BEM - pierwsze spojrzenie

BEM, tak na szybko, to bardzo popularna konwencja (metodologia) nazywania klas węzłów (obiektów/znaczników) HTML oraz tworzenia selektorów w CSS.

Pierwsze co rzuca się w oczy gdy spojrzymy w HTML strony budowanej w oparciu o reguły BEM, to podwójne podkreślenia i podwójne myślniki oraz dziwne powtórzenia w nazwach klas.

```
<nav class="menu">
  <div class="logo">
    <h1 class="logo__name">New&Extra Ltd.</h1>
    <img class="logo__image" src="" alt="">
  </div>
  <ul class="menu__item-list">
    <li class="menu__item menu__item--open">
      <a href="#" class="menu__link">Home</a>
    </li>
    <li class="menu__item">
      <a href="#" class="menu__link">Product</a>
    </li>
  </ul>
</nav>
```

BEM - o co chodzi

BEM to przede wszystkim:

- myślenie o interfejsie (stronie) jako zbiorze (niezależnych) komponentów.
- konwencja (zestaw reguł) określająca sposób nazywania elementów HTML i tworzenia selektorów w CSS.

BEM - cechy

- Modułowość w zakresie struktury HTML (niezależne komponenty) i CSS. Struktura strony (interfejs) zbudowana z klocków (komponentów).
- Architektura CSS oparta o klasy, z minimalną rolą zagnieżdżeń oraz marginalnym wykorzystania reguł kaskadowości.
- Oparty o trzy typy obiektów. Blok (komponent), element oraz modyfikując je, ale nie występujący jako samodzielny obiekt, modyfikator.
- Konwencja BEM nie rozwiązuje jednoznacznie wszystkich możliwych sytuacji, pozostawiając pewną dowolność (wybór) deweloperom.

Po co BEM (czy inna metodologia)?

Organizacja kodu jest kluczowa w każdym większym projekcie.

- wspólne reguły pozwalają pracować w grupie i w dłuższym okresie czasu
- unikamy chaosu w kodzie
- kod skalowalny i wielokrotnego użytku

BEM

Block - często określany też komponentem. Niezależny element interfejsu.

```
<nav class="main-menu">...</nav>
```

Element - część komponentu, która jest zależna od bloku i nie występuje poza blokiem.

```
<nav class="main-menu">
     <ul class="main-menu__vertical-list">...</ul>
</nav>
```

Modifier - niestandardowy blok lub element (rozszerzony/zmieniony). Modyfikator nie występuje samodzielnie.

```
<nav class="main-menu">
     <li class="menu__item">...</li>
     <li class="menu__item menu__item--disabled menu__item--visited">...</li>
</nav>
```

Kiedy powstaje blok czy element

Zgodnie z konwencją BEM o bloku (komponencie) czy elemencie (zależnej od komponentu części) możemy mówić wtedy gdy elementowi (węzłowi) HTML nadamy klasę wg reguł BEM.

Dlatego mówimy tu ciągle tylko o konwencji, sposobie myślenia o interfejsie i sposobie w jaki nazywamy poszczególne części interfejsu. Samo nadanie klasy nie powoduje bowiem, o czym doskonale wiemy, że poza naszym myśleniem, taki obiekt uzyskuje dodatkowe cechy czy możliwości.

BEM - Block

Blok to np. stopka, przycisk, slider, menu, formularz, artykuł (czy lista artykułów), modal (popup), logo, wyszukiwarka, moduł profilu, post (w social media), belka ze zgłoszeniem błędu, checkbox, kontener “z czymś” itd...

- Samodzielna część interfejsu (strony/aplikacji). Często określany też (niezależnym) komponentem.
- Technicznie blokiem jest element HTML (węzeł), który posiada klasę z nazwą bloku.
- Z punktu widzenia HTML jeden blok może być zagnieżdżony w innym (np. przycisk w formularzu czy logo w menu).
- To czy uznać dany element za komponent (i tworzyć blok) jest kwestią oceny dewelopera. W niektórych sytuacjach przycisk, checkbox, logo będą komponentami a w innych nie.

BEM - Block

- Bloki są równe (hierarchicznie), niezależne, choć oczywiście ich układ w strukturze HTML może mieć wpływ na właściwości i zachowanie.
- Blok zawsze może być użyty niezależnie i wielokrotnie.
- Jeden blok może być użyty w innym bloku (komponent przycisku może być częścią komponentu artykułu).
- Nazwa bloku powinna być unikalna i powinna opisywać w prosty sposób czym jest dany komponent (blok). Nie powinna być zbyt długa. Przykładowe nazwy klas bloków to np. , <button class=”button”> czy <button class=”btn”>.
- Nazwa bloku jest też nazwą klasy <nav class=”menu”>

BEM - Block

- W CSS wskazujemy blok za pomocą jego klasy

```
.container {}
.articles {}
.main-menu {}
.user-list {}
.registration-form {}
.form {}
```

ŹLE

```
section.container {}
articles .articles {}
.wrapper .main-menu {}
ul.user-list {}
.registration-form.focus {}
form.form {}
```

BEM - Element

Element to np. pole formularza, grafika w slajderze, tytuł w artykule, pozycja w menu czy na liście, adres w stopce.

- Element zawsze występuje w bloku. Nie występuje bez bloku, poza blokiem.
- Każdy element może należeć tylko do jednego bloku.
- Element jest częścią komponentu.

BEM - Element

- Twórcy BEM uznali, że ponieważ element jest związany z blokiem znaczeniowo, to jest powiązany z nim także nazwą. Nazwa bloku jest częścią nazwy klasy elementu np.

```
<form class="registration-form"> /* blok (komponent) */
      <input class="registration-form__input-type-number" type="number"> /* element */
</form>
```

- W nazwie klasy elementu między nazwę bloku a nazwą elementu znajduje się podwójne podkreślenie.

```
<form class=”newsletter-form>
      <input class=”newsletter-form__input-user-name”>
</form>
```

BEM - Element

- Nazwa elementu musi być unikalna w obrębie bloku. Jeśli więc chcemy stworzyć element title w bloku article to nie ma problemu by element z nazwą title pojawił się też w bloku header <h1 class="header__title> i w innym miejscu struktury HTML <h1 class="article__title>. Wynika to oczywiście z tego, że powstają tu dwie klasy, które precyzyjnie identyfikują o jaki element nam chodzi (czy to w arkuszach stylów czy w oczach innych deweloperów).

Często mówimy, że nazwa bloku stanowi przestrzeń nazw, co sprawia, że kod w obrębie bloku jest niezależny.

BEM - Element

- Nazwa elementu w naszym przykładzie poniżej to link. Nazwa klasy elementu składa się z nazwy bloku oraz nazwy elementu oddzielonych podwójnym podkreśleniem, np. <a class=”menu__link>

- Warto zwrócić uwagę na to, że nazwa elementu, to coś innego, niż nazwa klasy. Jednak w praktyce, to nie jest istotne i większość osób po prostu określa nazwę elementu jako tożsamą z nazwą klasy czyli zamiast element link jak w przykładzie napisze/powie/pomyśli element menu__link (oczywiście nadal poprawnie jest powiedzieć element link w komponencie/bloku menu).

BEM - Element

- W CSS wskazujemy element oczywiście poprzez odwołanie się do jego klasy

```
.footer__address {}
.articles__search {}
.main-menu__logo {}
.user-list__name {}
.image-gallery__img {}
```

BEM - Modifier

Gdy dany blok czy element występuje w innej wersji niż “standardowa” używamy modyfikatora, czyli w praktyce dodajemy dodatkową klasę do elementu HTML.

- Modyfikator pozwala dodać/zmienić właściwości standardowej (podstawowej) wersję bloku lub elementu.
- Nazwa modyfikatora powinna informować/wskazywać do czego jest, co robi, jak wpływa na wyświetlenie czy jaki ma stan.

BEM - Modifier

- Nazwa klasa modyfikatora jest tworzona przez dodanie nazwy modyfikatora do nazwy klasy bloku lub klasy elementu. Dodatkowo nazwa modyfikatora jest rozdzielona za pomocą dwóch myślników (minusów).

block--modifier

```
<ul class=”article-list”> /* wersja standardowa bloku */
/* poniżej użycie trzech modyfikatorów type-news (rodzaj treści), horizontal (układ), is-open (stan) */
<ul class=”article-list article-list--type-news article-list--horizontal article-list--is-open”>
```

block__element--modifier

```
<li class=”article-list__item”>  /* wersja standardowa elementu */
<li class=”article-list__item article-list__item--color-blue”> /* użycie modyfikatora color-blue*/
```

BEM - Modifier

- Jako klasa, modyfikator nigdy nie występuje samodzielnie w elemencie HTML. Klasa modyfikatora jest używana w znaczniku HTML zawsze w połączeniu z klasą bloku czy elementu, które modyfikuje np.

  <a class="menu__link menu__link--important">

  <a class="menu__link--important"> /* źle */

- do bloku/elementu może być przypisanych wiele modyfikatorów

  <nav class="menu menu--vertical menu--is-open">

  <img class="article__image article__image--left article__image--large">

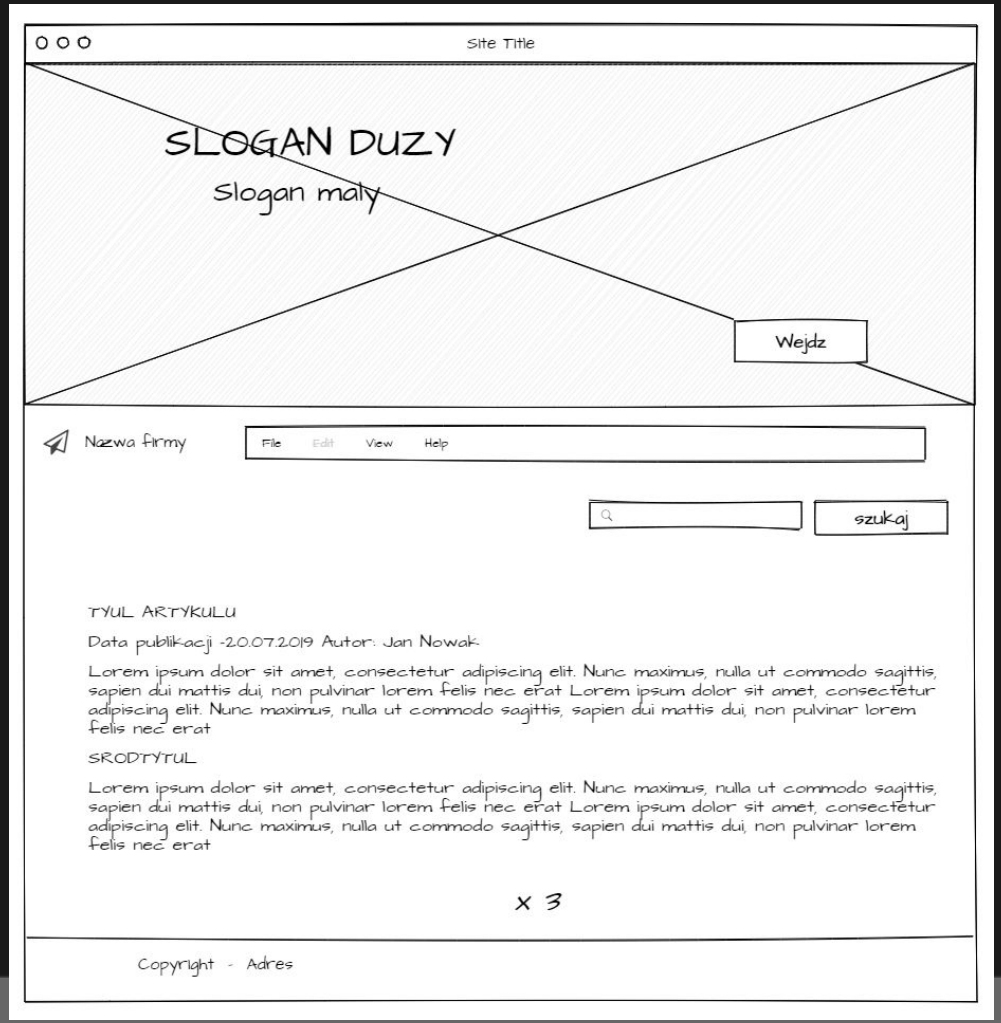
  <form class="search search--users search--large">

BEM - Modifier

- Modyfikator reprezentuje zmianę i rozszerzenie czyli modyfikację. Nie dublujemy tych samych właściwości i wartości w regule dla modyfikatora - dodajemy nowe właściwości lub modyfikujemy istniejące.

```
.button {
    font-size: 20px;
    color: rgb(200,126,126);
    background-color: rgba(34,226,226,0.8);
}
```

```
.button--theme-halloween {
    font-size: 20px;  /* źle */
    font-family: “comic sans”;
    background-color: rgb(6,6,6);
}
```


Site Title
SLOGAN DUZY
Slogan maly
Wejdz
Nazwa firmy
File
Edit
View
Help
szukaj
TYUL ARTYKULU
Data publikacji -20.07.2019 Autor: Jan Nowak
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc maximus, nulla ut commodo sagittis, sapien dui mattis dui, non pulvinar lorem felis nec erat Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc maximus, nulla ut commodo sagittis, sapien dui mattis dui, non pulvinar lorem felis nec erat
SRODTYTUL
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc maximus, nulla ut commodo sagittis, sapien dui mattis dui, non pulvinar lorem felis nec erat Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc maximus, nulla ut commodo sagittis, sapien dui mattis dui, non pulvinar lorem felis nec erat
x 3
Copyright - Adres

Reguły BEM

Omówmy zbiór wybranych, istotnych reguł metodologii BEM dotyczących tworzenie nazw, klas i selektorów.

Zobaczmy też różne sytuacje, z którymi można się spotkać i w których konwencja nie zawsze mówi co robić albo zostawia nam wybór. W bardzo wielu sytuacjach stosuje się różne praktyki w zależności od preferencji deweloperów.

Nazywanie klas

Klasa pełni rolę nie tylko identyfikatora (dla celów arkuszy stylów), ale także informuje:

- czym jest dana część interfejsu (blokiem, elementem, modyfikatorem),
- za co odpowiada dana część interfejsu, jaką rolę pełni, jakie ma cechy czy, czasami, w jakim jest stanie (aktywny, wyłączony, otwarty, ukryty)

Przykładowe nazwy w modyfikatorach

/* właściwości elementu/bloku w modyfikatorze */

<button class="btn btn--large"></button>
<button class="button button--yellow-layout"></button>
<header class="page-header page-header--primary"></header>
<article class=”article article--theme-punk”></article>
<p class=”article__text article__text--size-large”></article>

/* stan w modyfikatorze */

<form class="form form--user form--state-focus"></form>
<nav class="mobile-menu mobile-menu--open"></nav>
<li class”menu__item menu__item--is-hidden”>

Podwójne podkreślenie (underscore)

Tworząc klasę elementu zawsze używamy podwójnego podkreślenie, które oddziela nazwę bloku od nazwy elementu.

BLOK__ELEMENT

Podwójny minus/myślnik (hyphen)

Tworząc klasę modyfikatora bloku czy elementu zawsze używamy podwójnego myślnika, który oddziela nazwę bloku od nazwy modyfikatora czy nazwę klasy elementu od modyfikatora.

BLOK--MODYFIKATOR

BLOK__ELEMENT--MODYFIKATOR

Nazwy z myślnikami

Tworząc nazwy bloków, elementów i modyfikatorów najlepiej jest korzystać z myślników - oczywiście o ile nazwa danej części interfejsu ma być dwu (czy więcej) członowa.

nazwa-bloku

nazwa-bloku--nazwa-modyfikatora

nazwa-bloku__nazwa-jakiegos-elementu

nazwa-bloku__nazwa-elementu--nazwa-tego-modyfikatora

Nazwy w konwencji camelCase

Mniej popularna, ale dozwolona i spotykana jest składnia wielbłądzia. Każdy kolejny człon nazwy piszemy wielkimi literami.

nazwaBloku

nazwaBloku--nazwaModyfikatora

nazwaBloku__nazwaJakiegosElementu

nazwaBloku__nazwaElementu--nazwaTegoModyfikatora

Nazwy w konwencji snake_case

Możemy również pisać nasze nazwy z użyciem podkreślników. Każdy kolejny człon nazwy oddzielamy podkreśleniem. Pamiętajmy, że podwójne podkreślenie jest jednak zarezerwowane dla oddzielenie elementu od bloku.

```
nazwa_bloku
nazwa_bloku--nazwa_modyfikatora
nazwa_bloku__nazwa_jakiegos_elementu
nazwa_bloku__nazwa_elementu--nazwa_tego_modyfikatora
```

Element w elemencie - struktura HTML a BEM

Czy BEM odwzorowuje strukturę HTML?.

Block__Element__Element–Modifier

np.

.menu__list__li--active

.menu__list__item__link--active

Czy to jest dobre?

Struktura w HTML

nav
  ul
    li
      a

Element w elemencie - struktura HTML a BEM

Czy BEM odwzorowuje strukturę HTML?.

Block__Element__Element–Modifier /* źle */

np.
.menu__list__li--active /* źle */
.menu__list__item__link--active /* źle */

Struktura w HTML
nav
ul
li
a

Czy to jest dobre?

NIE, to nie jest poprawna praktyka! BEM to BEM, a nie BEEM czy BEEEM :)

BEM nie komunikuje struktury zagnieżdżeń, która występuje w elemencie.

Używaj w nazwie tylko raz podwójnego podkreślenia (__).

BEM komunikuje tylko że dany element jest częścią komponentu.

Element w elemencie - struktura HTML a BEM

```
/* HTML - źle, klasa nie powinna odwzorowywać zagnieżdżeń w HTML */
<a class="menu__list__item__link">...</a>
/* CSS - choć zdarzy Ci się zobaczyć takie rozwiązania to są one odradzane*/
.menu__list__item__link {}

/* HTML - element BEM odwołuje się bezpośrednio do bloku a nie innego elementu*/
<a class="menu__link">...</a>
/* CSS - block__element - zgodnie z konwencją */
.menu__link {}
```

Struktura w HTML

nav
ul
li
a

Zagnieżdżenie elementu BEM w elemencie BEM (odradzane)

Zapamiętaj: odwzorowania struktury HTML w strukturze BEM, jak poniżej nie jest dobrą praktyką.

```
<nav class="top-menu">
     <ul class="top-menu__vertical-list">
          <li class="top-menu__vertical-list__item">
               <a href="" class="top-menu__vertical-list__item__link">Home</a>
          </li>
     </ul>
</nav>
```

BEM tworzy własną strukturę typu “drzew” nałożoną na drzewo DOM.

Zagnieżdżenie elementu BEM w elemencie BEM (odradzane)

Element częścią bloku, a nie innego elementu.

```
/* Dobrze */
<ul class="menu__item-list">
    <li class="menu__item">
      <a href="" class="menu__link">Home</a>
    </li>
</ul>

/* Spotykane, może być, pokazanie zagnieżdżenia przez konwencje nazywania elementu */
<ul class="menu__list">
    <li class="menu__list-item">
      <a href="" class="menu__list-item-link">Home</a>
    </li>
</ul>
```

Selektory w BEM - id?

```
/* HTML */
<h1 id="page-title">Nazwa strony</h1>
/* CSS */
#page-title {} - nie
```

- id tylko raz na stronie, a BEM jest (co do zasady) modułowy, czyli każdy komponent powinien być wielokrotnego użytku
- część osób korzysta z id w połączeniu z JS, ale też niekoniecznie. Korzystając z BEM raczej powinieneś całkowicie wyeliminować identyfikatory.

Selektory w BEM - tagi/znaczniki?

/* HTML */

<h1>Nazwa strony</h1>

/* CSS */

h1 {} - nie (poza plikiem/stylami konfiguracyjnym/bazowymi)

main article h1 {} - nie, zagnieżdżenia, tym bardziej połączone ze znacznikami.

- Nie, bo zmienia się struktura (interfejs) z h1 na h2. Zmienia tagu wymusza zmiany w CSS
- Precyzyjne wskazywanie o których tag nam chodzi wymaga budowania zagnieżdżonych selektorów co czyni nasz CSS mniej czytelnym i mniej elastycznym

Selektory w BEM - kombinowane (złożone)?

```
/* HTML */
<h1 class="title large">Nazwa strony</h1>
<article class="article article--main">Nazwa strony</article>
/* CSS */
.title.large {} - nie
.article.article--main {} - nie
article--main {} - tak
article {} - tak
```

- nie, bo kaskadowość może nam tu namieszać (dwie klasy nadpisują jedną)
- kod jest mniej czytelni i mniej elastyczny

Selektory w BEM - znaczniki i klasy razem?

```
/* HTML */
<h1 class=”article__title”>Nazwa strony</h1>
/* CSS */
h1.article__title {} - nie
.article__title {} - tak
```

- nie, bo kaskadowość może nam tu namieszać (klasa i tag nadpisują klasę)
- precyzyjne wskazywanie tagu i klasy czyni kod mniej uniwersalnym
- zamiast BEM misz masz. Zasad nie może być zbyt wiele (BEM jasno mówi o selektorach opartych na klasach), bo jak jest zbyt wiele to trudniej ich przestrzegać. BEM mówi: “Cchesz być fajny? To, nie używaj tagów w połączeniu z klasami, używaj klas”.

Selektory w BEM - klasy i zagnieżdżone klasy

```
/* HTML */
<article class=”article article--main”>
  <h1 class=”article__title”>Nazwa strony</h1>
</article>

/* CSS */
.article {} /* wyizolowanie stylu każdej części interfejsu - blok */
.article--main {} /* jw - blok w zmodyfikowanym stanie*/
.article__title {} /* jw - element w bloku - przy okazji czytelność*/
.article--main .article__title {} /* przykład zagnieżdżenia - element w zmodyfikowanym bloku */
```

Selektory CSS zgodne z BEM - zagnieżdżenia

REASUMUJĄC - Nie korzystamy z zagnieżdżeń, chyba, że są one nam potrzebne (a nie są często)

<nav class="vertical-menu"><ul class="items">...
</nav> /* źle, bo lista nie będzie tu raczej blokiem */

```
nav ul.items {} - źle (bo selektory elementów/znaczników html)
.vertical-menu .items {} - źle (bo niepotrzebne zagnieżdżenie)
```

<nav class="vertical-menu"><ul class="vertical-menu__items">...
</nav> /* dobrze */

```
.vertical-menu .vertical-menu__items {} - źle (bo niepotrzebne zagnieżdżenie)
.vertical-menu__items {} - dobrze
```

Selektory CSS zgodne z BEM - zagnieżdżenia

```
<nav class=”vertical-menu vertical-menu--theme-gray”>
        <ul class=”verticla-menu__items”>...</ul>
</nav>
```

By nadać właściwości liście, odwołujemy się w stylach do obu sytuacji.

Gdy blok bez modyfikatora, by stworzyć regułę dla elementu wystarczy użyć selektora klasy:

```
.verticla-menu__items {}
```

Gdy blok posiada modyfikator, określając właściwości elementu możemy użyć zagnieżdżenia w selektorze:

```
.vertical-menu--theme-gray .vertical-menu__items {}
```

Kiedy blok?

Blok (komponent) - gdy dana część interfejsu jest niezależna od innych komponentów i może być wielokrotnie użyta w różnych konfiguracjach.

Blok - gdy z punktu widzenia przejrzystości kodu (a więc subiektywnie) czy tworzenia struktury wyodrębnienie danego komponentu ma sens. Np.

<article class=”article”>

 lub/czy

Kiedy blok?

Blok tworzymy też często dla wrapperów np. dla potrzeb stworzenia siatki. Są to bloki, które istnieją tylko ze względu na tworzony układ. Np.:
<body class=”page”><nav class=”menu page__menu”>.

Blok jako grupowanie wspólnych właściwości - Taki blok jest używany w parze z innymi blokami czy elementami).
W przykładzie blok “text”, który przechowuje pewne właściwości (color, font, wielkość fonta i interlinii itd.) Np.:

<address class=”footer__address text ”>
<article=”article text”>

Blok czy element. A może blok i element?

button - przycisk to element czy blok (komponent)? To zależy ...

```
/* 1. Przycisk jako element komponentu
całkowicie odmienny, dedykowany headerowi */
<header class="header">
 <button class="header__button"></button>
</header>
```

```
/* 2. Przycisk jako komponent + modyfikator */
/* gdy wykorzystywany wielokrotnie w różnych miejscach interfejsu, a dodatkowo właściwości
bardzo podobne w każdym miejscu */
<header class="header">
 <button class="button button--header"></button>
</header>
```

```
/* 3. mix - część osób korzysta, inni preferują nie
łączyć dwóch komponentów w jednym*/
<header class="header">
 <button class="button header__btn">Klik</button>
</header>
```

Mix

Blok (komponent) jest najczęściej reprezentowany przez jakiś element (węzeł) HTML,

Jeden element HTML stanowi strukturę obejmującą cały komponent

```
<header class="slider">/* w środku elementy tego komponentu */</header>
<nav class="mobile-menu">/* w środku elementy tego komponentu */</nav>
```

Jednak może się zdarzyć, że jeden element HTML, reprezentuje więcej niż jeden komponent. Reprezentuje np. blok (komponent) ale też element (część) innego bloku, jak w przykładzie poniżej. Wtedy mówimy o mixie.

```
<header class="header">
  <div class="logo header__logo">/*inne elementy*/</div>
</header>
```

Mix

```
/* HTML */
<header class="header">
    <div class="logo header__logo">/*inne elementy*/</button>
</header>

/* CSS */
.logo { wspólne cechy }
.header__logo { pozycja w układzie np. position: absolute czy margin/padding }
```

W praktyce część osób korzysta z mixów, a część nie. Wiele osób korzysta też z mixów jeszcze w inny sposób, np, łącząc dwa bloki i inne alternatywne układy. .

Dwa komponenty (bloki) czy modyfikator?

```
/* 1. Dwa osobne komponenty */
<nav class="menu-horizontal">
 <ul class="menu-horizontal__list">
 ...
 </ul>
</nav>

<nav class="menu-vertical">
 <ul class="menu-vertical__list">
 ...
 </ul>
</nav>
```

Decyzja? Gdzie zmiany i jak dużo.
Nie ma jednej dobrej odpowiedzi.
Pamiętajmy choćby o DRY.

```
/* 2. Jeden komponent i modyfikator*/
<nav class="menu menu--horizontal">
 <ul class="menu__list">...</ul>
</nav>

<nav class="menu menu--vertical">
 <ul class="menu__list">...</ul>
</nav>

/* 3. modyfikować elementy a nie sam komponent */

<nav class="menu">
 <ul class="menu__list menu__list--vertical">...</ul>
</nav>
```

Stan jako osobna klasa (blok)

Część deweloperów wykorzystuje też nazwę klasy jak sposób na przechowywanie stanu np. tworzy klasę .is-active .is-hidden is-open. <nav class=”menu is-hidden”>
Czasami można się więc spotkać z określeniem, że Blok przechowuje nie tylko komponent ale i jego stan. Taki zapis ułatwia pracę z JavaScript, ale nie jest konieczny. Część osób po prostu stworzy modyfikator: <nav class=”menu menu--is-hidden”>

W praktyce spotkasz się z dwoma sposobami informowania o stanie (i pewnie jeszcze innymi konwencjami):
<button class=”btn is-active”>Uruchom</button>
<button class=”btn btn--is-active”>Uruchom</button>

Klasa w każdym elemencie?

W każdym, który musisz stylować, poza znacznikami bazowymi, gdzie możesz odwołać się (najlepiej w pliku bazowym styli np. base.css) bezpośrednio do nich

```
body {margin: 0; padding: 0;}
p { font-size: 1.2rem; color: #333aaa; } /* podstawowe style */
```

Ale oczywiście również do nich możesz dodać klasy np.

```
<body class="page">
```

czy bezpośrednio w komponentach, np.:

```
<article class="article">
        <p class="article__text">
/*Część osób zostawi p bez klasy, chyba, że będzie posiadać inne właściwości niż bazowe.*/
```

selektor uniwersalny i najprostszy reset

```
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}
```

Część osób używa w pliku bazowym CSS. Część osób stosuje resetowanie stylów tego typu jednak na poziomie komponentu a nie globalnym. Część osób korzysta z normalize.css lub jeszcze innego rozwiązania (ale to juz w sekcji o optymalizacji).

BEM - korzyści

- Zrozumiały struktura komponentów
- Czysty, łatwiejszy w pracy, utrzymaniu i debugowaniu kod.
- Layouty i komponenty wielokrotnego użytku.
- Stosując zasady łatwiej (szybciej) piszesz kod.
- Ktoś z zewnątrz łatwiej zrozumie projekt, podobnie ty za wiele miesięcy jeśli zajrzysz kod łatwiej się do niego ponownie wdrożysz.

BEM - krytyka

- Długie (brzydkie) nazwy
- Jak to klasy w (prawie) każdym elemencie pogięło Was?? - Przyzwyczaj się!
- A co, kaskadowość zła?
- Nieczytelny HTML (morze klas)
- Problemy, które wymagają kombinowania/naginania (większe projekty, dziwne przypadki)
- Trudne początki, trzeba się przestawić

BEM - reguły są po to by ich przestrzegać, ale...

Sam będziesz modyfikował zasady BEM, zespoły w których będziesz zapewne pracować, też je modyfikują. Perfekcjonizm nie zawsze jest wymagany, choć nie idź na łatwiznę. Łam reguły świadomie, a nie dlatego, że ich nie znasz czy nie masz ochoty zastanowić się jak rozwiązać problem.

Nie dla innych niż klasy oparte na selektorach - poza wyjątkami jak plikiem bazowy.
Nie dla zagnieżdżeń selektorów - choć są sytuacje, gdzie trzeba coś zagnieździć.

Nie bój się eksperymentować. Nie obawiaj się błędów.

BEM - reguły są po to by ich przestrzegać, ale...

Z drugiej strony jak czasami z kimś współpracuje i widzę jego BEM i mój BEM, to mam wrażenie, że są jakieś dwa różne BEM-y ;) Najlepsze są strony gdzie część rzeczy mamy w BEM, część w czymś jak BEM, część klas wynika z użycia frameworków, a część nikt nie wie skąd wynika i kto to wymyślił :)

Wiele osób stosuje konwencję “jak im pasuje”, wiele zespołów wypracowuje jakieś własne zasady czy minimalne zasady. Ja sam lubię eksperymentować i zdarza mi się naginać BEM :)

Ucząc się, staraj się jednak przestrzegać podstawowych zasad, zdefiniowanych w BEM

BEM - najlepiej z Sass

BEM bardzo fajnie komponuje się z preprocesorem Sass - składnia, czy struktura plików w połączeniu z importowanie, które zapewniają modułowość.

Nie przypadkiem więc za chwilę przejdziemy do nauki preprocesora Sass a potem połączymy obie te rzeczy (Sass & BEM).

W tym kursie często będzie używana konwencja BEM, a kiedy będzie używana to bardzo często właśnie z Sass - czeka nas więc jeszcze sporo praktyki i różne przypadki, których pewnie w tej prezentacji nie uwzględniliśmy :)