

2. Praktyka CSS



Wyzwania:

- Poznamy zaawansowane selektory oraz pseudoklasy CSS.
- Zbudujemy kilka komponentów, żeby zgłębić koncepcje związane z CSS.

Sugestia

Przed rozpoczęciem tego modułu (lub po teorii) warto przejść ten kurs [Więcej o CSS](#), aby przećwiczyć w praktyce podstawowe zagadnienia, co pomoże przy wykonywaniu zadań.

Clear

2.1. Zaawansowane właściwości



W poprzednim module poznaliśmy podstawy HTML i CSS, jednak był to jedynie wierzchołek góry lodowej. Sam CSS posiada dużo więcej zaawansowanych funkcji, z których część poznamy w niniejszym module. Lwia część pracy front-end developera polega właśnie na tworzeniu odpowiednich stylów, tak aby zapewnić użytkownikom optymalne korzystanie ze strony internetowej. Z tego powodu poznamy CSS bliżej.

CSS w wersji 3

CSS ewoluował wraz z gwałtownym rozwojem technologii mobilnych oraz internetu. Jego najnowszą wersją jest CSS3, który dodał nowe funkcjonalności będące w dużej mierze odpowiedzią na te wydarzenia.



W odróżnieniu od swojego poprzednika, CSS3 nie ma jednolitej, jasnej specyfikacji, lecz został podzielony na moduły, z których każdy ma nieco inny status. World Wide Web Consortium (W3C) odpowiedzialne za specyfikację tej technologii może wydać pozytywną rekomendację w zależności od stopnia dojrzałości danego rozwiązania. Oznacza to, że dana technologia jest sprawdzona i stabilna.

Wtedy kolejnym krokiem jest zaimplementowanie (zaprogramowanie) obsługi takiej nowej właściwości CSS przez twórców przeglądarek internetowych (takich jak Google Chrome czy Mozilla Firefox) w najnowszej wersji swojego produktu. W tym momencie front-endowcy mogą zacząć używać nowo wprowadzonej właściwości bez obawy o to, że nie zadziała ona u użytkownika strony.

Niezależnie od opisanego wyżej procesu wiele komponentów CSS3 jest już powszechnie używanych, mimo że nie mają oficjalnego statusu rekomendacji, jak np. media queries czy animacje.

Pełna specyfikacja CSS (nie tylko 3) dostępna jest na stronie W3C pod adresem <https://www.w3.org/Style/CSS/specs.en.html>.

Dobre praktyki

Gdy zaczynamy pracować nad coraz bardziej skomplikowanymi projektami wymagającymi więcej linii kodu, coraz większego znaczenia nabiera utrzymanie kodu w porządku, tak aby był czytelny. Warto wyrobić sobie nawyk pisanie "czystego kodu" od samego początku, niezależnie od tego, ile liczy sobie linii.

Kolejnych trudności przysparza praca w zespole nad jednym projektem. Kod, który dla nas jest jasny, nie musi taki być dla drugiej osoby. Stąd też w środowisku front-end developerów (i w ogóle programistów) utarły się pewne konwencje, które mają na celu ułatwienie pracy i uniknięcie problemów z czytaniem kodu. Można powiedzieć, że odbiorcami naszego kodu są nie tylko osoby wyświetlające strony, ale także osoby czytające go, tj. my sami oraz inni programiści.

Dodawanie komentarzy



Komentarze w kodzie zazwyczaj wykorzystywane są jedynie po to by oddzielić jedną sekcję kodu CSS od drugiej, dotyczącej innych elementów. W większych projektach, żeby uniknąć ładowania zbędnych stylów do widoków, na których nie są wykorzystywane, tworzy się osobne pliki ze stylami i podpinają je do widoków, na których są wykorzystywane.

Komentarze w CSS dodajemy w ten sposób:

```
1 | /* my comment */
```

W HTML komentarz możemy dodać tak:

```
1 | <!-- my comment -->
```

Wewnątrz komentarzy możemy dodać dowolny tekst i nie będzie on mieć wpływu na wygląd i zachowanie strony. Dobrą praktyką jest dodawanie komentarzy objaśniających za co odpowiedzialna jest dana grupa reguł CSS np.

```
1 | /*  
2 |    === HEADER ===  
3 | */
```

Dodając taki komentarz, jak wyżej nad kilkoma stylami w pliku CSS dotyczącymi nagłówka (opisanie grupy stylów), oszczędzimy sobie i innym trudu związanego z ustaleniem do czego służy dana grupa stylów (reguł CSS). Można wyróżnić więcej takich grup.

Za pomocą komentarza możemy też dezaktywować część kodu bez usuwania go. Sprawdza się to znakomicie w sytuacji, gdy nasza strona nie działa tak jak powinna i chcemy, "wyłączając" niektóre jej fragmenty sprawdzić, który z nich powoduje niepoprawne działanie całości.

Przyjęło się, że komentarze w HTML i CSS dodawane są **w języku angielskim**.

Formatowanie



Źle sformatowany kod staje się nieprzejrzysty dla czytelnika. Dlatego też warto zadbać o jednolity wygląd naszego kodu w całym pliku CSS.

Choć różne formatowanie kodu często nie powoduje błędnego wyświetlania strony, to warto próbować stosować pewne ogólnie przyjęte konwencje. Reguła w CSS najczęściej sformatowana jest tak, jak poniżej:

```
1 | .btn-icon {  
2 |     width: 200px;  
3 |     height: 80px;  
4 | }
```

Każdy selektor czy właściwość są dodawane od nowej linii. Następnie po spacji (w tej samej linii) dajemy znak { (otwierający nawias klamrowy, *pot. wąsy*) otwierający definicję naszej reguły CSS. Każda właściwość CSS zaczyna się już od nowej linii, dla wartości po dwukropku też często daje się spację. Kończy się średnikiem (jego brak może powodować błędy na stronie!). Do tego każda właściwość powinna być "wcięta" w prawo za pomocą tabulatora (klawisza "Tab"). Nawias zamykający zbiór właściwości powinien być od nowej linii bez wcięcia. Takie formatowanie sprawi, że kod będzie bardziej przyjazny w czytaniu.

Istnieją różne konwencje związane z formatowaniem CSS (np. zamiast tabulatora dla wcięcia, daje się 2 spacje), a kluczem jest tutaj konsekwencja w ich stosowaniu. Dzięki temu będziemy mogli łatwo odnaleźć w kodzie to, co nas interesuje.

Przykładową konwencję formatowania można znaleźć w [przewodniku udostępnianym przez Google](#).

Nazewnictwo

Głównie dotyczy to nazw klas, które są najczęściej stosowane w selektorach reguł CSS.

Po pierwsze, należy przyjąć anglojęzyczne nazewnictwo klas. Gdy pracujesz nad dużymi projektami, istnieje duże prawdopodobieństwo, że trafisz do międzynarodowego zespołu. Nazwy klas powinny być związane z ich funkcjami, umiejscowieniem, typem stylowanego elementu lub kontekstem. Dobre nazwy klas sprawiają, że kod będzie bardziej zrozumiały dla czytających go, co pozwoli na szybsze nawigowanie lub wdrożenie się w dany kod.



Przykładowo nazwa klasy **main-header** (jeśli headerów na stronie mamy kilka) jest łatwiejsza do zrozumienia niż np. **m**.

Podobnie jak w przypadku formatowania kodu, tak i tutaj możemy znaleźć konwencje. Rozwiązaniem mającym wielu zwolenników jest **BEM** (z ang. Block, Element, Modifier). Więcej na ten temat można poczytać w [specyfikacji BEM](#). Inne popularne rozwiązania to między innymi **SMACSS** (<https://smacss.com/>) oraz **SUITCSS** (<http://suitcss.github.io/>). Jednak w ramach tego kursu nie będziemy używać BEM, gdyż warto nauczyć się komponowania tego nazewnictwa samemu.

Niezależnie od umiejętności aplikanta, często rekruterzy IT patrzą w pierwszej chwili na kod kandydata właśnie pod kątem formatowania, opisanie wartościowymi komentarzami (jeżeli są potrzebne) i nazewnictwa. Na tym etapie często wychodzi amatorskość czy brak doświadczenia, dlatego warto dbać o wygląd kodu, i to nie tylko CSS :)

Elementy blokowe i liniowe

Przejdźmy do konkretów. Każdy element w HTML jest domyślnie albo liniowy (dopuszcza po bokach inne elementy, np. ``) albo blokowy (zajmuje 100% szerokości kontenera, w którym się znajduje, np. `<p>`). Odpowiada za to właściwość **display**, którą możemy definiować jak w przykładzie poniżej:

```
1 | .example {  
2 |     display: block;  
3 | }
```

Parafrazując, ta właściwość wpływa na najbliższe otoczenie elementu w HTML. Najczęściej wykorzystywane wartości dla **display** to:

- **block** - element blokowy. Element tego rodzaju zaczyna się zawsze od nowej linii i zajmuje całą dostępną szerokość. Elementom które domyślnie są blokowe nie musisz nadawać właściwości `width: 100%`; ponieważ domyślnie zajmują 100% dostępnej przestrzeni! Przykładowe elementy domyślnie blokowe to `<div>`, `<h1>` - `<h6>`, `<p>`, `<form>`, `<footer>`, `<section>`.
- **inline** - element liniowy. Nie zaczyna się od nowej linii i zajmuje tylko tyle miejsca, ile jest niezbędne. Przykładowe elementy domyślnie wyświetlane jako inline to ``, `<a>`, ``.
- **inline-block** - element liniowo-blokowy. Podobnie jak element liniowy wyświetlany jest w jednej linii. Różnica polega na tym, że takiemu elementowi można nadać

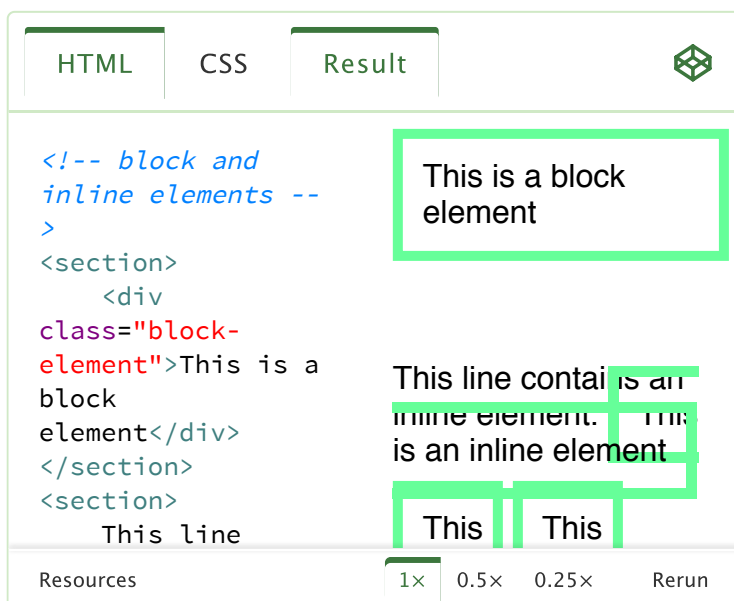


określoną wysokość i szerokość (**width** oraz **height**). Ma on często zastosowanie w pasku nawigacji strony, gdzie poszczególne linki do podstron wyświetlane są w jednej linii i mają określone wymiary.

- **none** - element taki nie zostanie wyświetlony, jego miejsce może być zajęte przez inne elementy. Może to mieć zastosowanie, jeśli chcemy wyświetlić jakiś element np. dopiero w momencie, gdy użytkownik najedzie kursorem na ikonę.

Jak już mówiliśmy, każdy element HTML posiada jakąś wartość domyślną dla tej właściwości, przykładowo `<div>`, `<h1>` oraz `<p>` są elementami blokowymi, z drugiej strony `` jest elementem liniowym. Możemy dowolnie zmieniać wyświetlanie interesujących nas elementów w pliku CSS.

Zwróć uwagę na kod poniżej oraz sposób, w jaki poszczególne elementy pojawiają się na stronie. Nadaliśmy tam poszczególnym `div`om różne właściwości `display`, zmieniając w ten sposób ich wymiary oraz rozmieszczenie na stronie (spróbuj pobawić się tym kodem zmieniając wartości dla **`display`**).



Właściwość **`display`** jest kluczowa nie tylko przy budowaniu elementów strony, ale także przy komponowaniu układu całej strony. Ale o tym później :)

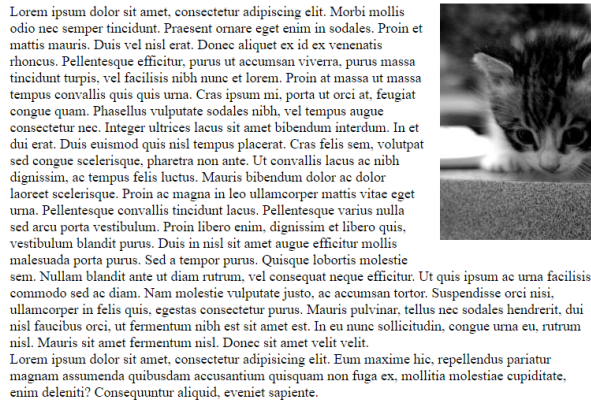
Floaty i align

Jeżeli weźmiesz do ręki dowolne czasopismo, z pewnością dostrzeżesz wiele fotografii, które otoczone są "opływającym" je tekstem. Aby przenieść takie rozwiązanie do stron internetowych, wykorzystujemy właściwość `float`. Jednak jest to tylko jedno z



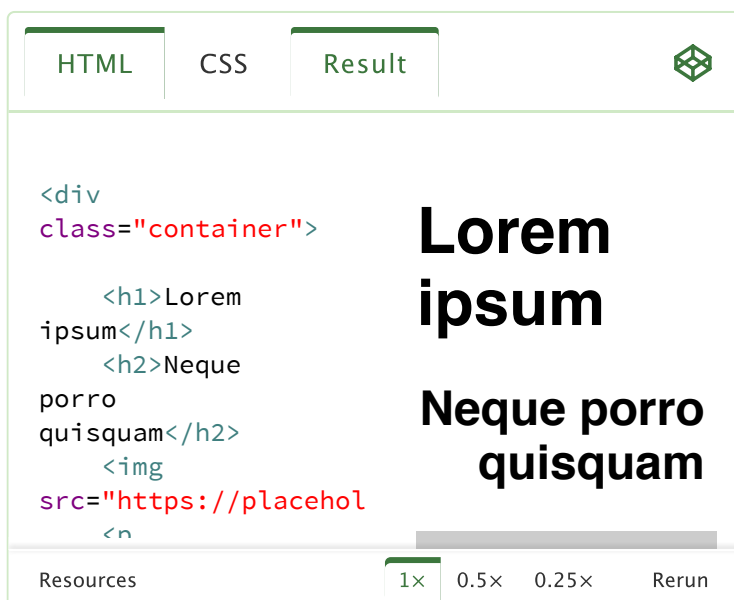
zastosowań, jakie nam ona daje. W praktyce **float** używany jest do rozmieszczenia treści na stronie. Za jego pomocą możemy zbudować złożony układ strony www.

Przyjrzyjmy się grafice poniżej, gdzie znajduje się najprostszy przykład zastosowania float. Obrazek z deklaracją **float: right;** został przesunięty na prawą stronę, a tekst opływa go z lewej.



Warto również zwrócić uwagę na zmianę wyświetlania przy korzystaniu z floatów. Jeżeli przykładowo nadamy **float: left;** elementowi ****, który domyślnie wyświetlany jest jako **inline**, wówczas zmieni to też jego miejsce w przepływie dokumentu i spowoduje, że stanie się on elementem z właściwościami blokowymi. Oznacza to, że będzie można mu nadać również np. właściwości **width** oraz **height**.

Aby lepiej zrozumieć działanie **float**, warto poznać też właściwość **text-align** (wartości: **left** - domyślna, **right**, **center**, **justify**). Ta druga służy do wyrównywania tekstu najczęściej w elemencie blokowym (coś jak w MS Word). W efekcie zastosowania **text-align** w tej "przestrzeni wyrównywania" używając **text-align** nie ma już miejsca na coś, co to opływa. Natomiast kiedy używamy **float** to nasz element może się wcisnąć między inne.

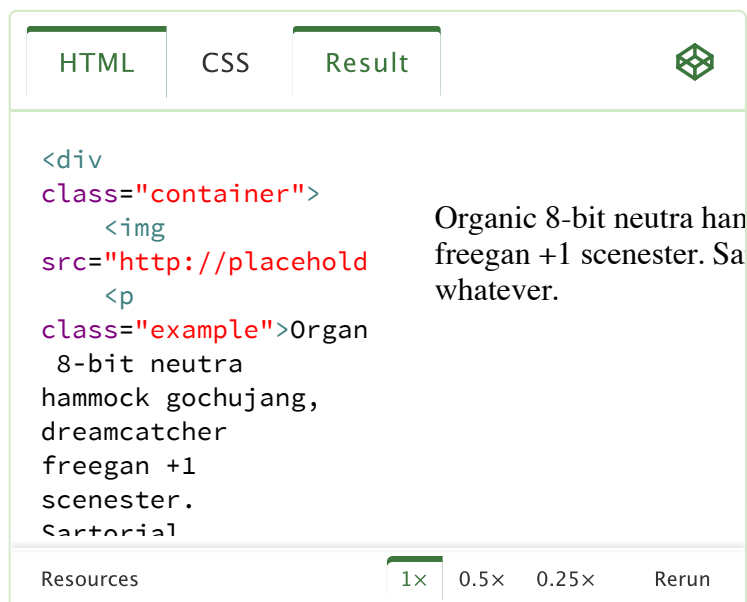


(powyżej sprawdź jak działa float i text-align, pozmieniaj ich wartości w edytorze Codepen!)

Clear

Elementy z właściwością **float** mogą czasami zachowywać się w sposób mało przewidywalny. Mówiąc wprost, to po części działa po prostu błędnie (ku zgrozie wielu developerów), dlatego zaczęto prace nad *flex-boksem* (o nim w kolejnych modułach).

Problemem floata jest to, że elementy z taką właściwością czasami "uciekają" poza swój kontener. Mamy `<div>` z jakimś tłem, a w nim tekst i zdjęcie, które ma być po prawej stronie tego tekstu, więc `` będzie mieć **float: right;**. Jeśli zdjęcie jest wyższe niż tekst po lewej, "wyjedzie" ono na dole poza granice diva. Aby temu przeciwdziałać, korzystamy z właściwości **clear**, a w zasadzie z pewnej gotowej reguły CSS z użyciem tej właściwości. Poniżej znajduje się przykład:



Błędne działanie floata zobaczysz, gdy usuniesz lub zakomentujesz w edytorze z linku ostatnią regułę w CSS. Teraz ją przywróć. Ta reguła naprawia działanie floata. Nie będziemy tłumaczyć jej konstrukcji, wielu programistów nawet nie zna jej na pamięć i kopiuje ją z projektu do projektu. Służy ona do wspierania kontenerów, które zawierają elementy "pływające" :)

Floaty często używa się też do budowy layoutu, jednak jak widać przysparza to pewnych problemów. Wynika to z tego, że **float** w założeniu miał służyć jedynie do wyżej wymienionego otaczania ilustracji tekstem. Wykorzystanie tej właściwości do



rozmieszczenia elementów na stronie (layout) jest pewnego rodzaju **trikiem**, nie do końca zgodnym z zamierzeniami twórców. Niezależnie od tego, trik ten jest wielokrotnie stosowany i można uznać, że stał się on standardem.

Pozycje

Zdarza się, że chcemy precyzyjnie określić, w którym miejscu ma się znaleźć element HTML na stronie. Sposób umieszczenia elementu na stronie oraz jego ewentualny wpływ na przepływ treści pozostałej części dokumentu określany jest za pomocą właściwości **position** (innymi słowy właściwość position określa sposób pozycjonowania w CSS). Najczęściej stosowany jest z podaniem pozycji elementu za pomocą podania offsetu (przesunięcia) w poszczególnych kierunkach (**top**, **bottom**, **left**, **right**).

Sposób pozycjonowania określamy w CSS za pomocą właściwości **position**. Poniżej opisano jej główne wartości, które może przyjmować.

Static

Domyślną wartością dla tej właściwości jest **static**, co oznacza, że element będzie umieszczony w normalnym przepływie układu strony, a deklaracje offsetu będą ignorowane.

Relative

Nadanie elementowi pozycji **relative** oraz wskazanie tzw. offsetu przesunie go względem pierwotnego położenia, jak w poniższym przykładzie. Taka reguła przesunie wszystkie elementy z klasą **example** o **20px** w lewo względem ich pierwotnej pozycji.

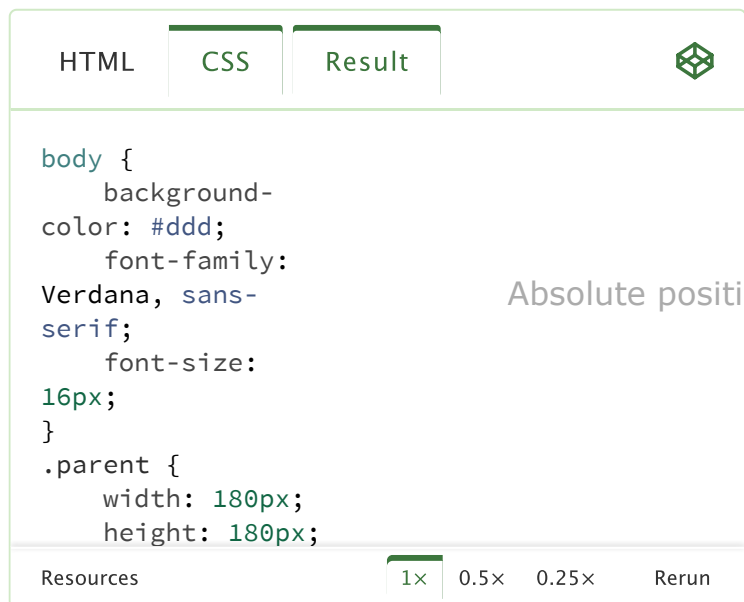
```
1 | .example {  
2 |     position: relative;  
3 |     left: 20px;  
4 | }
```

Absolute

Sprawy komplikują się nieco w przypadku pozycjonowania absolutnego (**absolute**). Element z takim pozycjonowaniem zostaje usunięty z przepływu treści dokumentu (tzn. jego położenie nie będzie mieć wpływu na pozostałą treść), a jego pozycja będzie określona **względem najbliższego rodzica tego elementu o pozycji innej niż static**. Rodzic (kontener) to inaczej element, wewnątrz którego zagnieźdzona jest interesująca



nas treść. Zwróć uwagę na poniższy kod. Znajduje się tam przykład rozwiązania korzystającego z pozycjonowania absolutnego. Element `div` z klasą **parent** ma pozycję **relative**, jednak bez podania offsetu nie wpływa to w żaden sposób na jego położenie. Jego dziecko, oznaczone klasą **example**, ma przypisaną pozycję absolutną, przesunięcie o 55 pikseli w górę oraz 30 pikseli w lewo. Warto również zaznaczyć, że właściwości odpowiadające za przesuwanie elementów mogą przyjmować wartości ujemne.



Fixed

Przejdźmy do wartości **fixed**. W tym wypadku element będzie pozycjonowany względem okna przeglądarki. Oznacza to, że pozostanie w jednym miejscu, gdy będziemy przewijać stronę. Być może widziałeś np. pasek nawigacji przyklejony do góry strony **albo spis treści, który widzisz obok :)** Jest to właśnie przykład elementu z pozycją **fixed**.

Zasady przemieszczania go są identyczne jak w przypadku **absolute**. Aby stworzyć belkę przyklejoną do góry przeglądarki, jak w przykładzie poniżej, nadajemy elementowi **position: fixed;** oraz odległość od górnej krawędzi okna przeglądarki **top: 0;**.



Tyle wystarczy na razie o pozycjonowaniu. Poznamy teraz nieco bardziej zaawansowane techniki CSS, które pozwolą nam na precyzyjne określanie wyglądu naszej strony.

Zagnieżdżanie selektorów

Poznaliśmy już selektory dla tagów, klas oraz identyfikatorów. Co jednak w sytuacji gdy chcemy zmienić wygląd linków (kilku), ale tylko tych, które znajdują się w nagłówku? Możemy oczywiście stworzyć dla nich odpowiednią klasę. Jednak każdorazowe dodawanie klasy do linku będącego w nagłówku to stosunkowo duży nakład pracy oraz spora ilość zbędnego kodu.

Aby osiągnąć interesujący nas efekt możemy skorzystać z zagnieżdżania selektorów. Wygląda to następująco:

```
1 | header a {  
2 |     /* Styles for the links inside header */  
3 | }
```

Dzięki temu możemy określić wygląd tylko tych linków, które znajdują się wewnątrz elementu **header**. Możemy też korzystać z większej liczby selektorów:

```
1 | header .main-navigation a {  
2 |     /* Styles for the links inside .main-navigation inside header  
3 | }
```

Należy jednak przy tym pamiętać, że zagnieżdżanie selektorów ma swoją cenę w postaci prędkości ładowania strony. Dobrą praktyką jest niezagnieżdżanie zbyt dużej ilości selektorów.

Podobnie działa selektor wybierający element będący bezpośrednim potomkiem, który tworzymy za pomocą znaku `>`.

```
1 | .example > div {  
2 |     /* Properties of divs that are direct inside .example */  
3 | }
```

W przykładzie powyżej, z użyciem znaku `>`, wybierzemy tylko te elementy `div`, które są **bezpośrednim** potomkiem (bezpośrednio zagnieżdżone) elementu HTML o klasie `example`. Elementy położone o dalsze poziomy niż w drzewie dokumentu HTML nie będą objęte działaniem tego stylu.

Grupowanie

Kolejnym przydatnym skrótem przy pracy z CSS jest **grupowanie** selektorów. Możemy z niego skorzystać, gdy chcemy ten sam kod CSS przypisać wielu klasom lub elementom. Zamiast wpisywać go każdorazowo do wielu elementów, możemy zebrać powtarzające się właściwości i przypisać je do grupy selektorów. Robimy to wymieniając je po prostu po przecinku:

```
1 | h1, h2, h3 {  
2 |     text-align: center;  
3 |     font-weight: bold;  
4 | }
```

W ten sposób przypisaliśmy za pomocą jednej reguły ten sam styl do trzech typów nagłówków.



Wybieranie elementów-rodzeństwa

Możemy wybrać za pomocą selektorów CSS elementy będące rodzeństwem danej klasy bądź elementu. Służą do tego selektory `+` oraz `~`.

Aby wybrać element występujący jako następny możemy skorzystać z konstruktora `+` (konstruktor to łącznik selektorów), tak jak we fragmencie poniżej. W tym wypadku tworzymy regułę tylko dla jednego elementu - w przykładzie poniżej będzie to `<p>` będące bezpośrednio po elemencie z klasą `example`.

```
1 | .example + p {  
2 |     /* adjacent sibling selector */  
3 | }
```

Co by było w wypadku, gdy będziemy chcieli wybrać **wszystkie** paragrafy będące bezpośrednim rodzeństwem klasy `example`? Wówczas z pomocą przychodzi nam selektor `~`, tak jak w przykładzie:

```
1 | .example ~ p {  
2 |     /* general sibling selector */  
3 | }
```

The screenshot shows a web development tool interface with three tabs: HTML, CSS, and Result. The HTML tab is active, displaying the following code:

```
<div class="cont">  
  <div  
    class="example">  
    <p>Hi! I  
    am a paragraph! I  
    am inside the  
    "example"  
    class</p>  
  </div>  
  <p>I am also a  
  paragraph, but I  
  am outside!</p>  
</div>
```

The Result panel shows the rendered output. The first paragraph, "Hi! I am a paragraph! I am inside the 'example' class", is highlighted with a green border. The second paragraph, "I am also a paragraph, but I am outside!", is rendered as a level 1 heading, "I am a level 1 heading".

W powyższym przykładzie możesz sprawdzić zagnieżdżanie i grupowanie selektorów oraz wskazywanie rodzeństwa



Pseudoklasy

Pseudoklasy pozwalają nam na wpływanie na wygląd elementów dokumentu, gdy znajdzie się on w pewnym stanie. Przez zmianę stylów CSS określonych za pomocą selektora pseudoklasy możemy sprawić, że jej wygląd będzie się zmieniał pod wpływem interakcji z użytkownikiem.

Jednym z najczęstszych zastosowań pseudoklas jest zmiana wyglądu odnośników (linków), gdy użytkownik najedzie na niego myszką. Efekt taki możemy osiągnąć za pomocą pseudoklasy **:hover**. W przypadku pseudoklas korzystamy tylko z jednego dwukropka, jak w przykładzie poniżej:

```
1 | a:hover {  
2 |     color: red;  
3 | }
```

Taki zapis sprawi, że element `<a>`, po najechaniu na niego kursorem, zmieni kolor na czerwony.

Inną popularną pseudoklasą typową dla odnośników jest **:visited**, dzięki której możemy określić wygląd odnośnika, który został już odwiedzony.

```
1 | a:visited {  
2 |     color: green;  
3 | }
```

Wybrany (kliknięty, ale jeszcze nie odwiedzony) link może wyglądać jeszcze inaczej dzięki pseudoklasie **:active**.

```
1 | a:active {  
2 |     color: #0000FF;  
3 | }
```

Pseudoklasa **:hover** dostępna jest również dla innych elementów niż odnośnik. Przykład poniżej zmienia tło dla pozycji na liście wewnątrz nagłówka po najechaniu na niego myszką.



```
1 | header li:hover {  
2 |     background-color: #52B3D9;  
3 | }
```

PAMIĘTAJ: `a:hover` musi występować po `a:link` i `a:visited` w kolejności definiowania stylów w CSS. Podobnie `a:active` piszemy po `a:hover`!

N-ty potomek

Innym interesującym rodzajem pseudoklas są te umożliwiające nam wybranie elementów będących n-tym w kolejności potomkiem swojego elementu rodzica. Możemy w ten sposób np. zdefiniować unikalny wygląd dla pierwszego paragrafu danej sekcji lub wybrać tylko interesującą nas pozycję na liście. Możemy do tego wykorzystać następujące pseudoklasy:

```
1 | p:first-child {  
2 |     /* First paragraph of parent div */  
3 | }  
4 | li:last-child {  
5 |     /* Last item in the ul or ol list */  
6 | }  
7 | p:nth-child(3) {  
8 |     /* Paragraph, being the n-th child of the parent - in this e  
9 | }
```

Wystarczy już teorii o CSS. Możesz do niej stale wracać lub posiłkować się innymi źródłami. W kolejnych submodułach zajmiemy się budowaniem komponentów CSS!

Sugestia

Jeżeli chcesz dowiedzieć się więcej na temat selektorów warto przejść ten kurs [Selektory CSS](#), aby przećwiczyć w praktyce podstawowe zagadnienia oraz dowiedzieć się o innych ciekawych selektorach i ich kompilacjach :)

Jeśli masz wątpliwości do powyższego materiału, to - zanim zatwierdzisz - zapytaj na czacie :)

✓ Zapoznałem(a) się!



2.2. Komponent: zafiksowane menu dwupoziomowe



W tym submodule zbudujemy menu wielopoziomowe przy użyciu poznanych w poprzednich sekcjach technik HTML i CSS. Takie zadanie może się zdarzyć na rozmowach kwalifikacyjnych na stanowisko związane z front-endem. Przy rozwiązywaniu możesz korzystać z materiałów znajdujących się w poprzedniej sekcji.

UWAGA: Możesz czytać poniższą instrukcję wykonania lub - jeśli wiesz już więcej, przewinąć stronę na koniec **tego submodułu do zadania** i próbować samemu wykonać je według przedstawionej grafiki (bez instrukcji). Do instrukcji zajrzyj dopiero w razie problemów.

Struktura projektu

Zacniemy od zbudowania szkieletu HTML dla naszej strony.

1. Wstaw poprawną, podstawową strukturę stron do pliku *index.html* i podepnij do niej plik *style.css* :)
2. Wprowadź do pliku *index.html* element `<nav>`.
3. Wewnątrz niego dodaj nieuporządkowaną listę wraz z kilkoma pozycjami (``) oraz wprowadź przykładowe teksty do tych elementów listy np. O mnie, Kontakt itp.
4. Każda przeglądarka dodaje nam domyślne style do list, dlatego dla selektora `ul` musisz wyzerować `padding` i margines oraz wyłączyć punktory (za pomocą właściwości `list-style-type`).
5. Wewnątrz każdej pozycji na liście powinien znaleźć się też znacznik `<a>` który póki co nigdzie nie będzie prowadził (atrybut `href` równy `#`).

Gdy HTML jest już gotowy, zastanowimy się nad tym, jak zdefiniować właściwości CSS. Najpierw dodamy style, które będą miały zastosowanie w całym dokumencie.

1. Zaimportuj z Google Fonts czcionkę "Quicksand" oraz dodaj ją do pliku. Zrób to podobnie jak w poprzednim module.
2. Dodaj następujące style dla znacznika body: `font-family: "Quicksand"; margin: 0; background: lightblue;`



Poziome menu pierwszego poziomu

Po dodaniu odpowiednich stylów czas stworzyć menu pierwszego poziomu.

1. W pliku **style.css** dodaj deklarację **display: inline-block;** dla elementów listy **** znajdujących się wewnątrz elementu **<nav>**. To wyświetli elementy listy poziomo.
2. Dodajmy style odpowiedzialne za wygląd nawigacji. Dla znacznika **<nav>** nadaj kolor tła **#4577ED**. Wyśrodkuj tekst.
3. Ostylujemy teraz nasze "przyciski" wewnątrz listy. Czyli tagi **<a>**. Określ ich wysokość na **50px** (nie zapomnij o tym że **<a>** są domyślnie inlinowe więc musisz im dodać **display: inline-block;**) i dodaj wysokość linii (**line-height**) również równą **50px**. To wyśrodkuje ich zawartość w pionie. Dodaj też **padding: 0 10px;**, to sprawi że elementy "odkleją" się od siebie.
4. Do tych samych linków **<a>** dodaj **color: white** oraz **text-decoration: none**.
5. Całość wygląda już przyjemnie jednak nasza nawigacja aż się prosi o to żeby dodać jakiś efekt na **:hover** dla naszych przycisków :) Sugerujemy kolor tła **#3255a7**. Ambitniejsi mogą też zmierzyć się z **efektem przejścia dla zmiany koloru**.

Rozwijane menu drugiego poziomu

Teraz możemy dodać menu drugiego poziomu. Będzie się wyświetlać dopiero po najechaniu na link pierwszego rzędu. Zaczniemy od HTML.

1. W pliku **index.html**, wewnątrz każdego elementu ****, zaraz za elementem **<a>** dodaj element **** wraz z kilkoma pozycjami oraz linkami, tak jak poprzednio. Element ten powinien znaleźć się na tym samym poziomie zagnieżdżenia, co uprzednio dodane linki.
2. Ustawmy elementy naszego menu drugiego poziomu w pionowym rzędzie. W tym celu dodaj selektor **nav ul ul li** oraz dodaj do niego odpowiednią własność - **display: block;**.
3. Aby ukryć menu niższego poziomu (póki co będzie schowane :)), stwórz regułę **display: none;** dla list znajdujących się wewnątrz innych list w panelu nawigacji. Skorzystaj w tym celu ze znanych nam zagnieżdżonych selektorów CSS.

Wyświetlamy listę drugiego poziomu



Przeanalizujemy strukturę naszego menu. Link drugiego poziomu jest elementem listy pierwszego poziomu. Aby wybrać listy drugiego rzędu stworzylibyśmy następujący selektor: `nav li ul { }`.

W ten sposób wybieramy listy `` będące wewnątrz elementów listy pierwszego poziomu - tj. znajdujące się wewnątrz taga ``. Chcemy wyświetlić te listy dopiero w momencie, gdy użytkownik najedzie kursorem na element ``, zatem nasz selektor powinien wyglądać następująco: `nav li:hover ul { ... }`. Do wyświetlenia listy wystarczy ustanowienie odpowiedniej wartości właściwości `display` wewnątrz takiego selektora - w tym wypadku innej niż `none`.

1. Stwórz selektor wybierający listy drugiego rzędu po najechnięciu myszką na element listy pierwszego rzędu, tak jak w opisie powyżej.
2. Dodaj właściwość `display: block` wewnątrz utworzonego wyżej selektora. Menu w tym momencie zachowuje się dość dziwnie ale już to naprawiamy :)
3. Żeby lista drugiego poziomu nie rozpychała nam nawigacji, musimy ją "wyciągnąć z obiegu". Uzyskamy to dzięki dodaniu (tej zagnieżdżonej liście) `position: absolute;`.
4. Została nam ostatnia sprawa do załatwienia. Nasza lista drugiego poziomu jest jakaś taka przeźroczysta. Dajmy jej tło, najlepiej takie samo jak ma cała nawigacja :)
5. Ok, już jest fajnie, ale jak najedziemy na linki drugiego poziomu to widzimy, że ich podświetlenie nie jest na całą szerokość nawigacji drugiego poziomu. Jak się już pewnie domyślasz, dzieje się tak przez wzgląd na to, że linki (których tło zmieniamy po najechnięciu) nie zajmują całej dostępnej przestrzeni. Zachowanie, którego od nich oczekujemy, zapewni nam odpowiednie wyświetlanie tych linków czyli `display: block;` **Uwaga!** Tworząc selektor, upewnij się, że wyświetlanie blokowe będą miały tylko linki nawigacji drugiego poziomu! W przypadku linków nawigacji pierwszego poziomu nadal chcemy żeby były liniowo-blokowe :)

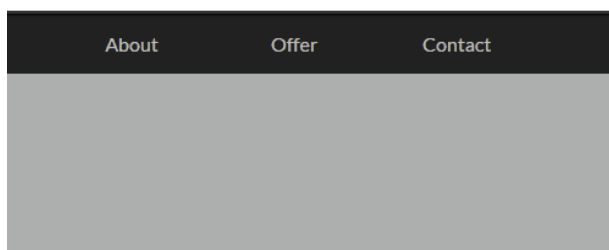
Voila! Menu pojawiać się będzie po najechnięciu kursorem na link pierwszego rzędu.

Zadanie: menu dwupoziomowe

Wykonaj komponent nawigacji zgodnie z opisem powyżej lub wyłącznie na bazie grafiki (animacji).

Przykład:





Podgląd zadania

Przejdź do projektu ✓

2.3. Komponent: galeria obrazków w oparciu o float oraz prosty jumbotron



Jumbotron

W tym submodule stworzymy obrazek zajmujący całą dostępną szerokość wraz z nagłówkiem (tzw. *jumbotron* - o pochodzeniu tej nazwy dowiesz się w rozdziale poświęconemu Bootstrapowi). Dodamy do niego elegancką galerię obrazków korzystających z właściwości **float**. Zbudujemy do tego celu własną siatkę elementów o równej szerokości. To popularne rozwiązanie, które z pewnością widziałeś na niejednej stronie internetowej.

Jumbotron zazwyczaj zajmuje całą dostępną szerokość oraz wysokość strony i natychmiast przykuwa uwagę odwiedzającego. Najczęściej umieszcza się w nim duży nagłówek z hasłem zachęcającym użytkownika do zapoznania się z treścią strony lub wykonania innej czynności.

UWAGA: Możesz czytać poniższą instrukcję wykonania elementu lub - jeśli wiesz już więcej - przewinąć stronę na koniec **tego submodułu do zadania** i spróbować wykonać je samodzielnie według przedstawionej grafiki (bez instrukcji). Do instrukcji zajrzyj dopiero w razie problemów.

Budujemy Jumbotron - struktura HTML



1. Utwórz sekcję (<section>) z klasą **jumbotron**.
2. Wewnątrz dodaj nagłówek <h1>Feel The Nature</h1>.

Obraz w tle

Teraz dodamy i dostosujemy obraz w tle. Podążaj za instrukcjami.

Dla reguły o selektorze **.jumbotron** dodaj następujące właściwości:

1. szerokość **100%**,
2. wysokość **100vh**,
3. górny **padding 42vh**,
4. wyśrodkowanie tekstu,
5. obraz w tle **url(http://kodilla.com/static/lessons/blue-mountain.jpg)**,
6. **background-size: cover;**,
7. **background-position: center;**,
8. i **overflow: hidden;**.

Stylujemy napis

Teraz ostylujemy napis, aby w odpowiedni sposób zwracał uwagę użytkownika.

Utwórz regułę, która dla **h1** będącego potomkiem jumbotrona nada następujące właściwości:

1. margines **0**
2. czcionkę rozmiaru **38px**, grubości **500**, koloru białego i rodzaju "Quicksand".

Dodajemy przycisk

Teraz dodamy przycisk, który ma na celu zachęcenie użytkownika do wykonania akcji.

W kodzie HTML, wewnątrz sekcji z klasą **jumbotron**, dodaj element <button> z napisem "Start adventure", a następnie ostyluj go dodając następujące właściwości:

1. margines górny **20px**,
2. **padding: 15px 40px;**,
3. kolor tła **#4577ED**,
4. czcionka rozmiaru **20px** koloru białego i rodzaju "Quicksand",
5. **border: none;**.

Efekt na :hover

Na koniec dodamy lekki efekt podczas najechnania na nasz przycisk.

1. Dodaj regułę, która w momencie, gdy przycisk stworzony w poprzednim kroku będzie w stanie **:hover**, zmieni kolor jego tła na **#7E9FED**.
2. Dla selektora **.jumbotron button** dodaj **cursor: pointer;** i **transition: .3s**.



Tworzymy galerię - HTML

Szkielet naszej galerii jest bardzo prosty. Dodamy jedną sekcję wraz z kilkoma obrazkami.

1. Utwórz sekcję z klasą **gallery**.
2. Wewnątrz tej sekcji dodaj cztery obrazki. Wykorzystaj do tego celu listę obrazków znajdującą się poniżej. Pamiętaj o atrybucie **alt**

Grafiki do galerii:

- <http://i.imgur.com/AGEgF5B.jpg>
- <http://i.imgur.com/7Ff7rDB.jpg>
- <http://i.imgur.com/KUt6tGb.jpg>
- <http://i.imgur.com/Ue2tUMV.jpg>

Skorzystamy z właściwości **box-sizing**. Dzięki niej zdefiniujemy, że do rozmiarów elementów (**width** oraz **height**) będą doliczane ich ramki oraz **padding**.

1. Do pliku *style.css* dodaj poniższy fragment:

```
1  * {  
2      box-sizing: border-box;  
3  }
```

2. Do elementów **img** wewnątrz klasy **gallery** dodaj **float: left;** oraz **width: 25%;**. To ustawi obrazki w jednym rzędzie i wypełni nimi całą dostępną przestrzeń.

Miniaturowe obrazki w galeriach z reguły są klikalne i otwierają większą wersję zdjęć.

Aby zasugerować użytkownikowi, że dany element można kliknąć, obsłuż stan **:hover** dla obrazków w galerii i dodaj do niego własność **opacity: 0.8;** oraz **cursor: pointer;**.

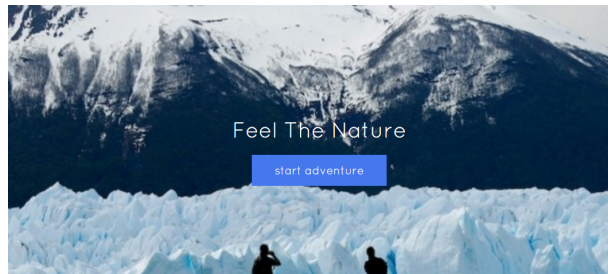
Zadanie: galeria i jumbotron

1. Zbuduj jumbotron zgodnie z instrukcjami w submodule.
2. Niżej, w tym samym projekcie, stwórz galerię, również zgodnie z instrukcjami z submodułu.



Zrób wg instrukcji lub ambitniej tylko na bazie przedstawionej grafiki.

Jumbotron:



Galeria (po najechaniu zdjęcie się rozjaśnia):



[Link do gifa z galerią](#)

Podgląd zadania

Przejdź do projektu ✓

2.4. Komponent: kolorowe przyciski z animacją opartą o zmianę gradientu



Sugestia

W kolejnych submodułach będziemy poruszać kwestie graficzne w CSS oraz tworzyć animacje. W związku z tym warto przejść ten kurs [Grafika i animacje w CSS](#), aby przeciwiczyć w praktyce podstawowe zagadnienia, co pomoże przy wykonywaniu zadań.



Zajmiemy się teraz stworzeniem kolorowych guzików z ciekawą animacją przy wskazaniu ich kursorem (hover). Tego rodzaju efekty są bardzo atrakcyjne i robią duże wrażenie na klientach, więc warto zapoznać się z tą techniką.

Czym jest gradient?

Gradienty w CSS pozwalają na stworzenie płynnych przejść pomiędzy kolorami tła. Posiadają one wiele różnorodnych właściwości pozwalających osiągać złożone efekty, jednak w tym ćwiczeniu skorzystamy ze stosunkowo prostych rozwiązań.

UWAGA: Możesz czytać poniższą instrukcję wykonania lub jeśli wiesz już więcej - przewinąć stronę na koniec [tego submodułu do zadania](#) i próbować samemu wykonać je według przedstawionej grafiki (bez instrukcji). Dopiero w razie problemów możesz posiłkować się instrukcjami.



Do zdefiniowania gradientu jak powyżej wystarczy nam następująca deklaracja:

```
1 | background: linear-gradient(red, blue);
```

W praktyce potrzebne jest też zastosowanie vendor prefixów, by gradient taki wyświetlił się prawidłowo we wszystkich przeglądarkach, więc nasz kod będzie wyglądał najczęściej następująco:

```
1 | .grad {  
2 |     background: red; /* for browsers that do not support gradient  
3 |     background: -webkit-linear-gradient(red, yellow); /* Safari  
4 |     background: -o-linear-gradient(red, yellow); /* Opera 11.1 -  
5 |     background: -moz-linear-gradient(red, yellow); /* Firefox 3.  
6 |     background: linear-gradient(red, yellow); /* standard syntax  
7 | }
```

Na potrzeby tego ćwiczenia będziemy korzystać ze standardowej składni pomijając prefiksy.

Aby kontrolować kąt pod jakim przebiega gradient, możemy też podać jego obrót w stopniach, tak jak poniżej:



```
1 | background: linear-gradient(-45deg, green, yellow);
```

Możemy też ustalać przezroczyste kolory:

```
1 | background-image: linear-gradient(45deg, #f1c40f 50%, transparen
```

Tworzenie animowanych przejść

Aby stworzyć płynne przejście pomiędzy jednym stanem a drugim (np. w przypadku użycia pseudoklasy `:hover`) skorzystamy z właściwości `transition`. Dzięki niej wystarczy zdefiniować czas trwania oraz właściwość, którą chcemy animować, aby otrzymać prostą animację. W poniższym przykładzie zmieniamy rozmiar elementu `<div>` po najechaniu na niego kursorem myszki przez użytkownika.

```
1 | div {  
2 |     width: 100px;  
3 |     height: 100px;  
4 |     background: red;  
5 |     transition: width 2s;  
6 | }  
7 | div:hover {  
8 |     width: 300px;  
9 | }
```

Aby lepiej zrozumieć przejścia, zobaczmy teraz jak zastosować animacje w praktyce!

Utwórzmy szkielet HTML

W tej lekcji zajmiemy się stworzeniem kolorowych, animowanych przycisków z wykorzystaniem gradientów. Jak zwykle powinniśmy zacząć od napisania szkieletu w kodzie HTML, a w następnych ćwiczeniach zajmiemy się stylowaniem.

1. Utwórz 3 przyciski `<button>`, w każdy wpisz dowolny tekst.
2. Do pierwszego przycisku dodaj klasę `btn-purple`, do drugiego `btn-yellow`, a do trzeciego `btn-blue`.



Stylowanie przycisków według klasy

Będziemy mieć 3 przyciski o - jak już pewnie się domyślasz, 3 różnych kolorach. Każdy z nich będzie posiadał pewną ilość wspólnego kodu CSS, który zgodnie z zasadą **DRY** (*Don't Repeat Yourself*), musimy wyodrębnić w oddzielną regułę.

W pliku *style.css* utwórz regułę rozpoczynającą się selektorem `[class^="btn-"]` i wewnątrz niej zadeklaruj następujące właściwości:

1. wyświetlanie liniowo-blokowe,
2. `margin: 4px 2px;` oraz `padding: 15px 30px;`,
3. minimalną szerokość równą 160 pikseli,
4. czcionkę o rozmiarze 16 pikseli i o stylu pogrubionym,
5. `border-radius: 5px;`,
6. oraz `cursor: pointer;`.

Kolor i gradient

Teraz pokolorujemy nasze przyciski i ustawimy tło jako gradient. W tej chwili nie zobaczymy jeszcze oczekiwanego efektu, ale w następnym ćwiczeniu gradient posłuży nam do zrobienia animacji.

1. Dla każdego z przycisków nadaj `background-color: inherit;` i dwupikselową, ciągłą ramkę.
2. Dla `btn-purple` nadaj kolor czcionki i ramki: `#8e44ad` oraz `background-image: linear-gradient(90deg, #8e44ad 50%, transparent 50%);`.
3. Dla `btn-yellow` nadaj kolor czcionki i ramki: `#f1c40f` oraz `background-image: linear-gradient(45deg, #f1c40f 50%, transparent 50%);`.
4. Dla `btn-blue` nadaj kolor czcionki i ramki: `#3498db` oraz `background-image: linear-gradient(135deg, #3498db 50%, transparent 50%);`.

Pozycjonowanie tła i przesunięcie na :hover

Teraz schowamy nasz gradient, a następnie dodamy kod, który sprawi, że po najechaniu kursorem na przycisk, gradient wypełni go w całości.

1. Dla każdego z przycisków ustaw: `background-position: 100% 0;` oraz `background-size: 300%;`.



2. Dla każdego z przycisków w stanie `:hover` ustaw `background-position: 0 100%`; i `color: white`;

Uruchamiamy animacje!

Teraz dodamy efekt przejścia (`transition`), dzięki któremu zobaczymy jak nasz gradient w tle zmienia swoją pozycję według reguł, które zadeklarowaliśmy wcześniej.

1. Na końcu reguły `[class^="btn-"]` dodaj: `transition: 0.4s;`.

Zadanie: gradientowe przyciski

Wykonaj komponent nawigacji zgodnie z opisem powyżej lub wyłącznie na bazie grafiki (animacji).

Przyciski:



Podgląd zadania

Przejdź do projektu ✓

2.5. Komponent: gwiazdka z wypełnieniem pojawiającym się po najechaniu kursorem



Tym razem naszym zadaniem będzie stworzenie bardziej skomplikowanych guzików. Będą one miały kształt gwiazdki oraz ikonę, a przy wskazaniu kursorem (hover) będą zmieniać kolor tła i ikony.

Tego rodzaju elementy na pierwszy rzut oka mogą wyglądać jakby były stworzone w programie graficznym i wyświetlone jako obrazki. Dzięki stworzeniu ich za pomocą CSS będziemy jednak mogli płynnie animować zarówno kolor tła jak i ikony, osiągając bardzo ciekawy efekt.

Przekonasz się też jak duże możliwości dekoracji daje nam korzystanie z pseudoelementów w połączeniu z transformacjami (transform) oraz przejściami (transition).

Transformacje CSS

W tym zadaniu wykorzystamy transformacje CSS. Dzięki nim możemy elementy obracać, powiększać lub zmieniać ich nachylenie.

Definiujemy je w następujący sposób:

```
1 | div {  
2 |     -ms-transform: translate(50px,100px); /* IE 9 */  
3 |     -webkit-transform: translate(50px,100px); /* Safari */  
4 |     transform: translate(50px,100px);  
5 | }
```

Takie deklaracje pozwalają przesunąć element o 50 pikseli w prawo i 100 pikseli w dół.

Innym interesującym nas rodzajem transformacji jest `rotate()` oraz `scale()`. Pierwszym z nich możemy obrócić interesującą nas treść o określoną ilość stopni - w tym wypadku o 20 stopni w lewo.

```
1 | transform: rotate(-20deg);
```

Za pomocą `scale()` możemy elementy powiększać i pomniejszać:



```
1 | transform: scale(2, 3);
```

Określamy to za pomocą proporcji: w przykładzie powyżej powiększamy szerokość elementu dwukrotnie, zaś wysokość trzykrotnie.

Pseudoelementy


Kolejnym rozwiązaniem, które tutaj zastosujemy, są tzw. pseudoelementy. Pozwalają one na stylowanie części elementu (pierwsza litera, pierwsza linia) lub dodawanie jakiejś treści "przed" lub "za" interesującym nas elementem za pomocą czystego CSS. Tworzymy je, dodając do selektora danego elementu pseudoelement, tak jak w przykładzie poniżej. Treść określamy za pomocą właściwości **content**. Tutaj dodajemy tekst przed nagłówkiem.

```
h1::before {  
  content: "Nagłówek: ";  
}
```

HTML

CSS

Result



```
p::before {  
  content: "Witaj,  
wyświetlam się  
przed "  
}  
p::after {  
  content: "em, a  
ja za tekstem!"  
}
```

Witaj, wyświetlam się
przed tekstem, a ja za
tekstem!

Resources

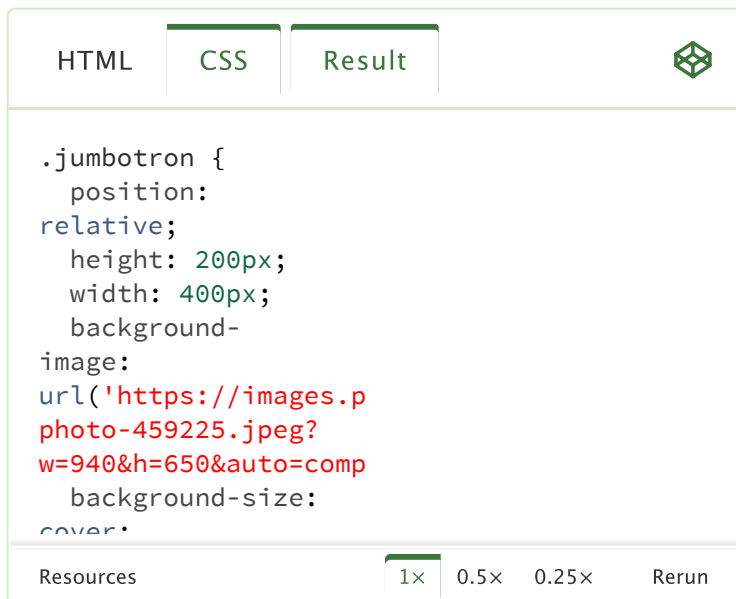
1x

0.5x

0.25x

Rerun

Powyżej pokazaliśmy nałatwiejszy sposób użycia pseudoelementu. Zaraz zajmiemy się trudniejszym przypadkiem:



Jak widzisz, nasz pseudoelement otrzymał pozycję **absolute**. Teraz my decydujemy, jak będzie się zachowywał i gdzie go umieścimy. Zauważ także, że rodzic pseudoelementu (w tym przykładzie element **.jumbotron**) otrzymał pozycję **relative**, więc pseudoelement będzie pozycjonowany względem niego.

Tego typu pseudoelementy wykorzystywane są często do dodawania różnych ozdób przy użyciu CSS. Jeśli ciekawość nie daje Ci spokoju i chcesz zobaczyć, co można zrobić za pomocą innych pseudoklas, odwiedź [link](#).

::before czy :before ?

Podwójny dwukropek występuje dla rozróżnienia pseudoelementów od pseudoklas. Taki zabieg pojawił się wraz z wydaniem wersji CSS3 i w najnowszych standardach jest to zapis jak najbardziej poprawny.

Szkielet HTML

W tej sekcji nauczysz się jak stworzyć gwiazdkę, która wypełni się kolorem po najechaniu na nią kursorem.

Zacznijmy od stworzenia szkieletu HTML, który ostylujemy w następnych zadaniach.

Skorzystamy tutaj z popularnego zasobu ikon jakim jest [Font Awesome](#). Aby to zrobić, wystarczy dodać gotowy plik ze stylami do naszej strony. Ikony dodaje się następnie poprzez zastosowanie pustego elementu `<i></i>` wraz z odpowiednimi klasami, np. `<i class="fa fa-twitter"></i>` dodaje ikonę Twittera. W ten sposób dodajemy jako tekst, więc możemy modyfikować ich wygląd za pomocą czystego CSS.



Krok 1

Aby skorzystać z Font Awesome wewnątrz `<head>` dodaj link do pliku:

`https://maxcdn.bootstrapcdn.com/font-awesome/4.5.0/css/font-awesome.min.css`

Pamiętaj, że aby swobodnie nadpisywać jego domyślne właściwości w Twoim pliku `style.css`, link do arkusza `font-awesome.min.css` musi znajdować się **przed** linkiem do Twojego pliku CSS.

Krok 2

Następnie stwórz 3 elementy `<div>`. Pierwszemu nadaj klasy `star8 gold`, drugiemu `star8 orange`, a trzeciemu `star8 blue`.

Krok 3

Potem w każdym z nich umieść po jednej ikonie. W pierwszej wstaw `<i>` z klasami `fa fa-hand-o-up`, w drugiej `fa fa-diamond`, a w trzeciej `fa fa-flask`.

Tworzymy gwiazdkę

W tej części stworzymy kształt gwiazdki. Ośmioramienna gwiazda może być stworzona poprzez nałożenie na siebie dwóch kwadratów, obróconych pod kątem 45° względem siebie. Skorzystamy z pseudoelementu `:before`. Każda z gwiazdek ma nadaną klasę `star8`, dzięki czemu możemy stworzyć pewien wspólny kod, odnoszący się do wszystkich trzech gwiazd.

Krok 4

Napisz selektor klasy `star8` i wpisz w nim następujące wartości: wyświetlanie liniowo-blokowe, `margin: 2em`, `transform: rotate(22deg)`; oraz `transition: background linear .3s`.

Krok 5

Następnie utwórz selektor `.star8:before` i za jego pomocą ustaw: biały kolor tła, `content: ""`; (bez tego nasz dodatkowy element nie wyświetli się), `position: absolute`;, `top: -5px`;, `left: -5px`; oraz obrót o 45° za pomocą właściwości `transform`.

Krok 6



Kolejnym krokiem jest napisanie selektora wybierającego aż 3 różne klasy: `.star8`, `.star8::before`, i `.fa`. Ustaw szerokość i wysokość tych elementów na `80px`.

Teraz stwórz selektor CSS, który wybierze element o klasie `gold` oraz `gold::before`. Ustaw wartość ramki (`border`) na `6px solid #ffc107`.

Krok 7

Postępując analogicznie, ustaw ramkę elementów `.orange`, `.orange::before` na `6px solid #fd7e33`, a w przypadku elementów `.blue`, `.blue::before` na `6px solid #3fb0ac`.

Ikony

W tej części zajmiemy się ikonami. Powiększymy je i sprawimy, by nie były ustawione krzywo jak teraz.

Krok 8

Napisz selektor wybierający ikony z nadaną klasą `fa`. Użyj go do ustawienia właściwości: `margin: 0;`, `position: absolute;`, białego tła i rozmiaru czcionki `60px`.

Krok 9

Następnie, korzystając z pseudoelementu `::before`, utwórz selektor dotyczący tego, co znajduje się "przed" elementami wybranymi w poprzednim kroku. Obróć ikony o `-22°`, ustaw `position: absolute;` i `margin: 10px`.

Ożywmy gwiazdki!

Twoim gwiazdkom wciąż brakuje wypełnienia, które pojawiałoby się po najechaniu na nie kursorem. Ikony również nie reagują w żaden sposób na działania użytkownika Twojej strony.

Ostatnim zadaniem będzie sprawienie, by po najechaniu kursorem na gwiazdkę, wypełniała się kolorem ramki, a ikona zmieniała swój kolor z czarnego na biały. Czas się tym zająć!



Krok 10

Napisz selektor `.gold:hover::before`, `.gold:hover`, `.gold:hover i.fa` i ustaw kolor tła złotej gwiazdki na `#ffc107`.

Krok 11

Niech pomarańczowa gwiazdka ma tło w kolorze `#fd7e33`. Wykorzystaj w tym celu selektor `.orange:hover::before`, `.orange:hover`, `.orange:hover i.fa`.

Krok 12

A teraz ustaw tło gwiazdki niebieskiej na `#3fb0ac` za pomocą selektora `.blue:hover::before`, `.blue:hover`, `.blue:hover i.fa`.

Krok 13

Ostatnim krokiem będzie dodanie łagodnego przejścia (zmiany koloru). Utwórz selektor `.gold`, `.gold::before`, `.gold i.fa` i umieść w nim deklarację `transition: ease-in-out .3s`. Pamiętaj, że ikona po najechaniu na gwiazdkę ma kolor biały! Dodaj też analogiczny kod dla pozostałych kolorów gwiazdek.

Zadanie: Gwiazdki oceny

Wykonaj komponent nawigacji zgodnie z opisem powyżej lub wyłącznie na bazie grafiki (animacji).

Gwiazdki:



Podgląd zadania

Przejdź do projektu ✓



2.6. Komponent: panel mediów społecznościowych.



Dodamy do strony element odpowiedzialny za linki do naszych profili w różnorodnych mediach społecznościowych. Stworzymy w tym celu kolejną sekcję z pięcioma przyciskami. Każdy z nich będzie prowadził do innego portalu społecznościowego.

Wstępne przygotowanie i HTML

Standardowo zaczynamy budowę od markupu HTML. Skorzystamy tutaj ponownie z ikon Font Awesome.

Krok 1

Wewnątrz części `<head>` dokumentu umieść link do arkusza CSS `https://maxcdn.bootstrapcdn.com/font-awesome/4.5.0/css/font-awesome.min.css`. Tak samo jak w przypadku poprzedniego komponentu :)

Krok 2

W pliku `index.html` dodaj sekcję z klasą `social`, a wewnątrz niej 5 przycisków `<button>`.

Krok 3

Następnie wewnątrz przycisków dodamy ikony z Font Awesome. Wewnątrz każdego przycisku dodaj po jednej ikonie z listy poniżej:

- `<i class="fa fa-twitter"></i>`
- `<i class="fa fa-facebook"></i>`
- `<i class="fa fa-google-plus"></i>`
- `<i class="fa fa-instagram"></i>`
- `<i class="fa fa-github"></i>`

Wspólne właściwości przycisków oraz ikon.



Jak zapewne zauważysz, przyciski posiadają pewne właściwości wspólne, np. wymiary, jednak posiadają też style unikalne, takie jak kolory ikon. Aby unikać powtarzania dodamy do każdego przycisku wspólną klasę **icon-btn** oraz klasę unikalną odpowiadającą nazwie portalu społecznościowego.

Krok 4

Do każdego przycisku dodaj wspólną klasę **icon-btn** oraz po jednej klasie unikalnej (odpowiednio **twitter**, **facebook**, **google-plus**, **instagram**, **github**).

Krok 5

Przed przyciskami dodaj znacznik **h2** wraz z tekstem "Say hello!". Orz przypisz mu następujące style. Kolor **#505050** oraz wielkość czcionki **40px**.

Krok 6

Stwórz selektor dla klasy **social** i dodaj **margin: 0 auto;**, szerokość 500 pikseli, wyśrodkowanie tekstu, **padding: 50px;**.

Krok 7

Przejdźmy do ustalenia wymiarów naszych elementów - będziemy się posługiwać klasą **.icon-btn**. Na początek, przypisz jej następujące wymiary: wysokość i szerokość równe 75 pikseli. Ustal obramowanie na 0, zaś kolor na **#FFFFFF**. Nadaj im także **border-radius: 50%** i marginesy.

Krok 8

W przypadku przycisków kursor myszki często zmienia się na wskaźnik. Dlatego też dodaj właściwość **cursor: pointer** do wyżej wymienionej klasy.

Krok 9

Czas zająć się ikonami. Są one równego rozmiaru, dlatego dopisz do selektora **.icon-btn** i dodaj tam wielkość czcionki równą **40** pikselom.

Właściwości poszczególnych przycisków oraz hover

Krok 10



Wszystkie ikony wewnątrz przycisków początkowo są czarne. Aby to zmienić, stwórz dla każdej z nich odpowiedni selektor, a następnie przypisz im kolor tekstu:

- twitter: #4099FF
- facebook: #3B5998
- google-plus: #DB5A3C
- instagram: #5C3D2E
- github: #4183C4

Krok 11

Przy najechaniu kursorem myszki można zobaczyć dwie zmiany: przycisk się powiększa oraz zmienia się jego kolorystyka. Tekst staje się biały, tło zaś przyjmuje kolor poszczególnej ikony. Aby osiągnąć te efekty dodaj efekt **hover** dla wspólnej klasy **icon-btn** nadaj **transform: scale(1.10);**.

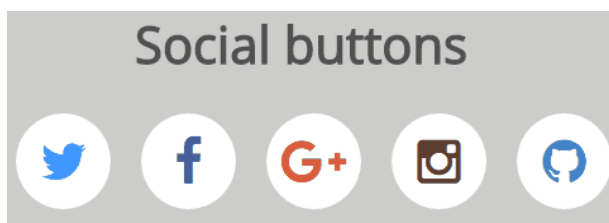
Krok 12

Aby zmienić kolor przycisku, stwórz dla każdej oddzielnej klasy efekt **hover**, ustalając dla niego kolor tła równy temu, jaki przypisaliśmy poprzednio ikonom, a kolor tekstu określ jako **#FFFFFF**. Stworzyliśmy panel mediów społecznościowych.

Zadanie: Panel social media

Wykonaj komponent nawigacji zgodnie z opisem powyżej lub wyłącznie na bazie grafiki (animacji).

Przyciski:



Podgląd zadania

Przejdź do projektu ✓



Regulamin

Polityka prywatności

© 2019 Kodilla

