pasja-informatyki.pl

Programowanie webowe

Mirosław Zelent

Odcinek #2

ten, w którym zbudujemy strukturę witryny
i zrozumiemy float:left i display:inline-block

Spis treści

Prolog

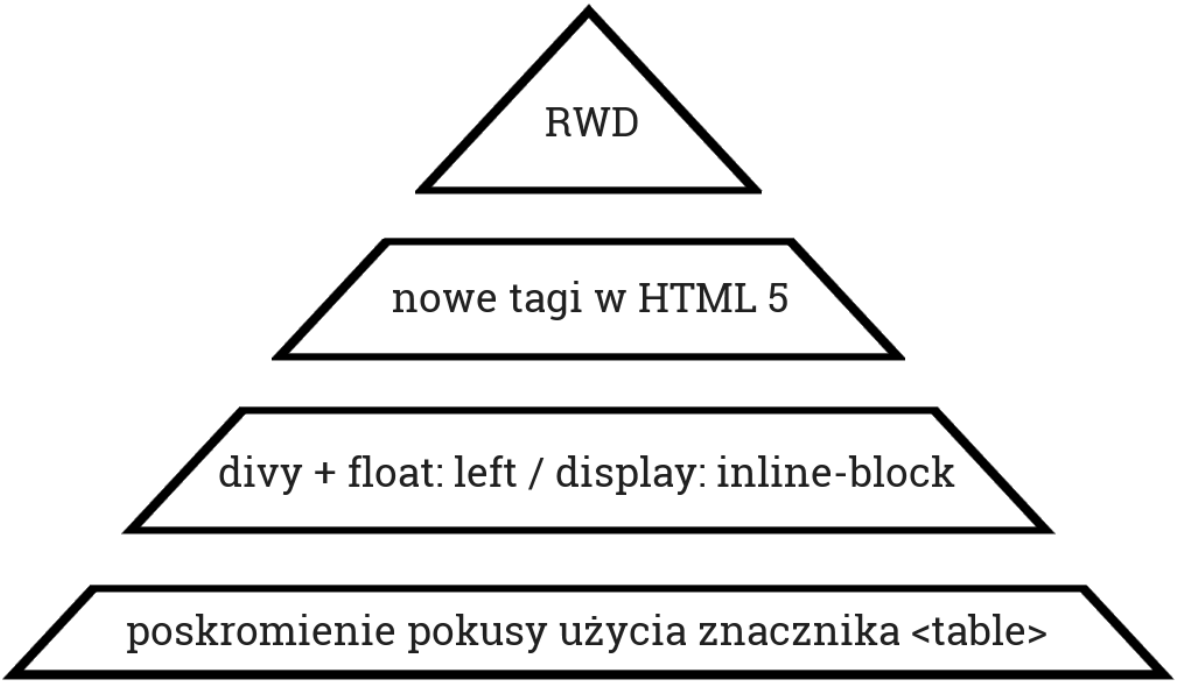
Po zapoznaniu się z sekcją <head> witryny, pora najwyższa nauczyć się tworzenia struktury ciała witryny w sekcji <body> dokumentu HTML. Naszym celem dzisiaj jest zrozumieć sposoby tworzenia klasycznej siatki elementów blokowych, tak aby podczas egzaminu móc stworzyć zadaną strukturę w ciągu zaledwie 5-7 minut. W praktyce jednak, dzisiejszy odcinek zaprosi nas do eksploracji całej gamy przeróżnych metod budowania szkieletów witryn internetowych - od tabel, przez divy, po nowe znaczniki strukturalne wprowadzone w HTML5 i responsywne siatki (RWD = ang. Responsive Web Design). Rozwiniemy także umiejętności miękkie, potrzebne do skutecznego komunikowania się w celu wymiany poglądów i doświadczeń zawodowych bez niepotrzebnych emocji, animozji, zachowań dogmatycznych, czy stawiania siebie w roli nieomylnego, zasiadającego w wygodnym fotelu eksperta, sędziego.

Przede wszystkim jednak, nadal zachowujemy podczas naszej pracy pierwotną, atawistyczną radość z uczenia się nowych umiejętności. Kiedy nauka wynika bardziej z naturalnej ciekawości i pasji, aniżeli z przymusu, rezultaty bywają zaskakująco dobre. To nasze umysły i ciała, jako systemy otwarte, naturalnie iskrzą nowymi połączeniami i asocjacjami, ucząc się i rozumiejąc coraz więcej, co też daje nam wiele radochy i satysfakcji. Bycie początkującym w jakiejkolwiek dziedzinie bardzo często wiąże się ze sporą dawką entuzjazmu i poczucia niczym nieskrępowanej, dziecięcej frajdy. Sztuką samą w sobie (którą niektórzy nazywają ścieżką własnego mistrzostwa) jest utrzymać reżim treningowy także kiedy ten początkowy entuzjazm siłą rzeczy się wyczerpie. Więcej o przemijającej motywacji, o długoterminowym charakterze treningu i sposobach skutecznej nauki napisałem w tym miejscu – polecam lekturę osobom zainteresowanym. A my bierzmy się do pracy!

Dlaczego nie budujemy struktury na tabelach?

Jak tworzymy strukturę witryny internetowej? Pozwólcie, że odpowiem na tak postawione pytanie parafrazując graficznie - na obrazku poniżej - piramidę potrzeb Abrahama Maslowa, zamieniając ją w piramidę sposobów tworzenia szkieletu sekcji body dokumentu HTML.

Przedstawiam naturalną ewolucję metod rozmieszczania elementów na wirtualnym płótnie przeglądarki internetowej:



Podstawę piramidy stanowi powstrzymanie się od intuicji użycia tabeli do planowania rozmieszczenia elementów witryny. Jakkolwiek samo stosowanie tabel to zdecydowanie w web developerce relikt przeszłości, to jednak ludzie po dziś dzień wykazują naturalną chęć użycia wierszy i kolumn do planowania ciała dokumentu HTML i to raczej nigdy się nie zmieni - czytamy gazety, książki, ulotki - jesteśmy przyzwyczajeni do kolumnowego rozkładu tekstu i grafik.

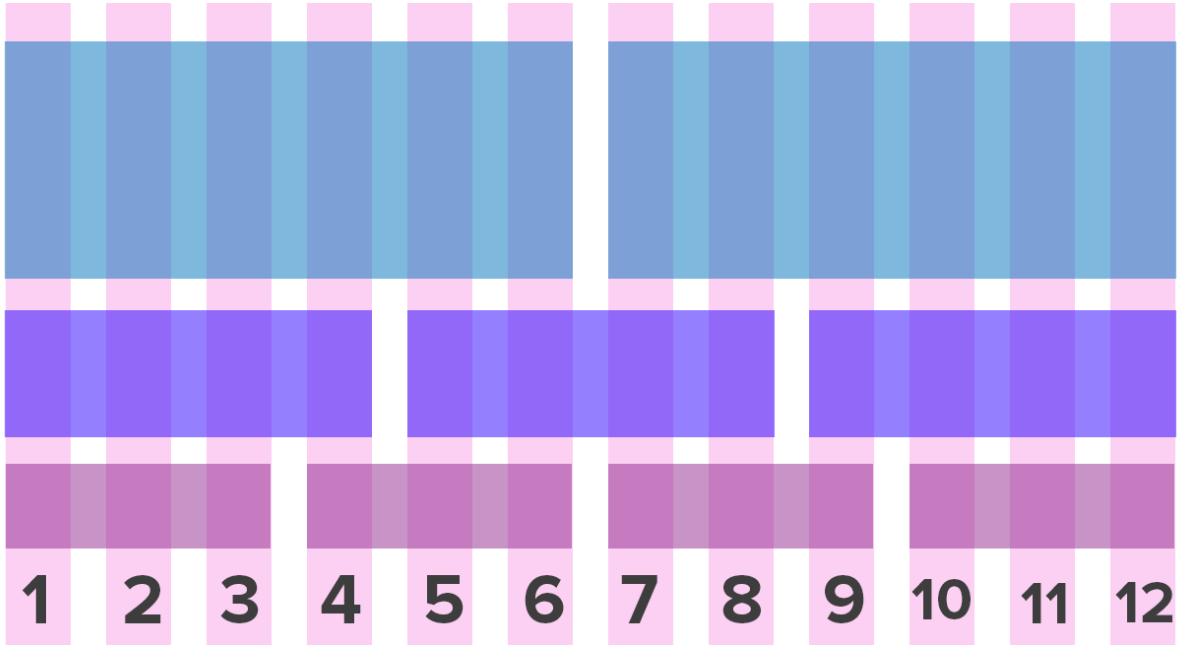
Tabela wydaje się być idealna - przy jej pomocy łatwo zorganizować sobie szpalty tekstu i na tej podstawie rozplanować content artykułu. Często tak zresztą postępujemy w edytorach typu Word, Libre Office Writer, WordPad. Arkusze kalkulacyjne (Excel, Libre Office Calc) także zachowują strukturę wierszy i kolumn, podobnie jak bazy danych, faktury VAT, potwierdzenia przelewu czy nawet paragony.

Nie należy więc ganić siebie (albo innej osoby) za intuicję użycia tabeli do rozmieszczenia zawartości witryny - często taką bezpardonową, wręcz prześmiewczą wobec innych manierę, wykazują młodzi entuzjaści tworzenia stron internetowych (którzy nauczyli się już dlaczego to podejście akceptowalne nie jest). Zapominają oni wówczas, iż sami także taką chęć na początkowym etapie nauki wykazywali, albo może pamiętają o tym w głębi podświadomości, ale chcąc podkreślić jak wiele już wiedzą o tworzeniu witryn, namiętnie piętnują innych za choćby zwykłe zadanie pytania o możliwość wykorzystania tabel.

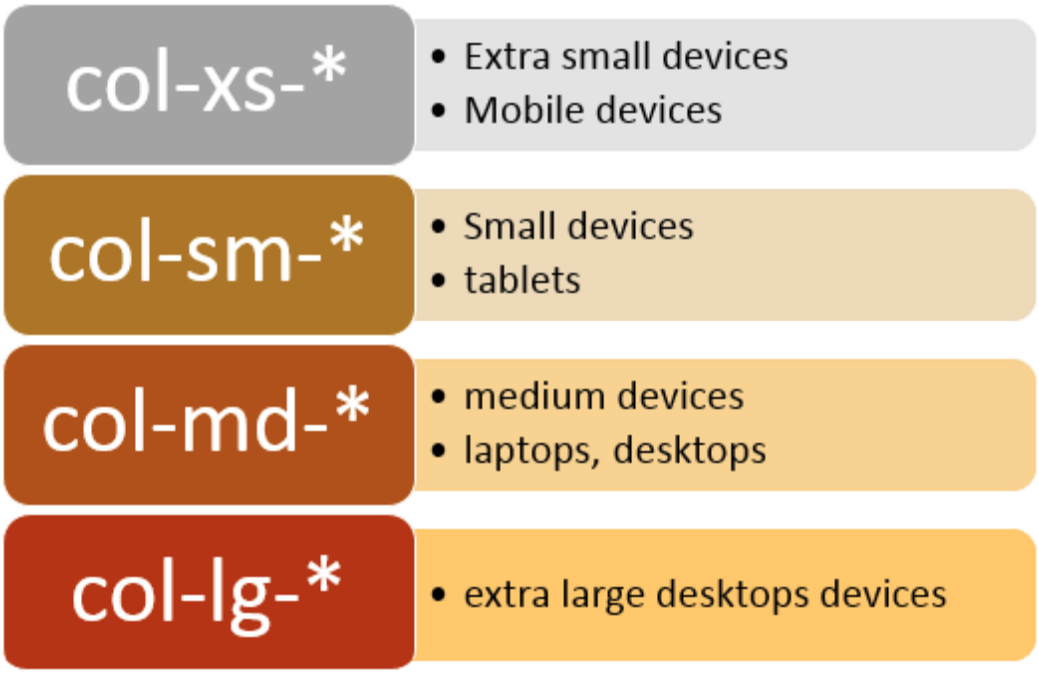
Dlaczego więc tabele nie nadają się do tworzenia struktury witryny? Poniżej zestawiłem siedem "grzechów głównych" stosowania w tym celu znacznika table:

1. Niepełne rozdzielenie zawartości witryny od opisu jej wyglądu (problemy ze stylizowaniem tabel w CSS).
2. Komórka w tabeli nie jest autonomicznym elementem takim jak blok (div albo nowy kontener w HTML5).
3. Tag <table> nie jest pomyślany jako znacznik strukturalny – ma służyć do tabelarycznego prezentowania informacji.
4. Stosowanie tabel skutkuje powstawaniem nadmiarowego kodu opisującego strukturę witryny.
5. Przeglądarka wolniej ładuje i renderuje zawartość strony, której szkielet zbudowano na tabeli.
6. Tabele nie są SEO friendly (ang. Search Engine Optimization - optymalizacja w wyszukiwarkach internetowych).
7. Szkielet tabelaryczny uniemożliwia zastosowanie responsywności (różnego układania bloków w zależności od aktualnego rozmiaru ekranu).

Samo posiadanie intuicji do stworzenia swoistej siatki wierszy i kolumn jest dobre i naturalne. Później zresztą, pracując już na stronach responsywnych, przekonasz się, iż takie myślenie tabelaryczne stało się także podstawą responsywnych frameworków, takich jak np. Bootstrap. W Bootstrapie posługujemy się siatką (tzw. gridem) divów (albo innych elementów blokowych znanych z HTML5) złożoną z maksymalnie dwunastu kolumn:



I oczywiście to nadal są struktury blokowe, a nie tabele, niemniej jednak to myślenie o stronie jako o siatce wierszy i kolumn, ewidentnie tam występuje. Dodatkowo, w Bootstrapie określamy także dostępne w witrynie, predefiniowane rozmiary ekranu:



W zależności od obecnego rozmiaru okna przeglądarki, witryna potrafi zatem odpowiednio dostosować swoją strukturę do zaistniałych okoliczności (z użyciem JavaScript i CSS). Trudno więc nazwać Bootstrapa dokładną realizacją tabelarycznej intuicji, niemniej jednak sama idea wierszowo-kolumnowej budowy szkieletu witryny, idea posiadania układu kartezjańskiego o dwunastu jednostkach, rzeczywiście w gridzie występuje. Zatem fakt, iż posiadamy taką „tabelaryczną" intuicję i że w pewien sposób kusi nas użyć znacznika <table> przestaje nas dziwić – w końcu ekran przeglądarki to płaszczyzna, a położenie elementu na płaszczyźnie opisujemy zazwyczaj dwiema współrzędnymi.

Trzeba jednak poskromić tę pokusę ze względu na poważne, negatywne implikacje użycia <table> jako fundamentu witryny. A najlepiej przekuć tę pokusę w miłość do autonomicznych elementów blokowych, które rzeczywiście lepiej nadają się do frywolnego bazgrania po wirtualnym płótnie okna przeglądarki. Więcej o wadach tabel w drugim odcinku serii HTML.

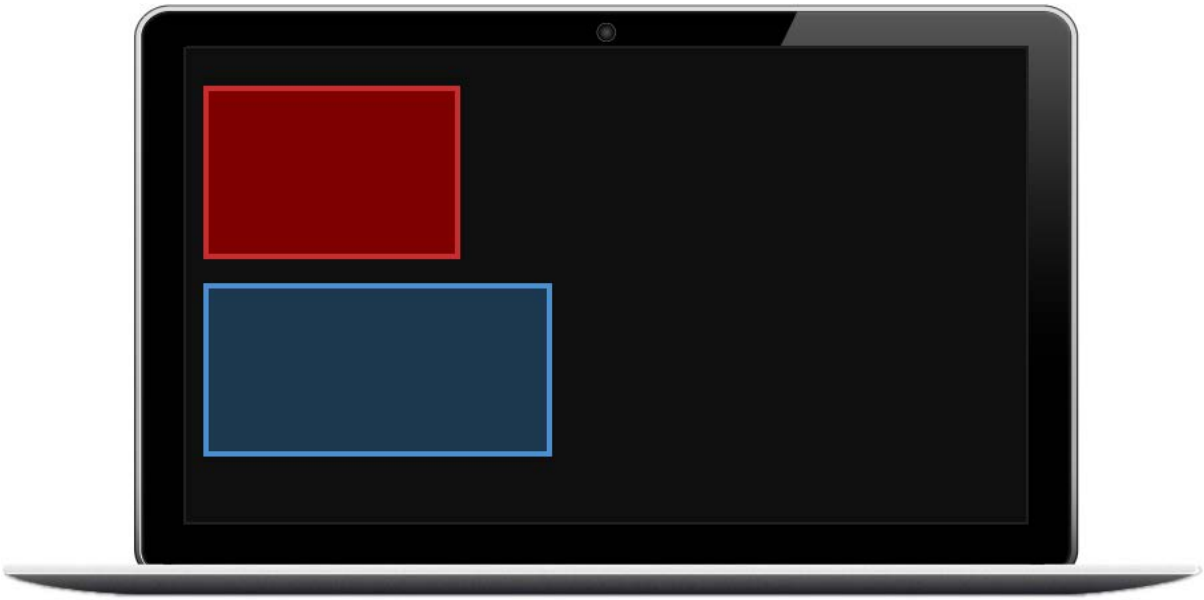
Elementy blokowe, domyślne zachowanie bloków

Do zbudowania szkieletu współczesnej witryny internetowej używamy tak zwanych **znaczników strukturalnych**. Klasycznym przykładem takiego znacznika jest (od ang.

divide = podział strony). Divy nazywane są też często elementami blokowymi, albo wprost - blokami. Element blokowy ma to do siebie, że zawsze układa się pod poprzedzającym go elementem. Na przykład: mamy w kodzie witryny dwa divy: czerwony i niebieski.

Nawet jeżeli ten pierwszy blok zajmuje jedynie 30% dostępnego na ekranie miejsca, to i tak następny, niebieski div, ułoży się domyślnie pod czerwonym. Pomimo, że zmieściłby się obok, to i tak ułoży się pod poprzednim - tak właśnie domyślnie zachowują się elementy blokowe:

```
<div class="czerwony"></div><div class="niebieski"></div>
```

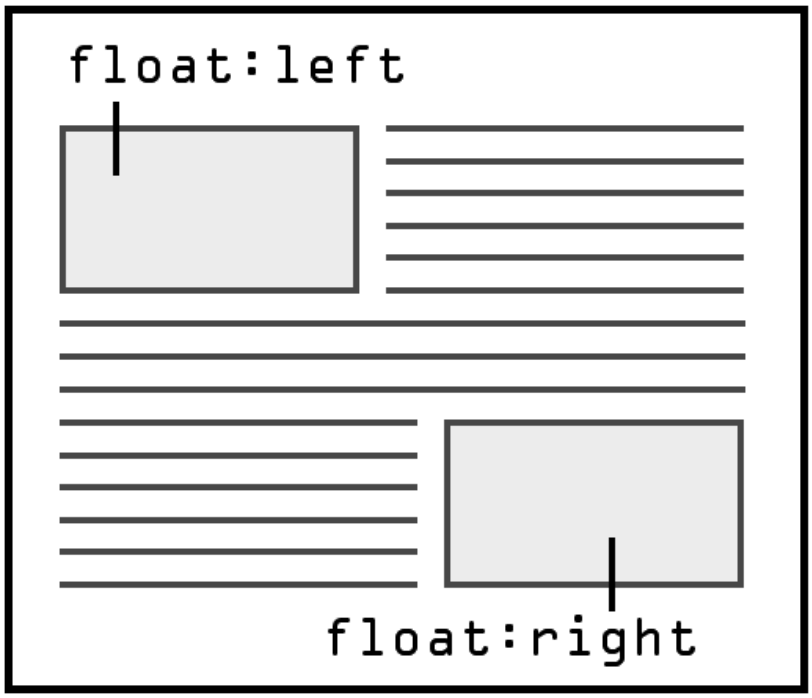


Oczywiście możemy wpłynąć na te bloki odpowiednimi zapisami w CSS, tak aby mogły układać się także obok siebie (pełniej zapełniając przestrzeń okna przeglądarki) i właśnie tego dziś się nauczymy. Nie można przecież skutecznie budować interfejsów witryn, wiedząc jedynie, jak układać bloki jeden pod drugim.

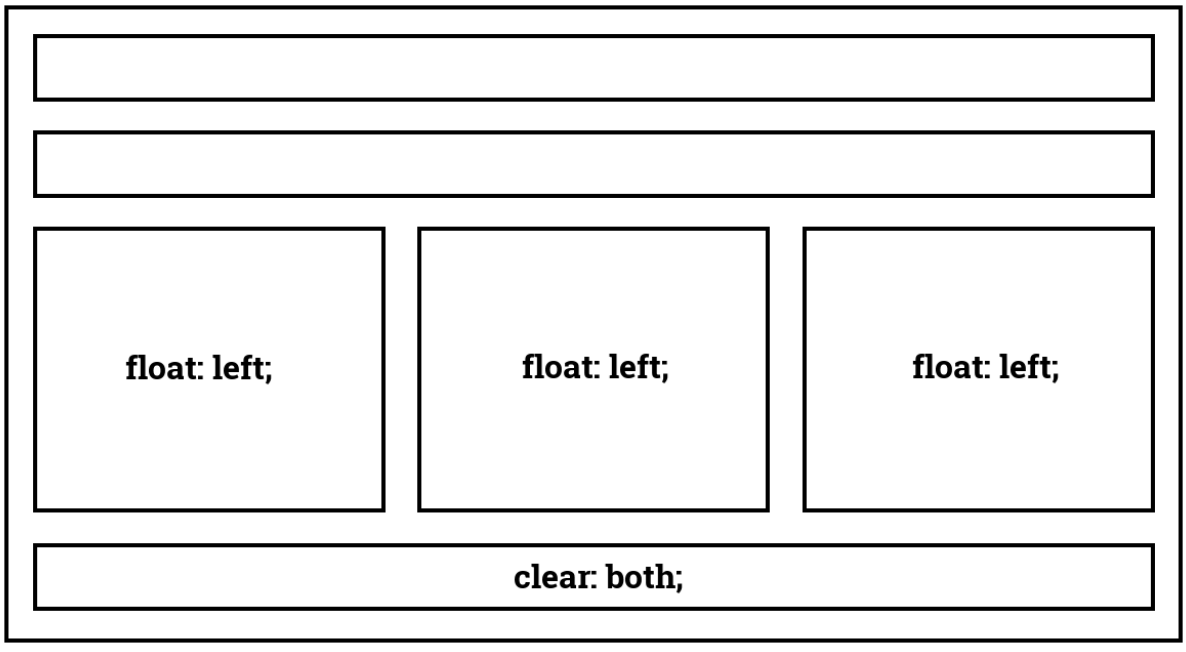
Istnieją dwie uznane metody układania bloków obok siebie – jedna wymaga użycia w stylach tzw. opływania float:left połączonego z jego wyczyszczeniem clear:both, zaś w drugiej używamy zapisu display: inline-block.

Metoda float:left w połączeniu z clear:both

Float to realizacja konceptu tzw. opływania elementu - znamy tę sztuczkę z Worda czy z innego edytora biurowego, w których można było ustawić opływanie wstawionych obrazów tekstem:



W praktyce metoda ta polega na tym, iż wszystkie bloki, które mają być ułożone obok siebie, będą posiadały w swoich klasach CSS dodatkowy atrybut float:left. A następny blok, który już nie ma się układać poziomo, ma posiadać zapis czyszczący opływanie: clear:both;



Atrybut float:left włączył opływanie do lewej strony dla każdego kolejnego bloku, stąd na końcu tego „piętra” trzeba nam jeszcze to opływanie wyłączyć – i dlatego też w stopce pojawił się zapis: clear:both. Oto cała filozofia – to wszystko co trzeba wiedzieć, aby opływania poprawnie użyć.

Zalety podejścia float:left

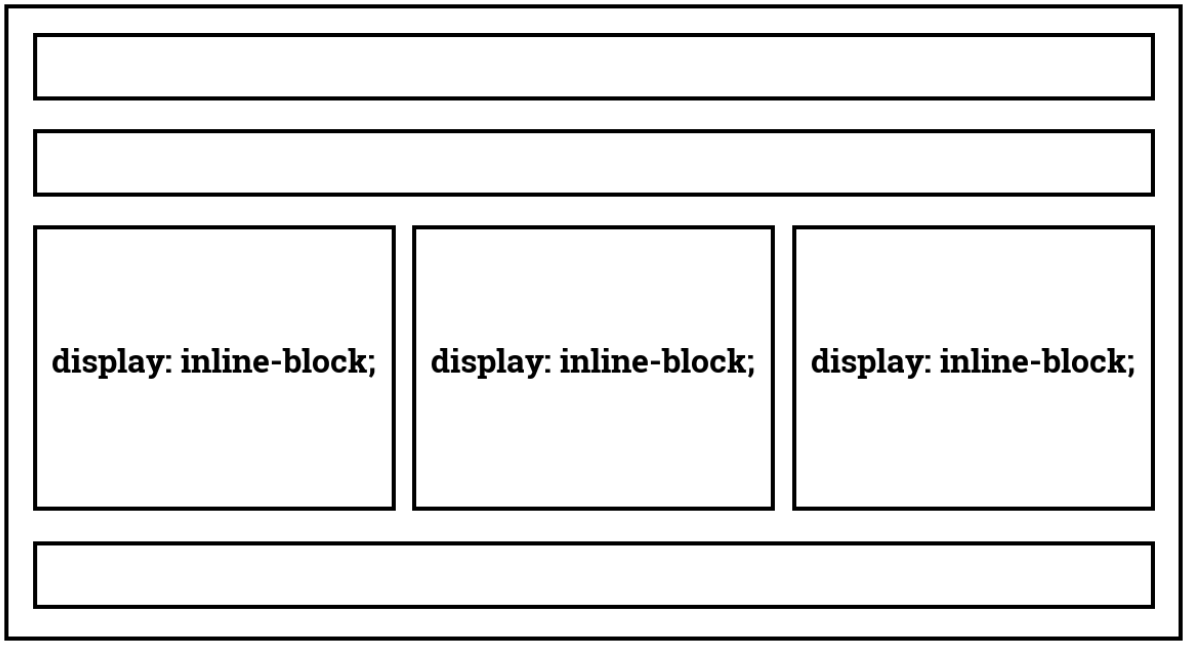
- Opływanie poprawnie zadziała we wszystkich przeglądarkach internetowych, nawet w IE 6/7 (wspaniała kompatybilność wsteczna);
- Brak jakichkolwiek problemów ze znakami białymi, które to problemy występują w konkurencyjnym podejściu display: inline-block (niewymagane są obejścia bądź hacki likwidujące problem znaków białych).

Wady podejścia float:left

- Zachodzi konieczność wyczyszczenia opływania zapisem clear:both w dodatkowym elemencie blokowym (choć można zapis clear:both umieścić w bloku przewidzianym w layoucie);
- Elementy blokowe z włączonym opływaniem zostają „wyjęte" z naturalnego flow witryny (i dlatego zachodzi konieczność wyłączenia opływania clear: both);
- Rozwiązanie to zostało opracowane na potrzeby opływania obrazów tekstem, i dopiero później zaadoptowane do układania z jego pomocą bloków strukturalnych.

Metoda display:inline-block

Atrybut display:inline-block wstawiamy do każdego bloku, który ma być wyświetlony w linii – oto cała filozofia tego rozwiązania. Nie zachodzi tutaj w ogóle potrzeba stosowania zapisu clear: both, gdyż nie „wyrywamy" bloków z domyślnego flow witryny:



Niestety w metodzie tej występują problemy z występowaniem znaków białych (odstępów poziomych) pomiędzy blokami ustawionymi w linii. Niewielkie pionowe odstępy (choć występują) nie stanowią taki wielkiego problemu jak „spacje" poziome.

Zalety podejścia display: inline-block

- Nie występuje potrzeba czyszczenia float:leftu z użyciem clear:both w dodatkowym elemencie blokowym;
- Bloki z ustawionym inline-blockiem „słuchają" ustawienia text-align:center kontenera, w którym się znajdują;
- Metoda nowsza, pomyślana jako alternatywa do „wyrywającego" bloki z flow witryny opływania.

Wady podejścia display: inline-block

- Metoda nie działa poprawnie w IE 6/7 (słabsza kompatybilność wsteczna),
- Problemy z występowaniem znaków białych (odstępów pomiędzy blokami) – w zależności od zastosowanej metody poradzenia sobie z tym problemem, wystąpią odmienne „skutki uboczne".

Metody rozwiązania problemu znaków białych w podejściu display: inline-block

- Usunięcie znaków białych w kodzie HTML (tracimy „wcięcia" = gorsza czytelność kodu);
- Ustawienie negatywnych marginesów -4px;
- Ustawienie font-size:0 kontenera oraz dodatniego font-size blokom wewnątrz (rodzi problemy z ustawianiem wielkości czcionek z użyciem em oraz %);
- Zmniejszenie szerokości bloków, tak aby uwzględnić istniejące odstępy;
- pominięcie zamykających tagów (wygląda ohydnie w kodzie);
- stosowanie komentarzy w HTML <!-- --> tak, aby „symulowały" one znaki końca linii.

Zestawienie metod można jeszcze samodzielnie poszerzyć googlując np. frazę: *display inline block remove space between divs*.

W mojej osobistej opinii, najlepszą metodą rozwiązania problemu znaków białych jest zwyczajne usunięcie ich z kodu HTML (spomiędzy divów ustawionych w linii).

Flexbox

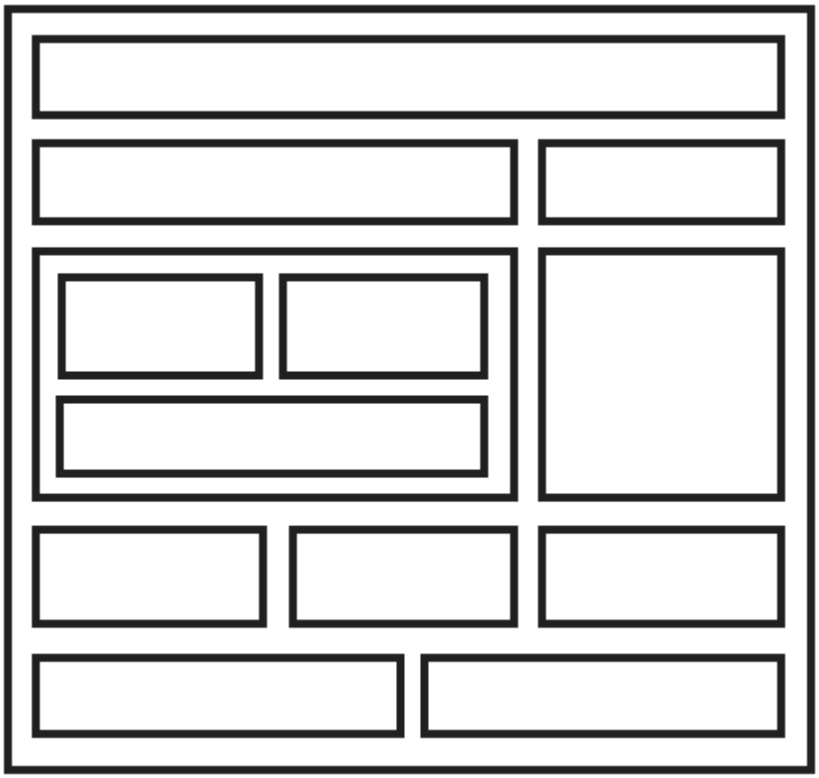
To nowy, alternatywny sposób tworzenia struktury witryny, który być może stanie się w

przyszłości pełnoprawnym standardem W3C. Aktualnie problemem (jak w przypadku wielu nowych rozwiązań) jest słaba kompatybilność wsteczna takiej struktury. Jednak stosowanie kontenerów zawierających zapisy *display: flex;* oraz *display: inline-flex;* wydaje się być na tyle ciekawym rozwiązaniem problemu tworzenia siatki bloków, iż warto już teraz przyjrzeć mu się bliżej. Zachęcam do wykonania samodzielnego researchu w sieci, zaś na sam początek warto zajrzeć tutaj.

Zadanie do samodzielnego wykonania

Na podstawie wiedzy z dzisiejszego odcinka kursu wykonaj przedstawioną poniżej strukturę divów – dokładne wymiary, proporcje, nazwy klas, jak i kolory bloków dobierz samodzielnie.

Uwaga: odstępy widoczne na schemacie są jedynie umowne (mają obrazować wzajmne położenie bloków) – w praktyce divy powinny być przyklejone do siebie bez żadnych odstępów. Wybierz swoją ulubioną metodę (float:left vs. display:inline-block) albo wykonaj dwie wersje layoutu.



Nic nie zastąpi samodzielnej pracy z kodem – to właśnie doświadczenia potrzebujesz na egzaminie! Podczas pracy popełnisz błędy, pogłówkujesz, naprawisz je – zastosujesz zdobytą wiedzę w praktyce. Nawet najlepszy wykład, książka czy film nie zastąpi samodzielnej pracy z

kodem – nie lekceważ więc zadania domowego i niech zawsze najważniejszym etapem nauki programowania będzie dla Ciebie cierpliwa praca w edytorze połączona ze śledzeniem poczynionych zmian w przeglądarce internetowej.

Umiejętności miękkie w dyskusji

Dyskusje poszerzają nasze horyzonty myślowe, pozwalają nam poznać inne punkty widzenia oraz przyjrzeć się danemu zagadnieniu z szerszej perspektywy. Jednak czasami któraś ze stron dyskusji może wykazywać cechy neurotyczne, zwłaszcza kiedy zachowuje się dogmatycznie, lub kiedy ustawia siebie w roli jedynego, niepodważalnego autorytetu i sędziego.

Najważniejsze w wartościowej dyskusji jest zatem odpowiednie podejście do partnera dialogu. Należy przede wszystkim okazać szacunek do rozmówcy i kierować się pozbawioną emocji, logiczną retoryką, zamiast dogmatyczną, autorytarną manierą.

Stworzenie przestrzeni do dialogu wymaga nieinwazyjności i sporego taktu, jednak właśnie takie dyskusje długo się pamięta i najwięcej można z nich wynieść.

Sztuka retoryki i erystyki to jedna z najtrudniejszych umiejętności miękkich do opanowania, jednak stwarza ona unikalne możliwości rozwijania swoich pasji oraz nawiązywania wartościowych kontaktów międzyludzkich. Jak to zwykle bywa, najcenniejsza jest codzienna praktyka, dlatego zapraszam do szlifowania swoich umiejętności na naszym forum.