### 

### 

### 

### Wstęp

* Front-end Developer - osoba odpowiedzialna za tworzenie warstwy wizualnej strony/serwisu WWW (na podstawie projektu od grafika) oraz dodawanie pewnych interakcji lub elementów dynamicznych (np. po kliku jakiegoś elementu coś się zmienia na stronie). Tworzy tzw. szablony lub konstruuje pełne strony internetowe, ale mające głównie charakter prezentacyjny. Jednak bardziej zaawansowani front-endowcy potrafią tworzyć także podstawowe rozwiązania funkcjonalne (interakcja z komunikacją z bazą danych). Taka osoba powinna znać m.in. technologie: HTML5, CSS3, SASS, wybrany framework front-end, podstawy JavaScript/jQuery.
* Back-end Developer - osoba odpowiedzialna za logikę aplikacji wpływającą na jej zachowania (funkcjonalności) i za konstrukcję bazy danych, w której są przechowywane dane wyświetlane w aplikacji (we front-endzie). Odpowiada ona także za wydajność i bezpieczeństwo rozwiązania. Zestaw potrzebnych technologii może być różny, przykładowo: język back-end - PHP lub Python, baza danych - MySQL lub MongoDB, system dla serwera - Linux lub Windows Server.
* JavaScript (JS) Developer - często koduje "na przecięciu" obu powyższych specjalizacji, budując zaawansowane interakcje we front-endzie lub wspierając komunikację pomiędzy front-endem a bazą danych w back-endzie. Zaawansowany JS Developer używa jednego z frameworków (bibliotek), np.: Angular, React, Backbone lub Ember.
* Full-stack Developer lub webdeveloper - doświadczona osoba, która łączy w sobie kompetencje front-end, back-end i JavaScript developera, czyli potrafi stworzyć w całości kompletną aplikację.

### Rozwiązania internetowe

* Strony internetowe - mające głównie rolę "wizytówki". Ich cechą jest to, że mają w zasadzie wyłącznie rolę informacyjną. Praktycznie nie posiadają elementów funkcjonalnych.
* Serwisy internetowe - bardziej rozbudowane rozwiązania np. portale społecznościowe, fora, serwisy informacyjne.
* Aplikacje internetowe - rozwiązania, które posiadają pewne funkcjonalności, np. narzędzia online takie jak zarządzanie listą zadań.
* Aplikacje mobilne - programy instalowane na telefonie posiadające zaawansowane funkcjonalności.

### Co to jest HTML

HTML (ang. *HyperText Markup Language*) jest to hipertekstowy język znaczników (ang. tag, *markup*). Dlaczego *hipertekstowy*? Bo można w nim używać linków, które kierują do innych plików HTML, czyli można "skakać" pomiędzy stronami :)

HTML jest podstawowym językiem do budowy strony internetowej. Zawiera treść strony i organizuje strukturę tej treści. Niektórzy mylnie twierdzą, że za pomocą języka HTML definiujemy wygląd strony, ale tak nie jest. HTML dotyczy struktury informacji na stronie.

Poprzez określanie struktury należy rozumieć, że HTML określa typy treści, którą zawiera, jej kolejność i zagnieżdżenia. Jeśli chodzi o typy treści to są to m.in.:

* teksty, paragrafy, znaki końca linii
* linki
* nagłówki
* grafiki, zdjęcia
* kontenery treści (grupujące inne elementy HTML).

### Tagi HTML i struktura

Głównym elementem kodu HTML jest TAG (ang. *znacznik*). Jest to specjalne słowo kluczowe umieszczone wewnątrz ostrych nawiasów (znak mniejszości i większości). Budowa tagu wygląda tak:

<tag\_name>content</tag\_name>

Nazwa zawarta pomiędzy ostrymi nawiasami określa typ tagu. W uproszczeniu, tagi to kod HTML, a teksty między nimi to treść. Tagi okalają te teksty, co pozwala określić strukturę dokumentu HTML - co jest czym.

###### **Jakie są cechy tagów?**

* Tagi HTML najczęściej występują w parach np. <p> i </p> (tagi paragrafu).
* Pierwszy tag w parze to tag otwierający, drugi jest tagiem zamykającym.
* Tag zamykający jest zapisany podobnie jak otwierający, ale dodaje się znak / przed jego nazwą (slash, a nie backslash).
* Są też tagi niewymagające zamykającego taga (a nawet nie należy go wstawiać), takie jak <br> (złamanie linii, pot. enter).

###### **Struktura dokumentu HTML i przykładowe tagi**

Każdy dokument HTML ma ściśle określoną strukturę podstawową. Struktura, czyli kod początkowy, który w zasadzie powtarza się w każdym pliku HTML. Dopiero w ramach tej struktury dodajemy swoją treść i inne tagi. Struktura wygląda tak (to co jest w <body> jest opcjonalne):

<!DOCTYPE html>   
<html>   
 <head>   
 <meta charset="UTF-8"> <title>My first website</title>   
 </head>   
<body>   
<h1>My first heading</h1>   
 <p>My first paragraph and <a href="https://www.google.pl/">link</a>.</p> <img src="image.jpg" alt="sand" title="my holiday">   
</body>   
</html>

Poszczególne elementy (tagi) struktury to:

* Deklaracja DOCTYPE, która określa typ tego dokumentu jako HTML.
* Treść pomiędzy <html> i </html> to nasz dokument HTML.
* Treść pomiędzy <head> i </head> dostarcza informacji o stronie, które są ważne dla przeglądarki, ale najczęściej niewidoczne dla internauty.
* Treść w tagach <meta> (zawsze zagnieżdżone w <head></head>) zawiera konkretne informacje ważne dla przeglądarki. Aby poprawnie wyświetlały się polskie znaki za pomocą tagu meta dodajemy kodowanie UTF-8 - <meta charset="UTF-8">.
* Treść pomiędzy <title> i </title> to tytuł strony. Widać go na samej górze okna przeglądarki, na zakładce (tab).
* Treść w tagach<link> to zazwyczaj podpięcie arkusza stylów - omówimy to dokładniej w kolejnym submodule.
* Treść pomiędzy <script> i </script> to dołączenie skryptów do strony - póki co nie będziemy się tym zajmować.
* Treść pomiędzy <body> i </body> jest tym, co widzisz potem w przeglądarce. Tu wstawiasz teksty i zdjęcia dla swojej witryny WWW.... i opcjonalne elementy niżej:
* Treść pomiędzy <h1> i </h1> to nagłówek.
* Treść pomiędzy <p> i </p> utworzy paragraf tekstu.
* Treść pomiędzy <a href="#"> i </a> definiuje link, który ma prowadzić pod dany w atrybucie href adres.
* Znacznik <img src="file.jpg" alt=""> wstawia plik grafiki na stronę z danego źródła src (ang. *source*).

#### **Zapamiętaj!**

W sekcji head utrzymuj porządek - najpierw umieszczamy tag <meta>, następnie <title>, <link> i ostatnie <script>.

###### 

###### **Atrybuty tagów**

Tagom (znacznikom) HTML możemy dodawać atrybuty, które określą dodatkowe właściwości tagu. Dodajemy je w następujący sposób:

<tag\_name attribute1="valueX" attribute2="valueY">content</tag\_name>

Atrybuty zawsze należy dodawać do tagu otwierającego. Można też dodać kilka atrybutów do jednego tagu, rozdzielając je spacją. Wartość atrybutu podaje się po znaku równości, zawierając ją w cudzysłowie.

Dla różnych tagów da się stosować różne atrybuty, które mogą przyjmować określoną wartość. Np. aby dodać obrazek do dokumentu HTML, należy dodać taki kod:

<img src="image.jpg" alt="sand" title="my holiday">

Po znaku < mamy nazwę taga, czyli img (ang. *image*), a następnie atrybut src, który wskazuje nazwę pliku grafiki w cudzysłowie. Tuż po src znajdują się atrybuty title oraz alt. title powoduje, że po najechaniu kursorem na stronie na tę grafikę, pokaże się napis zawierający wartość tego atrybutu. Natomiast alt pozwala nam określić tekst zastępczy dla obrazka, który ma się pojawić, gdy ten nie będzie mógł być wyświetlony - powinien on zwięźle opisywać treść grafiki.

### **Zagnieżdżanie tagów**

Istotnym aspektem struktury dokumentu HTML jest to, że możemy zagnieżdżać tagi jeden w drugim:

<body> <h1>My first heading</h1> <p>My first <a href="http://google.pl">paragraph</a>.</p> <img src="image.jpg" alt="sand" title="my holiday"> </body>

Zauważ, że tag paragrafu <p> jest zagnieżdżony w elemencie <body>, a w nim dodatkowo jest zagnieżdżony link <a href="...">. Tworzy to tzw. strukturę drzewiastą. Tagi zagnieżdżone o poziom niżej warto w kodzie wcinać w prawo (klawiszem TAB), aby struktura dokumentu była bardziej czytelna. Wtedy szybciej widzimy co jest w czym zagnieżdżone. Pamiętaj, że wszystkie dzieci w danym tagu powinny mieć to samo wcięcie.

Stosuje się też nazewnictwo co do relacji tagów wg zagnieżdżenia. Na bazie przykładu z kodu powyżej wygląda to tak:

* child (dziecko) - <p> jest dzieckiem <body>,
* parent (rodzic) - <body> jest rodzicem <p>,
* sibling (rodzeństwo) - <h1> jest rodzeństwem <img> i <p>.

W polskich rozmowach często programiści łamią sobie język:

*"Skasuj ten link (<a>)! Który? No tego childa (czyt. czajlda) paragrafu. Nie widzę… No ten zagnieżdżony 2 poziomy w dół, w body i potem w p!".*

Nie brzmi to pięknie, ale często tak to wygląda w praktyce. Choć przykład jest dość prosty jak na tak długą dyskusję :)

Pamiętaj także, aby pilnować poprawnego zagnieżdżenia. Poniższy przykład jest nieprawidłowy:

<div> <p>Hello</p> <p>coders</p> </div>

Oba Tagi <p> powinny mieć takie samo wcięcie względem <div>. Prawidłowe zagnieżdżenie w tym przypadku będzie wyglądało następująco:

<div> <p>Hello</p> <p>coders</p> </div>

#### **Ważne**

Prawidłowe wcięcia są niesamowicie istotne. Gdy już będziesz mieć sporo linii kodu i trafi ci się jakiś błąd, to brak prawidłowego zagnieżdżenia sprawi, że będziesz mieć spore problemy ze znalezieniem go. Dotyczy to zarówno Ciebie, jak i osób, które będą Ci pomagać.

### Podstawy CSS

### **Podłączenie CSS do HTML i przeciążanie**

Pliki ze stylami CSS podpinamy w nagłówku strony (między tagami <head> a </head>) za pomocą tagu <link>. Jest to tag, który nie wymaga zamykania.

Przykładowe podpięcie do HTML pliku ze stylami wygląda następująco:

<link rel="stylesheet" type="text/css" href="style.css">

W powyższym przykładzie za pomocą atrybutu href wskazujemy plik, w którym znajdują się utworzone przez nas reguły. Metoda ta zadziała prawidłowo dla pliku *style.css*znajdującego się w tym samym katalogu co plik HTML, do którego podpinamy style. Jeżeli plik ze stylami jest głębiej, np. w katalogu o nazwie *stylesheets*, wtedy atrybut hrefwyglądałby następująco: href="stylesheets/style.css" (ścieżka względna, bez pełnego adresu pliku typu C:\file\...).

Style są czytane (i stosowane) przez przeglądarkę "z góry na dół". Oznacza to, że jeżeli na początku pliku stworzymy regułę nadającą akapitom kolor czerwony, a pod koniec pliku stworzymy podobną regułę CSS nadającą im kolor pomarańczowy, to w ostateczności nasze akapity będą wyświetlane na pomarańczowo (ostatni styl nadpisze poprzednie). Dzieje się tak, ponieważ styl z pomarańczowym kolorem tekstu znajduje się niżej, w związku z czym przeglądarka uznaje go za bardziej aktualny.

Analogicznie jest w przypadku, gdy podpinamy dwa pliki ze stylami.

<link rel="stylesheet" type="text/css" href="first-file.css"> <link rel="stylesheet" type="text/css" href="second-file.css">

### **Konstrukcja stylu**

Przykładowa reguła (styl) CSS wygląda następująco:

p { color: red; font-size: 24px; }

Interpretujemy ją w ten sposób:

selector { property: value; property: value; }

Opis elementów reguły:

* selector - określa element HTML, którego dotyczy styl,
* { ... } - klamry, oznaczają blok kodu, wewnątrz nich trzymamy właściwości oraz ich wartości,
* property - właściwość, to coś jak cecha wyglądu, np.: kolor, wysokość, szerokość itp.,
* : - dwukropek, oddziela właściwość od wartości jaką jej przypisujemy,
* value - wartość, określa jaka ta cecha ma być, np. dla koloru określa, że ma być zielony, a dla wysokości, że ma ona wynosić np. 300px itp.,
* ; - średnik, stawiamy zawsze na końcu linii kodu, jest to uniwersalna praktyka funkcjonująca w bardzo wielu językach programowania.

### **Frequently Used Styles :)**

Jest pewna popularna grupa właściwości CSS, których używa się bardzo często. Abyś nie musiał błądzić po setkach właściwości, poniżej wymieniamy najpopularniejsze:

* color - określa kolor tekstu w elemencie np. color: black,
* background - za jego pomocą możemy określić właściwość tła np. kolor background:red,
* border - określa parametry ramki elementu np. border: 2px solid #ccc, co da nam ramkę o grubości 2px, ciągłą (solid), koloru szarego,
* display - wyznacza sposób wyświetlania elementu np. display: none,
* height - odpowiada za wysokość podawaną najczęściej w pikselach (px) lub procentach (%) np. height: 200px,
* width - determinuje szerokość np. width: 100px,
* margin - odpowiada za marginesy elementów, np. margin: 20px sprawi, że element będzie miał dwudziestopikselowy margines,
* padding - odpowiada za przestrzeń między zawartością elementu a jego ramką, np. padding: 50px,
* font-family - określa czcionkę, która zostanie użyta do wyświetlania tekstu wewnątrz elementu np. font-family: 'Open sans',
* text-align - odpowiada za rozmieszczenie tekstu, za pomocą tej właściwości można wyśrodkować tekst lub wyrównać go do jednej z krawędzi (bądź obu), np. styl {text-align: center} wyśrodkuje tekst znajdujący się w elemencie HTML (kontener).

Listę wszystkich właściwości oraz ich wartości znajdziesz na stronie [W3schools](http://www.w3schools.com/css/default.asp). To bezcenne źródło informacji.

### **Kolory w CSS**

Jest kilka sposobów na zdefiniowanie koloru (np. kolor czcionki lub tła). Stosowane są w zależności od potrzeb i upodobań programistów.

###### **Przykłady najpopularniejszych form zapisu kolorów:**

* nazwy koloru w języku angielskim - blue, green, black, white ([link do pełnej listy](http://www.w3schools.com/cssref/css_colors.asp));
* heksadecymalny (hex) - czyli kod koloru zapisany w systemie szesnastkowym, przykładowe kolory to: #000000 (czarny), #00FF00 (kolor limonki), #FFFFFF (biały). Można także zapisać kolor w ten sposób #FFF (biały) gdzie każdy znak finalnie się "podwaja". Nie musisz znać sposobu jak "obliczyć" dany kolor, wystarczy Ci tzw. *color picker* do generowania oczekiwanego koloru (przykładowy na [Kodilla](http://kodilla.com/pl/tools), ale możesz też wygooglować w sieci);
* RGB - jest to wypadkowa 3 kolorów (red, green, blue). Zapisuje się go w ten sposób: np. rgb(255, 0, 0) daje kolor czerwony, a rgb(255, 255, 255) - kolor biały. Każda z wartości po przecinku to liczba z zakresu 0-255. Podobnie jak dla hex, programiści najczęściej generują kolory w RGB za pomocą *color pickerów*;
* RGBa - działa analogicznie jak RGB, ale ma jeszcze dodatkowy (czwarty) parametr odpowiadający za przeźroczystość, np. rgba(255, 0, 0, 1) wyświetli kolor czerwony, a rgba(255, 255, 255, 0.5) da nam kolor biały, ale półprzezroczysty. Ostatnia wartość może przyjmować wartości od 0 (całkiem przeźroczysty) do 1 (zupełnie nieprzezroczysty). Wartości podajemy jako ułamek z kropką (nie przecinkiem).

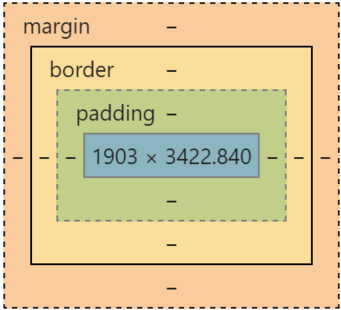
Najczęściej używa się kolorów zapisanych heksadecymalnie, głównie ze względu na prosty zapis. Minusem tego zapisu jest to, że nie można ustawić w nim przeźroczystości. Dlatego, jeśli jest nam ona potrzebna, sięgamy po zapis RGBa.

### **Box-model w CSS**

Box-model to założenie, że każdy umieszczony na stronie element HTML jest swego rodzaju pudełkiem (kontenerem), a każde takie pudełko posiada kilka cech wyglądu, które go charakteryzują.

Wyobraź sobie pudełka umieszczane wewnątrz siebie jak matrioszki :) Najmniejszym z nich jest treść (content), którą chcesz umieścić. Drugie pudełko to padding, czyli odległość treści od ramki (border) będącej kolejnym pudełkiem. Dla ramki też określamy jej grubość. Największym z pudełek jest margines, czyli margin - wyznacza on w jakiej odległości od brzegów strony lub innych elementów powinien znajdować się ten element, którym się aktualnie zajmujemy.

Zrozumienie tego ułatwi Ci poniższa grafika pochodząca z narzędzia deweloperskiego *Chrome Developer Tools* będącego częścią przeglądarki Chrome. Niebieskie pole to pudełko z treścią, które ma pewną przestrzeń (padding) od ramki , która też ma pewną grubość (border). A za pomocą margin określamy odległość innych elementów HTML od bordera naszego pudełka z treścią.



### **Składowe pudełek**

Nasze "pudełka", w przeciwieństwie do tych prawdziwych, nie muszą mieć wszystkich ścian. Możliwe jest na przykład ustawienie tylko jednej części ramki dla elementu HTML. W tym celu do nazwy właściwości CSS (padding, border, margin) dodajemy przyrostek definiujący kierunek:

* -top, jeśli chcemy ustawić właściwość dla góry elementu HTML;
* -right, gdy zależy nam na ustawieniach dotyczących prawej strony;
* -bottom dla właściwości dotyczących dołu;
* -left, gdy określamy właściwości po lewej.

Zatem chcąc ustawić ramkę na górze elementu, skorzystamy z właściwości border-top.

### **Wartości "po zegarze"**

Wiesz już, jak zrobić ramkę tylko na górze elementu. Co w przeciwnym przypadku - gdy Twoim zamiarem jest zrobienie ramki na każdym boku pudełka oprócz góry? Czy trzeba napisać 3 linijki kodu odpowiadające za właściwości border-right, border-bottom i border-left, zostawiając border-top domyślnie ustawione na 0?

Na szczęście NIE - w takim przypadku wykorzystamy znów właściwość border. Jeśli dla border podamy cztery wartości, np. border: 0 1px 2px 3px, ustawimy inną grubość ramki dla każdego brzegu elementu. Skąd jednak wiedzieć w jakiej kolejności wpisywać wartości? To proste - podajemy je zgodnie z ruchem wskazówek zegara, zaczynając od górnej - zatem w podanym przykładzie nie mielibyśmy górnej ramki (jej grubość jest ustawiona na 0px), prawa miałaby 1px, dolna 2px, a lewa 3px grubości.

#### **Dla dociekliwych**

Przeglądarki czytają arkusze stylów które im dostarczamy i je interpretują. To już wiemy. Ale jak sobie radzą z takimi zapisami jak ten?

border: 10px 5px 10px 5px;

Otóż przeglądarka tłumaczy sobie taki zapis na porozbijane, dokładniejsze zapisy jak w przykładzie poniżej:

border-top: 10px; border-right: 5px; border-bottom: 10px; border-left: 5px;

Spostrzegawcze osoby zauważą, że dla górnej i dolnej ramki wartości są takie same. Również dla bocznych mają one tą samą wartość. Może da się to jeszcze jakoś uprościć? Oczywiście, że się da :)

Ten zapis:

border: 10px 5px 10px 5px;

Możemy również przedstawić w ten sposób:

border: 10px 5px

A przeglądarka i tak go sobie przetłumaczy tak:

border-top: 10px; border-right: 5px; border-bottom: 10px; border-left: 5px;

Jest wiele właściwości, które zachowują się w ten sposób np. margin czy padding.

### **Dobre praktyki z CSS - walidacja kodu, debugowanie**

Kiedy skończymy pisać reguły w CSS, warto sprawdzić ich poprawność. W tym celu odwiedzamy stronę [css-validator.org](http://www.css-validator.org/), na której mamy możliwość sprawdzenia naszego kodu. W walidatorze możemy podać adres naszej witryny, wrzucić gotowy plik albo wkleić nasz kod.

Dobrym zwyczajem jest także debugowanie kodu. W tym celu musimy otworzyć narzędzie dla developerów, klikając na interesującym nas elemencie prawym przyciskiem myszy i wybierając opcję "Zbadaj element". Po lewej stronie Inspektora widzimy nasz kod HTML, prawa strona to reguły CSS. Za pomocą tego narzędzia możemy zobaczyć jakie reguły działają na dany element.

### **Projektowanie**

Projekt strony WWW może powstawać m.in. na 2 poniższe sposoby:

* Dostajesz projekt graficzny od grafika i na jego podstawie tworzysz szablon html/css.
* Tworzysz stronę dla siebie bez projektowania jej w Photoshopie, najpierw tworzysz jej ogólny zarys (prototyp) i na tej podstawie samą stronę.

My w tej sekcji zajmiemy się na razie drugim podejściem.

Przed każdym projektem należy się zastanowić, jakie elementy chcesz mieć na swojej stronie oraz jakie treści chcesz na niej umieścić. W każdym przypadku (prawie zawsze), przy tworzeniu strony używamy nagłówka i stopki. To możemy uznać za standard.

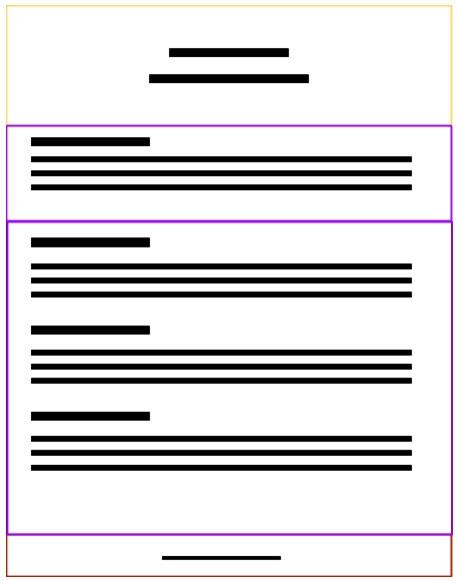
*Zastanówmy się nad treścią.* Dobrze byłoby się przedstawić i napisać kilka zdań na swój temat. To będzie nasza pierwsza sekcja.

W drugiej sekcji umieścisz pozostałe informacje pod kolejnymi nagłówkami, wypiszesz więcej ciekawostek na swój temat. Pochwalisz się swoim hobby, swoją wymarzoną pracą, ulubioną muzyką czy filmami.

Oczywiście możesz umieścić inne informacje niż te, które sugerujemy, najważniejsze by zachować założoną strukturę HTML :)

Dobrą praktyką jest rozrysowanie sobie projektu. Wystarczy kartka papieru i długopis. Dzięki takiej wizualizacji łatwiej jest oszacować to, jakich elementów potrzebujemy do osiągnięcia celu oraz jak rozlokować przestrzeń strony. Wielu webdeveloperów robi tak przed rozpoczęciem pracy z kodem.

Poniżej zobaczysz przykładowy szkic tego, co planujemy zbudować. Żółtym kolorem jest zaznaczony nagłówek, fioletowym podział na sekcje, bordowego użyliśmy dla stopki, a dla tekstów zastosowaliśmy grubsze czarne kreski.



Czasami spotkasz się z określeniem takiego rysunku jako: makiety lub prototypu (ang. *mockup, wireframe*). Jest to nic innego, jak wstępna wizualizacja układu strony. Jak już mówiliśmy, do rysowania prototypu można użyć kartki lub specjalnego programu online do tego celu, np. [Moqups](https://moqups.com/). Posiada on zestaw wielu gotowych elementów, za pomocą których możesz możesz szybko stworzyć prototyp swojej strony.

Prototyp ma tę zaletę, że zanim zakodujesz stronę, możesz kilka razy zastanowić się czy czegoś nie zmienić. Pozwala to oszczędzić czas na dokonywanie takich modyfikacji na zakodowanym szablonie HTML/CSS.

### **Struktura HTML**

Rozpoczniemy od stworzenia podstawowej struktury dokumentu zgodnej ze standardami HTML5 oraz podpięcia pliku ze stylami. Wspominaliśmy już o niej.

Następnie użyjmy tagów wprowadzonych w HTML5, takich jak:

* header dla nagłówka,
* section dla sekcji zawierających tekst (aby je odróżnić, dodaj im dwie różne klasy),
* footer, by stworzyć stopkę na dole strony. Pamiętaj o stworzeniu dwóch sekcji.

Odwzorowanie w kodzie HTML struktury rozrysowanej na makiecie powinno więc wyglądać tak:

<!DOCTYPE html> <html> <head> <meta charset="UTF-8"> <title>My website title</title> <link rel="stylesheet" type="text/css" href="style.css"> </head> <body> <header></header> <main class="container"> <section class="about"></section> <section class="interests"></section> </main> <footer></footer> </body> </html>

Równocześnie stworzymy w pliku CSS selektor wybierający body i nadamy mu margines wynoszący 0 (dla wartości zero nie trzeba przypisywać jednostki, np. px). Sprawi to, że nie będzie domyślnego marginesu jaki przyjmuje body.

Wygląda to tak:

body { margin: 0; }

#### Najprostszy layout

Co jeśli chcesz, aby całość była wyśrodkowana w kolumnie, jak na większości stron? Czyli nasze pudełka będą miały stałą szerokość, a marginesy po bokach będą także o równej szerokości, która będzie ustalana automatycznie zależnie od szerokości przeglądarki.

Wystarczy, że wszystkie nasze pudełka umieścimy w jednym głównym pudełku (kontenerze), czyli w tagu <main> o klasie container. Następnie w CSS dla tego kontenera (selektor dla klasy) należy dać styl margin: 0 auto, który dla całego kontenera zawartości po jego prawej i lewej nada automatyczny margines (dzięki wartości auto). Teraz ustalamy maksymalną szerokość (max-width) tego kontenera np. na 700px (bez tego automatyczne marginesy po bokach nie zadziałają bo element będzie miał cały czas 100% domyślnej szerokości).

Możesz też nadać wyśrodkowanemu kontenerowi w CSS biały kolor tła (#FFF), a całej stronie body kolor szary (#CCC). Wtedy wizualnie kontener będzie wyraźnie oddzielony.

Po stworzeniu szkieletu i prostego layout strony czas dokładnie zająć się każdym z elementów po kolei.

#### Nagłówek

Nagłówek (header) znajduje się w górnej części strony, zatem jest pierwszym jej elementem, jaki widzi odwiedzający. Użyjemy go do powitania naszych gości :)

W nagłówku strony umieść paragraf z powitaniem (tag <p>) oraz swoje imię, które zamkniesz w tagu <h1>.

Kod HTML nagłówka powinien więc wyglądać tak:

<header> <p>Hello, my name is</p> <h1>Tomek</h1> </header>

Przejdź do pliku CSS. Najpierw za pomocą selektora wybierającego nagłówek nadaj mu padding równy 30px. Korzystając np. z darmowego z serwisu [pexels](https://www.pexels.com/), znajdź zdjęcie, które umieścisz w tle nagłówka strony za pomocą właściwości background-image: url(link go grafiki).

Następnie stwórz selektor, za pomocą którego ostylujesz tylko nagłówek h1 będący dzieckiem elementu <header> i pogrub tekst w nim zawarty oraz spraw, by wszystkie litery napisu były wielkie (podpowiedź: przyda Ci się do tego właściwość text-transform).

Powiększ tekst nagłówka h1 do rozmiaru 48px. Kolejny selektor niech wybiera zarówno h1, jak i p. Ustaw taki kolor tekstu, aby wyróżniał się na tle wybranego przez Ciebie zdjęcia. Niech tekst będzie wyśrodkowany.

Wzorcowe style:

header { padding: 30px; background-image: url(https://static.pexels.com/photos/...); } header h1 { font-weight: bold; text-transform: uppercase; font-size: 48px; } header p, header h1 { text-align: center; color: white; }

Efekt w przeglądarce:



#### Kaskadowość stylów, dziedziczenie, selektory

Ostateczne formatowanie strony jest zależne od wielu czynników. Istnieje hierarchia reguł CSS, która decyduje jak finalnie będzie wyglądał formatowany element. Jeżeli dany element ma przypisaną więcej niż jedną regułę, np. dla koloru tekstu, to zastosowanie będzie mieć ta, która jest wyżej w hierarchii.

Od najważniejszych:

1. styl lokalny (inline),
2. rozciąganie stylu (span),
3. wydzielone bloki (div),
4. wewnętrzny arkusz stylów,
5. import stylów do wewnętrznego arkusza,
6. zewnętrzny arkusz stylów,
7. import stylów do zewnętrznego arkusza,
8. atrybuty prezentacyjne HTML - np. color="...", width="..." i inne).

Wyobraźmy sobie sytuację, że w pliku style.css umieścimy:

div { color: red; }

a nasz HTML będzie wyglądał:

<div style="color: green">content</div>

Według kaskadowości stylów nasz div będzie miał kolor zielony, gdyż styl inline ma wyższy priorytet ważności i pierwszeństwo w modyfikowaniu koloru czcionki w podanym divie.

Kolejnym czynnikiem wpływającym na końcowy styl elementów jest dziedziczenie. Każdy element html posiada swoich “rodziców" od których otrzymuje style i "dzieci", którym je narzuca. Trudne? Wyobraźmy sobie sytuację:

<body> <div> <p>Content</p> </div> </body>

Kiedy do elementu <body> dodamy właściwość color: green, to tekst diva, który jest w środku <body>, będzie zielony. Odziedziczy on kolor po swoim rodzicu.

Duży wpływ na finalne stylowanie strony ma stopień ważności selektorów.

Od najważniejszych:

* P1 - selektory identyfikatora: #name { color: #000; },
* P2 - selektory atrybutu, klasy i pseudoklasy .name { color: #000; },
* P3 - selektory pseudoelementu i elementu p { color: #000; },

#### Sekcje

Zajmijmy się teraz sekcjami. Umieścisz w nich kilka zdań o sobie - czym się zajmujesz, ile masz lat - wpiszesz to, na co masz ochotę.

W pierwszej sekcji utwórz nagłówek <h2> informujący, o czym będzie tekst znajdujący się poniżej. Stwórz paragraf i umieść w nim te informacje.

<h2>Who am I?</h2> <p>A few words about me.</p>

Druga sekcja to rozwinięcie informacji o Tobie - dodaj tam nieuporządkowaną listę (tag <ul>) z Twoimi hobby oraz napisz w paragrafie kilka słów o swoich umiejętnościach. Niech lista oraz paragraf będą poprzedzone nagłówkami h2.

<h2>I have different interests:</h2> <ul> <li>programming</li> <li>...</li> <li>and many others!</li> </ul> <h2>More about me:</h2> <p>I have a lot of skills!</p>

Teraz nadajemy style dla kontenera. Używając selektora .container, dodajemy mu szerokość (width) 700px oraz margin: 0 auto;.

Mamy już zawartość i kontener, teraz napisz selektor wybierający elementy <section>. Nadaj im padding wynoszący po 20% z prawej i lewej strony.

Warto, aby sekcje różniły się od siebie stylistycznie. Korzystając z nadanych obu sekcjom różnych klas, możesz na przykład użyć innego koloru nagłówków h2 dla pierwszej sekcji niż dla drugiej.

Pamiętaj, że aby Twoja strona dobrze wyglądała, warto dobrać do niej odpowiednie kolory. Na początku wystarczy, że użyjesz "bezpiecznych", pasujących do siebie w różnych konfiguracjach, kolorów przedstawionych na stronie [Flat UI Colors](https://flatuicolors.com/).

Możesz także wybrać także inny schemat kolorów (ang. color scheme) za pomocą [tego narzędzia](http://paletton.com/). W górnej części narzędzia możesz wybrać schemat 1-, 2-, 3-, 4-kolorowy. Takie bezpieczne zestawy pozwalają bez kompetencji graficznych w miarę dobrze dobrać kolory, aby przyjemnie się komponowały.

Gdy już wybierzesz kolory, jakich chcesz użyć na swojej stronie, utwórz kolejne dwa selektory. Najpierw dla nagłówków h2 będących potomkiem elementu o klasie nadanej pierwszej sekcji (tutaj: about) i ustaw w nim właściwości: margines górny 60px, rozmiar czcionki 28px i jeden z wybranych kolorów czcionki. Drugi selektor niech wybiera nagłówki h2 w drugiej sekcji. Wpisz w nim właściwości: margines 40px od góry i drugi z wybranych kolorów czcionki.

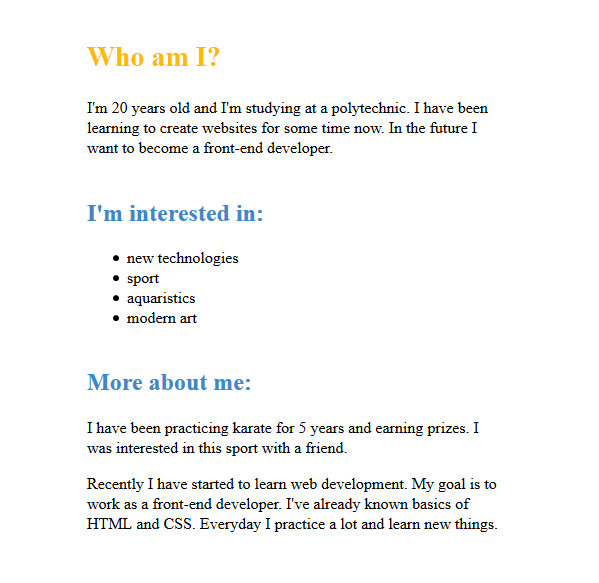
Kod obu sekcji powinien teraz wyglądać tak:

<section class="about"> <h2>Who am I?</h2> <p>I'm 20 years old and I'm studying at a polytechnic. I have been learning to create websites for some time now. In the future I want to become a front-end developer.</p> </section> <section class="interests"> <h2>I'm interested in:</h2> <ul> <li>new technologies</li> <li>sport</li> <li>aquaristics</li> <li>modern art</li> </ul> <h2>More about me:</h2> <p>I have been practicing karate for 5 years and earning prizes. I was interested in this sport with a friend.</p> <p>Recently I have started to learn web development. My goal is to work as a front-end developer. I've already known basics of HTML and CSS. Everyday I practice a lot and learn new things.</p> </section>

A dotyczące ich (oraz kontenera) style CSS:

.container { width: 700px; margin: 0 auto; } section { padding: 0 20%; } .about h2 { margin-top: 60px; font-size: 28px; color: #FFBA08; } .interests h2 { margin-top: 40px; color: #3F88C5; }

Twoje sekcje powinny teraz wyglądać podobnie, ale możesz wybrać inne kolory i wielkości w pikselach:



Stronie możesz nadać dowolny wygląd, tematykę. Możesz także dodać zdjęcia.

#### Stopka

W stopce zwykle umieszcza się dane kontaktowe, linki do profilów w mediach społecznościowych, informacje o stronie, a często także mapę jej zawartości w przypadku większych portali.

Umieść w niej paragraf ze swoim imieniem i nazwiskiem oraz tekstem "You will find me on:", umieszczając za nim linki do mediów społecznościowych wraz z ich nazwami (np. You will find me on: Facebook | Twitter - gdzie w formie napisów Facebook oraz Twitter będą ukryte linki do Twoich profilów na tych stronach). Jeśli w fazie tworzenia strony nie chcesz podawać linków, niech wartość atrybutu href tagu a wynosi "#".

<p>You will find me on: <a href="#">Facebook</a> | <a href="#">Twitter</a></p>

W pliku CSS stwórz selektor wybierający stopkę. Nadaj jej właściwości: górny margines 60px (by odsunąć ją od reszty strony), padding: 30px; oraz dowolny kolor tła.

Następnie napisz selektor, który będzie dotyczył paragrafu tekstu umieszczonego w stopce. Nadaj w nim właściwości odpowiadające za wycentrowanie tekstu w poziomie i ustawienie koloru tekstu.

Ostatni selektor niech dotyczy linków umieszczonych w tym paragrafie. Spraw, by zniknęły podkreślenia (wykorzystaj do tego właściwość text-decoration), a tekst był pogrubiony i miał wyróżniający się kolor.

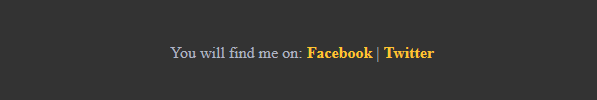
Struktura HTML stopki powinna wyglądać więc tak:

<footer> <p>You will find me on: <a href="#">Facebook</a> | <a href="#">Twitter</a></p> </footer>

A dotyczący jej CSS:

footer { margin-top: 60px; padding: 30px; background: #333333; } footer p { text-align: center; color: #B0B5C4; } footer a { text-decoration: none; color: #FFC532; font-weight: bold; }

Po zinterpretowaniu kodu przez przeglądarkę stopka powinna wyglądać tak, ale możesz skorygować to pod swój gust:



### **Lifting czcionki**

Bardzo ważnym elementem webdesignu (oprócz kolorów - używaj schematów kolorów!), o który dba każdy front-endowiec, jest czcionka.

Na swojej stronie możesz użyć dowolnej czcionki, wystarczy, że dodasz do niej link w części <head> w pliku HTML bądź regułę @import z jej nazwą w pliku CSS.

Strona [Google Fonts](https://www.google.com/fonts) oferuje całą kolekcję darmowych czcionek. Znajdziesz tam także treści, które należy umieścić w pliku HTML/CSS, by użyć wybranego kroju czcionki.

Spróbuj także wykorzystać inne właściwości CSS, o których nie wspomnieliśmy w kursie, a jakie uda Ci się znaleźć na [W3schools](http://www.w3schools.com/css/default.asp).

Dla przykładu zastosujemy dwie czcionki: Montserrat oraz Rubik. By umożliwić ich użycie na stronie, na górze pliku CSS umieśćmy:

@import url(https://fonts.googleapis.com/css?family=Montserrat:400,700); @import url(https://fonts.googleapis.com/css?family=Rubik:400,500);

Niech podstawową czcionką na stronie będzie Rubik, z kolei Montserrat zastosujmy do nagłówków h1 oraz h2. W tym celu w selektorze body dodaj właściwość: font-family:'Rubik', sans-serif;.

Następnie napisz selektor zbiorowy wybierający wszystkie nagłówki h1 i h2 na stronie. Umieść w nim właściwość: font-family: 'Montserrat', sans-serif;. Od razu lepiej!

W swoim projekcie wykorzystaj te same kroje czcionek lub znajdź inne.

### **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 jego 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>.

### **Dodawanie komentarzy**

Komentarze w kodzie zazwyczaj wykorzystywane są jedynie po to by oddzielić jedną sekcje 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 podpina je do widoków, na których są wykorzystywane. Jako, że kod sam w sobie powinien być dokumentacją dla siebie, raczej się ich unika. Jednak warto wiedzieć jak wyglądają :)

Komentarze w CSS dodajemy w ten sposób:

/\* my comment \*/

W HTML komentarz możemy dodać tak:

<!-- 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.

/\* === HEADER === \*/

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:

.btn-icon { width: 200px; height: 80px; }

Każdy selektor czy właściwość są dodawane od nowej linijki. 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 linijki, 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 wcięć, 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.](https://google.github.io/styleguide/htmlcssguide.html)

### **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 sprawią, 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](http://getbem.com/naming/). 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, opisania 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. <span>) albo blokowy (zajmuje 100% szerokości kontenera, w którym się znajduje, np. <p>. Odpowiada za to właściwość display, którą możemy definiować jak w przykładzie poniżej:

.example { display: block; }

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 linijki 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>,<fomr>, <footer>, <section>.
* inline - element liniowy. Nie zaczyna się od nowej linijki i zajmuje tylko tyle miejsca, ile jest niezbędne. Przykładowe elementy domyślnie wyświetlane jako inline to <span>, <a>,<img>.
* 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 <span> jest elementem liniowym. Możemy dowolnie zmieniać wyświetlanie interesujących nas elementów w pliku CSS.

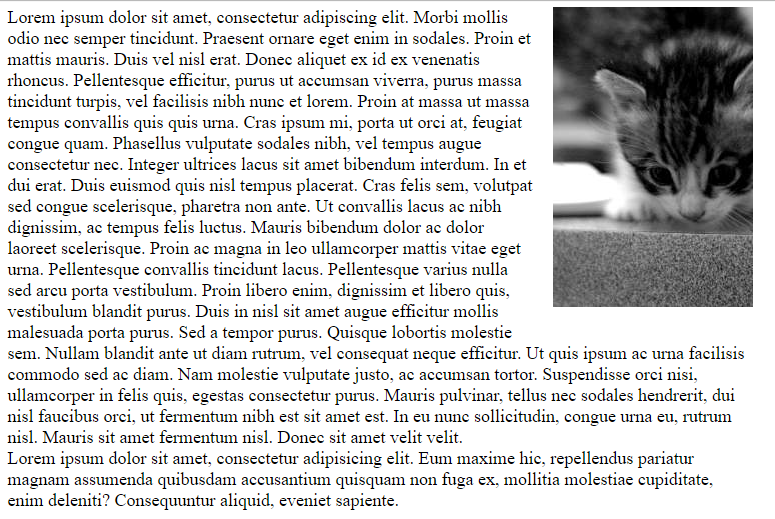
Zwróć uwagę na kod poniżej oraz sposób, w jaki poszczególne elementy pojawiają się na stronie. Nadaliśmy tam poszczególnym divom różne właściwości display, zmieniając w ten sposób ich wymiary oraz rozmieszczenie na stronie (spróbuj pobawić się tym kodem zmieniając wartości dla display).

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 <span>, który domyślnie wyświetlany jest jako inline, wówczas zmieni to też jego miejsce w [przepływie](https://www.w3.org/TR/CSS21/visuren.html#normal-flow) 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!)

<https://codepen.io/kodilla/pen/KzEpad>

### **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 <img> 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.

<https://codepen.io/kodilla/pen/jqJEjg>

### **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.

.example { position: relative; left: 20px; }

#### **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 zagnieżdżona 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;.

Pod poniższym linkiem zobacz jak działają wymienione właściwości, możesz je dowolnie modyfikować.

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:

header a { /\* Styles for the links inside header \*/ }

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

header .main-navigation a { /\* Styles for the links inside .main-navigation inside header \*/ }

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 >.

.example > div { /\* Properties of divs that are direct inside .example \*/ }

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 niżej 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:

h1, h2, h3 { text-align: center; font-weight: bold; }

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.

.example + p { /\* adjacent sibling selector \*/ }

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:

.example ~ p { /\* general sibling selector \*/ }

W ten sposób możemy przypisać właściwości CSS dla całej grupy akapitów będących rodzeństwem klasy example, np. znajdującym się w tym samym kontenerze na tym samym poziomie struktury HTML.

<https://codepen.io/kodilla/pen/RadWRw>

### **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:

a:hover { color: red; }

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.

a:visited { color: green; }

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

a:active { color: #0000FF; }

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.

header li:hover { background-color: #52B3D9; }

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:

p:first-child { /\* First paragraph of parent div \*/ } li:last-child { /\* Last item in the ul or ol list \*/ } p:nth-child(3) { /\* Paragraph, being the n-th child of the parent - in this example it's third \*/ }

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

<http://webkod.pl/kurs-css/lekcje/dzial-1/efekt-przejscia-transition-css3>

### **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.

### **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**](https://kodilla.com/pl/bootcamp-module/76/3/2704#2-4) **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:

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:

.grad {  
 background: red; /\* for browsers that do not support gradients \*/  
 background: -webkit-linear-gradient(red, yellow); /\* Safari 5.1 - 6.0 \*/  
 background: -o-linear-gradient(red, yellow); /\* Opera 11.1 - 12.0 \*/  
 background: -moz-linear-gradient(red, yellow); /\* Firefox 3.6 do 15 \*/  
 background: linear-gradient(red, yellow); /\* standard syntax \*/  
}

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:

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

Możemy też ustalać przezroczyste kolory:

background-image: linear-gradient(45deg, #f1c40f 50%, transparent 50%);

### **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.

div {  
 width: 100px;  
 height: 100px;  
 background: red;  
 transition: width 2s;  
}  
div:hover {  
 width: 300px;  
}

### **Jednostki Miar**

Najczęstsze jednostki miar:

- **px** - najpopularniejszy jednostka, określająca wielkość w pikselach, np. dla monitora komputera,

- **%** - procentowa wartość jest zależna od kontekstu, np. szerokości nadrzędnego kontenera (rodzica),

- **em** - jest mnożnikiem aktualnego rozmiaru elementu nadrzędnego,

- **rem** - bazuje na podstawowym rozmiarze czcionki strony (16px domyślnie lub takim jaki jest przypisane dla tagu HTML)

- **vw** - bazuje na szerokości okna (Viewport Width) 1vw = 1/100 szerokości okna,

- **vh** - bazuje na wysokości okna (Viewport Height) 1vh = 1/100 wysokości okna.

### **Fonty**

PS. Nazwy czcionek składające się z kilku słów podajemy w regule CSS w cudzysłowach (font-family: "Times New Roman";), a te składające się z jednego słowa bez cudzysłowów (font-family: Arial;).

<link href='adres czcionki na serwerze' rel='stylesheet' type='text/css'>  
@import url(link-do-czcionki);

#### **Rodzaje czcionek**

Każda czcionka ma przypisany pewien rodzaj określający jej charakter. Niektóre są bardziej formalne, inne mniej zobowiązujące.

Podstawowe rodzaje czcionek to:

- **szeryfowe** (serif) są czcionkami z dodatkowymi elementami ozdobnymi (np. kreseczki na dole i górze dużej litery 'i'), często używane w przypadku stron o charakterze humanistycznym (np. *Times*),

- **bezszeryfowe** (sans serif) są czcionkami bez dodatkowych elementów ozdobnych, najczęściej używane (np. *Arial*),

- **o stałej szerokości** (monospace) są czcionkami, w których każdy znak zajmuje taką samą przestrzeń, używana są do prezentacji kodu (np. *Courier*),

- **pisanki** (handwritting), czcionki imitujące pismo odręczne, spotykany np. na stronach o charakterze humorystycznym (np. *Comic Sans*).

Tenże rodzaj wstawia się tu font-family: Arial, ...; i mówi on, że jeśli na komputerze nie ma czcionki Arial lub innej wskazanej, to ma załadować inną czcionkę szeryfową z systemu komputera.

### **Łamanie lini**

Zdarzają się jednak sytuacje kiedy chcemy aby dany fragment tekstu nie był dzielony na dwa lub więcej wierszy. Aby uzyskać taki efekt używamy właściwości white-space (rozumiane jako "spacje") z wartością nowrap (ang. *nie zawijać*).

#### **Przestrzeń między literami** letter-spacing oraz word-spacing **Transformacje tekstu**

Wartości dla text-transform to:

- uppercase duże litery,

- lowercase małe litery,

- capitalize każda pierwsza litera wyrazu będzie duża, pozostałe będą takie jakie zostały wprowadzone.

Klasie hero zmień wysokość na równą całej **wysokości** okna. Nasz baner uzyskał 100% wysokości okna podglądu :) To modny zabieg na stronach głównych.

### **Dynamiczne wyliczanie wartości**

Funkcja calc() pozwala wykonywać podstawowe działania arytmetyczne. Wystarczy, że w nawiasach zapiszemy działanie, na przykład width: calc(100px - 75px); zwróci wartość 25px.

**Zwróć uwagę na spacje w konstrukcji funkcji calc().** Pomiędzy liczbami i znakiem działania matematycznego, muszą być zawsze spacje!

**Dane do obliczeń:**

Wysokość stopki wraz z marginesem: **82px**.

Wysokość banera wraz z marginesem: **120px**.

Suma wysokości powyższych elementów: **202px**.

## **GRID**

## **3.2. Budowa layoutu**

#### **Sugestia**

Przed rozpoczęciem tego modułu (lub po teorii) warto przejść ten kurs [Budowa layoutu](https://kodilla.com/pl/courses/budowa-layoutu), aby przećwiczyć w praktyce podstawowe zagadnienia, co pomoże przy wykonywaniu zadań.

Zanim zaczniemy przekuwać projekty graficzne na strony internetowe, musimy przyswoić sobie pewien zasób wiedzy na temat samego projektowania strony oraz dobrych praktyk z tym związanych.

Tworząc stronę od zera, front-endowcy wielokrotnie napotykają te same trudności dotyczące rozmieszczenia treści na stronie oraz ich wyglądu. Z racji tego, że nie są to nowe problemy, istnieją dla nich pewne sprawdzone rozwiązania, czyli tzw. wzorce (patterns). Takim wzorcem może być np. umieszczenie treści w trzech kolumnach, które na urządzeniach mobilnych wyświetlają się jedna pod drugą, czy też klikalne logo, które prowadzi do strony głównej.

Niektórzy przyjmują stanowisko, że na skutek stosowania wzorców wiele stron wygląda obecnie bardzo podobnie, przez co jest coraz mniej miejsca na nowatorskie rozwiązania i eksperymenty. Można powiedzieć, że jest to pewnego rodzaju spektrum, na którego jednym końcu znajdują się sprawdzone i skuteczne, lecz powtarzalne, wzorce, a na drugim znajdziemy układy mniej oczywiste i eksperymentalne. Naszym zadaniem będzie wyważenie obu tych podejść.

### **Przykłady układów**

Poniżej przedstawiamy kilka różnych layoutów stron. Być może część z nich jest Ci już znana. Skupimy się teraz na sposobie, w jaki są one zorganizowane graficznie, by odnaleźć powtarzające się elementy:

* [Time Magazine - trzy kolumny](http://time.com/)
* Strona magazynu Time, pomimo pewnego natłoku treści, ma wyraźny podział na trzy kolumny wraz z menu znajdującym się na górze. Nawiązuje w ten sposób do sposobu, w jaki składane są tradycyjne gazety.
* [Dogma - minimalizm dwóch kolumn](http://dogma.eu/)
* Na przeciwległym biegunie znajduje się minimalistyczna strona agencji Dogma przedstawiająca jej portfolio. Poszczególne projekty tworzą galerię w dwóch kolumnach, a każda część galerii prowadzi do strony zawierającej opis projektu.
* [Nice Portfolio - Stały sidebar](http://niceportfol.io/)
* Stały sidebar umieszczony po lewej stronie dokumentu zawiera menu, zaś cała zawartość umieszczona jest po prawej. Warto również przejrzeć strony, które się tam znajdują!
* [Avocode 2015 - one/single page layout](https://2015.avocode.com/)
* Strony z informacjami pojawiającymi się w miarę ich przewijania stały się hitem i można uznać je za standard wśród nowoczesnych layoutów. W takich stronach linki zazwyczaj nie przekierowują nas na nowe podstrony, tylko na sekcje w obrębie tej samej strony. Często też w miarę przewijania wyświetlane są użytkownikowi animacje. W powyższym przykładzie technika ta znalazła zastosowanie do stworzenia animowanej infografiki przedstawiającej trendy w projektowaniu graficznym na potrzeby Internetu.

### **Dobre praktyki komponowania layoutu**

Co do zasady, **front-endowiec nie jest projektantem**, lecz pewna fundamentalna wiedza z zakresu projektowania graficznego jest przydatna, jeśli nie niezbędna. Dzięki niej możemy lepiej porozumieć się z grafikiem, a także samemu rozpoznać dobre (lub niekoniecznie dobre) rozwiązania i wykorzystać tę wiedzę do sprawniejszego budowania strony. Poniżej znajduje się lista dobrych praktyk związanych z tworzeniem layoutu strony od podstaw.

#### **KISS - Keep it simple, stupid!**

Jedną z najważniejszych zasad, które powinniśmy sobie przyswoić, jest dążenie do prostoty. Skomplikowany interfejs jest trudny w użytkowaniu i zniechęca użytkowników (i klientów!) do korzystania z naszej strony. Ponadto prostszy HTML i CSS powoduje, że strona wczytuje się w przeglądarce szybciej, nie skazując użytkowników na czekanie.

#### **Korzystaj ze znanych wzorców.**

Omówiliśmy już zalety korzystania ze sprawdzonych wzorców. To, że znajdują się one na wielu stronach, wynika z tego, że przetrwały one próbę czasu. Możemy założyć, że jeżeli coś sprawdza się na stronie z milionami użytkowników, sprawdzi się też na mniejszą skalę.

#### **Ustal wizualną hierarchię.**

Istotną sprawą jest to, aby uwaga użytkownika była skupiona na tym, co jest najważniejsze. W kontekście layoutu ma to znaczenie zwłaszcza przez ustalenie kolejności i rozmiarów poszczególnych treści, ich rozmiarów, kolorów, odstępów pomiędzy elementami.

#### **Nie każ użytkownikom myśleć.**

Jest to założenie podobne do pierwszego, jednak nie do końca. Twój layout powinien być zrozumiały dla użytkownika, czyli funkcja poszczególnych jego elementów powinna być jasna od momentu ich ujrzenia. Ponadto użytkownik powinien być w stanie znaleźć bez problemu potrzebne mu funkcje i treści.

#### **Zadbaj o czytelność tekstu.**

Twój tekst powinien mieć dostatecznie duży rozmiar zapewniający czytelność na każdym ekranie. Ponadto interlinie oraz odstępy między literami powinny być dostatecznie duże, by tekst oraz litery nie najeżdżały na siebie.

Spraw by Twój tekst oddychał! Nie żałuj odstępów i marginesów, by zapewnić każdemu elementowi odpowiednią przestrzeń. Pamiętaj też o odpowiednim kontraście pomiędzy tłem a tekstem. Projektując stronę, trzeba pamiętać o starszych użytkownikach ze słabszym wzrokiem. Słabszy kontrast ogranicza czytelność tekstu przy wyświetlaniu na mniejszych ekranach. Także wtedy gdy na wyświetlacz świeci słońce lub gdy tekst wyświetlany jest na starszym monitorze.

Konsorcjum W3C ustanowiło pewne [minimalne standardy](https://www.w3.org/TR/UNDERSTANDING-WCAG20/visual-audio-contrast-contrast.html) w tej kwestii: minimalny kontrast pomiędzy tekstem a tłem powinien wynosić 4,5 : 1.

Możemy sprawdzić kontrast np. za pomocą [tego narzędzia.](http://webaim.org/resources/contrastchecker/)

### **Właściwości rozmieszczenia**

Layout strony możemy kształtować, nadając elementom HTML naszej strony odpowiednie właściwości w pliku CSS, które ustalą pozycję naszej treści na stronie. Do takich stylów, które determinują układ strony, należą m.in. następujące właściwości CSS:

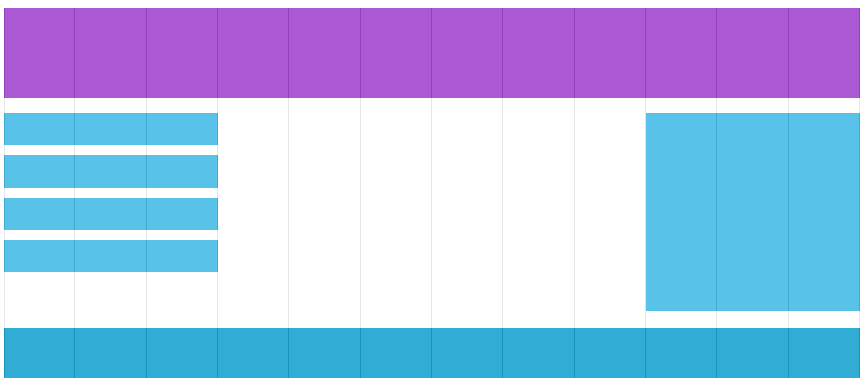
* **Display** - sposób wyświetlenia elementu HTML.
* **Position** - pozycja elementu HTML.
* **Float** - opływanie elementu HTML przez inne elementy.

Poprzez przypisanie odpowiednich wartości tym właściwościom będziemy mogli precyzyjnie określić, w którym miejscu ma znaleźć się dany element oraz w jakiej relacji ma on pozostawać względem pozostałych.

Ze względu na mnogość urządzeń, na których obecnie wyświetlane są strony internetowe, zapewnienie odpowiedniego wyświetlania nabiera kolosalnego znaczenia. Nowoczesnym sposobem na rozwiązanie tego problemu jest tzw. Flexbox, z którym zapoznasz się w kolejnym module.

### **Czym jest grid**

Znamy już zasady tworzenia layoutu, teraz dowiemy się, jak zbudować za pomocą poznanych wcześniej właściwości przydatne narzędzie w jego budowie. Mowa o tzw. **gridzie**, czyli siatce wierszy i kolumn, wewnątrz której będziemy mogli rozmieszczać elementy na stronie. Jest to sprawdzone rozwiązanie mające wielu zwolenników. Wiele stron jest zbudowanych w oparciu o siatkę 12 **kolumn**, wewnątrz których możemy umiejscowić nasze elementy, tak jak na ilustracji poniżej:



Czemu akurat 12? Taka liczba kolumn daje dużą swobodę w rozmieszczaniu: możemy ustalić, że element po lewej będzie zajmować 1/3 widoku, a pozostała część 2/3, definiując szerokość jednej jako 8 kolumn, a drugiej jako 4. Dwunastokolumnowy layout dzieli się na różne sposoby: 2, 3, 4 itp.

Jak się to stosuje? Stworzymy klasy col-X, gdzie X to będzie szerokość, jaką chcemy nadać interesującemu nas elementowi. Na przykład, nadając elementowi klasę col-3, sprawimy, że będzie on zajmował szerokość trzech kolumn, tj. 3/12, czyli 25% szerokości (tak, jak niebieskie elementy w przykładzie powyżej), .col-4 - czterech itd.

### **Proces budowy własnego gridu**

Poniżej omówimy dokładnie sposób tworzenia własnego gridu. Nie musisz wykonywać go teraz. Na końcu submodułu będą zadania, w których to zrobisz.

Zacznijmy od ustalenia wartości właściwości box-sizing na border-box dla wszystkich elementów. W ten sposób do wymiarów poszczególnych elementów doliczane będą ich dopełnienia (padding) oraz obramowanie (border). Robimy to za pomocą uniwersalnego selektora:

\* {  
 box-sizing: border-box;  
}

Kolejnym krokiem będzie zdefiniowanie wiersza (row).

Nasza treść będzie w ten sposób zorganizowana w poziome wiersze, które następnie będą zawierać kolumny. Dzięki temu możemy umieścić kilka elementów w kolumnach jeden pod drugim i upewnimy się, że nie będą ze sobą kolidować.

Jako że kolumny będą opierały się na floatach, musimy mieć pewność, że nie będą uciekały ze swoich wierszy. Wykorzystamy w tym celu tzw. clearfix hack. Jest to pewnego rodzaju trik, który zapewni prawidłowe wyświetlanie zawartości. Należy do tego stworzyć klasę row wraz z pseudoelementami :before oraz :after, a następnie dodać tam następujący kod:

.row:before,  
.row:after {  
 content:"";  
 display: table ;  
 clear: both;  
 }

Rozwiązanie takie uchodzi za niezbyt eleganckie z tego względu, że jest to trik wykorzystujący właściwości wyświetlania tabeli. Jednak osiągamy w ten sposób nasz cel i clearfix przyjął się jako pewnego rodzaju obejście problemu.

W odpowiedzi na potrzebę prawidłowego wyświetlania elementów na stronach powstał flexbox (o którym więcej napiszemy w dalszej części kursu), jednak z racji tego, że nie jest on wspierany przez stare przeglądarki wciąż mające duży udział w rynku, warto poznać i takie rozwiązania, jak clearfix. Zasady jego działania są dość skomplikowane, jednak nie musimy się na razie w nie zagłębiać. Istotne jest, że robi to, czego od niego oczekujemy.

Istnieje także możliwość zagnieżdżania w sobie kolumn/wierszy. Wówczas szerokość elementu będzie procentową wartością szerokości elementu rodzica, a nie całej strony. Pamiętaj, że jeśli chcemy zagnieździć kilka kolumn w jednej, nigdy nie robimy tego bezpośrednio, tylko w kolumnie umieszczamy wiersz, w którym tworzymy kolumny. Struktura HTML wygląda więc tak:

<div class="col-8">  
 <div class="row">  
 <div class="col-4">A column with a width of 1/3 of the parent column</div>  
 <div class="col-4">A column with a width of 1/3 of the parent column</div>  
 <div class="col-2">A column with a width of 1/6 of the parent column</div>  
 <div class="col-2">A column with a width of 1/6 of the parent column</div>  
 </div>  
</div>

Tą samą zasadą kierujemy się, zagnieżdżając w sobie wiersze - umieszczamy wiersz, w nim kolumnę, a dopiero wtedy kolejne wiersze.

### **Budowa kolumn**

Przejdźmy do stworzenia stylów odpowiedzialnych za poszczególne kolumny naszego layoutu. Jest to najbardziej skomplikowane zagadnienie ze względu na to, że istnieje co najmniej kilka metod ich pozycjonowania, a ponadto trzeba wziąć pod uwagę wyświetlanie na różnych szerokościach ekranu. Teraz skupimy się na podstawowym gridzie, zaś aspekt responsywności poznamy lepiej w module temu poświęconym.

Chcemy, aby wszystkie elementy znajdowały się w jednym rzędzie, niezależnie od tego, ile zajmują miejsca. Dlatego też dodamy do wszystkich klas .col właściwość float: left, oraz minimalną wysokość. Aby przypisać to do wszystkich klas skorzystamy tutaj z selektora atrybutu.

Pozwala on nam wybrać wszystkie elementy HTML, które posiadają dany atrybut lub jego określoną wartość. Przykładowo, stosując a[target], wybieramy wszystkie elementy a posiadające atrybut [target] o dowolnej wartości.

Jeżeli dodamy tam sformułowanie a[target="\_blank"], wybierzemy tylko te linki, które otwierają się w nowej karcie (mają atrybut target="\_blank"). Dodając gwiazdkę \* przed znakiem równości, wybierzemy wszystkie te elementy, które mają dany atrybut, a jego wartość zawiera w sobie określony ciąg znaków. W przykładzie poniżej wybieramy wszystkie elementy, które w nazwie klasy mają col-:

[class\*='col-'] {  
 float: left;  
 Min-height: 1px;  
}

Następnie przypiszemy do poszczególnych wymiarów kolumn odpowiednie szerokości. Jeżeli mamy 12 kolumn, zaś nasz kontener zajmuje 100% szerokości, to pojedyncza kolumna będzie mieć 100 / 12 = 8.33% szerokości. Natomiast jeżeli chcemy, aby element rozciągał się np. na dwie kolumny, mnożymy 8,33% \* 2 = 16,66% itd. Dlatego też teraz przypiszemy do każdej klasy odpowiednią szerokość:

.col-1 {width: 8.33%;}  
.col-2 {width: 16.66%;}  
.col-3 {width: 25%;}  
.col-4 {width: 33.33%;}  
.col-5 {width: 41.66%;}  
.col-6 {width: 50%;}  
.col-7 {width: 58.33%;}  
.col-8 {width: 66.66%;}  
.col-9 {width: 75%;}  
.col-10 {width: 83.33%;}  
.col-11 {width: 91.66%;}  
.col-12 {width: 100%;}

Oczywiście zawsze można dodać dodatkową klasę col i jej przypisać wszystkie wspólne właściwości. Wówczas wystarczy ją dodać do kolumny w ten sposób:

<div class="col col-2"></div>

Nasz grid jest gotowy! Teraz możemy przypisywać naszym elementom szerokość wyrażoną w liczbie kolumn, jak w przykładzie poniżej:

Co w sytuacji, gdy chcemy przesunąć elementy, np. żeby miały margines po lewej stronie równy dwóm kolumnom? Wystarczy w tym celu stworzyć nową grupę klas i zamiast szerokości width przypisać im te same wartości właściwości margin-left. Omijamy naturalnie przesunięcie o 12 kolumn, gdyż wówczas element będzie usunięty poza widok strony. Dodajemy też jedną klasę .col-offset-0, by móc wyzerować margines z lewej strony.

.col-offset-0 {margin-left: 0;}  
.col-offset-1 {margin-left: 8.33%;}  
.col-offset-2 {margin-left: 16.66%;}  
.col-offset-3 {margin-left: 25%;}  
.col-offset-4 {margin-left: 33.33%;}  
.col-offset-5 {margin-left: 41.66%;}  
.col-offset-6 {margin-left: 50%;}  
.col-offset-7 {margin-left: 58.33%;}  
.col-offset-8 {margin-left: 66.66%;}  
.col-offset-9 {margin-left: 75%;}  
.col-offset-10 {margin-left: 83.33%;}  
.col-offset-11 {margin-left: 91.66%;}

Teraz możemy nie tylko ustalać szerokość elementów, ale także ich rozmieszczenie w poziomie. W ten sposób za pomocą przypisywania klas możemy precyzyjnie określić położenie treści na stronie. Ze względu na swoją prostotę, takie rozwiązanie stało się bardzo popularne i można je zaobserwować na wielu stronach.

Aby określić odstępy pomiędzy zawartością kolumn możemy zdefiniować właściwość padding. W ten sposób zawartość naszych kolumn nie będzie przylegać do ich krawędzi, co jest oczywiście pożądane. Przychodzi nam tutaj z pomocą stworzony już przez nas selektor atrybutu:

[class\*="col-"]{  
 float: left;  
 min-height: 1px;  
 padding: 12px;  
}

### **Grid w praktyce**

Kiedy zadbaliśmy o prawidłowe działanie naszych kolumn, możemy stworzone przez nas klasy zastosować w praktyce do opracowania całego layoutu strony. Spójrz na przykład znajdujący się poniżej:

Wykorzystaliśmy tam napisane przez nas klasy do poukładania layoutu strony. Najpierw stworzyliśmy jeden <div> z klasą container, wewnątrz którego będzie znajdować się cała treść. W ten sposób wyśrodkowaliśmy zawartość i stworzyliśmy marginesy po lewej i prawej stronie.

Teraz możemy dodawać treść. Nagłówek powinien zajmować całą szerokość strony, więc stworzyliśmy <div> z klasą row, a samemu nagłówkowi nadaliśmy klasę col-12 - to nadaje mu szerokość 100%.

Poniżej stworzyliśmy dwie sekcje: po lewej i po prawej stronie. Jedna z nich zajmuje ¼ szerokości, a druga ¾ . Aby osiągnąć ten efekt nadaliśmy odpowiednio klasy col-3 oraz col-9 dla poszczególnych sekcji.

Poniżej widać jeszcze inne rozwiązanie, również często stosowane na wielu stronach w sekcjach przedstawiających np. poszczególne usługi. W tym wypadku zastosowaliśmy cztery elementy z klasą col-3. Poniżej tej sekcji znajduje się stopka - tutaj analogicznie do nagłówka nadaliśmy jej klasę col-12.

Na marginesie warto też powiedzieć, że stworzyliśmy tutaj zestaw stylów, których możemy używać wielokrotnie (tzw. reusable styles). Jest to kolejny dobry zwyczaj, który warto praktykować, tworząc kod CSS. W sytuacji gdy pracujemy nad złożonym projektem, możemy nasz układ kolumnowy zastosować w wielu miejscach.

I to nie tylko w jednym projekcie! Dobry front-endowiec wraz z czasem i doświadczeniem tworzy swoją kolekcję fragmentów kodu, które możemy wykorzystywać ponownie, są to tzw. snippets.

Front-endowcy często dzielą się swoimi rozwiązaniami, więc polecamy szukanie snippetów w Internecie, jak i umieszczanie tam swoich rozwiązań, by ułatwić pracę innym. Przykładowe kolekcje znajdują się np. na stronie [css-tricks.](https://css-tricks.com/snippets/) Niezastąpiona jest też biblioteka wzorców na znanym nam już serwisie [Codepen](http://codepen.io/patterns/).

### **Podsumowując:**

Ogólnie staraj się trzymać następujący układ:

<!-- HTML5 tag with our class, for example features, about, jumbotron -->  
<section class="about">  
 <div class="container">  
 <!-- clearfix - row must be the parent of the columns! -->  
 <div class="row">  
 <div class="col-6"></div>  
 <div class="col-6"></div>  
 </div>  
 </div>  
</section>

### **Pixel perfect?**

Kolejną przeszkodą do pokonania na drodze od projektu graficznego do gotowej strony jest pojawiająca się wątpliwość dotycząca tego, do jakiego stopnia szczegółowości odwzorowywać projekt graficzny na stronie internetowej, czyli pytanie o tzw. pixel perfect. W wolnym tłumaczeniu pojęcie pixel perfect oznacza, że projekt graficzny został odwzorowany z dokładnością do pojedynczego piksela. Perfekcja odwzorowania świadczy o wysokiej jakości wykonanej pracy. Jak łatwo się domyślić, odwzorowanie wszystkiego z dokładnością co do piksela bywa trudne lub praktycznie niemożliwe i również wymaga poświęcenia temu odpowiedniego czasu w komunikacji ze zleceniodawcą.

Spróbujmy wyobrazić sobie, że ktoś otwiera naszą stronę na czarno-białym czytniku Kindle, a ktoś inny na 60-calowym telewizorze z UltraHD. Nie ma technicznej możliwości, żeby to wyglądało identycznie. Rozważając zagadnienie perfekcyjnego odwzorowania projektu graficznego, trzeba wziąć pod uwagę skalę projektu i jego koszty oraz przede wszystkim oczekiwania klienta.

### **Wypełniacze**

Zdarza się, że we wczesnej fazie projektu nie mamy jeszcze do dyspozycji pełnej treści, jaka docelowo znajdzie się na stronie, dlatego też warto jest mieć pewien zasób tzw. "wypełniaczy". W przypadku tekstu jest to używane od wieków *Lorem Ipsum*. Tekst jest pozbawiony sensu, lecz zachowuje równowagę w liczbie liter, dzięki czemu łatwiej zobaczyć wygląd tekstu bez skupiania się na jego treści. Generator tekstu Lorem Ipsum dostępny jest pod adresem [www.lipsum.com](http://www.lipsum.com/). Dostępne są też generatory "wypełniaczy" elementów HTML: <http://html-ipsum.com/>. Aby przetestować wygląd tekstu z polskimi znakami, możemy tez skorzystać z tekstu Pana Tadeusza <http://lipsum.pl/index.php>. W tytułach dobrze jest wstawić pangramy, czyli krótkie zdania zawierające dużo polskich znaków. Fonty używane w tytułach są często bardziej wymyślne i nie mają wszystkich polskich znaków.

Przejdźmy teraz do zdjęć. Poznaliśmy już serwis pexels, gdzie można znaleźć darmowe zdjęcia. Innymi popularnymi serwisami z darmowymi zdjęciami są m.in. [Unsplash](https://unsplash.com/) oraz [stocksnap.io](http://stocksnap.io/)

Często potrzebujemy szybko dodać zdjęcia o konkretnych wymiarach, które można umieścić w layoucie strony zamiast np. bannerów reklamowych. Wówczas, zamiast pobierać zdjęcie i wycinać je do odpowiedniego rozmiaru, możemy skorzystać z kilku serwisów, które zwracają obrazki w dowolnych wymiarach, wykorzystując parametry umieszczane w wywołaniu HTTP.

Korzystamy z nich, dodając poniższy adres do atrybutu "src", jak w przykładzie poniżej:

<img src="http://placehold.it/350x150" alt="">

W tym wypadku dostaniemy obrazek z serwisu placehold.it o wymiarach 350 na 150 pikseli. Spróbuj otworzyć link w przeglądarce albo dodać go do projektu! Aby nieco urozmaicić swoje wypełniacze, mamy też serwisy zwracające np. zdjęcia:

<img src="http://lorempixel.com/300/300/" alt="">



### **Logika działania strony**

Po zbudowaniu frontu przychodzi czas na przygotowanie tego, co się dzieje się za kulisami, po stronie serwera. To jest tzw. back-end. Z reguły w przypadku dużych projektów szablon HTML/CSS przekażesz do innej osoby (lub całego zespołu) zajmującej się back-endem. Jednak nie zawsze tak jest. W przypadku małych zleceń, np. strony firmy, bloga lub sklepu internetowego popularnym rozwiązaniem będzie podłączenie front-endu do systemu zarządzania treścią, tzw. CMS-a (Content Management System).

Popularnymi CMS-ami są np. Wordpress, Joomla! lub Drupal. Dzięki nim klienci mogą modyfikować zawartość swojej strony przy pomocy przyjaznego interfejsu bez konieczności każdorazowego zaglądania do kodu, nie mówiąc już o dodaniu możliwości obsługi np. sprzedaży czy dodawania wpisów na blogu. Warto mieć na uwadze ten etap tworząc front-end. Trzeba upewnić się, że strona będzie działać zgodnie z założeniem w sytuacji, gdy klient będzie modyfikował jej zawartość. Jednak tymi zagadnieniami nie będziemy się zajmować.

### **Właściwości rozmieszczenia**

Layout strony możemy kształtować, nadając elementom HTML naszej strony odpowiednie właściwości w pliku CSS, które ustalą pozycję naszej treści na stronie. Do takich stylów, które determinują układ strony, należą m.in. następujące właściwości CSS:

* Display - sposób wyświetlenia elementu HTML.
* Position - pozycja elementu HTML.
* Float - opływanie elementu HTML przez inne elementy.

Poprzez przypisanie odpowiednich wartości tym właściwościom będziemy mogli precyzyjnie określić, w którym miejscu ma znaleźć się dany element oraz w jakiej relacji ma on pozostawać względem pozostałych.

Ze względu na mnogość urządzeń, na których obecnie wyświetlane są strony internetowe, zapewnienie odpowiedniego wyświetlania nabiera kolosalnego znaczenia. Nowoczesnym sposobem na rozwiązanie tego problemu jest tzw. Flexbox, z którym zapoznasz się w kolejnym module.

# **Responsywny layout!**

### **AWD, RWD…**

Z pewnością wiele razy trafiałeś na strony, których adres rozpoczyna się od m. lub mobile. (lub inne konwencje), gdy odwiedzasz je na urządzeniu mobilnym. Taki oddzielny adres jest stosowany właśnie w przypadku urządzeń przenośnych.

Gdy wpisujesz w telefonie np. adres wp.pl, ta strona za pomocą kodu JavaScript rozpoznaje typ urządzenia (np. smartfon) i przekierowuje Cię na specjalny adres, w tym wypadku <http://www.wp.pl/mini.html>. Jak widać, taka wersja jest lepiej dostosowana do mniejszych wyświetlaczy.



Opisane powyżej podejście to tzw. AWD, czyli Adaptive Web Design. W takim wypadku strona mobilna (smartfony, tablety) i desktopowa (komputery stacjonarne, laptopy) to dwa oddzielnie realizowane projekty. Inny szablon front-endowy buduje się dla smartfona, a inny dla laptopa.

Takie podejście sprawia jednak, że trzeba napisać znacznie więcej kodu, by je stworzyć. Także wszelkie zmiany wyglądu, np. na stronie głównej, wymagają modyfikacji nie w jednym, ale w dwóch szablonach. Takie podejście jest stosowane, kiedy:

* grafik/projektant z jakiegoś powodu założył, że wersja mobilna będzie się bardzo różnić od desktopowej i łatwiej jest wtedy stworzyć oddzielny szablon,
* witryna ma ogromny ruch internetowy i przeznaczony dla mobile szablon jest zoptymalizowany pod kątem wydajności i szybkości ładowania versus obciążenie serwerów.

A propos pierwszej wymienionej przyczyny - trend jest taki, aby mocno ujednolicać wizualnie widok desktopowy i mobilny. Ma to głównie zaletę "brandingową" - niezależnie od urządzenia, z którego korzysta, internauta wie, na jakiej jest stronie.

Zupełnie odmiennym podejściem w zakresie dostosowywania witryny pod wiele urządzeń jest RWD, czyli Responsive Web Design. Opiera się ono na tworzeniu jednego szablonu pod wszystkie urządzenia wraz z dodatkowymi regułami w CSS, które mówią o tym, jak ma wyglądać strona na poszczególnych rozdzielczościach.

Oznacza to, że niezależnie od tego, czy oglądasz stronę na telefonie, tablecie, komputerze, czy też telewizorze, Twoja przeglądarka otrzymuje zawsze te same pliki HTML oraz CSS i używa odpowiednich reguł CSS w zależności od rozmiarów ekranu. Np. poniższa strona to jeden szablon HTML/CSS, który dostosowuje się do różnych urządzeń.

### **Web-mobile a aplikacje mobilne**

Istnieją dwie główne drogi dotarcia do użytkownika mobilnego. Pierwsza to aplikacje mobilne, tzw. natywne. Są to po prostu programy, które instalujesz np. z Google Play. Tworzy się je oddzielnie na każdą platformę (system) mobilną:

* Google Android - wersje od 4.0 do 6.x,
* iOS (iPhone OS) - obecnie głównie wersje 9+,
* Windows Mobile, Blackberry, Symbian - rzadko; głównie obsługuje się dwa powyższe systemy ([link do statystyk](https://d28wbuch0jlv7v.cloudfront.net/images/infografik/normal/chartoftheday_4431_smartphone_operating_system_market_share_n.jpg)).

Sposób ten ma oczywiście swoje niezaprzeczalne zalety, gdyż daje pełne możliwości wykorzystania telefonu, których nie daje nam przeglądarka. W aplikacjach mobilnych możemy skorzystać z takich funkcji, jak np. aparat, GPS czy żyroskop. Zazwyczaj wymaga to każdorazowego tworzenia, a następnie aktualizacji aplikacji z myślą o konkretnej platformie. Nie tylko oznacza to dużo więcej pracy, ale też wymaga dużo większej wiedzy na temat specyfiki danego systemu. Do tworzenia takich aplikacji najczęściej wykorzystuje się języki Java (dla Android, nie mylić z JavaScript) i m.in. Swift (dla iOS).

Drugi typ rozwiązań mobilnych to responsywne strony internetowe (tzw. web-mobile). Są one podobne do tych tradycyjnych, jednak skonstruowane w taki sposób, aby uwzględniać charakterystyczne cechy urządzeń mobilnych (mały rozmiar ekranu, dotykowa obsługa).

Chociaż zewnętrznie często są do siebie podobne, to aplikacje i strony mobilne znacznie się różnią. Te pierwsze oferują więcej funkcjonalności (dostępnych także offline) i przeważnie są bardziej rozbudowane (np. gry).

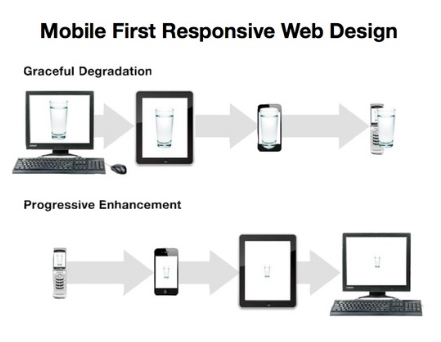
Strony mobilne są prostsze, nie działają bez dostępu do Internetu, ale są bardziej uniwersalne. Działają na wszystkich urządzeniach i systemach mobilnych, na których jest przeglądarka internetowa. Ich zaletą jest też to, że działają niezwłocznie, bez konieczności pobierania i instalowania czegokolwiek.

Ponadto strony internetowe posiadają wspólny kod dla wszystkich platform, a aplikacje na iOS oraz Androida lub inny system tworzone są w zupełnie inny sposób.

Na marginesie dodać można, że różnice między aplikacjami mobilnymi a web-mobile w ostatnim czasie nieco się zacierają, gdyż powstają coraz bardziej skuteczne techniki tworzenia aplikacji natywnych z wykorzystaniem HTML/CSS/JS (tzw. hybrid mobile apps).

### **Dwie drogi w RWD**

W tym kursie zajmujemy się web-mobile z użyciem RWD. W zakresie tworzenia stron responsywnych wyróżnia się 2 podejścia.



#### **Graceful degradation**

Czyli przypadek, w którym zaczynamy od wersji strony dla ekranów o dużej rozdzielczości (desktop) i dopiero potem dostosowujemy ją do ekranów mniejszych.

Polega to najczęściej na upraszczaniu zbudowanego layoutu pod mobile i zmianie ułożenia poszczególnych elementów tak, aby zapewnić im odpowiednie wyświetlanie na mniejszych ekranach. Raczej powinno się unikać ograniczania funkcjonalności strony, ale czasami nie ma wyjścia :) Nie wszystko da się "upakować" w dziesięciokrotnie mniejszy ekran.

#### **Progressive enhancement**

Znane także jako mobile first. Polega na stworzeniu najpierw wersji dostosowanej do rozdzielczości właściwych urządzeniom mobilnym, a następnie rozbudowywanie jej i przystosowanie do większych ekranów. Zgodnie z tą metodą tworzymy najpierw prostą wersję mobilną, którą następnie rozbudowujemy dla ekranów szerszych. W tym podejściu unikamy sytuacji, w której tworzymy spektakularny produkt wykorzystujący wiele zaawansowanych technologii w wersjach desktopowych tylko po to, żeby odkryć, jak wiele z nich nie działa na przeglądarkach mobilnych.

Które podejście wybrać? Nie jest jednak tak, że jedno podejście jest w naturalny sposób lepsze czy gorsze. W praktyce ciągle trwa wśród front-endowców gorąca debata o wyższości jednego sposobu nad drugim. My - po początkowym zachwycie nad mobile-first - skłaniamy się bardziej ku temu, aby zaczynać od desktopów, ale ze świadomością, że potem czeka nas mobile.

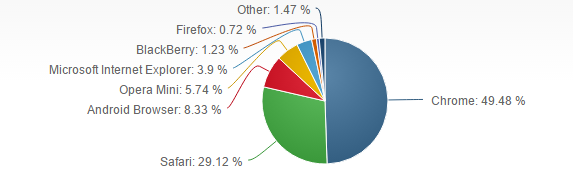
### **Przeglądarki mobilne**

****

Sytuacja na rynku przeglądarek mobilnych szybko się zmienia, dlatego dobrze jest mieć ogólne pojęcie o popularności poszczególnych przeglądarek. Wśród desktopowych królują: Google Chrome, Firefox, Safari, Internet Explorer (Edge) oraz Opera.

Na rynku mobilnym do najpopularniejszych należą "odchudzone" wersje przeglądarek desktopowych, czyli: Chrome, Opera, Safari czy Internet Explorer. Poza tym popularne są również przeglądarki typowe dla systemów mobilnych, jak Android Browser czy UC Browser.

Kwestia ilu użytkowników ma dana przeglądarka mobilna jest istotna z tego względu, że starsze wersje przeglądarek nie wspierają nowoczesnych rozwiązań HTML, CSS oraz JavaScript. Może się więc okazać, że nasza mobilna strona internetowa, która korzysta z takich rozwiązań, nie zadziała u sporej części użytkowników. Dlatego też przy tworzeniu strony mobilnej musimy zastosować metodę podobną do tej, którą omawialiśmy w poprzednim module i rozważyć, jakie przeglądarki będziemy wspierać (poniżej statystyki użycia przeglądarek internetowych).



Aby nie testować strony na pięćdziesięciu przeglądarkach mobilnych, można ogólnie przyjąć obsługę Chrome (prawie to samo, co Firefox i Android Browser) i Safari. Z racji niskiego udziału w rynku Windows Phone na rynku można odpuścić Internet Explorera, chyba że trafisz na klienta z takim telefonem :) Opera mini jest także używana na starych telefonach, ale robienie pod nią RWD jest praktycznie niemożliwe. Postaw na razie na Chrome i Safari :)

Budując stronę z RWD, trzeba też wziąć pod uwagę specyfikę urządzeń mobilnych. Użytkownicy korzystający z komputerów desktopowych mają do dyspozycji klawiaturę i mysz, a w przypadku tabletów i smartfonów interakcja z użytkownikiem odbywa się za pomocą ekranu dotykowego.

W efekcie musimy wziąć pod uwagę specyfikę tej interakcji przy tworzeniu interfejsu użytkownika. Elementy nawigacji powinny być dostatecznie duże, tak aby zapewnić ich łatwą klikalność na ekranie dotykowym. Jednak nie powinny one przysłaniać pozostałej treści. Trzeba też pamiętać, że na urządzeniach mobilnych nie ma takiego stanu jak :hover, co jest logicznie zrozumiałe (nie ma stanu palca nad elementem).

Dobrze jest też zastanowić się, czy użytkownicy mobilni nie będą korzystać ze strony w poszukiwaniu nieco innych informacji niż desktopowi. W praktyce może to oznaczać, że wizualna hierarchia poszczególnych treści na urządzeniach mobilnych będzie się nieco różnić.

## **Jak to się robi?**

Responsywne strony budowane są przy użyciu trzech głównych metod:

* regułach media queries,
* layoucie dostosowującym się do szerokości okna przeglądarki,
* responsywnych obrazkach.

### **Media queries**

Jest to najczęściej wykorzystywana do budowania stron w podejściu RWD reguła w ramach CSS. Media queries umożliwiają dostosowanie sposobu wyświetlania witryny, najczęściej do rozmiarów ekranu. Są to dodawane w pliku CSS reguły sprawiające, że niektóre style zostają uruchomione tylko dla urządzeń określonego typu lub o określonej rozdzielczości.

Dzięki temu możemy sprawić, że na komputerze strona będzie oferować więcej treści bądź rozszerzoną funkcjonalność w stosunku do wersji mobilnej. Za pomocą tych reguł można także sprawić, że dany element inaczej wyświetli się w mniejszej rozdzielczości (będzie np. mniejszy).

#### **Przykład użycia**

Zobaczmy to na przykładzie.W osobnym oknie przeglądarki otwórz stronę <https://www.smashingmagazine.com/>. Pobaw się nią, zmniejszając i zwiększając stopniowo szerokość przeglądarki wszerz. Obserwuj, jak zachowują się elementy na stronie (szczególnie linki w menu oraz zdjęcia). Postaraj się znaleźć rozmiary, w których strona zasadniczo zmienia swój wygląd (np. z układu z dwoma lub trzema kolumnami przechodzi w jedną).

Jeśli otwierasz stronę w Chrome, włącz także Developer Tools (F12, aby był na dole). Wtedy zmieniając rozmiar przeglądarki, w prawym górnym rogu zobaczysz aktualną rozdzielczość.

#### **Breakpointy**

Momenty zmiany układu strony to tak zwane breakpointy. Projektując czy tnąc stronę, określamy takie breakpointy. Potem określamy reguły z wartościami pikselowymi tych breakpointów w CSS (przy pomocy media queries). Tworzą się swego rodzaju przedziały (pomiędzy breakpointami), w obrębie których strona przyjmuje dany układ/wygląd).

Poniżej podajemy składnię reguły media queries (w nawiasach zwykłych nigdy nie dajemy średnika, a nawiasy wąsy, jak widać, są podwójnie zagnieżdżone):

@media (max-width: 600px) {  
/\* Niżej dodajemy reguły CSS, które zadziałają tylko dla ekranów o maksymalnej szerokości 600px \*/  
 .class {  
 color: #000;  
 }  
}

Działanie powyższego przykładu możesz zobaczyć w prosty sposób: zmień szerokość podglądu i zobacz, jak zmienia się kolor sekcji.

#### **Rodzaje obsługiwanych urządzeń**

W regułach @media możemy także określić typ urządzenia, które chcemy obrać za cel. Najpopularniejsze to all, screen, tv czy print. Istnieją też bardziej egzotyczne rodzaje jak braille - dla urządzeń dla niewidomych czy 3d-glasses dla okularów 3d.

Możesz nic nie wpisywać (tak robi się najczęściej) i wtedy dana reguła zostanie zastosowana do wszystkich urządzeń, w tym screen, których rozdzielczość najbardziej nas interesuje.

#### **Operatory logiczne**

Poza typami urządzeń możemy także używać operatorów logicznych takich jak and, not, only. Dzięki temu możemy stworzyć rozbudowane wyrażenia, w których precyzyjnie określimy, jaki typ urządzenia chcemy objąć regułą.

Poniżej przykład wykorzystania operatorów logicznych do określenia typu urządzenia i jego rodzaju:

@media screen and (min-width: 800px) and (max-width: 1024px) {...}

Reguła ta wskazuje na ekran urządzenia i zostanie zastosowana w przypadku, gdy jego szerokość będzie się mieściła w przedziale od 800px (minimalna szerokość) do 1024px (maksymalna szerokość). Poza tym przedziałem zdefiniowane w regule style (w {...}) nie zadziałają.

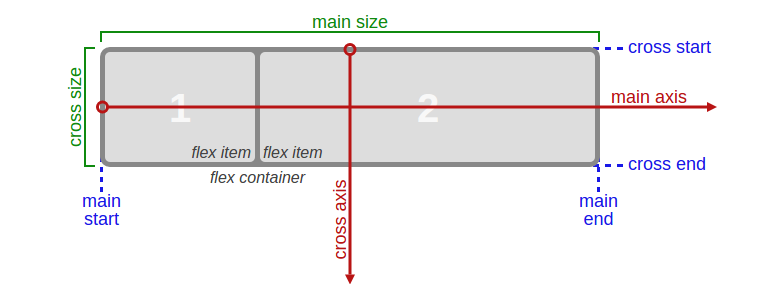
#### **Warunki**

W regule, w zwykłych nawiasach, określa się następujące właściwości i ich wartości:

1. Min-width i max-width - określające szerokość urządzenia (a dokładnie viewportu, o czym przeczytasz poniżej).
2. Orientation - orientacja pionowa lub pozioma urządzenia. Dostępne wartości to portrait (wysokość ekranu jest większa niż szerokość) oraz landscape (na odwrót). Ma to szczególnie zastosowanie w dostosowywaniu strony do wyświetlacza tabletu lub telefonu, które mogą być trzymane przez użytkownika zarówno pionowo jak i poziomo.
3. Resolution - przy jej pomocy możemy określić reguły CSS, które będą miały zastosowanie tylko, jeśli "gęstość upakowania pikseli" w urządzeniu będzie miała określoną wartość (podanej w dpi, czyli plamce na cal lub dpcm - plamce na centymetr).

W 99% przypadków używa się właściwości max-width i min-width, gdzie określa się dodatkowe działanie stylów w danych przedziałach rozdzielczości poziomej (pomiędzy breakpointami).

### **Terminologia**

****

Anatomię layoutu tworzonego przy użyciu flexboksa przedstawia nam powyższy schemat. Będziemy korzystać przede wszystkim z dwóch elementów: flex container, czyli rodzica interesujących nas elementów, oraz flex item, czyli jego bezpośrednich potomków w drzewie dokumentu. Main axis to główna oś, wzdłuż której nasze elementy będą rozmieszczane, zaś jej długość określana jest jako main size. Dla drugiej osi (cross axis) będzie to określane analogicznie za pomocą cross start oraz cross size.

### **Kontrolowanie zachowania kontenera**

Wszystkie właściwości, jakie udostępnia nam flexbox, możesz przetestować na stronie <http://the-echoplex.net/flexyboxes/>

Aby rozpocząć budowę layoutu opartego o flexbox, musimy nadać kontenerowi poniższą właściwość:

.flex-container {  
display: -webkit-flex;  
display: flex;  
}

Dodana w ten sposób własność display wystarczy do uruchomienia tej funkcjonalności w większości przeglądarek. Powyższa deklaracja stworzy kontener będący elementem blokowym, możemy stworzyć również element liniowy za pomocą deklaracji

.inline-flex-container {  
display: -webkit-inline-flex;  
display: inline-flex;  
}

Przyjrzymy się teraz deklaracjom, które możemy wykorzystać do manipulowania rozmieszczeniem elementów wewnątrz kontenera.

#### **Flex-direction**

Ustala kierunek i zwrot głównej osi (main-axis). Domyślną wartością jest row, czyli rozmieszczenie elementów w poziomym rzędzie, od lewej do prawej. Za pomocą wartości column tworzymy pionową kolumnę. Możemy odwrócić kolejność elementów wewnątrz kontenera, dodając odpowiednio row-reverse oraz column-reverse.

#### **Flex-wrap**

Domyślne elementy wewnątrz kontenera będą umieszczone w jednej linijce, jednak za pomocą tej właściwości możemy to zmienić. Gdy przypiszemy kontenerowi właściwość flex-wrap: wrap, elementy będą rozmieszczone w wielu linijkach. Podobnie jak poprzednio możemy tutaj zmienić kierunek, deklarując wrap-reverse.

#### **Justify-content**

Pozwala na rozmieszczenie elementów wzdłuż głównej osi. Mamy tutaj następujące możliwości:

* flex-start (domyślne) - elementy będą umieszczone na początku linii,
* flex-end - na końcu linii,
* center - na środku,
* space-between - elementy będą rozciągnięte wzdłuż głównej osi; pierwszy będzie na początku linii, ostatni na końcu, a pozostałe będą rozmieszczone na środku,
* space-around - podobne do space between, jednak zachowane będą równe odległości pomiędzy poszczególnymi elementami; pierwszy i ostatni element będą oddalone od krawędzi kontenera jedynie o połowę odległości pomiędzy poszczególnymi komórkami.

#### **Align-items**

Za jego pomocą możemy modyfikować rozmieszczenie elementów na osi poprzecznej (cross-axis). Można przyjąć, że jest to funkcjonalność podobna do justify-content, jednak w wersji dla osi poprzecznej. Podobnie jak w poprzednim przypadku mamy tutaj do dyspozycji różne możliwości.

* flex-start - górne krawędzie elementów będą umieszczone przy górnej krawędzi kontenera,
* flex-end - analogicznie jak w przypadku flex-start, jednak dla dolnych krawędzi: będą one dociągnięte do dolnych krawędzi kontenera,
* center - elementy będą umieszczone centralnie na osi poprzecznej,
* baseline - elementy będą rozmieszczone tak, by zachować ich linię bazową, tj. by litery w poszczególnych elementach były umieszczone na jednej linii,
* stretch - zawartość będzie rozciągnięta, aby wypełnić kontener (warto przy tym zaznaczyć, że wartości min-width/max-width będą zachowane).

#### **Align-content**

Umożliwia rozmieszczenie linii z elementami wewnątrz kontenera w sytuacji, gdy jest ich więcej niż jedna linijka (dlatego też nie będzie to mieć znaczenia wtedy, gdy nie zadeklarowaliśmy odpowiednio właściwości flex-wrap).

### **Właściwości Flexboksa dla elementów wewnątrz kontenera**

Flexbox daje nam również możliwość manipulowania poszczególnymi elementami. Dzięki nim możemy przykładowo zmieniać kolejność elementów lub ich rozmiar.

#### **Order**

Elementy wewnątrz flexboksa będą domyślnie wyświetlane w takiej kolejności, w jakiej znajdują się w źródle strony. Możemy to zmodyfikować, dodając właściwość order do interesującego nas elementu. Przyjmuje on wartość liczby całkowitej.

#### **Flex-grow**

Definiując flex-grow możemy proporcjonalnie powiększyć konkretny element, jeżeli tego potrzebujemy. Przykładowo: jeżeli wszystkie elementy mają zdefiniowany flex-grow: 1, a jeden będzie określony z flex-grow: 2, będzie on zajmował dwa razy więcej wolnego miejsca (jeżeli będzie taka możliwość wewnątrz kontenera).

#### **Flex-shrink**

Podobnie jak w poprzednim przypadku, jednak tym razem nasz element zostanie pomniejszony.

#### **Flex-basis**

Określa domyślny rozmiar elementu. Za nim rozmieszczona jest pozostała zawartość i przydzielone jej miejsce wewnątrz kontenera. Wartość auto powoduje, że będzie to rozpatrzone na podstawie własności width oraz height. Możemy również przydzielić rozmiar elementu na podstawie zawartości nadając wartość content.

#### **Flex - właściwość skrótowa**

Podobnie jak w wypadku definiowania marginesów, możemy zdefiniować jednocześnie flex-grow, flex-shrink oraz flex-basis. Domyślną wartością jest flex: 0 1 auto;.

#### **Align-self**

Ta właściwość to nic innego jak nadpisanie położenia pojedynczego elementu, które było zdefiniowane przy użyciu align-items. Dlatego też może przyjmować identyczne wartości.

<https://codepen.io/kodilla/pen/5f2a660935d96ebaf17c090017257bf3/>

## Formularze

<input type="text">

<fieldset>

<div>

<input type="radio" name="gender" value="female">

<span>Female</span>

</div>

<div>

<input type="radio" name="gender" value="male">

<span>Male</span>

</div>

</fieldset>