Menu

Czy to dla mnie?

1

W dzisiejszych czasach, kiedy właściwie każdy element pracy na komputerze wiąże się z korzystaniem z Internetu, ważne jest znanie zasad funkcjonowania sieci oraz stron internetowych.

Książkę tę kierujemy do dwóch rodzajów ludzi:

- chcących nauczyć się, jak tworzyć i projektować strony internetowe,
- posiadaczy własnej strony internetowej (bloga, sklepu internetowego czy systemu zarządzania treścią) pragnących mieć większą kontrolę nad jej zawartością.

Jedyne, co potrzebujesz do nauki to komputer z zainstalowaną przeglądarką internetową i dowolnym edytorem tekstowym. Nie ma znaczenia czy korzystasz z zaawansowanego programu, czy ze zwykłego notatnika. Najważniejsze, aby wygodnie Ci się z niego korzystało.

Czy to trudne?

Przeglądając kod dowolnej strony internetowej możemy odnieść wrażenie, że jest on bardzo trudny i skomplikowany. Ilość kodu oraz mnogość znaczników mogą odstraszyć początkowego użytkownika.

Jednak, gdy się w niego zgłębimy okaże się, że w rzeczywistości to logiczna układanka złożona z mniejszych puzzli. Poznawanie krok po kroku tych najmniejszych elementów spowoduje, że wszystko powoli zaczyna nabierać sensu i całość od razu wydaje się łatwiejsza. Na pewno nie trzeba być "programistą", żeby zacząć przygodę z tworzeniem stron internetowych.

Zrozumienie podstawowych elementów HTML i CSS pomoże każdemu, kto pracuje w sieci: projektanci mogą tworzyć bardziej atrakcyjne i użyteczne strony, redaktorzy lepsze treści, a marketingowcy mogą efektywniej komunikować się ze swoimi odbiorcami.

Jak to wszystko działa?

Zanim zaczniemy prawdziwą przygodę z tworzeniem stron internetowych ważnym jest, abyśmy poznali zasady funkcjonowania Internetu oraz witryn w nim się znajdujących. Zapoznamy się również z terminologią, jakiej będziemy używali w niniejszej książce. A więc do dzieła!

Przeglądarka internetowa

Ludzie korzystający z Internetu używają oprogramowania, które nazywamy **przeglądarką internetową**. Przykładowymi aplikacjami pozwalającymi surfować po Internecie są: *Chrome*, *Firefox*, *Opera*, *Internet Explorer* czy *Safari*.

W celu odwiedzenia strony internetowej musimy wpisać adres w przeglądarce lub skorzystać z wbudowanych zakładek.

Firmy zajmujące się tworzeniem przeglądarek internetowych dbają o to, aby ich oprogramowanie spełniało najnowsze standardy i było przyjazne dla użytkownika. Ważne, by zawsze korzystać z najnowszej wersji przeglądarki, gdyż pozwoli nam to uniknąć błędów w wyświetlaniu stron, a w skrajnych przypadkach zabezpieczy nas przed ewentualnym dziurami i błędami w aplikacji.

Podczas pracy nad tym podręcznikiem będziemy używali najnowszej wersji przeglądarki *Chrome*. Jednak możesz używać innej przeglądarki - o ile będzie aktualna, nie powinna sprawiać problemów z nauką języka HTML oraz przeglądaniem przykładów zawartych w tej publikacji.

Serwer

Samo stworzenie strony internetowej to dopiero początek zabawy z Internetem. Kolejnym etapem jest udostępnienie jej szerokiemu światu za pomocą **serwera**.

Kiedy przeglądarka internetowa chce pobrać stronę internetową łączy się właśnie z serwerem, udostępniającym zasoby witryny. To specjalny komputer, działający 24 godziny na dobę oraz posiadający stały dostęp do Internetu. Dzięki temu każda osoba podłączona do sieci będzie mogła w dowolnym czasie odwiedzić naszą stronę.

Oczywiście możliwe jest hostowanie ze swojego własnego komputera. Jednak w praktyce takie rozwiązanie nie opłaca się, gdyż koszt prądu oraz łącza internetowego wielokrotnie przewyższa ceny oferowane przez **firmy hostingowe**.

Nasza szkoła na szczęście posiada własne serwery, które będą bezpiecznie i bezpłatnie udostępniać twoją stronę internetową. Jeśli tylko będzie ona odpowiednio zbudowana i nie będzie drastycznie obciążać zasobów szkoły to nie powinno być problemów aby właśnie tam była przechowywana.

Domena

Żeby strona była chętnie odwiedzana przez gości musi posiadać odpowiedni adres. Jest on w praktyce uzależniony tylko i wyłącznie od naszej wyobraźni i zasobności portfela.

Domena to tak naprawdę "adres zamieszkania" naszej strony. Na jej podstawie użytkownicy mogą trafić bezpośrednio do serwera, na którym jest przetrzymywana.

Urządzenia

Pisząc stronę internetową musimy wziąć pod uwagę niezliczoną liczbę urządzeń, za pomocą których użytkownicy będą ją przeglądać. Mowa tu o komputerach stacjonarnych, laptopach, tabletach, smartfonach czy nawet telewizorach lub zegarkach. Każde z nich cechuje się innym rozmiarem ekranu, mocą obliczeniowa czy prędkością łącza.

Pamiętajmy więc, aby nasza strona była jak najlepiej zoptymalizowana i odpowiednio dostosowana do różnego rodzaju urządzeń. Nawet jeśli mrugające literki wyglądają dobrze na laptopie, na telefonie komórkowym już niekoniecznie.

Wyszukiwarki internetowe

Wygląd to nie wszystko. Po kodzie znajdującym się na Twoim serwerze nie raz przemknie robot indeksujący dane do wyszukiwarek internetowych. Wtedy nie ma większego znaczenia kolor tekstu czy tła używany w witrynie. On skupia się na elementach znajdujących się na stronie. Na ich podstawie ocenia jakość udostępnionych przez Ciebie treści oraz przypasowuje je do odpowiednich fraz w wynikach wyszukiwania.

Dlatego będziemy kłaść duży nacisk na poznawane typy elementów. Pozwoli to już na starcie wyróżnić Twoją stronę na tle konkurencji.

Czy taka zabawa jest droga w utrzymaniu?

Tutaj nie da się udzielić jednoznacznej odpowiedzi. Wszystko zależy od tego, co będziemy przetrzymywać na naszej stronie. Przy obliczaniu potencjalnych kosztów należy wziąć pod uwagę kilka czynników:

- **serwer** — dla prostych stron koszt utrzymania waha się od kilku do kilkunastu złotych miesięcznie. Większe projekty używające rozbudowanego oprogramowania oraz mające dosyć duży ruch gości wymagają większych wydatków, nawet kilkuset złotych miesięcznie. Nam na szczęście na początek wystarczy zupełnie darmowy serwer szkolny.

- **domena** — celem zapamiętania adresu przez naszych gości musimy zadbać o jego atrakcyjność. Najczęściej używanymi w Polsce końcówkami są `.pl`, `.com` oraz `.eu`. Ich koszt to przeważnie 50-100zł rocznie, w zależności od pośrednika zakupu. Firmy hostingowe udostępniają też darmowe końcówki, które możemy wykorzystać serwując naszą stronę z ich serwera.
- **koszt oprogramowania** — budując stronę samemu jedynym kosztem będzie nasz prywatny czas. Aczkolwiek gdy zdecydujemy się skorzystać z gotowego oprogramowania (sklepu internetowego, forum dyskusyjnego), koszty wzrosną - od kilkudziesięciu złotych za proste skrypty, po tysiące dolarów za specjalistyczne. W praktyce da się uniknąć tych opłat stosując darmowe oprogramowanie, niejednokrotnie przewyższające jakością płatne wersje.

Jak są tworzone strony internetowe

Wszystkie strony internetowe używają języków HTML oraz CSS, ale zaawansowane systemy zarządzania treścią, blogi, czy sklepy internetowe potrafią korzystać z wielu innych języków programowania takich jak JavaScript, PHP czy rozbudowanych baz danych. W tej książce skupimy się na dwóch pierwszych językach.

Co możesz zobaczyć

Kiedy odwiedzasz stronę internetową twoja przeglądarka pobiera z serwera zestaw komend, które przekazują jej jak ma ona wyglądać. HTML jest odpowiedzialny za treść i strukturę, natomiast CSS wpływa na wygląd poszczególnych elementów.

W Internecie znajdują się także dodatkowe elementy, takie jak obrazki, pliki dźwiękowe czy filmy wideo. W książce poznasz sposoby na umieszczenie ich w witrynie.

JavaScript odpowiada za interakcję z użytkownikiem. Jest to zaawansowany język, jednak śmiało możesz zacząć poznawać go zaraz po zostaniu specjalistą od HTML i CSS.

HTML5 i CSS3

HTML5 oraz CSS3 to języki, które przez dziesięciolecia ewoluowały i obecnie pozwalają na tworzenie bardzo zaawansowanych i rozbudowanych witryn.

Wiele elementów wchodzących w skład tych standardów jest cały czas rozbudowywanych i poszerzanych. Większość przeglądarek na bieżąco wprowadza aktualizacje, pozwalające nam cieszyć się coraz to nowszymi technologiami.

A więc zaczynajmy!

2

Każdy dokument, który tworzymy np. w edytorze tekstowym ma określoną strukturę. Możemy wyodrębnić w nim nagłówki, akapity, listy, obrazki, itp. Jest to wizualna część naszej pracy: to co napiszemy, znajdzie się w naszym dokumencie.

Tagi

Zobacz na kod **1.1**

Poszczególne elementy mogą mieć swoje miejsce w hierarchii: tytuł strony jest najważniejszy, po nim znajdziemy różnej wielkości nagłówki, a na końcu zwykły tekst. Podobnie jest w języku HTML: każdy poszczególny fragment tekstu możemy opisać za pomocą specjalnych znaczników. Zobaczmy na poniższy przykład:

HTML

```
<h1>Najważniejszy nagłówek</h1>
<p>Ten tekst może być krótkim streszczeniem tego, co znajdziemy
w kolejnych częściach dokumentu<p>

<h2>Nagłówek drugiego stopnia</h2>
<p>To troszkę dłuższy opis, zawierający dużo
praktycznych i mniej praktycznych informacji</p>

<h2>Kolejny nagłówek</h2>
<p>Pod tym akapitem znajduje się linia przerwy</p>
<br/>
<p>Nad tym akapitem znajduje się linia przerwy</p>
```

Jak widać na powyższym przykładzie, elementy kodu HTML tworzymy za pomocą **znaczników** w nawiasach trójkątnych, zwanych też **tagami**. Większość elementów, jak akapity czy nagłówki wymaga dwóch (otwierający i zamykający, mający przed nazwą znak `/`),

ale np. do wyświetlenia obrazka (`<img/>`) użyjemy jednoczłonowego tagu.

Każdy z elementów strony internetowej mówi przeglądarce, co może się wewnątrz niego znajdować i w jaki sposób ma ona renderować zawartość znajdującą się pomiędzy tagami.

CIEKAWOSTKA

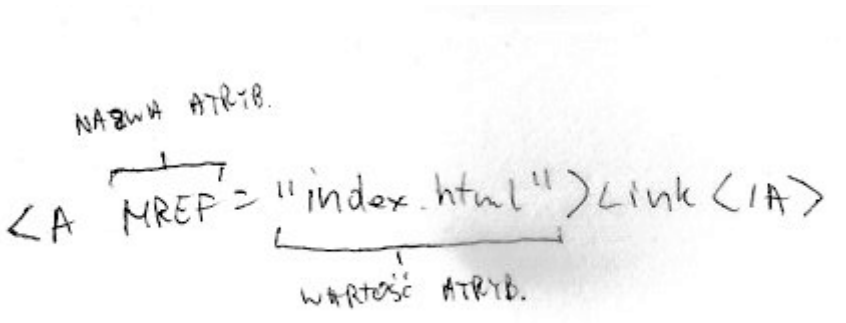
Tagi `<h1>` oraz `<h2>` z poprzedniego przykładu oznaczają nagłówki, natomiast `<p>` oznacza zwykły akapit.

Atrybuty

Atrybuty zostały stworzone, aby lepiej opisywać zawartość poszczególnych elementów. Każdy z nich składa się z **nazwy** oraz **wartości** ujętej w cudzysłów.

```html
<p lang="en">Paragraph in English</p>
<p lang="fr">Paragraphe en Français</p>
```

Powyższy tag `<p>` nadal przekazuje przeglądarce, że zawiera tekst, natomiast o jego języku informuje atrybut `lang` - tutaj wystąpił angielski — `en` i francuski — `fr`.



Przyjęło się, że atrybuty, podobnie jak tagi piszemy małymi literami.

Białe znaki

Język HTML jest bardzo elastyczny w kwestii białych znaków. Jeżeli użyjemy spacji, tabulatorów, czy enterów przeglądarka je zignoruje i wyświetli ten sam tekst. Przykładowy kod może wyglądać tak:

```HTML
<pierwszy-znacznik>
    <drugi-znacznik>
        element
    </drugi-znacznik>
</pierwszy-znacznik>
```

lub tak:

```HTML
<pierwszy-znacznik><drugi-znacznik>element</drugi-znacznik></pierwszy-znacznik>
```

Oba zostaną zinterpretowane przez przeglądarkę dokładnie w ten sam sposób. Zalecamy jednak stosować odpowiednie formatowanie naszego kodu w postaci znaków nowej linii oraz tabulatorów, gdyż poprawia to jego czytelność i powoduje, że w przyszłości będziemy mieli mniej problemów z jego zrozumieniem.

Zobacz na kod **1.2**

DOCTYPE

Ponieważ istnieje kilka wersji HTML, każda strona powinna zaczynać się od deklaracji `DOCTYPE`. Dzięki temu powiemy przeglądarce która wersja jest obecnie wyświetlana. Jest to zawsze pierwszy element na na stronie.

```html
<!DOCTYPE html>
```

HTML

Będziemy używać HTML w 5 wersji. Jest to obecnie najnowsza wersja i stale rozwijana.

Głowa i ciało dokumentu

Cała strona internetowa to tak naprawdę tag `<html>` w skład którego wchodzą wszystkie pozostałe tagi. Możemy wyróżnić **dwie główne części**: `<head>`, która zawiera informacje o stronie oraz `<body>`, która zawiera strukturę naszej strony - czyli ciało.

```html
<html>
    <head></head>
    <body></body>
</html>
```

Te 3 tagi są unikalne - nie mogą się powtórzyć nigdzie więcej na stronie.

Domyślne kodowanie

Strony internetowe początkowo pisało się tylko w języku angielskim. Jednak wraz z rozpowszechnieniem się dostępu to globalnej sieci strony internetowe zaczęto tworzyć również w innych językach.

Problem zaczął się pojawiać w momencie, gdy język posiadał nieangielskie znaki, np. `ą` czy `ł`. Ograniczona ilość dostępnego miejsca mogącego przechować dodatkowe znaki spowodowała wymuszenie zastosowania różnych rodzajów kodowania strony. I tak np. strony pisane w języku rosyjskim miały inne kodowanie niż np. strony pisane w języku polskim.

Problem zaczynał się nasilać w momencie, gdy na jednej stronie chcieliśmy skorzystać z kilku języków. W tym celu stworzono specjalny sposób kodowania znaków nazywany `utf-8`.

Za jego pomocą możemy przechować praktycznie każdy znak powszechnie używany na stronach.

Aby poinformować przeglądarkę, że będziemy używać `utf-8` musimy na początku sekcji `<head>` dodać fragment:

```HTML
<meta charset="utf-8" />
```

Zauważmy, że jest to jednoczłonowy tag. Od teraz nie powinniśmy już mieć problemów z wyświetlaniem polskich znaków.

Tytuł strony

Tytuł strony umieszczamy wewnątrz tagu `<title>`. Fragment ten musi znajdować się wewnątrz sekcji `<head>`. Element `<title>` może się pojawić tylko raz na stronie:

```HTML
<title>Moja pierwsza strona</title>
```

Tytuł strony będzie najważniejszą informacją o naszej stronie. Będzie pojawiał się na karcie z naszą stroną internetową w przeglądarce, w zakładkach czy w wynikach wyszukiwania. Pamiętaj, żeby faktycznie reprezentował on nasza stronę.

Jak tworzyć odpowiedni tytuł strony dowiesz się z rozdziału na temat SEO.

Komentarze

Czasami potrzebujemy skomentować część strony, albo tymczasowo ukryć fragment kodu. Możemy tego dokonać za pomocą komentarzy. Tworzymy je zaczynając od `<!--` a kończąc na `-->`. Zobaczmy przykład:

```HTML
<body>
    <!-- tutaj muszę coś dodać -->
    Moja strona internetowa
</body>
```

Wszystko co znajdzie się wewnątrz komentarza nie będzie wyświetlane na stronie, ale cały czas będzie dostępny w źródle strony.

Podsumowanie

Postarajmy się teraz zebrać wszystkie poprzednie właściwości kodu HTML5 i stwórzmy podstawowy szablon strony internetowej, który będziemy używać w dalszej nauce:

HTML

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>Moja pierwsza strona internetowa!</title>
    </head>
    <body>
        Ale to jest proste!
    </body>
</html>
```


ZADANIE

Stwórz swoją pierwszą stronę internetową składającą się z tytułu oraz jednego zdania. Plik zapisz pod nazwą `index.html`, a następnie spróbuj otworzyć go w przeglądarce.

Początki z językiem HTML

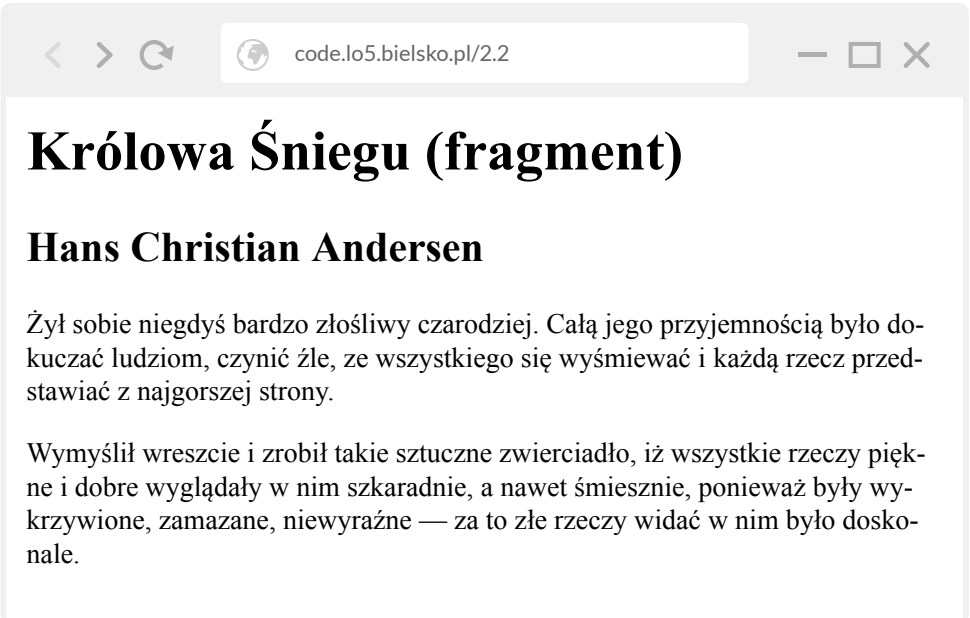
3

HTML

```
<h1>Królowa Śniegu (fragment)</h1>
<h2>Hans Christian Andersen</h2>
<p>
    Żył sobie niegdyś bardzo złośliwy czarodziej. Całą jego
przyjemnością było dokuczać ludziom, czynić źle, ze wszystkiego się
wyśmiewać i każdą rzecz przedstawiać z najgorszej strony.
</p>
<p>
    Wymyślił wreszcie i zrobił takie sztuczne zwierciadło, iż
wszystkie rzeczy piękne i dobre wyglądały w nim szkaradnie,   a nawet
śmiesznie, ponieważ były wykrzywione, zamazane, niewyraźne — za to złe
rzeczy widać w nim było doskonale.
</p>
```

W przedstawionym fragmencie kodu znajdują się dwa główne nagłówki oznaczone znacznikami `<h1>`. Do podtytułów użyto tagów `<h2>`, a do zwykłej treści `<p>`.

Zobacz jak powyższy przykład będzie wyglądał w oknie przeglądarki:



Akapity `<p>`

```HTML
<p>Paragraf, akapit, czyli jak zwał, tak zwał</p>
```

Pierwszym znacznikiem, który poznasz, jest tag oznaczający fragment tekstu. Akapity następujące po sobie są rozdzielone jedną linijką przerwy. Pisząc strony internetowe najwygodniej jest każdy zwykły fragment treści pisać przy użyciu właśnie tego znacznika.

Zobacz na kod **2.1**

Nagłówki `<h1>` do `<h6>`

```HTML
<h1>Tytuł</h1>
<h2>Podtytuł</h2>
```

Oprócz zwykłej treści w swojej stronie na pewno będziesz używał nagłówków do akapitów. Język HTML daje nam **6 rozmiarów** nagłówków: od `<h1>` do `<h6>`. Wielkość wiąże się z pozycją, dlatego tak, jak w powyższym przykładzie, im większy jest „tytuł” tym jest ważniejszy. Podobnie jak przy znaczniku `<p>`, dwa kolejne tytuły rozdzielone są jedną linijką przerwy.

Zobacz na kod **2.2**

O znaczniku `<sup>`, czyli indeksie górnym, dowiesz się więcej w następnym paragrafie.

Podstawowe formatowanie

```HTML
<h2>I zasada termodynamiki</h2>
<p>
    <strong>Zmiana energii wewnętrznej</strong> ciała lub układu jest
równa sumie algebraicznej <strong>ciepła</strong> wymienionego między
ciałem (<em>układem</em>), a otoczeniem i <strong>pracy</strong>
wykonanej przez to ciało (<em>układ</em>) nad otoczeniem lub odwrotnie.
</p>
<p>
```

```
    <sub>&#916;</sub>E<sub>w</sub> = Q + W (&#954;)
</p>
```

W tym przykładzie oprócz widocznych nagłówków i tekstu widzimy tekst ważny (pogrubione wyrazy, np. "pracy", "ciepła") oznaczony znacznikiem `<strong>`, tekst mniej ważny (pisany kursywą, czyli "wykładnik adiabaty") oznaczony tagiem `<em>` oraz indeksy przy wzorach - dolne `<sub>` i górne `<sup>`. Powyższy kod w przeglądarce będzie się prezentował następująco:

CIEKAWOSTKA

Zapewne już zauważyłeś/aś w kodzie `&#916;` i `&#954;`. Są to encje, czyli znaki specjalne, które w języku HTML mają swoje własne unikalne kody. Pierwsza z użytych encji to grecka litera delta (Δ), a druga to kappa. (κ)

Tekst ważny `<strong>` i zaakcentowany `<em>`

```
<strong>Tekst ważny</strong>
<em>Tekst zaakcentowany</em>
```

HTML

Zobacz na kod **2.3** 

Oprócz tego, że pierwsza linijka powyższego tekstu będzie wytłuszczona, a druga napisana kursywą, to użycie tych znaczników niesie informację o jego ważności. Jest to nie tylko formatowanie samej czcionki (wizualny efekt), ale zwłaszcza wyróżnienie tych informacji na poziomie interpretowania kodu. Możemy przyjąć, że znacznik `<strong>` służy do tekstu bardzo ważnego, a znacznik `<em>` do tekstu mniej ważnego.

POMYŚL

Nagłówki `<h1>` - `<h6>` są jedną z najważniejszych części tekstu, dlatego użycie wewnątrz nich znacznika `<strong>` nie ma sensu. Możemy jednak zaakcentować część nagłówka poprzez tag `<em>`. Zachęcamy Cię do eksperymentowania ze znacznikami w podobny sposób.

Łamanie tekstu

Poznaliśmy już kilka tagów HTML, postarajmy się przeanalizować poniższy fragment kodu:

HTML

```
<p>Pierwsza linia<br />Druga linia</p>
<p>Trzecia linia</p>
<p>Czwarta linia</p>
<p>Piąta linia<hr />Szósta linia</p>
```

Znak nowej linii `<br>`

```HTML
<p>Pierwsza linia<br />Druga linia</p>
```

Wiemy już, że białe znaki użyte w kodzie nie pojawią się na wyświetlonej stronie. Tekst możemy rozdzielać poprzez umieszczanie go w kilku akapitach, ale co zrobić, jeżeli chcemy w dowolnym miejscu przejść do nowej linijki? Musimy użyć znaku końca linii, czyli `<br />`. Jest to jeden z niewielu znaczników, których nie zamykamy.

Inny przykład: snippets/2/1

Pozioma linia `<hr>`

```HTML
<p>
   Tekst przed linią
   <hr />
   Tekst po linii
</p>
```

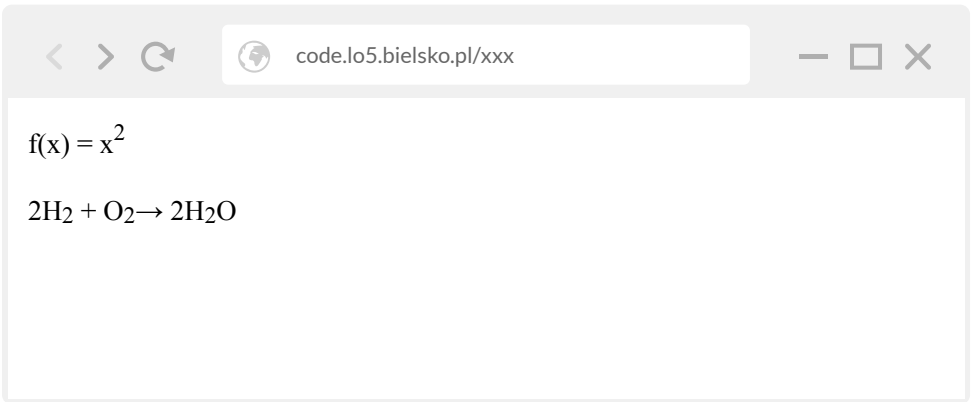
Tego znacznika używamy w celu rozdzielenia tekstu poziomą linią. Jest on drugim wyjątkiem, którego użycie nie wiąże się z jego zamknięciem.

Indeksy `<sup>` i `<sub>`

```
f(x) = x<sup>2</sup>
<br /><br />
2H<sub>2</sub> + O<sub>2</sub>→ 2H<sub>2</sub>O
```

HTML

Oznaczają kolejno indeks górny i indeks dolny. Powyższy kod zostałby wyświetlony na stronie internetowej jako:



del i ins

```
<h2>Ciepło <ins>molowe</ins> <del>właściwe</del></h2>
<p>
Liczbowo jest równe ilości ciepła potrzebnego do ogrzania <ins>jednego
mola gazu</ins> <del>jednostki masy (1kg)</del> o 1&#176;C lub 1K.<br
/>Ciepło molowe dla cząsteczek mających jednakową ilość atomów jest
takie samo.
</p><hr / size="10px" noshade="true" width="300px">

<h2>Ciepło przemiany</h2>
<p>
Jest ono równe liczbowo ilości ciepła potrzebnego (otrzymanego) do
zmiany stanu skupienia 1 kg substancji w <abbr>temp.</abbr> przemiany
pod ciśnieniem atmosferycznym.
</p><hr / size="10px" noshade="true" width="300px" align="left">

<h2>Prawo Fouriera</h2>
<p>
<blockquotes>Gęstość przewodzonego strumienia ciepła jest wprost
```

HTML

```
proporcjonalna do <em>gradientu temperatury</em>.</blockquotes>
</p>
```

W pierwszym akapicie, aby przejść do następnej linijki, użyliśmy znacznika `<br />`. Aby oddzielić od siebie kolejne części tekstu wstawialiśmy poziomą linię za pomocą `<hr />`. Tekst przekreślony dotyczący ciepła właściwego wprowadziliśmy przez `<del>`, a nowy o ciepłu molowym (podkreślony) przez `<ins>`. Dodatkowo skrót "temp." wyróżniliśmy przez `<abbr>`, a cytowaną treść prawa Fouriera przez `<blockquotes>`.

Długi cytat `<blockquote>`

```HTML
<blockquote cite="http://wolnelektury.pl/katalog/lektura/
ballady-i-romanse-pani-twardowska.html">
Jedzą, piją, lulki palą,<br />
Tańce, hulanki, swawola […]
</blockquote>
```

Znacznik ten oznacza długi cytat. Wszystkie linijki tekstu w nim zawarte będą zaczynały się od tabulatora. Z reguły używamy tego znacznika, kiedy chcemy przytoczyć czyjąś wypowiedź lub gdy cytujemy dany utwór. Wewnątrz niego można umieszczać tylko elementy blokowe, o których więcej dowiesz się w następnych rozdziałach.

Atrybut `cite`

Dodatkową informacją jest adres URL strony internetowej, z której tekst jest cytowany. Możemy podać go jako wartość dla argumentu `cite`.

Skrót `<abbr>`

```HTML
<p>Ogromne podziękowania dla <abbr>mgr</abbr> Włodzimierza Raczka z
cierpliwość.</p>
```

Dzięki temu znacznikowi możemy wyróżnić w kodzie użyte skrócone formy wyrazów. Stosowanie go dostarcza dodatkowych informacji przeglądarkom i innym programom.

Tekst skasowany `<del>` i wstawiony `<ins>`

```HTML
<p>Bielsko-Biała to miasto w województwie <del>bielskim</del> śląskim
<ins>na prawach powiatu</ins>.</p>
```

Pierwszy z nich wyróżnia tekst, który został usunięty – zostanie wyświetlony przekreśloną czcionką. Drugiego z kolei użyjemy, gdy do starego tekstu dodamy nowy – zostanie on podkreślony.

Inny przykład: snippets/2/9

Wstęp do CSS

4

Arkusze CSS

Do tej pory dowiedzieliśmy się, z jakich podstawowych elementów zbudowane są strony internetowe. Używanie znaczników jest bardzo ważne w zarządzaniu treścią, jednak nie daje dużych możliwości formatowania. Dlatego by swobodnie manipulować szatą graficzną, będziemy korzystać z drugiego narzędzia po języku HTML – stylów CSS.

CSS - Cascading Style Sheets (z ang. kaskadowe arkusze stylów) to specjalny język utworzony jedynie w celu ułatwienia i uelastycznienia możliwości formatowania elementów. Pozwala on edytować wygląd i sposób wyświetlania danych treści.

Taki podział ułatwia pracę webmasterom. Treść strony zawiera się w plikach HTML, a jej wizualna część w arkuszu CSS. Z tego powodu nie omawiamy tych struktur osobno, ponieważ dopiero razem dają oczekiwane efekty. Style CSS stały się integralną częścią języka HTML.

Struktura

Cała praca przy edytowaniu elementów za pomocą styli CSS polega na nadawaniu odpowiednich wartości wybranym cechom.

Styl lokalny

HTML

```
<p style="color: brown; font-style: italic">
Prawie wszystkie wyrazy ciągu – to wszystkie z wyjątkiem
<strong>skończonej</strong> ich liczby.
</p>
<p style="color: green">
Suma dwóch ciągów zbieżnych do 0 jest zbieżna do 0.
</p>
<p style="color: gray">
Iloczyn dwóch ciągów zbieżnych do 0 jest zbieżny do 0.
</p>
```

Możemy odnosić się do wybranych elementów na dwa sposoby. Pierwszy z nich to styl lokalny, czyli zastosowany wewnątrz pojedynczego znacznika. Spójrz na przykładowy fragment kodu:

```HTML
<p align="right" style="color: blue">To jest niebieski akapit.</p>
```

Oprócz wyrównania tekstu do prawej krawędzi używając parametru `align`, nadaliśmy mu kolor niebieski za pomocą stylów CSS. Wystarczy poinformować przeglądarkę o tym, że nagle zaczynamy pisać w języku CSS poprzez `style=` i w podwójnych cudzysłowach podać komendy, które rozdzielamy za pomocą średników `;`.

[obrazek, coś w stylu: style=”cechaA: wartośćA; cechaB: wartośćB”]

Jeżeli wewnątrz znacznika, w którym użyłeś styli CSS, są inne znaczniki, to formatowanie jest zastosowane do każdego z nich.

Zewnętrzny arkusz styli

Wyobraź sobie teraz, że masz do napisania stronę internetową, której treść będzie się składać z kilkudziesięciu części o różnych kolorach, wielkościach, czcionkach. Używanie osobnych styli do każdego znacznika byłoby zbyt czasochłonne, dlatego możemy stworzyć osobny plik i w nim pisać kod, który sformatuje kilka elementów jednocześnie.

Zewnętrzne arkusze stylów mają rozszerzenie `.css`. Przeglądarka użyje je, jeśli w sekcji `head` zamieścimy następujący kod:

```HTML
<link rel="stylesheet" href="style.css" />
```

Ten tag informuje o pliku `style.css`, znajdującym się w tym samym katalogu,

```HTML
<link rel="stylesheet" href="css/akapity.css" />
```

natomiast powyższy odwołuje się do `akapity.css` w folderze `css`.

Selektory

Pozostaje teraz kwestia uporządkowanego formatowania kilku elementów jednocześnie - to jedna z kluczowych umiejętności przy pisaniu własnej strony internetowej.

Selektory typu i potomka

HTML

```
<h1>
  Twierdzenie Bezout
</h1>
<p>
  Wielomian W(x) jest podzielny przez dwumian x-a wtedy i tylko wtedy,
gdy <strong>W(a)=0</strong>.
</p>

<h2>
  Inne twierdzenia:
</h2>
<p>
  Jeżeli pierwiastkiem wielomianu o współczynnikach całkowitych
i wyrazie wolnym a<sub>0</sub> jest x<sub>0</sub>, to
<strong>x<sub>0</sub>|a<sub>0</sub></strong>.
</p>
<p>
  Jeżeli suma współczynników wielomianu jest równa 0, to pierwiastkiem
tego wielomianu jest liczba <strong>x<sub>0</sub>=1</strong>.
</p>
```

CSS

```
h1 {
    font-family: Georgia, serif;
}

h2 {
    font-family: Verdana, sans-serif;
}

p {
    color: brown;
    font-style: italic;
}

p strong {
     color: red;
}
```

W powyższym przykładzie kod napisany w języku HTML powinien być dla Ciebie zrozumiały. Spójrz jednak na te kilka linijek kodu CSS. Napisanie

```
p {
    color: brown;
    font-style: italic;
}
```
CSS

spowodowało, że każdemu paragrafowi `<p>` nadano brązowy kolor i styl czcionki `italic` (pochylony). W tym przykładzie `p` jest selektorem. Pozostałe selektory to `h1` i `h2`. Są to najprostsze selektory typu. Z kolei fragment

```
p strong {
    color: red;
}
```
CSS

nadał elementom wewnątrz znaczników `<strong>`, które są wewnątrz `<p>`, czerwony kolor. W tym przypadku `p strong` jest selektorem potomka, ponieważ elementy `<strong>` są niżej w hierarchii niż elementy `<p>`.

UWAGA!

Jeżeli korzystasz z zewnętrznych arkuszy CSS, to po każdym nadaniu cechy musi się znaleźć średnik - nawet, jeżeli jest to tylko jedna cecha dla danego selektora. Inaczej było przy stylowaniu lokalnym, gdzie średników używaliśmy tylko przy kilku cechach.

Selektor uniwersalny i grupowanie selektorów

```
<h1>
  Twierdzenie o pierwiastkach całkowitych
</h1>
<p>
Jeżeli wielomian W(x) ma pierwiastek całkowity, to jest on
```
HTML

```
<strong>dzielnikiem wyrazu wolnego</strong>.
</p>
<h2>
  Inne twierdzenie o wielomianie o współczynnikach całkowitych:
</h2>
<p>
  Jeżeli współczynniki wielomianu W(x) są całkowite oraz różne od
siebie liczby a i b też są całkowite, to
<strong>a-b|W(a)-W(b)</strong>.
</p>
```

CSS

```
* {
    font-family: Georgia, serif;
}

p, h2 {
    font-style: italic;
}

strong {
    color: green;
}
```

W powyższym przykładzie zastosowaliśmy selektor uniwersalny `*`, który wszystkim elementom nadał wybraną czcionkę. Z kolei selektor `p, h2` przypisał pochyloną czcionkę zarówno elementom `<p>` jak i `<h2>`, dlatego właśnie są one rozdzielone przecinkiem - są pogrupowane.

Selektory dziecka i braci

HTML

```
<h1 align="center">
  Ciągi
</h1>
<p>
  <strong>Ciąg</strong> - funkcja, której argumentami jest zbiór liczb
naturalnych <ins>lub jego podzbiór</ins>.
</p>
<p>
  <strong>Ciąg <ins>liczbowy</ins></strong> - jego wartociami są
liczby.
</p>
<p>
```

```
  <strong>Ciąg <del>nie</del>skończony</strong> - okrelony na
<del>nie</del>skończonym podzbiorze liczb <ins>naturalnych</ins>.
</p>
```

```
* {
    font-family: Verdana, serif;
}

p > ins {
    font-style: italic;
}

strong ~ ins {
    color: red;
}

del + ins {
    color: green;
}
```
CSS

Najpierw weźmy pod uwagę fragment:

```
p > ins {
    font-style: italic;
}
```
CSS

`p > ins` jest selektorem dziecka. Wskazuje on na te elementy `<ins>`, które znajdują się bezpośrednio w `<p>`. Dlatego słowo "liczbowy", które jest w `<strong>`, który jest w `<p>`, nie zostało napisane kursywą. Właśnie ta właściwość odróżnia ten selektor od selektora dziedziczenia.

Spójrzmy teraz na:

```
del + ins {
    color: green;
}
```
CSS

W tym fragmencie został użyty selektor braci, czyli `del + ins`. Oznacza on, że każdy element `<ins>`, który znajduje się bezpośrednio po elemencie `<del>`, będzie miał zielony kolor. Gdyby pomiędzy nimi był jakiś inny element niż zwykły tekst, ten selektor nie zadziałałby. Dlatego tylko słowo "naturalnych" zostanie pokolorowane na zielono.

Wcześniej jednak mamy:

```css
strong ~ ins {
    color: red;
}
```

CSS

Został tu zastosowany ogólny selektor braci. W odróżnieniu od poprzedniego selektora braci, tutaj mogą znajdować się inne elementy pomiędzy `<strong>` i `<ins>`.

Po interpretacji kodu powinieneś/powinnaś zauważyć, że oba fragmenty "lub jego podzbiór" i "naturalnych" powinny być czerwone. Byłoby tak, gdyby ogólny selektor braci znajdował się przed selektorem braci. Na tym przykładzie wyraźnie widzisz, że kolejność w języku CSS ma duże znaczenie.

Klasy

Samo stosowanie selektorów daje ogromne możliwości przy edytowaniu wyglądu strony. Dodatkowym ułatwieniem jest wprowadzenie klas.

```html
<h1>
  Twierdzenie Sinusów
</h1>
<p class="tresc">
  W dowolnym trójkącie stosunek długości boku do sinusa kąta
przewległego temu bokowi jest wielkością stałą dla tego trójkąta.
</p>
<p align="center" class="wzor">
  a : sin&#945; = b : sin&#946; = c : sin&#947;
</p>
<h1>
  Twierdzenie Cosinusów
</h1>
<p class="tresc">
  W dowolnym trójkącie kwadrat długości boku jest równy sumie
kwadratów długości pozostałych boków zmniejszonej o podwojony iloczyn
```

HTML

```
długości tych boków i cosinusa kąta między tymi bokami.
</p>
<p align="center" class="wzor">
  a<sup>2</sup> = b<sup>2</sup> + c<sup>2</sup> - 2ab cos&#945;
</p>
```

CSS

```
h1 {
  font-style: italic;
}

.tresc {
  color: brown;
}

.wzor {
  color: green;
}
```


CIEKAWOSTKA

W podanym przykładzie zastosowano kolejne encje, czyli `&#945;`, `&#946;`, `&#947;`, które oznaczają kolejne greckie litery: alfę, betę i gammę.

W kodzie HTML mamy dwa rodzaje paragrafów – treści twierdzeń i wzory. Aby móc zastosować do nich różne style, nadaliśmy im klasy odpowiednio `tresc` i `wzor`. Polegało to na nadaniu atrybutowi `class` wartości, która będzie wybraną przez nas nazwą.

UWAGA!

Nazwa klasy może być jednym wyrazem składającym się tylko z małych i dużych liter oraz z cyfr, myślników i podkreślników. W dodatku nie może zaczynać się od cyfr i myślników, a stosowanie polskich liter jest niewskazane.

W powyższym przykładzie treści twierdzeń (elementy o klasie `tresc`) mają brązowy kolor, a wzory (elementy o klasie `wzor`) są pisane zieloną czcionką. Jest tak, ponieważ wpisaliśmy w arkuszu CSS `.tresc` i `.wzor`. Jak widzisz, jeżeli przed nazwą klasy postawimy kropkę, to stylowanie będzie zastosowane do wszystkich elementów należących do danej klasy.

Zmodyfikujmy lekko poprzedni przykład:

HTML

```
<h1 class="twierdzenie">
  Twierdzenie Sinusów
</h1>
<p class="twierdzenie">
  W dowolnym trójkącie stosunek długości boku do sinusa kąta
przewległego temu bokowi jest wielkością stałą dla tego trójkąta.
</p>
<p align="center" class="wzor">
  a : sin&#945; = b : sin&#946; = c : sin&#947;
</p>
<h1 class="twierdzenie">
  Twierdzenie Cosinusów
</h1>
<p class="twierdzenie">
  W dowolnym trójkącie kwadrat długości boku jest równy sumie
kwadratów długości pozostałych boków zmniejszonej o podwojony iloczyn
długości tych boków i cosinusa kąta między tymi bokami.
</p>
<p align="center" class="wzor">
  a<sup>2</sup> = b<sup>2</sup> + c<sup>2</sup> - 2ab cos&#945;
</p>
```

CSS

```
.twierdzenie {
  color: brown;
}

p.twierdzenie {
  font-style: italic;
}

.wzor {
  color: green;
}
```

Teraz zarówno nagłówki `<h1>` jak i akapity `<p>` mają klasę `twierdzenie`, dlatego wszystkie te elementy na stronie miałyby brązowy kolor. Ale

```css
p.twierdzenie {
  font-style: italic;
}
```

oznacza, że tylko paragrafy `<p>` (bez nagłówków `<h1>`) mają mieć pochyloną czcionkę. Wystarczyło przed kropką wpisać wybrany selektor.

Listy i podzbiory klas

```html
<h1 align="center">
  Przykładowe zadania z trygonometrii:
</h1>
<p class="numer">
  zad.1.
</p>
<p class="tresc cos">
  Obliczyć kąty trójkąta o bokach a=5, b=7 i c=9.
</p>
<p class="numer">
  zad.2.
</p>
<p class="tresc cos">
  Rozwiązać trójkąt o bokach długości 3 i 4 i kącie pomiędzy nimi
równym 90 stopni.
</p>
<p class="numer">
  zad.3.
</p>
<p class="tresc sin">
  Udowodnij, że stosunek długości boku do sinusa kąta naprzeciwko
niego jest równy długości promienia okręgu opisanego na tym trójkącie.
</p>
<p class="numer">
  zad.4.
</p>
<p class="tresc sin cos">
  Na punkt działają dwie siły: 10N i 15N, a kąt, jaki tworzą ze sobą,
jest równy 105 stopni. Obliczyć wielkość wypadkowej tych sił.
</p>
```

```css
p.numer {
  font-family: Verdana;
}

p.tresc.sin {
  font-style: italic;
}

p.tresc.cos {
  color: blue;
}

p.tresc.sin.cos {
  line-height: 15px;
}
```

Jeżeli chcemy do jednego elementu przypisać kilka klas, to wystarczy podać je w cudzysłowach i porozdzielać spacjami (lub innymi białymi znakami). Widzimy to wyraźnie w `<p class="tresc sin cos">`. Wiesz już, co oznacza fragment:

```css
 p.numer {
  font-family: Verdana;
}
```

Spójrz teraz na:

```css
p.tresc.cos {
  color: blue;
}
```

Jego wykonanie skutkuje nadaniem koloru niebieskiego tym elementom, które jednocześnie mają znacznik `<p>` i obie klasy `tresc` i `cos`. Wymogi te w tym przykładzie spełnia treść zadania pierwszego i czwartego.

Z kolei fragment

```css
p.tresc.sin.cos {
  line-height: 15px;
}
```

zmieni wysokość linii tylko dla treści zadania czwartego, ponieważ tylko ona ma znacznik `<p>` i wszystkie klasy `tresc`, `sin` i `cos`.

Identyfikatory

Jeżeli wiemy, że dana klasa pojawi się w kodzie tylko raz, to nie ma sensu jej stosować. Wygodniejsze jest wtedy zastosowanie identyfikatora.

```html
<h1 align="center" id="naglowek">
  Wybrane zadania z <ins>równań trygonometrycznych</ins>:
</h1>
<p class="numer">
  zad.1.
</p>
<p class="tresc">
  Rozwiązać równanie: tg<sup>2</sup> x = 3.
</p>
<p class="numer">
  zad.2.
</p>
<p class="tresc">
  Rozwiązać równanie: ctg x = 1.
</p>
<p class="numer">
  zad.3.
</p>
<p class="tresc">
  Rozwiązać równanie: cos<sup>2</sup>x = -cos x.
</p>
```

```css
#naglowek {
  font-style: italic;
}

#naglowek ins {
  color: blue;
}
```

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

Jak widzisz, identyfikator wprowadzamy w HTMLu za pomocą `id=`. Identycznie jak przy klasach, wartość w cudzysłowach jest nazwą identyfikatora.

W arkuszu CSS jedyną różnicą jest stosowanie `#` zamiast `.`.

Zwróć uwagę na to, że jeżeli chcemy odnieść się tylko do elementów w danym znaczniku, który jest w innym znaczniku o znanej klasie lub identyfikatorze, to nazwę tego „najgłębszego" znacznika podajemy na końcu. Gdybyśmy do powyższego kodu CSS dodali

CSS
```
p.tresc sup {
  color: red;
}
```

to tylko wykładniki przy potęgach w treściach zadań zostałyby pokolorowane na czerwono.

Pseudoelementy

Wiemy już, że strony internetowe składają się z elementów, którym możemy nadawać klasy i identyfikatory. Są jednak takie obiekty, które w żaden sposób nie zostały przez nas wyróżnione, a które również możemy osobno edytować.

Pseudoelementami są szczególne części innych elementów, np. pierwsze linie i litery akapitów. Wyróżnienie ich osobno, chociażby poprzez wstawianie ich w osobnych akapitach z wyróżnioną klasą, byłoby strasznie czasochłonne. Właśnie dlatego język CSS pozwala nam się do nich bezpośrednio odnieść.

HTML
```
<h3>
  Potęgi o wykładniku naturalnym
</h3>
<p>
  Założenie: a jest rzeczywiste, n i m są naturalne<br />
```

```
  # a<sup>n</sup> * a<sup>m</sup> = a<sup>n+m</sup><br />
  # a<sup>n</sup> / a<sup>m</sup> = a<sup>n-m</sup><br />
  # (a<sup>n</sup>)<sup>m</sup> = a<sup>n*m</sup><br />
  # a<sup>n</sup> * b<sup>n</sup> = (ab)<sup>n</sup><br />
  # a<sup>n</sup> / b<sup>n</sup> = (a/b)<sup>n</sup>, b różne od 0
</p>
```

```
h3 {
  color: brown;
}

p {
  line-height: 30px;
}

p:first-line {
  color: grey;
}

p:first-letter {
  font-size: 150%;
}
```
CSS

W tym przykładzie zastosowaliśmy pseudoelementy pierwszej linii i pierwszej litery. Fragment

```
p:first-line {
  color: grey;
}
```
CSS

pierwszej linii każdego akapitu nadaje kolor szary, a fragment

```
p:first-letter {
  font-size: 150%;
}
```
CSS

pierwszą literę każdego akapitu zwiększą do 150%.

Spójrz teraz na przykład:

HTML

```
<h2>
  Rysowanie wykresów funkcji
</h2>
<p>
  f(x) = x<sup>3</sup> - 6x<sup>2</sup> + 9x -4
</p>
<p>
  g(x) = (x<sup>2</sup> - 1)(x<sup>2</sup> - 4)
</p>
<p>
  h(x) = -x<sup>3</sup> + 3x<sup>2</sup> + 9x - 11
</p>
<h2>Asymptoty funkcji</h2>
<p>
  f(x) = x + 1/x
</p>
<p>
  g(x) = x<sup>3</sup> / (3(x<sup>2</sup>-4)
</p>
```

CSS

```
h2 {
  color: brown;
}

h2:after {
  content: "-zadania:";
  color: grey;
}

p:first-letter {
  font-size: 150%;
}

p:before {
  content: "Narysuj";
  color: grey;
}

p:after {
  content: ".";
  color: grey;
}
```

Fragment

```css
h2:after {
  content: "-zadania:";
  color: grey;
}
```

oznacza, że po każdym nagłówku `<h2>` ma się znaleźć tekst „-zadania:”, który ma mieć szary kolor. Bardzo podobny efekt otrzymaliśmy za pomocą `p:after`, dzięki któremu po każdym akapicie wstawiliśmy kropkę.

Z kolei polecenie

```css
p:before {
  content: "Narysuj";
  color: grey;
}
```

przed każdym akapitem wypisze napis „Narysuj”, również szarą czcionką.

Zwróć uwagę na to, że fragment

```css
 p:first-letter {
  font-size: 150%;
}
```

powiększył litery „N”, a nie litery „f”, „g”, „h”. Możemy stąd wnioskować, że tekst „Narysuj” stał się częścią każdego paragrafu. W tej sytuacji nawet zmiana kolejności fragmentów

```css
p:first-letter {
  font-size: 150%;
}

p:before {
  content: "Narysuj ";
  color: grey;
}
```

nic nie zmieni, ponieważ „Narysuj” już zawsze będzie pierwszym słowem paragrafów `<p>`, więc ich pierwszą literą zawsze będzie „N”.

Przyjrzyjmy się teraz formule `content:`. W powyższych przykładach pozwoliła ona wstawić nam w odpowiednie miejsca tekst podany w cudzysłowach, po czym sformatować go według podanych reguł.

Oprócz tekstu możemy również wstawiać obrazy. Dokładniej o nich dowiesz się w dalszej części tej książki. Jeżeli jednak chodzi o wstawianie ich poprzez `content`, to można to zrobić tak, jak poniżej:

```css
p:after {
  content: url(ścieżka_do_obrazka/obrazek.gif);
  /*dalsze stylowanie*/
}
```

Wystarczy po słowie `content:` wpisać `url`, a w nawiasach okrągłych ścieżkę dostępu do obrazka.

Krótkie cytaty `<q>` i cudzysłowy

Cudzysłowy należy wstawiać przed i po danym tekście, dlatego idealnie nadaje się do tego zadania użycie pseudoelementów z `before` i `after`. Znacznikiem do krótkich cytatów w języku HTML jest `<q>`. Fragment kodu CSS

```css
q {
  quotes:   ' " '   ' " '   ' < '   ' > ';
}
```

wskazuje, jakich cudzysłowów przeglądarka ma używać dla znacznika `<q>`. Pierwsza para to dwa cudzysłowy – otwierający i zamykający. Druga para jest podana na wypadek, gdyby wewnątrz zwykłego cytatu był zagnieżdżony kolejny. Jeżeli chcemy, to możemy wskazać rodzaj cudzysłowów dla kilku następnych zagnieżdżeń. Jeżeli tego nie zrobimy, to np. tutaj dla trzeciego poziomu „cytatu w cytacie” zostanie użyta ostatnia z podanych par. Zobacz, jak to zadziała w praktyce:

```html
<p><q>
  Więcej jest rzeczy na ziemi i w niebie,
niż się śniło waszym filozofom.
  </q>
  Hamlet
</p>
```

HTML

```css
q {
  quotes: '"' '"';
}

q:before {
  content: open-quote;
}

q:after {
  content: close-quote;
}
```

CSS

Dwa ostatnie polecenia w kodzie CSS powinny być w większości jasne. Pierwszy z nich przed cytatem wstawia cudzysłów otwierający, a drugi z nich po cytacie wstawia cudzysłów zamykający.

Pseudoklasy

Po elementach były pseudoelementy, więc po klasach muszą być pseudoklasy.

Wszystkie poznane dotąd selektory bazują na strukturze HTML. To od niej zależy, jakie znaczniki mają dane elementy, jakie mają one klasy, identyfikatory, jak głęboko są zagnieżdżone. Teraz poznasz narzędzie, które po części wyłamuje się z tego schematu, ponieważ będziemy mogli od tej pory brać pod uwagę interakcję użytkownika, język dokumentu oraz strukturę strony.

Wiele pseudoklas wiąże się z konkretnymi elementami, np. z hiperłączami. Dlatego w tym rozdziale nie poznasz wszystkich, a jedynie te najważniejsze i najbardziej uniwersalne pseudoklasy.

Pseudoklasa `:lang()` i reguła `@font-face`

HTML

```
<h3>
  Wybrane cytaty z Hamleta
</h3>
<p lang="en">
  Though this be madness, yet there's method in't.
</p>
<p lang="pl">
  Choć to szaleństwo, lecz jest w nim metoda.
</p><br />
<p lang="en">
  More matter with less art.
</p>
<p lang="pl">
  Więcej treści, a mniej sztuki.
</p><br />
<p lang="en">
  To be, or not to be, that is the question.
</p>
<p lang="pl">
  Być albo nie być, oto jest pytanie.
</p>
```

CSS

```
@font-face {
   font-family: Vczcionka;
   src: url('fonts/V.otf');
}

p:lang(en) {
   font-family: Vczcionka;
   font-style: italic;
}
```

W powyższym przykładzie tekst jest napisany w dwóch różnych językach, co zostało zaznaczone poprzez atrybut `lang`. Wiedząc to, możemy stylować osobno akapity z polskim i angielskim tekstem. Tutaj fragmentom angielskim nadaliśmy czcionkę poprzez

CSS

```
p:lang(en) {
   font-family: Vczcionka;
   font-style: italic;
}
```

Zauważ jednak, że wcześniej jest

```css
@font-face {
   font-family: Vczcionka;
   src: url('fonts/V.otf');
}
```

Użyliśmy reguły `@font-face`, dzięki której utworzyliśmy własną czcionkę `Vczionka`. Najważniejsze w tym procesie jest określenie nazwy naszej czcionki w `font-family:` oraz podanie adresu do niej względem arkusza w `src: url( /*adres*/ )`. My użylibyśmy w przykładzie czcionki z pliku `V` z folderu `fonts`.

Pseudoklasa `:empty` i `:not()`

```html
<h2>
  Okrąg wpisany w czworokąt
</h2>
<p>
</p>
<h4>
  Oznaczenia:
</h4>
<p class="oznaczenia">
  a, b, c, d - długości boków czworokąta<br />
  s - połowa obwodu
</p>
<p>
</p>
<h4>
  Twierdzenie:
</h4>
<p class="twierdzenie">
  <strong></strong>W czworokąt można wpisać okrąg wtedy i tylko wtedy,
gdy <strong>a + c = b + d</strong>. Pole czworokąta opisanego na
okręgu o promieniu r: <strong>P = sr</strong>.
</p>
```

```css
p:empty {
  background-color: black;
}
```

```
p:not(strong) {
  color: brown;
}
```

Zacznijmy od omówienia fragmentu

```
p:empty {
  background-color: black;
}
```
CSS

Pseudoklasa `:empty` odnosi się do pustych elementów, czyli tych, które nie mają potomka, oraz do tych, których się nie zamyka (np. `<br>`). My odnieśliśmy się do pustych akapitów `<p>`. W tym przykładzie mamy ich aż dwa. Powinny być one wyróżnione czarnym tłem. Z kolei we fragmencie

```
p:not(strong) {
  color: brown;
}
```
CSS

użyliśmy pseudoklasy negacji, czyli `:not`. Dzięki temu pokolorowaliśmy na brązowo wszystkie akapity z wyjątkiem tych fragmentów tekstów, które są objęte w znacznik `<strong>`.

Pseudoklasy `:first-child`, `:last-child`, `:only-child`

Jak już pewnie zdążyłeś/-aś zauważyć, struktura HTML przypomina jedną wielką rodzinę, której członkami są elementy należące do danych znaczników, a pokrewieństwo określa hierarchia. Bazując na tej analogii możemy wyróżnić rodziców i dzieci. Logiczne jest, że dzieci znajdują się niżej w hierarchii, czyli w znaczniku głębszym. Dzięki temu możemy wyróżnić pierwsze dziecko, ostatnie oraz jedynaków.

Jeżeli dalej nie jesteś przekonany do „tej” rodziny, to przeanalizuje ten kod:

```
<h3>
  Liczby pierwsze mniejsze od 30:
```
HTML

```
</h3>
<p>
  <strong>2</strong>,
  <strong>3</strong>,
  <strong>5</strong>,
  <strong>7</strong>,
  <strong>11</strong>,
  <strong>13</strong>,
  <strong>17</strong>,
  <strong>19</strong>,
  <strong>23</strong>,
  <strong>29</strong>,
</p>
<h3>
  Jedyna parzysta liczba pierwsza:
</h3>
<p>
  <strong>2</strong>
</p>
```

CSS

```
strong:first-child {
  color: brown;
}

p > strong:first-child {
 font-style: italic;
}

strong:last-child {
  color: blue;
}

p > strong:only-child {
  color: orange;
}
```

Pseudoklasa `:first-child` odnosi się do pierwszego dziecka, w tym przypadku do pierwszego pojawiającego się elementu `<strong>`. Zapisy `strong:first-child` i `p > strong:first-child` różnią się od siebie tylko tym, że w tym drugim określiliśmy dokładnie rodzica, czyli element bezpośrednio nad elementem `<strong>` (czyli `<p>`).

Bardzo podobnie działa pseudoklasa `:last-child`, która wskazuje na ostatnie dziecko. Z kolei pseudoklasa `:only-child` pozwala nam odnieść się do „jedynaków”, czyli do tych elementów, które nie mają „rodzeństwa”.

Po analizie powinieneś/-aś bez problemu zrozumieć, dlaczego liczba 2 w pierwszej części jest napisana pochyloną i brązową czcionką, liczba 29 ma niebieski kolor, a liczba 2 w drugiej części jest pochylona i pomarańczowa.

Pseudoklasy `:nth-child()` i `:nth-last-child()`

HTML

```
<h3>
  Kolejne liczby naturalne mniejsze od 10:
</h3>
<p>
  <strong>1</strong>,
  <strong>2</strong>,
  <strong>3</strong>,
  <strong>4</strong>,
  <strong>5</strong>,
  <strong>6</strong>,
  <strong>7</strong>,
  <strong>8</strong>,
  <strong>9</strong>,
</p>
```

CSS

```
strong:nth-child(2n+1) {
  color: brown;
}

p > strong:nth-last-child(3n) {
  font-style: italic;
}
```

Subtelną różnicę pomiędzy `strong` a `p > strong` możemy pominąć, ponieważ omówiliśmy to przy poprzednim przykładzie. Skupmy się za to na samych pseudoklasach. Dzięki ich użyciu możemy zastosować wybrany przez nas wzór z niewiadomą `n`, aby powybierać interesujące nas „dzieci". Za liczbę `n` przeglądarka będzie podstawiać kolejne liczby naturalne z zerem włącznie.

Różnica pomiędzy użytymi pseudoklasami jest taka, że pierwsza numeruje dzieci od początku do końca (liczby od 1, a nie od 0). Z kolei druga pseudoklasa numeruje dzieci od ostatniego do pierwszego.

Łatwo zauważyć, że fragment

```css
strong:nth-child(2n+1) {
  color: brown;
}
```

pokoloruje nieparzyste liczby na brązowo, a dzięki

```css
p > strong:nth-last-child(3n) {
  font-style: italic;
}
```

co trzecia liczba od końca będzie pochylona.

Dla ułatwienia można również stosować zamiast wzorów `odd` i `even`. Pierwszy z tych wyrazów odnosi się do elementów o numerach nieparzystych (zamiast 2n+1), a drugi do elementów parzystych (zamiast 2n).

Dosyć wyraźnie obrazuje to poniższy przykład, gdzie liczby nieparzyste mają białe tło i czarną czcionkę, a liczby parzyste na odwrót:

```html
<h3>
  Kolejne liczby naturalne mniejsze od 10:
</h3>
<p>
  <strong>1</strong>,
  <strong>2</strong>,
  <strong>3</strong>,
  <strong>4</strong>,
  <strong>5</strong>,
  <strong>6</strong>,
  <strong>7</strong>,
  <strong>8</strong>,
  <strong>9</strong>,
</p>
```

```css
strong:nth-child(odd) {
  color: black;
  background-color: white;
}
```

```
p > strong:nth-last-child(even) {
  color: white;
  background-color: black;
}
```

Ta wiedza przyda Ci się głównie do stylowania list oraz tabel, które poznasz już niebawem. Dzięki temu Twoja strona będzie bardziej urozmaicona, a treść nie będzie musiała być wszędzie monotonna. Zachęcamy Cię do poeksperymentowania z pisaniem wzorów-cykli. Sprawdź, jak przeglądarka zachowa się, gdy wprowadzisz liczby ujemne i inne bardziej skomplikowane operacje.

Model pudełkowy

5

Każdy element obecny na stronie zawiera się w prostokątnym obszarze, nazywanym pudełkiem. Przeanalizuj model pudełkowy poniżej - zwróć uwagę na kolejność poszczególnych warstw.

1. Zawartość (ang. content) – np. tekst lub obrazek
2. Marginesy wewnętrzne (ang. padding) - obejmują też tło
3. Obramowanie (ang. border) - pozwala wizualnie odseparować element
4. Marginesy zewnętrzne (ang. margin) - przezroczyste, zwiększają odległość od innych elementów

```
<p>Ten akapit ma na celu pokazanie jak wygląda pudełko w praktyce</
```
HTML

```
p {
border: 5px solid;
padding: 5px;
margin: 10px;
}
```
CSS

Efekt:

Elementy HTML dzielimy na blokowe i liniowe. Aby je dobrze zrozumiec popatrzmy na przykład:

```HTML
<div class="parent">

    <div class="bg">
Ten tekst bardzo dobrze pokazuje różnicę pomiędzy</div>
 elementami blokowymi, a liniowymi
</div>

<div class="parent">

    <span class="bg">
Ten tekst bardzo dobrze pokazuje różnicę pomiędzy
</span>
 elementami blokowymi, a liniowymi
</div>
```

```CSS
.bg {
background: red;
line-height: 20dp;
}
```

Elementy blokowe

```HTML
<div class="bg">
Ten tekst bardzo dobrze pokazuje różnicę pomiędzy</div>
 elementami blokowymi, a liniowymi
```

Jak widzimy na przykładzie elementy blokowe domyślnie zajmują całą możliwą szerokość oraz zaczynają i kończą się od nowej linii - zaliczamy do nich m. in. `<p>`, `<h1>`, `<table>`, `<ul>`.

Elementy liniowe

```HTML
<span class="bg">
Ten tekst bardzo dobrze pokazuje różnicę pomiędzy
```

```
</span>
 elementami blokowymi, a liniowymi
```

Zmieniając w przykładzie element blokowy `<div>` na liniowy `<span>` uzyskujemy zupełnie inny efekt. Rozmiar elementów liniowych zależy od ich zawartości, mogą występować obok siebie. Niedozwolone jest zamieszczanie w nich elementów blokowych. Przykładowo `<abbr>`, `<b>`, `<i>`.

Grupowanie elementów

Grupowanie elementów ułatwia ich formatowanie - dzięki znacznikom jak `<div>` czy `<span>` da się zastosować je do wszystkich elementów w danym "pojemniku" na raz, co znacznie skraca czas pracy. Dodatkowo bez nich nie byłoby możliwe uzyskanie niektórych efektów jeśli chodzi pozycjonowanie (o tym więcej dowiesz się w rozdziale poświęconym obrazkom).

Div

Div jest elementem blokowym, więc możemy przechowywać w nim elementy każdego typu.

HTML
```
<div class="zbior">
    <p>Ten tekst jest elementem blokowym</p>
    <b>Ten tekst jest elementem liniowym</b>
    <div>
        <p>Ten div jest zagnieżdżony</p>
    </div>
</div>
```

CSS
```
div#zbior {
background: white;
color: black;
}
```

W tym wypadku cały tekst w divie będzie koloru białego, natomiast jego tło czarne.

Span

Span to element liniowy, przechowujemy w nim tylko elementy tego typu. Poza tym działa podobnie jak `<div>` i spełnia podobne funkcje.

```HTML
<p>Zarówno div jak i <span><b><i>span</i></b><span>to bardzo użytec
tagi.</p>
```

Linki

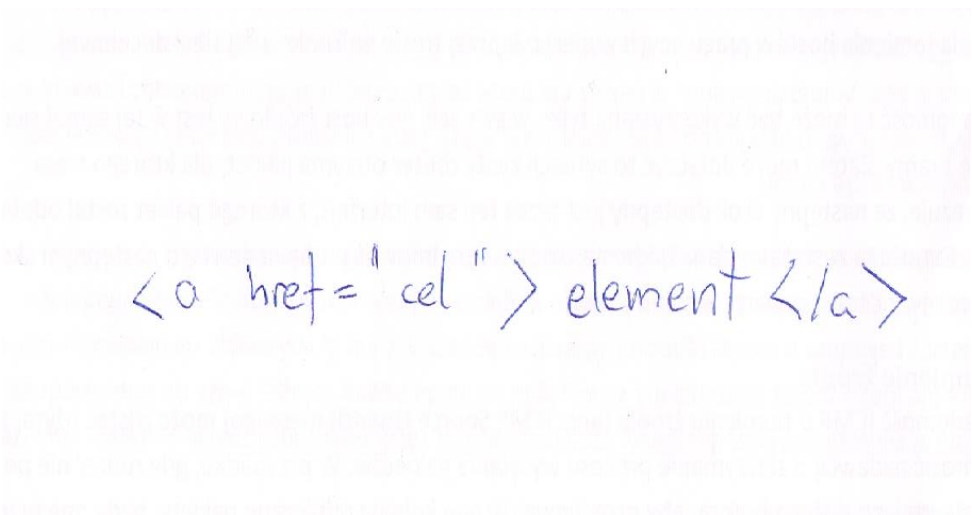
6

- struktura hiperłącza
- linki do innych stron `http://`
- linki do podstron (linki relatywne, absolutne)
- kotwice `#link`
- linki do maili `mailto:`
- atrybut `target`

W HTMLu możemy stosować hiperłącza (linki), czyli elementy które są odnośnikiem do innego dokumentu lub innego miejsca w danym dokumencie. Każdy element może być hiperłączem.

Struktura hiperłącza

Link jest definiowany przez tag `<a>` i wymaga atrybutu `href`, który zawiera adres docelowy.



Na przykład:

```HTML
<a href="link_do_dokumentu">tekst_wyświetlany_na_stronie</a>
```

```HTML
<a href="link_do_dokumentu"><img src="ścieżka_do_obrazka"/></a>
```

```HTML
<h1>Biografia zespołu Pink Floyd</h1>

<a href="#episod1" class="kotwice">Wstęp</a>
<a href="#episod2" class="kotwice">Lata 1963-67</a>
<a href="#episod3" class="kotwice">Lata 1968-77</a>
<a href="#episod4" class="kotwice">Lata 1978-85</a>
<a href="#episod5" class="kotwice">Lata 1986-95</a>
<a href="#episod6" class="kotwice">Zespół dzisiaj</a>

<br/>

<img src="../images/pinkfloyd.jpg">

<br/>

<a name="episod1"><h2>Wstęp</h2></a>
...dużo tekstu we wstępie&#8230;
<a name="episod2"><h2>Lata 1963-67: początek</h2></a>
...znowu dużo tekstu&#8230;
<a name="episod3"><h2>Lata 1968-77: przemiana i sukces</h2></a>
...i ponownie dużo tekstu w kolejnej częśći&#8230;
<a name="episod4"><h2>Lata 1978-85: era Waters`a</h2></a>
...dużo tekstu&#8230;
<a name="episod5"><h2>Lata 1986-95: era Gilmour-led</h2></a>
...tu trochę mniej niż wcześniej ale nadal dużo tekstu&#8230;
<a name="episod6"><h2>Pink Floyd dziś</h2></a>
...na zakończenie także dużo tekstu.

<a href=https://pl.wikipedia.org/wiki/Pink_Floyd target=_blank
class="inne">Źródło</a>

Inne słynne zespoły:
<a href="gunsroses.html" class="inne">Guns N' Roses</a>
<a href=http://slynnezespoly.pl/metallica.html
class="inne">Metallica</a>
<a href="extreme.html" class="inne">Extreme</a>

Zauważyłeś błąd? <a
href="mailto:admin@slynnezespoly.pl?cc=programista@slynnezespoly.pl?bcc
=programista2@slynnezespoly.pl?subject=z%20biletu?body=Line1-Witam,%0D%
0ALine2-znalazłem%20błąd%20w&#8230;">Napisz do nas!</a>
```

```css
a.kotwice {
color: black;
text-decoration: underline;
}

a.kotwice:hover {
color: green;
text-decoration: underline;
}

a.kotwice:visited {
color: black;
text-decoration: none;
}

a.kotwice:active {
color: red;
text-decoration: none;
}
```

Efekt:

Linki do innych stron

Aby odwołać się do innej strony w atrybucie `href` podajemy dokładny link do strony z przedrostkiem `http://` .

```html
<a href=https://pl.wikipedia.org/wiki/Pink_Floyd target=_blank
class="inne">Źródło</a>
```

Linki do podstron

Zobacz na kod **2.8.1**

Możemy także odwoływać się do innych stron umieszczonych na naszym serwerze. W tym wypadku w atrybucie `href` podajemy adres pliku. Adres może być względny (relatywny) lub bezwzględny (absolutny).

```html
Inne słynne zespoły:
<a href="gunsroses.html" class="inne">Guns N' Roses</a>
<a href=http://slynnezespoly.pl/metallica.html
```

```
class="inne">Metallica</a>
<a href="extreme.html" class="inne">Extreme</a>
```

Adresowanie względne

Ten sposób adresowania określa ścieżkę do pliku względem aktualnego pliku. Jest bardzo przydatne jeśli chcemy przenieść naszą stronę na inny serwer, ponieważ nie musimy zmieniać wszędzie ścieżki.

```HTML
<img src="../images/pinkfloyd.jpg">

...

Inne słynne zespoły:
<a href="gunsnroses.html" class="inne">Guns N' Roses</a>
<a href="extreme.html" class="inne">Extreme</a>
```

Jeśli wpiszemy tylko nazwę plików oznacza to że jest on w tym samym miejscu (folderze) co nasz plik z którego go wywołujemy. Możemy się też przejść do innego folderu pisząc jego nazwę wraz z ukośnikiem (`folder/`) lub za pomocą znaku `..` cofnąć się do folderu nadrzędnego.

Adresowanie bezwzględne

Stosując to adresowanie określamy dokładnie ścieżkę do pliku.

```HTML
Inne słynne zespoły:
<a href=http://slynnezespoly.pl/metallica.html
class="inne">Metallica</a>
```

Jeśli będziemy chcieli adres naszego serwera, będziemy musieli zmienić w każdym miejscu w dokumencie także link, dlatego to adresowanie stosuje się przede wszystkim jeśli odnosimy się do innych stron.

Etykiety `#link`

Służą one do odwoływania się do miejsc w danym dokumencie. Jest to bardzo przydatne jeśli mamy dużo tekstu.

Definicja:

```HTML
<a href="#nazwa_etykiety">tekt_wyświetlany_na_stronie</a>
```

Miejsce do którego się odwołujemy:

```HTML
<a name="nazwa_etykiety">tekst</a>
```

Na przykład:

```HTML
<a href="#episod1">Wstęp</a>
<a href="#episod2">Lata 1963-67</a>
<a href="#episod3">Lata 1968-77</a>
<a href="#episod4">Lata 1978-85</a>
<a href="#episod5">Lata 1986-95</a>
<a href="#episod6">Zespół dzisiaj</a>

...

<a name="episod1"><h2>Wstęp</h2></a>
...dużo tekstu we wstępie&#8230;
<a name="episod2"><h2>Lata 1963-67: początek</h2></a>
...znowu dużo tekstu&#8230;
<a name="episod3"><h2>Lata 1968-77: przemiana i sukces</h2></a>
...i ponownie dużo tekstu w kolejnej części&#8230;
<a name="episod4"><h2>Lata 1978-85: era Waters`a</h2></a>
...dużo tekstu&#8230;
<a name="episod5"><h2>Lata 1986-95: era Gilmour-led</h2></a>
...tu trochę mniej niż wcześniej ale nadal dużo tekstu&#8230;
<a name="episod6"><h2>Pink Floyd dziś</h2></a>
...na zakończenie także dużo tekstu.
```

W tym przykładzie jeśli klikniemy na któryś z odnośników na górze zostaniemy od razu przeniesieni do odpowiedniej części bez zbędnego i długiego przewijania.

Odsyłacz `mailto:`

Po naciśnięciu na ten odsyłacz otworzy się nam domyślny klient poczty na komputerze, aby móc wysłać e-maila do zdefiniowanego odbiorcy.

```HTML
<a href="mailto:nazwa@domena.pl">opis_linku</a>
```

Możemy także od razu zdefiniować innych odbiorców (DW i UDW), temat oraz treść:

```HTML
Zauważyłeś błąd? <a href="mailto:admin@slynnezespoly.pl?
subject=z%20biletu?body=Line1-Witam,%0D%0ALine2-znalazłem%20błąd%20
230;">Napisz do nas!</a>
```

```HTML
Zauważyłeś błąd? <a
href="mailto:admin@slynnezespoly.pl?cc=programista@slynnezespoly.pl
=programista2@slynnezespoly.pl?subject=z%20biletu?body=Line1-Witam,%0D%
0ALine2-znalazłem%20błąd%20w&#8230;">Napisz do nas!</a>
```


UWAGA!

Znak `%20` w powyższym kodzie oznacza spację, a `%0D%` nową linię

Atrybut `target`

Atrybut `target` pozwala nam określić gdzie będzie otwierany nowy dokument.

```HTML
<a href=https://pl.wikipedia.org/wiki/Pink_Floyd target=_blank
class="inne">Źródło</a>
```

Wartość	Opis
_blank	otwórz nowy dokument w nowej karcie lub oknie
_self	otwórz nowy dokument w tej samej karcie
_parent	otwórz nowy dokument w nadrzędnym oknie

Wartość	Opis
_top	otwórz nowy dokument w trybie pełnoekranowym

CIEKAWOSTKA

Domyślnie hiperłącze zostanie otwarte w tej samej karcie (`_self`).

Zobacz na kod **2.8.2** 

Pseudoklasy `:link` `:hover` `:visited` `:active` `:focus`

Pseudoklasy stosujemy, aby okrelić co się stanie z danym elementem kiedy użytkownik wykona określoną czynność (np. najedzie lub kliknie na niego). Są one definiowane w arkuszach stylów przez dopisanie do elementu pseudoklasy.

```
a:pseudoklasa {
cecha: wartość;
}
```

CSS

Pseudoklasy związane z linkami:

Pseudoklasa	Opis
`:link`	określa wygląd elementów bedących linkami
`:hover`	określa wygląd elementów gdy użytkownik najedzie kursorem na link
`:visited`	określa wygląd linków, które zostały już odwiedzone
`:active`	określa wygląd linków które są w danym momencie aktywne (np. w trakcie kliknięcia)
`:focus`	określa wygląd elementów które s w danym momencie zogniskowane (np. przez wybranie ich za pomocą klawisza TAB)

```
a.kotwice {
color: black;
text-decoration: underline;
}

a.kotwice:hover {
color: green;
text-decoration: underline;
}

a.kotwice:visited {
color: black;
text-decoration: none;
}

a.kotwice:active {
color: red;
text-decoration: none;
}```
```

CSS

Zobacz na kod **2.8.3**

Listy

7

Jest wiele okazji, kiedy musimy wyszczególnić elementy w postaci listy. HTML udostępnia ich trzy rodzaje: * **Listy numerowane**, gdzie każda pozycja na liście ma swój numer - na przykład może to być zestaw kroków, które muszą być wykonane aby upiec ciasto, * **Listy nieuporządkowane** - każdy podpunkt zaczyna się od graficznego znacznika, np. kropki lub myślnika, * **Listy definicji** - składają się z szeregu pojęć wraz z definicją każdego z nich.

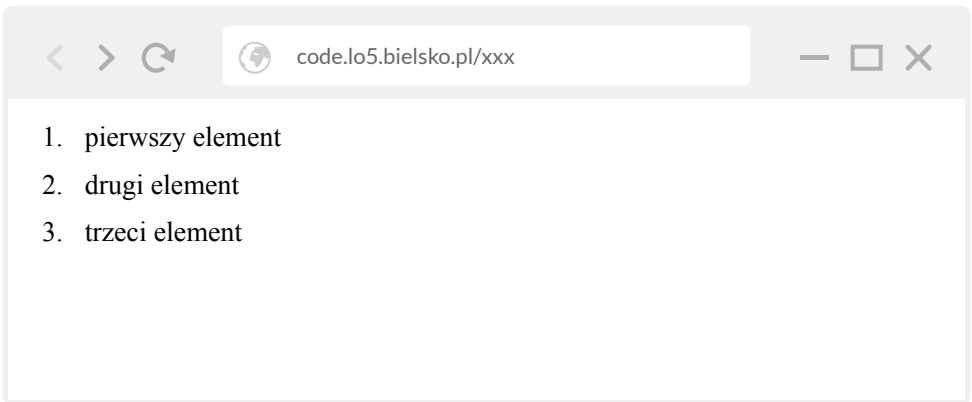
Listy numerowane - `<ol>`

Lista numerowana zaczyna się od tagu `<ol>`. Obejmuje on wszystkie poszczególne elementy listy złożone z tagów `<li>`.

HTML

```
<ol>
    <li>pierwszy element</li>
    <li>drugi element</li>
    <li>trzeci element</li>
</ol>
```

Podczas wyświetlania przeglądarka sama zadba o to, aby poszczególne elementy listy miały odpowiednie wcięcie. Powyższy efekt w przeglądarce będzie wyglądał następująco:



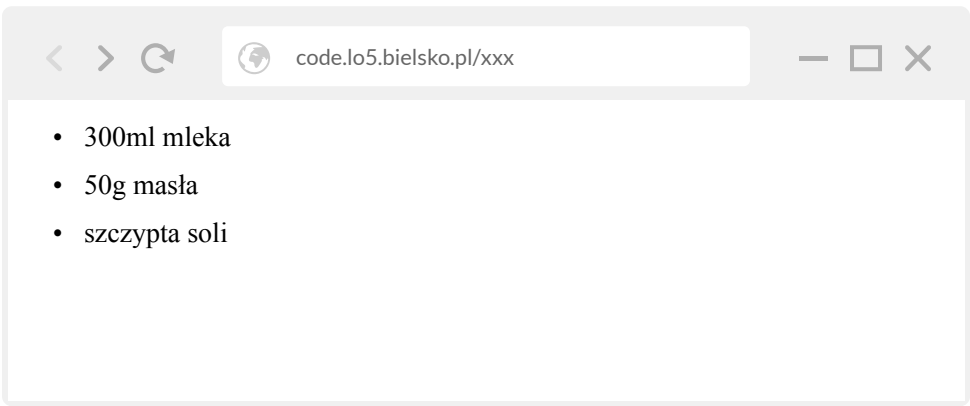
Pamiętajmy, że każdy element listy musi być objęty w `<li>`. Bezpośrednio wewnątrz listy nie może znaleźć się żaden inny element.

Listy nieuporządkowane - `<ul>`

Listy nieuporządkowane tworzymy w identyczny sposób jak listy numerowane, z tą różnicą, że stosujemy tag `<li>`:

```HTML
<ul>
    <li>300ml mleka</li>
    <li>50g masła</li>
    <li>szczypta soli</li>
</ul>
```

Efekt będzie podobny jak w poprzednim przykładzie, tylko zamiast cyfr każdy podpunkt będzie zaczynał się od kropki:



ZADANIE

Stwórz prosty przepis na omlety składający się z listy produktów (lista nienumerowana), listy czynności do wykonania (lista numerowana) oraz krótkiego opisu dania. Zadbaj, aby poszczególne sekcje miały odpowiednie nagłówki.

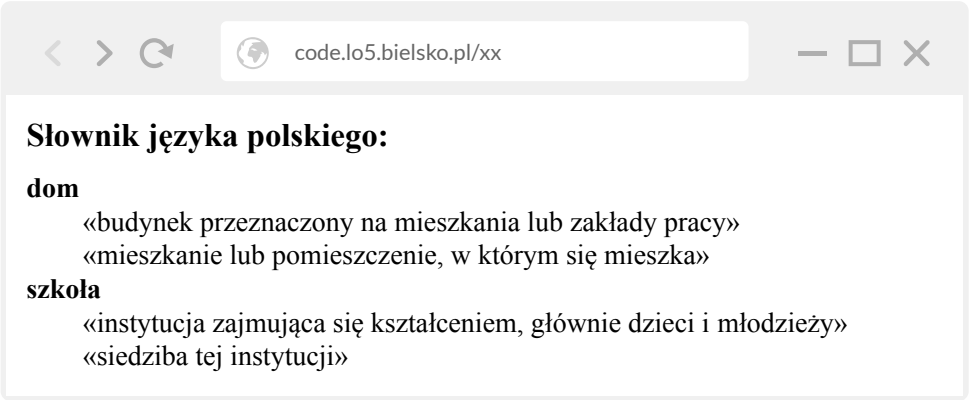
Listy definicji - `<dl>`

Lista definicji jest przydatna, gdy piszemy słownik, w którym znajdują się pewne wyrazy i ich objaśnienia. Objaśnienia są zwykle przesunięte bardziej w prawo, dzięki czemu lista

staje się czytelniejsza. Każda lista składa się z elementu `<dl>` i zawiera w sobie poszczególne terminy `<dt>` oraz opisy do nich `<dd>`:

```HTML
<h3>Słownik języka polskiego:</h3>
<dl>
    <dt>dom</dt>
    <dd>«budynek przeznaczony na mieszkania lub zakłady pracy»</dd>
    <dd>«mieszkanie lub pomieszczenie, w którym się mieszka»</dd>
    <dt>szkoła</dt>
    <dd>«instytucja zajmująca się kształceniem, głównie dzieci
i młodzieży»</dd>
    <dd>«siedziba tej instytucji»</dd>
</dl>
```

Efekt w przeglądarce będzie następujący:



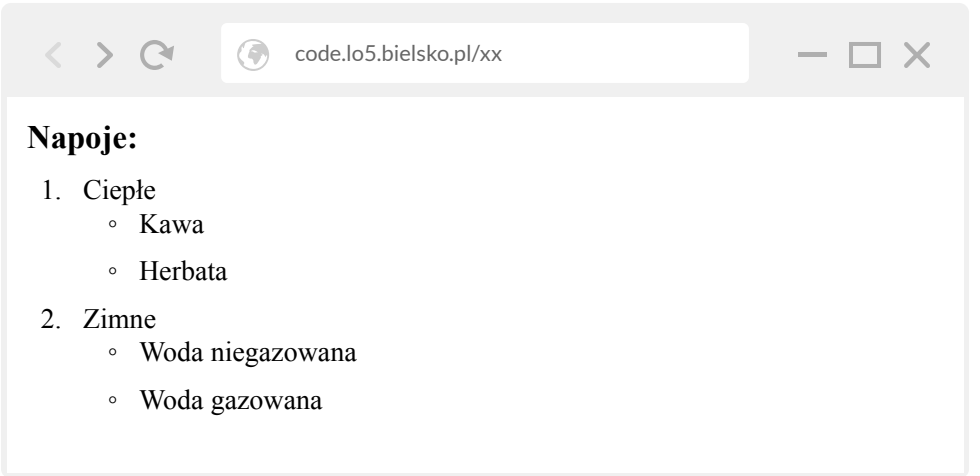
Czasami zachodzi potrzeba, aby dany termin posiadał kilka definicji lub kilka terminów mają taką samą definicję. Lista `<dl>` daje nam takie możliwości.

Listy zagnieżdżone

Listy możemy dowolnie zagnieżdżać pomiędzy sobą. Możemy tworzyć kolejne poziomy zagłębień list w poszczególnych elementach. Zobaczmy na przykład:

```HTML
<h3>Napoje:</h3>
<ol>
    <li>Ciepłe
        <ul>
```

```
            <li>Kawa</li>
            <li>Herbata</li>
        </ul>
    </li>
    <li>Zimne
        <ul>
            <li>Woda niegazowana</li>
            <li>Woda gazowana</li>
        </ul>
    </li>
</ol>
```



Musimy pamiętać, aby kolejne listy umieszczać wewnątrz tagu `<li>`, gdyż należą one do poszczególnych podpunktów. Przeglądarka automatycznie doda kolejne poziomy wcięcia tak, aby listy były przejrzyste.

Kolor

8

- kolor
- background kolor
- opacity

Obrazki

9

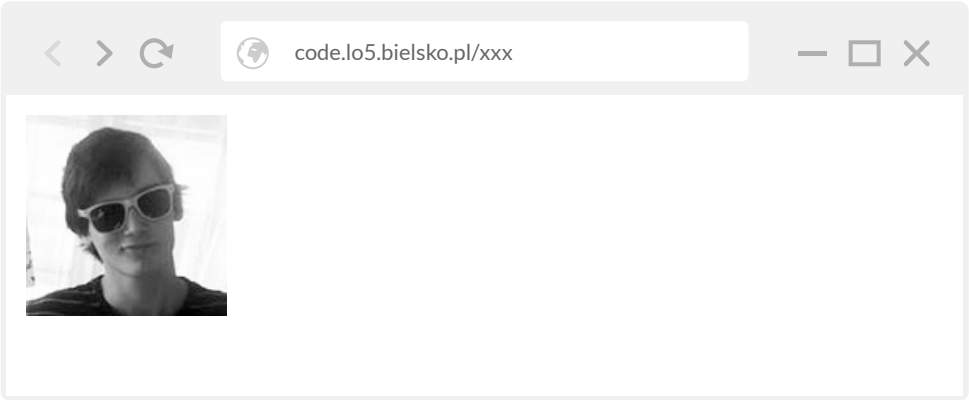
Obrazy są kolejną fundamentalną częścią każdej strony internetowej. Umiejętne ich dobranie, pozycjonowanie, ustawienie rozmiarów są następnymi umiejętnościami, które powinniśmy posiąść przy tworzeniu własnej strony.

Znacznik `<img>`

Zacznijmy od bardzo prostego przykładu:

```HTML
<img src="/img/avatar-13.jpg" alt="Niezłe ciacho xD">
```

Obrazy wstawiamy za pomocą znacznika `<img>`. Najważniejszym i wymaganym jego atrybutem jest `src`, w którym podajemy adres do naszej grafiki. Równie dobrze może to być plik w Internecie (jak w przykładzie) oraz plik na naszym komputerze. Zwróć uwagę, aby poprawnie podać ścieżkę dostępu do niego.



Pamiętajmy, żeby podać argument `alt`, czyli tekst alternatywny. Jego efekt nie zawsze jest widoczny na stronie, jednak warto o nim pamiętać. Dzięki niemu wyszukiwarki internetowe wiedzą co znajduje się na obrazku, a co za tym idzie nasza strona internetowa zyskuje większą wartość.

UWAGA!

Jeśli uważamy, że nie powinniśmy opisywać zawartości obrazka (np. jest to tło strony, albo element nic nie wnoszący do treści), wówczas dodajemy pusty atrybut `alt=""`.

Ustawianie wymiarów elementów

Umiemy dodawać już obrazki na naszą stronę, zobaczmy teraz jak możemy manipulować ich rozmiarami. Zobaczmy na kolejny przykład:

HTML

```
<img src="/img/avatar-1.jpg" alt="" class="ciacho-1">
<img src="/img/avatar-2.jpg" alt="" class="ciacho-2">
<img src="/img/avatar-3.jpg" alt="" class="ciacho-3">
<img src="/img/avatar-4.jpg" alt="" class="ciacho-4">
```

Stworzyliśmy 4 obrazki (standardowo mają po 100px szerokości), które mają klasy od `ciacho-1` do `ciacho-4`, dodajmy do nich odrobinę czarów w CSS:

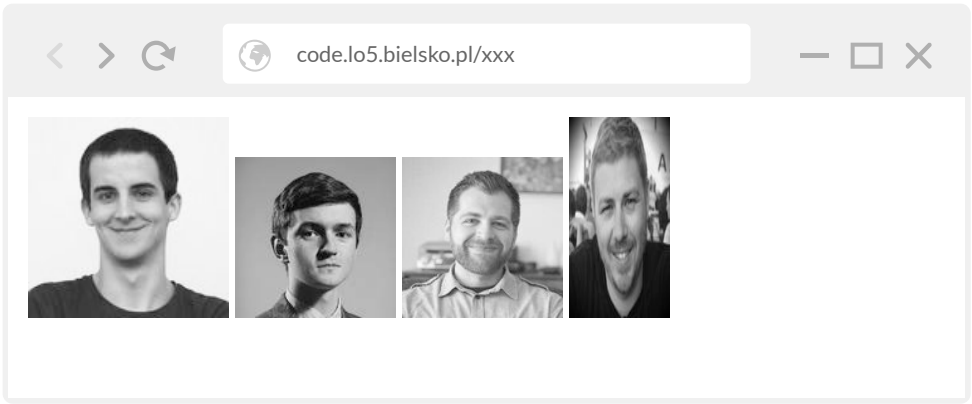
CSS

```
img.ciacho-2 {
  height: 80px;
}

img.ciacho-3 {
  width: 80px;
}

img.ciacho-4 {
  width: 50px;
  height: 100px;
}
```

Zobaczmy jaki efekt będzie w przeglądarce internetowej:



Postarajmy się teraz przeanalizować stworzony przez nas kod. Mamy na stronie 4 obrazki, pierwszemu nie nadaliśmy żadnych wymiarów w CSS, więc przeglądarka wyświetli go w naturalnych rozmiarach, czyli *100px* na *100px*. Drugiemu obrazkowi ustawiliśmy wysokość na *80px*, a że nie podaliśmy szerokości to przeglądarka automatycznie ją za nas dostosowała. Analogicznie jest z obrazkiem numer 3. Z kolei ostatni obrazek (`.ciacho-4`) ma ustawioną zarówno szerokość jak i wysokość - tutaj zdjęcie zostało zdeformowane, gdyż są to dokładnie takie wymiary jak podaliśmy.

UWAGA!

Jeśli podamy tylko jeden wymiar (wysokość lub szerokość) to przeglądarka automatycznie dopasuje drugą wartość tak, aby zdjęcie miało naturalne proporcje.

Przeanalizujmy kolejny przykład:

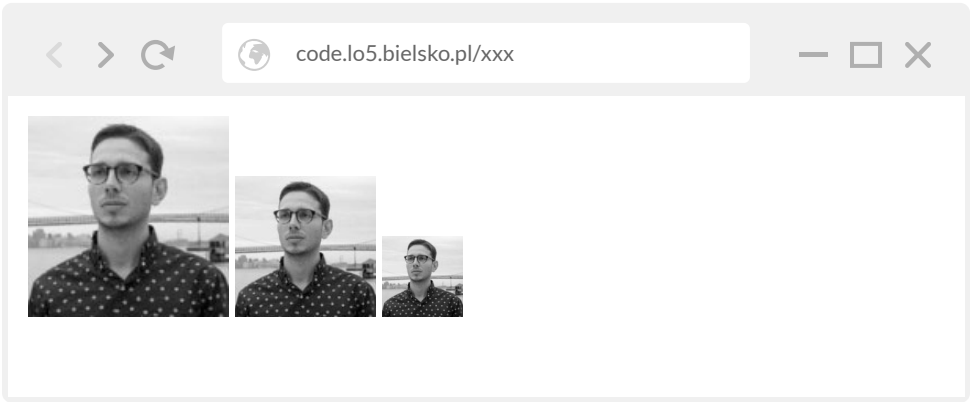
HTML

```
<img src="/img/avatar-9.jpg" alt="" class="avatar-large">
<img src="/img/avatar-9.jpg" alt="" class="avatar-middle">
<img src="/img/avatar-9.jpg" alt="" class="avatar-small">
```

CSS

```
.avatar-large { width: 100px; height: 100px; }
.avatar-middle { width: 70px; height: 70px; }
.avatar-small { width: 40px; height: 40px; }
```

Za pomocą kilku klas zrobiliśmy prosty mechanizm, który pozwoli nam na zmianę rozmiarów zdjęć na całej stronie internetowej. Jako, że użyliśmy klas, a nie identyfikatorów, to będziemy mogli używać naszych stylów do wielu elementów na stronie. Zobaczcie jaki jest efekt w przeglądarce:

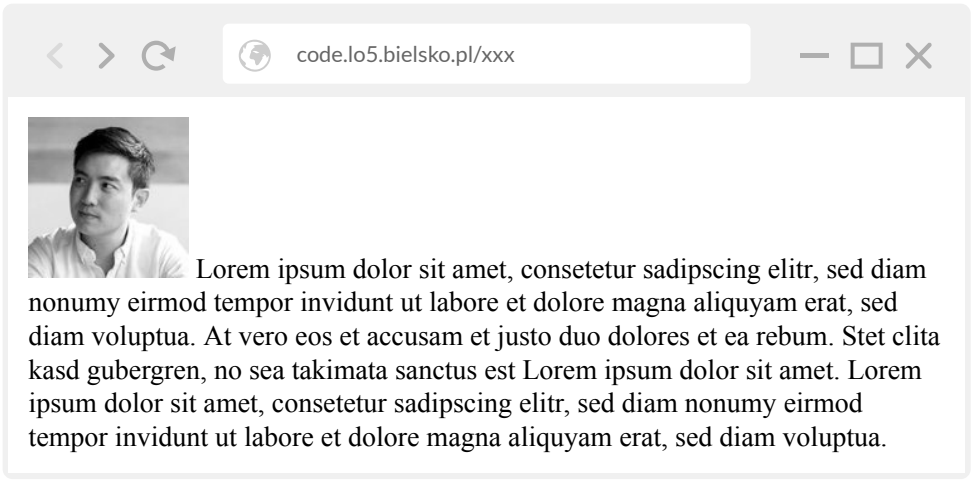


Ścieżki relatywne i absolutne

Opływanie elementów - `float`

Czasami zachodzi potrzeba, żeby wkomponować obrazek do tekstu, który je opisuje. Zobaczmy co przeglądarka zrobi, gdy wstawimy zdjęcie oraz dodamy trochę tekstu:

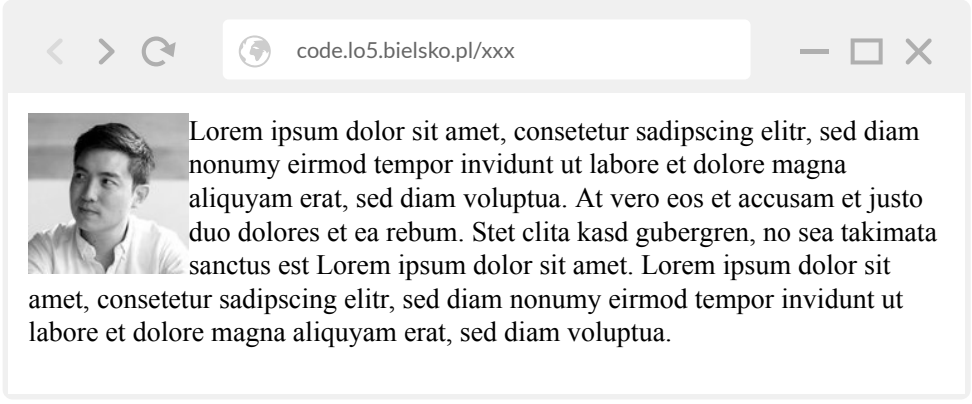
```HTML
<img src="img/avatar-9.jpg" alt="" >
Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam
nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat
(...)
```



Nie wygląda to zbyt atrakcyjnie. Obrazek oraz tekst standardowo są elementami liniowymi, więc przeglądarka będzie je wyświetlała obok siebie. Możemy to oczywiście zmodyfikować za pomocą kilku linii kodu:

```css
img {
    float: left;
}
```

Od teraz tekst oraz pozostałe elementy będą "opływać" obrazek z prawej strony. Efekt będzie taki jak sobie założyliśmy:



CIEKAWOSTKA

`float` możemy używać nie tylko do obrazków, ale możemy zastosować również np. do elemenu `<div>`. Pamiętajmy tylko żeby dodać do niego wtedy odpowiednią szerokość.

Pozbycie się opływania - `clear`

Jak w poprzednim dziale się dowiedzieliśmy `float` wpływa na wszystkie elementy które znajdują się za opływanym elementem. Czasami zachodzi potrzeba pozbycia się nieoczekiwanych efektów.

Załóżmy, że chcemy zrobić listę osób, składającą się ze zdjęcia oraz imienia i nazwiska. Każde zdjęcie będzie posiadało `float: left`, dzięki czemu znajdzie się po lewej stronie. Zobaczmy na przykład:

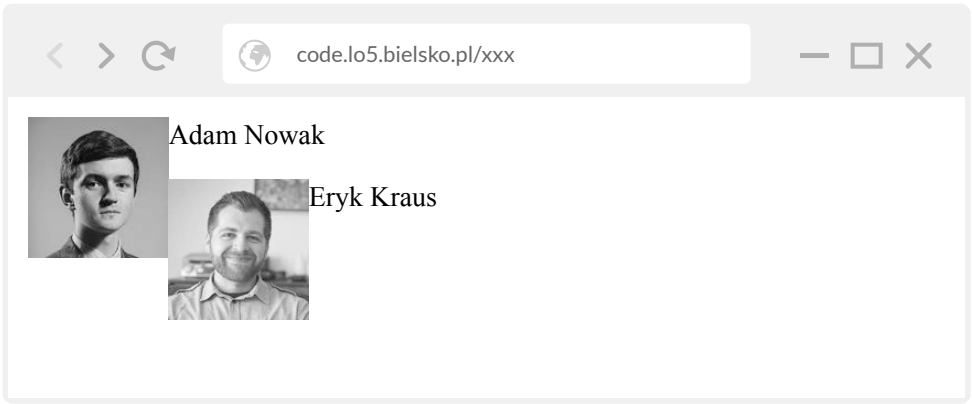
HTML

```
<p>
    <img src="img/avatar-2.jpg" alt=""> Adam Nowak
</p>
<p>
    <img src="img/avatar-3.jpg" alt=""> Eryk Kraus
</p>
```

CSS

```
img { float: left; }
```

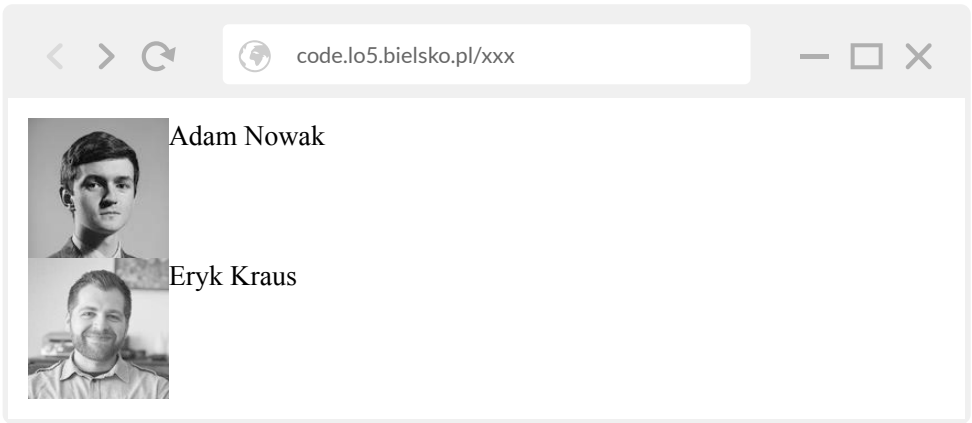
Oraz efekt działania:



Efekt jest nie do końca taki jak oczekiwaliśmy. `float: left` dodane do obrazka spowodowało, że wszystkie pozostałe elementy także zostały przesunięte do prawej, ponieważ wysokość obrazka jest większa, niż wysokość tekstu po prawej. Aby zapobiec temu, możemy pozbyć się opływania dla konkretnych elementów. Wystarczy dodać właściwość `clear` do `<p>` w kodzie CSS:

```css
p { clear: left; }
```

Efekt będzie dokładnie taki, jak sobie założyliśmy:



Powiedzieliśmy przeglądarce, żeby dla akapitów (w których jest obrazek oraz tekst) nie stosował już opływania elementów z prawej strony (`float: left`).

Dodawanie tła do elementów - `background`

Każdy element dodany w języku HTML może posiadać tło. Może to być zarówno obrazek, jak i zwykły kolor. Zobaczmy na przykład:

```html
<p>
    Ten element będzie posiadał tło.
</p>
```

HTML

```css
p {
    background: url(/img/background-21.png);
    height: 100px;
    font-size: 20px;
}
```

CSS

W tym przykładzie dodaliśmy jeden akapit z tekstem, a następnie za pomocą CSS nakazaliśmy nadać mu odpowiednie tło. Dodaliśmy przy okazji wysokość elementu, dzięki czemu efekt będzie lepiej widoczny:

Wewnątrz właściwości `background` podajemy ścieżkę do obrazka jakie ma być tłem, ujętą wewnątrz `url()`. Oczywiście nic nie stoi na przeszkodzie, aby zamiast obrazka podać kolor, np.:

```css
p {
    background: orange;
}
```

Od teraz akapit zamiast obrazka będzie posiadał pomarańczowe tło.

ZADANIE

Stwórz kilka kwadratów za pomocą `<div>`, które leżąc obok siebie będą tworzyły tęczę. Możesz do tego użyć następujących kolorów w CSS: `red`, `orange`, `yellow`, `greenyellow`, `cyan`, `blue` oraz `darkviolet`.

Pozycjonowanie `position`

Oprócz dobrania rozmiarów zwróć uwagę na to, jak elementy są pozycjonowane, czyli jak położenie jednych zależy od położenia drugich. Często nawet niewielka zmiana może skutkować rażącymi błędami.

Pozycjonowanie absolutne i relatywne

Wyobraź sobie, że tworzysz stronę i chcesz ustalić położenie nagłówka. W tym przypadku zawsze będzie on u samej góry np. z lewej strony. Idealnym rozwiązaniem jest zastosowanie pozycjonowania absolutnego - `position: absolute`. Poprzez parametry `right`, `top`, `left`, `bottom`, możemy ustawiać odległość obrazka od krawędzi odpowiednio: prawej, górnej, lewej i dolnej. W poniższym przykładzie został w ten sposób ostylowany obrazek z klasą `header`.

Z drugiej strony możemy łatwo przewidzieć, gdzie będzie znajdował się nasz obrazek na stronie, ponieważ elementy domyślnie są ustawiane blokowo. Aby przesunąć obrazek od jego pierwotnego ułożenia możemy użyć pozycjonowania relatywnego - `position: relative`. Wtedy również użyjemy parametrów `right`, `top`, `left`, `bottom`, jednak tutaj będą oznaczać one odległości obrazków od ich początkowych pozycji. W tym przykładzie jest tak pozycjonowany obrazek z klasą `image`.

```
<img class="header" src="http://lo5.bielsko.pl/image/banner/slidesh
5332af93540052.jpg" alt="naglowek">
<img class="image" src="http://lo5.bielsko.pl/public/news/attachments/
55686a9ff2fc6LOGO_Of.Fe.jpg" alt="logo VLO w Bielsku-Białej">
```
HTML

```
img.header {
  position: absolute;
  top: 0px;
  left: 0px;
}

img.image {
  position: relative;
  width: 200px;
  top: 150px;
  left: 10px;
}
```
CSS

Pozycjonowanie ustalone i statyczne

Na wielu stronach z pewnością dostrzegłeś „pływające menu”. Jest to dosyć częste zastosowanie pozycjonowania ustalonego - `position: fixed`. Elementy w nim mają ustaloną pozycję względem krawędzi ekranu monitora, dlatego nawet przewijając stronę w dół, przykładowe menu jest zawsze np. przy górnej krawędzi okna.

Z kolei dawne, normalne pozycjonowanie uzyskujemy dzięki pozycjonowaniu statycznemu - `position: static`.

```
<img class="logo" src="http://lo5.bielsko.pl/public/news/attachment
55686a9ff2fc6LOGO_Of.Fe.jpg" alt="logo V LO w Bielsku-Białej">
<img class="image" src="http://lo5.bielsko.pl/public/images/gallery/
images/0/00002.jpg" alt="obrazek1">
<img class="image" src="http://lo5.bielsko.pl/public/images/gallery/
images/0/00006.jpg" alt="obrazek2">
```
HTML

```
img {
  position: fixed;
}

img.image {
```
CSS

```
  width: 100%;
  position:static;
}

img.logo {
  width: 150px;
  top: 10px;
  left: 10px;
}
```

W powyższym przykładzie najpierw wszystkim obrazkom zostało przypisane pozycjonowanie `fixed`, a później obrazkowi o klasie `image` zostało przywrócone pierwotne pozycjonowanie poprzez `static`.

Wyświetlanie `display`

Wcześniej poznaliśmy już różnicę pomiędzy wyświetlaniem elementów blokowych i liniowych. Teraz możemy wybrać, jak chcemy potraktować obraz w tekście – jak element liniowy, czy blokowy.

Spójrz na poniższy przykład:

HTML
```
<p>
  VLO w Bielsku-Białej

<img class="logo" src="http://lo5.bielsko.pl/public/news/attachments/
55686a9ff2fc6LOGO_Of.Fe.jpg" alt="logo V LO w Bielsku-Białej">

  obchodzi XXV-lecie!
</p>
```

CSS
```
img.logo {
  width: 150px;
  display: block;
}
```

Teraz cecha `display` ma wartość `block`, więc obraz jest wyświetlany jako element blokowy, więc tekst znajduje się nad i pod nim. Jeżeli jednak zmienimy tą wartość na `inline`,

to wyświetlanie będzie takie samo, jak dla elementów liniowych i cały tekst z obrazem będą się znajdować w jednej linii.

Przyleganie `clear`

Poza ustawieniem elementów w pionie i w poziomie możemy zdecydować, czy one w ogóle mają ze sobą sąsiadować poprzez cechę `clear`.

```
<img class="logo" src="http://lo5.bielsko.pl/public/news/attachment
55686a9ff2fc6LOGO_Of.Fe.jpg" alt="logo V LO w Bielsku-Białej">
<p>
  V LO w Bielsku-Białej obchodzi XXV-lecie!
</p>
```

HTML

```
img.logo {
  width: 150px;
  float: left;
}

p {
  clear: left;
}
```

CSS

Podając wartość cechy `clear` decydujemy, do której strony danego elementu nie mogą przylegać inne elementy. W powyższym przykładzie jest fragment:

```
img.logo {
  width: 150px;
  float: left;
}
```

CSS

A z tego wynika, że obrazek powinien znajdować się po lewej stronie tekstu. Jednak akapit ma cechę `clear: left`, więc jego lewy brzeg nie może sąsiadować z innymi elementami, więc nie może on wcale sąsiadować z obrazkiem.

Oczywiście możemy również cesze `clear` nadać wartość `right`, ale również `both` - z żadnej strony nie mogą przylegać inne elementy oraz `none` - elementy mogą przylegać z obu stron.

UWAGA!

Cechę `clear` można stosować tylko do elementów wyświetlanych blokowo.

Nakładanie się elementów `z-index`

Podczas pozycjonowania zdarza się, że jedne elementy najeżdżają na inne. Pewnie zauważyłeś/-aś, że ostatni element w strukturze HTML będzie najbardziej na wierzchu. Nie jest to wygodne rozwiązanie, ponieważ przy pisaniu strony chcemy mieć jak największą kontrolę nad wszystkim. Tutaj możemy użyć polecenia `z-index`, które określa, który element leży na jakiej „głębokości". Im wyższą ma on wartość, tym bardziej jest on na wierzchu. Gdybyśmy w poniższym przykładzie nie skorzystali z tej cechy, to duży obrazek zakryłby tekst i logo.

HTML

```
<div id="header">
 <img class="logo" src="http://lo5.bielsko.pl/public/news/attachmen
55686a9ff2fc6LOGO_Of.Fe.jpg" alt="logo V LO w Bielsku-Białej">
<p><strong>
  V LO w Bielsku-Białej obchodzi XXV-lecie!
</strong></p>
</div>
<img class="image" src="http://lo5.bielsko.pl/image/big/gallery/images/
0/00006.jpg" alt="zdjecie">
```

CSS

```
#header {
  position: absolute;
  top: 20px;
  left: 20px;
  width: 330px;
  z-index: 2;
}

img.logo {
  width: 150px;
```

```
  float: left;
}

p {
  color: orange;
  background: white;
}

.image {
  width: 100%;
  position: absolute;
  top: 0px;
  left: 0px;
  z-index: 1;
}
```


CIEKAWOSTKA

Ekran monitora możemy traktować jako płaszczyznę ze współrzędnymi X i Y. W takim razie oś prostopadła do monitora i zwrócona od niego do nas to Z. Właśnie stąd wzięła się nazwa „z-index" – im wyższa wartość, tym „płyciej" położone są elementy.

Załącznik `<figure>`

Na zakończenie tego rozdziału chcieliśmy Ci przedstawić strukturę załącznika, czyli sposób, który z reguły jest używany do wyróżnienia w kodzie rysunków, diagramów, ilustracji, zdjęć itp.

HTML

```
<h2>
  Jubileusz Piątki!
</h2>
<figure id="logo">
     <img src="http://lo5.bielsko.pl/public/news/attachments/
55686a9ff2fc6LOGO_Of.Fe.jpg" alt="logo V LO w Bielsku-Białej">
     <figcaption>Logo V LO w Bielsku-Białej</figcaption>
</figure>
```

```css
figure > img {
  width: 300px;
}
```

Dzięki temu zabiegowi obraz może mieć swój własny podpis. Na pewno często spotkałeś/-aś się z tym w różnych podręcznikach, gdzie każdy obrazek miał tytuł lub dopisek „Rysunek 1 …”. Struktury `<figure>` i `<figurecaption>` służą właśnie w takich przypadkach.

Tabele

10

- struktura tabeli (table, tr, td)
- naglowki (th)
- łaczenie kolumn i wierszy (colspan, rowspan)
- duże tabele (thead, tbody, tfoot)

Struktura

Tabele definiowane sa przez tag `<table>`. W tabeli wyróżniamy wiersze (`<tr>`) i kolumny (`<td>`). Bardzo ważne jest, żeby definiować kolumny w wierszach, a nie na odwrót.

```html
<table>
    <tr>
        <td>1</td>
        <td>2</td>
        <td>3</td>
        <td>4</td>
    </tr>
    <tr>
        <td>1</td>
        <td>2</td>
        <td>3</td>
        <td>4</td>
    </tr>
</table>
```

HTML

Efekt:

Aby nasz tabela była widoczna musimy dodać do niej obramowanie. Uzyskamy to dodając atrybut `border` .

```html
<table border=”1”>
    <tr>
        <td>1</td>
        <td>2</td>
        <td>3</td>
        <td>4</td>
    </tr>
```

HTML

```
    <tr>
        <td>1</td>
        <td>2</td>
        <td>3</td>
        <td>4</td>
    </tr>
</table>
```

Efekt:

Elementy tabeli

Przyjrzyjmy się teraz dwóm przykładom:

HTML

```
<table>
    <caption>Plan lekcji 2B</caption>
    <tr>
        <th>Godziny</th>
        <th>Poniedziałek</th>
        <th>Wtorek</th>
        <th>Środa</th>
        <th>Czwartek</th>
        <th>Piątek</th>
    </tr>
    <tr>
        <td>08:00-08:45</td>
        <td>niemiecki</td>
        <td></td>
        <td>niemiecki</td>
        <td>wos</td>
        <td></td>
    </tr>
    <tr>
        <td>08:55-09:40</td>
        <td>religia</td>
        <td>historia</td>
        <td>geografia</td>
        <td>biologia</td>
        <td>pp</td>
    </tr>
    <tr>
        <td>09:40-10:35</td>
        <td>religia</td>
        <td>informatyka</td>
        <td>fizyka</td>
```

```
        <td>matematyka</td>
        <td>matematyka</td>
    </tr>
    <tr>
        <td>10:45-11:30</td>
        <td>historia</td>
        <td>polski</td>
        <td>polski</td>
        <td>pp</td>
        <td>chemia</td>
    </tr>
    <tr>
        <td>11:50-12:35</td>
        <td>angielski</td>
        <td>angielski</td>
        <td>edb</td>
        <td>wok</td>
        <td>matematyka</td>
    </tr>
    <tr>
        <td>12:45-13:30</td>
        <td>matematyka</td>
        <td>polski</td>
        <td>polski</td>
        <td>angielski</td>
        <td>angielski</td>
    </tr>
    <tr>
        <td>13:40-14:25</td>
        <td>wf</td>
        <td></td>
        <td>basen</td>
        <td>fizyka</td>
        <td></td>
    </tr>
    <tr>
        <td>14:35-15:20</td>
        <td>wf</td>
        <td></td>
        <td></td>
        <td>fizyka</td>
        <td></td>
    </tr>
</table>
```

CSS

```
table {
    border: 1;
```

```
    width: 100%;
}

caption {
    color: red;
    font-size: 200%;
}

th {
    background: #FF0;
}

td {
    text-align: center;
}

td:first-child {
    background: #FF0;
}

tr:nth-child(even) {
    background: #CCC
}

tr:nth-child(odd) {
    background: #FFF
}
```

Efekt:

HTML

```
<table border="2" width=100%>
<caption style="font-size:200%">Superchat - znajomi</caption>
    <thead>
            <tr>
            <td>Imię</td>
            <td>Nazwisko</td>
            <td>Adres</td>
            <td>Miasto</td>
            <td>Miejsce urodzenia</td>
            <td>Telefon</td>
        </tr>
         </thead>
         <tbody>
        <tr>
            <td>Krzysztof</td>
            <td>Najber</td>
            <td>Kwiatowa 4</td>
```

```
            <td>Żywiec</td>
            <td>Bielsko-Biała</td>
            <td>698731233</td>
        </tr>
        <tr>
            <td>Piotr</td>
            <td>Świerszcz</td>
            <td>Polna 2</td>
            <td colspan="2" class="scalone">Żywiec</td>
            <td>687234134</td>
        </tr>
        <tr>
            <td>Aneta</td>
            <td rowspan="2" class="scalone">Ryś</td>
            <td>Bogata 8</td>
            <td>Pietrzykowice</td>
            <td rowspan="2" class="scalone">Bielsko-Biała</td>
            <td>602345624</td>
        </tr>
        <tr>
            <td>Maria</td>
            <td>Biedna 10</td>
            <td>Łodygowice</td>
            <td>676937154</td>
        </tr>
        <tr>
            <td>Jakub</td>
            <td>Kowal</td>
            <td>Rzemieślnicza 17</td>
            <td colspan="2" class="scalone">Bielsko-Biała</td>
            <td>507094726</td>
        </tr>
         </tbody>
    <tfoot>
        <tr>
            <td colspan="6" class="scalone">Copyright by superStudio
2016</td>
        </tr>
    </tfoot>
</table>
```

CSS

```
.scalone {
    text-align:center;
}
thead {
    color:#eee;
    background:#ccc;
```

```
}
tbody {
    color:black;
}
tfoot{
    background: yellow;
}
```

Efekt:

Nagłówki `<th>`

Jak łatwo można się domyślić patrząc na powyższy przykład tag `<th>` może zastępować `<td>` i służy on do oznaczania danej komórki jako nagłówka.

```HTML
<tr>
    <th>Godziny</th>
    <th>Poniedziałek</th>
    <th>Wtorek</th>
    <th>Środa</th>
    <th>Czwartek</th>
    <th>Piątek</th>
</tr>
```

Znacznik `<caption>`

Znacznik ten służy do określania tytułu tabeli.

```HTML
<caption>Plan lekcji 2B</caption>
```

Pseudoklasy `:first:child` , `:nth-child(n)`

Te pseudooklasy służą do określania wyglądu pierwszego lub powtarzających się wierszów.

`:first-child`

Określa wygląd pierwszego wiersza lub kolumny.

```css
td:first-child {
    background: #FF0;
}
```

```css
tr:first-child {
    background: #FF0;
}
```

`:nth-child(n)`

Za pomocą tej pseudoklasy określamy wygląd wszystkich wierszy oprócz pierwszego. Możemy dzięki temu definiować wygląd co drugi lub co trzeci wiersz lub kolumnę. Uzyskujemy to pisząc równanie w nawiasie. Na przykład:

```css
tr:nth:child(2n) {
background: #ccc;
}
```

W tym wypadku uzyskamy co drugi wiersz koloru szarego. Liczba n jest licznikiem, czyli kolejnymi liczbami naturalnymi, które po wykonaniu działania w nawiasie dają numery wierszy których wygląd będziemy określać.

```css
tr:nth:child(3n) {
    background: #bbb;
}
tr:nth:child(3n+1) {
    background: #ddd;
}
tr:nth:child(3n+2) {
    background: #fff;
}
```

Efekt:

Możemy też stosować słowa kluczowe even i odd. Pierwsze z nich jest równoznaczne z formułą 2n, natomiast drugie z 2n+1.

```css
tr:nth-child(even) {
    background: #CCC
}

tr:nth-child(odd) {
    background: #FFF
}
```


CIEKAWOSTKA

Tych pseudoklas możesz używać także do innych znaczników.

Znaczniki `<thead>`, `<tbody>`, `<tfoot>`

Służą one do grupowania wierszy w nagłówek, treść i stopkę. Same nie zmieniają wyglądu. Do tego używamy css'a.

```html
<thead>
    <tr>
        <td>Imię</td>
        <td>Nazwisko</td>
        <td>Adres</td>
        <td>Miasto</td>
        <td>Miejsce urodzenia</td>
        <td>Telefon</td>
    </tr>
</thead>
<tbody>
    <tr>
        <td>Krzysztof</td>
        <td>Najber</td>
        <td>Kwiatowa 4</td>
        <td>Żywiec</td>
        <td>Bielsko-Biała</td>
        <td>698731233</td>
    </tr>
    ...
</tbody>
<tfoot>
    <tr>
        <td class="scalone">Copyright by superStudio 2016</td>
```

```
    </tr>
</tfoot>
```

CSS

```
thead {
    color:#eee;
    background:#ccc;
}
tbody {
    color:black;
}
tfoot{
    background: yellow;
}
```

Scalanie wierszy i kolumn (`rowspan` i `colspan`)

Są to atrybuty, które pozwalają nam scalać wiersze (`rowspan`) i kolumny (`colspan`). Używając tego atrybutu definiujemy tak naprawdę ile komórek w wierszu lub kolumnie ma zająć dana komórka.

HTML

```
<tr>
    <td>Krzysztof</td>
    <td>Najber</td>
    <td>Kwiatowa 4</td>
    <td>Żywiec</td>
    <td>Bielsko-Biała</td>
    <td>698731233</td>
</tr>
<tr>
    <td>Piotr</td>
    <td>Świerszcz</td>
    <td>Polna 2</td>
    <td colspan="2">Żywiec</td>
    <td>687234134</td>
</tr>
<tr>
    <td>Aneta</td>
    <td rowspan="2">Ryś</td>
    <td>Bogata 8</td>
    <td>Pietrzykowice</td>
    <td rowspan="2">Bielsko-Biała</td>
    <td>602345624</td>
</tr>
```

```
<tr>
    <td>Maria</td>
    <td>Biedna 10</td>
    <td>Łodygowice</td>
    <td>676937154</td>
</tr>
<tr>
    <td>Jakub</td>
    <td>Kowal</td>
    <td>Rzemieślnicza 17</td>
    <td colspan="2">Bielsko-Biała</td>
    <td>507094726</td>
</tr>
```

Formularze

11

- `<form>` (action, method)
- input (type, name)
- input (size, maxlength, placeholder)
- typy inputów (text, password)
- textarea
- radio, checkbox
- atrybuty checked, disabled
- select, option, optgroup, atrybut selected
- input file
- button, input button
- label (for)
- fieldset
- date input
- email, url input
- search input

Formularze są elementem języka pozwalającym na interreakcję z użytkownikiem. Umożliwiają one przesłanie danych przez użytkownika do wybranego przez nas miejsca.

Struktura

Formularze są definiowane przez tag `<form>` i jest praktycznie zawsze używany z atrybutami `action` i `method`. Pierwszy z nich określa co się stanie po wysłaniu danych, natomiast drugi metodę jakiej użyjemy. Należy pamiętać aby nie umieszczać formularza w formularzu.

```html
<form action="dane.php" method="post">
    (elementy formularza)
</form>
```

HTML

Przykład:

HTML

```
<div id="form">
       <form action="dane.php" method="post">
        <fieldset>
            <legend>Dane personalne</legend>
            <div>
                <p>Imię:</p>
                <input name="imie" type="text" class="tekst"/>
            </div>
            <div>
                <p>Nazwisko:</p>
                <input name="nazwisko" type="text" class="tekst"/>
            </div>
            <div>
                <p>Login:</p>
                <input name="login" type="text" class="tekst"/>
            </div>
            <div>
                <p>Hasło:</p>
                <input name="haslo" type="password" class="tekst"/>
            </div>
            <div>
                <p>E-mail:</p>
                <input name="email" type="email" class="tekst"
size="30" placeholder="name@domain.com"/>
            </div>
            <div>
                <p>Strona internetowa:</p>
                <input name="strona" type="url" class="tekst"/>
            </div>
            <div>
                <p>Data urodzenia:</p>
                <input name="data" type="date" class="tekst"/>
            </div>
            <div>
                <p>Płeć:</p>
                <input type="radio" name="plec" value="k" id="k"
checked/>
                <label for="k">Kobieta</label>
                <input type="radio" name="plec" value="m" id="m"/>
                <label for="m">Męczyzna</label>
                <input type="radio" name="plec" value="n"
id="n"disabled/>
                <label for="n">Niesprecyzowana</label>
            </div>
            <div>
                <p>Wykształcenie:</p>
                <select name="wyksztalcenie">
                    <optgroup label="Za niskie">
                        <option value="pods"
selected>Podstawowe</option>
```

```
                        <option value="gim">Gimnazjalne</option>
                    </optgroup>
                    <optgroup label="Wymagane">
                        <option value="zaszaw">Zasadnicze
zawodowe</option>
                        <option value="sr">Średnie</option>
                    </optgroup>
                    <optgroup label="Nadprogramowe">
                        <option value="wyz">Wyższe</option>
                    </optgroup>
                </select>
            </div>
            <div>
                <p>O mnie:</p>
                <textarea name="omnie" class="tekst" rows="6"
cols="50">Napisz coś o sobie..</textarea>
            </div>
            <div>
                <p>Zdjęcie:</p>
                <input type="file" name="zdjecie" accept="image/*">
            </div>
        </fieldset>
        <fieldset>
            <legend>Zgody</legend>
            <input type="checkbox" name="zgody" value="1"/>Oświadczam,
że zapoznałem i akceptuję regulamin.<br/>
            <input type="checkbox" name="zgody" value="2"/>Wyrażam
zgodę na przetwarzanie moich danych w celach
                    realizacji zamawianych usług.<br/>
            <input type="checkbox" name="zgody" value="3"/>Wyrażam
zgodę na przetwarzanie moich danych w celach
                    marketingowych.<br/>
            <input type="checkbox" name="zgody" value="4"/>Chcę
otrzymywać newsletter.<br/>
        </fieldset>
        <input name="wyslij" type="button" value="Wyślij"/>
    </form>
</div>
```

Efekt:

Atrybut `action`

Dzięki temu atrybutowi definiujemy czynność wykonaną po wysłaniu formularza. Zazwyczaj przekazujemy te dane w postaci zmiennych do plików .php, które odpowiadają za ich przetworzenie. Innym sposobem może być wysłanie ich e-mailem, jednak nie zadziała to jeśli użytkownik nie ma ustawionego klienta pocztowego.

```
Dane.php?imie=jan&nazwisko=nowak…
```

Atrybut `method`

Ten atrybut pozwala nam wybrać jedną z dwóch metod HTTP do przesłania danych. Pierwsza z nich – GET – przesyła dane w sposób jawny w postaci zmiennych.

```
Dane.php?imie=jan&nazwisko=nowak…
```

Natomiast POST, czyli druga z metod przesyła dane niejawnie (dane nie są widoczne pasku adresu). Używamy jej w momencie kiedy chcemy przesłać wrażliwe informacje (np. hasła) lub gdy są one bardzo obszerne.

Atrybut `name`

```HTML
<div>
    <p>Imię:</p>
    <input name="imie"/>
</div>
```

Aby poprawnie wysyłać dane każdy element formularza musi mieć swoją nazwę, która jest unikalna dla każdego pola. Pod tą nazwą zostaną wysłane dane.

Tag

```HTML
<div>
    <p>Imię:</p>
    <input name="imie" type="text"/>
</div>
```

Jest to najczęściej używany tag w formularzach. Pozwala nam wprowadzać różne typy danych, zaczynając od zwykłego tekstu, a kończąc na plikach z komputera. Forma danych przyjmowanych przez niego zależy od atrybutu `type`.

Typy inputów

text

```HTML
<div>
    <p>Imię:</p>
    <input name="imie" type="text" class="tekst"/>
</div>
```

Jest to domyślna wartość atrybutu `type`. Wpisany tekst jest jawny.

password

```HTML
<div>
    <p>Hasło:</p>
    <input name="haslo" type="password" class="tekst"/>
</div>
```

Wartość `password` zamienia wszystkie znaki w tekście na kropki. Dlatego najczęściej używana jest do wpisywania haseł. Musimy jednak pamiętać, że wysyłanie zawartości tego pola odbywa się już tekstem jawnym, dlatego jeśli użyjemy metody `GET` zawartość będzie widoczna w pasku adresu.

date

```HTML
<div>
    <p>Data urodzenia:</p>
    <input name="data" type="date" class="tekst"/>
</div>
```

Ten typ pozwoli nam zamienić zwykłe pole w przyjazne użytkownikowi pole do wpisania daty.

Efekt:

email i url

```HTML
<div>
    <p>E-mail:</p>
    <input name="email" type="email"/>
</div>
```

```HTML
<div>
    <p>Strona internetowa:</p>
    <input name="strona" type="url" class="tekst"/>
</div>
```

Dzięki tym wartościom atrybutu nasze pole będzie służyło do wpisania odpowiednio adresu e-mail oraz URL.

file

```HTML
<div>
    <p>Zdjęcie:</p>
    <input type="file" name="zdjecie" accept="image/*">
</div>
```

Pole `<input>` umożliwia nam także przesłanie plików. Uzyskujemy to podając wartość `file` w atrybucie `type`. Możemy dokładnie określić jaki rodzaj plików użytkownik będzie mógł przesłać. Określamy go za pomocą atrybutu `accept`. Wartość `image/*`, podana w przykładzie, pozwoli na przesłanie tylko obrazków. Możemy ją zastąpić wartością `video/*`, aby móc przesłać tylko filmy. Jeśli chcemy umożliwić przesłanie tylko danego rozszerzenia to podajemy je w wartości atrybutu poprzedzone kropką, np. `.jpg`.

Atrybuty inputów

size

```HTML
<input name="email" type="email" class="tekst" size="30" />
```

Atrybut `size` pozwala nam określić szerokość pola.

maxlenght

```HTML
<div>
    <p>Nr. telefonu:</p>
    +48 <input name="tel" type="tel" class="tekst" maxlength="9"
size="9"/>
</div>
```

Atrybut `maxlenght` pozwala nam określić maksymalną ilość znaków w danym polu.

placeholder

```HTML
<input name="email" type="email" class="tekst" size="30"
placeholder="name@domain.com"/>
```

Dzięki tem atrybutowi w polu pojawi się jego opis, który znika po wpisaniu pierwszego znaku.

Textarea

```HTML
<div>
    <p>O mnie:</p>
    <textarea name="omnie">Napisz coś o sobie..</textarea>
</div>
```

Innym, ale też przydatnym polem jest `<textarea>`. Umożliwia ona wpisanie dużej ilości tekstu o większej ilości tekstu (`<input>` pozwala wprowadzić dowolnie długi, ale tylko jeden wiersz). Dodatkowo użytkownik może sam zmieniać jego wielkość.

Atrybuty `rows` i `cols`

```HTML
<div>
    <p>O mnie:</p>
    <textarea name="omnie" rows="6" cols="50">Napisz coś
o sobie..</textarea>
</div>
```

Dzięki tym atrybutom możemy określić domyślny rozmiar pola. Jednak użytkownik nadal może dostosować ją do swoich potrzeb.

Radio

```HTML
<div>
    <p>Płeć:</p>
    <input type="radio" name="plec" value="k"/>Kobieta
    <input type="radio" name="plec" value="m"/>Mężczyzna
    <input type="radio" name="plec" value="n"/>Niesprecyzowana
</div>
```

Zdarzają się sytuację w których użytkownik może udzielić tylko jednej odpowiedzi na zadane pytanie. Aby to uzyskać stosujemy pola jednokrotnego wyboru. Definiujemy je za pomocą typu `radio`. Atrybut `value` definiuje co zostanie wysłane jako wartość zmiennej.

Checkbox

```HTML
<input type="checkbox" name="zgody" value="1"/>Oświadczam, że
zapoznałem i akceptuję regulamin.<br/>
<input type="checkbox" name="zgody" value="2"/>Wyrażam zgodę na
przetwarzanie moich danych w celach realizacji zamawianych usług.<br/>
<input type="checkbox" name="zgody" value="3"/>Wyrażam zgodę na
przetwarzanie moich danych w celach marketingowych.<br/>
<input type="checkbox" name="zgody" value="4"/>Chcę otrzymywać
newsletter.<br/>
```

Checkbox'y, podobnie jak pola jednokrotnego wyboru, definiują nam jakich odpowiedzi możemy udzielić na zadane pytanie. W odróżnieniu od typu `radio` umożliwiają wielokrotny wybór.

Atrybuty `checked` i `disabled`

```HTML
<div>
    <p>Płeć:</p>
    <input type="radio" name="plec" value="k" id="k" checked/>
    <input type="radio" name="plec" value="m" id="m"/>
```

```
        <input type="radio" name="plec" value="n" id="n"disabled/>
</div>
```

Zarówno do typu `radio` , jak i `checkbox` możemy stosować atrybuty `checked` i `disabled` . Pierwszy z nich określa, że dane pole będzie domyślnie zaznaczone, natomiast drugi pozwala wyłączyć możliwość zaznaczenia danego pola.

Select

HTML

```
<div>
    <p>Wykształcenie:</p>
    <select name="wyksztalcenie">
        <option value="pods">Podstawowe</option>
        <option value="gim">Gimnazjalne</option>
        <option value="zaszaw">Zasadnicze zawodowe</option>
        <option value="sr">Średnie</option>
        <option value="wyz">Wyższe</option>
    </select>
</div>
```

Znacznik `<select>` pozwala nam stworzyć listę wybieraną. Jest to przydatne w momencie, w którym potrzebowalibyśmy stworzyć dużą ilość przycisków `radio` przez co nasza strona byłaby nieprzejrzysta. Tag `<select>` definiuje listę natomiast znaczniki `<option>` jej elementy.

Optgroup

HTML

```
<div>
    <p>Wykształcenie:</p>
    <select name="wyksztalcenie">
        <optgroup label="Za niskie">
            <option value="pods">Podstawowe</option>
            <option value="gim">Gimnazjalne</option>
        </optgroup>
        <optgroup label="Wymagane">
            <option value="zaszaw">Zasadnicze zawodowe</option>
            <option value="sr">Średnie</option>
        </optgroup>
        <optgroup label="Nadprogramowe">
            <option value="wyz">Wyższe</option>
```

```
        </optgroup>
    </select>
</div>
```

Bardzo przydatnym znacznikiem przy listach wybieranych jest `<optgroup>`. Możemy dzięki niemu pogrupować odpowiedzi.

Atrybut `selected`

```
<div>
    <p>Wykształcenie:</p>
    <select name="wyksztalcenie">
        <optgroup label="Za niskie">
            <option value="pods" selected>Podstawowe</option>
            <option value="gim">Gimnazjalne</option>
        </optgroup>
        <optgroup label="Wymagane">
            <option value="zaszaw">Zasadnicze zawodowe</option>
            <option value="sr">Średnie</option>
        </optgroup>
        <optgroup label="Nadprogramowe">
            <option value="wyz">Wyższe</option>
        </optgroup>
    </select>
</div>
```
HTML

Podobnie jak przy radiach i checkboxach możemy tu wybrać który element listy ma być domyślnie wybrany. Służy do tego atrybut `selected`.

Fieldset

```
<fieldset>
        <legend>Zgody</legend>
        <input type="checkbox" name="zgody" value="1"/>Oświadczam, że
zapoznałem i akceptuję regulamin.<br/>
        <input type="checkbox" name="zgody" value="2"/>Wyrażam zgodę
na przetwarzanie moich danych w celach realizacji zamawianych
usług.<br/>
        <input type="checkbox" name="zgody" value="3"/>Wyrażam zgodę
na przetwarzanie moich danych w celach marketingowych.<br/>
    <input type="checkbox" name="zgody" value="4"/>Chcę otrzymywać
```
HTML

```
newsletter.<br/>
</fieldset>
```

Znacznik `<fieldset>` pozwala nam logicznie grupować elemnty formularza. Elementy w środku znacznika zostaną otoczone ramką. Możemy całej grupie nadać też nazwę za pomocą tagu `<legend>`.

Label

```HTML
<div>
    <p>Płeć:</p>
    <input type="radio" name="plec" value="k" id="k" checked/>
        <label for="k">Kobieta</label>
    <input type="radio" name="plec" value="m" id="m"/>
    <label for="m">Męczyzna</label>
    <input type="radio" name="plec" value="n" id="n"disabled/>
    <label for="n">Niesprecyzowana</label>
</div>
```

Każdemu z pól możemy przypisać etykietę. Jest ona definiowana przez znacznik `<label>`. Aby poprawnie z niej korzystać musimy przypisać elementom atrybut `id`. Aby dopasować etykiety do pól używamy atrybutu `for` który przyjmuje taką samą wartość jak id pola do którego chcemy przypisać etykietę.

Button

```HTML
<input name="wyslij" type="button" value="Wyślij"/>
```

Aby umożliwić użytkownikowi przesłanie formularza definiujemy przycisk przez ustawienie typu pola `<input>` na `button`.

Audio i wideo

12

Przyszła teraz kolej, aby przedstawić Ci sposób na zamieszczenie na stronie bardziej złożonych multimediów niż obrazy, czyli audio i wideo. Różnica pomiędzy nimi jest niewielka, toteż nie powinno stanowić to dla Ciebie żadnego problemu.

Znacznik `<source>`

Fundamentalne znaczenie przy zamieszczaniu tego typu treści ma znacznik `<source>`. Jest on powiązany z `<audio>` i `<video>`, nie posiada swojego odpowiednika zamykającego.

Najważniejszy jego atrybut to `src`, którego wartością będzie link lub ścieżka dostępu do żądanego audio lub wideo. Kolejnym - `type` - określa rodzaj medium i jego format, np. `type=”audio/mp3”` albo `type=”video/mp4”`. Do określenia typu urządzeń, na które plik jest przeznaczony używamy `media`, np. `media=”screen and (min-width:555px)”`, `media="handheld not (grid:0)"` czy też `media="tv and (scan:interlace)"`.

Audio

```
<audio controls="controls" loop="loop" autoplay="autoplay"
preload="preload">
<source src="/music.mp3" type="audio/mpeg" />
<source src="/music.wav" type="audio/wav" />
Twoja przeglądarka ma kłopoty z audio
</audio>
```

HTML

Do znacznika `<audio>` możemy dopisać kolejne atrybuty, jednak lepiej zrobić to w skrócony sposób. Powyższy przykład jest równoważny z:

```
<audio controls loop autoplay preload>
<source src="/music.mp3" type="audio/mpeg" />
<source src="/music.wav" type="audio/wav" />
Twoja przeglądarka ma kłopoty z audio
</audio>
```

HTML

Atrybut `controls` powoduje wyświetlanie interfejsu użytkownika, `loop` odtwarza dźwięk w pętli, `autoplay` automatycznie rozpoczyna odtwarzanie po załadowaniu strony, a `preload` powoduje wstępne załadowanie pliku przed podjęciem decyzji użytkownika o jego odtworzeniu.

Znacznik `<audio>` zawiera w sobie znaczniki `<source>`. Jeżeli pierwszy z nich z jakiegoś powodu nie może być odtworzony przez przeglądarkę, to brany pod uwagę będzie następny. Jest to alternatywa na wypadek sytuacji, w której przeglądarka nie obsługuje niektórych plików audio. Jeżeli jednak żaden element `<source>` nie został poprawnie odczytany, to możemy na samym końcu wyświetlić tekst z informacją o kłopotach.

UWAGA!

W tej chwili jedynymi obsługiwanymi plikami audio są: mp3, wav (bez Internet Explorera) i ogg (bez Internet Explorera i Safari).

Wideo

HTML

```
<video controls muted poster="http://lo5.bielsko.pl/public/news/
attachments/55686a9ff2fc6LOGO_Of.Fe.jpg" width="450" height="300" >
<source src="/movie.mp4" type="video/mp4">
Twoja przeglądarka ma kłopoty z wideo.
</video>
```

Znacznik `<video>` ma dokładnie takie same właściwości jak `<audio>`, więc `<source>`, powiadomienia o błędach oraz atrybuty `controls`, `loop`, `autoplay`, `preload` działają w ten sam sposób.

Nowością jest jedynie atrybut `muted` powodujący, że już w trakcie ładowania filmu ścieżka audio będzie wyciszona, atrybut `poster`, którego wartością jest ścieżka do obrazka, który będzie wyświetlony przed odtworzeniem filmu, oraz oczywiście wymiary `height` i `width`.

UWAGA!

W tej chwili jedynymi obsługiwanymi plikami wideo są: mp4, webm (bez Internet Explorera i Safari) i ogg (też bez Internet Explorera i Safari).

Zwróć uwagę na rozmiar plików audio i wideo, spowolniający działanie stron internetowych. O ile wczytywanie obrazów nie zajmuje zbyt dużo czasu, multimedia często warto zastąpić najprostszymi odnośnikami do nich.

Zaawansowane elementy

13

- iframe
- script

Struktura strony

14

HTML5 wprowadza nowy zestaw elementów, które pozwalają szczegółowo określić strukturę strony. Pokażemy je teraz (z niektórymi być może już spotkałeś się wcześniej), gdyż potrafisz już zbudować podstawową strukturę strony i będzie Ci łatwiej je zrozumieć. Tagi które poznasz, będą odgrywać ważną rolę w budowaniu kolejnych stron w przyszłości.

W tym rozdziale nauczysz się:

- nowych elementów jakie oferuje HTML5,
- poznasz w jaki sposób mogą tworzyć alternatywę dla elementu `<div>`,
- jak nauczyć stare przeglądarki obsługi tych elementów.

Tradycyjny szablon HTML

Przez długie lata autorzy stron internetowych wykorzystywali tagi `<div>` do grupowania poszczególnych elementów strony. Takie elementy jak nagłówek, artykuł, stopka czy menu boczne były rozróżniane za pomocą klas i identyfikatorów umieszczonych wewnątrz tagu. Zobaczmy na poniższy przykład:

Jest to klasyczny przykład układu strony, np. bloga. Na samej górze strony (`#page`) znajduje się nagłówek (`#header`), który najczęściej zawiera takie informacje jak logo, nazwa firmy oraz nawigacja (`#nav`). Poniżej znajdziemy dwie kolumny: główna treść (`#main`) w skład której wchodzi kilka artykułów (`.article`) oraz po prawej menu boczne (`#sidebar`). Strona jest zakończona stopką (`#footer`).

Tak zbudowany szablon jest bardzo zrozumiały dla człowieka i nie stwarza większych problemów podczas wprowadzania poprawek, czy rozbudowywania strony. Ma jednak zasadniczą wadę: nie jest całkowicie zrozumiały dla przeglądarki, czy wyszukiwarki internetowej. Jak wiemy z poprzednich rozdziałów `<div>` jest traktowany jako element neutralny. Przeglądarka wie i rozumie, że wewnątrz znajduje się jakaś treść, jednak nie wie dokładnie co jest w środku. Na szczęście wraz z wprowadzeniem HTML5 dodano nowe elementy, które pomogą tworzyć bardziej logiczną strukturę strony.

Nowe elementy szablonu w HTML5

HTML5 wprowadza nowy zestaw elementów, które pozwalają podzielić stronę na poszczególne części. Nazwy tych elementów jednoznacznie wskazują na rodzaj zawartości, jaka się wewnątrz nich znajduje. Zobaczmy jak będzie się prezentował przykładowy szablon:

Powyższy przykład ma identyczny wygląd jak ten z poprzedniego rozdziału, jednak jest zbudowany za pomocą nowych znaczników. Na przykład nagłówek został objęty za pomocą tagu `<header>`, główna nawigacja to `<nav>`, a poszczególne artykuły to `<article>`.

Wszystkie te elementy mają takie same właściwości jak `<div>`, jednak dodatkowo opisują co się znajduje wewnątrz nich. Przykładowo robot indeksujący naszą stronę kładzie większy nacisk na to co znajduję się wewnątrz artykułów (`<article>`), niż to co znajduje się w stopce strony (`<footer>`). Taki kod jest również łatwiejszy do modyfikowania w przyszłości.

Lista nowych elementów

Tag	Opis
`<header>`, `<footer>`	**Nagłówek** oraz **stopka** - mogą być użyte nie tylko do opisywania całej strony, ale również jako dodatki np. do `<article>`. Zazwyczaj dodaje się wtedy do nich takie informacje jak autor czy data dodania artykułu.
`<nav>`	**Nawigacja** - umieszczamy wewnątrz niego główne elementy nawigacyjne strony, takie jak menu, linki w stopce, itp.
`<article>`	**Artykuł** - określa sekcję reprezentującą pojedynczą część serwisu (np. artykuł, wywiad, itp.)
`<section>`	**Sekcja dokumentu** - to tematyczna grupa treści, zwykle zawierająca nagłówek. Przykładami sekcji mogą być: rozdziały książki lub działy na stronie
`<aside>`	**Wstawka** - oznacza sekcję na stronie, która jest tylko nieznacznie powiązana tematycznie z treścią elementu, w którym się znajduje.

Tag	Opis
`<main>`	**Główna treść** - za pomocą tego tagu oznaczamy główną część naszego serwisu, np. lista wiadomości na głównej stronie.

Jak stare przeglądarki rozumieją nowe elementy?

Przeglądarka, która nie została zaprojektowana do obsługi nowych elementów po prostu je zignoruje i w zależności od kontekstu potraktuje jako element liniowy albo blokowy. Jako, że wszystkie nowe elementy są traktowane jako substytut dla `<div>`, to żeby były wyświetlane tak samo jak on musimy w kodzie CSS dodać kilka linijek:

```css
header, section, footer, aside, nav, article, main {
  display: block;
}
```

Od teraz wszystko powinno działać jak należy.

UWAGA!

Stare wersje Internet Explorera nadal mają duże problemy z wyświetlaniem nowych elementów. Jeśli bardzo nam zależy na tym, aby i na nim działała nasza strona, musimy dodać poniższy fragment do sekcji `<head>`:

```html
<!--[if lt IE 9]>
<script src="http://html5shiv.googlecode.com/svn/trunk/
html5.js"></script>
<![endif]-->
```

Praktyczne informacje

15

- seo
- analitycs

Podsumowanie

16