CSS preprocesor

www.michalmarek.sk

Obsah

SASS - CSS Preprocesor (0)

Ako písať prehľadnejšie a kvalitnejšie CSS.

Čo je preprocesor

Vo všeobecnosti je preprocesor program, ktorého výstup bude následne spracovaný ďalším programom. V tomto seriály sa zameriame práve na CSS preprocesor, ktorý nám značne uľahčí dodržiavať takzvaný **DRY(Don't repeat yourself)**, a tým zvýši prehľadnosť a udržovateľnosť nášho kódu, ktorá je pri vývoji veľmi dôležitá. V klasickom surovom CSS nemôžme využívať také vlastnosti ako premenné, vnáranie kódu, mixiny(znovu použiteľné bloky kódu), podmienky, cykly a pod. Vďaka SASS a podobným CSS preprocesorom sú nám tieto prvky dostupné.

Aké CSS preprocesory poznáme

Medzi najznámejšie CSS preprocesory patrí SASS a LESS. Je samozrejme mnoho ďalších ako Stylus, Switch CSS, CSS Cacheer ale tento seriál je výhradne zameraný na najznámejší z nich, a to SASS. Táto "nultá" kapitola je len takým jemným úvodom do SASS a preprocesorov ako takých. V následujúcich častiach budeme prechádzať podrobne všetkými prvkami a vlastnosťami SASS.

SASS - v skratke

Sass znamená **S**yntactically **A**wesome **S**tyle**s**heets alebo syntakticky úžasné štýly a je napísaný v jazyku Ruby tvorcami Hampton Catlin, Nathan Weizenbaum a Chris Eppstein. Podporuje dve syntaxi. Syntax s koncovkou súboru **.sass** a **.scss**. Syntax .sass je podobná jazyku Ruby, kde sa nepoužívajú bodkočiarky, zátovrky a pod. Syntax .scss alebo Sassy CSS je podobná klasickému CSS.

Syntax .sass

```
a
  color: red
  text-decoration: none
```

Syntax .scss

```
a {
   color: red;
   text-decoration: none;
}
```

Základné vlastnosti SASS:

1. Premenné
2. Vnáranie kódu
3. Čiastočné súbory a import
4. Mixiny (znovu použiteľné bloky kódu)
5. Dedičnosť selektorov
6. Operátory, Funkcie, Podmienky, Cykly atď.

Všetky tieto základné vlastnosti SASS si prejdeme po jednotlivých kapitolách a prvé na čo sa zameriame bude využitie premenných.

Všetky podrobné informácie ohľadom SASS nájdete na oficiálnej stránke.

Inštalujeme SASS

Než sa pustíme do jednotlivých častí SASS, pozrieme sa na to ako ho nainštalovať.

Windows

Na Windows potrebujeme ako prvé nainštalovať Ruby. Najjednoduchší spôsob je použiť nástroj Ruby Installer. Po inštalácií tohto nástroja Vám bude dostupná aj Ruby konzola, vďaka ktorej budete mocť používať Ruby knižnice.

Mac

Pre užívateľov Mac-u mám dobrú správu, Ruby je predinštalované a môžme sa pustiť rovno do inštalácie.

Samotná inštalácia

Do konzoly zadajte príkaz **gem install sass**. V prípade, že Vám nebude tento príkaz fungovať, skúste **sudo gem install sass**. A je to :).

SASS - Premenné (1)

Uľahčujeme si prácu vďaka premenným

K čomu su určené

Premenné sú jednou z hlavných výhod SASS a podobných preprocesorov. Umožňujú nám jednoducho si nazvať určitú CSS vlastnosť, ktorú môžme opakovane používať odkazovaním sa na jej názov, čo je efektívnejšie ako písať tú istú vlastnosť stále dookola. Túto výhodu pocítime hlavné v prípade neskorších úprav a zmien v kóde.

Vytvárame si našu prvú premennú

V SASS sa používa na vytvorenie premennej znak "$" a za ním následuje zvolený názov premennej. Premenná sa deklaruje podobne ako vlastnosť v CSS viď príklad 1.1.

Príkad 1.1

.scss

```
$base-color: #222;

body {
   color: $base-color;
}
```

.css

```
body {
   color: #222;
}
```

Vo výstupe CSS vidno, že SASS jednoducho nahradí premennú za jej hodnotu. Akákoľvek hodnota, ktorá môže byť použitá pre CSS vlastnosť, môže byť použitá ako hodnota premennej.

Ak chcete zkompilovať .scss alebo .sass súbor do .css, zadajte do konzoly príkaz **sass style.scss style.css**, kde "style.scss" je názov SASS súboru a "style.css" je názov súboru, do

ktorého Vám bude vygenerované surové CSS. Tiež môžte použiť príkaz **sass --watch style.scss:style.css**, ktorý Vám bude ihneď všetky zmeny v SASS premietať do CSS.

Názov premennej

SASS povoľuje pri názve premennej používať rovnaké znaky, aké sa môžu používať pri selektoroch a vlastnostiach v CSS. Taktiež znaky ako "-" a "_". Ja osobne preferujem pomlčky ako v predošlom príklade, ale taktiež je možné pomenovať premennú ako "$base_color". Či už použijete prvú alebo druhú variantu, SASS vyhodnotí oba rovnako, pretože "$base-color" a "$base_color" odkazuje na rovnakú premennú.

Odkazujeme sa na premenné

Ako som spomínal, premenná je jednoducho nahradená za hodnotu, ktorú ma definovanú, a preto môže byť umiestnená kdekoľvek ako ľubovoľna CSS vlastnosť ako je uvedené v príklade 1.2. Taktiež môže vlastnosť alebo hodnota premennej odkazovať na inú premennú ako v príklade 1.3.

Príklad 1.2

.scss

```
$contrast-color: #42ecaa;

body {
   border-top: 3px $contrast-color solid;
}
```

Príklad 1.3

.scss

```
$bg-color: #ddd;
$base-border: 1px solid $bg-color;

h1 {
   border-top: $base-border;
}
```

Záverom

Ako ste si mohli všimnúť, pracovať s premennými je veľmi jedoduché a užitočné. Premenné môžu obsahovať ešte príznak alebo označenie **!default**, ktoré si ale vysvetlíme v spojení s čiastočnými súbormi a importom súborov. V následujúcej časti sa zameriame na vnáranie kódu, ktoré nás vrámci efektivity opät posunie ďalej ;).

SASS - Vnáranie kódu (2)

Opäť bodujeme v rámci efektívnosti a prehľadnosti kódu

Poďme na to

Ďalšou “neefektívnou” časťou CSS je opakované písanie rovnakých selektorov. SASS má aj na toto účinnú zbraň, vnáranie kódu. Často sa stane, že potrebujeme určiť štýl výhradne pre element v nejakom rodičovskom elemente. Ale čo v prípade, ak máme týchto elementov oveľa viac? Pozrime sa na príklad 2.1.

Príklad 2.1

.css

```
#header .content h1 { color: #222; }
#header .content a { color: #666; }
#header .menu { border: 1px solid #d2d2d2; }
```

V tomto príklade môžme vidieť, že sa nám často opakuje selektor “#header” a “.content”. Ak budeme chcieť neskôr upraviť kód a zmeniť selektor “#header” na “header”, budeme musieť upraviť tento selektor na všetkých miestach. Vďaka SASS a vlastnosti vnárania kódu to bude oveľa jednoduchšie (Príklad 2.2).

Príklad 2.2

.scss

```
#header {
   .content {
      h1 { color: #222; }
      a { color: #666; }
   }

   .menu {
      border: 1px solid #d2d2d2;
   }
}
```

Ako vidíme, stačí upraviť jeden selektor a hotovo :). Tento kód určite prída aj na prehľadnosti, je ale možné, že ak ste naučení na starší spôsob, môže to chvíľku trvať kým si zvyknete. Pravidlo vnárania kódu nie je obmedzené iba na vnárané selektory ale selektor “#header” môže obsahovať aj obyčajné CSS vlastnosti (Príklad 2.3).

Príklad 2.3

.scss

```
#header {
   background-color: #ccc;

   .content {
      h1 { color: #222; }
      // ...
   }
}
```

Troška odbočím. V tomto príklade som použil tzv. **Tichý komentár** “// ...”. Tichý preto, lebo nám slúži hlavne na organizáciu a prehľadnosť kódu SASS. Ak si neželáme, aby boli tieto komentáre súčasťou výstupu CSS, použijeme práve tento zápis komentára. V inom prípade môžme použiť bežný CSS komentár “/* … */”.

Odkazujeme sa na rodičovský selektor

Keď SASS kompiluje vnorený kód, tak pred vnorené selektory(.content, .menu) jednoducho pridá rodičovsky selektor s medzerou(#header .content, #header .menu). Ak potrebujeme naštýlovať pseudo element alebo pseudo triedu, ako napríklad “:hover”, nebude nám to fungovať (Príklad 2.4).

Príklad 2.4

.scss

```
a {
   color: #adadad;
   :hover {
      color: #666;
   }
```

```
}
```

.css

```
a { color: #adadad; }
a :hover { color: #666; }
```

SASS jednoducho berie tento zápis resp. ":hover" ako potomka a výstup bude "a :hover". Preto použijeme špeciálny rodičovský selektor **"&"** (Príklad 2.5).

Príklad 2.5

.scss

```
a {
   color: #adadad;
   &:hover {
      color: #666;
   }
}
```

.css

```
a { color: #adadad; }
a:hover { color: #666; }
```

Tento **"&"** SASS selektor je jednoducho nahradený za rodičovský selektor "a".

Vnáranie skupín selektorov

Ak potrebujeme definovať pre určitú skupinu selektorov rovnaké vlastnosti v rámci nejakého rodičovského selektora, jednoducho túto skupinu selektorov vnoríme do rodičovského selektora (Príklad 2.6).

Príklad 2.6

.scss

```
.content {
   h1,
   h2,
   h3 {
      color: #222;
```

```
    }
}
```

.css

```
.content h1,
.content h2,
.content h3 {
    color: #222;
}
```

Vnárame kombinátory (>, +, ~)

Pri kombinátoroch je to s vnáraním rovnaké (Príklad 2.7).

Príklad 2.7

.scss

```
#header {
    > a {
        color: #222;
    }
}
```

.css

```
#header > a { color: #222; }
```

Vnárame vlastnosti

CSS selektory nie sú jediné na čo môžme využiť vnáranie. Taktiež môžme vnárať vlastnosti. Túto možnosť však nebudeme využívať tak často ako pri selektoroch (Príklad 2.8).

Príklad 2.8

.scss

```
#header {
    border: {
        style: solid;
```

```
        width: 1px;
        color: #ccc;
    }
}
```

.css

```
#header {
    border-style: solid;
    border-width: 1px;
    border-color: #ccc;
}
```

Musím ale podotknúť, že je určite jednoduchšie zapísať tento príklad ako "border: solid 1px #ccc;", ale sú prípady, kedy je to užitočné. To Vám ale prezradím troška neskôr :).

Záverom

Ako sme sa mohli opäť presvedčiť, tak nám SASS ponúka ďalšiu skvelú vlastnosť, ktorou je vnáranie kódu. Asi nám neuľahčí toľko písania ale rozhodne zvýši prehľadnosť a udržovateľnosť nášho kódu. V ďalšej časti sa môžme tešit na čiastočne súbory a import, ktoré nám umožnia si rozdeliť našu aplikáciu do logických celkov a udržiavať si v nej poriadok :).

SASS - Čiastočné súbory a import (3)

Rozdeľujeme CSS do menších a lepšie udržovateľných súborov

Direktíva @import

Ak potrebujeme na stránku vložiť viac CSS súborov, môžme okrem "link" tagu v hlavičke použiť CSS direktívu **@import**. Táto direktíva umožňuje zahrnúť všetky štýly jedného (importovaného) CSS súboru do súboru iného. V prípade obyčajného CSS musí prehliadač urobiť zvlášť požiadavku pre každý zahrnutý CSS súbor, čo spôsobuje pomalšie načítavanie stránky.

SASS má rovnakú @import direktívu, ale pri kompilácií zjednotí všetky CSS súbory do jedného súboru. Taktiež nevyžaduje špecifikovať celý názov súboru. Môžte vynechať príponu súboru ".sass" alebo ".scss". To je užitočná vec v prípade, ak importujete súbory niekoho iného bezohľadu na to, akú SASS syntax používa. My budeme používať túto direktívu hlavne pre rozdelenie CSS do menších a lepšie udržovateľných súborov. Táto možnosť rozdeľovania súborov je praktická hlavne pri väčších projektoch.

Čiastočné súbory

Keď rozdeľujeme SASS štýly do viacerých súborov pomocou direktívy @import, tak nám vytvorí pre každý import samostatný CSS súbor. Môžme importovať aj také štýly, pre ktoré nepotrebujeme vytvoriť samostatný CSS súbor, preto využijeme čiastočné súbory. SASS má pre čiastočné súbory pravidlo, že každý **názov súboru musí začínať znakom "_"**. Pri samotnom importe ale tento znak uvádzať nemusíte, čiže namiesto "@import 'base/_variables.scss';" stačí napísať "@import 'base/variables';".

Počiatočne hodnoty premenných

Teraz sa vratím trošku späť ku premenným, kde som spomenul príznak **!default**. V prípade ak deklarujete premennú viackrát, tak posledná deklarovaná hodnota premennej je jej konečná hodnota (Príklad 3.1).

Príklad 3.1

.scss

```
$base-color: #222;
$base-color: #333;

body { color: $base-color; }
```

.css

```
body { color: #333; }
```

CSS výstup je taký, ako sme očakávali, ale nie vždy budeme vyžadovať takéto správanie. Príznak !default som nechal až na túto časť, pretože uzko súvisí s importovaním súborov. Tento príznak v jednoduchosti znamená, že **ak je premenná už deklarovaná, tak používaj túto hodnotu, inak používaj počiatočnú hodnotu, ktorá je označená príznakom !default**. Poďme na príkad, ktorý nám to ozrejmí viac (Príklad 3.2).

Príklad 3.2

_variables.scss (importovaný súbor)

```
$base-color: #222 !default;
$base-font-family: Arial, sans-serif;
```

base.scss

```
$base-color: #666;
$base-font-family: Verdana, sans-serif;

@import “variables”;

body {
   color: $base-color;
   font-family: $base-font-family;
}
```

.css

```
body {
   color: #666;
   font-family: Arial, sans-serif;
}
```

Na tomto príklade môžme vidieť rozdiel medzi použitím samotnej premennej a premennej s príznakom !default. Ak definujeme hodnotu premennej pred importovaním súboru, tak "$base-color: #222 !default;"**bude ignorovaná práve kvôli príznaku !default**.

Vnárané importy

Na rozdiel od bežného CSS môžte v SASS použiť @import aj v rámci CSS pravidla (Príklad 3.3).

Príklad 3.3

_title.scss (importovaný súbor)

```
h1 {
  background: #222;
  color: #adadad;
}
```

.scss

```
.dark {
  @import “title”;
}
```

.css

```
.dark {
  h1 {
    background: #222;
    color: #adadad;
  }
}
```

Ak by sme definovali nejaké premenné alebo mixiny v importovanom súbore, tak by boli dostupné iba v rámci tohto CSS pravidla.

Záverom

SASS podporuje aj bežné CSS importy, ako je napríklad importovaný súbor s koncovkou ".css", názov importovaného súboru ako URL adresa alebo pomocou url(). Importy a čiastočné súbory sú z hľadiska udržovateľnosťi a organizácie nášho SASS kódu veľmi dôležité, najmä pri väčších

projektoch. V následujúcej časti sa pozrieme na ďalšiu zaujímavú vlastnosť SASS, a to mixiny alebo inými slovami znovu použiteľné bloky kódu.

SASS - Mixiny (4)

Znovupoužiteľné bloky kódu

Začíname

Ako sme sa naučili v časti o premenných SASS - Premenné (1), slúžia nám na uloženie jednoduchej hodnoty, ktorú opakovane používame v našom kóde na rôznych miestach. Častokrát sa ale stane, že ukladanie takýchto hodnôt nám nebude stačiť a budeme potrebovať opakovane používať väčšie časti kódu. V SASS môžme tento problém vyriešiť pomocou mixinov. Mixiny sa definujú použitím direktívy **@mixin** a následne jeho názvom (Príklad 4.1).

Príklad 4.1

.scss

```
@mixin border-radius {
   -webkit-border-radius: 10px;
   -moz-border-radius: 10px;
   -ms-border-radius: 10px;
   -o-border-radius: 10px;
   border-radius: 10px;
}
```

Mixin môže byť následne použitý kdekoľvek v našich štýloch pomocou direktívy **@include**. Táto direktíva jednoducho vezme všetky štýly v mixine, a vloží ich tam, kde sme definovali @include direktívu (Príklad 4.2).

Príklad 4.2

.scss

```
.container {
   background-color: #222;
   border: 3px solid #000;
   @include border-radius;
}
```

.css

```
.container {
   background-color: #222;
   border: 3px solid #000;
   -webkit-border-radius: 10px;
   -moz-border-radius: 10px;
   -ms-border-radius: 10px;
   -o-border-radius: 10px;
   border-radius: 10px;
}
```

Na tomto príklade môžme vidieť, ako si uľahčíme prácu vďaka mixinom. Už nemusíte pracne a dookola písať definíciu zaoblených rohov pre všetky prehliadače, stačí iba raz. Opäť sa držíme hesla **DRY(Don't repeat yourself)** :).

Používame CSS pravidla v mixinoch

Okrem jednoduchých vlastností môžme v mixinoch použiť aj CSS pravidla so selektormi a ich vlastnosťami. Ak mixin obsahuje CSS pravidla, ktoré sú následne vložené pomocou @include v nejakom rodičovskom pravidle, tak CSS pravidla budú vnorené v rámci tohto rodiča (Príklad 4.3).

Príklad 4.3

.scss

```
@mixin no-style {
   list-style: none;

   li {
      list-style-type: none;
   }
}

ul {
   @include no-style;
}
```

.css

```
ul {
   list-style: none;
```

```
}

ul li {
   list-style-type: none;
}
```

V CSS pravidle môžme v rámci mixinu tiež použiť SASS rodičovský selektor **“&”** (Príklad 4.4).

Príklad 4.4

.scss

```
@mixin colored-link {
   color: red;
   &:hover { color: orange; }
   &:visited { color: blue; }
}

a {
   @include colored-link;
}
```

.css

```
a          { color: red; }
a:hover  { color: orange; }
a:visited { color: blue; }
```

Práca s CSS pravidlami v mixinoch a s rodičovským selektorom je rovnaká ako sme videli v časti vnárania kódu SASS - Vnáranie kódu (2).

Predávame do mixinov argumenty

Aby mohli byť naše mixiny viac flexibilné, SASS nám umožňuje do nich predávať argumenty. Je to podobné, ako by sme písali nejakú funkciu v Javascripte alebo inom podobnom jazyku (Príklad 4.5).

Príklad 4.5

.scss

```
@mixin border-radius($radius) {
```

```
    -webkit-border-radius: $radius;
    -moz-border-radius: $radius;
    -ms-border-radius: $radius;
    -o-border-radius: $radius;
    border-radius: $radius;
}

.container {
    background-color: #222;
    @include border-radius(20px);
}
```

.css

```
.container {
    background-color: #222;
    -webkit-border-radius: 20px;
    -moz-border-radius: 20px;
    -ms-border-radius: 20px;
    -o-border-radius: 20px;
    border-radius: 20px;
}
```

Takto jednoducho si môžme spraviť vlastný mixin, ktorý nám bude generovať ľubovoľnú veľkosť zaoblených rohov. Takýto mixin a mnoho ďalších Vám ponúka aj užitočný framework napísany v SASS pod názvom Compass. Okrem mixinov tiež ponúka kopec iných užitočných nástrojov, ako je práca s CSS spritmi, URL adresami, matematickými funkciami atď.

Keď vkladáte mixin a predávate mu viacej argumentov, môže byť niekedy zmätok v tom, ktorý argument čo znamená a v akom poradí majú následovať. Preto SASS umožňuje pomenovať argumenty pomocou zápisu “$meno: hodnota” (Príklad 4.6).

Príklad 4.6

.scss

```
@mixin colored-link($base, $active, $visited) {
    &, &:hover { color: $base; }
    &.active { color: $active; }
    &:visited { color: $visited; }
}
```

```
a {
    @include colored-link($base: blue, $active: red, $visited: gray);
}
```

Majme ale na pamäti, že poradie argumentov môže byť ľubovoľné iba vtedy, ak sú v @include prítomné všetky argumenty, ktoré sú definované v mixine.

Počiatočne hodnoty argumentov

Argumenty môžu mať tiež svoju počiatočnú hodnotu, ktorá je použitá v prípade, ak pri vkladaní (@include) mixinu daný argument nepredáme. Počiatočné argumenty majú zápis ako pomenovaný argument, a to "$meno: pociatocna-hodnota". Počiatočná hodnota môže byť akákoľvek CSS hodnota vrátane iných argumentov (Príklad 4.7).

Príklad 4.7

.scss

```
@mixin colored-link($base, $active: $base, $visited: $base) {
   &, &:hover { color: $base; }
   &.active { color: $active; }
   &:visited { color: $visited; }
}

a {
   @include colored-link(#222, #666);
}
```

.css

```
a          { color: #222; }
a:hover { color: #222; }
a.active { color: #666; }
a:visited { color: #222; }
```

Záverom

Mixiny sú veľmi zaujímavou vlastnosťou SASS a umožňujú nám opakované používanie štýlov. Nie je to ale jediná vlastnosť SASS, ktorá nám umožňuje opakované využívanie. Môže sa stať, že budete potrebovať znovu použiť nejaké triedy. SASS má ešte inú vlastnosť pre opakované použite, a to selektorovú dedičnosť, ktorú si predstavíme v následujúcej časti.

SASS - Selektorová dedičnosť (5)

Znovu použiteľnosť kódu pomocou dedenia štýlov.

Čo je selektorová dedičnosť

Posledná doležitá metoda pre opätovné využitie kódu sa nazýva selektorová dedičnosť. Táto metóda nám umožňuje dediť štýly z jedného selektora do druhého. Definujeme ju pomocou direktívy **@extend**a následne názvom selektora, z ktorého chceme dediť (Príklad 5.1).

Príklad 5.1

.scss

```
.notice {
   background: #666;
   border: 1px solid #222;
   font-weight: normal;
}

.notice-big {
   @extend .notice; border-width: 3px;
}
```

.css

```
.notice,
.notice-big {
   background: #666;
   border: 1px solid #222;
   font-weight: normal;
}

.notice-big {
   border-width: 3px;
}
```

Do triedy “.notice-big” sme podedili všetky štýly z triedy “.notice”, to znamená, že HTML element s atribútom “class=’notice-big’” bude mať štýly ako keby mal atribút “class=’notice notice-big’”.

Táto dedičnosť sa ale neaplikuje len na samotný “.notice” selektor ale aj na iný selektor v spojení s “.notice” (Príklad 5.2).

Príklad 5.2

.scss

```
.notice {
  background: #666;
  border: 1px solid #222;
  font-weight: normal;
}

.notice a {
  text-decoration: none;
}

p.notice {
  font-weight: bold;
}

.notice-big {
  @extend .notice; border-width: 3px;
}
```

.css

```
.notice,
.notice-big {
  background: #666;
  border: 1px solid #222;
  font-weight: normal;
}

.notice a,
.notice-big a {
  text-decoration: none;
}

p.notice,
p.notice-big {
  font-weight: bold;
}
```

```
.notice-big {
  border-width: 3px;
}
```

Ako dedičnosť funguje

Dedičnosť nie je jednoduché vloženie rovnakých štylov, ako to je pri premenných a mixinoch. Základnou ideou pri direktíve @extend je, že ak selektor ".big-notice" rozširuje selektor ".notice", tak všade v štýloch kde sa objaví trieda ".notice", bude vložená trieda ".big-notice", ako sme mohli vidieť v CSS výstupe príkladov 5.1 a 5.2. V jednoduchosti povedané, vytvorí sa multiselektor.

Rozšírena dedičnosť

Každé CSS pravidlo môže používať direktívu @extend a každé pravidlo môže byť rozšírené. Vo väčšine prípadov budete využívať dedičnosť jednej triedy, ale môžte rozšíriť aj samotný HTML element (Príklad 5.3).

Príklad 5.3

.scss

```
a { text-decoration: underline; }

.disabled {
  color: #666;
  @extend a;
}
```

.css

```
a, .disabled {
  text-decoration: underline;
}

.disabled {
  color: #666;
}
```

V prípade, že direktíva @extend rozširuje komplexnejší selektor, budú sa štýly aplikovať len na element, ktorý zodpovedá danému selektoru. Ak máme napríklad triedu ".big-error", ktorá

rozširuje(@extend) selektor ".warning.error", bude dediť štýly iba pre ".warning.error" alebo "p.warning.error", ale nie pre samotné ".warning" a ".error". Ak je selektor "#content .error", ktorý rozširuje ".big-error" tak iba element ".error" v rámci elementu "#content" bude dediť štýly triedy ".big-error". Na element ".error" mimo "#content" nebudú tieto štýly aplikované.

Direktíva zástupný symbol - placeholder (%)

Často sa Vám stane, že budete dediť štýly z nejakej triedy ale túto triedu samotnú v CSS nebudete využívať. Na to, aby sa nám táto trieda zbytočne nevytvárala vo výslednom CSS, použijeme placeholder **%** (Príklad 5.4).

Príklad 5.4

.scss

```
%notice {
  background: #666;
  border: 1px solid #222;
  font-weight: normal;
}

.notice-big {
  @extend %notice;
  border-width: 3px;
}
```

.css

```
.notice-big {
  background: #666;
  border: 1px solid #222;
  font-weight: normal;
  border-width: 3px;
}
```

Kedy použiť dedičnosť a kedy mixiny

V niektorých ohľadoch sú mixiny podobné ako CSS triedy. Obe metody totiž umožňujú pomenovať bloky štýlov, a niekedy je zmätok v tom či použiť mixiny alebo triedy. Najdoležitejší

rozdiel medzi mixinom a triedou je v tom, že triedy sú určené pre použitie v HTML, zatiaľ čo mixiny pre použitie v rámci CSS.

Mixiny by mali byť “prezentačné”, čiže určujú ako má vyzerať pravidlo v CSS. Triedy by mali byť “sémantické”, čiže označujú význam elementu. Ako sme mali v príklade 5.2, “.notice” je sémantický názov triedy, čiže určuje význam elementu, na rozdiel od mixinu, ktorý hovorí len o vizuálnom štýle (“border-radius”), ako bolo v časti o mixinoch.

Ešte jeden rozdiel nastáva v tom, že vďaka **@extend sa vytvára multiselektor** ako v príklade 5.2(“.notice a, .notice-big a”), ale **@include by vytvoril samostatné selektory** s rovnakou vľastnosťou, čo už odbočuje od našej filozofie DRY pri sémantických prvkoch.

Záverom

Dedičnosť nám umožňuje definovať vzťah medzi sémantickými triedami a používať tento vzťah na udržiavanie čistého, sémantického a dobre udržovateľného CSS. Dôležité je si uvedomiť rozdiel medzi mixinom a selektorovou dedičnosťou. Samozrejme môžte dedičnosť a mixiny kombinovať. V následujúcej a zrejme poslednej časti sa pozrieme na operátory, funkcie, podmienky, cykly atď.

SASS - Operátory, funkcie, cykly, … (6)

Objavujeme posledné vlastnosti SASS.

Začíname

Do tejto chvíle sme sa dozvedeli o mnohých výhodách SASS ako sú premenné, vnáranie kódu, mixiny, dedičnosť a pod. Tieto vlastnosti sú nám užitočné na to, aby sme čo najviac eliminovali opakujúce sa štýly a vytvárali tak čo najprehľadnejší a dobre udržovateľný kód. Mixiny a premenné ale môžu vyjadrovať iba taký štýl a takú hodnotu akú im definujeme. Môže sa stať, že budeme potrebovať mixiny alebo premenné mierne odlišné v rôznych situáciach. V tomto prípade nám pomôžu skriptovacie vlastnosti SASS.

Výrazy

Sú najbežnejším spôsobom ako manipulovať s CSS hodnotami v SASS. Výraz je čokoľvek, čo sa nachádza v definícií vlastnosti. V bežnom CSS sú to hodnoty ako “underline”, “20px” alebo viacero hodnôt ako “1px solid #666”. V SASS môžme použiť okrem premenných aj matematické operátory +, -, *, / a % (Príklad 6.1).

Príklad 6.1

.scss

```
$number-one: 10;
$number-two: 20px;

body {
   padding-top: $number-one + $number-two;
}
```

.css

```
body {
   padding-top: 30px;
}
```

Dátové typy

Každá hodnota v CSS vlastnosti alebo v SASS premennej má nejaký typ. V závislosti na týchto typoch môžu fungovať rôznymi spôsobmi. Hodnoty ako "#fff" alebo "blue" reprezentujú farbu textu, farbu pozadia atď. Hodnoty typu "20px" reprezentujú číslo a sú používané pre "width", "padding" a pod. Ešte máme hodnoty typu "none", "auto", "middle", ktoré reprezentujú mnoho ďalších vecí. SASS všetkym typom rozumie a umožňuje s nimi manipulovať. Spôsobom akým s nimi manipuluje, je závislý od konkrétneho dátového typu.

Dátový typ - Reťazec

V SASS rozlišujeme dva typy reťazcov. Reťazec v úvodzovkách (“Open Sans”) a reťazec bez úvodzoviek (underline, bold, ...). Hlavný rozdiel medzi týmyto typmi reťazcov je taký, že reťazec v úvodzovkách môže obsahovať akýkoľvek znak okrem znaku (“) a reťazec bez úvodzoviek nemôže začínať číslami alebo špeciálnymi znakmi, a nesmie taktiež obsahovať znaky ako “*” a “&”. S reťazcami sa využíva najmä operátor "+". Pri spájaní reťazcov sa vždy vytvorí nový reťazec. Jeho typ závisí podľa typu práve spájaných reťazcov (Príklad 6.2).

Príklad 6.2

.scss

```
// string + string => string
// “string” + “string” => “string”
// string + “string” => string
// “string” + string => “string”
// string + 1 => string1
// “string” + 1 => “string1"
```

Iné operácie nie sú podporované a ak sa použije napríklad operátor "-" (string - string), tak výsledok bude (string-string).

Dátový typ - Číslo

V SASS ako aj v CSS majú čísla dve časti. Prvá je číselná hodnota a druhá, voliteľná časť je jednotka čísla (px, em, ...). S číslami môžme robiť základné aritmetické operácie a taktiež použiť operáciu modulo, ktorá hovorí o zvyšku po delení dvoch čísel. Modulo označujeme znakom **%**. Pri násobení dajte pozor, aby ste nenásobili obe čísla s jednotkami, ako napríklad “20px * 20px”,

pretože SASS by to vyhodnotil ako chybu. Pri delení môže vzniknuť jedna zvláštnosť, a to že v prípade keď použijete operáciu deleno, napríklad s reťazcom, potom sa tento výraz vyhodnotí ako spojenie reťazcov. V prípade delenia môžte používať pri oboch číslach ich jednotky (Príklad 6.3).

Príklad 6.3

.scss

```
// 20px*10px => 200px*px => error
// 20px*10 => 200px
// 20px/10px => 2
// 20px/10 => 2px
// auto/10px => auto/10px
```

Dátový typ - Farba

Farby môžme zapísať v CSS rôznymi spôsobmi. Najbežnejší spôsob je zapísať farbu ako hexadecimálnu reprezentáciu kanálov RGB. Farba "#fff" reprezentuje 255(R-red), 255(G-green), 255(B-blue). Túto farbu môžme zadefinovať aj pomocou funkcie rgb(255, 255, 255).

Ďalšie funkcie sú hsl(), ktorá je podobná rgb(), hsla() a rgba(), kde môžme navyše zadefinovať transparentnosť farby. Farby môžme ešte zadefinovať pomocou názvu ako “blue”, “red” atď. V prípade farieb môžme používať všetky obyčajné aritemtické operácie, ale nie je to pre nás tak užitočné ako práca s funkciami.

Dátový typ - Zoznam

Zoznamy sú jednoduché sekvencie hodnôt, ktoré sú používané pre skladanie vlastností ako "border" alebo "background". Vlastnosť "1px solid #222" je zoznam troch hodnôt. Hodnoty môžu byť oddelené medzerami alebo čiarkami (Príklad 6.4). V CSS môžme používať iba jednotlivé hodnoty, ale v SASS môže zoznam obsahovať ďalšie zoznamy hodnôt (Príklad 6.5).

Príklad 6.4

.scss

```
$color1: #222;
$color2: #444;
```

```
$color3: #666;

$colors: $color1 $color2 $color3;
$colors: $color1, $color2, $color3;
```

Príklad 6.5

.scss

```
$color1: #222;
$color2: #444;
$color3: #666;
$color4: #222;
$color5: #444;
$color6: #666;

$colors: $color1 $color2 $color3, $color4 $color5 $color6;

//Iny zapis vnoreneho zoznamu
$colors: ($color1 $color2 $color3), ($color4 $color5 $color6);
$colors: ($color1 $color2) ($color3 $color4) ($color5 $color6);
```

Aritemtické operácie nemajú pri zoznamoch žiadne praktické využitie. Zoznamy sú veľmi praktické hlavne pri použití direktívy **@each**.

Dátový typ - Boolean

Tento dátovy typ budeme používať hlavne pri direktíve **@if**. Boolean nemá aritmetické operátory, ale namiesto nich má operátory "and", "or" a "not". Môžu byť použité s akoukoľvek hodnotou, ale najčastejšie budú používané s hodnotou typu Boolean. Ešte máme k dispozicií operátory <, <=, >, >=, ==.

Funkcie

Funkcie v SASS sa používajú podobne ako v CSS. V SASS sú navyše vyhodnotené ako súčasť výrazu a taktiež vracajú hodnotu. Funkcia môže mať aj pomenované argumenty podobne ako v mixine. Teraz sa pozrieme na niektoré vstavané SASS funkcie, ktoré nám môžu byť veľmi užitočné.

Funkcie - Čísla

SASS má niekoľko vstavaných funkcií pre prácu s číslami ako napríklad "ceil($cislo)", ktorá nám zaokrúhli číslo smerom nahor, "percentage($cislo)", ktorá nám zkonvertuje desiatkové číslo na percento a mnoho iných, podobných funkcií (Príklad 6.6).

Príklad 6.6

.scss

```
$number: 11.2px

body {
   font-size: ceil($number);
}
```

.css

```
body {
   font-size: 12px;
}
```

Funkcie - Farby

Funkcie farieb môžme rozdeliť do dvoch kategórií. Funkcie, ktoré nám vrátia informácie o farbe a funkcie, ktoré transformujú pôvodnú farbu na novú. Napríklad funkcia "alpha($farba)" je jedna z informatívnych funkcií a vráti alpha kanál konkrétnej farby. Na druhej strane funkcia "grayscale($farba)" je transformačná a vráti odtieň šedej (Príklad 6.7).

Príklad 6.7

.scss

```
$color: #ac2211;

body {
   background: complement($color);
}
```

.css

```
body {
    background: #5f5f5f;
```

```
}
```

Funkcie - Zoznamy

SASS má vstavané funkcie aj pre zoznamy a jedna z najviac používaných je funkcia "nth($zoznam, $index)", ktorá vráti konkrétny prvok zoznamu. Musíme si ale dať pozor, že SASS indexuje od jednotky (Príklad 6.8). Ďalšia užitočná funkcia pre zoznam je "length($zoznam)", ktorá vráti počet prvkov v zozname.

Príklad 6.8

.scss

```
$colors: (#222, #444, #666);
$first-color: nth($colors, 1);

body {
    color: $first-color;
}
```

.css

```
body {
    color: #222;
}
```

Všetky vstavané SASS funkcie nájdete v dokumentácií SASS.

Funkcie - Vlastná definícia

Na definovanie vlastných funkcií nám slúži direktíva **@function** a je pre nás užitočná hlavne pri opakovanom vypočte alebo transformácií farby. Vlastná funkcia pracuje podobne ako mixin, ale na rozdiel od mixinu nám funkcia musí vrátiť nejaký výsledok. Vrátenie výsledku definujeme pomocou direktívy **@return** (Príklad 6.9).

Príklad 6.9

.scss

```
$base-height: 100px;
```

```
@function set-smaller-height($height) {
   @return ($height - 20px) / 2;
}

body {
   height: set-smaller-height($base-height)
}
```

.css

```
body {
   height: 40px;
}
```

Výrazy v selektoroch a názvoch vlastností

CSS vlastnosti nie sú jediné miesto, kde môžme umiestňovať v rámci SASS nejaké CSS hodnoty. SASS nám umožňuje vkladanie výrazov aj do selektorov a názvov vlastností pomocou špeciálnej syntaxi **"#{výraz}"**. Táto vlastnosť je známa ako interpolácia a je vhodná hlavne pre prácu s mixinami (Príklad 6.10).

Príklad 6.10

.scss

```
@mixin own-class-and-property($class, $property-name){
   .class-#{$class} {
      property-#{$property-name}: #222;
   }
}

@include own-class-and-property("own", "special");
```

.css

```
.class-own {
   property-special: #222;
}
```

Kontrolné direktívy

Kontrolné direktívy sú špeciálnym typom direktív, ktoré riadia akým spôsobom sa kus kódu v SASS vyrenderuje v rámci CSS. Tieto direktívy zapisujeme pomocou syntaxes **@directive {...}**. Medzi kontrolné direktívy patrí **@for**, **@each** a **@if**.

Kontrolná direktíva - @for

Direktíva môže byť zapísaná dvoma spôsobmi. Prvý zápis je **@for $i from 1 to 5 {...}** a druhý zápis je**@for $i from 1 through 5 {...}**. Pre každý zápis je premenná "$i" nastavená na 1 a každým cyklom narastá o 1. Rozdiel medzi prvým zápisom a druhým je taký, že pri prvom zápise premenná $i sa zastaví na čísle 4 a pri druhom zápise sa zastaví na čisle 5. Namiesto čísel 1 a 5 v direktíve môžte samozrejme použiť premenné (Príklad 6.11).

Príklad 6.11

.scss

```
@for $i from 1 through 5 {
   .class-number-#{$i} {
      border: $i + 0px solid #222;
   }
}
```

.css

```
.class-number-1 { border: 1px solid #222; }
.class-number-2 { border: 2px solid #222; }
.class-number-3 { border: 3px solid #222; }
.class-number-4 { border: 4px solid #222; }
.class-number-5 { border: 5px solid #222; }
```

Ako sme videli tak direktíva @for môže byť vcelku užitočná. Ešte užitočnejšia je ale direktíva @each, ktorou môžme prechádzať zoznamy.

Kontrolná direktíva - @each

Direktíva je užitočnejšia práve v tom, že prechádza všetkými prvkami na rozdiel od direktívy @for, ktorá len cykluje v určitom pevne danom intervale. Túto direktívu zapisujeme pomocou **@each $prvok in $zoznam** (Príklad 6.12).

Príklad

.scss

```
$sections: home, service, contact;

@each $section in $sections {
   .#{$section} {
      background-image: url(/images/#{$section}.jpg);
   }
}
```

.css

```
.home    { background-image: url(/images/home.jpg); }
.service { background-image: url(/images/service.jpg); }
.contact { background-image: url(/images/contact.jpg); }
```

Kontrolná direktíva - @if

Posledná kontrolná direktíva sa zapisuje pomocou **@if podmienka {...}**. Ak je podmienka vyhodnotená ako "true" blok kódu v tele direktívy sa vykoná, v inom prípade bude telo direktívy ignorované (Príklad 6.13). S direktívou @if môžme použiť aj direktívu **@else** (Príklad 6.14).

Príklad 6.13

.scss

```
$optimize-for-opera: 1;

.radius {
   @if $optimize-for-opera {
      -o-border-radius: 10px;
   }

   border-radius: 10px;
```

```
}
```

.css

```
.radius {
   -o-border-radius: 10px;
   border-radius: 10px;
}
```

Príklad 6.14

.scss

```
$optimize-for-opera: 1;

.radius {
   @if $optimize-for-opera > 1 {
      -o-border-radius: 10px;
   } @else {
      border-radius: 10px;
   }
}
```

.css

```
.radius { border-radius: 10px; }
```

Záverom

V tejto časti sme si predstavili všetky skriptovacie vlastnosti SASS a objavili sme ich veľkú výhodu. Súčasne sme si pozreli poslednú časť SASS, ktorá nám bude nápomocná pri vytváraní prehľadného a dobre udržovateľného kódu našich projektov.