass is an extension of CSS that adds power and elegance to the basic language.

Lists

Lists specify multiple values, which are separated using spaces or commas. Even single value are considered as a list.

SASS uses some of the list functions such as:

- **nth function**: It provides specific item of the list.
- **join function**: It joins multiple lists into one.
- **append function**: It appends the item to other end of the list.
- **@each directive**: It provides styles for the each item in the list.

For example, consider there are two types of list; the first list contains the following values which are separated using comma.

```
10px 11px, 15px 16px
```

If the inner list and the outer list have same separator, then you can use parentheses to specify where both the lists will start and stop. You can specify these lists as shown below:

```
{10px 11px} {15px 16px}
```

Maps

Maps are combination of keys and values in which keys are used to represent the values. Maps define values into groups and can be accessed dynamically. You can write a map expression as:

```
$map: (FirsyKey: frstvalue, SecondKey: secvalue, Thirdkey: thdvalue);
```

It uses some of the functions such as:

- **map-get**: It provides values of the map.
- **map-merge**: It adds values to the map.
- **@each directive**: It specifies styles for key/value pair in the map.

Maps represent empty key/value pairs as **( )** with no elements and uses the **inspect ($value)** function to display the output for the maps.

Colors

It is used for defining SassScript color value. For instance, if you are using color code as *#ffa500*, then it will display as *orange* color in the compressed mode. SASS provides the same output format as typed in the other output modes, which becomes an invalid syntax when a color is interpolated into a selector. To overcome this issue, use the color names within quotes.

SASS – Operations

SASS provides some of the operations such as number operations, color operations, string operations, Boolean operations and list operations. We will discuss these operations one by one as described below.

S. No.	Operation & Description
1	Number Operations It allows mathematical operations such as addition, subtraction, multiplication and division
2	Color Operations It allows using color components along with the arithmetic operations.
3	String Operations It uses + operation to concatenate strings.
4	Boolean Operations You can perform Boolean operations on SASS script by using *and*, *or* and *not* operators.
5	**List Operations** Lists represent series of values, which are separated using commas or space. For information about lists, see the lists section under **data types** section.

SASS – Number Operations

Description

SASS allows for mathematical operations such as addition, subtraction, multiplication and division. You cannot use incompatible units such as *px* * *px* or while adding number with *px* and *em* leads to produce invalid CSS. Therefore, SASS will display an error if you use invalid units in CSS. SASS supports relational operators like <, >, <=, >= and equality operators = =, !=.

Division and /

SASS allows division operation (/) on numbers as we do in normal CSS. You can use division (/) operation in three situations.

- If the value is stored in a variable or returned by function.
- If parentheses are outside the list and value is inside, the value will be surrounded by parentheses.
- If value is a part of arithmetic expression.

Subtraction, Negative Numbers, and -

Using SASS, you can perform some operations such as subtraction of numbers (10px - 5px), negating a number (-5), unary negation operator (-$myval) or using identifier (font-size). In some of the cases, these are useful like:

- you can use spaces both sides of - when performing subtraction of numbers
- you can use space before the - , but not after the negative number or a unary negation
- you can enclose the unary negation within parentheses separated by space (5px (-$myval))

Examples are-

- It can used in identifiers such as *font-size* and SASS allows only valid identifiers.
- It can be used with two numbers without space i.e. 10-5 is similar to 10 - 5.
- It can be used as beginning of a negative number (-5).
- It can be used without considering space such as 5 -$myval is similar to 5 - $myval.
- It can be used as unary negation operator (-$myval).

Example

The following example demonstrates the use of number operations in the SCSS file:

```
<html>
<head>
   <title>Number Operations</title>
   <link rel="stylesheet" type="text/css" href="style.css" />
   <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css">

   <script
src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>

   <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/js/bootstrap.min.js"></script>

</head>

<body>

   <div class="container">

   <p class="para1">SASS stands for Syntactically Awesome Stylesheet..</p>

   <h2>Hello...Welcome to Sass</h2>

   <h3>Hello...Welcome to Sass</h3>

   <p class="para2">Hello...Welcome to Sass</p>

   </div>

</body>

</html>
```

Next, create file *style.scss*.

style.scss

```
$size: 25px;
h2{
   font-size: $size + 5;
}
h3{
   font-size: $size / 5;
}
.para1 {
   font-size: $size * 1.5;
}
.para2 {
   font-size: $size - 10;
}
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\style.scss:style.css
```

Next, execute the above command; it will create the *style.css* file automatically with the following code:

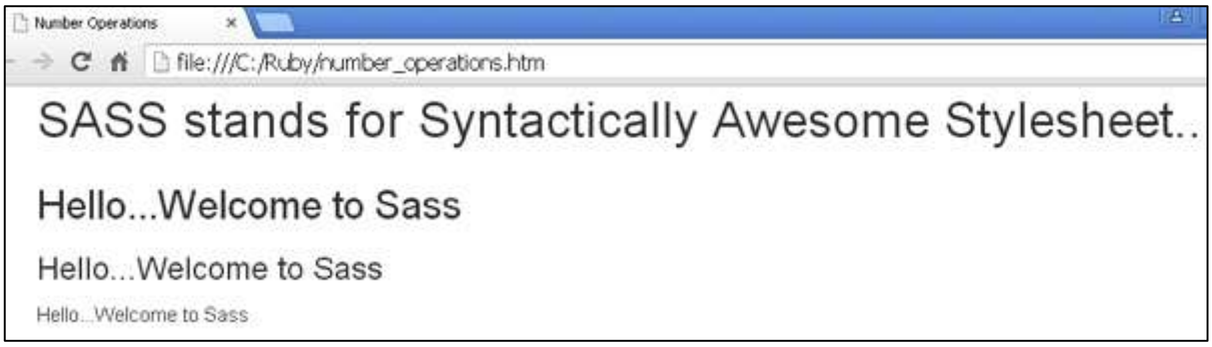
style.css

```
h2 {
  font-size: 30px;
}
h3 {
  font-size: 5px;
}
.para1 {
  font-size: 37.5px;
}
.para2 {
  font-size: 15px;
}
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in **number_operations.html** file.
- Open this HTML file in a browser, an output is displayed as shown below.



SASS stands for Syntactically Awesome Stylesheet..

Hello...Welcome to Sass

Hello...Welcome to Sass

Hello...Welcome to Sass

SASS – Color Operations

Description

SASS allows using color components along with the arithmetic operations and any color expression returns a SassScript color value.

Example

The following example demonstrates the use of color operations in the SCSS file:

```
<html>
<head>
   <title>Color Operations</title>
   <link rel="stylesheet" type="text/css" href="style.css" />
   <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css">

   <script
src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>

   <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/js/bootstrap.min.js"></script>

</head>

<body>

   <div class="container">

   <h3>This is Example of Sass Color Operations</h3>

   <p>SASS stands for Syntactically Awesome Stylesheet..</p>

   </div>

</body>

</html>
```

Next, create file *style.scss*.

style.scss

```
$color1: #333399;

$color2: #CC3399;

p{

  color: $color1 + $color2;

}
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\style.scss:style.css
```

Next, execute the above command; it will create the *style.css* file automatically with the following code:

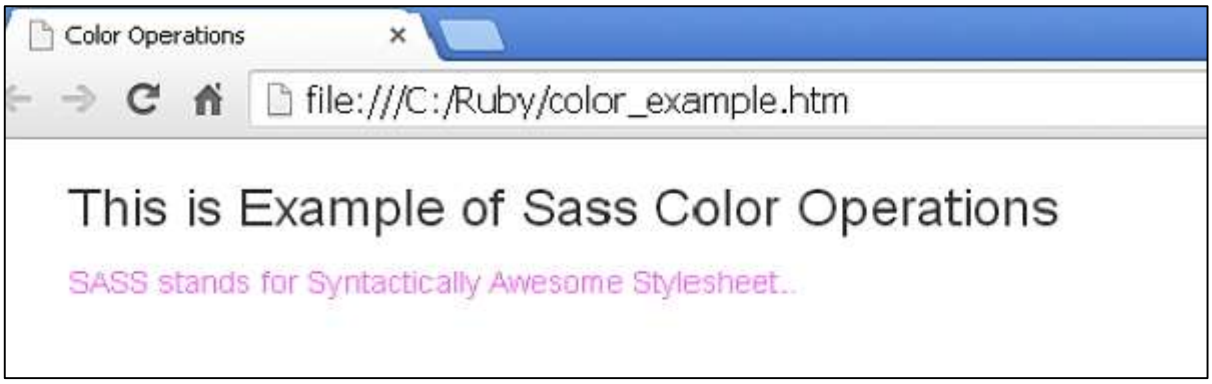
style.css

```
p {

   color: #ff66ff;

}
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in **color_operations.html** file.
- Open this HTML file in a browser, an output is displayed as shown below.



This is Example of Sass Color Operations

SASS stands for Syntactically Awesome Stylesheet..

SASS – String Operations

Description

You can use + operation to concatenate strings (e.g. font-size: 5px+3px).

Example

The following example demonstrates the use of string operations in the SCSS file:

```
<html>
<head>
   <title>String Operations</title>
   <link rel="stylesheet" type="text/css" href="style.css" />
   <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css">
   <script
src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script
>
   <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/js/bootstrap.min.js"></scr
ipt>
</head>
<body>
   <div class="container">
   <h3>Example using Sass Strings Operations</h3>
   <p>SASS stands for Syntactically Awesome Stylesheet..</p>
   </div>
</body>
</html>
```

Next, create file *style.scss*.

style.scss

The following SCSS code is used to concatenate values, which increases the font size of the <p> tag statements.

```
p {
  font-size: 5px + 10px;
}
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\style.scss:style.css
```

Next, execute the above command; it will create the *style.css* file automatically with the following code:

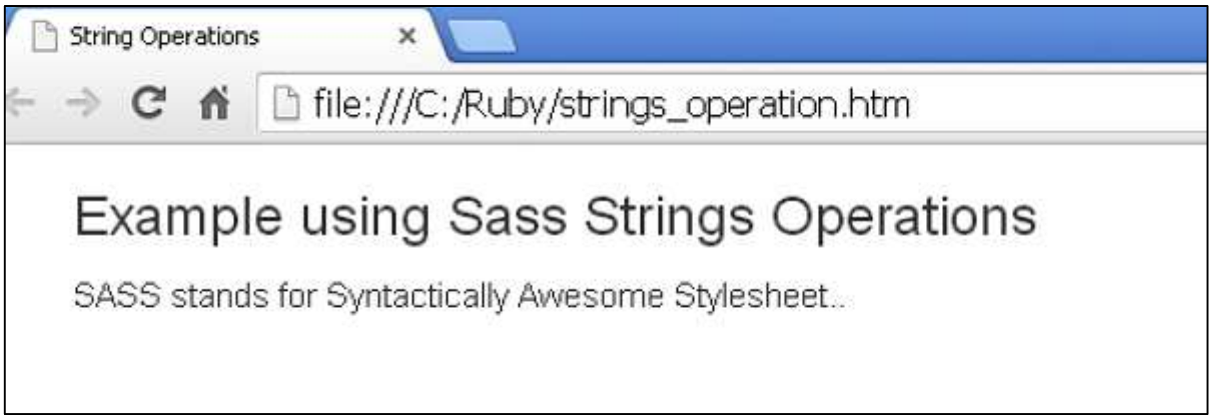
style.css

```
p {
  font-size: 15px;
}
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in **string_operations.html** file.
- Open this HTML file in a browser, an output is displayed as shown below.



Example using Sass Strings Operations

SASS stands for Syntactically Awesome Stylesheet..

SASS – Boolean Operations

Description

You can perform Boolean operations on Sass script by using *and*, *or* and *not* operators.

Example

The following example demonstrates the use of Boolean operations in the SCSS file:

```
<html>

<head>

   <title>Boolean Operations</title>

   <link rel="stylesheet" type="text/css" href="style.css" />

   <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css">

   <script
src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>

   <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/js/bootstrap.min.js"></scr
ipt>

</head>

<body>

   <h3>Example using Boolean Operations</h3>

   <p class="bool">SASS stands for Syntactically Awesome Stylesheet..</p>

</body>

</html>
```

Next, create file *style.scss*.

style.scss

```
$age:20;
.bool {
    @if ($age > 10 and $age < 25) {
        color: green;
   }
}
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\style.scss:style.css
```

Next, execute the above command; it will create the *style.css* file automatically with the following code:

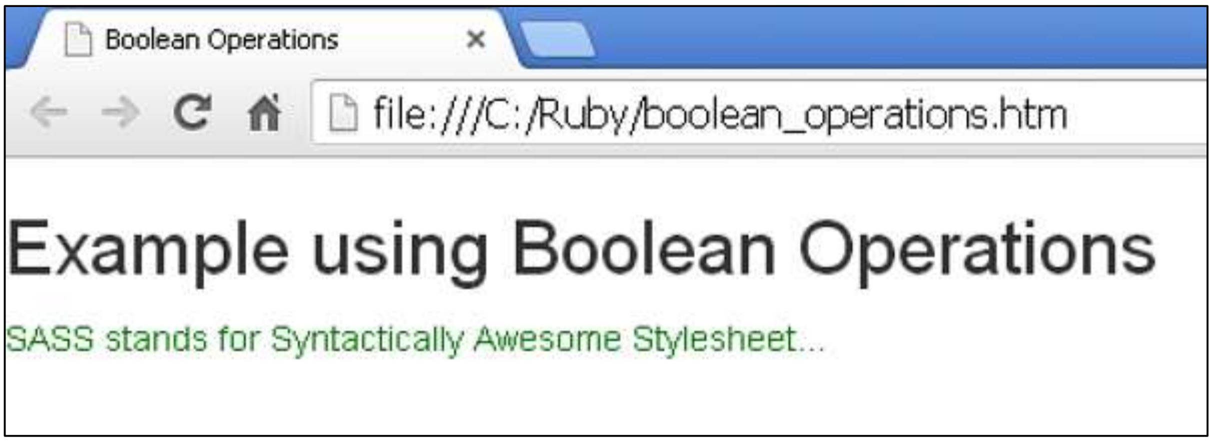
style.css

```
.bool {

  color: green;

}
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in **boolean_operations.html** file.
- Open this HTML file in a browser, an output is displayed as shown below.



Example using Boolean Operations

SASS stands for Syntactically Awesome Stylesheet...

SASS – Parentheses

Description

Parentheses are a pair of signs, which are usually marked off by round brackets ( ) or square brackets [], providing symbolic logic that affect the order of operations.

Example

The following example demonstrates use of parentheses in the SCSS file:

```
<html>
<head>
   <title>String Operations</title>
   <link rel="stylesheet" type="text/css" href="style.css" />
   <link                                                       rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css">
   <script
src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>
   <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/js/bootstrap.min.js"></scr
ipt>
</head>
<body>
   <div class="container">
   <h2>Example using Sass Parentheses</h2>
   <p>SASS stands for Syntactically Awesome Stylesheet..</p>
   </div>
</body>
</html>
```

Next, create file *style.scss*.

style.scss

```
p {
  font-size:  5px + (6px * 2);
  color:#ff0000;
}
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\style.scss:style.css
```

Next, execute the above command, it will create the *style.css* file automatically with the following code:

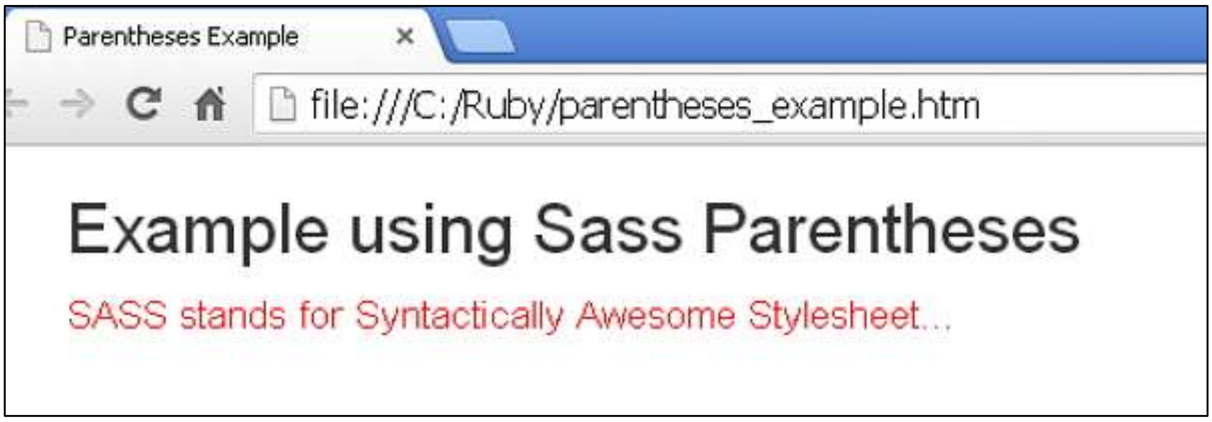
style.css

```
p {
  font-size: 17px;
  color: #ff0000;
}
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in **parentheses_example.html** file.
- Open this HTML file in a browser, an output is displayed as shown below.



Example using Sass Parentheses

SASS stands for Syntactically Awesome Stylesheet...

SASS – Functions

Description

SASS supports the use of functions by providing some keyword arguments, which are specified using normal CSS function syntax.

Syntax

```
p {
  color: hsl($hue: 0, $saturation: 50%, $lightness: 50%);
}
```

HSL stands for hue, saturation, and lightness, which are more intuitive for creating a set of matching colors by using saturation and lightness.

- **hue**: It represents the degree of color such as 120 for red, 240 for green, 290 for pastel violet etc.
- **saturation**: It is a percentage value that increases the saturation of color.
- **lightness**: It is a percentage value which decreases the lightness of color.

Example

The following example demonstrates the use of functions in the SCSS file:

```
<html>
<head>
   <title>Functions Example</title>
   <link rel="stylesheet" type="text/css" href="style.css" />
   <link rel="stylesheet" 
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css">
   <script 
src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>
   <script 
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/js/bootstrap.min.js"></script>
</head>
<body>
   <div class="container">
   <h2>Example using Functions</h2>
   <p>SASS stands for Syntactically Awesome Stylesheet..</p>
   </div>
</body>
</html>
```

Next, create file *style.scss*.

style.scss

Use the following SCSS code, which defines the HSL function on the SASS code.

```
p {
  color: hsl(290,60%,70%);
}
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\style.scss:style.css
```

Next, execute the above command; it will create the *style.css* file automatically with the following code:

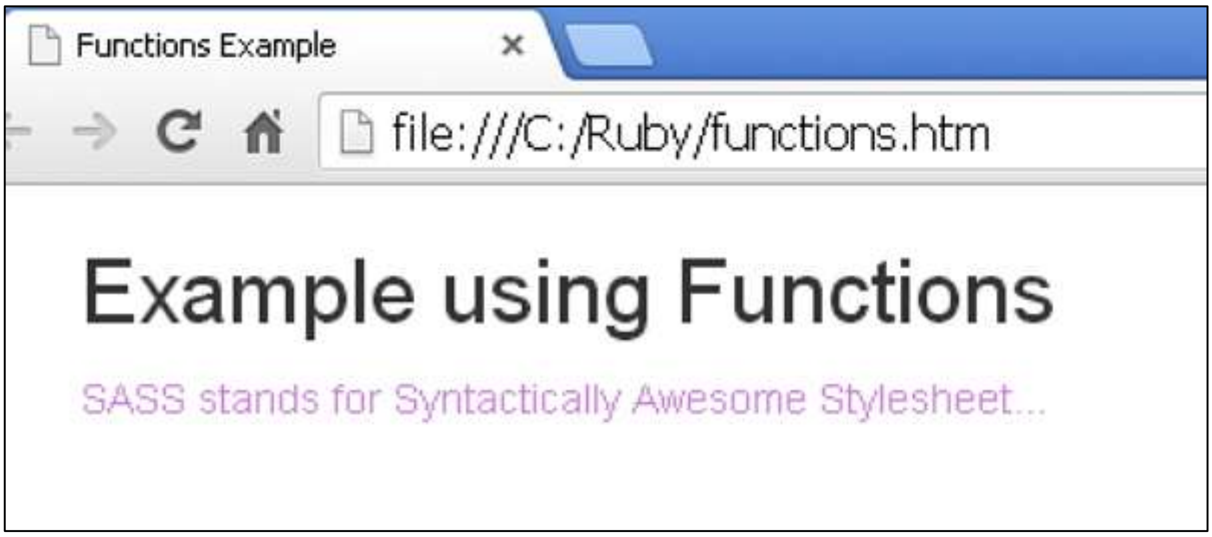
style.css

```
p {
  color: #d185e0;
}
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in **functions.html** file.
- Open this HTML file in a browser, an output is displayed as shown below.



Example using Functions

SASS stands for Syntactically Awesome Stylesheet...

SASS – Interpolation

Description

It provides SassScript variables in selectors and property names using **#{ }** syntax. You can specify variables or property names within the curly braces.

Syntax

```
#{$name}
```

Where *$name* is the name of the variable or property name.

Example

The following example demonstrates the use of interpolation in the SCSS file:

```
<html>
<head>
   <title>Interpolation</title>
   <link rel="stylesheet" type="text/css" href="style.css" />
   <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css">
   <script
src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>
   <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/js/bootstrap.min.js"></script>
</head>
<body>
   <div class="container">
   <h2>Example using Interpolation</h2>
   <p>SASS stands for Syntactically Awesome Stylesheet...</p>
   </div>
</body>
</html>
```

Next, create file *style.scss*.

style.scss

```
p:after {
  content: "I have #{8 + 2} books on SASS!";
}
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\style.scss:style.css
```

Next, execute the above command; it will create the *style.css* file automatically with the following code:

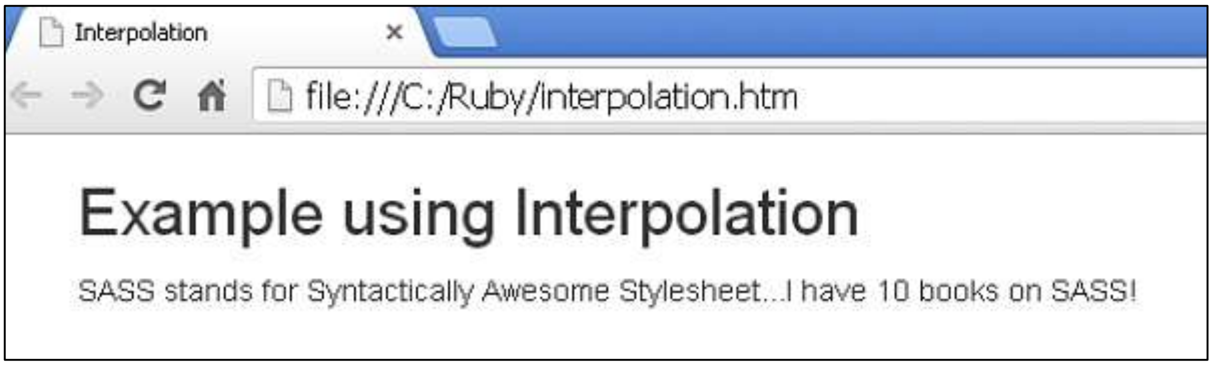
style.css

```
p:after {
  content: "I have 10 books on SASS!";
}
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in **interpolation.html** file.
- Open this HTML file in a browser, an output is displayed as shown below.



Example using Interpolation

SASS stands for Syntactically Awesome Stylesheet...I have 10 books on SASS!

SASS – & in SassScript

Description

You can select the parent selector by using the **&** character. It tells where the parent selector should be inserted.

Syntax

```
a {
    &:hover { color: green; }
}
```

The & character will be replaced with the parent selector a and changes the link color to green when you hover on the link.

Example

The following example demonstrates the use of & in the SCSS file:

```
<html>
<head>
   <title>& in SassScript</title>
   <link rel="stylesheet" type="text/css" href="style.css" />
   <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css">
   <script
src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script
>
   <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/js/bootstrap.min.js"></scr
ipt>
</head>
<body>
   <div class="container">
   <h2>Example using & in SassScript</h2>
   <a href="http://www.tutorialspoint.com/"> www.tutorialspoint.com </a>
   </div>
</body>
</html>
```

Next, create file style.scss.

style.scss

```
a {
    font-size: 20px;
    &:hover { background-color: yellow;}
}
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\style.scss:style.css
```

Next, execute the above command; it will create style.css file automatically with the following code:

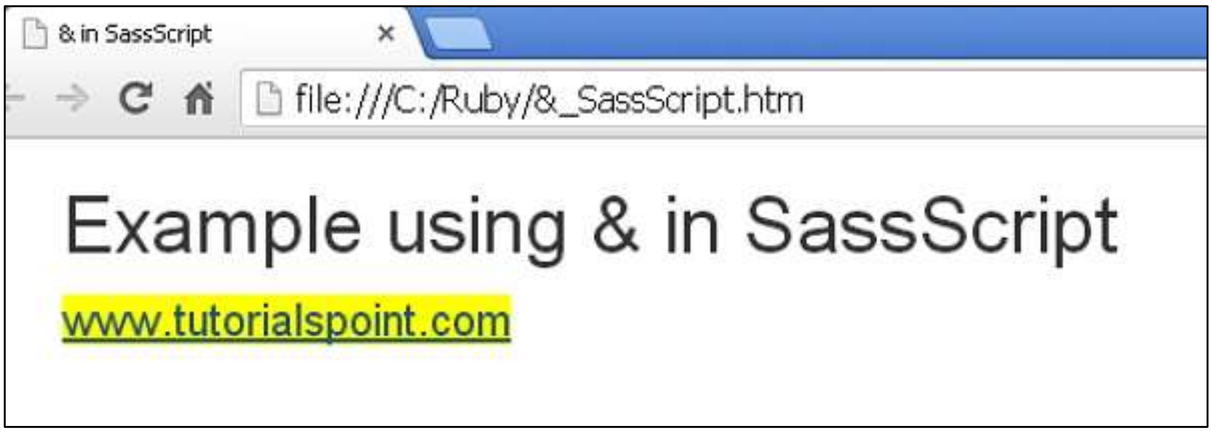
style.css

```
a {
  font-size: 20px;
}
a:hover {
    background-color: yellow;
}
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in **&_SassScript.html** file.
- Open this HTML file in a browser, an output is displayed as shown below.



Example using & in SassScript

www.tutorialspoint.com

SASS – Variable Defaults

Description

You can set the default values for variables by adding *!default* flag to the end of the variable value. It will not re-assign the value, if it is already assigned to the variable.

Example

The following example demonstrates use of variable defaults in the SCSS file:

```
<html>
<head>
   <title>Variable Defaults</title>
   <link rel="stylesheet" type="text/css" href="style.css" />
   <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css">
   <script
src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>
   <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/js/bootstrap.min.js"></scr
ipt>
</head>
<body>
   <div class="container">
   <h2>Example using Variable Defaults</h2>
   <p>Sass is an extension of CSS that adds power and elegance to the basic
language..</p>
   </div>
</body>
</html>
```

Next, create file *style.scss*.

style.scss

```
$myval1: null;
$myval1: "Sass Developed by Natalie Weizenbaum and Chris Eppstein" !default;
p:after {
  content: $myval1;
}
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\style.scss:style.css
```

Next, execute the above command; it will create the *style.css* file automatically with the following code:

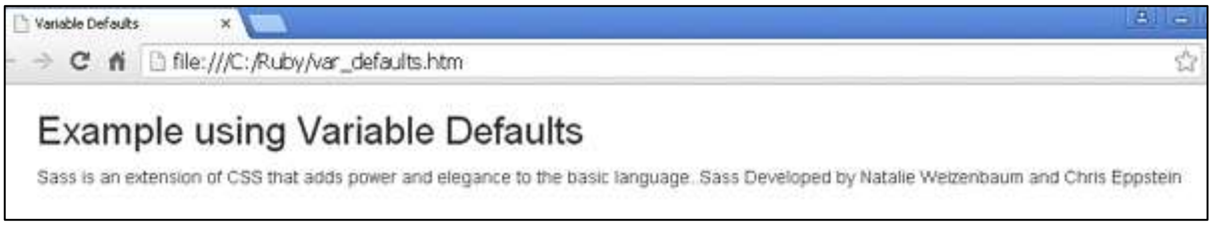
style.css

```
p:after {
  content: "Sass Developed by Natalie Weizenbaum and Chris Eppstein";
}
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in **var_defaults.html** file.
- Open this HTML file in a browser, an output is displayed as shown below.



Example using Variable Defaults

Sass is an extension of CSS that adds power and elegance to the basic language. Sass Developed by Natalie Weizenbaum and Chris Eppstein

8. SASS – @-Rules and Directives

The following table lists all the rules and directives, which you can use in SASS.

S. No.	Directives & Description
1	@import It imports the SASS or SCSS files, it directly takes the filename to import.
2	@media It sets the style rule to different media types.
3	@extend @extend directive is used to share rules and relationships between selectors.
4	@at-root @at-root directive is a collection of nested rules, which is able to make style block at root of the document.
5	@debug @debug directive detects the errors and displays the SassScript expression values to the standard error output stream.
6	@warn @warn directive is used to give cautionary advice about the problem; it displays the SassScript expression values to the standard error output stream.
7	@error @error directive displays the SassScript expression value as fatal error.

Sass – Import Directives

Description

Import directives, imports the SASS or SCSS files. It directly takes the filename to import. All the files which are imported in SASS will get combined in a single CSS file. There are few things that are compiled to a CSS when we use *@import* rule:

- File extension *.css*
- Filename begins with *http://*
- Filename is *url()*
- *@import* consist any media queries.

For example, create one SASS file with the following code:

```
@import "style.css";

@import "http://tutorialspoint.com/bar";

@import url(style);

@import "style" screen;
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\style.scss:style.css
```

The above code will be compiled to the CSS file as shown below:

```
@import url(style.css);

@import "http://tutorialspoint.com/bar";

@import url(style);

@import "style" screen;
```

Following are the ways to import files using *@import* rule:

Partials

Partials are SASS or SCSS files, which are written using underscore at the beginning of the name (_partials.scss). The partial file name can be imported in SASS file without using the underscore. SASS does not compile the CSS file. By using the underscore, it makes SASS understand that it is partial and should not generate the CSS file.

Nested @import

The *@import* directive can be included inside the *@media* rules and CSS rules. The base level file imports the content of the other imported file. The import rule is nested at the same place as the first *@import*.

For instance, create one SASS file with the following code:

```
.container

{

background: #ffff;

}
```

Import the above file to the following SASS file as shown below-

```
h4 {

  @import "example";

}
```

The above code will be compiled to the CSS file as shown below:

```
h4 .container {
  background: #ffff; }
```

Syntax

Given below is a syntax, used to import files, in the SCSS file:

```
@import 'stylesheet'
```

Example

The following example demonstrates the use of *@import* in the SCSS file:

import.htm

```
<html>
<head>
   <title>Import example of sass</title>
   <link rel="stylesheet" type="text/css" href="style.css"/>
</head>
<body class="container">
   <h1>Example using Import</h1>
   <h4>Import the files in SASS</h4>
   <ul>
    <li>Red</li>
    <li>Green</li>
   </ul>
</body>
</html>
```

Next, create file *_partial.scss*.

_partial.scss

```
ul{
    margin: 0;
    padding: 1;
}
li{
    color: #680000;
}
```

Next, create file *style.scss*.

style.scss

```
@import "partial";
.container
{
background: #ffff;
}

h1
{
color: #77C1EF;
}

h4
{
color: #B98D25;
}
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\style.scss:style.css
```

Next, execute the above command; it will create the *style.css* file automatically with the following code:

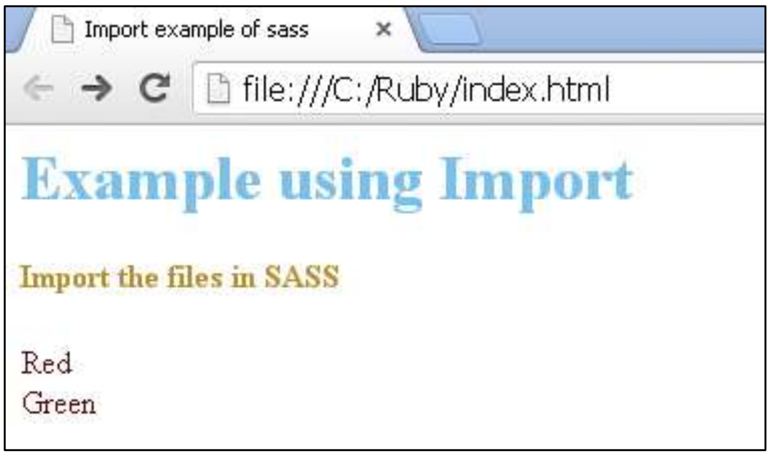
style.css

```
ul {
  margin: 0;
  padding: 1; }

li {
  color: #680000; }

.container {
  background: #ffff; }

h1 {
  color: #77C1EF; }

h4 {
  color: #B98D25; }
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in **import.html** file.
- Open this HTML file in a browser, an output is displayed as shown below.


Import example of sass

file:///C:/Ruby/index.html

Example using Import

Import the files in SASS

Red
Green

Sass – Media Directives

Description

@media directive sets the style rule to different media types. *@media* directive can be nested inside the selector SASS but it is bubbled up to the top level of the stylesheet.

Example

The following example demonstrates the use of *@media* in the SCSS file:

media.htm

```
<!doctype html>
<head>
 <title>Media directive Example</title>
 <link rel="stylesheet" href="media.css" type="text/css" />
</head>
<body class="container">
    <h2>Example using media directive</h2>
    <img src="/sass/images/birds.jpg" class="style">
</body>
</html>
```

Next, create file *media.scss*.

media.scss

```
h2{
color: #77C1EF;
}
.style{
width: 900px;
@media screen and (orientation: portrait){
width:500px;
margin-left: 120px;
}
}
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\media.scss:media.css
```

Nex,t execute the above command; it will create the *media.css* file automatically with the following code:

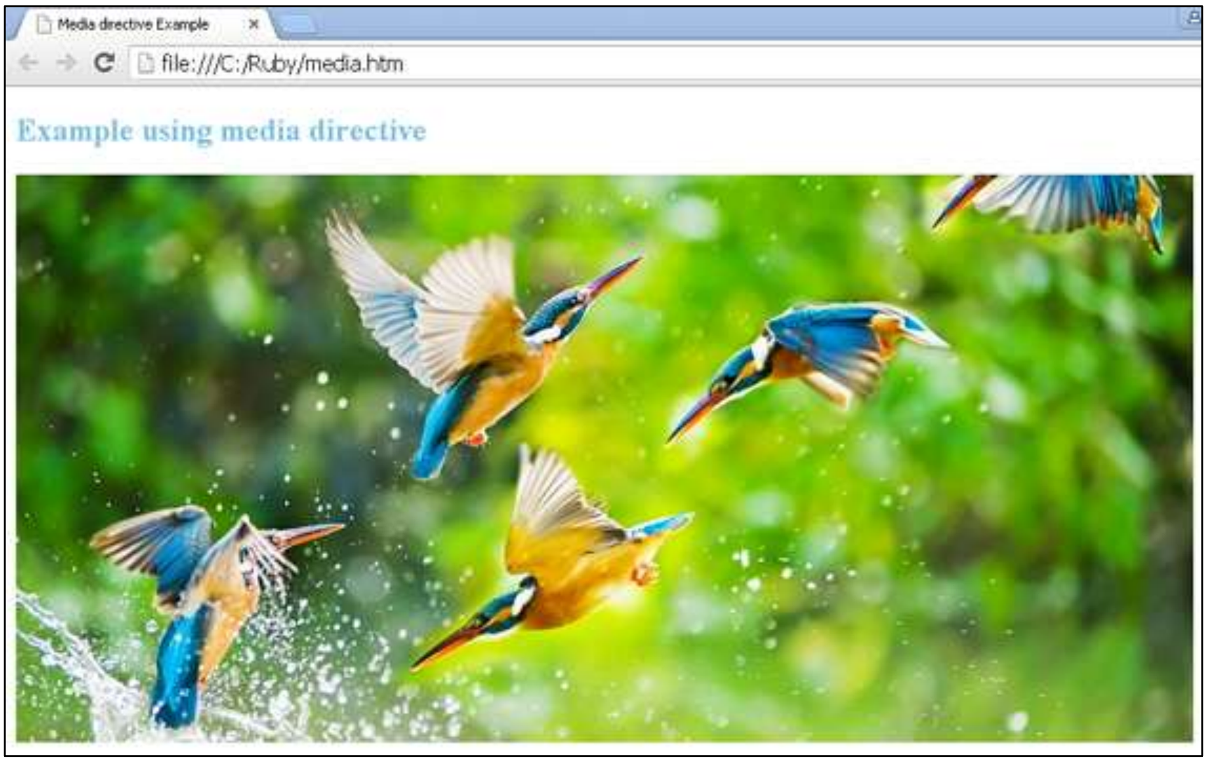
media.css

```
h2 {
  color: #77C1EF;
 }
.style {
  width: 900px;
  }
  @media screen and (orientation: portrait) {
    .style {
      width: 500px;
      margin-left: 120px;
      }
  }
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in **media.html** file.
- Open this HTML file in a browser, an output is displayed as shown below.



Example using media directive

Sass – Extend Directives

Description

The *@extend* directive is used to share rules and relationships between selectors. It can extend all another class styles in one class and can also apply its own specific styles. Following are the types of extend:

Types & Description	Syntax	Compiled code
Extending Complex Selectors It can extend the selector, which consist only a single element or class selector.	`h2{` `font-size: 40px;` `}` `.container{` `@extend h2` `}`	`h2, .container {` `font-size: 40px;` `}`
Multiple Extends More than one selector can be extended by a single selector.	`.style{` `font-size: 25px;` `font-style: italic;` `}` `h2{` `color: #61C8E1;` `}` `.container{` `@extend .style;` `@extend h2` `}`	`.style, .container {` `font-size: 25px;` `font-style: italic;` `}` `h2, .container {` `color: #61C8E1;` `}`

Chaining Extends
The first selector extended by second selector and the second selector is extended by third selector therefore this is known as chaining extends.
	`.style{`
	`font-size: 25px;`
	`font-style: italic;`
	`}`
	
	`h2{`
	`color: #61C8E1;`
	`@extend .style`
	`}`
	
	`.container{`
	`@extend h2`
	`}`
		`.style, h2, .container {`
		`font-size: 25px;`
		`font-style: italic;`
		`}`
		`h2, .container {`
		`color: #61C8E1;`
		`}`

Selector Sequences
The nested selector can use *@extend*by themselves.
Merging Selector Sequences
It merges two sequences i.e. one sequence extend another sequence that is present in other sequence.
	`.style{`
	`font-size: 25px;`
	`font-style: italic;`
	`color: #61C8E1;`
	`}`
	
	`h2 .container {`
	`@extend .style`
	`}`
	
	`.container .style a {`
	`  font-weight: bold;`
	`}`
	`#id .example {`
	`  @extend a;`
	`}`
		`.style, h2 .container {`
		`  font-size: 25px;`
		`  font-style: italic;`
		`  color: #61C8E1;`
		`}`
		
		`.container   .style   a,`
		`.container .style #id`
		`.example,  #id  .container`
		`.style .example {`
		`  font-weight: bold;`
		`}`

@extend - Only Selectors
It percent character(%) can be used anywhere a id or class, it prevents its own ruleset from being rendered to CSS.
	`.style a%extreme {`
	`  font-size: 25px;`
	`  font-style: italic;`
	`  color: #61C8E1;`
	`}`
	
	`.container {`
	`  @extend %extreme;`
	`}`
		`.style a.container {`
		`  font-size: 25px;`
		`  font-style: italic;`
		`  color: #61C8E1;`
		`  }`

The !optional Flag
The !optional flag is used to allow the *@extend* for not to create any new selector.
	`h2.important {`
	`  @extend .style`
	`!optional;`
	`}`
		`A blank compile page gets display.`

@extend in Directives
If *@extend* is used inside the *@media* then it can extend the selectors only that are present within the same directive blocks.
	`@media print {`
	`  .style {`
	`   font-size: 25px;`
	`   font-style: italic;`
	`  }`
	`  .container {`
	`   @extend .style;`
	`   color: #61C8E1;`
	`  }`
	`}`
		`@media print {`
		`  .style, .container {`
		`    font-size: 25px;`
		`    font-style: italic;`
		`}`
		`  .container {`
		`   color: #61C8E1;`
		`  }`
		` }`

Example

The following example demonstrates the use of *@extend* in the SCSS file:

extend.htm

```
<!doctype html>
<head>
 <title>Extend Example</title>
 <link rel="stylesheet" href="extend.css" type="text/css" />
</head>
<body class="container">
    <h2>Example using Extend</h2>
    <p class="style">Lorem Ipsum is simply dummy text of the printing and
typesetting industry.</p>
</body>
</html>
```

Next, create file *extend.scss*.

extend.scss

```
.style{
 font-size: 30px;
 font-style: italic;
}
h2{
 color: #787878;
 @extend .style


}
.container{
 @extend h2
}
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\extend.scss:extend.css
```

Next, execute the above command, it will create *extend.css* file automatically with the following code:

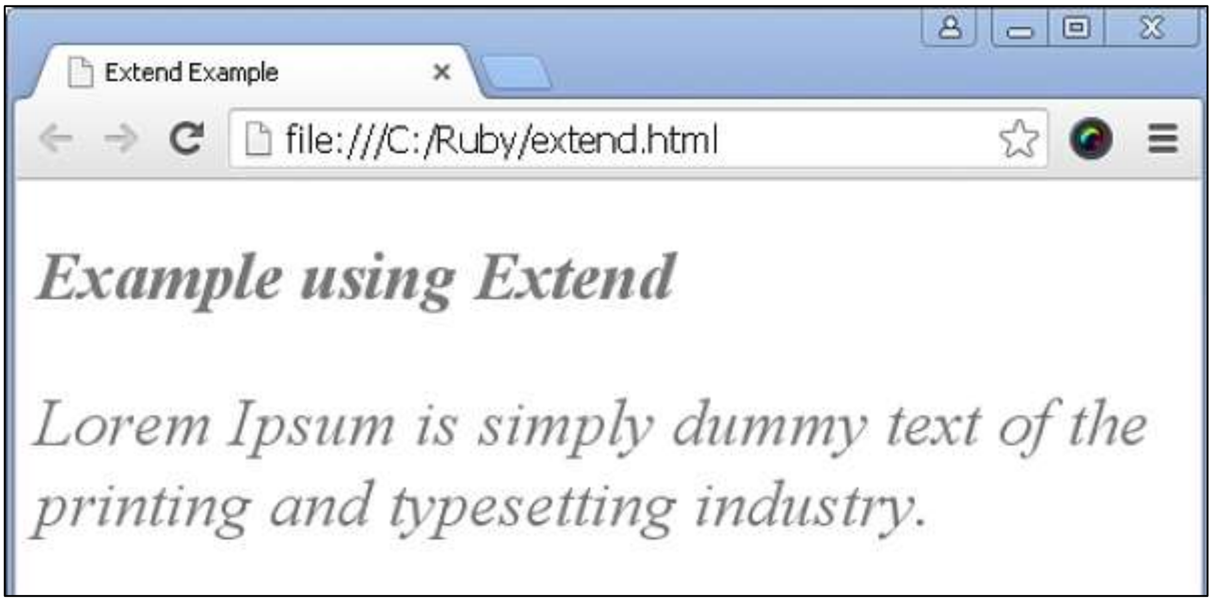
extend.css

```
.style, h2, .container {
  font-size: 30px;
  font-style: italic;
  }
h2, .container {
  color: #787878;
}
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in **extend.html** file.
- Open this HTML file in a browser, an output is displayed as shown below.



Example using Extend

Lorem Ipsum is simply dummy text of the printing and typesetting industry.

Sass – At-root Directives

Description

The *@at-root* directive is a collection of nested rules, which is able to make the style block at root of the document.

@at-root (without: ...) and @at-root (with: ...)

@at-root selector excludes the selector by default. By using *@at-root*, we can move the style outside of nested directive.

For instance, create one SASS file with the following code:

```
@media print {
  .style {
    height: 8px;
    @at-root (without: media) {
      color: #808000;;
    }
  }
```

The above code will be compiled to the CSS file as shown below:

```
@media print {
  .style {
    height: 8px;
    }
 }
.style {
   color: #808000;
}
```

Example

The following example demonstrates the use of *@at-root* in the SCSS file:

atroot.htm

```
<!doctype html>
<head>
 <title>At-root Example</title>
 <link rel="stylesheet" href="atroot.css" type="text/css" />
</head>
<body class="container">
    <h2>Example using at-root</h2>
    <p class="style">Lorem Ipsum is simply dummy text of the printing and
typesetting industry.</p>
</body>
</html>
```

Next, create file *atroot.scss*.

atroot.scss

```
h2{
color: #808000;
background-color: #DB7093;

@at-root {
.style{
 font-size: 20px;
 font-style: bold;
 color: #B8860B;
 }
 }
}
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\atroot.scss:atroot.css
```

Next, execute the above command; it will create the *atroot.css* file automatically with the following code:

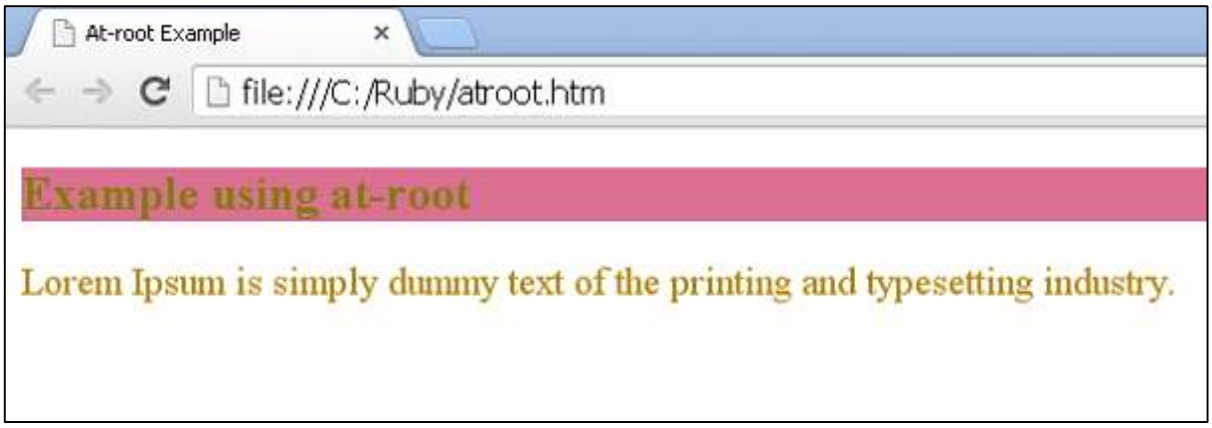
atroot.css

```
h2 {
   color: #808000;
   background-color: #DB7093;
}
.style {
   font-size: 20px;
   font-style: bold;
   color: #B8860B;
}
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in **atroot.html** file.
- Open this HTML file in a browser, an output is displayed as shown below.



Example using at-root

Lorem Ipsum is simply dummy text of the printing and typesetting industry.

Sass – Debug Directives

Description

The @debug directive detects the errors and displays the SassScript expression values to the standard error output stream.

Example

Given below is the stylesheet file saved with extension **.scss,** which is similar to the css file.

debug.scss

```
$font-sizes: 10px + 20px;
$style: (
  color: #bdc3c7
);

.container{
  @debug $style;
  @debug $font-sizes;
}
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

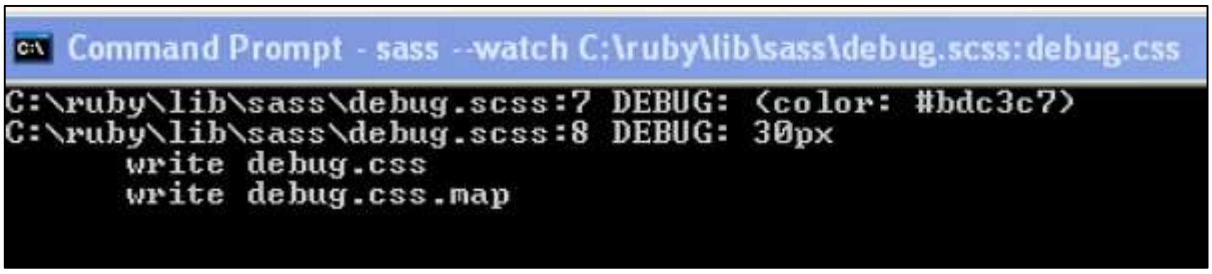
```
sass --watch C:\ruby\lib\sass\debug.scss:debug.css
```

When you run the above command, it will create the *debug.css* file automatically. Whenever you change the SCSS file, the *debug.css* file will be updated automatically.

Output

Let us carry out the following steps to see how the above given code works and gives you a debug error:

- Save the above given code in **debug.scss** file.
- Run the above-mentioned command line in the command prompt.



Command Prompt - sass --watch C:\ruby\lib\sass\debug.scss:debug.css

```
C:\ruby\lib\sass\debug.scss:7 DEBUG: (color: #bdc3c7)
C:\ruby\lib\sass\debug.scss:8 DEBUG: 30px
      write debug.css
      write debug.css.map
```

Sass – Warn Directives

Description

The @warn directive is used to give cautionary advice about the problem. It displays the SassScript expression values to the standard error output stream.

Example

Given below is the stylesheet file saved with extension **.scss** which is similar as css file.

warn.scss

```
$main-color:  #bdc3c7;

@warn "Darker: " darken($main-color, 30%);
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

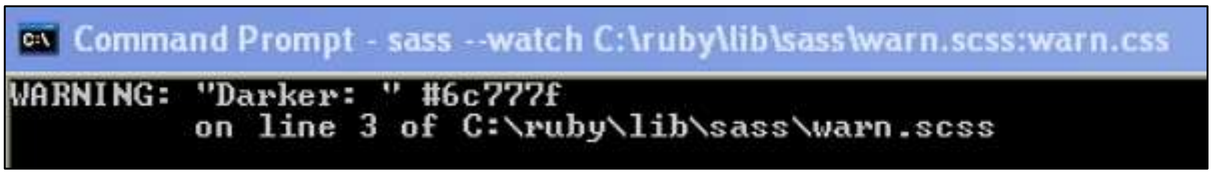
```
sass --watch C:\ruby\lib\sass\warn.scss:warn.css
```

When you run the above command, it will create the *warn.css* file automatically. Whenever you change the SCSS file, the *warn.css* file will be updated automatically.

Output

Let us carry out the following steps to see how the above given code works and gives a warning :

- Save the above given code in **warn.scss** file.
- Run the above-mentioned command line in command prompt.


Command Prompt - sass --watch C:\ruby\lib\sass\warn.scss:warn.css

```
WARNING: "Darker: " #6c777f
         on line 3 of C:\ruby\lib\sass\warn.scss
```

Sass – Error Directives

Description

The @error directive displays the SassScript expression value as fatal error.

Example

Given below is the stylesheet file saved with extension **.scss,** which is similar to the css file.

warn.scss

```
$colors: (
  blue: #c0392b,
  black: #2980b9,


);
@function style-variation($style) {
  @if map-has-key($colors, $style) {
    @return map-get($colors, $style);
  }
  @error "Invalid color: '#{$style}'.";
}
.container {
  style: style-variation(white);
}
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes by using below command:

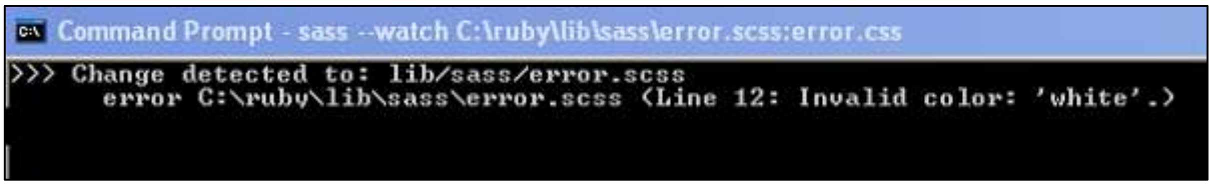
```
sass --watch C:\ruby\lib\sass\warn.scss:warn.css
```

When you run the above command, it will create *error.css* file automatically. Whenever you change the SCSS file, the *error.css* file will be updated automatically.

Output

Let us carry out the following steps to see how the above given code works and gives errors :

- Save the above given code in **error.scss** file.
- Run the above-mentioned command line in command prompt.



```
Command Prompt - sass --watch C:\ruby\lib\sass\error.scss:error.css
>>> Change detected to: lib/sass/error.scss
      error C:\ruby\lib\sass\error.scss (Line 12: Invalid color: 'white'.)
```

9. SASS – Control Directives & Expressions

In this chapter, we will study about **Control Directives & Expressions**. Styling based on some conditions or applying the same style many times with variations can be accomplished by using control directives and expressions, which are supported by SassScript. These control directives are advanced options used mainly in mixins. They require considerable flexibility, as they are a part of Compass libraries.

The following table lists the control directives and expressions used in SASS:

S. No.	Control Directives & Expressions with Description
1	**if()** Based on the condition, *if()* function returns only one result from two possible outcomes.
2	**@if** The *@if* directive accepts SassScript expressions and uses the nested styles whenever the result of the expression is anything other than *false* or *null*.
3	**@for** The *@for* directive allows you to generate styles in a loop.
4	**@each** In *@each* directive, a variable is defined which contains the value of each item in a list.
5	**@while** It takes SassScript expressions and untill the statement evaluates to false it iteratively outputs nested styles.

SASS – if() Function

Description

Based on the condition, this built-in *if()* function returns only one result from two possible outcomes. The result of the function can be referred to the variable that may not be defined or to have further calculations.

Syntax

```
if( expression, value1, value2 )
```

Example

The following example demonstrates the use of *if()* function in the SCSS file:

if_function.html

```
<html>
<head>
   <title>Control Directives & Expressions</title>
   <link rel="stylesheet" type="text/css" href="style.css"/>
</head>
<body>
   <h2>Welcome to TutorialsPoint</h2>
</body>
</html>
```

Next, create file *style.scss*.

style.scss

```
h2{
   color: if( 1 + 1 == 2 , green , red);
}
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\style.scss:style.css
```

Next, execute the above command; it will create the *style.css* file automatically with the following code:

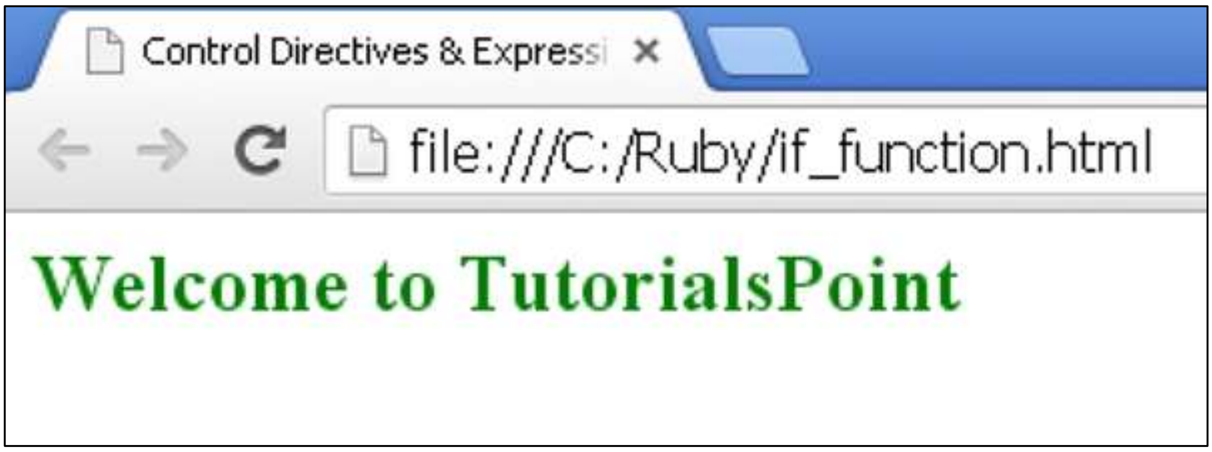
style.css

```
h2 {
  color: green; }
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in *if_function.html* file.
- Open this HTML file in a browser, an output is displayed as shown below.


Control Directives & Express
file:///C:/Ruby/if_function.html

Welcome to TutorialsPoint

SASS – @if Directive

Description

The *@if* directive is used to selectively execute the code statements based on the result of evaluating an expression. The *@if* statements can be followed by several *@else if* statements. Let us study them from the table given below-

S. No.	Types & Description
1	@if It accepts SassScript expressions and uses the nested styles whenever the result of the expression is anything other than false or null.
2	@else if It is used with the *@if* directive, whenever the *@if* statement fails then the *@else if* statements are tried and if they also fails then the *@else* is executed.

SASS – @if Directive

Description

The *@if* directive accepts the SassScript expressions and uses the nested styles whenever the result of the expression is anything other than *false* or *null*.

Syntax

```
@if expression {  //CSS codes are written here }
```

Example

The following example demonstrates the use of *@if* directive in the SCSS file:

```
<html>

<head>
```

```
   <title>Control Directives & Expressions</title>
   <link rel="stylesheet" type="text/css" href="style.css"/>
</head>
<body>
   <div class="container">
   <h2>Example for Control Directive & Expressions</h2>
   <p>SASS stands for Syntactically Awesome Stylesheet. </p>
   </div>
</body>
</html>
```

Next, create file *style.scss*.

style.scss

```
p{
    @if 10 + 10 == 20   { border: 1px dotted;   }
    @if 7 < 2     { border: 2px solid;  }
    @if null    { border: 3px double; }
}
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\style.scss:style.css
```

Next, execute the above command; it will create the *style.css* file automatically with the following code:

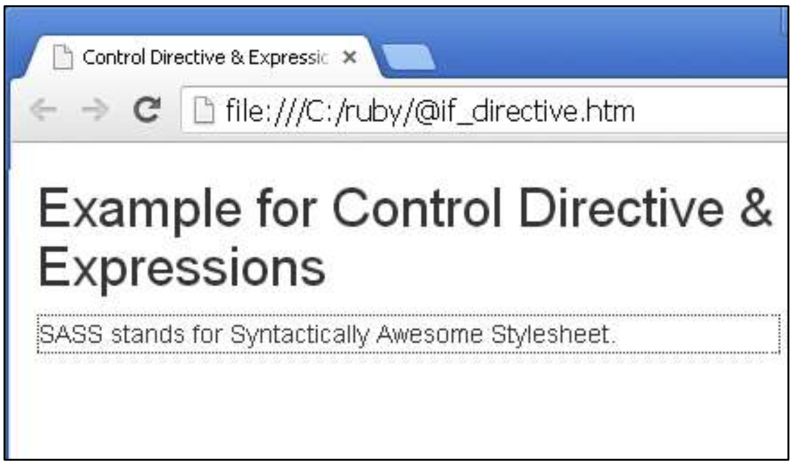
style.css

```
p {
  border: 1px dotted; }
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in ***@if_directive.html*** file.
- Open this HTML file in a browser, an output is displayed as shown below.


Control Directive & Expressic
file:///C:/ruby/@if_directive.htm

Example for Control Directive & Expressions

SASS stands for Syntactically Awesome Stylesheet.

SASS – @else if Directive

Description

The *@else if* statements are used with the *@if* directive. Whenever the *@if* statement fails then the *@else if* statements are tried and if they also fail, then the *@else* is executed.

Syntax

```
@if expression {
   // CSS codes
} @else if condition {
   // CSS codes
} @else {
   // CSS codes
}
```

Example

The following example demonstrates the use of *@else if* directive:

```
<html>
<head>
   <title>Control Directives & Expressions</title>
   <link rel="stylesheet" type="text/css" href="style.css"/>
</head>
<body>
   <div class="container">
   <h1>Example for Control Directive & Expressions</h1>
   <p>SASS stands for Syntactically Awesome Stylesheet.</p>
   </div>
</body>
</html>
```

Next, create file *style.scss*.

style.scss

```
$type: audi;
p {
  @if $type == benz {
    color: red;
  } @else if $type == mahindra {
    color: blue;
  } @else if $type == audi {
    color: green;
  } @else {
    color: black;
  }
}
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\style.scss:style.css
```

Next, execute the above command; it will create the *style.css* file automatically with the following code:

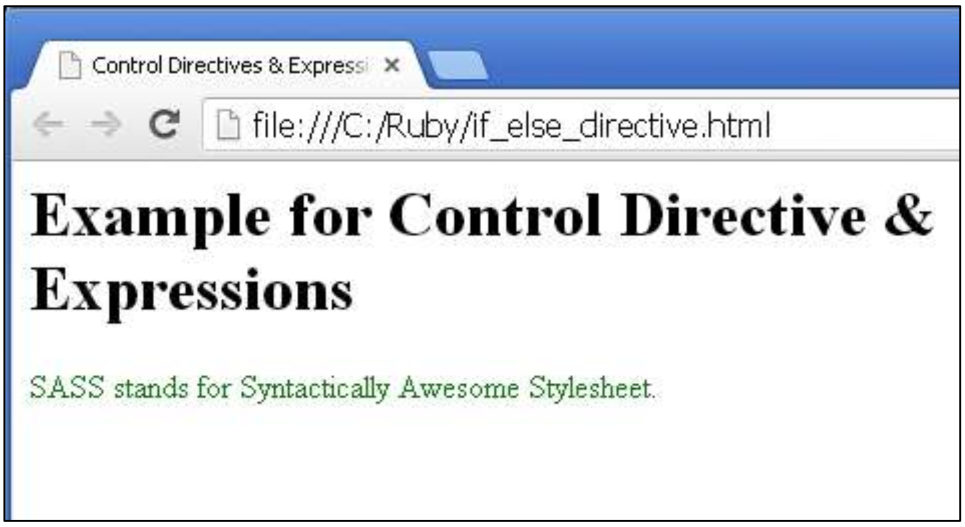
style.css

```
p {
  color: green; }
```

Output

Open this HTML file in a browser, an output is displayed as shown below:

- Save the above given html code in **if_else_directive.html** file.
- Open this HTML file in a browser, an output is displayed as shown below.


Control Directives & Expressi
file:///C:/Ruby/if_else_directive.html

Example for Control Directive & Expressions

SASS stands for Syntactically Awesome Stylesheet.

SASS – @for Directive

Description

The *@for* directive allows you to generate styles in a loop. The counter variable is used to set the output for each iteration.

The *@for* directive are of two types as shown in the table given below:

S. No.	Keywords & Description
1	**through** The *@for* uses the keyword **through** specifies the range including both the values of **<start>** and **<end>**.
2	**to** The *@for* uses **to** keyword specifies the range from **<start>** value to the value before **<end>** value.

SASS – through Keyword

Description

The *@for* directive uses the keyword **through** which specifies the range including both the values of **<start>** and **<end>**

Syntax

```
@for $var from <start> through <end>
```

The syntax is briefly explained below-

- **$var**: It represents the name of the variable like **$i**.
- **<start>** and **<end>**: These are SassScript expressions, which will return integers. If the *<start>* is greater than *<end>* then the counter variable is decremented and when *<start>* is lesser than *<end>* the counter variable will be incremented.

Example

The following example demonstrates the use of *@for* directive with *through* keyword:

```
<html>
<head>
   <title>Control Directives & Expressions</title>
   <link rel="stylesheet" type="text/css" href="style.css"/>
</head>
<body>
   <p class="p1">This is first line.</p>
   <p class="p2">This is second line.</p>
   <p class="p3">This is third line.</p>
   <p class="p4">This is fourth line.</p>
</body>
</html>
```

Next, create file *style.scss*.

style.scss

```
@for $i from 1 through 4 {
  .p#{$i} { padding-left : $i * 10px; }
}
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\style.scss:style.css
```

Next, execute the above command; it will create the *style.css* file automatically with the following code:

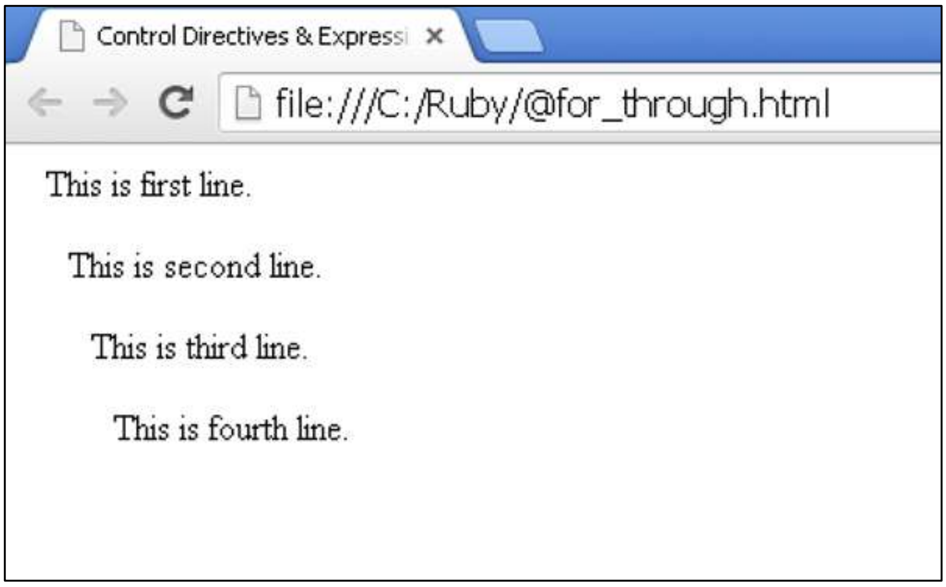
style.css

```
.p1 {
  padding-left: 10px; }


.p2 {
  padding-left: 20px; }


.p3 {
  padding-left: 30px; }


.p4 {
  padding-left: 40px; }
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in **@for_through.html** file.
- Open this HTML file in a browser, an output is displayed as shown below.


Control Directives & Expressi
file:///C:/Ruby/@for_through.html

This is first line.

This is second line.

This is third line.

This is fourth line.

SASS – to Keyword

Description

The *@for* directive uses the keyword **to** which specifies the range from **<start>** value to the value before **<end>** value.

Syntax

```
@for $var from <start> to <end>
```

The syntax is briefly explained below-

- **$var**: It represents the name of the variable like **$i**.
- **<start>** and **<end>**: These are SassScript expressions, which will return integers. If the *<start>* is greater than *<end>* then the counter variable is decremented and when *<start>* is lesser than *<end>* the counter variable will be incremented.

Example

The following example demonstrates the use of *@for* directive with *to* keyword:

```
<html>
<head>
   <title>Control Directives & Expressions</title>
   <link rel="stylesheet" type="text/css" href="style.css"/>
</head>
<body>
   <p class="p1">This is line one.</p>
   <p class="p2">This is line two.</p>
   <p class="p3">This is line three.</p>
   <p class="p4">This is line four.</p>
</body>
</html>
```

Next, create file *style.scss*.

style.scss

```
@for $i from 1 to 4 {
  .p#{$i} { padding-left : $i * 10px; }
}
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\style.scss:style.css
```

Next, execute the above command; it will create *style.css* file automatically with the following code:

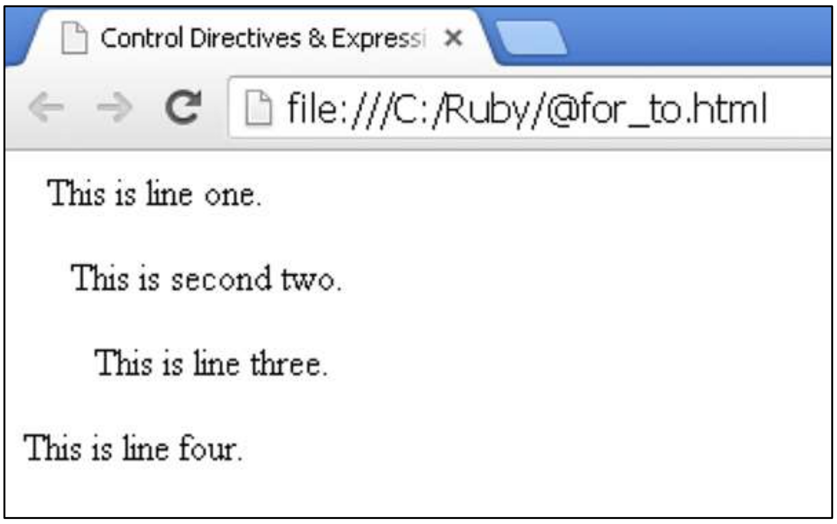
style.css

```
.p1 {
  padding-left: 10px; }

.p2 {
  padding-left: 20px; }

.p3 {
  padding-left: 30px; }
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in **@for_to.html** file.
- Open this HTML file in a browser, an output is displayed as shown below.



This is line one.

This is second two.

This is line three.

This is line four.

SASS – @each Directive

Description

SASS provides *@each* directive with multiple assignments and maps. We will discuss them one by one from the following table.

S. No.	Directive types & Description
1	@each This variable contains the value of each item in a list.

2	Multiple assignments Multiple values can also be used with *@each* directive.
3	Multiple assignments with maps. Multiple assignment works well with maps and they are considered as lists of pairs.

SASS – @each Directive

Description

In *@each*, a variable is defined which contains the value of each item in a list.

Syntax

```
@each $var in <list or map>
```

The syntax is briefly explained below-

- **$var**: It represents the name of the variable. *@each* rule sets **$var** to each item in the list and outputs the styles using the value of **$var**.
- **<list or map>**: These are SassScript expressions, which will return a *list* or a *map*.

Example

The following example demonstrates the use of *@each* directive:

```
<html>
<head>
   <title>Control Directives & Expressions</title>
   <link rel="stylesheet" type="text/css" href="style.css"/>
</head>
<body>
   <p class="p_red">This is line one.</p>
   <p class="p_green">This is line two.</p>
   <p class="p_yellow">This is line three.</p>
   <p class="p_blue">This is line four.</p>
</body>
</html>
```

Next, create file *style.scss*.

style.scss

```
@each $color in red, green, yellow, blue {
  .p_#{$color} {
    background-color: #{$color};
  }
}
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\style.scss:style.css
```

Next, execute the above command; it will create the *style.css* file automatically with the following code:

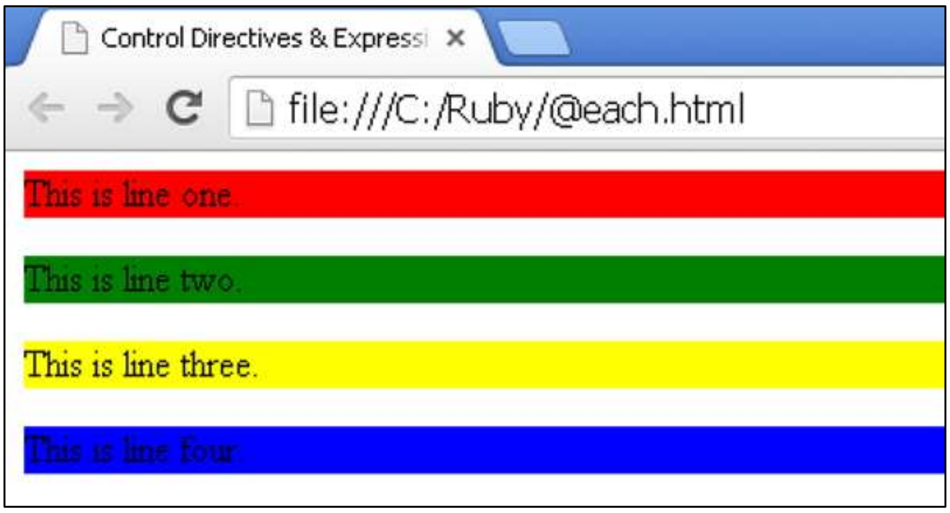
style.css

```
.p_red {
  background-color: red; }

.p_green {
  background-color: green; }

.p_yellow {
  background-color: yellow; }

.p_blue {
  background-color: blue; }
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in **@each.html** file.
- Open this HTML file in a browser, an output is displayed as shown below.



This is line one.

This is line two.

This is line three.

This is line four.

SASS – @each Multiple Assignment

Description

Multiple values can also be used with *@each* directive like **$var1**, **$var2**, **$var3**, **...** in **<list>**.

Syntax

```
@each $var1, $var2, $var3 ... in <list>
```

The syntax is briefly explained below-

- **$var1**, **$var2** and **$var3**: These represent the name of the variables.
- **<list>**: It represents list of lists, each variable will hold the element of the sub-lists.

Example

The following example demonstrates the use of *@each* directive with multiple values:

```
<html>
<head>
   <title>Control Directives & Expressions</title>
   <link rel="stylesheet" type="text/css" href="style.css"/>
</head>
<body>
   <p class="aqua">This is line one.</p>
   <p class="red">This is line two.</p>
   <p class="green">This is line three.</p>
</body>
</html>
```

Next, create file *style.scss*.

style.scss

```
@each $color, $border in (aqua, dotted),
                        (red, solid),
                        (green, double){
  .#{$color} {
    background-color : $color;
    border: $border;
  }
}
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\style.scss:style.css
```

Next, execute the above command; it will create the *style.css* file automatically with the following code:

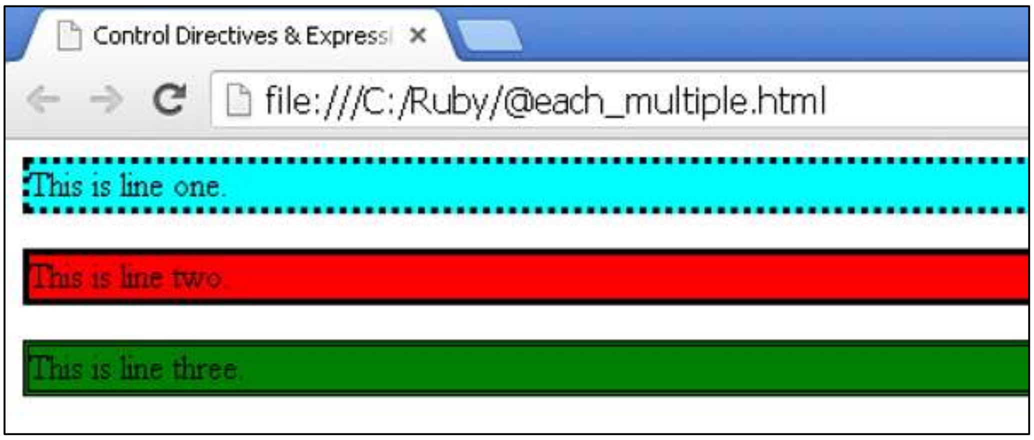
style.css

```
.aqua {
  background-color: aqua;
  border: dotted; }
.red {
  background-color: red;
  border: solid; }
.green {
  background-color: green;
  border: double; }
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in *@each_multiple.html* file.
- Open this HTML file in a browser, an output is displayed as shown below.


Control Directives & Express
file:///C:/Ruby/@each_multiple.html

This is line one.

This is line two.

This is line three.

SASS – @each Multiple Assignment with Maps

Description

The multiple assignment works well with maps and they are considered as lists of pairs. If you want to use the map, then you have to change *@each* statement and use multiple assignments.

Syntax

```
@each $var1, $var2 in <map>
```

The syntax is briefly explained below-

- **$var1**, **$var2**: These represents the name of the variables.
- **<map>**: It represents lists of pair.

Example

The following example demonstrates the use of multiple assignment with maps:

```
<html>
<head>
   <title>Control Directives & Expressions</title>
   <link rel="stylesheet" type="text/css" href="style.css"/>
</head>
<body>
   <h1>Welcome to Tutorialspoint</p>
   <h2>Welcome to Tutorialspoint</p>
   <h3>Welcome to Tutorialspoint</p>
</body>
</html>
```

Next, create file *style.scss*.

style.scss

```
@each $header, $color in (h1: red, h2: green, h3: blue) {
  #{$header} {
    color: $color;
  }
}
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\style.scss:style.css
```

Next, execute the above command; it will create the *style.css* file automatically with the following code:

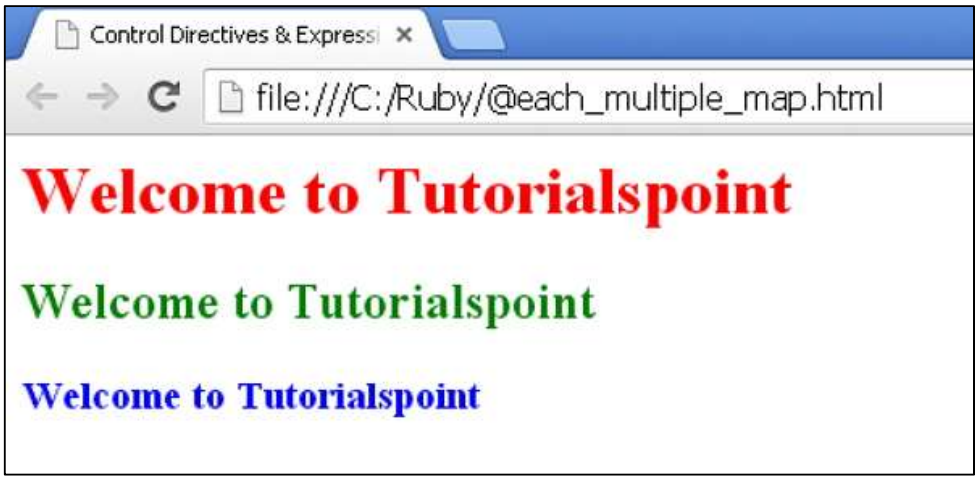
style.css

```
h1 {
  color: red; }
h2 {
  color: green; }
h3 {
  color: blue; }
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in *@each_multiple_map.html* file.
- Open this HTML file in a browser, an output is displayed as shown below.


Control Directives & Express
file:///C:/Ruby/@each_multiple_map.html

Welcome to Tutorialspoint

Welcome to Tutorialspoint

Welcome to Tutorialspoint

SASS – @while Directive

Description

Just as other control directives, the *@while* directive also takes SassScript expressions and until the statement evaluates to false, it iteratively outputs nested styles. The key thing to note is that counter variable needs to be incremented/decremented on each iteration.

Syntax

```
while(condition) {
  // CSS codes
}
```

Example

The following example demonstrates the use of *@while* directive:

```
<html>
<head>
   <title>Control Directives & Expressions</title>
   <link rel="stylesheet" type="text/css" href="style.css"/>
</head>
<body>
   <p class="paddding-50">This is line one with left padding 50. </p>
   <p class="paddding-40">This is line two with left padding 40.</p>
   <p class="paddding-30">This is line three with left padding 30. </p>
   <p class="paddding-20">This is line four with left padding 20. </p>
   <p class="paddding-10">This is line five with left padding 10. </p>
</body>
</html>
```

Next, create file *style.scss*.

style.scss

```
$i: 50;
@while $i > 0 {
  .paddding-#{$i} { padding-left: 1px * $i; }
  $i: $i - 10;
}
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\style.scss:style.css
```

Next, execute the above command; it will create the *style.css* file automatically with the following code:

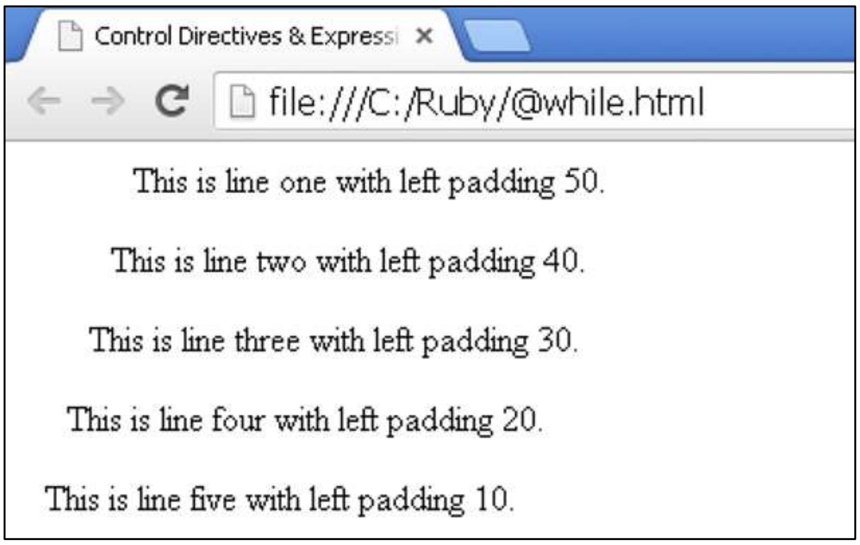
style.css

```
.paddding-50 {
  padding-left: 50px; }

.paddding-40 {
  padding-left: 40px; }

.paddding-30 {
  padding-left: 30px; }

.paddding-20 {
  padding-left: 20px; }

.paddding-10 {
  padding-left: 10px; }
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in **@while.html** file.
- Open this HTML file in a browser, an output is displayed as shown below.



This is line one with left padding 50.

This is line two with left padding 40.

This is line three with left padding 30.

This is line four with left padding 20.

This is line five with left padding 10.

10. SASS – Mixin Directives

Mixins allow creating a group of styles, which are reusable throughout your stylesheet without any need to recreation of non-semantic classes. In CSS, the mixins can store multiple values or parameters and call function; it helps to avoid writing repetitive codes. Mixin names can use underscores and hyphens interchangeably. Following are the directives present in Mixins-

S. No.	Directives & Description
1	Defining a Mixin *@mixin* directive is used to define the mixin.
2	Including a Mixin *@include* directive is used to include the mixins in the document.
3	Arguments The SassScript values can be taken as arguments in mixins, which is given when mixin is included and available as variable within the mixin.
4	Passing Content Blocks to a Mixin Block of styles are passed to the mixin.

Sass – Defining a Mixin

Description

The *@mixin* directive is used to define the mixins. It includes optionally the variables and argument after the name of the mixin.

Example

The following example demonstrates the use of *@mixin* in the SCSS file:

sample.htm

```
<html>
<head>
   <title> Mixin example of sass</title>
   <link rel="stylesheet" type="text/css" href="sample.css"/>
</head>
<body>
<div class="cont">
   <h1>Example using include</h1>
   <h3>Directive is used to define the Mixins, it includes variables and argument optionally.</h3>
</div>
</body>
</html>
```

Next, create file *sample.scss*.

sample.scss

```
@mixin style {
.cont{
 color: #77C1EF;
    }
}
@include style;
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\sample.scss:sample.css
```

Next, execute the above command; it will create the *sample.css* file automatically with the following code:

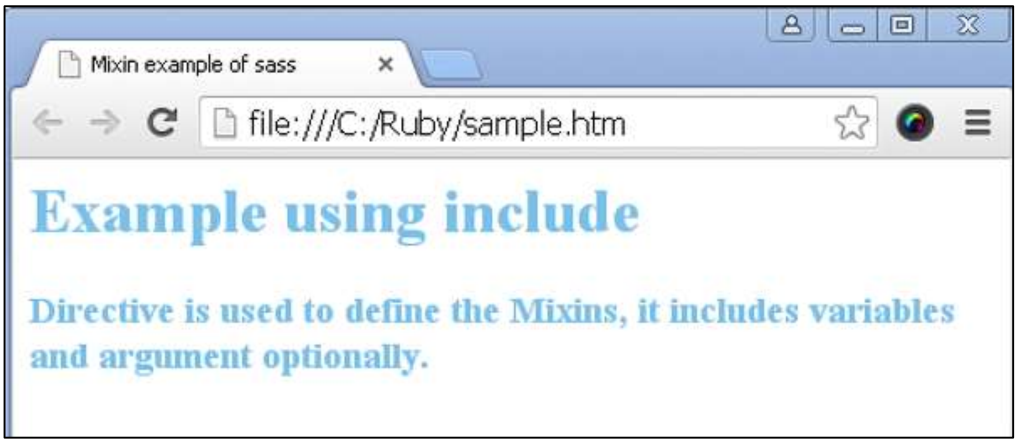
sample.css

```
.cont {
  color: #77C1EF;
  }
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in **sample.htm** file.
- Open this HTML file in a browser, an output is displayed as shown below.



Example using include

Directive is used to define the Mixins, it includes variables and argument optionally.

Sass – Including a Mixin

Description

The *@include* directive is used to include the mixin in the document. The name of mixin is taken and optional arguments are passed into it. The styles defined by the mixin can be included into the current rule.

Example

The following example demonstrates the use of including a mixin in the SCSS file:

sample.htm

```
<html>
<head>
   <title> Mixin example of sass</title>
   <link rel="stylesheet" type="text/css" href="sample.css"/>
</head>
<body>
<div class="cont">
   <h2>Example using include</h2>
   <h3>Different Colors</h3>
   <ul>
    <li>Red</li>
    <li>Green</li>
    <li>Blue</li>
   </ul>
  </div>
</body>
</html>
```

Next, create file *sample.scss*.

sample.scss

```
@mixin style {
.cont{
 background-color: #77C1EF;
 color: #ffffff;
    }
h3 {
 color: #ffffff;
 }
}
@include style;
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\sample.scss:sample.css
```

Next, execute the above command; it will create *sample.css* file automatically with the following code:

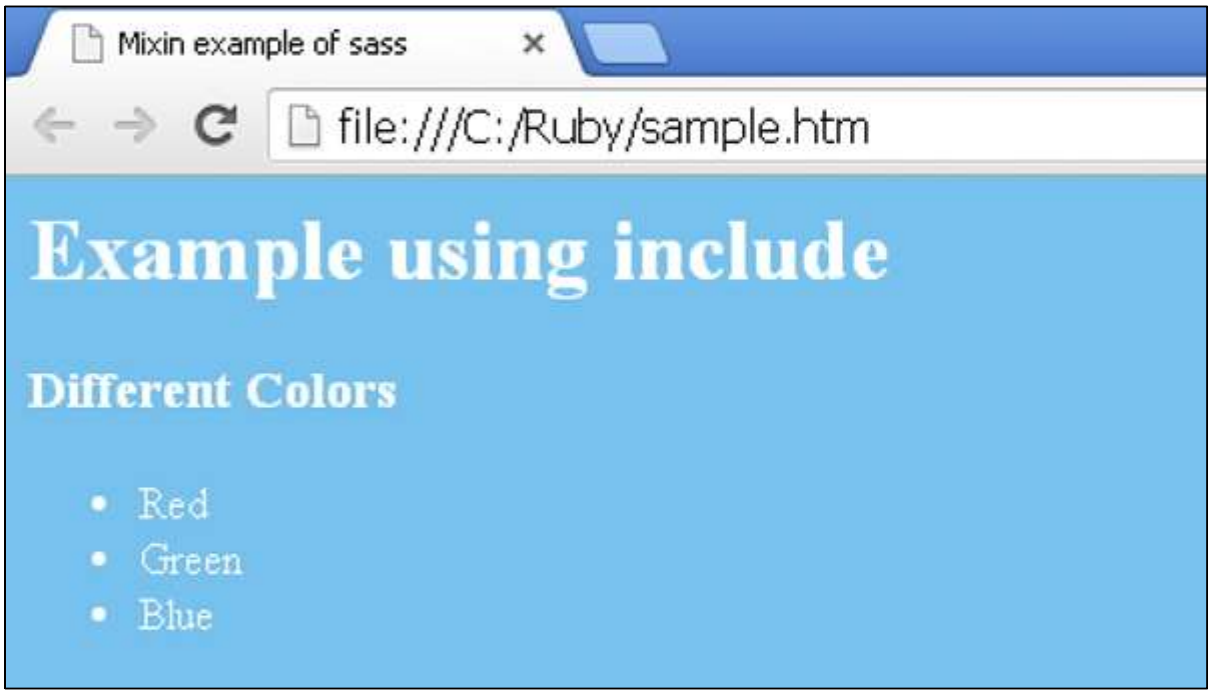
sample.css

```
.cont {
  background-color: #77C1EF;
  color: #ffffff;
 }
h3 {
  color: #ffffff;
 }
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in **sample.htm** file.
- Open this HTML file in a browser, an output is displayed as shown below.



Example using include

Different Colors

- Red
- Green
- Blue

Sass – Mixin Arguments

Description

The SassScript values can be taken as arguments in mixins, which are passed when mixins are included and are available as variable within the mixin. The argument is a name of a variable, which is separated by comma while defining a mixin. There are two types of arguments such as:

- Keyword Arguments
- Variable Arguments

Keyword Arguments

Explicit keyword argument can be used to include in mixins. The arguments, which are named can be passed in any order and the default values of argument can be omitted.

For instance, create one SASS file with the following code:

```
@mixin bordered($color, $width: 2px) {
  color: #77C1EF;
  border: $width solid black;
  width: 450px;
}
.style  {
  @include bordered($color:#77C1EF, $width: 2px);
}
```

The above code will be compiled to the CSS file as shown below:

```
.style {
  color: #77C1EF;
  border: 2px solid black;
  width: 450px;
 }
```

Variable Arguments

Variable argument is used to pass any number of arguments to mixin. It contains keyword arguments passed to the function or mixin. Keyword arguments passed to the mixin can be accessed using *keywords($args) function* which return values mapped to String.

For instance, create one SASS file with the following code:

```
@mixin colors($background) {
  background-color: $background;
}
$values: magenta, red, orange;
.container {
  @include colors($values...);
}
```

The above code will be compiled to the CSS file as shown below:

```
.container {
  background-color: magenta;
}
```

Example

The following example demonstrates the use of arguments in the SCSS file:

argument.htm

```
<html>
<head>
   <title> Mixin example of sass</title>
   <link rel="stylesheet" type="text/css" href="argument.css"/>
</head>
<body>
   <div class="style">
   <h1>Example using arguments</h1>
   <p>Different Colors</p>
   <ul>
    <li>Red</li>
    <li>Green</li>
    <li>Blue</li>
   </ul>
   </div>
</body>
</html>
```

Next, create file *argument.scss*.

argument.scss

```
@mixin bordered($width: 2px) {
  background-color: #77C1EF;
  border: $width solid black;
  width: 450px;
}
.style  {
  @include bordered(2px);
}
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\argument.scss:argument.css
```

Next, execute the above command; it will create the *argument.css* file automatically with the following code:

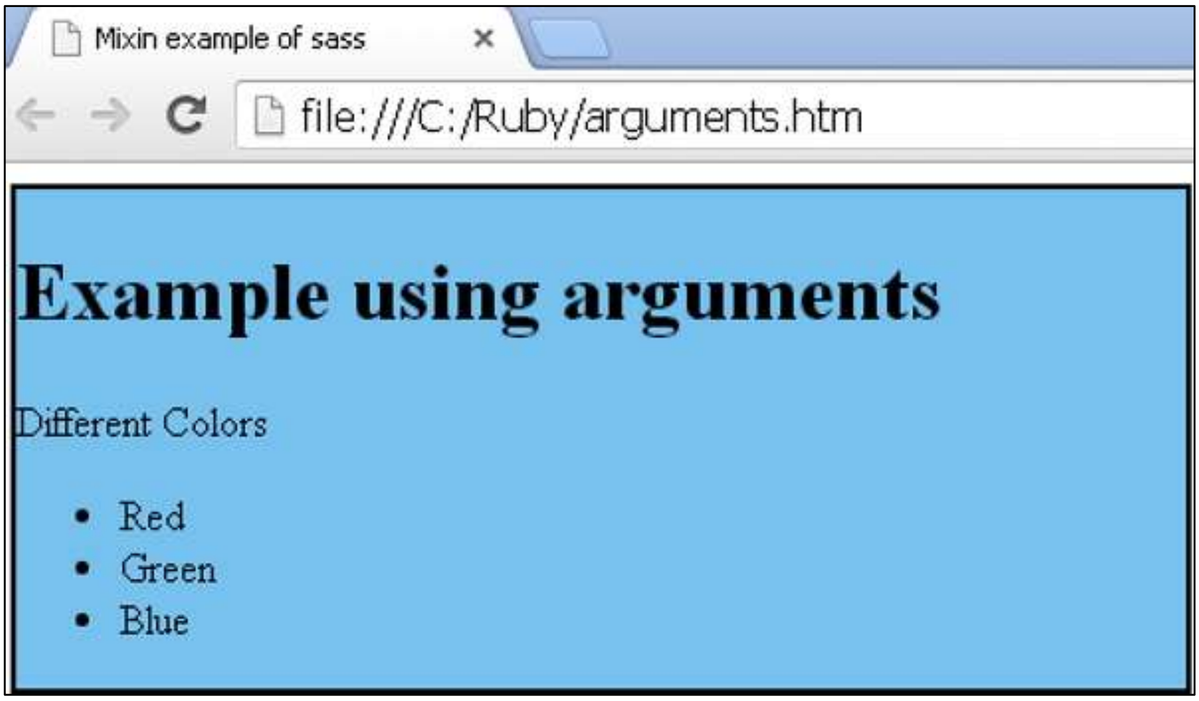
style.css

```
.style {
  background-color: #77C1EF;
  border: 2px solid black;
  width: 450px;
 }
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in **argument.htm** file.
- Open this HTML file in a browser, an output is displayed as shown below.



Example using arguments

Different Colors

- Red
- Green
- Blue

Sass – Passing Content Blocks to a Mixin

Description

Block of styles is passed to the mixin for the placement inside the styles. In *@content* directive location, styles gets included into the mixin.

Variable Scope and Content Blocks

The block of content is evaluated in the scope, which is passed to a mixin where block is defined.

Example

The following example demonstrates the use of passing content blocks to mixin in the SCSS file:

pass_content.htm

```
<html>
<head>
   <title>Mixin example of sass</title>
   <link rel="stylesheet" type="text/css" href="sample.css"/>
```

```
</head>
<body>
<div class="block">
     <h1>Example using passing content blocks</h1>
     <p>Different Colors</p>
     <ul>
       <li>Red</li>
       <li>Green</li>
       <li>Blue</li>
     </ul>
</div>
</body>
</html>
```

Next, create file *sample.scss*.

sample.scss

```
@mixin element{
   @content;
}
@include element{
  .block{
    color: green;
  }
}
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\sample.scss:sample.css
```

Next, execute the above command; it will create the *sample.css* file automatically with the following code:

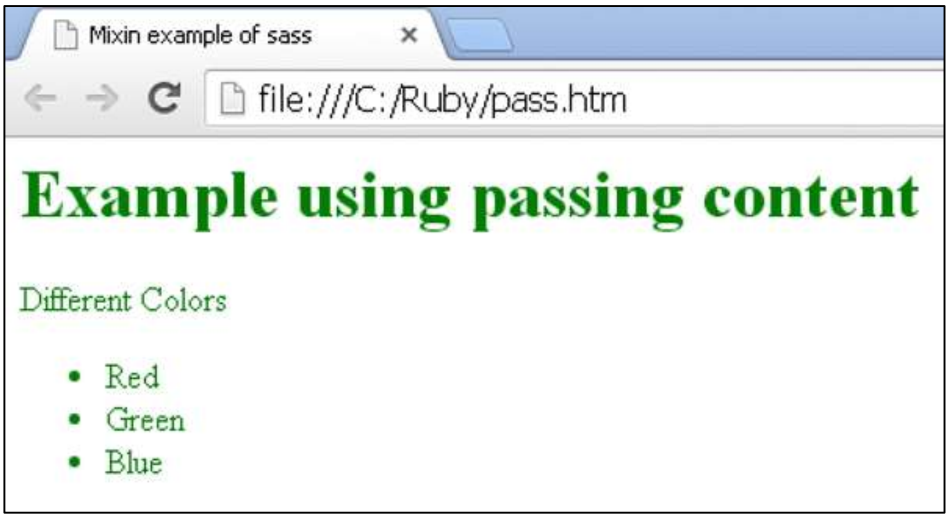
sample.css

```
.block {
  color: green;
}
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in **pass_content.scss** file.
- Open this HTML file in a browser, an output is displayed as shown below.



Example using passing content

Different Colors

- Red
- Green
- Blue

11. SASS – Function Directives

In this chapter, we will study about **Function Directives**. In SASS, you can create your own function and use them in your script context or can be used with any value. Functions are called by using the function name and with any parameters.

Example

The following example demonstrates the use of function directive in the SCSS file:

function_directive.htm

```
<html>
<head>
   <title>Nested Rules</title>
   <link rel="stylesheet" type="text/css" href="style.css" />
   <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css">
   <script
src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>
   <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/js/bootstrap.min.js"></scr
ipt>
</head>
<body>
   <div class="container" id="set_width">
   <h2>Example for Function directives</h2>
   <p>SASS stands for Syntactically Awesome Stylesheet. </p>
   </div>
</body>
</html>
```

Next, create file *style.scss*.

style.scss

```
$first-width: 5px;
$second-width: 5px;
@function adjust_width($n) {
  @return $n * $first-width + ($n - 1) * $second-width;
}
#set_width { padding-left: adjust_width(10); }
```

You can tell SASS to watch the file and update the CSS whenever SASS file changes, by using the following command:

```
sass --watch C:\ruby\lib\sass\style.scss:style.css
```

Next, execute the above command; it will create the *style.css* file automatically with the following code:

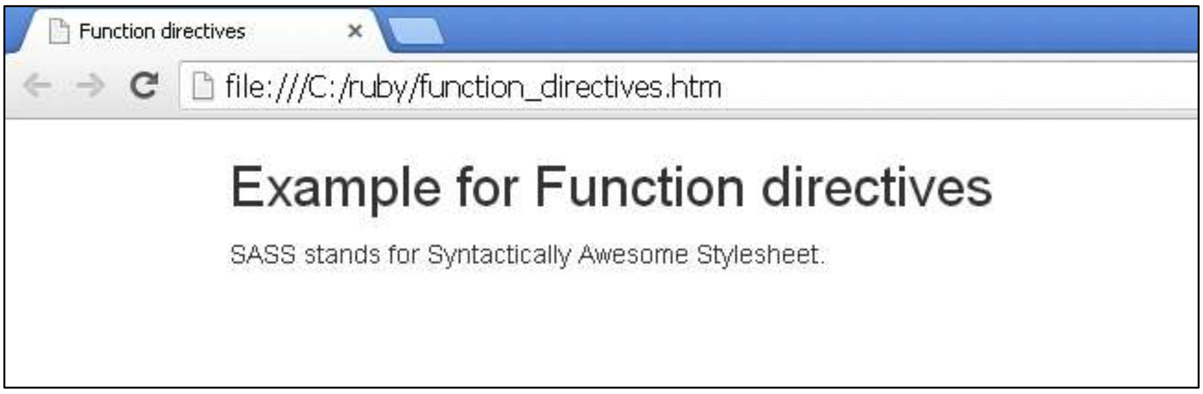
style.css

```
#set_width {
  padding-left: 95px; }
```

Output

Let us carry out the following steps to see how the above given code works:

- Save the above given html code in **function_directive.html** file.
- Open this HTML file in a browser, an output is displayed as shown below.



Example for Function directives

SASS stands for Syntactically Awesome Stylesheet.

In the output, you can see that the left-padding is being applied.

Just like mixin, function can also access globally defined variables and can also accept parameters. You should call the return value for the function by using **@return**. We can call the SASS-defined functions by using keyword parameters.

Call the above function as shown below.

```
#set_width { padding-left: adjust_width($n: 10); }
```

Naming Conventions

To avoid naming conflicts, function names can be prefixed so that they can be easily differentiated. Like mixins, variable arguments are also supported by user-defined functions. Functions and other SASS identifiers can use underscores(_) and hyphens(-) interchangeably.

For example, if a function is defined as **adjust_width**, it can be used as **adjust-width**, and vice versa.

12. SASS – Output Style

In this chapter, we will study about **SASS Output Style**. The CSS file that the SASS generates consists of default CSS style, which reflects the structure of document. The default CSS styling is good but might not be suitable for all situations; on other hand, SASS supports many other styles.

It supports the following different output styles:

:nested

Nested style is default styling of SASS. This way of styling is very useful when you are dealing with large CSS files. It makes the structure of the file more readable and can be easily understood. Every property takes its own line and indentation of each rule is based on how deeply it is nested.

For instance, we can nest the code in SASS file as shown below:

```
#first {
  background-color: #00FFFF;
  color: #C0C0C0; }
  #first p {
    width: 10em; }

.highlight {
  text-decoration: underline;
  font-size: 5em;
  background-color: #FFFF00; }
```

:expanded

In expanded type of CSS styling each property and rule has its own line. It takes more space compared to the Nested CSS style. The Rules section consists of properties, which are all intended within the rules, whereas rules does not follow any indentation.

For instance, we can expand the code in the SASS file as shown below:

```
#first {
  background-color: #00FFFF;
  color: #C0C0C0;
}
#first p {
  width: 10em;
}
.highlight {
  text-decoration: underline;
  font-size: 5em;
  background-color: #FFFF00;
}
```

:compact

Compact CSS style competitively takes less space than Expanded and Nested. It focuses mainly on selectors rather than its properties. Each selector takes up one line and its properties are also placed in the same line. Nested rules are positioned next to each other without a newline and the separate groups of rules will have new lines between them.

For instance, we can compact the code in the SASS file as shown below:

```
#first { background-color: #00FFFF; color: #C0C0C0; }
#first p { width: 10em; }
.highlight { text-decoration: underline; font-size: 5em; background-color: #FFFF00; }
```

:compressed

Compressed CSS style takes the least amount of space compared to all other styles discussed above. It provides whitespaces only to separate selectors and newline at the end of the file. This way of styling is confusing and is not easily readable.

For instance, we can compress the code in SASS file as shown below:

```
#first{background-color:#00FFFF;color:#C0C0C0}#first
p{width:10em}.highlight{text-decoration:underline;font-size:5em;background-
color:#FFFF00}
```

13. Extending SASS

You can extend the functionality of SASS to provide different types of features and customizations for users. To make use of these features, user should have knowledge of Ruby.

Defining Custom SASS Functions

You can define your own SASS functions while using Ruby API. You can add your custom functions by adding them to Ruby methods as shown in the following code:

```
module Sass::Script::Functions
  def reverse(string)
    assert_type string, :String
    Sass::Script::Value::String.new(string.value.reverse)
  end
  declare :reverse, [:string]
end
```

In the code you could see, the Function, declare, specifies the argument names for the function. If it fails then it will not accept any arguments even if the function is working and it also takes arbitrary keyword arguments. You can get Ruby values by using *value* accessor and access the color objects by using *rgb*, *red*, *green*, or *blue*.

Cache Stores

SASS stores cache of parsed documents, which can be reused without parsing again. SASS uses **:cache_location** to write cache files on the file system. It makes compilation of SASS files faster and if you delete cached files, they will be generated again when you compile next time. You can define your own cache store by setting the **:cache_store** option. This will write cache files on the file system or share cache files to ruby processes or machines. SASS uses instance of subclass of *Sass::CacheStores::Base* to store and retrieve cache results.

Custom Importers

SASS uses *@import* to import SCSS and SASS files and passes paths to *@import* rule to find an appropriate path code for specified paths. SASS importers use file system for loading the code and added to the load using database or different file naming scheme.

Single importer can take single file loading and can be placed in *:load_paths* array along with the paths of file system. While using *@import*, SASS looks for loaded paths, which import the path for the importer. When the path is found, the imported file is used. A user can inherit the importers from **Sass::Importers::Base**.