***SASS***

SASS to tak naprawdę css z nowymi możliwościami. Wszystko napisane w SASS jest przetwarzane przez tzw. pre-procesor ( z ang. procesor to przetwarzanie na kod CSS), przedrostek pre oznacza, że cały proces kompilacj wykonuje się przed wysłaniem kodu na server. Dzięki temu nie tracimy nic na wydajności. SASS dodatkowo pozwala rozłożenie jednego pliku na moduły/części.

SASS oraz LESS – dwa najpopularniejsze pre-procesory. Główne zalety obu to:

- generują błędy gdy takowe się pojawią

- mają algorytmu kompresujący automatycznie napisany przez nas styl – co sprawia, że plik css jest mniejszy i szybciej wczytuje się strona. – w jednej linii, wszystko obok siebie

- możemy mieć w tym samym momencie skompresowaną wersję css, oraz wersję dla nas zrozumiałą – przyjazną dla oka.

- można modularyzować kod CSS, przez co szybko można znaleźć daną część kodu. – możemy rozłożyć na: nagłówek osobno, stronę osobno, jakąś podstronę osobno. Możemy mieć nawet 30, czy więcej modułów, ale sa one kompilowane do jednego i nie trzeba wykonywać 30 requestów do serwera!

- mają kompilatory online oraz posiadają dobrą dokumentację

**Plusy SASS względem LESS:**

- posiada bibliotekę compass, która zawiera dużo przydatnych narzędzi i funkcji (np. sprites – czyli jeden obrazek z wielu małych obrazków)

- ma więcej zaawansowanych możliwości niż LESS

**Minusem SASS** względem LESS jest wymaganie tyczące się programu symulującego – np. Ruby environment

**Plusy LESS:**

- odpala się po stronie klienta z pomocą js – jednak jest to też minus, bo strona wczytuje się wolniej – kompresja dzieje się na bieżąco co spowalnia wczytywanie. Na szczęście jest możliwość kompilacji jak to robi SASS – czyli offline dzięki czemu nie będzie straty czasu.

- LESS zwraca łatwiejsze do zrozumienia powiadomienia o błędach

- LESS ma namespace – czyli przestrzeń nazw, co pozwala zwiększyć organizację struktury

**Minusem LESS** jest to , że nie może wykonywać operacji arytmetycznych na dwóch różnych typach, natomiast SASS może. Oba jednak mogą wykonywać same operacje.

***Instalacja SASS:***

Aby zainstalować sass, wpierwszej kolejności potrzebujemy JDK – JAVA DEVELOPMENT KIT oraz edyto tekstu – w naszym przpadku netbeans wystarczy. Nastpenie potrzeujemy Ruby – język programowania do tworzenia np. pre-procesorów. Jest to tak naprawdę środowisko, które umożliwia pracę z SASS. Ruby stworzył coś takiego jak keljnot, który jest paczką dla SASS. Pobieramy zatem Ruby, ale nie development kit, gdyż nie chcemy programować w tym języku, a jedynie środowisko, które umożliwi pracę z SASS. Po ściągnięciu i zainstalowaniu środowiska Ruby przechodzimy do cmd(wiersz poleceń) i będziemy instalowali SASS’a. Wskazówki możemy znaleźć na stronie sass-lang.com/install. Otierwamy wiersz poleceń i podajemy komendę nawiązującą do klejnotu Ruby: gem install sass (gem z ang. Klejnot). Aby instrukcja gem zadziałała, trzeba podczas instalacji Ruby zainstalować, aby path był znaleziony.

Dzięki komendzie gem install sass, zostaje to dodane do środowiska systemowego – jest ot zmienna środowiskowa – jest dostępne dla całego windowsa w każdym podfloderze. Jeżlei nie posiadamy pełnych praw – należy użyć komendy sudo gem install sass.

W celu sprawdzenia czy zostało zainstalowane – moemy wpisać w command prompt: sass -v

Kolejnym krokiem jest konfiguracja SASS w edytorze (w naszym przypadku netbeans). Tworzymy nowy projekt (może to być aplikacja html5) i nadajemy mu nazwę np. SASS. Po stworzeniu projektu przechodzimy do konfiguracji: zaznaczamy nasz projekt w window explorer i prawym przyciskiem wybieramy properties. Jedym z podkunktów jest CSS preprocessors. W oknie CSS preprocessors wybieramy przycisk Configure Executeables – musimy podać gdzie nasz SASS się znajduje i jak go uruchomić. Kolejno wybieramy w nowym oknie dla SASS path browse i wybieramy ścieżkę dla SASS. W oknie wyboru cofamy się do dysku C i tam powinien być (domyślnie) folder z Ruby. Wybieramy go i następnie wchodzimy w folder bin. W folderze tym powinien znajdować się plik sass.bat – jest to plik, który został zainstalowany za pomocą komendy w cmd. Kolejno akceptujemy poprzez apply. Ponownie znajdujemy się w oknie CSS preprocessors i zaznaczamy Compile SASS Files on Save. I teraz kiedy stworzymy folder scss a w nim plik o rozszerzeniu scss, to zostanie on przetworzony na plik css automatycznie. Po zaakceptowaniu zostaje już tylko stworzenie folderu SCSS oraz folderu CSS, a w folderze SCSS tworzymy plik o nazwie np. style i rozszerzeniu scss. Plik css zostanie automatycznie stworzony (dokona się automatyczna kompilacja). Poza plikiem css tworzony jest również plik css.map

Teraz możemy zaobserwować, ze cokolwiek co zostanie zapisane w pliku style.scss, zostanie przekompilowane do pliku style.css. Możemy w pliku scss zdefiniować zmienną $sizeOfHeader i przypisać od niej wartość 16px: $sizeOfHeader = 16px; Wyskoczy nam błąd, ale aby go zanalizować wybieramy na górze programu zamiast google chrome embedded webkit browser. Zmieniamy po prostu gdzie ma być odpalany nasz index.html. Dzięki temu odpalimy nasze program nie w przeglądarce a w netbeansie. Mamy dzięki temu podręczny kompilator, który jeżeli napotka jakiś błąd, to wyrzuca go w SCSS(compile) wraz z opisem tego błędu. Błąd podaje nam informacje, ze zamiast znaku = powinien być zastosowany znak : aby zdefiniować zmienną. Następnie możemy podstawić do definicji np. h1 nasza zmienną, która zostanie przypisana do font-size:

$sizeOfHeader: 16px;

h1{ font-size: $sizeOfHeader – 5px; }

Co skompiluje się do CSS :

h1{ font-size: 11px; }

Istnieje wiele stron internetowych, które pozwalają skompilować scss do css: np. sassmeister.com. Mamy tam trzy okna – scss, css oraz html. Możemy dzięki temu sprawdzć co się na bieżąco zmienia podczas definicji w SASS.

***Konfiguracja ręczna SASS***

Korzystając z innych edytorów (IDE) czy też nawet z notatnika, konfiguracja SASS przebiega inaczej niż w przypadku netbeans’a. Tworzymy w pierwszej kolejności nowy folder w miejscu gdzie jest nasz folder SASS (floder zawierający projekt sass), nazywamy go np. SASS MANUALLY. W folderze tym potrzebujemy stworzym 2 foldery – CSS oraz SCSS. Bierzemy z naszego wcześniejszego projektu styl.scss, wklejamy go do nowego folderu scss i otwieramy w notepadzie. Chcemy teraz zrobić, aby nasz plik css tworzył się automatycznie. Wchodzimy w start i otwieramy wiersz poleceń. Zmieniamy ścieżkę dostępu na miejsce gdzie jest folder scss oraz css (w folderze SASS Manually). Podajemy komendę cd i następnie po spacji ścieżkę do naszych dwóch folderów. Kolejno możemy przy użyciu komendy dir sprawdzić zawartość.Wchodzimy do folderu scss poprzez komendę „cd scss” i nastęnie chcemy skompilować plik. Wzorując się na dokumentacji trzeba podać: sass input.scss output.css. Wpisujemy zatem następującą komendę do wiersza poleceń: sass style.scss ../css/style.css

Wyrażenie ‘’../’’ powoduje że w command line schodzimy jeden poziom niżej (cofamy się o jeden poziom). Po tym wyrażeniu dajemy znak / i podajemy folder, do kórego chcemy wejść – czyli css, następnie znowu znak / i nazwę pliku, który ma być stworzony. Po wciśnięciu entera w command prompt, zostanie stworzony plik css w folderze css. Konfiguracja manualna została zatem przeprowadzona – nie jest ona jednak automatyczna. Aby zmiana była wykonywana automatycznie w pilku css, zawsze kieyd zajdzie zmiana w plik scss, trzeba ustawić flagę watch. Służy do tego komenda:

‘’sass –watch input.scss:output.css” – jest to ogólny wyglad komendy, który u nas przyjmuje postać:

sass –watch style.scss:../css/style.css - nasz output to ścieżka dostępu poprzez ../css/style.css

Po deklaracji takiej komendy powinniśmy otrzymać komunikat: Sass is watching for changes, to stop press ctr+c. Jest zatem teraz ustawiona automatyczna kompilacja do css.

***Co to source map***

Podczas kompilacji z scss do css generuje się dodatkowo coś takiego jak mapa (plik o rozszerzeniu css.map, map z ang. Przyporządkować). Mapy pozwalają w dużym stopniu ułatwić szuanie danych elementów w celu ich edycji. Sam plik mapa zawiera informację o wersji, źródle, pliku czy też nazwie. Zawiera ona też coś takiego jak mapowanie – pozwala to na odniesienie się do danego elementu, lecz nie do pliku css a pliku scss. A zatem kiedy w naszym oknie pobocznym (wyświetlonym po kliknieciu na danym element w oknie Browser DOM) z css properties wyświetlony jest element i odnośnik do niego, to jeżeli zawieramy mape, to zostaniemy przekierowani do pliku scss i tegoż elementu. Znajduje to zastosowanie, kiedy plik scss jest bardzo duży – możemy wtedy szybko odnaleźć dane formatowanie elementu.

***Sposoby przetwarzania CSS przez SASS***

W pliku SASS możemy dokładnie określić jak ma wyglądać plik css. Chcemy bowiem, aby był on jak najmniejszy bez zbędnych komentarzy i spacji. W dokuemntacji SASS na stronie możemy wejść w zakładkę output style, co określa nam styl wyjściowy pliku. Możemy ustawić coś takiego jak

.nested – styl domyślny

.expanded – styl najbardziej czytelny

.compact – cos pomiędzy całkowitą kompresją a nested

.compressed – najbardziej skompresowany

Domyślnie ustawiony jest styl .nested. Aby zmienić styl, wchodzimy w nasz projekt (zanzaczamy go myszką) i po kliknieciu prawym przyciskiem myszy wybieramy properties. Nastpenie wchodzimy w zakąłdkę css preprocessors i tej zakładcę mamy pole o nazwie Compiler options. W polu tym wpisujemy „--style compressed” i klikamy ok. Nasz plik css jest teraz skompresowany i klikając na ten plik w oknie exploratora i wybierając properties możemy podejrzeć jego wielkość.

W celu porównywania ustawiamy styl na expanded, aby był jak najbardziej czytelny – jednak jest to tylko zabieg treningowy i w noralnych projektach używa się compressed!!!

***Importowanie plików SASS***

Możliwość importowania różnego rodzaju plików do styli.scss pozwala zwiększyć czytelność i czystszość kodu oraz daje modularność – podzial na części. Nie tworzymy jednak kilku plików css, bo zwiększałoby to czas wczytywania się strony. Tworzymy zatem Wiecej plików SASS (SCSS). Jednak gdy stworzymy normalnie pliki, to zostaną one automatycznie przekompilowane na na css w folderze CSS. Aby tem zapobiedz, wystarczy podczas nadawania nazwy na początku podać podkreślnik (\_). Daje to info, aby plik ten nie był automatycznie kompilowany na CSS. Majac dwa plik scss, chcemy zaimportować dane z jednego do drugiego. Aby to wykonać, wystarczy napisać w pliku, do którego importujemy:

@impotr ‘’\_nazwaPlikuScss.scss’’;

Dzięki temu mamy dostęp do zmiennych i innych elementów pliku nazwaPlikuScss.scss w naszym pliku – np. w głównym pliku scss.

Daje to możliwość szybszego znalezienia szukanego do edycji elementu poprzez modularność. Wszystkie pliki scss i tak w wynik końcowym są jednym plikiem css, co jest dużą zaletą.

***Zmienne***

Zmienna to pojemnik do przechowywania wartości. Wartość tą możemy w szybki sposób zmienić. Zmienna tworzymy według schematu: dolar($), kolejno nazwaZmienne, potem dwukropek i następnie po spacji wartość zmiennej i średnik na końcu. Podczas nazywania zmiennych należy pamiętać, aby nazwa jasno określała role zmiennej. Poza zwykłą definicja zmiennej możemy również przypisywać do zmiennych zmienne, jeżeli których wartość się zmieni, to zmienne, do których sa przypisane, również się zmienią. Przykład zmennych:

$sizeOfHeader: 20px;

$color-orange: orange; //Podczas nazywania zmiennych sposujemy camelCase bądź "-" odzielające elementy nazwy

$color-red: red;

$header-color: $color-orange;

$link-color: $color-red;

I kolejno w pliku style.scss importujemy plik \_variables.scss dzięki czemu mamy dostęp do zmiennych:

@import "\_variables.scss";

h1{

font-size: $sizeOfHeader;

color: $header-color;

}

h2{

font-size: $sizeOfHeader - 5px;

color: $header-color;

}

a{

color: $link-color;

}

A zatem tak naprawdę nasz plik \_variables.scss to plik konfiguracyjny.

***Style zagnieżdżone – przykład na rozwijanym menu SASS***

Kiedy w pliku css definiujemy style, są one określane w następujący sposób:

nav ul{

margin: 0;

padding: 0;

list-style: none;

}

I kolejno kiedy chcemy teraz przejść do list item, aby go sformatować, ponownie definiujemy nav ul li:

nav ul li{

...

}

Dzięki SASS możemy zagnieżdżać style, odwołując się w danym bloku do kolejnego elementu będącego niżej:

nav ul{

margin: 0;

padding: 0;

list-style: none;

li{

display: block;

float: left;

}

}

Powyższy kod jest równoważny z kodem CSS umieszonym wyżej. Moglibyśmy rozbić również ul, tak jak zrobiliśmy z li, ale czy byłoby to poprawne. W tym wypadku nie byłoby problemu, ale zawsze trzeba widzieć do czego się odnosimy. Jeśli byśmy to rozbili na nav { ul{} li {} }, to w css ładowanym na server i do klienta uzyskalibyśmy coś takiego:

nav ul{ ….. }

nav li{ ….. }

W więc brak precyzyjneg określenia, że li mają być tylko z unordered list i jeżlei byłaby lista numerowana, to również jej list item byłyby wzięte pod uwagę. Trzeba się zatem pilnować.

W naszym przypadku jest to ok, gdyż chcemy wybrać wszystkie li.

Kolejno przechodzimy do formatowania elementów – chcemy, aby list item, które znajdują się w list item były domyślnie schowane i wyświetlanej jako block, żeby nie opływały po lewej (zamiast float: left; dajemy float: none;) oraz, żeby po najechaniu elementy te pojawiały się. Znajduąc się w li wchodzimy za pomocą zagnieżdżenia do dalszego li. Kiedy sformatujemy li zostaje nam ustawienie hovera, aby to zrobić musimy odwołać się do wyżej położonego elementu czyli tak jakby li ojca. Robimy to za pomocą znaczka & i kolejno dajemy :hover li. Drugir li musimy dodać, gdyż chcemy, aby zminanie uległo li znajdujące się li, po najechaniu na to li. A zatem kod wygląda następująco:

nav {

ul{

margin: 0;

padding: 0;

list-style: none;

}

li{

display: block;

float: left;

li{

display: none;

float: none;

}

&:hover li{

display: block;

}

}

}

Kolejno zajmujemy się anchorami – ustawiamy dla wszystkich display jako block ( dla wszystkich czyli a{} umieszczamy tylko w nav, większe zagnieżdżenie nie jest potrzebne). Także chcemy aby po najechaniu na a zmienił się kolor, więć w stylu a umieszczamy odwołanie do rodzica poprzez & i kolejno :hover. Wewnątrz powinnismy zmienić kolor, ale zgodnie z zasadą modułowości, powinniśmy odwołać się do zmiennej, która jest zdefiniowana wewnątrz innego pliku i zaimportowana (w pliku variables).

nav{

….

a{

display: block;

padding: 6px 30px;

text-decoration: none;

background-color: $link-nav-bg-color-default;

&:hover{ //odwołujemy sie do rodzica, czyli do a i po najechaniu na a zmieniamy coś

color: $link-nav-color-hover;

}

}

}

Ponownie &:hover odnosi się do parenta, czyli w tym przypadku do a (do każdego a w nawigacji) i po najechaniu na to a zostaje zmieniony kolor za pomocą zmiennej zdefiniowanej w pliku \_variables.scss

Chcemy teraz aby po najechaniu zmieniało się tło ale tylko tych anchorów, które są wewnątrz li li.

nav {

ul{

margin: 0;

padding: 0;

list-style: none;

}

li{

display: block;

float: left;

li{

display: none;

float: none;

a:hover{ //kiedy ktoś najedzie na link (a) zawarty wewnatrz li wewnątrz li)

background-color: $link-nav-bg-color-hover;

}

}

&:hover li{ // tak odnosimy się poprzez & do wyższego li i kolejno po najechaniu na nie jeżeli jest w nim li, to wyświetli sie jako block

display: block;

}

}

a{

display: block;

padding: 6px 30px;

text-decoration: none;

background-color: $link-nav-bg-color-default;

&:hover{ //odwołujemy sie do rodzica, czyli do a i po najechaniu na a zmieniamy coś

color: $link-nav-color-hover;

}

}

}

***MIXINS***

Mixin to wstawka. Wstawki robimy w następujący sposób: piszemy znak @ i kolejno mixin, następnie nazywamy naszą wstawkę (np. text-overflow), następnie dajemy nawiasy () i kolejno po spacji w nawiasach klamrowych definiujemy ciało wstawki, które chcemy wstawić w jakieś miejsce używając słówka include. Tworzą wstawki, możemy w każdej chwili je wstawić do danego kodu, bez konieczności pisania od początku formatu dla danego elementu – wystarczy wstawkę wrzucić. Tworzymy zatem nowy plik o nazwie np. \_mixins.scss, gdzie będziemy tworzyć nasze wstawki. Należy pamiętać o znaku podłogi\_ podczas nazywania pliku, aby nie został skompilowany do pliku csss od razu, co stworzyłoby dodatkowy plik css, a tego nie chcemy.

W myśl powyższego podziału, możemy stworzyć również nowy plik o nazwie \_nav.scss i wrzucić tam nasz styl tyczący się nawigacji.

Następnie stworzone pliki trzeba zaimportować do pliku głównego scss, poprzez @import „\_nazwaPliku”

Następnie zabieramy się za tworzenie wstawek. Najpierw stworzymy sobie jakiś akapit o klasie „akapit” w html, aby móc go formatować za pomocą wstawki.

Kolejno przechodziy do sass i definiujemy styl:

.akapit{

width: 200px;

height: 50px;

overflow: hidden;

white-space: nowrap; //nie zawijaj - wyświetli nam wszystko w jednej linii bez wzgledu na entery czy spacje

text-overflow: ellipsis; //daje nam kropki

}

Powyższy kod nie zawiera wstawki, jednak text sformatowaliśmy, aby się nie wyświetlał cały i nie był zawijany. Teraz wycinamy część kodu i wrzucamy ją do naszego pliku \_mixins, gdzie definiujemy następującą wstawkę :

@mixin textOverflow(){

overflow: hidden;

white-space: nowrap; //nie zawijaj

text-overflow: ellipsis; //daje nam kropki

}

A w pliku głównym wstawiamy wstawkę za pomocą @include i nazwaWstawki:

.akapit{

width: 200px;

height: 50px;

@include textOverflow;

}

Mixins mogą również przyjmować argumenty w nawiasach (). Argumenty te mogę w nawiasach przyjmować wartości domyśle, które zostaną uzyte, jeżeli nie zostanie podana przez nas wartość. Na przykład kiedy tworzymy gradient, to możemy zdefiniować początek i koniec gradientu (ich kolory):

@mixin horizontal(&startColor: $white, &endColor: $grey){I tutaj kod do gradient}

Kolejno wystarczy definiować kolory, albo podać własne podczas wywoływania wstawki:

.akapit{

width: 200px;

height: 50px;

@include textOverflow;

@include horizontalGradient(#FFC, #FF0);

}

Jeżeli mamy tylko podane argumenty w mixinie, to konieczne jest podanie wartości danych argumentów podczas wywoływania mixina. Jeżeli jednak są zdefiniowane wartości domyślne w mixinie, to nie jest konieczne podawanie wartości podczas wywoływania mixina jako wstawki w pliku głównym.

Tworzenie gradientu wymaga zdefiniowania dla każdej przeglądarki vendor prefixu, aby gdarient ten działa pod każdą z nich. Jednak nie jest to najlepsze rozwiązanie używając mixins. Vendor prefixy są już powoli przeżytkiem, gdyż większość przeglądarek wspiera większość elementów. Nie wiemy bowiem czy pisząc kod i używając vendor-prefixów, są one potrzebne czy już nie i czy np. za rok nie będą kompletnie potrzebne,a wyjdzie zapotrzebowanie na inne. Radą na to jest autoprefixer, który jest tak zwanym postprocesorem. Postprocesor jeszcze raz na końcu bierze css i przetwarza go na nowy css. Postprocesor autoprefixer tworzy nowy plik css, który jest tworzony względem strony caniuse. Autoprefixer sprawdza czy dany vendor prefix jest wymagany i jeżeli nie, to pomimo że go napisaliśmy, usuwa go i następnie przetwarza do pliku końcowego. Przetwarzanie to jest wykonywane na naszym komputerze, zatem nie tracimy na prąkości wczytywania strony, wysyłamy tylko plik wynikowy.

Istnieją również inne sposoby na vendor prefixy, takie jak np. dodanie pliku js, który za nas dodaje prefixy. Jest to jednak mniej efektywne – zwiększa czas pobierania przez użytkownika plików (wczytywania strony) oraz może wspierać przeglądarki, których nie chcielibyśmy wspierać. Możemy poza plikiem JS czy też postprocesorte autoprefixer skorzystać z biblioteki compass, która jest taki jakby zbiorem takich mixinów i daje duże możliwości.

***Funkcjonalność extended***

Podczas tworzenia styli dla kilku elementów często większość styli (kodu) względem wszystkich elementów będzie sie powtarzała. Inne będzie jedynie obramowanie czy też kolor. Aby nie powtarzać tekstu, wystarczy stworzyć tak jakby nową klasę w pliku scss i do niej wrzucić powtarzający się kod, a następnie w poszczególnych elementach rozszerzyć style tychże elementów o daną klasę (wcześniej utworzoną) poprze słówko @extend .nazwaKlasy; Przykładem są np. akapity, który dajemy takie same style, różnią się one jedynie kolorem obramowania (na przykład). Akapity mają odpowiednio klasy akapit1, akapit2, akapit3. Wyglada to następująco:

.akapit{ tutaj dajemy style, które się powtarzały – czyli np. font-size: 12px; itd. }

.akapit1{

border: 1px solid black;

@extend .akapit ;

}

.akapit2{

border: 1px solid orange;

@extend .akapit ;

}

.akapit3{

border: 1px solid red;

@extend .akapit ;

}

Dzięki słówku @extend rozszerzamy nasze elementy o inna klasę, a tym samym dodajemy inne style bez konieczności stosowania mixins, co również można by zrobić.

W extend nie możemy natomiast przesyłać zmiennych i tym podobnych.

Jednak kiedy się przyjrzymy, nie jest to dalej aż tak efektywne jak mogłoby być, ponieważ klasa akapit zabiera miejsce i jest również interpretowana w CSS. Zamiast klasy warto zatem utworzyć placeholdera – czyli miejsce do przechowywania czegoś – w ty przypadku właściwości. Będzie on zatem tylko używany do rozszerzenia danego selektora o właściwości przechowywane w tymże placeholderze. Różni się on od klasy jedynie znakiem – zamiast kropki stawiamy znak %, zarówno przy deklaracji placeholdera, jak i przy odwołaniu: @extend %akapit;

Placeholdery są bardzo podobne do mixinów – różnią się jedynie tym, ze w mixinach jest możliwość przesyłania zmiennej.

Kolejną czynnością jaką chcemy wykonać to wyśrodkować nasz akapit oraz rozsuwane menu. AByt to zrobić normalnie zadeklarowalibyśmy display: block oraz margin: 0 auto; dla akapitu oraz podobnie dla nav (dodatkowo określilibyśmy szerokość). Jednak lepiej jest to ująć jako mixin o nazwie np. center. W mixina tego wrzucamy kod:

@mixin center(){ display: block; margin: 0 auto; }

I kolejno zarówno w pliku \_nav jak i style załączamy naszą wstawkę do akapitu(style.scss) i naszej nawigacji nav(\_nav.scss). Należy pamiętać że załączamy daną wstawkę do danego selektora za pomocą @include nazwaWstawki. Jednak żeby to było możliwe, konieczne jest zaimportowanie pliku \_mixins.scss do pliku \_nav poprzez @import ‘\_nav.scss’ , aby możliwe było korzystanie z tego pliku.

Jednak w przypadku kiedy nie pobieramy żadnych argumentów, lepszym rozwiązaniem jest placeholder. Tworzymy zatem nowy plik o nazwie np. \_placeholders.scss i tam będziemy umieszczać nasze placeholdery. Placeholder tworzymy poprzez: %nazwaPlaceholdera { tutaj właściwości }

Należy pamietaż, że placegolderów nie importujemy – jedynie wystarczy dodać komendę @extend %nazwaPlaceholdera; W przeciwnym razie nasz placeholder byłby generowany podwójnie. Inaczej jest w przypadku mixinów, których plik trzeba zaimportować i dopiero potem dodać do danego selektora @include nazwaMixina();

***Funkcje w SASS***

Funkcja w SASSjest bardzo podobna do mixinów, alenie zwraca nam styli, a jakieś wartości np. po obliczeniach. Funkcję tworzymy w następujący sposób:

@function nazwaFunkcji($paratert1, $parametr2) { @return; }

Funkcje pozwalają wykonywać obliczenia itp. rzeczy. Aby odwołać się do funkcji wystarczy wypisać jej nazwę i podać w nawiasie argumenty, które przyjmuje. Należy pamiętać o zaimportowaniu pliku, w którym dana funkcja się znajduje, inaczej nie będzie odniesienia do wywołanej funkcji.

Ciekawe funkcje sa umieszczone w dokumentacji sass (sass-lang.com) gdzie możemy znaleźć np. funkcjie lighten($color, $amount), gdzie podajemy kolor i procentową wartość o ile go rozjaśnić.

Również ciekawa jest funkcja hsl, która przyjmuje 3 wartości – barwę, nasycenie oraz jasnść i na podstawie tych 3 wartości zwraca nam kolor: hsl($hue, $saturation, $lightness); Poza hsl jest również funcja hsla, która dodatkowo pobiera parametr $alpha – czyli przezroczystość.

Kiedy chcemy coś rozjaśnić lub ściemnić wystarczy użyć funkcji lighten, albo darken i w nawiasie podać kolor (może to być zmienna) i po przecinku procentową wartość zabiegu.