Summary on SASS

Main Idea: overcome limitations of vanilla CSS
(more expressive language, leading to shorter more powerful code)

We need a translator (=computer program) to translate from SASS/SCSS to vanilla CSS (the only thing the browser can handle)

Terminlogy

SASS is the program and for syntactically awesome style sheets (comes without curly brackets and semicolons)

SCSS is short for sassy cascading style sheets.

Bringing it to live

Translator comes in different implementations for all popular programming environments.
In our case: DCI teaches JavaScript → we use NPM packages of 3rd party tools (like Sass)

NPM organizes project setup in a package.json file.

```
npm init
npm install sass –save-dev
```

adding scripts for executing sass

```
scripts {
      “translate”: “sass input:output”,
      “watch”: “sass input:output --watch”
}
```

after defining such scripts, we can run them with npm run *scriptname*

```
npm run translate
```

Create `.scss` or .sass files and translate them with the npm script

Features of SASS

Variables

Idea: central management of reused property values. Write once, use very often.

Code:

Define them: `$variableidentifier : value;`

(value can be of any type, e.g. lengths, times, colors, strings, border-styles, etc.)

Use them: `p { property: $variableidentifier; }`

Nesting

Idea: Keep code easier to read / understand

Code:

```
selector1 {
      property1: value;
      selector2 {
            property2: value;
      }
}
```

Result in:

```
selector1 { property1: value; }
selector1 selector2 { property1: value; }
```

Partials

Idea: splitting up styles to different files and re-use them

```
@use “filename”
```

(you can skip extensions .scss .sass, call variables defined in the other file)

Mixins

Idea: stay DRY (don’t repeat yourself), reuse of property/value definitions.

Code: define:

```
@mixin identifier {
      property1: value;
      property2: value;
}
```

Code: “call”/use it:

```
selector {
      @include identifier;
}
```

Can optionally (very powerful) have parameters:

Code : define:

```
@mixin identifier($parameter1, $parameter2 : defaultvalue) {
      property1: $parameter1;
      property2: $parameter2;
}
```

Code: to use:

```
selector {
      @include identifier(50px, dotted);
}
```

Functions

Idea: make calculations or similar, and return a value to be used for a property, can also be controlled with parameters/arguments
Code: define:

```
@function double ( $parameter : defaultvalue) {
      @return $parameter * 2;
}
```

code: to call:

```
selector {
      property1: double(15px);
}
```

result (after translation):

```
selector {
      property1: 30px;
}
```

Variable Maps:

A Variable Map can hold more than one value within the same variable identifier.
Code to define:

```
$mymap : (
  “aprop” : value,
  “secondthing”: value
)
```

To call those “sub-values” you need to call a function map-get($mapvariable, “propertyname”)
Code to use:

```
selector {
      property: map-get($mymap, “aprop”);
}
```