Preprocessors

1 Preprocessors for CSS - Sass & SCSS

SCSS & **Sass**

These are preprocessing languages that are compiled to CSS. They are similar to Haml in the sense that they make writing code easier. Both come from the same origin but they have technical different syntaxes.

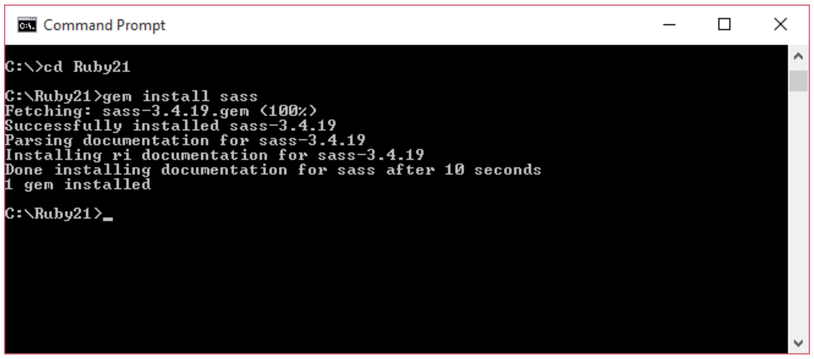
Sass = **S**yntactically **A**wesome **S**tylesheets – this came first and has a very strict indented syntax. A little bit after Sass, came SCSS, **S**assy **C**SS that provide the same power of Sass but with a more flexible syntax including the ability to write plain CSS.

In this course we will basically focus more on Sass.

2 Installing Sass

This preprocessor is also compiled using Ruby (SCSS is also compiled using Ruby), so, to use Sass, you need to install Ruby. After your install Ruby, you need to run the following command to install SCSS and Sass:
gem install sass

You might get this screen after it's installed – here the installation happened in a Windows 10 laptop and Ruby had been installed in a folder called Ruby21 – that's why you see the first command being ***cd Ruby21*** to change from the root of the C (hard drive) to that Ruby21 folder and only then I could use the command to install Sass.



```
C:\>cd Ruby21

C:\Ruby21>gem install sass
Fetching: sass-3.4.19.gem (100%)
Successfully installed sass-3.4.19
Parsing documentation for sass-3.4.19
Installing ri documentation for sass-3.4.19
Done installing documentation for sass after 10 seconds
1 gem installed

C:\Ruby21>_
```

NOTE: Again, if you are working in a Mac, you might need to type **sudo** in front of the command shown here to make sure you install the gem as a "super user" and not get blocked.

Files written in SCSS or Sass need to have the **.scss** or **.sass** extensions respectively.

When you want to convert one of these files to a **.css**, you should use the following command:
sass mystyles.sass mystyles.css

Remember that these file names are paths to where the files are/will be located. You need to specify the right path to the files in the command as the above command is assuming that both files are in the same folder.

Should you have ongoing changes, Sass can watch the file and recompile the CSS every time a change takes place – in order for this to happen, the following sass command needs to be run:
sass --watch mystyles.sass:mystyles.css

Or you can have the following command:
sass --watch styles/sass :wwwroot/css

In the command above, the watching will happen in the entire folder of Sass files and the compilation will happen for the respective CSS files at the wwwroot folder.

If by any chance, you receive a SCSS file, you can convert it to Sass to then convert to CSS. To convert to Sass, you would use the following command:
sass-convert styles.scss styles.sass

If you use the command below, you will be converting from Sass to SCSS
sass-convert styles.sass styles.scss

3 Syntax - Part 1

Sass has a very rigid syntax and any indenting or character errors will prohibit the styles from compiling. Sass omits all curly brackets, semicolons and relies on indentation and clear line breaks for formatting purposes. SCSS, on the other hand, is more similar to CSS and more flexible than Sass.

Example of SCSS

```
.main {
  color: red;
  font-style: italic;
  span {
    text-decoration: underline;
  }

}
```

Same example in Sass

```
.main
  color: red
  font-style: italic
  span
    text-decoration: underline
```

The compiled CSS would be:

```
.main {
  color: red;
  font-style: italic;
}
.main span {
  text-decoration: underline;
}
```

Which one to use will be a matter of preference. It seems that in the market, Sass has been a stronger choice and I believe it’s because it requires less characters and provides a cleaner syntax but, on the other hand, remember that it is very strict and have a steeper learning curve than SCSS

Be careful when nesting selectors as you should aim to be using specific selectors without raising specificity (remember the first chapters about performance).

Here is one example in Sass:

```
.nav
  background-color: cyan
  ul
    list-style-type: none
  li
    float:left
```

The compiled CSS would be:

```
.nav {
  background-color: cyan;
}
.nav ul {
  list-style-type: none;
}
.nav li {
  float: left;
}
```

Remember that you can also nest properties, not only selectors. The most popular uses of this may be seen with font, margin, padding and border.
One example here (the indentation should be :

```
section
  font:
    family: Arial, Verdana, sans-serif
    style: italic
    weight: bold
```

The compiled CSS would be:

```
section {
  font-family: Arial, Verdana, sans-serif;
  font-style: italic;
  font-weight: bold;
}
```

What if you want to use media queries?

```
.container
  width: 850px;
  @media screen and (max-width: 480px)
    width: 100%
```

Compiled CSS:

```
.container {
  width: 850px;
}
@media screen and (max-width: 480px) {
  .container {
    width: 100%;
  }
}
```

Parent Selector – Most commonly, the parent selector is used together with a pseudo class such as **:hover** but it does not need to be only that way as Sass provides a way to add styles to a previous selector with the use of the parent selector by using the ampersand **&**

Example:

```
a
  color: green
  &:hover
    color: orange
```

Compiled CSS:

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

The parent selector may also be used as the key selector, adding qualifying selectors to make compound selectors.

Example:

```
.navbtn
  background: linear-gradient(#7B7E90, #A5A7B3)
  .nogradient &
    background: url("gradient.png") 0 0 repeat-x
```

Compiled CSS:

```
.navbtn {
  background: linear-gradient(#7B7E90, #A5A7B3);
}
.nogradient .navbtn {
  background: url("gradient.png") 0 0 repeat-x;
}
```

Comments – are handled in a very similar way of Haml. The usual ***/**** and ****/*** work as intended within Sass but there is also a syntax for silent comments to completely remove a comment or lines of code from being compiled. The syntax for silent comments is two forward slashes ***//*** and then any content on that line or nested below that line, will be omitted from computation.

Example:
```
/* this is a normal comment */
p
  font-style: italic
// this is a line that will not be shown in CSS
h1
  font-size: 3em
```

Compiled CSS:
```
/* this is a normal comment*/
p {
  font-style: italic;
}

h1 {
  font-size: 3em;
}
```

4 Syntax - Part 2

Sass provides a great feature to CSS which is the use of variables that can be defined and then reused as necessary. Variables are defined with a dollar sign ($) followed by the name of the variable. You follow the name of the variable by colon :, then an empty space, and then you will have the value of the variable. The value of the variable can be a number, color, Boolean, string, null, or list of values separated by spaces or comma. So, for example, you could have the following code in Sass:

```
$font-base: 1em
$serif: Helvetica, Arial, “Lucida Grande”, sans-serif

p
  font: $font-base $serif
```

Compiled CSS:
```
p {
  font: 1em Helvetica, Arial, “Lucida Grande”, sans-serif;
}
```

Because variables can be used anywhere inside Sass, they may sometimes need to be interpolated – for example when the variables are being used in a class name, property name, or inside a string of plain text

For example, this could happen in Sass:
```
$location: downtown
$offset: left
.#{$location}
  #{$offset}: 20px
```

Compiled CSS:
```
.downtown {
```

left: 20px;
}

You can also do calculations in Sass and this can handle most problems related to addition, subtraction, division, multiplication, and rounding. Addition is done with the plus sign and it can be done with our without units of measurement but when using units, the unit tied to the first number in the equation will be the one used for the final value. Subtraction is done with the minus sign, multiplication with asterisk but, in this case only one of the numbers may include a unit of measurement or none. Using the percentage sign, you will get the remainder of the two numbers after being divided and this operation also only allows one number with unit of measurement.

So, in Sass you could have:
width: 40px + 6
width: 40px – 6
width: 40px * 6
width: 40px % 6

Compiled CSS:
width: 46px;
width: 34px;
width: 240px;
width: 4px;

What about the division? It's trickier because the forward slash, used for division, is also used in some CSS property values. When using one unit of measurement in division, the value will reside in that unit and when using two units of measurement, the result will be unitless.

Examples:
width: 200px / 10
width: (200px / 10)
width: (200px / 10px)
$w: 200px
width: $w / 10
width: 500px – 200px / 10

Compiled CSS:
width: 200px/10;
width: 20px;
width: 20;
width: 20px;
width: 480px;

You can combine mathematical operations in Sass and it will execute those operations following the same rule we have learned in Math, even when including parentheses.

Sass includes a lot of **built-in functions** (Sass Built-in Modules), example, the **percentage()** function turns a value into a percentage. The **round()**function rounds a value to the closest whole number, the **abs()** function finds the absolute value of a given number

Example:
width: percentage(2.5)
width: abs(-2.5px)

Compiled CSS:
width: 250%;
width: 2.5px;

But you can also create as function in Sass such as:

```
@function calculate-rem($size) {
  $rem-size: $size / 16px;
  @return #{$rem-size}rem;
}
```

This function receives the variable ***$size*** as a parameter and then will calculate another variable ***$rem-size*** by dividing ***$size*** by 16px and it will then use the ***@return*** to return the numeric value of ***$rem-size*** with the string "***rem***" – this function is, in reality, receiving a number in pixels (px) to convert to rem value.

Once this function is set, you can call this function via a mixin or directly such as:

```
.myclass {
   font-size: calculate-rem(20px);
}
```

Which will give you in CSS:

```
.myclass {
   font-size: 1.25rem;
}
```

You could also have a mixin as:

```
@mixin font-size($size) {
   font-size: calculate-rem($size);
}
```

And then use that mixin (do not worry, we will talk more about mixins in the following pages) with the @include such as:

```
div {
   @include font-size(36px);
}
```

The final CSS would show:

```
div {
  font-size: 2.25rem;
}
```

Sass is great to work with colors too. One of the most popular color features is the ability to change a hexadecimal color, or variable and convert it into an RGBa value. For example:
color: rgba(#3f4bc6, .2)

OR

$varcolor: #3f4bc6
color: rgba($green, .2)

Compiled CSS:
color: rgba(63, 75,198, .2);

Remember when we talked about operations? Well, Saas can also perform operations on colors using addition, subtraction, multiplication, division and these operations are performed on the red, green and blue components of the color as we change them as intended.

color: #a3a3a3 + #aaa
color: rgba(142, 198, 63, .7) / rgba(255, 255, 255, .7)

Compiled CSS:
color: #a3ae4d;
color: rgba(0, 0, 0, .7);

Here is an article that explains how to do operations with hexadecimal numbers. Of course, you can also use websites to convert from hexadecimal to decimal and then make the calculation and convert back to hexadecimal (using that same website to convert back). Or you can use certain websites that will do the calculation in hexadecimal.

Sass can also do color alterations that you can use, for example, to invert colors, find complementary colors, mix colors, or find the grayscale value of a color with the following respective functions: **invert()**, **complement()**, **mix()**, **grayscale()**. You can do even more with HSLa colors and some of the most popular used with these colors are: **lighten()**, **darken()**, **saturate()**,**desaturate()**. There is also: **adjust-hue()**, **fade-in()**, **fade-out()**.

There are more complex color manipulations you can do with Sass.

Extends – it provides a way to share and reuse styles without having to explicitly repeat the code or use additional classes – this provides a way to keep your code very modular and “DRY” (**D**on't **R**epeat **Y**ourself). Both elements and class selectors may be used as an extend and there is a placeholder selector built just for extends. You establish an extend by using the **@extend**rule followed by the selector you want to extend

Example:

```
.attention
  border:2px solid red
  padding: 5px

.attention-error
  @extend .attention
  background-color: yellow
  color: red
```

Compiled CSS:

```
.attention,
.attention-error {
  border: 2px solid red;
  padding: 5px;
}
.attention-error {
  background-color: yellow;
  color:red;
}
```

Placeholder Selector Extend – this avoids building a bunch of unused classes just for extends. It is initialized with a percentage sign **%** and is never directly compiled into CSS, instead, it is used to attach selectors to when it is called with an extend. Note below that the .attention selector never makes the CSS file.

```
%attention
  border: 2px solid red
  padding: 5px

.attention-error
  @extend %attention
  background-color: yellow
  color: red
```

Compiled CSS:

```
.attention-error {
  border: 2px solid red;
  padding: 5px;
}
.attention-error {
  background-color: yellow;
  color: red;
}
```

See below an example of the extend used with elements:

```
h2
  color: blue
  span
    font-style: italic

.subheading
  @extend h2
```

Compiled CSS:

```
h2, .subheading {
  color: blue;
}
h2 span, .subheading span {
  font-style: italic;
}
```

Mixins – you can easily template properties and values with mixins and share those among different selectors. Mixins allows arguments to be passed and that’s the difference when comparing those to extends because extends are fixed values. You use the **@mixin** rule followed by any potential arguments and then you outline any styles below the rule. In order to call a mixin from within a selector, you use the plus sign **+** followed by the name of the mixin and any desired argument values if needed.

Note: SCSS uses **@include** rule instead of the plus sign **+**

```
@mixin navbtn($color, $hovercolor)
  color: $color
  &:hover
    color: $hovercolor

.navbtn
  +navbtn($color: blue, $hovercolor: white)
```

Compiled CSS:

```
.navbtn {
  color: blue;
}
.navbtn:hover {
  color: white;
}
```

With the same example above, we can also specify, in the template (mixin) default argument values that, of course, may be overwritten.

```
@mixin navbtn($color: blue, $hovercolor:white)
  color: $color
  &:hover
    color: $hovercolor

.navbtn
  +navbtn($hovercolor: yellow)
```

Compiled CSS:

```
.navbtn {
  color: blue;
}
.navbtn:hover {
  color: yellow;
}
```

If the argument, inside the parentheses ends with **…**, this means that we can pass in comma separated values to the mixin.

Imports – Sass has the ability to import multiple .scss or .sass files and condense them into one single file – when you condense separated CSS files into one, you avoid the numerous HTTP requests that would happen otherwise. Instead of referencing all the different CSS files in your HTML document, you only refer the single Sass file importing all of the other stylesheets.

In the Sass document (example, one called styles.sass), you would have:

```
@import “normalize”
@import “layout”, “typography”
```

This means that you are importing **normalize.sass**, **typography.sass**, **layout.sass** into **styles.sass** and later this would be compiled into **styles.css**. Then, in the HTML, you would need only to refer to styles.css

Loops and Conditionals – they are part of the possibilities offered by Sass. In these types of processes, you will be using the relational operators (**<**, **<=**, **>**, **>=**) and the comparison operators (**==**, **!=**)

The If Function – the **@if** rule tests expressions then loads the styles beneath the expression that returns true. The initial if statement may be proceeded by many **else if** statements and one **else** statement.

Example:

```
$var: great

.verygood
  @if $var == great
    color: blue
  @else if $var == good
    color: orange
  @else
    color: red
```

Compiled CSS would be:

```
.verygood {
  color: blue;
}
```

Loop – Used with the **@for** rule which will output different sets of styles based on a counter variable. There are two different formats for the loop – **to and through**. The first one would have something as @***for $i from 1 to 5*** – it would output styles up to, but not including 5. The other form @***for $i from 1 through 5*** will output styles up to and including 5.

```
@for $grid from 1 to 4
  .col-#{$grid}
    width: 30px * $grid
```

The compiled CSS would be:

```
.col-1 {
  width: 30px;
}
.col-2 {
  width: 60px;
}
.col-3 {
  width: 90px;
}
```

Each loop – Used with the **@each** rule to return styles for each item in a list that may include multiple comma separated items.

```
@each $class in js, html, css
  .#{$class}-logo
    background-image: url("/img/#{$class}.png")
```

Compiled CSS:

```
.js-logo {
  background-image: url("/img/js.png");
}
.html-logo {
  background-image: url("/img/html.png");
}
.css-logo {
  background-image: url("/img/css.png");
}
```

While Loop – Using the **@while** to repeatedly return styles until the statement becomes false. You can control the counter variable and the directive accepts many different operators.

```
$counter: 1
@while $counter <= 3
  h#{$counter}
    font-size: 2em – ($counter * .25em)
  $counter: $counter + 1
```

Compiled CSS:

```
h1 {
  font-size: 1.75em;
}
h2 {
  font-size: 1.5em;
}
h3 {
  font-size: 1.25em;
}
```

There is an interesting website that will compile online (convert) the .sass into .css. Let us use this website to drag and drop the **test.sass** file you have and then, after processing the file, you will notice the compiled CSS version showing up below the part you dropped the css file. You can try with other exercises we had during the lecture part but remember not to simply copy and paste to create the SASS file because you might get errors as the text here is rich text format, not plain text. Besides, you need to remember that SASS also requires consistent indentation.

If you have installed Sass in your computer, via Ruby, then you can copy to the Ruby folder, the **test.sass** file and then run the following command in your command line (in Windows) or Terminal (in Mac) from inside the Ruby folder

```
sass test.sass test.css
```

You will notice that **test.css** is then created in the same folder as **test.sass** and you will also notice a file called **test.css.map** that is created too. You will also note that, at the end of the test.css file you will see the following line ***/*# sourceMappingURL=test.css.map */***

This file is what maps all of the compiled CSS back to the source Sass declarations. You do not need to worry about this file.

5 Organization of Sass code

It is important to have a good organization when you code the Sass as it makes it easier for you and co-workers to understand the code and be able to derive the CSS code from there. The CSS-Tricks website has some good tips on how to style/organize the Sass code.

6 Koala

It's a GUI application, that you can download from Koala's website that can be used to help you code Sass, or SCSS.

7 PostCSS

We have all faced the question: should I use vendor prefixes in my CSS such as -moz-, -ms-, -webkit-, etc.? We would need to check the many different CSS properties we want to use at Caniuse website or use something as Autoprefixer that checks caniuse.com and adds the necessary prefixes to our CSS code. Autoprefixes is one of the many PostCSS plugins out there.

PostCSS is a tool to transform CSS with JavaScript plugins – you can pick the plugins you need or even write a custom plugin yourself. Once the plugin is invoked, it will scan through your CSS code and perform the desired transformations. Some plugins do not work with plain CSS (only with Sass-like syntax).

One of the interesting uses for PostCSS is the stylelint plugin that points out errors in your CSS code (it supports the latest CSS syntax). For example, the following CSS code:

```
a { color: #a3; }
```

would give the following output after stylelint works on it:

```
app.css
2:10 Invalid hex color
```

There are other interesting plugins such as LostGrid that uses `calc()` to create grids based on fractions that you define without the need to pass too many options.

PostCSS can basically do the same work as Sass, Less and other preprocessors but PostCSS is modular and seems then to be faster. You can pick the features you need. There are even some PostCSS plugins such as PostCSS Sass and PreCSS that are essentially a replacement of Sass which means you can write your own preprocessor powered by PostCSS (check this article – PostCSS Deep Dive: Roll your own preprocessor.

If you want to know how you can set up and use PostCSS in your code, read the article PostCSS – Transforming your CSS with JavaScript.

8 Final observations

You will not be uploading those files to the server as the idea is that you will be using those files, with the preprocessor code, to update your code hopefully using less lines of code. Then, you will be compiling (converting) these files into **.css** and these are the ones that will continue to be uploaded to the web server as the browsers **DO NOT PARSE** these preprocessors files.

There are other preprocessors in the market such as: Jade, Slim, Stylus, LESS

Sass was shown in this class as they have huge community support. But, for example, if your project will be built in Node.js, it might be better for you to work with some other preprocessor?!?

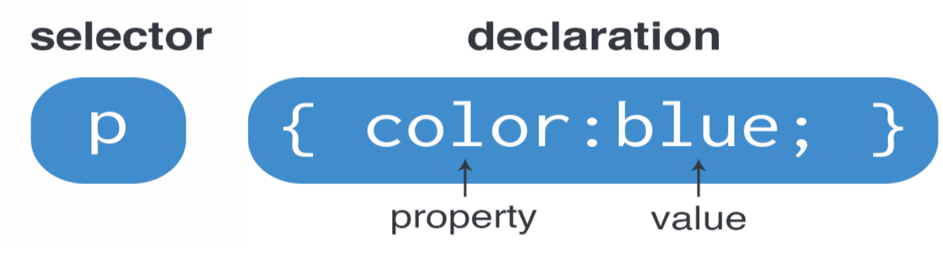
There are some other interesting points to be added here as a lot of people make confusion when talking about Sass or SCSS (known as Sassy CSS). We have studied that a **.sass** file represents a Sass code (Syntactically Awesome Stylesheets) and if you see a **.scss** file, it will be representing the SCSS. But, in reality we can consider that SCSS is half way between CSS and Sass and the difference between both is quite subtle but here are some good points:

- Sass is the name of the preprocessor
- SCSS is easier to learn than Sass (at least for most people)
- All the features are available for both syntaxes and everything that is available in SCSS is available in Sass

Some extra tips for your HTML and/or CSS

1 Using calc() in CSS

CSS has a special function – calcl() – for doing basic math. Remember that CSS is composed by the selector, then you have the CSS property, followed by : (colon) and then you have the CSS value. Here below is an image to show those terms in place:



You can only use calc() function in values, not in the CSS properties. For example:

```
.myclass {
     width: calc(100% - 20px);
     height: calc(100vh – 20px);
     padding: calc(1vw + 5px);
}
```

It can also be used for only part of a property – for example:

```
.myclass {
     margin: 15px calc(2vw + 5px);
     border-radius: 5px calc(18px / 3) 10px 20px;
}
```

NOTE: vw = viewport width – this unit is based on the width of the viewport and a value of 1vw represents 1% of the viewport width. The same applies to vh.

In the example above, the margin of the element with class = "myclass" will have 15px for margin-top and margin-bottom. The calc(2vw + 5px) will be applied for margin-left and margin-right.

You can also use calc() as part of another function that will make a part of a property. For example, if you are using the CSS gradient, you could have a code like:

```
.myclass {
     background: #1360A4 linear-gradient(to bottom, #1360A4, #1360A4 calc(50% - 15px),
#3594E8 calc(50% + 15px), #3594E8);
}
```

You **CANNOT** use calc() to compose strings, for example, the code below is **INVALID**!!!

```
.myclass::before {
     content: calc("Product " * 3);
}
```

The calc() function is used for the many different length units available in CSS – em, rem, px, pt, %, vh, vw, etc.

Numbers without units can be used in calc() function as well such as:

```
.myclass {
     line-height: calc(1.3 * 2);
}
```

You **CANNOT** use calc() in @media queries – the example below is **INVALID**!!!

```
@media screen and (min-width: calc(30rem + 5px)) {
     /* CSS properties for screen wider than 30rem */
}
```

When you use a CSS preprocessor, you can do something like this:

```
$padding: 1rem;

.myclass {
     margin-top: $padding * 3;     /* this will be processed to be 3rem */
}
```

There are some interesting points to observe in regards to the operators that you use in the calc() function. You use normal Math operators such as: +, -, *, and / but when you use the + and – you will need to have both numbers lengths. This means that if you code something as:

```
.myclass {
     padding: calc(5px + 2);
}
```

it will be **INVALID** and, in that case, **the padding will not even be processed/applied**.

When using / (division), the second number needs to be unitless. This means that if you have something as:

```
.myclass {
      padding: calc(10px / 2px);
}
```

it will be **INVALID**, and, again, **the padding will not be processed/applied**

And when using * (multiplication), make sure that one of the values is unitless (not necessarily the first or second value).

NOTE: WHITE SPACE matters!!! You need to leave a blank space between the first number, the operator, and the second number. Although this blank space is required only for + (addition) and – (subtraction), it's a good pattern to include the blank space even for * (multiplication) and / (division). But, DO NOT INCLUDE A BLANK SPACE BETWEEN THE WORD calc AND THE FIRST PARENTHESES!!!

When you use custom properties (CSS variables), you can have something like this:

```
html {
     --H: 100;
     --S: 100%;
     --L: 50%;
}
/* I have just defined variables H, S, and L to use in my html document and any children
of the html tag */

.myclass {
     color: hsl(calc(var(--H) + 20), calc(var(--S) - 25%), calc(var(--L) + 20%));
}
/* here I'm using the variables H, S, and L to calculate the font color of the element
with class = myclass */
```

Final Notes: There are some issues when using calc() and the most known issues are:

- IE 9 – 11 does not support the calc() used in box-shadow property
- IE 9 – 11, and Edge do not support width: calc() for table cells
- Firefox version 59 or below does not support calc() for color functions

2 Multi-column Layout (CSS Columns, Multicol)

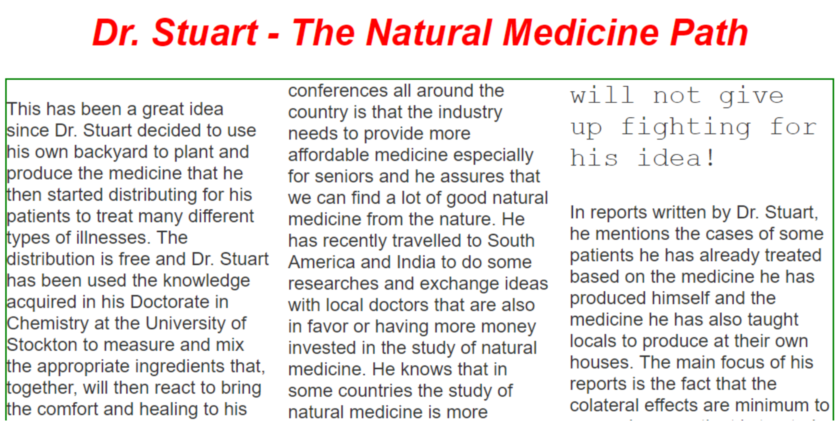
The basic idea with multicol is that you can take some content on your page and flow it into multiple columns. This is done with the help of one of the two CSS properties: **column-count** (specifies the number of columns that you want your content to break into) and **column-width** (specifies the width of the column which then leaves the browser responsible for figuring out how many columns will fit).

Everything will remain in normal flow and it does not matter which elements are inside the content that you are turning into a multicol. So, with multicol, you will keep the normal flow of the elements with the difference that they will be distributed inside one or more columns.

The browser support has grown in the past years and you can check it at Caniuse column-width and/or Caniuse column-count.

Open **multicol1.html** (that you will find among the files you downloaded for this course) in the browser and also look at the HTML and CSS that was coded.

You should see this result in the browser (the image below is showing only the upper part of the web page to save some space):



Dr. Stuart - The Natural Medicine Path

This has been a great idea since Dr. Stuart decided to use his own backyard to plant and produce the medicine that he then started distributing for his patients to treat many different types of illnesses. The distribution is free and Dr. Stuart has been used the knowledge acquired in his Doctorate in Chemistry at the University of Stockton to measure and mix the appropriate ingredients that, together, will then react to bring the comfort and healing to his

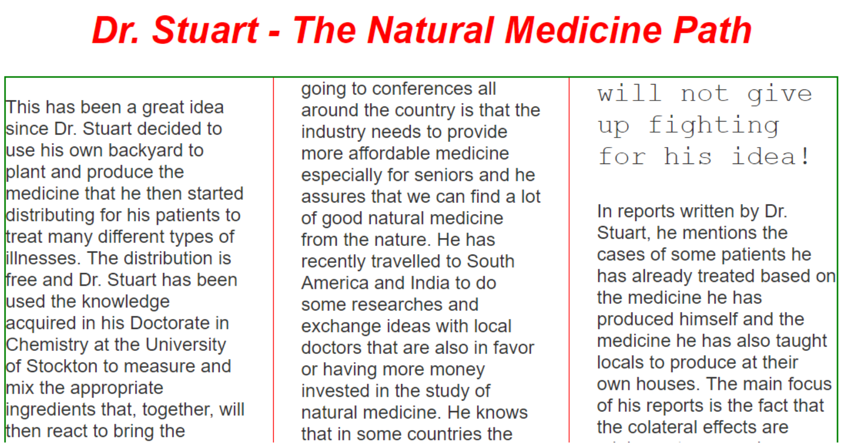
conferences all around the country is that the industry needs to provide more affordable medicine especially for seniors and he assures that we can find a lot of good natural medicine from the nature. He has recently travelled to South America and India to do some researches and exchange ideas with local doctors that are also in favor or having more money invested in the study of natural medicine. He knows that in some countries the study of natural medicine is more

will not give up fighting for his idea!

In reports written by Dr. Stuart, he mentions the cases of some patients he has already treated based on the medicine he has produced himself and the medicine he has also taught locals to produce at their own houses. The main focus of his reports is the fact that the colateral effects are minimum to

The column boxes that were created cannot be targeted which means you cannot address them with JavaScript, neither style only an individual box. All of the column (boxes) will have the same size and the only thing you can style is adding a rule between the columns (with **column-rule**) and you can also control the space between the columns (with **column-gap** that has a default value of 1em).

Open now **multicol2.html** in the browser and notice that in the CSS code we added the **column-gap** and **column-rule** properties. This is what you should see in the browser:



Dr. Stuart - The Natural Medicine Path

This has been a great idea since Dr. Stuart decided to use his own backyard to plant and produce the medicine that he then started distributing for his patients to treat many different types of illnesses. The distribution is free and Dr. Stuart has been used the knowledge acquired in his Doctorate in Chemistry at the University of Stockton to measure and mix the appropriate ingredients that, together, will then react to bring the

going to conferences all around the country is that the industry needs to provide more affordable medicine especially for seniors and he assures that we can find a lot of good natural medicine from the nature. He has recently travelled to South America and India to do some researches and exchange ideas with local doctors that are also in favor or having more money invested in the study of natural medicine. He knows that in some countries the

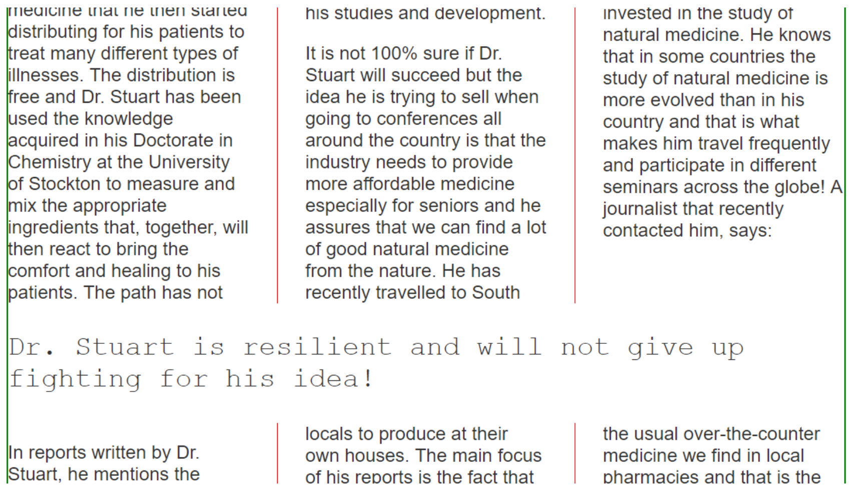
will not give up fighting for his idea!

In reports written by Dr. Stuart, he mentions the cases of some patients he has already treated based on the medicine he has produced himself and the medicine he has also taught locals to produce at their own houses. The main focus of his reports is the fact that the colateral effects are

As you can see, it's very simple to break the content of an element into columns.

Let's go further with another step. What if you wanted to have the blockquote element spanning over the column boxes that were created? You can achieve this layout by using the **column-span** CSS property to the element you want to span.

Open **multicol3.html** and notice that the blockquote has not the CSS property column-span applied to it and that's what you should see in the browser (the image below shows only the part where the content of the blockquote element is shown):



medicine that he then started distributing for his patients to treat many different types of illnesses. The distribution is free and Dr. Stuart has been used the knowledge acquired in his Doctorate in Chemistry at the University of Stockton to measure and mix the appropriate ingredients that, together, will then react to bring the comfort and healing to his patients. The path has not

his studies and development.

It is not 100% sure if Dr. Stuart will succeed but the idea he is trying to sell when going to conferences all around the country is that the industry needs to provide more affordable medicine especially for seniors and he assures that we can find a lot of good natural medicine from the nature. He has recently travelled to South

invested in the study of natural medicine. He knows that in some countries the study of natural medicine is more evolved than in his country and that is what makes him travel frequently and participate in different seminars across the globe! A journalist that recently contacted him, says:

```
Dr. Stuart is resilient and will not give up
fighting for his idea!
```

In reports written by Dr. Stuart, he mentions the

locals to produce at their own houses. The main focus of his reports is the fact that

the usual over-the-counter medicine we find in local pharmacies and that is the

It’s necessary to mention that the only possible values for column-span are none (default) and all. This means you cannot span over a certain number of columns (boxes) but you can achieve great layouts by combining multicol with other layouts such as grid or flexbox.

Now, open **multicol4.html** in the browser and notice that we separated the paragraph that starts with “A journalist that recently…”. Now, this is a separate paragraph and both the paragraph and blockquote elements were coded inside a div with the class=”block”. With the class block commented out in the CSS part, you will notice that the blockquote will be shown in a different column, separated from the paragraph that it’s related to. In order to guarantee that the blockquote will be presented in the same column (box) as the paragraph, you can use the **break-inside** CSS property with the value of **avoid**.

Go ahead and remove the comment (the /* and the */) from the .block class in the CSS part. Save and refresh the browser and now you should see the blockquote always together with the paragraph that it’s related to. The other CSS properties that can help you control the break of the columns are: **break-before** and **break-after**. You can read more details about these CSS properties at the W3Schools break-inside, W3Schools break-before, and W3Schools break-after.