CSS preprocessor - Sass eller Less

Toni Laukkanen

Examensarbete
Informations- och medieteknik
2017

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Informations- och medieteknik
Identifikationsnummer:	6155.
Författare:	Laukkanen, Toni Harald Antero
Arbetets namn:	CSS preprocessor – Sass eller Less
Handledare (Arcada):	Johnny Biström
Uppdragsgivare:	-

Sammandrag:
I detta examsarbete diskuteras om vilken CSS preprocessor är bättre Sass eller Less. Båda preprocessorer var gjorda med samma idé, att göra skrivande av långa CSS koder lättare. Detta abete hänvisar till flera artiklar och webbsidor som är gjorda av forskare eller de som gjorde Less eller Sass. Målet med arbetet är att hitta skillnaderna mellan dem och vilken är bättre. I arbetet går jag igenom olika egenskaper, styrkor och svagheter båda preprocessorer har, jag gör en grundläggande underökning på deras egenskaper och hurdana portar de har. Hur skall CSS preprocessorer användas och hur nyttiga det är för olika arbeten som kräver CSS. I slutsatsen förklarar jag vilken preprocessor är bättre och varför.

Nyckelord:	CSS, Less, Sass, preprocessor, Mixins, Ruby, JavaScript
Sidantal:	37
Språk:	Svenska
Datum för godkännande:	9.5.2017

DEGREE THESIS	
Arcada	
Degree Programme:	Information and media technology
Identification number:	6155
Author:	Laukkanen, Toni Harald Antero
Title:	CSS preprocessor – Sass or Less
Supervisor (Arcada):	Johnny Biström
Commissioned by:	

Abstract:
In this thesis, I discuss which CSS preprocessor is better Sass or Less. Both preprocessors were made with the same idea, making the writing of long CSS coding easier. This work refers to several articles and websites made by a researcher or those who wrote Less or Sass. The aim of the work is to find the differences between the two preprocessors and which one is better. In my work, I go through the different properties, strengths and weaknesses of both preprocessors, I do a basic examination of their properties and their ports. How should CSS preprocessors be used and how useful it is for different tasks that require CSS. In my conclusion, I explain which preprocessor is better and why.

Keywords:	CSS, Less, Sass, preprocessor, Mixins, Ruby, JavaScript
Number of pages:	37
Language:	Swedish
Date of acceptance:	9.5.2017

INNEHÅLL

Figurer

1. Inledning

I stora webbutvecklingsprojekt kan det vara besvärligt att hålla reda på allting. Ett sätt att förenkla arbetet är att använda CSS preprocessorer Less eller Sass men problemet är vilken skall jag använda.

Less eller Sass har varit en fråga som flera har försökt svara på inom webbutvecklingscirklar. I detta arbete så går jag igenom både Sass och Less, ser på deras styrkor och svagheter för att hitta vilket är det bättre CSS preprocessorn.

Less och Sass har mycket gemensamt. Skillnaderna är som mest i detaljerna, men de har liknande idéer.

1.1 Syfte och mål

Syftet med detta arbete är att hitta skillnaden mellan Sass och Less varför är den ena bättre än den andra och i vilka tillfällen borde den ena användas över den andra. Min hypotes är att Sass och Less har olika styrkor och borde användas i sina egna tillfällen.

1.2 Metoder och avgränsningar

Arbetet kommer att vara baserat på Google-sökningar och olika dokumentationer för arbetet. Arbetet fokuserar sig på Less och Sass, men jag kommer att först förklara kort om CSS. Till sist kommer jag att jämföra Sass och Less. Jag kommer också att ta upp kommentarer som användaren har. Flera av artiklarna är gamla och det kan begränsa giltigheten av informationen i dem.

1.3 Struktur

I första delen av arbetet förklaras det kort om CSS sedan en längre redogörelse av Sass och Less. I andra delen så jämförs Sass och Less steg för steg och slutligen resultatredovisning.

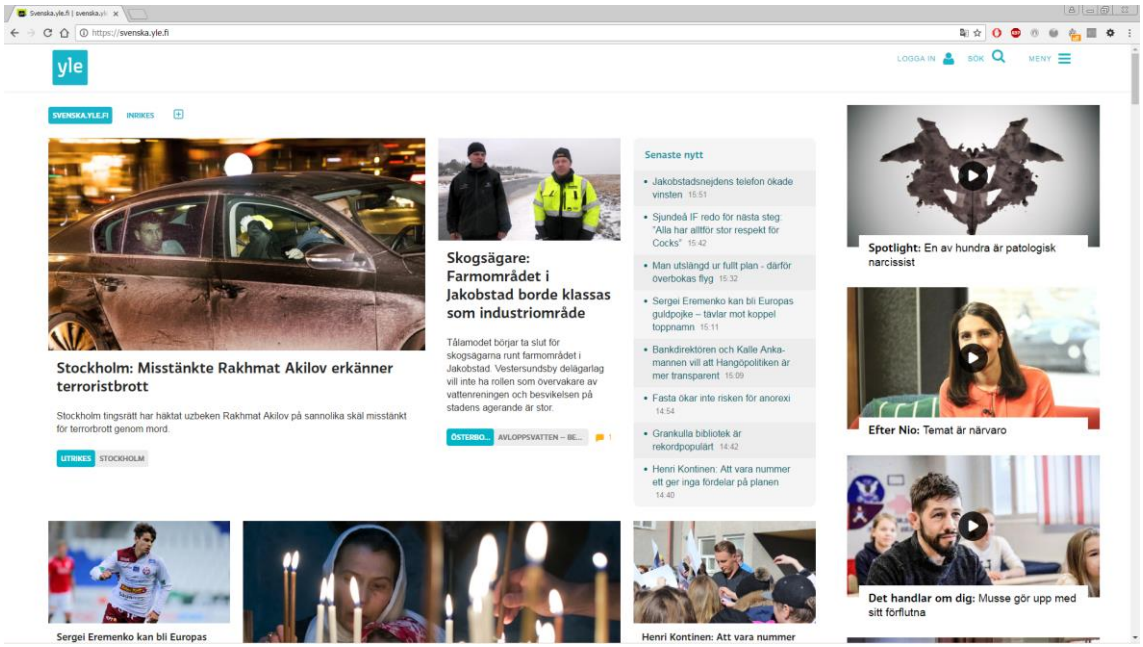
2. CSS

Cascading Style Sheets (CSS) är ett sätt att presentera färg och bilder i ett HTML dokument. Med CSS så kan man ange olika teckensnitt för texten eller förändra färgen eller storleken av texten även placeringen av text och bilder kan förändras. Den nyare versionen CSS3 har flera andra funktioner t.ex. rotation, skuggor bakom text och responsiv design. Den är också mera kompatibel för olika skärmstorlekar med hjälp av media queries i stilmallarna (Saleh 2015).

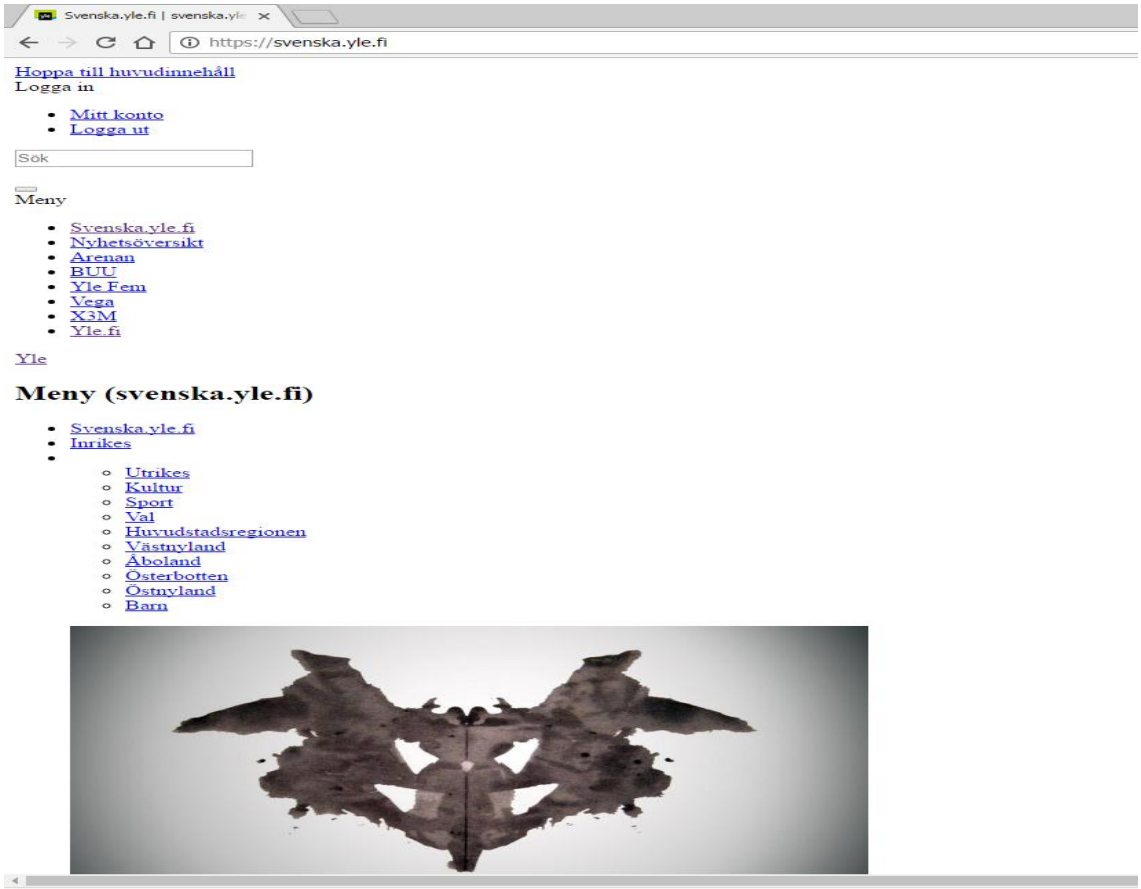
CSS var inte den första stilmallen. Det fanns flera andra t.ex. WYSIWYG (What You See Is What You Get) Dessa stilmallar hade flera av de samma funktionerna som CSS har. Men tre dagar före Netscape annonserade sin nya webbläsare, publicerade en författare det första CSS förslaget (namedCascading HTML style sheets – a proposal) [Lie 1994] i webben. Förutom att beskriva typsnitt, färger och layout av dokument som flera förslag hade gjort tidigare införde CSS nya funktioner för att redogöra för skillnader i publicering som införts i webben. Begreppet kaskad tillät både författare och användare för att påverka presentationen av ett dokument. Förhandlanden mellan behov och önskemål mellan läsaren och författaren var en av CSS ambitioner. Om det lyckas skulle författare få sin beskärda del av kontroll över presentationen och skulle inte var tvingade att använda presentations HTML och andra metoder. Användaren skulle serveras dokument i en form i vilken de kan välja mellan att acceptera författarens föreslagna presentation eller ange sina egna.

I flera fall skulle det inte finnas någon konflikt mellan författaren och användaren. I såna fall när ingendera vill specificera presentationen är det viktigt att webbläsaren har en standardformatmall som beskriver en standard presentation av HTML-dokument. CSS, definierar därför tre möjliga källor för formatmallar: författare, användare och webbrowser. CSS kan kombinera stilmallar från dessa källor för att bilda en presentation av ett dokument. Processen att kombinera flera formatmallar och lösa konflikter om de inträffar kallas kaskad. (Lie 2005)

På nästa sida finns det bilder av en webbsida med och utan CSS.



Figur 1 Yles Hemsida med CSS



Figur 2 Yles Hemsida utan CSS

3. Sass

Sass (syntactically awesome stylesheets) är en tillbyggnad av CSS3 som hjälper dig att skapa bättre stilmallar med mindre ansträngning. Sass frigör dig från repetition och ger dig verktyg för att vara kreativ. Eftersom du kan genomföra ändringar mycket snabbare, är du fri att ta risker i din design. Dina stilmallar kommer att kunna hålla jämna steg med växlande av färger och ändra HTML-märkningar. Samtidigt producerar du standardbaserad CSS som kan användas i alla miljöer. Sass-processorn är skrivet i Ruby, men för att använda Sass kräver det inte kunskap om Ruby-språket (Wynn 2013).

Sass är en stilmall språk som ursprungligen designades av Hampton Catlin och utvecklades av Natalie Weizenbaum. Efter de första versionerna fortsatte Weizenbaum och Chris Eppstein att utvidga Sass med SassScript, ett enkelt skript språk som används i Sass-filer. Sass publicerades första gången 28.11.2006 (Sass Wikipedia 2017).

Målet med Sass är att fixa brister i CSS. CSS, som alla vet, är inte det bästa språket i världen. Medan det är mycket enkelt att lära sig kan det ganska snabbt bli råddigt, särskilt på större projekt. Sass som ett metaspråk kan förbättra CSS:s syntax genom extra funktioner och praktiska verktyg. Sass vill vara konservativt när det gäller CSS-språket. Poängen är inte att göra CSS till ett fullt utrustat programmeringsspråk. Sass vill bara hjälpa med det där CSS misslyckas. På grund av detta är det inte svårare att komma igång med Sass än att lära sig CSS: det lägger helt enkelt till ett par extrafunktioner ovanpå den.

Ruby Sass är original Sass som är skriven i Ruby. Efter Ruby Sass har det kommit flera likande program men med olika portaler, den mest kända är LibSass som är skriven i C/C++. De nyaste versionerna av Ruby Sass och LibSass har total kompatibilitet som det beskrivs på sidan: http://sass-compatibility.github.io/ . Orsaken varför LibSass gjordes var för att Ruby språket var inte tillräckligt snabbt och det blir bara långsammare i större projekt. Äldre versionerna av LibSass var inte lika

kompatibla som Ruby Sass men den nyaste versionen är lika kompatibel som Ruby Sass så jag rekommenderar LibSass (Giraudel 2017).

3.1 Sass Kompatibilitet

En av de grundläggande principerna för Sass har alltid varit att förstå CSS så lite som möjligt. Som en CSS-preprocessor måste vi förstå syntaxen av CSS, men så mycket som vi kan försöka undvika lärande av meningen bakom stilarna. Det betyder att Sass har ingen aning om vilka egenskaper som är giltiga, vilka HTML-element som existerar, eller till och med i stor omfattning vad syntaxen för de flesta @ -rules är.

Designerna får mycket nytta av detta. Den mindre inbyggda kunskapen Sass har om CSS, desto mindre är det att fungera dåligt med nya CSS-funktioner. Designerna behöver inte lämna en ny funktionsförfrågan varje gång de vill använda en ny CSS-egenskap. Istället kommer äldre versioner av Sass gärna att fungera om inte den aktuella syntaxen ändras, vilket är väldigt sällsynt.

På grund av denna avkoppling behöver designers inte oroa sig över webbläsarkompatibilitet. Sass fungerar med vilken CSS som det används med. Användaren är den som bestämma vad som fungerar där, vilket ger dem mycket flexibilitet och ger designerna färre svåra beslut att utföra.

Idén bakom Sass är att lindra på svårigheten av att använda CSS, därför måste Sass vara lika enkel som CSS. Om du kan lära dig att använda CSS så kan du lära dig att använda Sass.

Men trots denna allmänna policy finns det alltid några fall där CSS-kunskapen visar sig vara nödvändig. En stor är @extend, som behöver veta mycket om betydelsen av

selektorer för att ordentligt förena dem och utplåna upprepningar. Egenskapsvärden kräver ibland också semantisk kunskap, till exempel måste designerna veta hur man tolkar färger (Sass and Browser 2017).

3.2 Sass och SCSS

Det finns två syntaxer tillgängliga för Sass. Den första kallas SCSS (Sassy CSS) och används i hela denna referens, är en utvidgning av CSS-syntaxen. Det betyder att alla giltiga CSS stilmall är en giltig SCSS-fil med samma innebörd. Dessutom förstår SCSS de flesta CSS-hack och leverantörsspecifik syntax, t.ex. IE: s gamla filtersyntax. Denna syntax förbättras med Sass-funktionerna. Filer som använder den här syntaxen har tillägget .SCSS.

Den andra och äldre syntaxen, känd som den indragna syntaxen, ger ett mer kortfattat sätt att skriva CSS. Den använder indragning i stället för parentes för att indikera nestning av selektorer, och nya rader hellre än semikolon för att skilja egenskaper. Vissa tycker att det här är lättare att läsa och snabbare att skriva än SCSS. Filer som använder den här syntaxen har .sass-tillägget. Den indragna syntaxen har alla samma egenskaper, även om vissa av dem har lite annorlunda syntax (Sass 2016).

Sass beskrev initialt en syntax, av vilken den definierande karaktäristiken var dess indragningskänslighet. Senare bestämdes det att Sass-underhållare skulle stänga gapet mellan Sass och CSS genom att tillhandahålla en CSS-vänlig syntax kallad SCSS eller Sassy CSS. Idén är att om det är giltigt CSS, är det giltigt SCSS (Giraudel 2017).

SCSS:

```
#main {
  color: red;
  font-size: 0.3em;

  a {
    font: {
      weight: bold;
      family: serif;
    }
    &:hover {
      background-color: #eee;
    }
  }
}
```

Sass:

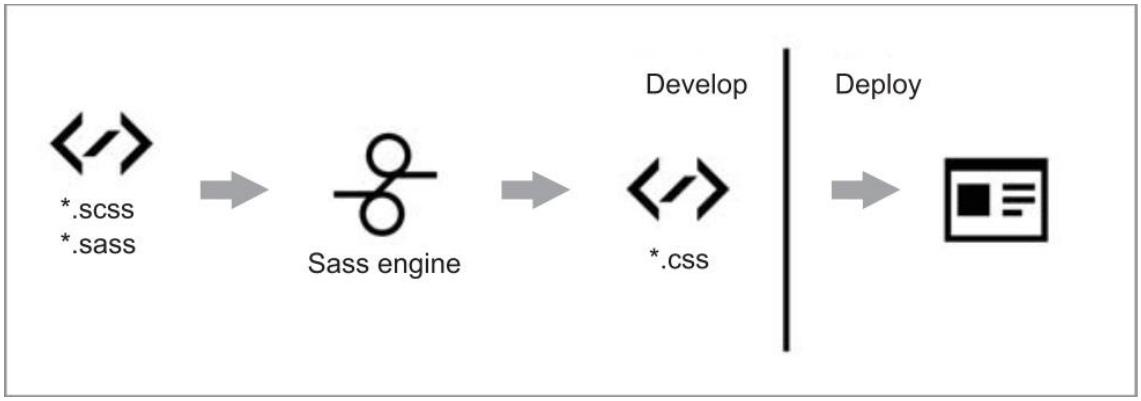
```
#main
  color: red
  font-size: 0.3em

  a
    font:
      weight: bold
      family: serif
    &:hover
      background-color: #eee
```

Den ursprungliga indragna syntaxen har en .sass-förlängning och är medvetande om mellanslag, så i stället för omringa egenskaper med hängslen sätter du in dem under selektor. Iställe för att använda semikolon, separeras varje egenskap med en ny rad. Som det visas i exemplen ovanför. Sass fortsätter att stöda både Sass och SCSS. Av de två syntaxerna är SCSS lättare att lära sig för att det liknar CSS. Det har bara lite extra.

3.3 Hur Sass Fungerar

Sass är inte magiskt språk som kan fixa alla dina problem. Det hjälper inte direkt dina färg-, typografi- eller layout val, men det kan hjälpa dig att implementera dina idéer snabbare, med mindre friktion.

När du använder Sass, sammanställer Sass-Engine dina stilmall-källfiler till totalt rena CSS under utvecklingsarbetet. Även om det finns många alternativ för att köra Sass-Engine, som sträcker sig från kommandoraden till serverns ramintegrering till GUI-verktyg, är poängen är att Sass producerar CSS samtidigt som du arbetar. Du distribuerar statisk CSS som du normalt skulle. Du kan ta nytta av Sass språkfunktioner för att skriva CSS mycket snabbare och lättare. (Wynn 2013)



Figur 3 Hur Sass fungerar

Figur 3 visar hur Sass förändras till CSS om du vill själv prova så kan du använda sidan: https://www.sassmeister.com/

Sass v3.4.21

```
$company-blue: #1875e7;
h1#brand {
color: $company-blue;
}
#sidebar {
background-color: $company-blue;
}
ul.nav {
float: right;
li {
float: left;
a{
color: #111;
}
&.current {
font-weight: bold;
}
}
}
#header ul.nav {
  float: right;
  li{
    float:left;margin-right:10px;
  }
}
#footer ul.nav {
  margin-top:1em;
  li{
    float:left;margin-right:10px;
  }
}
```

CSS

```
h1#brand {
  color: #1875e7;
}

#sidebar {
  background-color: #1875e7;
}

ul.nav {
  float: right;
}
ul.nav li {
  float: left;
}
ul.nav li a {
  color: #111;
}
ul.nav li.current {
  font-weight: bold;
}

#header ul.nav {
  float: right;
}
#header ul.nav li {
  float: left;
  margin-right: 10px;
}

#footer ul.nav {
  margin-top: 1em;
}
#footer ul.nav li {
  float: left;
  margin-right: 10px;
}
```

Figur 4 Exempel på hur Sass kan gör arbete lättare

Som det syns på figur 4 så gör Sass arbetet simplare. Vi kan ge färgen `#1875e7` ett namn `$company-blue` för att göra det lättare att komma ihåg. Med att ge färgen en variabel gör det arbetet lättare.

3.4 Användningen av Sass

För att använda Sass måste du först installera Ruby det kan göras från kommandoraden:

`gem install sass`

om du har Windows så kan det instaleras på: http://rubyinstaller.org/ .

Sass använder sig av nästlade regler. Sass tillåter CSS-regler att nästlas inuti varandra. Den inre regeln gäller då endast inom den yttre regelns selektor. Ett exempel på detta är `ul.nav` i figur 4. Detta hjälper till att undvika upprepning av föräldrarselektorer, och gör mycket komplicerade CSS-layouter med många nästade selektorer mycket simplare. Till exempel:

Sass v3.4.21

```
#main {
  width: 87%;

  p, div {
    font-size: 3em;
    a { font-weight: bold; }
  }

  pre { font-size: 5em; }
}
```

CSS

```
#main {
  width: 87%;
}
#main p, #main div {
  font-size: 3em;
}
#main p a, #main div a {
  font-weight: bold;
}
#main pre {
  font-size: 5em;
}
```

Figur 5 Exempel på hur Sass kan undvika repetition

Förutom den enkla CSS-egenskapssyntaxen stödjer Sass en liten uppsättning av tillägg som heter SassScript. SassScript tillåter egenskaperna att använda variabler, aritmetiska och andra extrafunktioner. SassScript kan användas i valfritt egenskapsvärde. Det lättaste sättet att använda SassScript är med variabler. SassScript variabler börjar med et dollartecken. Vanligtvis är variabler bara definierade inom selektorerna men med att använda `!global` kan det användas utanför selektorn som det visas i följande exempel:

Figur 6 SassScript exempel.

Strängar spelar en ganska stor roll i både CSS och Sass. De flesta CSS-värden är antingen längder eller identifierare, så det är ganska viktigt att hålla sig till några riktlinjer när man hanterar strängar i Sass.

För teckenkodning i Sass rekommenderas det starkt att tvinga UTF-8-kodning i huvudformatarket med @charset-direktivet. Se till att det är det första elementet i stilarket och det finns ingen karaktär av något slag före det.

```
@charset 'utf-8';
```

I Sass är numror en data typ som inkluderar allting från enhetslösa tal till längder, varaktighet, frekvenser, vinklar och så vidare. Detta gör att beräkningar kan köras på sådana mått. Sass gör ingen åtskillnad mellan siffror, heltal, floats så att släpande nollor (0) bör utelämnas. En nollängd (0) bör inte ha en enhet. Enhetsmanipulation bör anses som aritmetiska operationer, inte sträng operationer.

Färger upptar en viktig plats i CSS-språket. Naturligtvis så hjälper Sass mycket med färgmanipulation med hjälp av funktioner. `hue()`, `saturation()` och `lightness()` är färgmanipulationsfunktioner. Färger bör uttryckas i HSL när det är möjligt, sedan RGB, sedan hexadecimalt (i små och korta form). Färgnyckelord bör undvikas.

Lists är Sass versionen av array. Med Sass kan stilmallsförfattare definiera maps vilket är Sass-termen för associativa arrays, hashes eller till och med JavaScript-objekt. En map är en datastruktur som associerar nycklar till värden. Både nycklar och värden kan vara av någon datatyp, omfattande maps, det rekommenderas att inte använda komplexa datatyper som kartnycklar.

En särskild funktion som i Sass är missbrukad av många utvecklare är selektor nästning. Selektor nästling erbjuder ett sätt för stilmallsförfattare att skapa långa selektorer genom nästlingen av kortare selektorer inom varandra. Problemet med selektorns nästing är att det slutligen gör koden svårare att läsa. Man måste mentalt beräkna den resulterande selektorn ur indragningsnivåerna. Detta uttalande blir allt sannare eftersom selektorn blir längre och referenser till den nuvarande selektorn (&) tätare. Eventuelt kan du inte hålla reda på vad som händer längre, det är inte bara värt det. För att förhindra sådana situationer borde selektorn inte vara mera än 3 nivåer djup (Giraudel 2017).

Om du är intresserad av att lära dig att använda Sass så kan du använda sidan: http://sass-lang.com/

4. Less

Less är ett dynamiskt stilmallspråk som kan kompileras till CSS och köras på klientsidan eller serverns sida. Less var påverkad av Sass och har påverkat den nyare SCSS -syntaxen av Sass, som anpassade sin CSS-liknande blockformaterings syntax. Less är en öppen källkod. Dess första version skrevs i Ruby; I senare versioner har användningen av Ruby förminskas och ersatt av JavaScript. Den indragna syntaxen för Less är ett kapslat metaspråk, eftersom giltig CSS är giltig Less kod under samma semantik. Huvudskillnaden mellan mindre och andra CSS-förprocessorer är att Less tillåter realtidskompilering via less.js av webbläsaren (Less Wikipedia 2017).

Less skapades 2009 av Alexis Sellier, mer känd som @cloudhead. Ursprungligen skrivet i Ruby, var det sedan portat till JavaScript. I maj 2012 lämnade Alexis kontroll och utveckling över till ett team av medarbetare som nu hanterar, fixar och utvidgar språket.

Som en förlängning av CSS är Less inte bara bakåtkompatibel med CSS, extra funktionerna har tagits från nu existerande CSS-syntax. Detta gör inlärningen av Less mycket lätt, och om du har problem kan du bara falla tillbaka på grundläggande CSS programmering (Less 2017).

Den enkla designen av CSS gör det väldigt tillgänglig för nybörjare, men det utgör också begränsningar för vad du kan göra med det. Dessa begränsningar, som oförmågan att ställa in variabler eller att utföra operationer, innebär att vi oundvikligen måste upprepa samma programmeringsstilar på olika ställen. Det är inte en effektiv metod om du vill hålla fas vid DRY (don’t repeat yourself) metoden för mindre kod och enklare underhåll (Fadeyev 2010).

4.1 Less.js portering

Less har portats till flera olika programmeringsspråk:

Java

- Lesscss (Lesscss kompilerar Less-kod till CSS. Detta projekt har avslutats)
- Lesscss gradle plugin (En gradle plugin för att kompilera LESS till CSS, kompatibel med Less v1.7.0)
- Less Engine (Less Engine ger grundläggande tillgång till kärnan Less-funktionalitet. Det är ett kärnbibliotek som kan användas för en mängd olika JVM-baserade Less-applikationer.)
- Less CSS Compiler for Java (Less CSS kompilerare för Java är ett bibliotek för att kompilera LESS-källor till CSS. Kompilatorn använder Rhino, Envjs (simulerad webbläsarmiljö som skrivs i JavaScript) och den officiella Less JavaScript-kompilatorn.)
- Less4j (Infödd Java implementering.)
- JLessC (Less kompilatorn är totalt skriven i Java)

PHP

- Lessphp (Lessphp är en kompilator som genererar CSS från ett superset-språk som lägger till flera praktiska funktioner som ofta ses på andra språk.)
- BW Less CSS (WordPress Plugin så att du kan lätt bifoga Less stilmallar till ditt tema.)
- less.php(PHP port)
- ILess(PHP port)

.Net

- .Less (en .NET port för Less)
- BundleTransformer.Less (BundleTransformer.Less innehåller översättningsadapter ´LessTranslator´. Denna adapter gör översättning av LESS-kod till CSS-kod.

Innehåller även felsökning HTTP-hanteraren ´LessAssetHandler`, som är ansvarig för textutmatning av översatt LESS-tillgång.)

Python

- LESSCPY (En kompilerare skriven i Python för Less språket)

Ruby

- less.rb (Less.js i Ruby's V8 Engine)

Go

- less-go (Kör less.js i en inbäddad Javascript engine. Less kompilerare för Golang) (Less 2017).

4.2 Kompilator

Till att börja med Less, måste vi konfigurera en kompilator. Less syntax är icke-standard, per W3C-specifikation (World Wide Web Consortium). Webbläsaren skulle inte kunna bearbeta och göra utmatningen, trots att det har flera egenskaper som liknar CSS.

Less

```less
@color-base: #2d5e8c;
.class1 {
  background-color: @color-base;
  .class2 {
    background-color: #fff;
    color: @color-base;
  }
}
```

CSS

```css
.class1 {
  background-color: #2d5e8c;
}
.class1 .class2 {
  background-color: #fff;
  color: #2d5e8c;
}

```

Figur 7 Exempel på Less

Det finns flera verktyg för att kompilera CSS:

4.2.1 JavaScript

Less kommer med en less.js-fil som är lätt distribuerat till din webbplats. Du kan göra det med att skapa ett stilmall med .less extensionen och länka det i ditt dokument med attributet rel = "stylesheet / less".

```
<link rel="stylesheet/less" type="text/css" href="main.less" />
```

Du kan ladd JavaScript filen från nätet, eller göra en direkt link till CDN.

```
<link rel="stylesheet/less" type="text/css" href="main.less" />
<script
src="//cdnjs.cloudflare.com/ajax/libs/less.js/2.5.1/less.min.js"></script>
```

Less syntaxen kommer att kompileras samtidigt som sidan laddas. Det rekommenderas inte att använda JavaScript i produktionsfasen, eftersom det försämrar webbplatsens prestation.

4.2.2 CLI

Less är utrustat med ett inbyggt kommandoradsgränssnitt (CLI), lessc, som hanterar flera uppgifter förutom att bara kompilera Less syntaxen. Med CLI kan vi binda ihop koderna, komprimera filerna och skapa en källkarta. Kommandot är grundat på Node.js som effektivt tillåter kommandot att fungera över Windows, Linux och OS X. För att få det att fungera så behöver du Node.js instalerat på datorn och sedan kan du instalera det med följande kommandet:

```
npm install -g less
```

nu kan du använda lessc kommandot för att kompilera Less till CSS:

```
lessc test.less test.css
```

4.2.3 Task Runner

Task Runner är ett verktyg som automatiserar utvecklingsuppgifter och arbetsflöden. I stället för att köra kommandot `lessc` varje gång vi vill kompilera våra koder, kan vi installera Task Runner och ställa in den för att iaktta ändringar i Less filer och omedelbart kompilera Less till CSS vilket tillåter dig att arbeta dynamiskt.

4.2.4 Grafisk Gränssnitt

De som är inte vill använda Terminal- och kommandorad, kan istället välja ett grafiskt gränssnitt. Det finns flera applikationer som kan kompilera LESS för alla plattformar. Flera är för gratis och några kostar. Men det är inte viktigt vilken det är så länge som det fungerar och fungerar väl med dit arbetsflöde (Thoriq 2016).

4.3 Användningen av Less

När du vill använda dig av Less måste du först bestämma vilket språk du vill använda och hur du kompilerar det. Det simplaste att börja med är JavaScript. Metoden för att använda JavaScript beskrivs på ett tidigare kapitel:

```
<link rel="stylesheet/less" type="text/css" href="main.less" />
<script
src="//cdnjs.cloudflare.com/ajax/libs/less.js/2.5.1/less.min.js"></script>
```

På detta sätt behöver du inte ladda eller installera någonting.

4.3.1 Variabler

En variabel är ett symboliskt namn som är associerat med ett värde och vars samhörande värde kan ändras. Här är ett exempel på en variabel i Less:

Figur 8 Exempel på Less varabler

4.3.2 Mixins

Ibland kan vi skapa en stil som kommer att upprepas flera gånger i stilmallen. Ingenting hindrar dig från att tillämpa flera klasser på elementen i HTML, men det kan du också göra utan att lämna din stilmall, med hjälp av Less. Mixins är ett sätt att inkludera flera egenskaper från en regeluppsättning till en annan. Om du har följande klass:

```
.bording {
  border-top: dotted 2px black;
  border-bottom: solid 3px black;
}
```

Och vi vill använda samma egenskaper i en annan regeluppsättning så måste vi använda klassnamnet var vi vill ha egenskaperna:

```
#menu i {
  color: #121;
  .bording;
}

.post i {
  color: red;
  .bording;
}
```

Egenskaperna för den `.bording` klassen kommer nu att visas i både `#menu i` och `.post i`.

Less

```
.bording {
  border-top: dotted 1px black;
  border-bottom: solid 2px black;
}
#menu i {
  color: #121;
  .bording;
}

.post i {
  color: red;
  .bording;
}
```

CSS

```
.bording {
  border-top: dotted 1px black;
  border-bottom: solid 2px black;
}
#menu i {
  color: #121;
  border-top: dotted 1px black;
  border-bottom: solid 2px black;
}
.post i {
  color: red;
  border-top: dotted 1px black;
  border-bottom: solid 2px black;
}
```

Figur 9 Exempel på Less Mixins

4.3.3 Nästlade Regler

I CSS måst vi upprepa varje regeluppsättning, som kan leda till långa selektorer men med att använda Less nästing kan detta förkortas. Det finns ett exempel på följande sidan.

Less

```
#header {width: 300px;
  #nav {size : 500px;
    ul {float: right;
      &.li {color:yellow;
        a {color:red}
      }
    }
  }
}
```

CSS

```
#header {
  width: 300px;
}
#header #nav {
  size: 500px;
}
#header #nav ul {
  float: right;
}
#header #nav ul.li {
  color: yellow;
}
#header #nav ul.li a {
  color: #ff0000;
}
```

Figur 10 Exempel på nästlade regler i Less

4.3.4 Funktion

I Less är det möjligt att utföra funktioner, till exempel vi kan använda addition, subtraktion, multiplikation och division för att få värden till tal, färger och variabler i stilmallen. Låt oss säga att vi vill att element B ska vara fem gånger mera än element A. Då kan vi skriva det på detta sätt:

Less

```less
@height: 100px;
@size: 2cm * 3mm;
// omöjligt att konvertera
@incompatible-units: 3 + 7px - 8cm; // svaret blir 2px
@percentage: 0.33;

.element-A {
  height: @height;
}
.element-B {
  width: @height * 5;
}
.element-C {
  size: @size;
  height: @incompatible-units;
  width: percentage(@percentage);
}
```

CSS

```css
.element-A {
  height: 100px;
}
.element-B {
  width: 500px;
}
.element-C {
  size: 6cm;
  height: 2px;
  width: 33%;
}
```

Figur 11 Exempel på funktioner i Less

Exemplet visar hur matematiska funktioner kan användas i Less. Exemplet visar också hur multiplikation kan inte användas om talen kan inte konverteras t.ex. du kan inte konvertera px till cm.

4.3.5 Namespaces och accessorer

Vad händer om du vill gruppera variabler eller mixins i separata buntar? Du kan göra detta genom att nästla dem inuti en regeluppsättning med ett ID, som `#bundle`. Här är ett exempel på detta:

Less

```less
#bundle {
  .bordered {
    border: solid white;
    &:hover {
      background-color: white
    }
  }
}
#header a {
  color: black;
  #bundle > .bordered;
}
```

CSS

```css
#bundle .bordered {
  border: solid white;
}
#bundle .bordered:hover {
  background-color: #ffffff;
}
#header a {
  color: black;
  border: solid white;
}
#header a:hover {
  background-color: #ffffff;
}
```

Figur 12 Exempel på Namespaces och accessorer i Less

Observera att variabler som deklareras inom en namnrymd, kommer endast att exictera inom omfattningen till den namnrymden och kommer inte att vara tillgängliga utanför ramen via samma syntax som du skulle använda för att referera till en mixin (`#Namespace> .mixin-namn`). Som ett exempel, kan du inte göra följande: (`#Namespace> @ fungerar-inte`).

4.3.6 Scope

Omfattning av Less är väldigt likt det för programmeringsspråk. Variabler och mixins är söks förs lokalt, och om de inte hittas kommer kompilatorn att söka från en bredare omfattning.

Less

```less
header {
  @color: red;
  background-color: @color;
  nav {
      @color: orange;
      background-color: @color;
      a {
        color: @color;
      }
  }
}
```

CSS

```css
header {
  background-color: #ff0000;
}
header nav {
  background-color: #ffa500;
}
header nav a {
  color: #ffa500;
}
```

Figur 13 Exempel på Scopes i Less

I exemplet ovan har huvudet en röd bakgrundsfärg, men navens bakgrundsfärg kommer att vara orange eftersom den har variabeln `@color` i sin lokala omfattning, medan `a` kommer också att vara orange som är ärvt från sin närmaste förälder, `nav`.

4.3.7 Andra egenskaper

Det kommer att finnas tillfällen när du behöver inkludera ett värde som inte är ett CSS-syntax eller som Less känner inte.

Less

```
.weird-element {
  content: ~"/* en css hack";
}
```

CSS

```
.weird-element {
  content: /* en css hack;
}
```

Figur 14 Less exempel på okända värden

Figur 14 visar också användningen av kommentarer för att lämna meddelanden i koden.

Importerande fungerar också på Less t.ex. `@import ″typo.css″;`(Less 2017).

5. Less eller Sass

Det har varit flera debatter över vilken är bättre Less eller Sass. Sass var gjort först, Less tog sin inspiration från Sass och gjorde en preprocessor som liknade CSS mera än Sass. För att svara på denna utmaning så gjorde Sass SCSS som liknade CSS. Vilken preprocessor är slutligen bättre? För att svara på frågan går jag igenom olika egenskaperna av Sass och Less.

Huvudskillnaden mellan Less och Sass är hur de behandlas. Less är ett JavaScript-bibliotek och bearbetas därför på klientsidan. Medan Sass körs på Ruby och bearbetas på serversidan. Många utvecklare kanske inte väljer Less på grund av den extra tid som behövs för att JavaScript-motorn ska behandla koden och mata ut den modifierade CSS till webbläsaren. Det finns några sätt att undvika detta. Du kan använda Less endast under utvecklingsprocessen. När det är färdigt kan du kopiera och klistrar in Less utdata i en minimerare och sedan in i en separat CSS-fil som ingår i stället för Less filerna. Ett annat alternativ är att använda LESS.app för att kompilera och minska Less filerna. Båda alternativen minskar storleken för dina stilar, och undviker eventuella problem som kan uppstår av klientens webbläsare som inte kör JavaScript.

Såsom Less kan Sass också sammanställas lokalt, och den sammanställda CSS kan skickas med projektet, WordPress-tema, Expression Engine-mall eller vad som helst till servern precis som de tidigare CSS-filer (Hixon 2011).

Den enda inlärningskurvan är syntaxen. Du bör använda en app som CodeKit, LiveReload eller Mixture för att se och sammanställa dina författade filer. Du behöver inte veta någonting om Ruby eller kommandoraden eller något annat det är inte en faktor här. Det faktum att Sass är i Ruby och LESS är i JavaScript har liten konsekvens för de flesta potentiella användare.

5.1 Mixins

Med båda språken kan du skriva egna mixins för att hjälpa till med leverantörsprefix. Men vill du gå tillbaka till dina andra projekt och uppdatera prefixen manuellt för alla dina projekt? Då måste du också uppdatera dina handgjorda mixins-filer.

I Sass kan du använda Compass. Compass kommer att hålla sig uppdaterad, och hantera prefixsituationen för dig. Bourbon är också bra.

I Less finns det också några mixinbibliotek som kämpar för att vara bäst. Jag har förstått att tidigare i Less språket det inte var möjligt att bygga bra bibliotek ovanpå det.

I både Sass och Less är det fråga om vilket bibliotek är mera aktuellt (Coyier 2012).

5.2 Språk kapacitet

Less har en förmåga att göra "bevakade mixins". Dessa är mixins som bara påverkar när ett visst villkor är sant. Kanske vill du ange en bakgrundsfärg baserad på den aktuella textfärgen i en modul. Om textfärgen är ljus kommer du att villa ha en mörk bakgrund. Om det är ganska mörkt vill du ha en ljus bakgrund.

Less

```less
.box-1 {
  @text-color: black;
  .set-bg-color(@text-color);
}
.set-bg-color (@text-color) when
(lightness(@text-color) >= 50%) {
  background: black;
}
.set-bg-color (@text-color) when
(lightness(@text-color) < 50%) {
  background: white;
}
```

CSS

```css
.box-1 {
  background: white;
}

```

Figur 15 Less exempel på ”bevakade mixins”

Less kan också själv göra referensrekursion där en mixin kan fråga sig själv med ett uppdaterat värde som skapar en loop.

Less

```less
.loop (@index) when (@index > 0) {
  .myclass {
    z-index: @index;
  }
// kallar sig själv
.loop(@index - 1);
}
// Stoppar loopen
.loop (0) {}

// skriver ut värden
.loop (4);
```

CSS

```css
.myclass {
  z-index: 4;
}
.myclass {
  z-index: 3;
}
.myclass {
  z-index: 2;
}
.myclass {
  z-index: 1;
}

```

Figur 16 Less loop

Men det här är alla logiska / looping förmågor som Less har. Sass har faktiska logiska och looping operatörer på språket. if/then/else statements, for loops, while loops och each loops. Detta är bara ordentlig programmering. I denna matchen vinner Sass.

Less använder `@`, Sass använder `$`. Dollartecknet har ingen arvsbetydelse i CSS, medan @-tecknet har. Det gäller saker som att förklara @keyframes eller block av @media-frågor. Vilken av dessa är bättre kan drags ner till personlig preferens (Coyier 2012).

5.3 Matematik

För det mesta är matematiken likadan, men det finns några konstigheter med hur enheter hanteras. Less till exempel antar att den första enheten du använder är det du vill ha, och ignorerar ytterligare enheter (Coyier 2012).

Less

```
@units: 2 + 5px - 3g;
.basic_operations {
    height: @units;
}
```

CSS

```
.basic_operations {
  height: 4px;
}
```

Figur 17 Matematik exempel på Less

Samma funktion i Sass ger ett fel och förklarar att värdarna är inkompatibla.

Sass v3.4.21

```
$units: 2 + 5px - 3g;
.basic_operations {
    height: $units;
}
```

CSS

```
Incompatible units: 'g' and 'px'.
```

Figur 18 Matematik exempel på Sass

5.4 Stylus

Stylus är en annat CSS-preprocessor språk som var inspirerad av Sass. Stylus var byggd med Node.js och kan köras i en browser. Stylus är den största CSS-preprocessorn efter Sass och Less. Även om den är inte lika populär som Sass och Less så har den sin egen lilla kommun av användare. Här är ett exempel som togs från Stylus Hemsidan (Stylus 2017):

Stylus

```
border-radius()
  -webkit-border-radius: arguments
  -moz-border-radius: arguments
  border-radius: arguments

body a
  font: 12px/1.4 "Lucida Grande", Arial, sans-serif
  background: black
  color: #ccc

form input
  padding: 5px
  border: 1px solid
  border-radius: 5px
```

CSS

```
body a {
  font: 12px/1.4 "Lucida Grande", Arial, sans-serif;
  background: #000;
  color: #ccc;
}
form input {
  padding: 5px;
  border: 1px solid;
  -webkit-border-radius: 5px;
  -moz-border-radius: 5px;
  border-radius: 5px;
}
```

Figur 19 Stylus exempel

6. Slutsatser

I detta projekt så tittade jag på de två stora CSS-preprocessorerna Less och Sass. Efter en djup undersökning har jag kommit till slutsatsen att Sass är den bättre preprocessorn. Som jag tog upp i föregående kapitel har Sass flera egenskaper som Less har brist av. Det är också möjligt att ta upp argumentet att Sass var den som kom ut före och Less tog sina idéer från Sass.

Det finns flera hetsiga diskussioner online vilken är den bättre preprocessoren. Själv märkte jag att flera artiklar om Less var ganska defensiva och förklarade hur Less var bättre än Sass och hur bra och lätt det är att använda. Medan Sass sidor förklarade simpelt hur Sass fungerar och egenskaperna som den har.

Om du vill börja använda CSS-preprocessorer rekommenderas det att lära sig Sass, och om du har erfarenhet med CSS så är det inte svårt att lära sig Sass. Grundläggande idén bakom CSS-preprocessorer är att göra arbetet lättare, om det är svårt att använda preprocessorer vad är då poängen att använda det.

Om du är en Sass användare som vill byta till Less eller vice versa, så är det inga problem. Sass och Less har flera egenskaper de delar. Dina kunskaper från föregående preprocessorn kommer att hjälpa dig oerhört mycket.

Slutligen är det inte viktigt vilken CSS-preprocessor som du använder. Det viktiga är att du använder en. I stora projekt med flera värden som måste hållas reda på och besvärlig kod du måste skriva kan användningen av en preprocessor göra arbetet mycket lättare.

KÄLLOR

Saleh, Jango. 2015, Mätning av svarstider på grafiskt tunga webbutiker. *Högskolan i Skövde*. Vårtermin 2015. Tillgänglig: http://www.diva-portal.org/smash/get/diva2:935656/FULLTEXT01.pdf Hämtad: 7.4.2017

Lie, H. W. 2005, Cascading Style Sheets. Oslo Universitet. 29.3.2005. Tillgägnlig:https://www.wiumlie.no/2006/phd/css.pdf#%5B%7B%22num%22%3A294%2C%22gen%22%3A0%7D%2C%7B%22name%22%3A%22XYZ%22%7D%2C0%2C680%2C0%5D Hämtad: 11.4.2017

Lie, H. W. 1994, Cascading HTML Style Sheets. 10.10.1994. Tillgänglig: http://www.w3.org/People/howcome/p/cascade.html Hämtad: 11.4.2017

Wynn, Netherland; Nathan, Weizenbaum; Chris, Eppstein och Brandon, Mathis. 2013, Sass and Compass in Action 2013. Tillgänglig: https://manning-content.s3.amazonaws.com/download/3/deeb10c-27e1-44dc-8a97-e9c06ab62471/SaCiASampleCh01.pdf Hämtad: 18.4.2017

Sass (stylesheet language). Wikipedia 12.4.2017. Tillgänglig: https://en.wikipedia.org/wiki/Sass_(stylesheet_language) Hämtad: 18.4.2017

Sass (Syntactically Awesome StyleSheets).28.3.2016 Tillgänglig: http://sass-lang.com/documentation/file.SASS_REFERENCE.html Hämtad: 18.4.2017

Giraudel, Hugo 2017. Sass Guidlines 2017.Tillgänglig: https://sass-guidelin.es/ Hämtad: 19.4.2017

Sass and Browser Compatibility 2017. Tillgänglig: http://sass.logdown.com/ Hämtad: 21.4.2017

Less (stylesheet language). Wikipedia 30.3.2017. Tillgänglig: https://en.wikipedia.org/wiki/Less_(stylesheet_language) Hämtad: 24.4.2017

Less 2017. Less hemsida 4.1.2017. Tillgänglig: http://lesscss.org/ Hämtad: 24.4.2017

Fadeyev, Dmitry 2010. How To Use The LESS CSS Preprocessor For Smarter Style Sheets 6.12.2010 Tillgänglig: https://www.smashingmagazine.com/2010/12/using-the-less-css-preprocessor-for-smarter-style-sheets/ Hämtad: 24.4.2017

Thoriq, Firdaus 2016. LESS CSS – Beginner's Guide 2016. Tillgänglig: http://www.hongkiat.com/blog/less-basic/ Hämtad: 25.4.2017

Hixon, Jeremy 2011. An Introduction To LESS, And LESS Vs Sass 9.9.2011 Tillgänglig: https://www.smashingmagazine.com/2011/09/an-introduction-to-less-and-comparison-to-sass/ Hämtad: 27.4.2017

Coyier, Chris 2012. Sass vs. LESS 16.5.2012 Tillgänglig: https://css-tricks.com/sass-vs-less/ Hämtad: 27.4.2017

Stylus 2017. Stylus expressive, dynamic, robust CSS Tillgänglig: http://stylus-lang.com/ Hämtad: 28.4.2017