UNIVERZITA PALACKÉHO V OLOMOUCI

PEDAGOGICKÁ FAKULTA

Katedra technické a informační výchovy

Bakalářská práce

Eliška Plitzová

CSS PREPROCESORY – JEJICH SROVNÁNÍ
A DOPORUČENÍ PRO VÝBĚR

Olomouc 2017 vedoucí práce: Mgr. Jan Kubrický, Ph.D.

Prohlášení

Prohlašuji, že tato práce je mým původním autorským dílem, které jsem vypracovala samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používala nebo z nich čerpala, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

V Olomouci dne 17. 4. 2017 …….……

Poděkování

Děkuji Mgr. Janu Kubrickému, Ph.D. za odborné vedení bakalářské práce a poskytování cenných rad. Poděkování zároveň patří Mgr. Markétě Hávové za pomoc při gramatické kontrole práce.

Obsah

Úvod

Vymezení tématu práce a důvod výběru tématu

Tématem této bakalářské práce je srovnání nejpoužívanějších[1] CSS preprocesorů, jimiž jsou LESS, Sass a Stylus a doporučení pro jejich výběr. CSS preprocesory patří mezi mírně pokročilou znalostní výbavu kodérů a programátorů. Jejich používání je spojené s tvorbou moderních webových aplikací, jež jsou typické svou složitou strukturou a náročností na přizpůsobivost designu a obsahu rozměrům obrazovek zařízení. Jejich úkolem je také poskytnout uživateli kratší a přehlednější zdrojový kód a jeho snadnější a rychlejší zápis. Toho dosahují integrováním stylu zápisu kódu, který se blíží objektově orientovanému (založeného na myšlence objektů a jejich vzájemných interakcích), deklarativnímu (stojícího na myšlence tvorby aplikací pomocí definic) či imperativnímu (popisujícího úkony pomocí algoritmů) stylu programování dle zvoleného preprocesoru. Využívají zejména zanořování, dědičnosti, proměnných a také matematických funkcí. Právě vzhledem k těmto vlastnostem se preprocesory liší od klasického zápisu CSS kódu.

Důvodem výběru tohoto tématu bakalářské práce je jeho vztah k mé zálibě v tvorbě moderních webových prezentací a hledání řešení pro aktuální trendy ve web designu. Druhým důvodem je také zdokonalení vlastní práce při tvorbě webových prezentací a dosažení tak lepších výsledků za menší časový úsek. Znalost CSS preprocesorů považuji také za velké plus při budoucím výběru zaměstnání, jelikož znalost aktuálních trendů a technologií všeobecně v jakékoliv oblasti týkající se internetu bude v budoucnu hrát stále významnější roli pro udržení se v těchto oblastech na vrcholu.

[1] What are the best CSS preprocessors/postprocessors? Slant [online]. [cit. 2017-03-12]. Dostupné z: https://www.slant.co/topics/217/~best-css-preprocessors-postprocessors

Cíle práce

Cílem této bakalářské práce je porovnat nejrozšířenější CSS preprocesory LESS, Sass a Stylus, demonstrovat jejich rozdílné způsoby zápisu kódu, poukázat na jejich benefity, které nabízí oproti čistému CSS a které se často liší i v případě jednotlivých preprocesorů. Účelem práce je také objasnit, proč je vhodné se CSS preprocesory zabývat a používat je a ozřejmit jejich vliv na kvalitu kódu a rychlost vývoje, který mohou při správném zařazení do pracovních postupů mít. Závěr práce nastiňuje doporučení, čím se při výběru vhodného preprocesoru řídit a jakým způsobem k tomuto výběru přistupovat.

Výběr preprocesoru bývá brán jako složitý proces, jelikož neexistuje jasné rozdělení, který preprocesor je pro koho vhodný, zda je nutné znát veškerou syntaxi preprocesoru či pouze znát jejich principy. Závěr práce proto nastiňuje doporučení, čím se při výběru vhodného preprocesoru řídit a jakým způsobem k tomuto výběru přistupovat.

Metody dosažení cíle

V první teoreticko-metodologické části této práce se budu nejdříve věnovat HTML, jež je pro CSS nezbytný, jeho historickému vývoji a následně CSS jako takovým. V následující části se zaměřím na rozbor jednotlivých preprocesorů, co se jejich vlastností, funkcí a odlišností v zápisu kódu týče.

U CSS preprocesorů nelze určit, který je jednoznačně lepší, jelikož záleží na preferencích uživatele a na charakteru webové stránky. V práci se proto budu zabývat zhodnocením preprocesorů z kvalitativního hlediska s přihlédnutím k zjištěným teoretickým poznatkům z prostudované literatury a internetových zdrojů.

Použité prameny

Jelikož jsem k tomuto tématu nenašla žádnou publikaci v českém jazyce, vychází práce především z dokumentací jednotlivých preprocesorů, jež jsou dostupné na jejich oficiálních stránkách. Získané poznatky jsou dále podloženy anglicky psanou

literaturou, které je na toto téma více. Tématem tvorby moderních webových aplikací se zabývá například nakladatelství A Book Apart, avšak preproceserům se věnuje pouze okrajově. I z těchto důvodů bylo nutné dále čerpat především z internetových zdrojů a také v tomto případě bylo nutno se zaměřit opět na zahraniční zdroje, jelikož se v České republice problematikou CSS preprocesorů primárně nikdo nezabývá. Za zmínku stojí ale internetový portál Vzhůru dolů[2], jehož autorem je Martin Michálek, který už řadu let píše o tématech, ve kterých se věnuje moderním webovým postupům a technologiím CSS a HTML, které problematiku CSS preprocesorů zahrnují.

Základní teoretický aparát, jež se týká značkovacího jazyka HTML a kaskádových stylů, jsem čerpala převážně z knih HTML: tvorba dokonalých WWW[3] a CSS: chybějící manuál[4] a z oficiálních dokumentací jazyka HTML, které jsou dostupné na portále W3C[5].

Při hledání publikovaných zdrojů z řad studentů jsem našla pouze jedinou bakalářskou práci[6], která se CSS preprocesory zabývá. Autor však v závěru této práce shrnuje, že pro začátečníky v oblasti preprocesorů je vhodný preprocesor LESS, aniž by se v praktické části zabýval tvorbou CSS kódu pomocí zbylých zkoumaných preprocesorů. Takový závěr může být pro čtenáře zavádějící. Vezmeme-li v potaz, že rozhodnutí naučit se syntaxi některého z preprocesorů mohou učinit i programátoři, kteří již disponují elementárními znalostmi funkcí, cyklů apod. vlastností z programovacích jazyků, můžeme dojít k závěru, že pro takové uživatele může být vhodnější začít používat preprocesor jiný.

[2] *Vzhůru dolů* [online]. [cit. 2017-04-11]. Dostupné z: http://www.vzhurudolu.cz/

[3] KOSEK, Jiří. *HTML: tvorba dokonalých WWW stránek : podrobný průvodce* [online]. Praha: Grada, 1998 [cit. 2016-11-24]. Průvodce (Grada). ISBN 80-716-9608-0. Dostupné z: http://www.kosek.cz/html/html-tvorba-dokonalych-www-stranek.pdf

[4] MCFARLAND, David Sawyer. *CSS: chybějící manuál : tvorba nádherných webových stránek prostřednictvím CSS*. Praha: Grada, c2007. ISBN 978-80-247-2122-4.

[5] *W3C* [online]. 2016 [cit. 2016-11-24]. Dostupné z: https://www.w3.org/

[6] JINDA, Martin. *CSS preprocesory*. České Budějovice, 2014. Bakalářská práce. Jihočeská univerzita v Českých Budějovicích.

1 Teoretická východiska

Pojem, který bude v rámci práce hojně používán a jehož vymezení je zcela esenciální, je uživatel. Všeobecně je termínem *uživatel* v kontextu této práce označován člověk, jež využívá syntaxe některého z jazyků k tvorbě zdrojového kódu. Takového uživatele můžeme dále specifikovat:

- Programátor nebo softwarový inženýr, kterým je člověk, jež se zabývá tvorbou aplikací pomocí příslušného programovacího jazyka. Podle návrhu a připomínek také spolupracuje na vytváření uživatelského rozhraní pro tyto aplikace.
- Kodér, jímž je v českém jazyce označován člověk, který vytváří webové prezentace nebo webové aplikace. Doménou kodéra je především CSS, HTML a související technologie[7]. V anglickém jazyce je pojem „*coder*“ naopak spojen s kýmkoliv, kdo vytváří kód[8] a může se jednat tedy i např. o programátora.

Pojmem uživatel je v rámci této práce myšlena osoba kodéra. Budeme-li mluvit o uživateli programátorovi nebo softwarovém inženýrovi, budeme jej označovat souhrnně jako programátor.

1.1 HTML

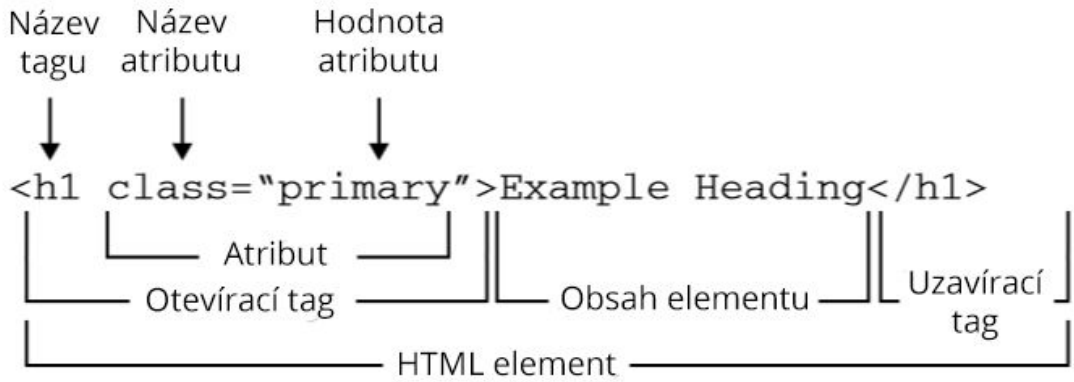
Hyper Text Markup Language (zkráceně HTML) je značkovací jazyk pro tvorbu webových stránek v informačním prostoru World Wide Web (zkráceně WWW), který umožňuje publikaci dokumentů na internetu. HTML využívá tzv. značek (anglicky *tags*) pro popsání struktury webové stránky. Elementy stránky, jako například nadpis, tabulka apod., jsou těmito značkami reprezentovány a na jejich základě je prohlížeč schopný sestavit odpovídající strukturu a obsah stránky.

Standardy HTML se zabývá mezinárodní konsorcium World Wide Web Consortium (W3C), jehož členové vyvíjejí webové standardy pro WWW, technické specifikace a pokyny pro procesy návrhu webu s cílem maximalizovat jejich potenciál

[7] MICHÁLEK, Martin. Co by měl umět webový kodér? In: *Vzhůru dolů* [online]. [cit. 2016-11-24]. Dostupné z: http://www.vzhurudolu.cz/prirucka/webovy-koder
[8] Coder. *Dictionary.com* [online]. [cit. 2016-11-24]. Dostupné z: http://www.dictionary.com/browse/coder

vývoje pro zajištění dlouhodobého růstu. Webové standardy jsou souhrnem doporučení webových technologií pro tvorbu webu.

Struktura HTML elementu se skládá z *tagů* a vlastního obsahu. *Tag* se může dále skládat z jednoho nebo více atributů a jejich hodnot. Počet *tagů* v elementu závisí na tom, zda se jedná o párový či nepárový.



Obrázek č. 1: Struktura HTML elementu[9]

První definice jazyka HTML byla vytvořena v roce 1991. Šlo o verzi s označením 0.9 a jejím autorem byl Tim Berners-Lee. Tato verze umožňovala text rozčlenit do několika logických úrovní, použít několik druhů zvýraznění textu a zařadit do textu odkazy a obrázky.

Požadavky uživatelů na WWW vzrůstaly, a tak producenti různých prohlížečů obohacovali HTML o nové prvky[10]. Velký skok vpřed zapříčinila až v roce 1997 verze 4.0, která kromě multimediálních objektů, rámů, rozšíření tabulek a formulářů přinesla podporu kaskádových stylů a skriptů. Dokument pro HTML 4.0 se stal standardem a oficiálním doporučením W3C pro tvorbu webu.

Po vydání HTML verze 4.01, která obsahovala pouze opravy chyb, se W3C rozhodlo o zastavení vývoje HTML a začalo pracovat na jazyku XHTML reformulací HTML 4.0, jehož zápis je na bázi značkovacího jazyka pro výměnu dat XML. Důvodem byla snaha snížit počet *tagů* a vynutit psaní syntakticky korektního kódu. *Tagy* v XHTML musí být na rozdíl od HTML psány malými písmeny, musí být v páru,

[9] MALHOTRA, Avinash. *Tech Altum Tutorial* [online]. [cit. 15.4.2017]. Dostupný na WWW: http://tutorial.techaltum.com/htmlTags.html v originálním anglickém znění.
[10] Historie a vývoj HTML. *Kosek.cz* [online]. [cit. 2016-11-24]. Dostupné z: http://www.kosek.cz/vyuka/4iz228/prednasky/uvod/foilgroup03.html

atributy nesmí být zapsány zkráceným zápisem a jejich hodnoty musí být uvedeny v uvozovkách.

HTML 4.01 i XHTML mimo jiné disponují třemi druhy zápisu syntaxe, které se označují jako Strict, Transitional a Frameset. Striktnější syntaxe byla motivována možností psát jednodušší parsery (programy nebo scripty, které prochází text nebo kód a třídí ho podle předem jasně definovaných pravidel), použitelné na málo výkonných zařízeních[11] (např. mobilní zařízení v dané době) a nepovoluje žádné *tagy* pro formátování. Syntaxe Frameset slouží pro případ použití rámů, které umožňují rozdělení okna prohlížeče na více částí[12]. XHTML bohužel nepřišlo s žádnou novou funkcionalitou a proto se příliš neujalo.

Poslední verzí HTML je HTML5, která je od roku 2014 oficiálním doporučením W3C a přichází s četnými inovacemi. Mezi ně patří například nová sémantika, změna a rozšíření elementů pro lepší strukturu, multimediální obsah a vektorová grafika společně s novými datovými typy a atributy. Používání HTML značek, které ovlivňují grafickou podobu stránek, je již minulostí[13]. Dnes se HTML používá pouze pro doplnění obsahu a vytvoření logické struktury. Chceme-li z kaskádových stylů vytěžit maximum, je nutné mít solidní HTML základ. O zbytek se postarají kaskádové styly. Jednoduchý a strukturovaný kód je také lepší pro vyhledávače.

Jak je z historických změn patrné, vzhledem k různým verzím HTML musíme brát na zřetel jistá omezení, která nás mohou při tvorbě dokumentu svazovat. Setkáme-li se s dokumentem, jež je napsán ve starší verzi jazyka, je zapotřebí se vyvarovat používání *tagů*, které nejsou v dané verzi jazyka formulovány a v případě XHTML dbát na jejich správný zápis, co se sémantiky a velikosti písmen týče. V případě HTML5 se dokonce nabízí využití nových sémantických *tagů*, které jsou typické jen pro tuto verzi. Protože k vývoji jazyka může docházet i jeho obohacováním ze strany prohlížečů, je vždy nutné zkontrolovat aktuální podporu konkrétního *tagu* či funkcionality v dalších prohlížečích včetně jejich verzí, jež mohou návštěvníci

[11] KOSEK, Jiří. *HTML: tvorba dokonalých WWW stránek : podrobný průvodce* [online]. Praha: Grada, 1998 [cit. 2016-11-24]. Průvodce (Grada). ISBN 80-716-9608-0. Dostupné z: http://www.kosek.cz/html/html-tvorba-dokonalych-www-stranek.pdf

[12] Historie a vývoj HTML. *Kosek.cz* [online]. [cit. 2016-11-24]. Dostupné z: http://www.kosek.cz/vyuka/4iz228/prednasky/uvod/foilgroup03.html

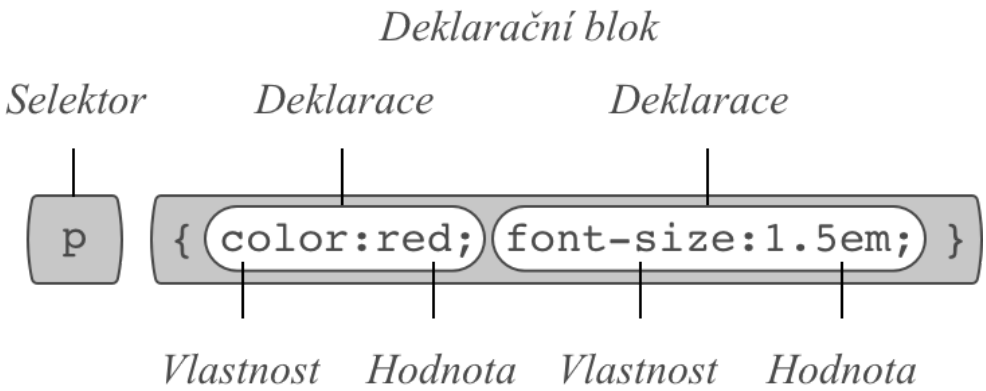
[13] MCFARLAND, David Sawyer. *CSS: chybějící manuál : tvorba nádherných webových stránek prostřednictvím CSS*. Praha: Grada, c2007. ISBN 978-80-247-2122-4.

takových budoucích webových stránek používat. Dbát by se také mělo na logickou strukturu použitých *tagů* a vybírat takové, které nezasahují do grafické stránky věci.

1.2 CSS

V předchozí části jsme se seznámili s prerekvizitou kaskádových stylů, jímž je HTML a nyní se dostáváme k samotným kaskádovým stylům. Kaskádové styly (CSS; cascade style sheets) fungují na systému pravidel, pomocí kterých vybírají z dokumentu HTML elementy a přidávají jim styl nebo mění jejich hodnoty. Jedná se tedy o deklarativní styl zápisu kódu.

Styl je tvořen dvěma prvky: selektorem, který reprezentuje jeden nebo více elementů webové stránky a deklaračním blokem, který obsahuje vlastní formátovací instrukce (deklarace), které využívají prohlížeče k naformátování selektoru. Každá deklarace má dvě části, vlastnost a hodnotu. Vlastnosti jsou formátovací volby, které představují nějaký efekt.



Obrázek č. 2: Struktura CSS stylu[14]

Výběr jednoho nebo více elementů lze uskutečnit pomocí jednoduchého selektoru na základě jména značky, třídy nebo identifikátoru. Druhou možností je výběr pomocí selektoru s požadovaným atributem a jeho hodnotou nebo selektoru s kombinátorem. Kombinátor je znak, který se zapisuje mezi elementy a slouží pro výběr sousedního (+) nebo obecného (~) sourozence, potomka (*mezera*) nebo přímého potomka (>). Pro výběr různých stavů elementů, kdy uživatel například přejede myší přes element nebo

[14] MCFARLAND, David Sawyer. *CSS: chybějící manuál : tvorba nádherných webových stránek prostřednictvím CSS*. Praha: Grada, c2007. ISBN 978-80-247-2122-4

klikne na odkaz, slouží pseudo-elementy. Název stavu se uvádí za dvojtečku za název elementu.

Nápad na vznik kaskádových stylů se datuje k roku 1990, kdy docházelo ze strany prohlížečů a jejich uživatelů k tlaku na možnost formátování obsahu[15]. CSS1 bylo konsorciem W3C přijato koncem roku 1996 jako oficiální doporučení. Jazyk umožňoval změnu fontů a vlastností dalších elementů jako jsou například velikost, zarovnání, odsazení, pozicování, změna barev a rozlišení elementů podle třídy nebo identifikátoru.

Nástup CSS2 přišel společně s jazyky XML a XHTML a to v roce 1999. Mezi nové funkce patřilo další rozšíření formátování textů, vizuální efekty, nové vlastnosti pro práci s velikostmi elementů, tabulky a přehrávání zvuků. Nejdůležitější funkcí byla ale podmínka, díky které bylo možné tvořit více stylů v závislosti na použitém zobrazovacím zařízení.

První návrhy CSS3 přišly už ve stejném roce jako CSS2. Jazyk byl nově rozdělen do *modulů*, ke kterým jednotlivě náleží vlastní dokumenty se specifikací. Řada *modulů* již byla schválena a jsou oficiálním doporučením W3C. Stále ale existuje plno *modulů*, které jsou označeny jako „*working draft*“ (fungující pracovní návrhy) a čekají na schválení. CSS3 už nyní umožňuje převratné funkce ve světě internetových aplikací. Patří mezi ně například nové textové efekty, vektorová grafika, 2D a 3D transformace, animace, přechody nebo media queries, které poprvé nabízejí možnost, jak přizpůsobit webovou prezentaci pro různá zařízení bez nutnosti změny samotného obsahu.

1.3 Propojení HTML a CSS

Pro uvedení stylů do funkčnosti je nutné jejich propojení a to ze strany HTML, přičemž existují dva možné způsoby:

1. **Interní CSS**: Styly jsou součástí HTML dokumentu. Nachází se buď mezi značkami `<style>` v hlavičce stránek nebo přímo v atributu *style* u konkrétního elementu. HTML dokument může také obsahovat nebo se odkazovat na soubor s JavaScriptovým kódem, který ovlivňuje chování a obsah webových stránek a může podobně jako HTML soubor obsahovat bloky stylů.

[15] The CSS saga. *W3C* [online]. [cit. 2017-03-12]. Dostupné z: https://www.w3.org/Style/LieBos2e/history/

2. **Externí CSS**: HTML dokument se odkazuje ze své hlavičky na zdrojový *.css* soubor. Tento způsob propojení je nejběžnější a poskytuje nespočet výhod[16]. Mezi ně patří mimo jiné i možnost využití CSS preprocesorů.

[16] The web standards model - HTML CSS and JavaScript. *W3C* [online]. [cit. 2017-03-12]. Dostupné z: https://www.w3.org/community/webed/wiki/The_web_standards_model_-_HTML_CSS_and_JavaScript

2 Charakteristika a základy syntaxe preprocesorů

Kaskádové styly se staly standardem pro popis vizuální stránky hypertextových dokumentů a jejich koncepce deklarativního popisu s sebou nese i některé nevýhody, jejichž řešením pomocí CSS preprocesorů se budeme věnovat v následujících kapitolách. Tyto slabiny v případě používání čistého CSS daly vzniknout právě CSS preprocesorům.

CSS preprocesor je open-source jazyk, který ze zdrojového kódu, zapsaného ve vlastní syntaxi daného preprocesoru, vygeneruje klasické validní CSS, které jsou na rozdíl od kódu pro CSS preprocesor čitelné prohlížeči. Jedná se tedy o kompilované jazyky. CSS preprocesory jsou napsané v jednom z programovacích jazyků – nejčastěji se jedná o JavaScript, PHP, Ruby nebo Python – a obohacují klasické CSS o nové jazykové vlastnosti, známé z těchto programovacích jazyků, mezi které patří například proměnné, zanořování, operátory nebo funkce či řeší jeho slabiny díky imperativnímu charakteru zmíněných jazyků. Souhrnně se tato vylepšení dají označit jako „*syntaktický cukr*“[17], jelikož nepřinášejí nové funkce pro popis vzhledu, nýbrž jen zjednodušují a zpřehledňují samotné CSS. CSS preprocesory spadající také do kategorie tzv. dynamických CSS (DCSS). Jedná se soubor technologií pro vytváření dynamických stylů pomocí serverových programovacích jazyků jako například PHP, ASP nebo Perl a kaskádových stylů (CSS).

V současné době existuje mnoho CSS preprocesorů, které nabízejí velmi podobné funkce s jinou syntaxí. Mezi tři nejpoužívanější CSS preprocesory patří podle portálu Slant[18] Sass, Stylus a LESS. Počet uživatelů těchto tří preprocesorů několikanásobně převyšuje počty u ostatních preprocesorů, mezi které patří například méně známý PostCSS, cssnext nebo Rework. Online průzkum s více než 13 000

[17] LESS: stejné CSS za méně peněz. *Zdroják.cz* [online]. [cit. 2017-03-12]. Dostupné z: https://www.zdrojak.cz/clanky/less-stejne-css-za-mene-penez/

[18] What are the best CSS preprocessors/postprocessors? Slant [online]. [cit. 2017-03-12]. Dostupné z: https://www.slant.co/topics/217/~best-css-preprocessors-postprocessors

odpověďmi od webových vývojářů ukázal, že přibližně 54 % webových vývojářů používá při vývoji CSS preprocesor[19].

LESS (*Leaner Cascading Style Sheets*) je open-source knihovna napsaná původně v jazyce Ruby a později přepsaná do jazyka JavaScript. LESS běží jak na serverové straně, pomocí knihoven Node.js a Rhino, tak i na straně klientské v prohlížeči. Pro kompilaci kódu lze využít nástroje třetích stran, jako například on-line editor less2css.org či doplňky do samotných editorů kódu.

Oficiální implementace Sass (*Syntactically Awesome Style Sheets*) je napsána v jazyce Ruby. Oproti LESS, který je napsán v JavaScriptu, je nutné jazyk Ruby nejprve nainstalovat. K dispozici jsou ale i implementace v jazycích PHP, C, .NET, Java, JavaScript a další (kompletní seznam jazyků je dostupný na adrese http://sass-lang.com/libsass). Sass také podporuje svou integraci do rozšíření Firefox Firebug (Firefox Firebug umožňuje editaci, debugging a monitorování CSS, HTML a JavaScriptu na jakékoliv stránce.). Stejně jako LESS může být Sass nainstalován na serverové i klientské straně. Pro kompilaci kódu do CSS lze opět využít aplikací třetích stran, jako například editor beautifytools.com/scss-compiler.php či doplňků přímo do editorů.

Třetí preprocesor Stylus je stejně jako LESS napsán v jazyce JavaScript. Jeho instalace tedy probíhá pomocí serverového frameworku Node.js a není proto zapotřebí žádná předchozí instalace programovacího jazyka. Stylus poskytuje všechny standardní schopnosti preprocesorů, ale i pokročilé funkce, jako například JavaScript API nebo integraci do Firefox Firebug.

Tato práce se blíže věnuje výše zmíněným třem nejvíce používaným CSS preprocesorům – LESS, Sass, Stylus. V následujících kapitolách se zaměříme na jejich syntaxi a základní společné vlastnosti, kterými jsou proměnné a zanořování.

2.1 Syntaxe

Jedním z problémů při výběru CSS preprocesorů může být neznalost syntaxe daného preprocesoru, nutnost učit se nový zápis a obavy s tímto spojené. Podobně jako je tomu u programovacích jazyků, i procesory se mezi sebou liší jinou syntaxí. Rozdíly nejsou

[19] Poll Results: Popularity of CSS Preprocessors. *CSS-Tricks* [online]. 2012 [cit. 2017-04-15]. Dostupné z: https://css-tricks.com/poll-results-popularity-of-css-preprocessors/

však natolik markantní. Všechny preprocesory navíc podporují klasický zápis CSS kódu. V případě LESS je tato syntaxe dokonce jedinou dostupnou. Příčinou je filozofie tohoto preprocesoru, která je pouhým rozšířením již zaběhlého CSS jazyka[20]. Přechod na LESS je tedy cestou nejmenšího odporu, co se učení nové syntaxe týče.

Co se týče syntaxe v preprocesoru Sass, nabízejí se následující dva způsoby zápisu kódu:

- Původní syntaxí s názvem „*indent*“ (zjednodušeně nazývaná jako „*Sass*“ nebo „*whitespace*“) – Koncovkou souboru je *.sass*. Syntaxe využívá odsazení pro rozlišení bloků a nových řádků pro rozlišení podmínek. Tato stručnější syntaxe není plně kompatibilní s CSS syntaxí a její výhodou je rychlejší zápis kódu.
- Novější „*SCSS*“ (*Sassy CSS*) syntaxí – Koncovkou souboru je .scss. Tato syntaxe využívá stejně jako CSS složených závorek pro formátování bloků kódu a středníků pro rozlišení nových řádků. Veškeré kódy preprocesoru Sass budou psány pomocí této syntaxe.

Obě syntaxe mají stejné vlastnosti a liší se pouze odlišným způsobem zápisu kódu. Syntaxe lze mezi sebou libovolně převádět pomocí příkazu `sass-convert` v příkazové řádce nebo pomocí on-line editorů (např. http://www.sassmeister.com/).

Stylus nabízí dokonce tři druhy syntaxe a to klasickou CSS syntaxi (tzn. se složenými závorkami, dvojtečkami a středníky), bez složených závorek a bez středníků i dvojteček. U posledních dvou syntaxí je zapotřebí v případě zanořování stylů striktně dodržovat odsazení od začátku řádku pomocí mezer. Oproti jiným preprocesorům lze mezi těmito syntaxemi libovolně přecházet v rámci jednoho souboru a kombinovat je. Koncovka souboru má tedy vždy totožnou podobu a to *.styl*. Možnost kombinace více syntaxí v rámci jediného souboru je oproti předešlým dvěmapreprocesorům velkou výhodou. Dojde-li k neúmyslnému vypuštění dvojtečky nebo středníku na konci řádku, není nutné hledat příčinu a místo vzniku chyby. Stylus takový soubor bez problému zkompiluje do validního CSS stylu.

[20] Getting started. *Less.js* [online]. [cit. 2017-04-11]. Dostupné z: http://lesscss.org/

Klasický zápis stylu v CSS nebo v LESS:

```
body {
  font: 12px Helvetica, Arial, sans-serif;
}
```

Zápis bez složených závorek, dostupný v Sass a Stylusu:

```
body
  font: 12px Helvetica, Arial, sans-serif;
```

Zápis bez středníků a dvojteček, dostupný ve Stylusu:

```
body
  font 12px Helvetica, Arial, sans-serif
```

V syntaxi Stylusu lze také vypustit i čárky, které slouží pro oddělení elementů, tříd a identifikátorů.

Kód ve Stylusu:

```
textarea
input
  border 1px solid #eee
```

Výsledný CSS kód:

```
textarea,
input {
  border: 1px solid #eee;
}
```

Jak lze z příkladů vidět, v preprocesorech Sass a Stylus dochází, na rozdíl od LESS, již ke změně syntaxe a je proto nutné se řídit jinými pravidly pro zápis. Celkově ale nejde o natolik podstatné změny, které by mohly ovlivnit budoucí průběh tvorby stylů. V případě Stylusu lze mezi těmito syntaxemi v rámci jednoho dokumentu volně přecházet dle potřeby, což mohou uvítat týmy, kde se podílí na tvorbě stylů více tvůrců.

Pro kompilaci kódu ze syntaxe preprocesoru do CSS je nutná instalace potřebného programovacího jazyka preprocesoru, ve kterém byl napsán nebo lze využít nástrojů třetích stran, jako jsou online kompilátory nebo doplňky do samotných editorů

kódu. Kódy v syntaxích jednotlivých preprocesorů lze mezi sebou také převádět, a to například pomocí online převodníku, dostupného na portále CSS PREprocessors[21].

Souhrnně lze tedy říci, že přechod mezi jednotlivými preprocesory je vzhledem k existenci konvertorů nenáročným procesem, a proto není počáteční volba preprocesoru na základě jeho syntaxe natolik důležitou otázkou, kterou by bylo nutné se na začátku zbytečně zabývat. Důležitější otázkou je, zda a případné jaké funkce od preprocesoru požadujeme.

2.2 Proměnné

Podpora proměnných je jednou z nejzákladnější funkcí tradičních programovacích jazyků, která v CSS chybí. Proměnná je v programování označení identifikátoru, který zastupuje nějakou informaci neboli hodnotu proměnné. Ta se může za běhu programu měnit nebo nabývat jiného datového typu. Pomocí proměnné lze v CSS specifikovat často používanou hodnotu a odkazovat se na ni z různých částí kódu. Nejčastěji se proměnné používají pro ukládání barev, fontů nebo číselných hodnot dalších parametrů.

Změníme-li hodnotu proměnné, změna se projeví globálně v celém dokumentu, což vede ke snadnější správě dokumentu. Změna může probíhat i automaticky v návaznosti na nějaký stav (např. změna šířky okna). Proměnnou lze využít také i jako pomocnou schránku při výpočtech.

Po kompilaci souboru jsou proměnné zpracovány a jejich vrácené hodnoty jsou vloženy do výsledného CSS souboru na místo jejich volaných názvů. Využití proměnných je užitečné zejména při práci s opakujícími se hodnotami.

Syntaxe zápisu proměnné v LESS začíná znakem @, v Sass a Stylusu znakem $, jehož zápis je v případě Stylusu volitelný. Následuje název proměnné, který se volí vždy takový, aby korespondoval s jejím obsahem, dvojtečka nebo v případě Stylusu znak rovnítka a hodnota proměnné, ukončená středníkem.

[21] *CSS PREprocessors* [online]. [cit. 2017-04-11]. Dostupné z: https://csspre.com/convert/

Kód v LESS:

```
@seaBlue: #428bca;
a:hover { color: @seaBlue; }
  footer { background-color:
@seaBlue; }
```

Kód ve Stylusu:

```
font-size = 14px

body
  font font-size Arial, sans-serif
```

Proměnnou lze definovat také uvnitř bloku stylů. Její rozsah je v takovém případě pouze pro tento blok a nelze ji tedy využít vně nebo v jiném bloku. Pokud chceme proměnnou využít i mimo tento blok, v Sass lze definovat zanořenou proměnnou jako globální, přidáním `!global` za její hodnotu.

Kód v Sass:

```
a:hover {
  $seaBlue: #428bca !global;
  color: $seaBlue;
}
footer {
  background-color: $seaBlue;
}
```

Výsledný CSS kód:

```
a:hover {
  color: #428bca;
}
footer {
  background-color: #428bca;
}
```

Další zajímavou funkcí v preprocesoru Stylus je možnost definovat proměnnou až v místě jejího prvního použití bez nutnosti její prvotní definice. Tímto se snižuje počet řádků v závislosti na počtu proměnných. V případě takového zápisu má proměnná rozsah pouze v bloku stylů, kde byla definována a nelze ji použít vně.

Kód ve Stylusu:

```
#logo
   width: w = 150px
   height: h = 80px
   margin-left: -(w / 2)
   margin-top: -(h / 2)
```

Výsledný CSS kód:

```
#logo {
  width: 150px;
  height: 80px;
  margin-left: -75px;
  margin-top: -40px;
}
```

Tento zápis lze navíc zjednodušit vynecháním proměnných. Ve Stylusu se totiž lze odkazovat přímo na hodnoty stylů pomocí znaku @. Kód z následujícího příkladu bude vyhodnocen totožně.

Kód ve Stylusu:

```
#logo
   width: 150px
   height: 80px
   margin-left: -(@width / 2)
   margin-top: -(@height / 2)
```

Z uvedených příkladů je zřetelné, že Sass a Stylus nabízí více možností v práci s proměnnými, které LESS neumožňuje. V případě Sass lze rozsah proměnné definovat uvnitř bloku a učinit tak proměnnou záměrně nedostupnou z jiných částí kódu nebo ji naopak rozšířit na globální. Stylus toto neumožňuje, ale je naopak vybaven více způsoby zápisu proměnných a je dokonce možné vypustit prvotní definice. Styl se tak nezanáší pomocným kódem, který ve skutečnosti nepopisuje to, jak má dokument vypadat.

2.3 Zanořování

Problémem při tvorbě responzivního designu je různorodost rozlišení zařízení, kde šířka jednoho zařízení orientovaného na výšku může zároveň odpovídat šířce displeje staršího zařízení orientovaného na šířku. Pomocí detekce rozlišení v media queries tak nelze detekovat hraniční bod zařízení se stejnou fyzickou velikostí a s totožným stylem používání. Při hledání breakpointů (hraničních rozměrů, při kterých vypadá rozložení webu stejně) navíc dochází k rozpadnutí designu jen v případě určitých elementů. Dává proto logiku využívat jako breakpointy přímo konkrétní elementy nebo části kódu

(hlavička, navigace apod.), než globálně pevně dané rozmezí pro všechny prvky[22]. Aktuální implementace CSS však umožňuje zanořovat elementy pouze do media queries, ale nikoliv naopak.

V CSS preprocesorech je zanořování společnou vlastností, které se používá namísto nebo v kombinaci s kaskádovým zápisem (tedy hierarchickým výčtem elementů za sebou). Na základě toho je zanořený kód více konzistentní a je podobný HTML struktuře – napodobuje strukturu tagů a zlepšuje vizuální hierarchii. Zanořování také umožňuje psát kód, který se neopakuje a zjednodušuje zápis dlouhých posloupností selektorů, přičemž každému z nich odpovídá jedna úroveň zanoření. Čím větší je hloubka zanořeného selektoru, tím se jedná o více specifikovaný styl.

Zanořovat lze i pseudo-třídy a pseudo-elementy a jejich zápis uvnitř bloku se provádí pomocí znaku & před dvojtečkou a názvem, a to v případě LESS, Sass i Stylusu.

Kód v LESS:

```
header {
  background-color: white;
  a {
    color: gray;
    &:hover {
      color: black;
    }
  }
}
```

Výsledný CSS kód:

```
header {
  background-color: white;
}

header a {
  color: gray;
}

header a:hover {
  color: black;
}
```

[22] Jaké breakpointy zvolit v responzivním webdesignu? *Vzhůru dolů* [online]. 2013 [cit. 2017-04-11]. Dostupné z: http://kratce.vzhurudolu.cz/post/46416507703/jak%C3%A9-breakpointy-zvolit-v-responzivn%C3%ADm-webdesignu

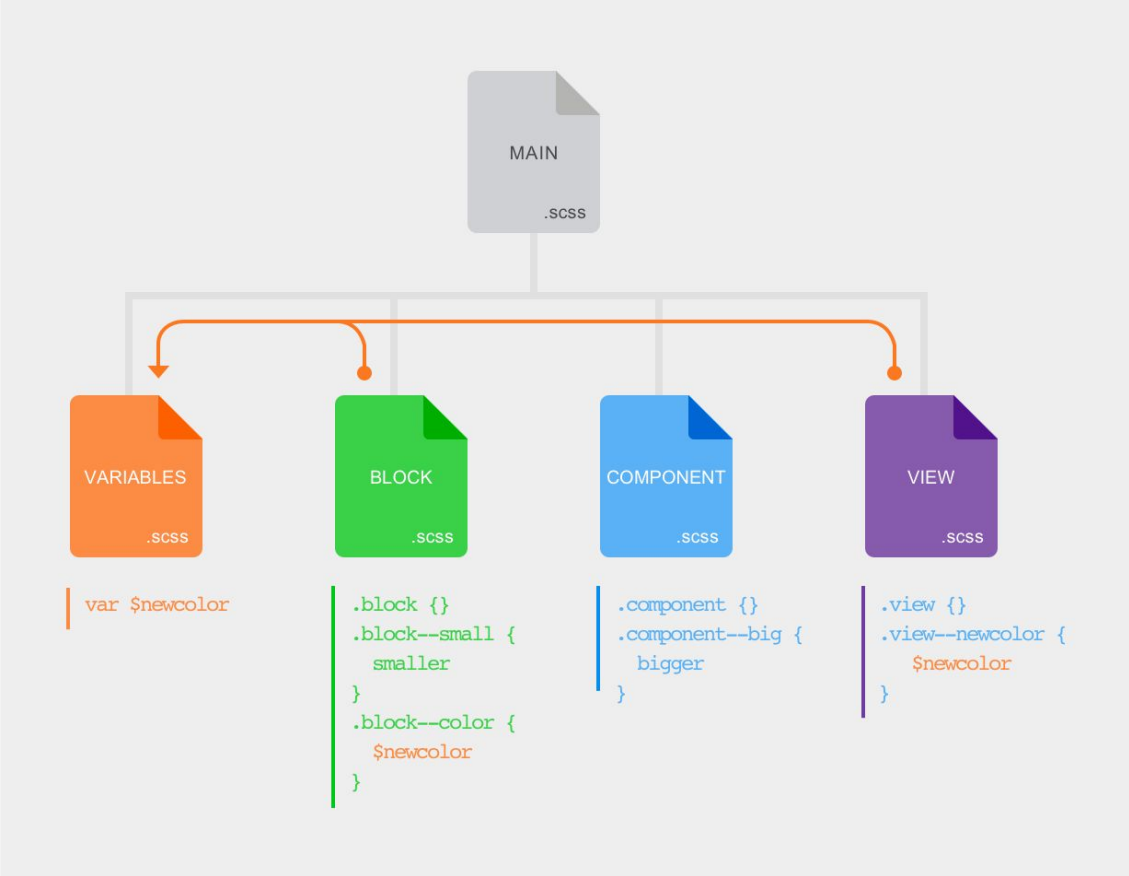
3 Modularizace kódu

Myšlenkou modulárního CSS kódu je tvorba specifických stylů, které slouží pouze k jedinému účelu v rámci daného bloku. Požadovaného efektu se dosahuje mixováním těchto méně generických tříd. Cílem modulárního CSS je redukovat závislosti a zachovat konzistentní kód, který může být rozdělen do více logických souborů (například proměnné, bloky, komponenty).[23]

Shlukování kódu do bloků je vzhledem k možnosti zanořování základní a nejvíce využívanou vlastností CSS preprocesorů. Díky tomu lze vytvářet logické bloky kódu, které lze následně členit do více souborů. Zanořování a další vlastnosti preprocesorů mohou být ve spojení s jednou z metodologií modulárního CSS (OOCSS, SMACSS nebo BEM) šikovným způsobem, jak kód organizovat.

OOCSS (*objektově orientované CSS*), SMACSS (*scalable and modular architecture for CSS*, popsáno na https://smacss.com/) a BEM (*block element modifier*, popsáno na https://en.bem.info/) jsou metodologie, které usilují o rozdělení kódu do opětovně využitelných bloků. Jádrem těchto metodologií je oddělení obsahu a struktury od vzhledu. Příkladem využití těchto metodologií ve spojení s preprocesory a jejich vlastnostmi může být sada stylů pro patičku webu umístěných ve vlastním souboru, který může být opětovně použit u jiného projektu pouze jeho připojením.

[23] A Quick Guide To Modular CSS – Don't Be Petty, Write Less Spaghetti!. *The Raygun Blog* [online]. 2016 [cit. 2017-04-15]. Dostupné z: https://raygun.com/blog/modular-css/



Obrázek č. 3: Strukturovaný příklad modulárního CSS[24]

Další možností organizace kódu je pomocí mixinů do modulů – kusů kódu, které se vztahují ke konkrétní funkční komponentě (např. tedy patička, menu nebo galerie) a lze je použít kdekoliv jinde nezávisle na projektu.

3.1 Mixiny

Vzhledem k deklarativní povaze není v čistém CSS možné vytvářet funkce nebo jiné bloky kódu pro pozdější opakované použití. V preprocesorech existují pro tyto účely mixiny, které slouží k vytváření skupin CSS deklarací a jsou tak způsobem, jak znovu využít již jednou definované vlastnosti tříd nebo identifikátorů. Mixiny jsou jednou z nejsilnějších a nejdůležitějších vlastností Sass a jsou ekvivalentem maker v jiných programovacích jazycích[25].

[24] HENWOOD, Kyle. *The Raygun Blog* [online]. [cit. 15.4.2017]. Dostupný na WWW: https://raygun.com/blog/modular-css/

[25] Leveraging Sass mixins for cleaner code. *The Sass way* [online]. 2011 [cit. 2017-04-15]. Dostupné z: http://thesassway.com/intermediate/leveraging-sass-mixins-for-cleaner-code

Mixiny mohou přijímat jeden nebo více parametrů, případně žádný, a tudíž se dělí na parametrické a neparametrické. Využít lze i již naprogramovaných mixinů, které je možné získat v knihovně na stránkách oficiální dokumentace[26]. Knihovnám a parametrickým mixinům se budeme blíže věnovat v kapitole číslo 3.5 o dostupných rozšířeních a v kapitole 4 o programatické logice.

Pomocí neparametrických mixinů se můžeme odkazovat na sadu definic konkrétní třídy nebo identifikátoru a tyto definice opětovně využívat v jiných sadách definic. Správným používáním mixinů tedy rapidně klesá redundance (informační nebo funkční nadbytek) kódu a zvyšuje se jeho přehlednost a možnost správy.

V následujícím příkladě je kromě použití mixinů také demonstrována jejich syntaxe v LESS, která nevyžaduje žádnou předchozí definici, jako je tomu v případě proměnných, a jde tedy jen o zavolání požadovaného bloku stylů.

Kód v LESS:

```
.displayBlock {
  display: block;
  margin: 0 20px;
  padding: 20px;
}

.team-wrap {
  background-color: white;
  .displayBlock;
}

#menu { .displayBlock; }
```

Výsledný CSS kód:

```
.team-wrap {
  background-color: white;
  display: block;
  margin: 0 20 px;
  padding: 20px;
}

#menu {
  display: block;
  margin: 0 20 px;
  padding: 20px;
}
```

3.2 Extend

Extend slouží v preprocesorech podobně jako mixiny k rozšiřování o nové styly. Cílem obou příkazů je vytváření bloků se styly, které se často opakují a vyvarování se tak psaní stejného kódu. Logika způsobu jejich použití je ovšem rozdílná.

Rozdílem mezi extend a mixin je ten, že voláním mixinu dojde k vložení stylů na toto místo volání, kdežto v případě extend dojde k vytvoření nového samostatného

[26] Frameworks Using Less. *Less.js* [online]. [cit. 2017-03-12]. Dostupné z: http://lesscss.org/3.x/tools/#frameworks-using-less-mixin-libraries

bloku se styly pro všechny rozšiřované selektory. Problémem, který může při používání extend nastat, je rozšíření velkého množství nesouvisejících selektorů, což vede k vyústění v následující výsledek. Po kompilaci může být vytvořený rozšiřující blok v extrémních případech i stovky řádků daleko od posledního rozšiřovaného bloku. Kód je tak sice *DRY (Don't Repeat Yourself*; kód, který se neopakuje*)*, ale vzniká zde problém s přehledností a možností organizace kódu, vzhledem k početnému výčtu nesouvisejících selektorů a následnou velkou mezerou mezi hlavními bloky.

Kód v Sass:

```
%bold-font {
  font-weight: bold;
}

h1 {
  @extend %brand-font;
  font-size: 2em;
}
.btn {
  @extend %brand-font;
  display: inline-block;
  padding: 1em;
}
.promo {
  @extend %brand-font;
  background-color: #BADA55;
  color: #fff;
}
.footer-message {
  @extend %brand-font;
  font-size: 0.75em;
}
```

Výsledný CSS kód:

```
h1, .btn, .promo, .footer-message {
  font-weight: bold;
}

h1 {
  font-size: 2em;
}
.btn {
  display: inline-block;
  padding: 1em;
}
.promo {
  background-color: #BADA55;
  color: #fff;
}
.footer-message {
  font-size: 0.75em;
}
```

Správné použití extend je proto pouze mezi souvisejícími bloky[27]. Po kompilaci tak dojde k vygenerování bloků, které jsou mezi sebou neodmyslitelně spjaty a sdílí spolu své vlastnosti z nějakého důvodu, nikoliv náhodně. Je důležité si tedy uvědomit, že extend vytváří vztahy a při jeho použití dochází k přeskládání selektorů a ke sdílení jejich znaků s dalšími rozšiřovanými selektory.

[27] When to use @extend; when to use a mixin. CSS Wizardry [online]. 2014 [cit. 2017-04-11]. Dostupné z: https://csswizardry.com/2014/11/when-to-use-extend-when-to-use-a-mixin/

Kód v Sass:

```
.btn,
%btn {
    display: inline-block;
    padding: 1em;
}

.btn-positive {
    @extend %btn;
    background-color: green;
    color: white;
}

.btn-negative {
    @extend %btn;
    background-color: red;
    color: white;
}

.btn-neutral {
    @extend %btn;
    background-color: lightgray;
    color: black;
}
```

Výsledný CSS kód:

```
.btn,
.btn-positive,
.btn-negative,
.btn-neutral {
    display: inline-block;
    padding: 1em;
}

.btn-positive {
    background-color: green;
    color: white;
}

.btn-negative {
    background-color: red;
    color: white;
}

.btn-neutral {
    background-color: lightgray;
    color: black;
}
```

Ve Stylusu i v Sass je extend příkazem. Jeho syntaxe je proto uvedena znakem @ a je nutné příkaz zanořit. V případě LESS je extend pseudo-třídou a podléhá proto stejným pravidlům syntaxe. Příkaz lze tedy zapsat dvěma způsoby, a to pomocí znaku & v zanořeném bloku nebo pomocí dvojtečky s vlastním názvem v závorce za selektorem.

```
.seriousError {
  &:extend(.error);
  border-width: 3px;
}

.seriousError:extend(.error) {
  border-width: 3px;
}
```

Kromě možnosti zamezení opakovaného zápisu může extend přispívat také k lepší možnosti kontroly a udržování kódu. V uvedeném příkladu je blok stylů `.serousError` rozšířen o styly selektoru `.error`. Pokud bychom tedy chtěli u více elementů použít dvou odlišných bloků stylů, museli bychom každému elementu přiřadit obě třídy a mít tuto skutečnost stále na mysli pro případ opětovného použití. S použitím `@extend` toto nemusíme řešit. Chceme-li u elementu použít dva bloky stylů, lze jednoduše jeden

z bloků rozšířit pomocí `@extend` o styly bloku druhého a v HTML přiřadit elementu jen tuto jednu třídu či identifikátor. Sloučené styly se aplikují na všechny elementy, které spadají pod první třídu.

Příkaz `@extend` ve Stylusu navíc umožňuje rozšiřovat selektory o styly odkazováním se na zanořený styl. Následující kód ve Stylusu by v případě zápisu v Sass při kompilaci do CSS skončil chybou.

Kód ve Stylusu:

```
form
   button
     padding: 10px

 a.button
   @extend form button
```

Výsledný CSS kód:

```
form button,
a.button {
          padding: 10px;
}
```

3.2.1 Rozšiřování více selektorů

Tato vlastnost příkazu `@extend` ve Stylusu umožňuje rozšiřovat více selektorů naráz. Praktické využití je čistě pro zkrácení zápisů a ušetření místa. Syntaxe a příklad použití je následující:

Kód ve Stylusu:

```
.a
  color: red

.b
  width: 100px

.c
  @extend .a, .b
  height: 200px
```

Výsledný CSS kód:

```
.a,
.c {
  color: #f00;
}
.b,
.c {
  width: 100px;
}
.c {
  height: 200px;
}
```

3.2.2 Volitelný extend

Při použití klasického `@extend` pro selektor, který nebyl dříve v dokumentu deklarován, by kompilace souboru do CSS skončila chybou. Stejně tak by skončila chybou

i v případě, pokud by byl selektor součástí delšího výčtu (např. `h1.notice`). Pro případ, kdy není jisté, zda byly styly pro samotný selektor deklarovány, lze využít tzv. volitelného `@extend`, který se zapisuje pomocí značky `!optional` za stávající syntaxi. Volitelný `@extend` je dostupný v preprocesorech Sass a Stylus.

Kód v Sass:

```
a.important {
  @extend .notice !optional;
}
```

Kód ve Stylusu:

```
$style2
  color: #fff

.btn
  @extend .style1 !optional, $style2
```

Výsledný CSS kód:

```
.btn {
  color: #fff;
}
```

3.2.3 Placeholder

Placeholdery slouží pro označení bloku stylů, u kterého je žádané, aby byl zkompilován pouze na těch místech, kde bude začleněn pomocí příkazu `@extend`. Sám o sobě tedy blok stylů není zkompilován a ve výsledném CSS se neobjeví. Tímto lze vytvářet například rozšiřující CSS knihovny bez toho, aniž by byl výsledný CSS soubor datově náročný. Syntaxe placeholderů je v Sass pomocí uvozujícího znaku % a v případě Stylusu pomocí znaku $.

Demonstrace syntaxe a využití placeholderu je v následujícím příkladu – pokud by placeholder `%extreme` nebyl pomocí příkazu `@extend` nikde využit, žádný element, spadající pod selektor `#context a`, by neobsahoval deklarované styly.

Kód v Sass:

```
#context a%extreme {
  color: blue;
  font-weight: bold;
  font-size: 2em;
}
```

Výsledný CSS kód:

```
#context a {
  color: white;
}
```

```
#context a {
  color: white;
}
```

3.3 Jmenné prostory

V rámci bezpečnější práce s kódem a orientaci v něm může být vhodné mixiny nebo proměnné seskupit do větších celků. Toho lze docílit buď rozdělením kódu do více souborů a jejich importem do jednoho hlavního souboru nebo využitím tzv. jmenných prostorů, které jsou dostupné pouze v preprocesoru LESS.

Jmenné prostory slouží ke sdružování stylů do celků, které spolu logicky souvisí. Mezi hlavní důvody takového seskupování patří:

- vyvarování se konfliktům pojmenování v rozsáhlejších projektech,
- enkapsulace (Enkapsulace v objektově orientovaném programování znamená, že k obsahu objektu se nedostane nikdo jiný, než sám objekt. Navenek se objekt projevuje jen svými operacemi a metodami.) a izolace kódu od okolí,
- organizace kódu.

Pro definování jmenného prostoru se využívá znaku #, vlastního názvu a složených závorek, ve kterých se nachází jeho obsah. Pomocí jmenného prostoru se lze odkazovat pouze na proměnné a mixiny, jež byly definovány uvnitř. Sdružovat lze i samotné jmenné prostory, a to jejich zanořením, čímž se vytváří hierarchická struktura.

Kód v LESS:

```
#package {
  @seaBlue: #428bca;
  .button {
    display: block;
    border: 1px solid black;
    background-color: grey;
```

Výsledný CSS kód:

```
#header a {
  display: block;
  border: 1px solid black;
  background-color: grey;
}
```

```
    &:hover {
      background-color: white
    }
  }
}

#header a {
  #package > .button;
}
```

```
#header a:hover {
  background-color: white;
}
```

3.4 Import

Funkce `@import` slouží k importování stylů z jiných souborů. Pokud preferujete organizaci stylů pomocí zápisu do oddělených souborů, CSS funkci importu již nabízí, ale obsahuje velký nedostatek, kvůli kterému může být taková organizace kódu na považenou. Rozdílem mezi tím, jak pracuje funkce import v CSS a v preprocesorech je, že každý CSS `@import` vytváří nový http požadavek na server pro stažení souborů se styly, kdežto import v jakémkoliv preprocesoru sloučí obsah importovaných souborů do jednoho výsledného zkompilovaného stylu. To znamená pouze jediný http požadavek pro stažení stylů, který umožňuje organizovat kód do souborů, aniž by docházelo k prodlužování rychlosti načítání stránky. Všechny proměnné nebo mixiny definované v importovaných souborech lze také použít v hlavním souboru, do kterého je import prováděn. Soubory jsou hledány ve stejném adresáři, ve kterém je umístěn hlavní soubor nebo lze případně specifikovat cestu.

Ve standardním CSS je funkce také omezena svým použitím pouze na začátek dokumentu před jakýmikoliv vlastními styly daného dokumentu, do kterého je importováno. V preprocesorech může být funkce importu v rámci souboru použita kdekoliv. Následující příklad by tedy v CSS nebyl funkční.

```
.body {
  background: #900;
}

@import "importovane_styly.css";
```

Import funguje obecně na principu iterace skrze pole složek, ve kterých je vyhledáván požadovaný soubor. Ve výchozím stavu je toto pole nastaveno na jedinou složku, jež je odvozena od názvu souboru. Není-li koncovka souboru definována, je hledán soubor s koncovkou *.styl, .less* nebo *.scss* v závislosti na preprocesoru. Následující příkaz by v případě Stylusu vyhledával soubor *border-radius.styl* ve složce *mixins.*

```
@import "mixins/border-radius"
```

3.4.1 Způsoby importu

V případě preprocesoru LESS nalezneme několik možností pro importování stylů. Syntaxe importu je v něm následující: `@import (method) "nazev_souboru.less";`, kde za slovo `method` lze dosadit jedno nebo více klíčových slov oddělených čárkami, pro specifikaci způsobu importu stylů. Dostupné jsou následující možnosti:

- *reference*: využije styly importovaného souboru, ale není celý součástí výstupu
- *inline*: zahrne soubor do výstupu, ale nezpracuje jej
- *less*: zachází se souborem jako s LESS souborem, nehledě na jeho příponu
- *css*: zachází se souborem jako s CSS souborem, nehledě na jeho příponu
- *once*: zahrne soubor pouze jednou (výchozí chování)
- *multiple*: zahrne soubor vícekrát
- *optional*: pokud není soubor nalezen, pokračuje v běžné kompilaci

V případě Sass lze využít kromě zmíněného následující zápis a pomocí interpolace dosazovat hodnoty do URL adresy dynamicky.

```
$family: unquote("Droid+Sans");
@import url("http://fonts.googleapis.com/css?family=#{$family}");
```

Výsledný CSS kód:

```
@import url("http://fonts.googleapis.com/css?family=Droid+Sans");
```

3.4.2 Import na úrovni zanoření

Preprocesory umožňují použít příkaz `@import` nejen na nulové úrovni zanoření, ale také i uvnitř bloků se styly nebo v podmínkách. Lze tak dosáhnout složitějšího podmíněného začleňování stylů a také menší datové náročnosti, jelikož do hlavního souboru nebude zbytečně importováno velké množství stylů, které nebudou v případně nevykonání podmínky využity.

3.4.3 Import indexových stylů

Příkaz `@import` ve Stylusu umožňuje automatický import indexových stylů. Soubory s názvem *index.styl* jsou importovány automaticky, podobně jako index.html v jazyku HTML. Uvedený předchozí příklad může být interpretován jak pro import souboru *border-radius.styl*, tak i pro import složky *border-radius*, která obsahuje styl s názvem *index.styl*. Uvedenému případu by odpovídala následující souborová struktura:

```
./tablet
  |-- index.styl
  |-- vendor.styl
  |-- buttons.styl
  |-- images.styl
```

Této funkcionality lze využít v případě, že chceme i importované styly strukturovat do oddělených souborů. Automaticky importovaný styl index.styl tak může agregovat další importované styly.

3.4.4 Require

Kromě příkazu `@import` lze využít ve Stylusu i příkazu `@require`, pomocí kterého lze importovat vybraný styl pouze jednou v celém dokumentu. Syntaxe příkazu `@require` je stejná jako pro `@import`.

3.4.5 Globbing

Globbing je funkcionalita frameworku Node.js, pomocí které lze ve Stylusu importovat soubory na základě shody s regulárním výrazem. Pomocí literálů a speciálních znaků (jejich přehled je dostupný na https://github.com/isaacs/node-glob#readme), lze popsat

velké množství souborů k importu v jediném relativně krátkém příkazu. Globbing se vztahuje k příkazům `@import` i `@require`.

Pro příklad mějme souborovou strukturu z kapitoly číslo 3.4.3 o importu indexových stylů bez souboru *index.styl*.

```
./tablet
  |-- vendor.styl
  |-- buttons.styl
  |-- images.styl
```

Import všech těchto stylů lze provést s využitím globbingu pomocí jediného příkazu:

Kód ve Stylusu:

```
@import 'tablet/*'
```

Výsledný CSS kód:

```
@import 'vendor.styl'
@import 'buttons.styl'
@import 'images.styl'
```

3.5 Rozšíření

Preprocesory mohou být rozšířeny o tzv. knihovny nebo pluginy, které agregují řady užitečných mixinů. Jelikož jsou mixiny způsobem, jak znovu využít již jednou definované vlastnosti tříd nebo identifikátorů, vytvořením vlastní knihovny nebo využitím nějaké již dostupné lze ušetřit čas při budoucí tvorbě kódu a také přispět k lepší organizaci kódu.

Pro rozšíření funkcionality v LESS existují *Less.js* soubory, což jsou modifikované JavaScript soubory v UMD (*universal module definition*) formátu a jejich tvorba vyžaduje programovací dovednosti. Pro připojení rozšiřujících *less.js* souborů v LESS slouží příkaz `@plugin`. Lze však využít i již naprogramovaných pluginů, které jsou dostupné v knihovně na stránkách oficiální dokumentace[28]. Pluginy patří mezi funkce LESS verze 3, která se chystá k uvolnění.

[28] Less.js Plugins. *Less.js* [online]. [cit. 2017-03-12]. Dostupné z: http://lesscss.org/3.x/tools/#plugins

Pro preprocesor Sass je dostupné velké množství open-source knihoven mixinů, mezi které patří například Bourbon, LESShat nebo Compass. Přehled dalších knihoven je k dispozici například na portále CSSAuthor[29].

V případě Stylusu existuje oficiální knihovna Nib[30], která obsahuje řadu mixinů, utilit, komponentů a generátor přechodů. Příkladem může být následující příklad, kde po importování knihovny v jediném řádku odpadá nutnost zabývat se vytvářením vlastního mixinu pro vytvoření.

Kód ve Stylusu:

```
@import 'nib';
body { background: linear-gradient(top, white, black); }
```

Výsledný CSS kód:

```
body {
  background: -webkit-gradient(linear, left top, left bottom,
    color-stop(0, #fff), color-stop(1, #000));
  background: -webkit-linear-gradient(top, #fff 0%, #000 100%);
  background: -moz-linear-gradient(top, #fff 0%, #000 100%);
  background: linear-gradient(top, #fff 0%, #000 100%);
}
```

[29] 65+ SASS Mixins Library. *CSS Author* [online]. 2016 [cit. 2017-04-11]. Dostupné z: http://www.cssauthor.com/sass-mixins-library/

[30] *Nib – CSS3 extensions for Stylus* [online]. [cit. 2017-04-11]. Dostupné z: http://tj.github.io/nib/

4 Programatická logika

Podmíněná logika umožňuje v programovacích jazycích kontrolovat běh programu a větvit kód. CSS tuto logiku postrádá, což je limitující a může vést k redundanci velmi podobného kódu. Kromě media queries, které se vztahují pouze na změny rozlišení a typu zařízení, zde není možnost, jak použít styl pouze v určitém případě při splnění definované podmínky. Preprocesory proto nabízejí matematické výpočty výrazy, podmínky, cykly a funkce pro jednorázové či opakované začleňování stylů s možnými variacemi.

4.1 Parametrické mixiny

Parametrické mixiny fungují podobně jako funkce v běžných programovacích jazycích. Pomocí parametrických mixinů lze dosáhnout dynamického výstupu, jehož výsledná hodnota je ovlivněna vstupní hodnotou. Díky tomu není zapotřebí vytvářet pro každý případ nový styl, ale stačí zavolat mixin s rozdílnými vstupními hodnotami, pro dosažení rozdílných výstupních stylů.

Parametrické mixiny přijímají jeden nebo více parametrů, které se volají s příslušnými hodnotami v závorce. Jednotlivé parametry se oddělují čárkou nebo středníkem. Parametry mohou mít také své výchozí hodnoty, které mohou a nemusí být voláním nahrazeny, jako např. parametr `padding` v následujícím příkladě v LESS:

Kód v LESS:

```
.mixin(@color; @padding: 10) {
  color: @color;
  padding: @padding;
}

.header-wrap div {
  background-color: black;
  .mixin(#008000, 20);
}
```

Výsledný CSS kód:

```
.header-wrap div {
  background-color: black;
  color: #008000;
  padding: 20;
}
```

Pokud nechceme uvnitř mixinu jednotlivě volat veškeré argumenty a chceme je vypsat hromadně, lze použít příkaz příkazu `@arguments`. Praktické využití tohoto příkazu je

například v případě vypisování stylů pro více prohlížečů, kde se mění pouze název selektoru a hodnoty zůstávají stejné nebo v případech, kdy se jednotlivé parametry k sobě vztahují a píší se vždy společně ke konkrétnímu stylu. Praktické využití může být v případě předávání hodnot k vlastnosti `border`, u které je vždy nutno uvést tloušťku, styl a barvu.

Kód v LESS:

```
.box-shadow(@x: 0; @y: 0; @blur: 1px; @color: #000) {
  -webkit-box-shadow: @arguments;
     -moz-box-shadow: @arguments;
          box-shadow: @arguments;
}
.box {
  .box-shadow(2px; 5px);
}
```

Výsledný CSS kód:

```
.box {
  -webkit-box-shadow: 2px 5px 1px #000;
     -moz-box-shadow: 2px 5px 1px #000;
          box-shadow: 2px 5px 1px #000;
}
```

Hromadného volání parametrů lze využít i v případě, kdy se může počet argumentů mixinu měnit. Příkladem může být následující mixin pro tvorbu přechodů, který lze pokaždé volat s jiným počtem parametrů (tedy barev, které chceme do přechodu zahrnout). Proměnná, která má sloužit pro více parametrů se v Sass zapisuje se třemi tečkami a musí být vždy posledním parametrem mixinu.

Kód v Sass:

```
@mixin linear-gradient($direction, $gradients...) {
  background-color: nth($gradients, 2);
  background-image: linear-gradient($direction, $gradients...);
}

.selector {
  @include linear-gradient(to right, magenta, red, orange);
}
```

Výsledný CSS kód:

```
.selector {
  background-color: red;
  background-image: linear-gradient(to right, magenta, red, orange);
}
```

Z uvedeného příkladu lze vidět, že na rozdíl od LESS se mixiny v Sass uvádějí stejně jako proměnné a to znakem @ a klíčovým slovem `mixin`. Jejich volání je taktéž odlišné, a to pomocí vestavěné funkce `@include`. Parametry v závorce se uvádějí pomocí znaku $. V uvedeném příkladu si také lze všimnout funkce `nth($gradients, 2)`, která slouží pro volání *n*-tého parametru proměnné.

Syntaxe vstupních parametrů mixinů ve Stylusu je bez znaků a v případě hromadných parametrů se třemi tečkami a volání mixinů je také bez znaků, podobně jako v LESS. To může být někdy i nevýhodou, například má-li mixin stejný název, jako vlastnost, kterou chceme deklarovat. V následujícím příkladu je přednostně volán mixin, namísto vlastnosti `borer-radius`.

Kód ve Stylusu:

```
border-radius(n)
  -webkit-border-radius n
  -moz-border-radius n
  border-radius n

form input[type=button]
  border-radius 5px
```

Výsledný CSS kód:

```
form input[type=button] {
  -webkit-border-radius: 5px;
  -moz-border-radius: 5px;
  border-radius: 5px;
}
```

4.2 Direktiva

Řídící direktiva a výrazy pomáhají v CSS preprocesorech vytvářet podobné styly, které závisí na splněné podmínce a vstupních hodnotách[31]. Sass a Stylus podporují standardní *if* / *else* výrazy a v případě Stylusu i unless, známé z programovacích jazyků. V případě LESS lze podobného chování dosáhnout pomocí tzv. strážců. Direktivami lze obecně

[31] An Introduction to CSS Pre-Processors: SASS, LESS and Stylus. *HTML Mag* [online]. 2014 [cit. 2017-04-15]. Dostupné z: https://htmlmag.com/article/an-introduction-to-css-preprocessors-sass-less-stylus

docílit větvení kódu. Využíváním direktiv tak lze například reagovat na určité stavy, jako je změna velikosti okna prohlížeče nebo orientace zařízení.

4.2.1 Funkce if()

Výraz `if()` je vestavěná funkce dostupná pouze v preprocesoru Sass. Tato funkce vrací příslušnou vstupní hodnotu na základě toho, zda je definovaná podmínka pravdivá (*true*) nebo nepravdivá (*false*). Syntaxe je následující:

```
if($podminka, a, b);
```

Tato vestavěná funkce slouží jako ternární operátor (operátor, pracující se třemi operandy) a její postup zpracování je následný:

1. Vyhodnotí se podmínka.
2. Je-li podmínka pravdivá, vyhodnotí se výraz `a`, který je výsledkem.
3. Je-li podmínka nepravdivá, vyhodnotí se výraz `b`, který je výsledkem. První výraz se nevyhodnocuje.

Vzhledem k tomu, že je vyhodnocen pouze argument, který koresponduje s výsledkem operace, je možné se odkazovat na proměnné, které nebyly definovány nebo na operace, které by v normálním případě způsobily chybu.

Praktickým příkladem pro využití funkce `if()` může být následující funkce v Sass, uvnitř které nalezneme dvě vnořené `if()` funkce.

```
@function clamp($value, $min, $max) {
  @return if($value > $max, $max, if($value < $min, $min, $value));
}
saturate($theme, clamp($base-exposure * 2, 0%, 100%));
```

Pro pochopení chování je zapotřebí postupovat stejným stylem vyhodnocování, jako postupuje sama funkce. Výstupem je buď vstupní hodnota `$value` nebo v případně překročení mezních hodnot `$min` či `$max` jejich příslušná hodnota. Uvedená funkce

může být použita například pro ošetření vstupu jiné funkce, kde je vyžadována číselná hodnota v určitém rozsahu.

4.2.2 If, else, else if

If, *else* a *else if* jsou výrazy pro podmíněné začleňování stylů. Na rozdíl od vestavěné funkce `if()` v Sass vrací při splněné podmínce celý blok vnořených stylů. Výraz *if* může být doplněn o libovolný počet *else if* výrazů a jedním *else* výrazem. Je-li výsledkem vyhodnocené podmínky hodnota *false*, přechází vyhodnocování na další *else if* výrazy, dokud není podmínka splněna nebo není dosaženo výrazu *else*. Pokud se za *if* nevyskytují další výrazy, výsledkem není žádná hodnota.

Podmíněné výrazy lze využít v kombinaci s parametrickými mixiny, čímž lze dosáhnout uzavřené konstrukce podmínek s možností využít vstupních parametrů v podmínkách. Pomocí toho dosahujeme onoho větvení kódu, které je závislé na vstupu.

Syntaxe podmíněných výrazů probíhá v Sass pomocí znaku @ a v případě Stylusu pomocí znaku $, jehož užití je dobrovolné. Příkladem použití a demonstrace syntaxe v Sass může být následující kód. Pomocí jediného řádku s příkazem pro volání mixinu se můžeme rozhodnout, který blok stylů chceme začlenit. Výsledkem je, vzhledem ke kladné podmínce, definované přímo v místě volání, blok stylů mixinu `flex`.

Kód v Sass:

```
@mixin flex {
  display: flex;
  ...
}

@mixin grid($flex) {
  @if $flex { @include flex; }
  @else { display: block; }
}

@include grid(true);
```

4.2.3 Unless

Tento příkaz preprocesoru Stylus je převzat z programovacího jazyka Ruby, přestože je Stylus napsán v jazyce JavaScript. Je negací příkazu `if` a lze jej proto vyjádřit následujícím zápisem: `if (!(vyraz))`. V tomto kontextu se tedy nejedná o příkaz, který by přinášel do jazyka novou funkcionalitu, jako spíše jen o příkaz pro lepší logickou souvislost kódu. Následující příklad s příkazem `unless` lze tedy přepsat pomocí příkazu `if` a negací ostatních parametrů. Výsledkem obou případů bude: `body { margin: 5px 10px; }`.

Kód ve Stylusu:

```
padding-override = true

unless padding-override is defined
and !padding-override
   padding(x, y)
     margin x y

 body
   padding 5px 10px
```

Totožný kód ve Stylusu:

```
padding-override = true

if padding-override is not
defined and padding-override
   padding(x, y)
     margin x y

 body
   padding 5px 10px
```

Příkaz lze ovšem využít pro podmíněné volání mixinů na základě existence jiného stylu. V následujícím příkladě je aplikován styl `z-index` pouze pod podmínkou, pokud dosud nebyl specifikován v bloku stylů, odkud je mixin volán.

Kód ve Stylusu:

```
position()
  position: arguments
  z-index: 1 unless @z-index

#logo
  z-index: 20
  position: absolute

#logo2
  position: absolute
```

Výsledný CSS kód:

```
#logo {
  z-index: 20;
  position: absolute;
}

#logo2 {
  position: absolute;
  z-index: 1;
}
```

4.3 Strážci

Selektor, jmenný prostor nebo mixin může mít v LESS tzv. strážce (*CSS guard*), neboli podmínku, při jejímž splnění jsou definované styly teprve použity. Následující zápisy strážců jsou vyhodnoceny totožně:

```
#namespace when (@mode=huge) {
  .mixin() { /* */ }
}

#namespace {
  .mixin() when (@mode=huge) { /* */ }
}
```

Jak již bylo řečeno v úvodě kapitoly o direktivách, strážci jsou v LESS způsobem, jak dosáhnout logického větvení kódu na základě splnění vstupní podmínky. Společně se zanořováním lze vytvářet konstrukce, které jsou podobné konstrukcím *if / else* z programování, větvit kód a reagovat na různé situace.

Kód v LESS:

```
.color(@width) {
  & when(@width>1200) {
    color: red;
  }
  & when(@width<=1200) {
    color: green
  }
}

p { .color(950) }
```

Výsledný CSS kód:

```
p { color: green; }
```

4.4 Cykly

V programování se cykly rozumí sekvence instrukcí, které se neustále opakují, dokud není dosaženo určité podmínky. Jedná se o základní myšlenku, které se využívá při psaní programů[32]. V CSS se cykly využívají pro vytváření sad selektorů nebo hodnot, které mají na pozadí nějaký matematický vztah[33]. Obecně existují v programovacích jazycích i CSS preprocesorech tří druhy cyklů[34]:

- For cykly, které mají předem definovaný počet opakování.
- While cykly, které jsou nejvíce obecné a jejich ukončení je závislé na dosažení vstupní podmínky.
- Each cykly, jejichž počet opakování odpovídá počtu prvků listu, daného na vstupu.

4.4.1 For

Cyklus *for* je v programování využíván pro vytváření smyček, které se opakují do té doby, dokud nedojde ke splnění podmínky. Podobným způsobem pracuje i výraz `@for` v Sass, případně `for` v syntaxi Stylusu, který opakovaně vrací sadu stylů. Během každého jeho cyklu dochází k přibližování se k požadované podmínce. Cykly obsahují předem definovaný počet opakování a nepřijímají žádné parametry a jsou také velkým pomocníkem při procházení listů, tabulek nebo jiných posloupností, kde dochází k pravidelnému navyšování v určitém vzoru.

V Sass existují dvě varianty cyklu *for*. Syntaxe je následující, kde `<start>` a `<end>` zastupují celočíselné hodnoty:

```
@for $i from <start> through <end> {
  ...
}

@for $i from <start> to <end> {
  ...
}
```

[32] What is loop? *Whatis.com* [online]. 2005 [cit. 2017-04-15]. Dostupné z: http://whatis.techtarget.com/definition/loop
[33] Loops. *CSS PREprocessors* [online]. 2014 [cit. 2017-04-15]. Dostupné z: https://csspre.com/loops/
[34] Loops in CSS Preprocessors. *CSS-Tricks* [online]. 2016 [cit. 2017-04-15]. Dostupné z: https://css-tricks.com/loops-css-preprocessors/

Proměnná `$i` může mít libovolný název a slouží zde jako čítač cyklů. Na počátku cyklu je nastavena její hodnota na hodnotu `<start>`. Při každém cyklu je hodnota této proměnné předána do bloku vnořených stylů a poté navýšena či snížena o hodnotu jedné. Je-li hodnota `<start>` nižší než `<end>`, hodnota proměnné `$i` se bude každým cyklem navyšovat a naopak. Rozdíl mezi uvedenými zápisy je v klíčových slovech `through` a `to`. První zápis zahrnuje hodnotu `<end>`, druhý nikoli. Jeho běh tedy skončí v momentě, kdy je hodnota proměnné `$i` větší nebo menší než hodnota `<end>` v závislosti, zda má cyklus navyšující či snižující charakter.

Stylus nabízí možnost totožné iterace skrze list prvků a to pomocí konstrukce `for` / `in`. Syntaxe cyklu v tomto preprocesoru je následující:

```
for <val-name> [, <key-name>] in <expression>
```

Spolu s využitím mixinů, listů a interpolace lze šikovně generovat styly, které závisí na počtu prvků v listu, který je vstupním parametrem. Interpolace umožňuje v CSS preprocesorech využívat proměnné i v názvech selektorů a vlastností. V následujícím příkladě je s pomocí cyklu `for` vygenerován grid systém, který slouží k umisťování a pozicování obsahu a využívá se v CSS frameworcích, jako je například Bootstrap[35] (viz obrázek č. 3).

Kód ve Stylusu:

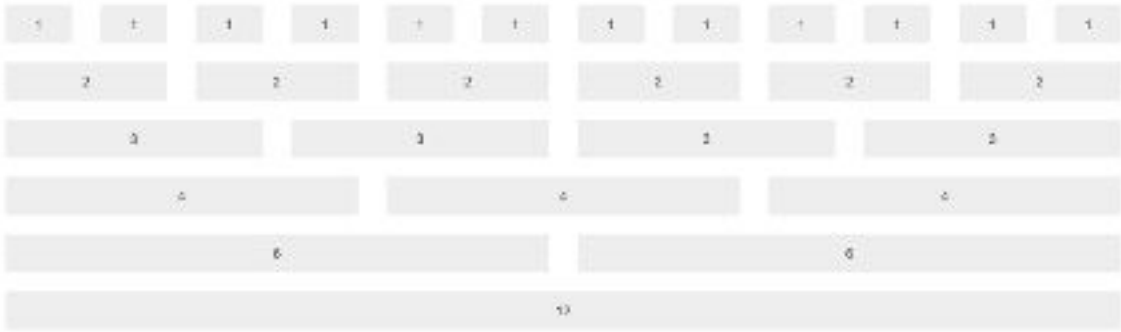
```
grid(props...)
  for prop in props
    .col-{prop[0]}
      width: unit(prop[1], \%)

grid(12 100, 11 50, 10 33.3, 9 25, 8
20, 7 16.66, 6 14.2, 5 12.5, 4 11.1, 3
10, 2 9, 1 8.3)
```

Výsledný CSS kód:

```
.col-12 { width: 100%; }
.col-11 { width: 50%; }
.col-10 { width: 33.3%; }
.col-9 { width: 25%; }
.col-8 { width: 20%; }
.col-7 { width: 16.66%; }
.col-6 { width: 14.2%; }
.col-5 { width: 12.5%; }
.col-4 { width: 11.1%; }
.col-3 { width: 10%; }
.col-2 { width: 9%; }
.col-1 { width: 8.3%; }
```

[35] Layout: Grid system. *Bootstrap* [online]. [cit. 2017-03-12]. Dostupné z: https://v4-alpha.getbootstrap.com/layout/grid/



Obrázek č. 4: Výchozí grid systém Boostrapu[36]

4.4.2 While

Cyklus `@while` je příkaz pouze preprocesoru Sass a má podobné chování jako `@for`. Na rozdíl od něj ale využívá podmínky, které přijímají parametry (viz následující příklad) a lze tak dosáhnout komplexnějších smyček. Stejně jako `@for` se tento cyklus opakuje do doby splnění podmínky a při každém jednotlivém cyklu vrací blok stylů.

Kód v Sass:

```
$i: 6;
@while $i > 0 {
  .item-#{$i} { width: 2em * $i; }
  $i: $i - 2;
}
```

Výsledný CSS kód:

```
.item-6 { width: 12em; }
.item-4 { width: 8em; }
.item-2 { width: 4em; }
```

4.4.3 Each

Cyklus `@each` slouží v Sass, podobně jako v programovacích jazycích, k iteraci (opakovanému používání funkce s cílem přibližování se cíli a jeho dosažení) skrze list s prvky, na jehož základě generuje styly. Jednotlivé prvky listu jsou postupně procházeny a jejich hodnoty mohou být libovolně použity v těle cyklu. Výhodou tohoto cyklu je jeho nespecifikovaný počet opakování, který závisí na délce množiny na vstupu. Syntaxe cyklu `@each`, který je dostupný pouze v případě preprocesoru Sass, je následující:

```
@each $var in <list>
```

[36] SPURLOCK, Jake. O'Reilly [online]. [cit. 15.4.2017]. Dostupný na WWW: https://www.safaribooksonline.com/library/view/bootstrap/9781449344573/ch01.html

Proměnná `$var` může mít libovolný název a její hodnota se v průběhu mění, jelikož v každém cyklu zastupuje postupně jednotlivé hodnoty prvků z listu. Průběh iterace lze demonstrovat na následujícím příkladu, ve kterém jsou za pomocí listu, vestavěné funkce `index()`, cyklu `@each` a interpolace generovány styly.

Kód v Sass:

```
$colors: #111 #222 #333 #444 #555;
@each $color in $colors {
    $i: index($colors, $color);
    .item-#{$i} {
        color: $color;
    }
}
```

Výsledný CSS kód:

```
.item-1 { color: #111111; }
.item-2 { color: #222222; }
.item-3 { color: #333333; }
.item-4 { color: #444444; }
.item-5 { color: #555555; }
```

4.5 Vlastní funkce

V Sass je možné definovat i vlastní funkce, které jsou velmi podobné mixinům. V obou případech jsou hodnoty vstupních parametrů stejných typů a jejich počet je libovolný. Rozdílem je jejich návratová hodnota, která není v případě vlastních funkcí blok stylů, ale může být jakýmkoliv datovým typem, který Sass podporuje (tedy číslo, řetězec, barva, logická hodnota nebo list). Mohou také obsahovat více úseků, ve kterých může dojít k jejich ukončení a vrácení výsledné hodnoty, k čemuž slouží příkaz `@return`. Syntaxe a základní kostra funkcí je následující:

```
@function nazevFunkce ($parametr1, $parametr2) {
  ...
  @return vystup;
}
```

Zatímco účelem mixinů je minimalizace vkládání opakujícího se kódu, cílem funkcí je odstranit opakující se logiku. Jejich použití je proto vhodnější tehdy, má-li být výsledkem hodnota, která může být použita někde jinde[37].

[37] Sass Basics: The Function Directive. *SitePoint* [online]. 2015 [cit. 2017-04-15]. Dostupné z: https://www.sitepoint.com/sass-basics-function-directive/

4.6 Vestavěné funkce

LESS, Sass i Stylus nabízí širokou škálu vestavěných funkcí. Rozebíráním všech jednotlivých funkcí jazyků se nebudeme zabývat. Jejich kompletní seznam je dostupný na oficiálních stránkách s dokumentací těchto preprocesorů[38, 39, 40]. V následujících kapitolách se zaměříme pouze na vybrané funkce, spadající do větších celků, které jazyk nejvíce rozšiřují.

V případě jediného Stylusu lze jakékoliv funkci přiřadit vlastní název neboli tzv. alias. Pro vytvoření aliasu se postupuje podobně, jako při vytváření proměnných v syntaxi tohoto preprocesoru.

Kód ve Stylusu:

```
plus = add
plus(1, 2)      // => 3
```

4.6.1 Výpočty a matematické funkce

Zastoupení matematických funkcí v čistém CSS je omezeno pouze na operátory pro základní matematické operace a na funkci `calc()`, která slouží pro samotný výpočet a tedy vůbec pro využívání těchto operátorů. CSS postrádá jakékoliv možnosti například pro zaokrouhlování nebo mocnění.

První praktickou vlastností preprocesorů, co se výpočtů týče, je proto možnost využívání operátoru bez nutnosti vkládání veškerých výpočtů do funkce `calc()`. Výpočty mohou být samozřejmě kombinovány se závorkami pro dosažení komplexnějších kalkulací.

Kód v Sass:

```
$itemWidth: 100px;

.box {
  width: $itemWidth * 1.5;
  padding: (($itemWidth * 1.5) - $itemWidth) / 2;
}
```

[38] Functions. *Less.js* [online]. [cit. 2017-03-12]. Dostupné z: http://lesscss.org/functions/
[39] Sass Documentation. *Sass (Syntactically Awesome StyleSheets)* [online]. [cit. 2017-03-12]. Dostupné z: http://www.sass-lang.com/documentation/Sass/Script/Functions.html
[40] Built-In Funstions. *Stylus* [online]. [cit. 2017-03-12]. Dostupné z: http://stylus-lang.com/docs/bifs.html

Výsledný CSS kód:

```
.box {
  width: 150px;
  padding: 25px;
}
```

Preprocesory navíc obsahují několik vestavěných funkcí, které pomáhají manipulovat s čísly. Jedná se o funkce `round()`, `ceil()`, `floor()`, `abs()`, `min()`, `max()`, `random()` a `percentage()` pro převod pixelů na procenta. Sass obsahuje pouze tyto základní funkce. LESS rozšiřuje počet dostupných funkcí o mocnění a umocňování (`pow()` a `sqrt()`) , goniometrické funkce (`sin()`, `asin()`, `cos()`, `acos()`, `tan()`, `atan()`) a funkce `pi()`, `mod()` a `convert()` pro převod číselných, časových a úhlových jednotek. Stylus kromě všech zmíněných funkcí obsahuje navíc funkce `avg()`, `sum()` a odlišnou funkci `base-convert()` pro převod číselných soustav jako náhradu za funkci `convert()`.

4.6.2 Definice barev a práce s nimi

LESS, Sass i Stylus poskytují pomoc při práci s barvami a nabízí funkce pro definici barevných modelů (funkce `rgb()` a `hsl()` plus jejich variace včetně průhledných variant s alfa kanálem) a několik dalších funkcí pro změnu a manipulaci s barvami, jako například možnost převodu hexadecimální hodnoty barvy na hodnotu RGBa. Kromě čísel může být matematika prováděna také na barvách. Výpočty jsou prováděny nad červenou, modrou a zelenou složkou.

Kód v Sass:

```
color: rgba(#8ec63f, .25);
color: #8ec63f + #666
color: #8ec63f * 2
```

Výsledný CSS kód:

```
color: rgba(142, 198, 63, 0.25);
color: #f4ffa5;
color: #ffff7e;
```

Barevné úpravy v LESS a Sass nabízí funkce pro invertování barev (`invert()`), hledání doplňkových barev (`complement()`), kombinování barev dohromady (`mix()`, `tint()`, `shade()`), převod barev na stupně šedi (`grayscale()`), změnu sytosti (`saturate()` a `desaturate()`) nebo zesvětlení či ztmavení (`lighten()` a `darken()`). V Sass i Stylusu

lze také měnit průhlednost a v případě Stylusu využít pokročilejších funkcí pro míchání kanálů (`negation()`, `multiply()`, `overlay()` a další).

Pomocí uvedených funkcí pro práci s barvami lze například globálně ztmavovat či zesvětlovat barvy celého webu či vytvářet přechody čistě pomocí CSS namísto zbrzďováním načítání stránek při použití obrázků.

4.6.3 Práce s textovými řetězci

Preprocesory nabízí možnost formátování nebo nahrazování textových řetězců. Toho lze využít například při dynamickém dosazování textových hodnot do URL adresy nebo v případě CSS vlastnosti `content`, která umožňuje přidávat HTML obsah před nebo za stylovaný prvek.

V případě Sass lze k transformaci textových řetězců využít funkce pro vkládání a odebírání uvozovek (`quote()`, `unquote()`), vkládání znaků na určité pozice (`str-insert()`), extrahování textových řetězců z určitých pozic (`str-slice()`) nebo pro převod na malá či velká písmena (`str-upper-case()`, `str-lower-case()`). Stylus dále nabízí funkce pro vyhledávání textových řetězců (`match()`) a jejich nahrazování (`replace()`).

4.6.4 Práce s listy a mapami

Listy a mapy patří mezi datové typy preprocesorů. Při práci s typickými funkcemi preprocesorů lze v určitých případech listů a map šikovně využít. Jejich použití je nejčastěji jako zdroj vstupních hodnot pro cykly, funkce nebo mixiny, kde je nutné předat neznámé množství parametrů. Jedním z běžných příkladů využití map může být přiřazování různých barev k různým ikonám, kde každá dvojice hodnot bude obsahovat barvu a název. Pomocí cyklu `@each` a interpolace lze následně přistupovat k jednotlivým hodnotám párů mapy.

Kód v Sass:

```
$social: (
  'facebook': #3b5999,
  'twitter': #55acee,
  'linkedin': #0077B5,
  'google': #dd4b39,
);
```

```
@each $name, $color in $social {
  .wrapper-social div.#{$name} {
    background: $color;
    &::before {
      content: url(https://example.com/#{$name}.png);
    }
  }
}
```

Výsledný CSS kód:

```
.wrapper-social div.facebook { background: #3b5999; }
.wrapper-social div.facebook::before {
  content: url(https://example.com/facebook.png);
}

.wrapper-social div.twitter { background: #55acee; }
.wrapper-social div.twitter::before {
  content: url(https://example.com/twitter.png);
}

.wrapper-social div.linkedin { background: #0077B5; }
.wrapper-social div.linkedin::before {
  content: url(https://example.com/linkedin.png);
}

.wrapper-social div.google { background: #dd4b39; }
.wrapper-social div.google::before {
  content: url(https://example.com/google.png);
}
```

Podobně jako v uvedeném příkladu, lze vytvářet a generovat styly na základě listů nebo map s údaji o breakpointech, šířkách, barvách apod. Výhodou od určitého momentu je kratší zápis a jedno centralizované místo se všemi hodnotami a údaji pro rychlou změnu a možnost správy.

Mezi funkce, které Sass pro práci s listy a mapami nabízí, patří například přidávání a odebírání párů map (`map-merge()`, `map-remove()`), extrahování hodnot map (`map-get()`, `map-keys()`, `keywords()`), přidávání či slučování listů (`join()`, `merge()`, `append()`, `zip()`), extrahování a nahrazování prvků listů (`nth()`, `set-nth()`) validace párů map (`map-has-key()`) a další. Stylus nabízí funkce stejné pouze s občasným odlišným názvem. Preprocesor LESS nabízí pouze funkce `length()` a `extract()`

5 Proč CSS preprocesory používat

5.1 Slabiny čistého CSS

Jak již bylo zmíněno v kapitole o kaskádových stylech, CSS je jazykem deklarativního charakteru a jeho zápis je tedy lineární. Jazyk také postrádá možnost vytvářet podmínky, vlastní funkce a nedisponuje žádným způsobem, jak styly uvnitř souboru strukturovat. S příchodem zařízení s přístupem na internet dává smysl budovat webové aplikace responzivní. Napříč media queries může ale vznikat mnoho duplicitního kódu. V CSS také neexistuje možnost, jak si uložit například barvu nebo celé bloky stylů, které se opakují u více elementů nebo v media queries.

S rostoucími požadavky na uživatelská rozhraní se webové aplikace stávají robustnějšími a komplexnějšími a dochází tak ke snaze ohýbat CSS k věcem, ke kterým tento jazyk nebyl vytvořen[41]. V případě nových funkcí u CSS3 je nutné také často uvádět i prefixované vlastnosti pro různé prohlížeče (viz obrázek č. 5). Prefixy slouží k označení CSS vlastností, jejichž podpora je v případě daného prohlížeče stále ve vývoji. Jelikož jednotlivé prohlížeče ignorují kód s odlišným prefixem než vlastním, je nutné napsat téměř totožný styl se všemi prefixy prohlížečů, kde požadujeme aby byl styl vykreslen podle našich požadavků.

Vzhledem k charakteru jazyka tedy není možné psát kód, který se nebude opakovat (tzv. „*DRY*“ kód) a v určitém bodě tak začne vždy docházet k jeho redundanci. Čitelnost a možnost správy takového souboru bude klesat s jeho velikostí.

Hlavní otázkou by proto nemělo být, zda jsou obecně preprocesory užitečné či nikoliv. Vzhledem k tomu, že preprocesory reagují na uvedené slabiny CSS, jsou při správném použití vhodnou součástí pracovních postupů. Důležitější otázkou, kterou je potřeba si klást, je, zda je vhodné preprocesory začít používat či nikoliv.

[41] CEDERHOLM, Dan. Sass for Web Designers. New York: Book Apart, 2013. ISBN 978-1937557126.

Obrázek č. 5: Přehled prefixů prohlížečů[42]

5.2 Pro koho jsou CSS preprocesory vhodné

Přestože začlenění preprocesorů do pracovního postupu doporučuje i společnost Google [43], toto rozhodnutí by mělo být provedeno na základě výše odborné úrovně, co se znalosti CSS týče a také po analýze požadavků projektu a týmu jako celku[44]. Existují případy, kdy používání preprocesoru není vhodným rozhodnutím a jeho přítomnost by mohla naopak škodit.

Prvním případem je projekt, na kterém pracuje začátečník nebo tým složený ze začátečníků. Pokud není uživatel s jazykem dostatečně seznámen, preprocesory mohou celkovou výkonnost při tvorbě kódu snižovat. Pro používání preprocesorů je zapotřebí nejprve pochopit a být schopen používat základní principy jazyka CSS a poté je možné přejít na další stupeň a začít využívat pokročilejší metody, jako například CSS preprocesory. Je zapotřebí si uvědomit, že preprocesory sami od sebe neprodukují

[42] TAMADA, Srinivas. 9lessons [online]. [cit. 15.4.2017]. Dostupný na WWW: http://www.9lessons.info/2010/12/css3-for-web-design-part-1.html
[43] Set Up CSS and JS Preprocessors. Google Developers [online]. 2017 [cit. 2017-04-15]. Dostupné z: https://developers.google.com/web/tools/setup/setup-preprocessors
[44] The Power of Sass and Why You Should Embrace CSS Preprocessors. *1stWebDesigner* [online]. 2016 [cit. 2017-04-15]. Dostupné z: https://1stwebdesigner.com/power-sass-why-use-css-preprocessors/

špatný kód, ale vývojáři ano. Pokud bude preprocesoru předložen špatný, nadbytečný a komplikovaný kód, výsledek nebude uspokojivý.

Druhým případem jsou malé webové projekty, kde se předpokládá krátký proces tvorby bez očekávaných častých či vůbec nějakých změn. Preprocesor v tomto případě nebude natolik užitečný a nebude využito jeho plného potenciálu.

V případě velkých a komplexních projektů, kde se na vývoji podílí více lidí, se preprocesor stává užitečným nástrojem[45]. Rozsáhlé projekty často spočívají v tom, že je nutné doručit výsledek v konkrétním čase klientovi, který se nestará o to, co se dělo na pozadí. Zavedení nových nástrojů může být v případě týmu, který nedrží krok s novými trendy, však náročným procesem, přestože by se mohlo jednat ve výsledku o urychlení vývoje. Implementace by mohla zabrat příliš mnoho času a ne každý tým má čas a prostředky vynaložit úsilí pro zavedení nových nástrojů a postupů[46].

Jedinec nebo tým zkušených vývojářů je proto pro přijetí preprocesoru skvělým prostředím, ale pouze tehdy, je-li sám vývojář či jiný člen týmu zkušenou osobou, která umí zacházet s rozsáhlými CSS soubory a strukturovat je. Pro správné využití preprocesoru je nejprve důležité definovat a zavést strategii pro organizaci CSS[47].

5.3 Vliv na rychlost pracovního postupu

Díky možnosti preprocesorů vytvářet proměnné a odkazovat se na ně, namísto přepisování jejich hodnot, dochází k urychlení vývoje. Pokud je zapotřebí aktualizace hodnot, stačí ji tak provést na jediném místě, nikoli na 20. Podobně díky existenci mixinů a jejich schopnosti vytvářet bloky stylů pro opakované použití je možné měnit vzhled a chování elementu opět na jediném místě.

Příkladem, jak může preprocesor šetřit čas při vývoji, je využití mixinů nebo rozšiřujících open-source knihoven, jako například Nib, Bourbon, LESShat nebo Compass. Díky nim je možné využívat plno užitečných funkcí, které poskytují podporu

[45] Why do some people use CSS preprocessors? Quora [online]. 2014 [cit. 2017-04-15]. Dostupné z: https://www.quora.com/Why-do-some-people-use-CSS-preprocessors

[46] Why are there still people alive today that don't use CSS Preprocessors ? *Hashnode* [online]. 2016 [cit. 2017-04-15]. Dostupné z: https://hashnode.com/post/why-are-there-still-people-alive-today-that-dont-use-css-preprocessors-cipr6r3dc002dz753y1pxgqwf

[47] The Power of Sass and Why You Should Embrace CSS Preprocessors. *1stWebDesigner* [online]. 2016 [cit. 2017-04-15]. Dostupné z: https://1stwebdesigner.com/power-sass-why-use-css-preprocessors/

mezi prohlížeči[48]. Díky automatické kompilaci zápisu do dalších prefixů pro jednotlivé prohlížeče je možné se více zaměřit na psaní stěžejního kódu a nechat preprocesor zajistit potřebnou kompatibilitu, což má mimo jiné vliv na rychlost vývoje.

5.4 Vliv na kvalitu kódu

Kromě vlivu na rychlost pracovních postupů můžeme CSS preprocesory posuzovat také z pohledu kvality, možnosti jednoduché správy a přehlednosti kódu.

Syntaxe preprocesorů kromě jazyka LESS, jehož prioritou je rozšíření deklarativní CSS filosofie, je naopak imperativního a objektově orientovaného charakteru[49], což umožňuje vyhnout se psaní stejného či velmi podobného kódu. Tím se snižuje objem souboru a mimo jiné i jeho přehlednost.

Při vytváření webu a přemýšlení nad správnou architekturou a škálovatelností je nutné se soustředit nad veškerým zápisem CSS kódu tak, aby bylo možné v budoucnu zavést například responzivní design, nové breakpointy, měnit barevnou identitu, layout a další rozšíření a změny, které se pojí s rostoucím webem[50]. I přes zavedení všech metod, mezi které patří OOCSS nebo BEM, bude flexibilita stále omezena na schopnost vyhledat a nahradit kód.

Čím je webová aplikace složitější, tím je její CSS rozsáhlejší, komplexnější a vyžadující údržbu. Díky zanořování, mixinům nebo importu lze kód modularizovat do menších logických fragmentů kódu, které lze distribuovat do souborů, podobně jako knihovny. O takto modularizovaném kódu lze přemýšlet jako o stavebních kamenech. Navíc díky chytrému importu nedojde ke kompilaci veškerého CSS, ale pouze užívaného.

Responzivní design v CSS vzniká zanořováním kódu do media queries, které se umisťují na konec souboru[51]. V Sass je způsob tvorby responzivního designu více integrovaný a to pomocí zanoření do konkrétního bloku elementu, ke kterému se media

[48] Why do some people use CSS preprocessors? Quora [online]. 2014 [cit. 2017-04-15]. Dostupné z: https://www.quora.com/Why-do-some-people-use-CSS-preprocessors
[49] Průvodce CSS preprocesory: který vybrat? *Vzhůru dolů* [online]. [cit. 2017-03-12]. Dostupné z: http://www.vzhurudolu.cz/blog/15-css-preprocesory-4
[50] Why do some people use CSS preprocessors? Quora [online]. 2014 [cit. 2017-04-15]. Dostupné z: https://www.quora.com/Why-do-some-people-use-CSS-preprocessors
[51] 6 Reasons Sass Should Be In Every Front-End Developer's Toolbox. *Upwork* [online]. 2016 [cit. 2017-04-15]. Dostupné z: https://www.upwork.com/hiring/development/reasons-to-use-sass-css-preprocessor/

query vztahuje. Tento přístup je mnohem čistější, jelikož jsou breakpointy díky možnosti zanořování obsaženy přímo v bloku konkrétního elementu. Preprocesory tak usnadňují vytvářet responzivní design lépe.

CSS preprocesory také disponují mnohými vestavěnými např. matematickými funkcemi nebo možností si funkce vytvářet sami a reagovat tak na různé stavy, které mohou v prostředí webových aplikacích nastat.

6 Závěr

Každý CSS preprocesor, který byl v této práci pokryt (LESS, Sass a Stylus) dává vývojářům možnost využívat běžně nepodporované funkce a zároveň zachovat kompatibilitu kódu s prohlížečem a jeho čistotu. Preprocesory jsou navrženy tak, aby pomáhaly vytvářet strukturovaný a organizovaný kód, který je snadnější na správu a zároveň tak, aby ušetřily čas při jeho vývoji, což je v dlouhodobém horizontu užitečné pro každého vývojáře. CSS preprocesory proto stojí za to začlenit do pracovního postupu přes počáteční čas, který je nutné investovat do naučení a osvojení.

Na jedné straně jsou si preprocesory v základu velmi podobné. Na druhé straně má každý svoji ideologii a přístup k vykonávání stejných úkolů, což se projevuje v drobných odlišnostech mezi pokročilými funkcemi. Ve výsledku rozdíly nejsou natolik zásadní, aby nebylo možné v průběhu zvolit jiný. Tomu přispívá existence online nástrojů pro převod kódů mezi syntaxemi zmíněných preprocesorů. Výběr vhodného CSS preprocesoru je proto otázkou osobních preferencích, které závisí i na konkrétním projektu – co je pro něj typické, jak je náročný, jak jej může ovlivnit zvolený preprocesor a zda na projektu pracuje jediný člověk nebo více kolaborantů. Pokud přispívá do společného souboru více vývojářů, může být Stylus v rámci týmu výhodnější alternativou, vzhledem k jeho otevřené syntaxi, kterou lze v rámci souboru míchat.

Výběr konkrétního preprocesoru tedy není nejdůležitějším aspektem implementace preprocesoru do pracovního postupu, pokud se nejedná o práci s velmi unikátní sadou požadavků, při kterých by zvolený preprocesor mohl znemožnit vytvoření požadovaných stylů. Při výběru vhodného preprocesoru je nutné brát v potaz úroveň znalostí uživatele či celého týmu a současně je nutné zohlednit i jejich časové možnosti pro začlenění preprocesorů do pracovních postupů.

Použitá literatura

65+ SASS Mixins Library. *CSS Author* [online]. 2016 [cit. 2017-04-11]. Dostupné z: http://www.cssauthor.com/sass-mixins-library/

6 Reasons Sass Should Be In Every Front-End Developer's Toolbox. *Upwork* [online]. 2016 [cit. 2017-04-15]. Dostupné z: https://www.upwork.com/hiring/development/reasons-to-use-sass-css-preprocessor/

A Quick Guide To Modular CSS – Don't Be Petty, Write Less Spaghetti!. *The Raygun Blog* [online]. 2016 [cit. 2017-04-15]. Dostupné z: https://raygun.com/blog/modular-css/

An Introduction to CSS Pre-Processors: SASS, LESS and Stylus. *HTML Mag* [online]. 2014 [cit. 2017-04-15]. Dostupné z: https://htmlmag.com/article/an-introduction-to-css-preprocessors-sass-less-stylus

Built-In Funstions. *Stylus* [online]. [cit. 2017-03-12]. Dostupné z: http://stylus-lang.com/docs/bifs.html

CEDERHOLM, Dan. Sass for Web Designers. New York: Book Apart, 2013. ISBN 978-1937557126.

Coder. *Dictionary.com* [online]. [cit. 2016-11-24]. Dostupné z: http://www.dictionary.com/browse/coder

CSS PREprocessors [online]. [cit. 2017-04-11]. Dostupné z: https://csspre.com/convert/

Frameworks Using Less. Less.js [online]. [cit. 2017-03-12]. Dostupné z: http://lesscss.org/3.x/tools/#frameworks-using-less-mixin-libraries

Functions. *Less.js* [online]. [cit. 2017-03-12]. Dostupné z: http://lesscss.org/functions/

Getting started. Less.js [online]. [cit. 2017-04-11]. Dostupné z: http://lesscss.org/

Historie a vývoj HTML. *Kosek.cz* [online]. [cit. 2016-11-24]. Dostupné z:

http://www.kosek.cz/vyuka/4iz228/prednasky/uvod/foilgroup03.html

Jaké breakpointy zvolit v responzivním webdesignu? *Vzhůru dolů* [online]. 2013 [cit. 2017-04-11]. Dostupné z:

http://kratce.vzhurudolu.cz/post/46416507703/jak%C3%A9-breakpointy-zvolit-v-responzivn%C3%ADm-webdesignu

JINDA, Martin. *CSS preprocesory*. České Budějovice, 2014. Bakalářská práce. Jihočeská univerzita v Českých Budějovicích.

KOSEK, Jiří. *HTML: tvorba dokonalých WWW stránek : podrobný průvodce* [online]. Praha: Grada, 1998 [cit. 2016-11-24]. Průvodce (Grada). ISBN 80-716-9608-0. Dostupné z: http://www.kosek.cz/html/html-tvorba-dokonalych-www-stranek.pdf

Layout: Grid system. *Bootstrap* [online]. [cit. 2017-03-12]. Dostupné z: https://v4-alpha.getbootstrap.com/layout/grid/

LESS: stejné CSS za méně peněz. *Zdroják.cz* [online]. [cit. 2017-03-12]. Dostupné z: https://www.zdrojak.cz/clanky/less-stejne-css-za-mene-penez/

Less.js Plugins. *Less.js* [online]. [cit. 2017-03-12]. Dostupné z: http://lesscss.org/3.x/tools/#plugins

Leveraging Sass mixins for cleaner code. *The Sass way* [online]. 2011 [cit. 2017-04-15]. Dostupné z: http://thesassway.com/intermediate/leveraging-sass-mixins-for-cleaner-code

Loops. *CSS PREprocessors* [online]. 2014 [cit. 2017-04-15]. Dostupné z: https://csspre.com/loops/

Loops in CSS Preprocessors. *CSS-Tricks* [online]. 2016 [cit. 2017-04-15]. Dostupné z: https://css-tricks.com/loops-css-preprocessors/

MALHOTRA, Avinash. *Tech Altum Tutorial* [online]. [cit. 15.4.2017]. Dostupný na WWW: http://tutorial.techaltum.com/htmlTags.html

MCFARLAND, David Sawyer. *CSS: chybějící manuál : tvorba nádherných webových stránek prostřednictvím CSS*. Praha: Grada, c2007. ISBN 978-80-247-2122-4.

MICHÁLEK, Martin. Co by měl umět webový kodér? In: *Vzhůru dolů* [online]. [cit. 2016-11-24]. Dostupné z: http://www.vzhurudolu.cz/prirucka/webovy-koder

Nib – CSS3 extensions for Stylus [online]. [cit. 2017-04-11]. Dostupné z: http://tj.github.io/nib/

Poll Results: Popularity of CSS Preprocessors. *CSS-Tricks* [online]. 2012 [cit. 2017-04-15]. Dostupné z: https://css-tricks.com/poll-results-popularity-of-css-preprocessors/

Průvodce CSS preprocesory: který vybrat? *Vzhůru dolů* [online]. [cit. 2017-03-12]. Dostupné z: http://www.vzhurudolu.cz/blog/15-css-preprocesory-4

Sass Basics: The Function Directive. *SitePoint* [online]. 2015 [cit. 2017-04-15]. Dostupné z: https://www.sitepoint.com/sass-basics-function-directive/

Sass Documentation. *Sass (Syntactically Awesome StyleSheets)* [online]. [cit. 2017-03-12]. Dostupné z: http://www.sass-lang.com/documentation/Sass/Script/Functions.html

Set Up CSS and JS Preprocessors. *Google Developers* [online]. 2017 [cit. 2017-04-15]. Dostupné z: https://developers.google.com/web/tools/setup/setup-preprocessors

SPURLOCK, Jake. *O'Reilly* [online]. [cit. 15.4.2017]. Dostupný na WWW: https://www.safaribooksonline.com/library/view/bootstrap/9781449344573/ch01.html

The CSS saga. *W3C* [online]. [cit. 2017-03-12]. Dostupné z: https://www.w3.org/Style/LieBos2e/history/

The Power of Sass and Why You Should Embrace CSS Preprocessors. *1stWebDesigner* [online]. 2016 [cit. 2017-04-15]. Dostupné z: https://1stwebdesigner.com/power-sass-why-use-css-preprocessors/

The web standards model - HTML CSS and JavaScript. *W3C* [online]. [cit. 2017-03-12]. Dostupné z: https://www.w3.org/community/webed/wiki/The_web_standards_model_-_HTML_CSS_and_JavaScript

Vzhůru dolů [online]. [cit. 2017-04-11]. Dostupné z: http://www.vzhurudolu.cz/

W3C [online]. 2016 [cit. 2016-11-24]. Dostupné z: https://www.w3.org/

What are the best CSS preprocessors/postprocessors? *Slant* [online]. [cit. 2017-03-12]. Dostupné z: https://www.slant.co/topics/217/~best-css-preprocessors-postprocessors

What is loop? *Whatis.com* [online]. 2005 [cit. 2017-04-15]. Dostupné z: http://whatis.techtarget.com/definition/loop

When to use @extend; when to use a mixin. *CSS Wizardry* [online]. 2014 [cit. 2017-04-11]. Dostupné z: https://csswizardry.com/2014/11/when-to-use-extend-when-to-use-a-mixin/

Why are there still people alive today that don't use CSS Preprocessors ? *Hashnode* [online]. 2016 [cit. 2017-04-15]. Dostupné z: https://hashnode.com/post/why-are-there-still-people-alive-today-that-dont-use-css-preprocessors-cipr6r3dc002dz753y1pxgqwf

Why do some people use CSS preprocessors? *Quora* [online]. 2014 [cit. 2017-04-15]. Dostupné z: https://www.quora.com/Why-do-some-people-use-CSS-preprocessors

Seznam obrázků

ANOTACE

Jméno a příjmení:	Eliška Plitzová
Katedra:	Katedra technické a informační výchovy
Vedoucí práce:	Mgr. Jan Kubrický, Ph.D.
Rok obhajoby:	2017

Název práce:	CSS preprocesory – jejich srovnání a doporučení pro výběr
Název v angličtině:	CSS preprocessors – their comparison and recommendations for selection
Anotace práce:	Tato bakalářská práce má za cíl porovnat nejrozšířenější CSS preprocesory LESS, Sass a Stylus, demonstrovat jejich rozdílné způsoby zápisu kódu, poukázat na jejich benefity a usnadnit proces výběru vhodného preprocesoru pro tvorbu moderních webových prezentací. CSS preprocesory umožňují uživateli rychlejší zápis kódu a to zejména díky zanořování, matematickým funkcím, proměnným, objektově orientovanému a deklarativnímu či imperativnímu stylu zápisu kódu, který se blíží programování. Výsledkem je také objasnění, proč je vhodné CSS preprocesory používat, jaké výhody nabízí oproti čistému CSS a jaký dopad může mít jejich správné začlenění do pracovního postupu.
Klíčová slova:	HTML, CSS, kaskádové styly, CSS preprocesor, LESS, Sass, Stylus
Anotace v angličtině:	The aim of this bachelor thesis is to compare most popular CSS preprocessors LESS, Sass and Stylus, demonstrate their different approach to writing code, highlight their benefits and to facilitate the process of selecting a suitable preprocessor for creating modern

	web presentations. CSS preprocessors allow the user to write faster code, mainly by nesting, mathematical functions, variables, object-oriented, declarative or imperative code writing style that is approaching programming. The result is also a clarification of why it is appropriate to use CSS preprocessors, what benefits they offer compared to pure CSS, and what impact their proper integration into the workflow might have.
Klíčová slova v angličtině:	HTML, CSS, cascade style sheets, CSS preprocessor, LESS, Sass, Stylus
Přílohy vázané v práci:	
Rozsah práce:	61
Jazyk práce:	čeština