# CSS-Präprozessoren: Saas

Kristina Radeva

# Introduction

- CSS extensions
  - Nested rules
  - Nested properties
  - Parent selectors
  - Placeholder selectors
- Comments
- Variables
- Interpolation
- @ rules
- @ control directives and expressions
- @ mixin directives
- @ function directives

# CSS extensions

## Nested rules

CSS rules can be nested within one another. Inner rules apply only within outer rule's selector. This extension allows to reduce repetitions or parent selectors and makes parent - child relations more visible.

## Nested properties

Allows nesting of properties using the same namespace, like size, weight and family in font.

# CSS extensions

## Parent selectors

& character makes it possible to execatly specify where the parent selector should be inserted.

## Placeholder selectors

Instead of # or . prefix in case of ids and classes they use %. By default rulesets having this kind of selector won't be rendered to css output file. They can however be used with @extend directive which allows building styles libraries and practical including of parts of library only when it is needed.

# Comments

- In Saas multi line comments /* */
  and single line comments // are allowed
- Multi line comments are directly rendered to css, while single line comments won't appear in output css file.

# Variables

- Variables are defined similarly to properties but they have $ sign before the name.
- – and _ can be used interchangeably in variables names.
- Example: $body_width: 1000px;
- By default scope of each variable is the level of nested selector where it is defined. With !global flag they are available in whole document.

# Variables - data types

- Numbers ( also as px, pt, em etc. )
- Strings ( unquoted, quoted using ' ' , quoted using " " )
- Colors
- Nulls
- Booleans
- Key - value pairs ( key1: value1, key2: value2 )
- Lists , maps ( lists of key - value pairs )
- Functions references

# Interpolation

- Allows using variables not only in property values but also in selectors and property names using #{$var-name}.
- When it is used on variables in property value, variable won't be affected by any functions, and functions be treated as plain CSS, not evaluated by Saas.

# @ rules

## @import

- For including other .sass/.scss files. All included files will be merged together to one css file.
- All variables and mixins from included files can be used in main file.
- If the file extension is .css, file name begins with http:// or there are media queries in @import, statement will be rendered to css file instead of being interpreted by Saas.

## @media

- Used for media specified styles.
- In Saas @media directives can be nested.

# @ rules

## @extend

- Allows inheritance of styles.
- Selector A inherits styles of another selector B with B { @extend A}.
- A single selector can extend more other selectors.
- Multi level extending ( chaining ) is also possible.

## @debug, @warn

Used for printing value of expressions to standard error output stream.

## @atroot

Moves one or more rules to the root of document instead of being nested beneath their parent selectors.

# @ control directives and expressions

## @if @else if @else

```
@if 1 + 1 == 2 { border: 1px solid; }
$width = 100px;
@if $width ≤ 100 { height: 200px;}
@else { height: 500px;}
```

## @for

Used for repeating nested styles while iterating over a variable.

```
@for $i from 0 to 10 {
.box–#{$i} { width: 1px * $i } }
```

## @each

Used for iterating through values in a list or map.

```
@each $var in list / map { ... }
```

## @while

Used for repeating nested styles until given statement evaluates to false.

# @ mixin directives

Used for defining styles which can be reused throughout the stylesheet without needing to repeat them. They can take arguments which allows to produce a wide variety of styles with very few mixins.

## Example

```
@mixin customized-border($color, $width, $style: dashed) {
border: { color: $color; width: $width; style: $style; }
}
p { @include customized-border(blue, 100px, solid); }
```

# @ function directives

They allow to define own functions that can be used in any place in document. They can access global variables, have multiple statements including control flow. They can take arguments and must contain a return statement.

## Example

```
$width: 100px !global;
@function scale-width( $n ) {
@if $n ≤ 100px { @return $width; }
@else { @return $n * $width; }
}
```