MODUŁ 2: Separacja treści i formatowania w HTML i CSS

Cel modułu

W niniejszym module zapoznamy się z zagadnieniem przetwarzania dokumentu znacznikowego na przykładzie języka HTML. Ponieważ nie rozpatrujemy na razie zagadnień języka programowania manipulującego strukturą takiego dokumentu, ograniczymy tu nasze rozwiązania do kwestii określania formatowania dokumentu w sposób realizujący postulat separacji zagadnień (ang. *separation of concerns*). Zapoznamy się również z podstawami języka CSS oraz wyjaśnimy pojęcia kaskadowości. Ocenimy ewolucję języka HTML z punktu widzenia ww. separacji zagadnień oraz deklaratywności przetwarzania.

Efekty kształcenia

Niniejsza lekcja służy osiągnięciu następujących efektów kształcenia.

Wiedza

Po ukończonym kursie student / studentka:

- Charakteryzuje model danych wynikły z reguł składniowych oraz reguł przetwarzania dokumentu HTML,
- Objaśnia postulat separacji zagadnień na przykładzie definicji struktury i sposobu sformatowania strony WWW,
- Objaśnia mechanizm ewaluacji dokumentu HTML posiadającego definicję stylów,
- Wyjaśnia właściwość kaskadowości arkuszy stylów CSS,
- Wskazuje zalety i wady poszczególnych sposobów łączenia arkuszy CSS z dokumentem HTML,
- Definiuje pojęcia pseudo-klas i pseudo-atrybutów,
- Objaśnia wpływ wprowadzenia nowych znaczników HTML5 na sposób nanoszenia formatowań na dokument HTML.

Umiejętności

Po ukończonym kursie student / studentka:

- Stosuje do dokumentu WWW definicje stylów zewnętrzne, zanurzone i inline,
- Stosuje różnorodne selektory CSS w arkuszach stylów zewnętrznych i zanurzonych,
- Stosuje elementy-kontenery celem formatowania struktur wieloelementowych,
- Określa formatowania dokumentu obejmujące właściwości tekstu barw, obramowań i inne,
- Stosuje arkusze CSS celem określenia układu treści (*layout*) w dokumencie,
- Różnicuje definicje stylu zależnie od docelowego medium prezentacji dokumentu,
- Samodzielnie uzupełnia swoje umiejętności za pomocą dostępnych w WWW materiałów w języku polskim i angielskim.

Uzasadnienie

Języki HTML i CSS tworzą fundament warstwy prezentacyjnej witryn i aplikacji WWW. Rozwój tych języków ilustruje zarazem doskonalenie separacji zagadnień struktury treści i jej formatowania (tj. rozmieszczenia i wyglądu). Języki te stanowią zarazem sugestywny przykład roli standardów we wdrażaniu technologii Internetu. Istnieje wprawdzie szereg zaawansowanych narzędzi uwalniających

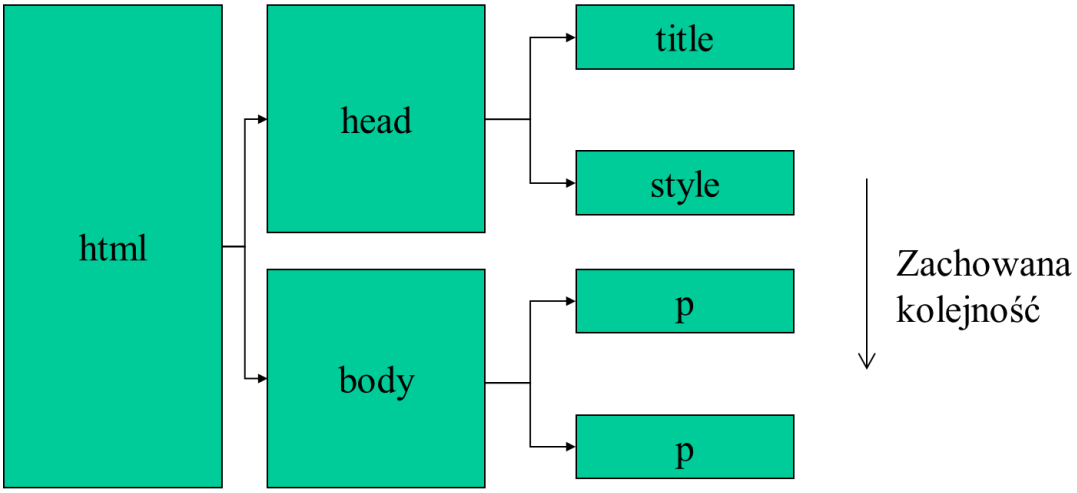
twórców stron WWW od konieczności budowania definicji stylów od zera, jednakże biegła znajomość języków HTML i CSS pozostają kluczowymi kompetencjami zarówno dla projektanta stron, jak też dla programisty aplikacji WWW.

Wprowadzenie

Stosując specyfikacje stylów stajemy przed problemem dostarczenia do struktury dokumentu odpowiednich informacji formatujących oraz przeliczenia tak powstałej struktury na postać wynikową prezentowaną użytkownikowi. Estetyka i elastyczność w obliczu przyszłych zmian dyktują tutaj dążenie do zdefiniowania wyglądu dokumentu za pomocą specyfikacji jak najlepiej odseparowanej od jego treści.

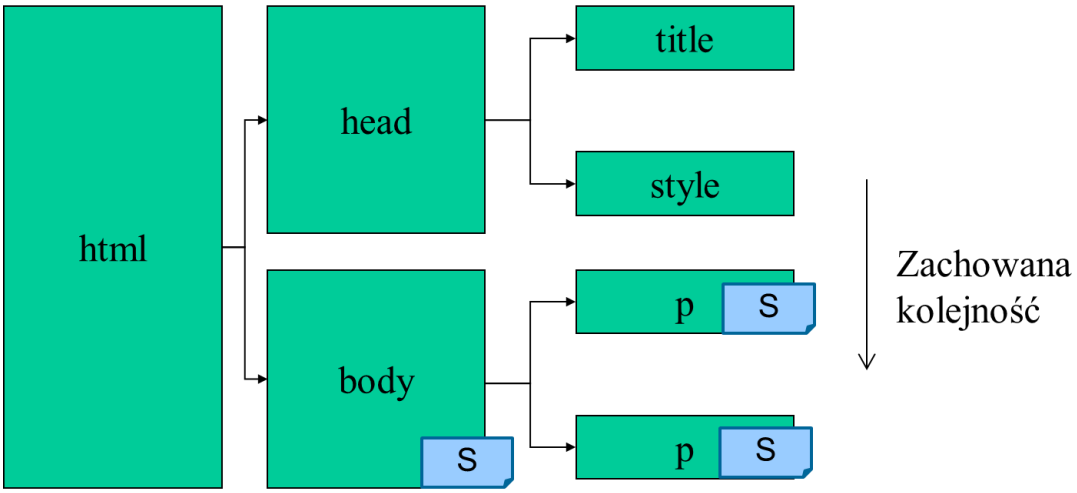
Możemy tu zatem wyróżnić 3 kroki:

1) Identyfikacja struktury dokumentu



Zgodnie z istotą dokumentu znacznikowego mamy tu do czynienia ze specyficznym wariantem hierarchicznego modelu danych, w którym węzły mogą mieć charakter elementów, atrybutów i zawartości tekstowej oraz zachowywana jest informacja o wzajemnej kolejności węzłów potomnych.

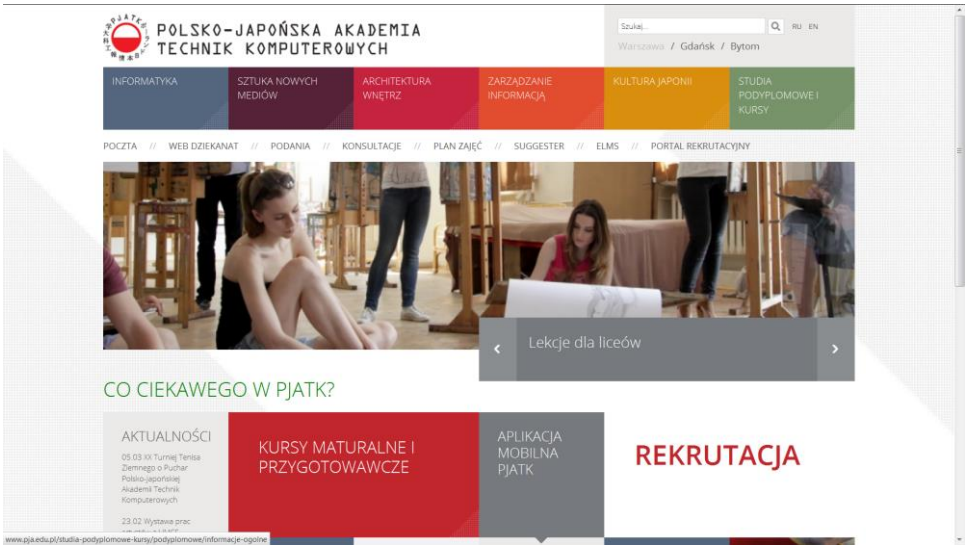
2) Naniesienie na tę strukturę informacji o formatowaniach



Celem podłączenia arkusza stylów jest przypisanie odpowiednim elementom przeznaczonych dlań poleceń formatujących. Można to zrobić w sposób bezpośredni – korzystając z atrybutów *style*

poszczególnych elementów, jednak naszym celem jest osiągnięcie bardziej wyrafinowanego, oszczędniejszego objętościowo i elastyczniejszego powiązania.

3) Przeliczenie struktury na widok prezentowany użytkownikowi.



Krok ten polega na wykorzystaniu ww. opisanej polecaniami formatującymi struktury i zbudowaniu na jej podstawie, oraz na podstawie wiedzy o środowisku i ograniczeniach danej przeglądarki, wynikowego wyglądu strony.

Kontrola bieżąca	( )	*komentarz odpowiedzi*
Tekst dokumentu wraz z naniesionymi nań znacznikami tworzy strukturę danych, którą możemy scharakteryzować jako:		
Sekwencja drzew elementów		Nie. Dokument znacznikowy musi mieć dokładnie jeden element-korzeń.
Zbiór drzew elementów, gdzie elementy potomne tworzą sekwencję		Nie. Dokument znacznikowy musi mieć dokładnie jeden element-korzeń.
Drzewo elementów, gdzie elementy potomne tworzą sekwencję	x	Tak. Zagnieżdżanie elementów daje strukturę drzewiastą, a kolejność podelementów może nieść informację.
Drzewo elementów, gdzie elementy potomne są wzajemnie nieuporządkowane		Nie. Kolejność, w jakiej umieszczono podelementy może nieść informację.

Sposób przyłączania arkuszy stylów

Zakładamy tutaj, że konstrukcja dokumentu HTML oraz semantyka jego znaczników jest nam już znana. Aby zatem wprowadzić formatowania, potrzebujemy poznać dwa dalsze zagadnienia:

1) Najważniejsze polecenia formatujących, które możemy zastosować do elementów,
2) Dostępne metody łączenia specyfikacji stylów z treścią dokumentu.

Polecenia formatujące

Przedstawienie pełniejszej listy poleceń formatujących przekracza ramy niniejszego dokumentu. Zamiast tego odsyłamy do licznych tutoriali i źródeł referencyjnych (np. www.w3schools.com). Ograniczmy się tutaj do zaznaczenia, że każde z poleceń składa się z etykiety właściwości i wartości:

właściwość: wartość

np. `color: green`

oraz, że w przypadku stosowania wartości wielowyrazowych potrzebujemy stosować cudzysłowy. Umieszczając zestaw poleceń formatujących musimy zadbać o ich separowanie średnikami.

Polecenia formatujące obejmują m.in. następujące kategorie:

- Definiowanie tła (własność *background*): grafika w tle, barwa tła;
- Tekst (własność *tekst*): barwa, odstępy, zmiana wielkości liter, ozdobniki, wcięcia, wyrównanie;
- Font (własność *font*): rodzaj fontu, rozmiar (procenty lub punkty), modyfikator (pochylenie itp.), wariant (np. wersaliki), rozciągnięcie tekstu;
- Rodzaje obramowań (właściwość *border*);
- Rozmiary marginesów elementu (właściwość *margin*);
- Marginesy komórek (właściwość *padding*);
- Wypunktowanie i numeracja (właściwość `list-style`);

Najprostszym sposobem sprawdzenia danego polecenia w działaniu jest jego bezpośrednie umieszczenie w atrybucie *style* wybranego elementu.

Powiązanie arkusza z dokumentem

Wyżej wspomniana metoda powiązania (zwana ***inline***), pomimo, że pozwala na wykorzystanie bogatych możliwości poleceń CSS, pod względem pielęgnacyjności kodu oraz generowanej objętości posiada te same bolączki co niesławne znaczniki formatujące (obecnie – *deprecated*) w dawniejszych wersjach HTML.

Optymalną jest konfiguracja, w której całość ustawień stylistycznych udaje się wyodrębnić do osobnego pliku. Nazywamy je stylami **zewnętrznymi**. Mogą być przyłączone umieszczonym wewnątrz elementu nagłówkowego elementem *link*:

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

Zaletą takiego połączenia jest potencjalna oszczędność na czasie ładowania stron i objętości przesyłanych danych. Jeśli mianowicie wiele stron naszej witryny korzysta z tego samego arkusza, to po pobraniu pierwszej z nich arkusz zostanie przez przeglądarkę zbuforowany i użyty dla przetwarzania kolejnych.

Gdy potrzebujemy określić style specyficznie dla jednej strony, możemy zastosować style **zanurzone**. Ich stosowanie polega na umieszczaniu deklaracji stylów wewnątrz elementu *style* w elemencie nagłówkowym strony – np.:

```
<style type="text/css">
h1 { color: orange; font-family: impact }
</style>
```

Można wreszcie połączyć oba podejścia używając klauzuli **`@import`** dotyczącej zewnętrznego arkusza, poprzedzającej deklaracje stylów zanurzonych:

```
<style type="text/css">
@import url(naszArkusz.css);
```

```
h1 { color: orange; font-family: impact }
</style>
```

Selektory i pojęcia z nimi związane

We wszystkich ww. wariantach (z wyjątkiem stylów *inline*), poleceniom formatującym musi towarzyszyć odwołanie do zadanych miejsc w strukturze formatowanego dokumentu. Odwołanie takie nazywamy *selektorem*.

Do najprostszych selektorów należy selektor uniwersalny (symbol *), wskazujący na element o dowolnej nazwie oraz selektor elementowy, stanowiący nazwę elementu (np. selektor h1 w powyższym przykładzie).

Możliwości odwołań wzbogacają obecne w strukturze dokumentu HTML atrybuty identyfikatora (id), klasy (class) oraz wszelkie inne atrybuty. Gdy potrzebujemy objąć formatowaniem fragment treści niepokrywającej się z zastaną strukturą dokumentu, możemy ją wzbogacić wplatająco dodatkowe elementy **`div`** (blokowe) lub **`span`** (wewnątrzliniowe), lub też wskazać podział struktury alternatywny względem zastanej struktury znaczników (elementy **`col`** i **`colgroup`**).

Podsumujmy składniki struktury dokumentu, których mogą się „chwycić” nasze selektory:

- Typ elementu (identyfikowany nazwą) – np.

```
<h1>…</h1>, <table>…</table>, <p>…</p>
```

- Klasa elementu (specjalny atrybut) – np.

```
<h1 class="pilny_news">…</h1>
```

- Identyfikator elementu (unikalny w dokumencie) – np.

```
<p id="notka_prawna">…</p>
```

- Specjalne znaczniki „opakowujące” + użycie atrybutu identyfikatora lub klasy – np.

```
<div id="notka_prawna">…</div>,
<span class="notka_prawna">…</span>
```

- Znaczniki elementów „poprzecznych” (podobnie) – np.

```
<col class="suma">…</col>
<colgroup class="kwartaly">…</colgroup>
```

Kontrola bieżąca	( )	*komentarz odpowiedzi*
Element języka CSS, w którym możemy odnaleźć pewne pokrewieństwa do języków zapytań to:		
Właściwość		Nie. Właściwość przyda nam się dopiero wtedy, gdy wpierw określimy, do których elementów ma się ona stosować.
Selektor	x	Tak. Selektor to rodzaj zapytania określający, co z treści dokumentu ma być przedmiotem podanych za nim instrukcji formatujących.
Wartość		Nie. Wartość przyda nam się dopiero wtedy, gdy wpierw określimy, do których elementów ma się ona stosować.
Kontener		Nie. Kontener jest elementem języka HTML. Stanowi część przeszukiwanej struktury.

Z podstawowych selektorów warto wspomnieć:

- Grupy selektorów – np. **`h1, h2, p`** (selekcjonują wszystkie elementy o wskazanych nazwach)
- Selektor elementów zagnieżdżonych – np. **`ul ol`** (dotyczy elementów `ol` zagnieżdżonych wewnątrz `ul`)
- Selektor klasy (mogą lecz nie muszą być poprzedzone nazwą elementu) – np. **`.pilne`**
- Selektor indentyfikatora (j.w.) – np. **`#bibliografia`**
- Selektor bezpośredniego zagnieżdżenia – np. **`div > p`** (tu oznacza akapit zagnieżdżony bezpośrednio w `div`)
- Selektor bezpośredniego sąsiedztwa – np. **`h1 + p`** (tu oznacza akapit występujący tuż po elemencie `h1`)

Obok treści oznaczonych explicite[1], selektory mogą się też odwołać do zawartości, którą można zidentyfikować implicyte, to znaczy na podstawie kontekstu dokumentu i/lub stanu dokumentu w oknie przeglądarki. Selektory takie nazywamy selektorami **pseudo-klas** i **pseudo-elementów**.

Kontrola bieżąca	( )	*komentarz odpowiedzi*
Selektor w postaci **`div ol`** selekcjonuje:		
Elementy `ol` zawarte wewnątrz elementów `div`	x	Tak.
Elementy `div` przypisane do klasy "`ol`"		Nie. W selektorach nazwy klas poprzedzamy kropką.
Element `div` mający identyfikator równy "`ol`"		Nie. W selektorach wartości identyfikatorów poprzedzamy symbolem `#`.
Elementy `div` oraz elementy `ol`		Nie. Chcąc zbudować sumę selektorów powinniśmy rozdzielić je znakiem przecinka.

Pseudo-klasą nazywamy nazwaną charakterystykę elementu HTML pozwalającą wyselekcjonować spełniające je elementy analogicznie, jak gdyby posiadały one odpowiedni atrybut `class`. Przykładami selektorów pseudo-klas są:

- **`:hover`** – kursor myszy nad elementem
- **`:first-child`** – pierwsze dziecko elementu
- **`:focus`** – element, który ma focus (jest aktywny)
- **`:link`** – nieodwiedzony link
- **`:visited`** – odwiedzony link
- **`:active`** – element aktywny

Pseudo-elementem nazywamy taką część treści dokumentu, którą można na podstawie zadanych kryteriów wyodrębnić, ale która nie jest jawnie wyznaczona znacznikami. Przykładami selektorów pseudo-elementów są:

[1] To znaczy dosłownie, wprost. Tutaj np. – przez podanie nazwy, klasy, albo identyfikatora elementu, który pragniemy wskazać.

- `:first-letter`
- `:first-line`
- `:before, :after`

  np. punktor przed tekstem każdego nagłówka 1 poziomu: `h1:before { content:url(icon1.gif); }`

Zauważmy, że obok oszczędności składniowej, walorem niektórych pseudoklas i pseudoelementów jest to, że przynależność doń może zmieniać się w trakcie interakcji użytkownika z dokumentem (np. poprzez zmianę szerokości okna, przesunięcie kursora itp.).

Kaskada i sposób wyznaczania wiążącej deklaracji

Dana część treści dokumentu może być przedmiotem formatowania przez różne, niekiedy konfliktujące ze sobą, deklaracje stylów. Specyfikacja standardu CSS zawiera reguły rozstrzygania takich konfliktów. Są one stosunkowo intuicyjne: deklaracja bardziej lokalna przesłania bardziej globalną, późniejsza wcześniejszą, a selektor bardziej szczegółowy waży więcej niż bardziej ogólny.

Ściślej rzecz ujmując, najważniejszym kryterium ważności definicji stylów jest ich źródło pochodzenia (przeglądarka, użytkownik, autor dokumentu) i ewentualna obecność flagi `!important`. Kryteria te wyznacza tzw. *porządek kaskady*, który można podsumować następująco.

1. Skompletuj deklaracje zgodne co do selektorów i typu medium
2. Sortuj deklaracje wg pochodzenia i flagi ważności `( !important )`
    1. Przeglądarka
    2. Użytkownik – normalne
    3. Autor – normalne
    4. Autor – ważne
    5. Użytkownik – ważne
3. Sortuj wewnątrz ww. grup według „specyficzności”
4. Sortuj równoważne według kolejności zdefiniowania.

Powyższe należy interpretować następująco: np. każda deklaracja z grupy 4 (autor – ważne) ustąpi wobec dotyczącej tej samej treści i właściwości deklaracji z grupy 5 (użytkownik – ważne[2]). Z kolei, jeśli mamy np. dwie różne deklaracje z grupy 3 (autor – normalne), to ważniejsza z nich będzie ta bardziej „specyficzna”.

[2] Odnajdź na WWW instrukcję, jak sporządzić i wprowadzić plik CSS ze stylami ważnymi użytkownika dla Twojej przeglądarki. Za jej pomocą możesz np. określić rodzaj i rozmiar czcionki dla elementów **p** w każdym odwiedzanym serwisie, albo np. tylko dla wskazanej witryny.

Na czym polega „specyficzność”? Zgodnie ze specyfikacją CSS 2.1 jest kalkulowana poprzez konkatenację[3] 4 liczb A-B-C-D („wygrywa” największa), gdzie poszczególne komponenty mają następujące kryteria.

A. = 1 gdy definicja z wartości atrybutu style; inaczej = 0
B. Liczba atrybutów **`id`** użytych (i związanych!) w selektorze
C. Liczba innych związanych atrybutów i pseudo-klas
D. Liczba nazw elementów i pseudo-elementów związanych w selektorze

Wynika stąd, że np. selektor posiadający jeden związany identyfikator będzie bardziej znaczący niż dowolnie złożony selektor oparty na klasach, pseudo-klasach czy innych atrybutach.

Wreszcie, pomiędzy równie specyficznymi deklaracjami „wygrywa” ta, która znajduje się później (niżej) w arkuszu (to właśnie dlatego, chcąc osiągnąć intuicyjne zachowanie kombinacji stylów wewnętrznych i importowanych, klauzulę `@import` umieszczamy na początku zawartości elementu `style`).

<<SCREENCAST S2 – wsparcie dla definiowania stylów i weryfikacji ich działania – wtyczka Firebug>>

Kontrola bieżąca	( )	*komentarz odpowiedzi*

Pewien akapit jest selekcjonowany przez selektory dwóch poniższych deklaracji. Odpowiedz, w jakim kolorze zostanie wyświetlony tekst tego akapitu.
`div.tresc p { color: green }`
`p#streszczenie { color: blue }`		

Zielony, bo selektor tej deklaracji zawiera więcej nazw.		Nie. Liczba związanych w selektorze nazw może odegrać rolę, ale liczą się też inne kryteria – m.in. rodzaje bytów, których ten nazwy dotyczą.
Zielony, bo uwzględniona zostanie deklaracja napotkana wcześniej.		Nie. Istnieją tutaj silniejsze kryteria niż kolejność deklaracji.
Czarnym (domyślnym dla przeglądarki), gdyż powyższe konfliktujące ze sobą deklaracje powodują błąd.		Nie. Taka rozbieżność deklaracji nie spowoduje anulowania obydwu.
Niebieski, bo selektor tej deklaracji zawiera więcej wartości atrybutu.	x	Tak. Obecność w selektorze atrybutu silnie zwiększa wagę danej deklaracji.

CSS a definiowanie układu strony

Podsumowaniem dość rozległego tematu zastosowania CSS do rozmieszczenia treści strony (*layout*), niech będą następujące podstawowe uwagi.

Elementy dokumentu dzielimy na blokowe (*block* - zajmujące domyślnie całą szerokość okna dokumentu) oraz wewnątrz-liniowe (*inline*).

[3] Pojęcie konkatenacji jest nam zapewne znane z podręczników języków programowania: oznacza wyrażenie łączące rezultaty jego podwyrażeń w pojedynczy łańcuch tekstowy. Mówiąc prościej - intencją autorów specyfikacji było tu po prostu to, by np. różnica w kryterium B pomiędzy dwoma selektorami była brana pod uwagę tylko wtedy, gdy selektory te mają identyczną wartość kryterium A itd. W innych dziedzinach podobną gradację kryteriów możemy odnaleźć np. w regułach wyłaniania kolejności drużyn w tabeli rozgrywek sportowych.

Domyślnie elementy są prezentowane zgodnie z ich kolejnością występowania wewnątrz dokumentu. O ile jednak poprzez CSS nie zmienimy struktury dokumentu, to możemy istotnie wpłynąć na zmianę tego sposobu rozmieszczenia – np.

- `visibility: hidden;` - spowoduje ukrycie danej treści (z pozostawieniem jednak rezerwowanej jej przestrzeni)
- `display: none;` - całkowite pominięcie danej treści przy prezentacji;
- zmiana pozycji elementu na inną niż domyślna – zob. niżej.

Składniki treści można pozycjonować w jeden z następujących trybów.

Właściwość `position`:

- `fixed` => względem okna przeglądarki
- `relative` => względem pozycji domyślnej
  - Pozostawia zarezerwowaną pozycję w normalnym układzie
  - Stosowany przy kontenerach tworzonych na elementy pozycjonowane w trybie `absolute`
- `absolute` => względem rodzica, który jest pozycjonowany nie-statycznie
- `static` => domyślna
- `inherited` => odziedziczona

Powinniśmy jednak być ostrożni przy określaniu rozmiarów i pozycji bezwzględnie, gdyż rozmiar okna przeglądarki znacznie odbiegający od zakładanego przez nas może istotnie pokrzyżować nasze plany.

Mamy też do dyspozycji właściwość `z-index` – określa względną kolejność w stosie pokrywających się elementów (np. element o wartości z-index równej -1 będzie mógł skryć się pod elementem mającym tę wartość równą 0).

Bardzo użyteczna, relatywnie prosta w stosowaniu i elastyczna jest właściwość `float`. Pozwala elementowi na otaczanie go przez inne. Wskazanie jako jej wartości `right` lub `left` spowoduje przyciągnięcie elementu do prawej lub lewej strony okna. Właściwość `clear` natomiast (dopuszczalne wartości to `left | right | both | none | inherit`) wskazuje kierunek z otoczenia elementu, gdzie nie dopuszcza on elementów pływających.

Aktualnie wspierane wersje specyfikacji CSS zapewniają bogate możliwości cieniowana, kolorowania, efektów gradientowych czy zaokrąglania kształtów. Tym samym zmniejsza się zapotrzebowanie na wplatanie do układu strony elementów graficznych (paski, przyciski itp.). Niemniej jednak technika polegająca na umieszczeniu w dokumencie pustych lub niepustych elementów **`div`** i wiązanie z nim pojedynczych lub powtarzalnych obrazów graficznych jako właściwość `background:url(`*<nazwa pliku graficznego>*`)` pozostaje ważnym sposobem określania wyglądu strony.

Kontrola bieżąca	( )	*komentarz odpowiedzi*
W dokumencie HTML mamy kolejno umieszczone akapity (elementy `p`) o zawartościach tekstowych: *pierwszy*, *drugi*, *trzeci*. Czy za pomocą CSS możemy sprawić, że zostaną wyświetlone w przeglądarce w odwrotnej kolejności?		

Nie. Kolejność elementów w dokumencie niesie informację i deklaracje CSS nie są władne jej zmieniać.		Odpowiedź błędna. Wprawdzie CSS nie modyfikuje struktury dokumentu HTML, ale sposobem jej wyświetlenia może wszechstronnie sterować.
Nie. Możemy co najwyżej ukryć wybrane akapity stosując dlań właściwość `display: none;`		Odpowiedź błędna. Naturalnie, możemy ukryć wybrane elementy w ten sposób, ale na tym nie kończą się możliwości oferowane przez CSS.
Tak. Możemy je w ten sposób rozmieścić za pomocą właściwości `position`.	x	Tak. Możemy swobodnie rozmieszczać elementy dokumentu niezależnie od ich kolejności występowania w źródłowym dokumencie HTML.

Aktualne wersje HTML a problem formatowania dokumentu

W niniejszym module interesuje nas jedynie niewielki wycinek przedsięwzięcia związanego z rozwojem języka HTML5. Nie nawiązujemy bowiem na razie do dwóch kluczowych czynników motywujących jego rozwój:

- Udogodnień adresowanych dla budowania aplikacji WWW opartych na technologiach działających po stronie serwera;
- Nowych interfejsów programistycznych oferowanych przez przeglądarki, przeznaczonych do wykorzystania przez język skryptowy działający w ramach strony po stronie klienta.

Ograniczmy się zatem na razie do oceny podejścia tego standardu do problemu separacji treści i struktury od formatowania.

Jak się wydaje, aktualna propozycja jest próbą poszukiwania kompromisu pomiędzy minimalnością języka a ustandaryzowaniem wsparcia powszechnie występujących w treści dokumentu elementów semantycznych.

HTML5 za punkt wyjścia traktuje zbiór elementów znany ze specyfikacji HTML 4.01 i XHTML 1.0. Jest to więc zestaw oczyszczony z porzuconych (dzięki wprowadzeniu CSS) znaczników czysto formatujących, ale z drugiej strony – nie aż tak purystycznie minimalny jak postulaty standardu XHTML 2.0 (który m.in. zakładał rezygnację z elementów **`h1`**–**`h6`** na rzecz jednego **`h`**, czy też usunięcie elementu **`a`**).

Jak zobaczyliśmy, ww. tradycyjny zbiór znaczników, w tym generyczne znaczniki „opakowujące” **`div`** oraz **`span`**, plus elementy **`col`** i **`colgroup`** wspierające formatowanie tabel, zapewniają, dzięki możliwości oznaczenia ich identyfikatorami, klasami czy innymi atrybutami, wszechstronne możliwości budowania selektorów CSS i przyłączania w ten sposób elementów formatujących.

Przykładami nowych elementów HTML5 są:

Elementy sekcyjne - zastępują generyczne **`div`** i **`span`** :

- `section` – wyodrębnia elementy treści
- `header` – nagłówek sekcji, artykułu, strony (intencja: treść wprowadzająca, hiperłącza)
- `footer` – stopka sekcji, artykułu, strony (intencja: informacje o autorstwie, prawach autorskich, kontakcie, mapa strony, zasoby powiązane)

    - `nav` – główny blok hiperłączy nawigacyjnych
    - `article` – samodzielny, kompletny składnik treści (np. wpis, komentarz, news)
    - `address` – dane kontaktowe
- Elementy blokowe:
    - `aside` – przypis boczny,
    - `figure` – samodzielny, niezależnie pozycjonowany składni,
    - `blockquote` – treść stanowiąca cytat
- Elementy interaktywne:
    - `dialog` – element dialogowy aplikacji
    - `summary` – streszczenie, nagłówek, legenda
    - `details` – w połączeniu z summary – pozwala zwijać szczegóły
    - `menu`, `command` – paski narzędzi i polecenia
- Elementy semantyczne:
    - `time` – data lub czas w postaci czytelnej maszynowo
    - `meter` – wartość skalarna, do wizualnej prezentacji

Z tego punktu widzenia, nowa grupa elementów HTML5 (niekiedy również łącznie zwana semantycznymi), może nie mieć krytycznego znaczenia. Jest jednak przydatna jako podstawa dla:

- Ułatwienia automatycznej analizy dokumentu przez oprogramowanie,
- Budowę predefiniowanych domyślnych sposobów prezentacji i obsługi poszczególnych elementów przez poszczególne platformy klienckie.

Powyższa lista elementów nie jest naturalnie wyczerpująca. Pośród nich są m.in. również elementy dedykowane interakcji aplikacji z użytkownikiem:

- `mark` – treść zaznaczona,
- `progress` – pasek postępu, kontrolowany programistycznie.

Do tych zagadnień powrócimy przy omawianiu rozwiązań programistycznych dla stron WWW.

Zadanie

Sporządź poprawny składniowo i strukturalnie dokument HTML5. Jest wskazane, aby dokument ten nawiązywał do aktualnie zakładanego wybranego tematu pracy zaliczeniowej i np. reprezentował stronę główną lub stronę „O nas” proponowanej witryny. Dokument powinien obejmować:

- Dokument HTML5.
- Zewnętrzną definicję stylów w postaci arkusza CSS.
- Przykłady formatowań nagłówków, paragrafów, tabeli, list wypunktowanych, numerowanych;
- Zastosowanie CSS dla uzyskania złożonego układu strony wykorzystującego obrazy (np. ozdobny motyw graficzny na całej szerokości górnej krawędzi strony, element blokowy z odnośnikami oraz element blokowy z główną treścią dokumentu umieszczone obok siebie, stopka dokumentu na całej szerokości).
- Struktura strony powinna obejmować: menu z hiperłączami, główną treść dokumentu, stopkę.
- Obraz.

Ponadto, niech ww. dokument określa odrębny sposób formatowania na potrzeby wydruku, w którym:

- Treść dokumentu będzie wypisana czcionką szeryfową **Times** lub podobną.
- Element nawigacyjny z hiperłączami nie będzie występował w wydruku.