**Div + Div**

Święty Graal początkującego **front end developera**, czyli opanowanie łączenia **div**‚ów w jeden spójny layout. Niektórzy powiadają, że zrozumienie tego tematu jest kluczem do opanowania sztuki tworzenia stron internetowych, jednako jego brak może prowadzić do poważnych kondycji zdrowotnych z nerwicą i [syndromem Tourette’a](http://pl.wikipedia.org/wiki/Zesp%C3%B3%C5%82_Tourette%E2%80%99a) na czele. Przed użyciem skonsultuj się z osobami w twoim najbliższym otoczeniu, aby na czas twej nauki zdążyli je opuścić. Gotowy czy nie, zaczynamy.

Jak już wspomnieliśmy wcześniej ([Lekcja 7 Elementy Blokowe i Liniowe](http://how2html.pl/elementy-blokowe-liniowe/)) elementy <div>, to **elementy blokowe**. Element blokowy to taki, który zajmuje całą szerokość elementu, w którym się znajduje (tzw. **rodzica**), pod warunkiem, że zostanie wypełniony treścią. Elementy **html**nie wypełnione treścią nie zostaną wyświetlone wcale, chyba że określimy ich parametry w **CSS**(o tym w dalszej części lekcji) .

Rodzicem wszystkich elementów wyświetlanych na ekranie jest element <body>, tzn. wszystkie elementy **html**wyświetlane w przeglądarce są zagnieżdżone w elemencie body. Element body, jeśli nie określimy inaczej, zajmie szerokość całego okna naszej przeglądarki. Jeśli okno naszej przeglądarki będzie liczyło **1200px**, tożsamo szerokość elementu body wyglądać będzie. Jednak zostawmy już element body, pamiętajmy po prostu, że on tam też jest w tej całej układance i zacznijmy operować na **div**‚ach.

**Width, czyli szerokość**

Jak już wspomnieliśmy o **oknie przeglądarki internetowej** to wspomnijmy również o tym, że każdy z nas może mieć inną szerokość tego okna. Od czego to zależy? Zależy to od **rozdzielczości monitora**, na którym pracujemy. Rozdzielczość podawana jest w pikselach w stosunku ***szerokość na wysokość***, czyli np. 1366px na 768px, tzn. że naraz możemy zobaczyć 1366 pikseli wszerz i 768 pikseli wzdłuż. Dlaczego na **kursie html** piszę o rozdzielczości ekranu, dlaczego to jest takie ważne? Ważne jest to ze względu na wybór szerokości strony, którą tworzymy. Jeśli ktoś używa monitora o szerokości np. 1200px, a my stworzymy stronę o szerokości 1300px, to żeby zobaczyć te 100px będzie musiał za każdym razem przewijać stronę w bok.

Takie rozwiązanie byłoby bardzo uciążliwe, dlatego staramy się wybierać taką szerokość, która w danym momencie pasuje do większości monitorów. Nie będę tu rozpisywał się na temat odpowiedniego doboru szerokości tworzonej strony, bo nie to jest tematem tej lekcji. Wspomniałem o tym zagadnieniu, bo jest ono praktycznym wyjaśnieniem sensu deklarowania szerokości w ogóle. Skoro już o deklarowaniu szerokości, to jak tego dokonujemy?

W tym celu stwórzmy sobie prostą stronę z jednym div’em o klasie container i przygotujmy element <style> do wypełnienia. Wszystkie elementy niech znajdą się wewnątrz elementu <body>. Element **style**również zamieścimy w sekcji body na potrzeby tej lekcji, w celu zminimalizowania ilości znaczników pojawiających się w przykładach poniżej. Chociaż dobrą praktyką jest stosowanie **plików zewnętrznych** do styli, bądź umieszczanie ich w sekcji <head>. Element style zadziała również w sekcji body, jeśli umieścimy go ponad elementem, do którego chcemy dodać style. Do dzieła.

<style>

 .container {}

</style>

<div class="container"></div>

Żeby element <div> o klasie container został wyświetlony, musimy go najpierw wypełnić treścią. Dodajmy jeden krótki paragraf wewnątrz elementu container.

<style>

 .container {}

</style>

<div class="container">Jeden krótki paragraf</div>

Jeden krótki paragraf

Element container już się nam wyświetla. A dlaczego klasę naszego div’a nazwałem container? Tłumaczenie ang. rzeczownika container, czyli pojemnik nie najlepiej odwzorowuje jej zastosowanie. Posłużymy się więc czasownikiem contain, który to po ang. znaczy zawierać, bądź mieścić w sobie. Tak, teraz już lepiej. Możemy powiedzieć, że element div o klasie container mieści w sobie inne elementy html, dlatego też jest to najczęściej stosowana klasa do określania zbiorczego kontenera dużej ilości danych. Używanie klasy container nie jest obowiązkiem, jest jedynie dobrą praktyką stosowaną przez wielu developerów. Tak na prawdę klasę tą możemy nazwać dowolnie.

Klasę container stosuję się głównie do określania szerokości strony. Jak byśmy tego dokonali? Całą treść na stronie zamieścilibyśmy wewnątrz elementu container i zadeklarowali szerokość np. 960px. Wtedy żaden element na stronie nie byłby szerszy od 960px.

Dobrze już wiemy po co nam klasa container, teraz zadeklarujmy jej szerokość. W tym celu użyjemy reguły CSS o nazwie width, czyli właśnie rzeczona szerokość. Dodajmy tą regułę CSS do elementu style i ustawmy szerokość klasy container na 300px.

<style>

 .container {

 width: 300px;

 }

</style>

<div class="container">Jeden krótki paragraf</div>

Jeden krótki paragraf

W zasadzie nic nam się nie zmieniło. Żeby poznać wizualnie efekty naszej deklaracji szerokości, pokolorujmy klasę container, np. na zielono. W tym celu dodajemy kolejną regułę CSS do klasy container mianowicie background-color i przypiszemy jej wartość green.

<style>

 .container {

 width: 300px;

 background-color: green;

 }

</style>

<div class="container">Jeden krótki paragraf</div>

Jeden krótki paragraf

Teraz już gołym okiem możemy zaobserwować szerokość naszego kontenera. Możecie teraz sami poćwiczyć ustawianie różnych jego szerokości. Poniżej lista z szerokościami do ustawienia.

* 30px
* 2000px
* 50%

O co ja widzę, procenty. Tak właśnie, szerokość możemy również wyrażać w procentach. Ale procentach w odniesieniu do czego? W procentach odnośnie do szerokości rodzica, czyli do elementu, w którym danym element się znajduje. Nasz **kontener**w tym przypadku będzie przyjmował wartości **połowy**szerokości elementu <body>. Wystarczy, że zmniejszymy okno naszej przeglądarki i wraz ze zmianą okna będzie zmieniała się również szerokość naszego kontenera. Dlaczego tak się dzieje? Element <body>jest **elementem blokowym**, czyli elementem , który zajmuje całą szerokość okna przeglądarki. Wraz ze **zmianą szerokości okna przeglądarki** zmienia się również jego szerokość. Hmm ciekawe…
Szerokość wyrażana w pikselach jest nazywana ustaloną (**fixed**), natomiast szerokość wyrażana w procentach płynną (**fluid**).

Do naszego kontenera dodamy **div**‚a o klasie **content**(z ang. treść, zawartość) i umieścimy w nim nasz **paragraf**. Następnie szerokość kontenera ustalimy na stałą wynoszącą 300px, a kontentu na płynną wynoszącą 50% i dodamy niebieski kolor tła, żeby się wyróżniał.

<style>

 .container {

 width: 300px;

 background-color: green;

 }

 .content {

 width: 50%;

 background-color: greenyellow;

 }

</style>

<div class="container">

 <div class="content">

 <p>Jeden krótki paragraf</p>

 </div>

</div>

Jeden krótki paragraf

Szerokość elementu o klasie **content**będzie zawsze równa połowie szerokości kontenera. W tym przypadku wyniesie 150px, ale jakbyśmy chcieli zmienić szerokość kontenera np. na 400px to analogicznie, wartość szerokości elementu content zostanie zmieniona na 200px.

**Height, czyli wysokość**

W naszym dwuwymiarowym świecie stron internetowych oprócz szerokości, możemy również manipulować wysokością elementów html. Domyślnie wysokość elementu warunkowana, jest jego zawartością, tzn. element jest np. tak wysoki ile znajduje się tekstu w danym elemencie html. Wysokość również możemy wyrażać w wartościach względnych, czyli w %, oraz w bezwzględnych czyli w px. Podczas tej lekcji będziemy posługiwać się wartościami wyrażonymi w px. Zdefiniujmy wysokość elementu content na 200px.

<style>

 .container {

 width: 300px;

 background-color: green;

 }

 .content {

 width: 50%;

 background-color: greenyellow;

 height: 200px;

 }

</style>

<div class="container">

 <div class="content">

 <p>Jeden krótki paragraf</p>

 </div>

</div>

Jeden krótki paragraf

Widzimy, że nie tylko wysokość elementu content została powiększona, ale również wysokość naszego kontenera. Tak jak już wspomniałem powyżej wysokość elementu html, jeśli nie jest zdefiniowana, zostanie dostosowana do zawartości. W tym przypadku zawartością naszego kontenera jest element content o wysokości 200px i dlatego też wysokość kontenera również wynosi tyle samo.

**Zmiana kolorów**

Zanim przystąpimy do dalszych działań to musimy koniecznie zmienić te kolory bo są okropne. Kolorów będziemy uczyć się podczas kurs CSS, na razie tylko będę podawał wartości, które uzupełnimy, żeby przyjemniej nam się pracowało. Wartości te będą zaczynały się od krzyżyka, czy hash taga #.

**Dalsze działania…**

Wyobraźmy sobie, że prowadzimy gazetę i chcemy stworzyć taki szablon artykułu z dwoma
kolumnami położonymi obok siebie. Jaki element html w tym celu wybieramy? Div? Tak bardzo dobrze wybraliśmy. Jest to <div>, a więc skonstruujmy taki artykuł. Skonstruujemy artykuł o szerokości 400px z dwoma równo podzielonymi kolumnami. Klasę artykułu nazwiemy jak zwykle obcojęzycznie, będzie to article, a następnie kolumny odpowiednio col-1 oraz col-2. Poszczególnym kolumnom nadamy szerokość po połowie, a wysokość ustawimy na 200px. Na razie nie będziemy dodawać tekstu to do kolumn.

Wcześniej wspomniałem, że jeśli nie dodamy treści do elementu html, nie zostanie on wyświetlony. Czy zawsze jest to prawdą? W przypadku elementu liniowego tak, jednak w przypadku elementu blokowego nie zawsze. Element blokowy pojawi się na ekranie za każdym razem, gdy będzie miał określoną szerokość oraz wysokość, niezależnie czy jest wypełniony treścią czy nie.

<style>

 .article {

 width: 400px;

 background-color: #d7d7d7;

 }

 .col-1 {

 width: 50%;

 background-color: #499afa;

 height: 200px;

 }

 .col-2 {

 width: 50%;

 background-color: #fab949;

 height: 200px;

 }

</style>

<div class="article">

 <div class="col-1"></div>

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

</div>



Hmmm… Druga kolumna została wyświetlona pod spodem. Pomimo, że jest dość miejsca obok kolumny pierwszej, żeby ta druga się zmieściła. Przecież na papierze dzielą się po połowie miejscem artykułu. Co jest nie tak? Elementy div to elementy blokowe, a jak pamiętacie z lekcji o elementach blokowych każdy element blokowy zaczyna się na nowej linii. Więc co możemy zrobić, aby zostały one wyświetlone obok siebie. W tym celu musimy skorzystać z reguły CSS o nazwie float. Float to po obcojęzycznemu dryfować, jednak nie wiem czy jest to najlepsze określenie na wytłumaczenie co się stanie z elementami blokowymi po dodaniu reguły float. A więc dodajmy regułę float i zobaczmy jak zmieni się nam layout.

<style>

 .article {

 width: 400px;

 background-color: #d7d7d7;

 }

 .col-1 {

 float: left;

 width: 50%;

 background-color: #499afa;

 height: 200px;

 }

 .col-2 {

 float: left;

 width: 50%;

 background-color: #fab949;

 height: 200px;

 }

</style>

<div class="article">

 <div class="col-1"></div>

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

</div>



Kolumny są już obok siebie. Osiągnęliśmy swój cel i stworzyliśmy szablon gazety. Naczelny na pewno będzie z nas zadowolony. Hmm… No fajnie, Naczelny zadowolony, kolumny złączone, ale nadal nie wiemy o co chodzi w tych floatach, dlaczego przypisaliśmy tę wartość akurat do kolumny, a nie do innego elementu i dlaczego left, a może jednak right? Przejdźmy dalej i dowiedzmy się o co tak na prawdę chodzi.

**Float**

Reguła float przypisana do elementu blokowego, zmienia rodzaj jego rozmieszczenia na stronie. Pamiętacie pewnie z lekcji 7 ([Elementy Blokowe i Liniowe](http://how2html.pl/elementy-blokowe-liniowe/)), że elementy blokowe zawsze zajmują nową linię i domyślnie zajmują całą jej szerokość. Z powyższego przykładu o naszej gazecie możemy wywnioskować, że nawet jak zmniejszymy ich szerokość to nadal okupują całą linię. Dlaczego tak się dzieje? Można stwierdzić, że niezależnie jaką szerokość ma element blokowy to i tak zawsze zajmie całą linie. Tak jakby siedział sam przy stole i rezerwował resztę miejsc dla siebie bo lubi jadać samemu. Przedstawmy to na rysunku.



Niebieski element blokowy ma 50% szerokości, a pomarańczowy element blokowy ma zdefiniowaną szerokość jako 33%. Z powodzeniem te dwa elementy zmieściłyby się na jednej linii. Jednak są na dwóch oddzielnych. Odpowiedzialna jest za to jakby taka rezerwacja miejsca, która w przypadku elementów blokowych sięga do końca linii. Reguła float pozwala na niejako oswojenie elementów blokowych. „Ależ proszę bardzo niech Pani siada na mojej linii. Szampana Może?” Oswojone elementy, czy bardziej fachowo dryfujące elementy, pozbywają się tej rezerwacji i mogą się do siebie spokojnie przyklejać.



Widzimy, że po dodaniu float’a elementy te przykleiły się do siebie i nawet zostawiły jeszcze wolne miejsce (oznaczone kolorem szarym) dla kolejnego elementu blokowego, jakbyśmy takowy dodali. Element taki oczywiście musiałby być nie szerszy niż to pozostawione miejsce, czyli w tym przypadku 100%-(50%+33%)=17%. Trochę matematyki nigdy nigdy nie zaszkodzi. Element, który byłby szerszy niż 17% nawet z przypisaną regułą float znalazłby się na kolejnej linii. Elementy html jeśli na danej linii mają za mało miejsca wtedy zawsze przenoszą się do nowej linii.



Żółty element blokowy o szerokości 75% znajduje się już w nowej linii, niezależnie dodanej do niego reguły float.

**Float left, right**

Elementy dryfujące mogą dryfować w lewo, bądź w prawo. Co to znaczy? Oznacza to, że możemy określić, w którą stronę mają zostać wyrównane do lewej czy do prawej. Do tej pory używaliśmy wartości left. Zobaczmy co się stanie jak zmienimy wyrównanie na prawą stronę.



Elementy zostały rzeczywiście wyrównane do prawej i pozostawiły miejsce z lewej strony, ale element niebieski zamienił się miejscem z elementem pomarańczowym. Czemu tak się stało? Przedstawmy kod, jaki musielibyśmy napisać, żeby osiągnąć to co na powyższym rysunku.

<style>

 .blue {

 float: right;

 width: 50%;

 height: 150px;

 background-color: #499afa;

 }

 .orange {

 float: right;

 width: 33%;

 height: 150px;

 background-color: #fab949;

 }

 .yellow {

 float: right;

 width: 75%;

 height: 150px;

 background-color: #fae248;

 }

</style>

 <div class="blue"></div>

 <div class="orange"></div>

 <div class="yellow"></div>

Element niebieski w naszym kodzie został zapisany najwyżej spośród wszystkich trzech elementów blokowych, czyli to znaczy, że on będzie interpretowany przez przeglądarkę internetową jako pierwszy. Posiada regułę float o wartości right, czyli zostanie wyrównany do prawej strony elementu, w którym się znajduje, czyli w tym przypadku body. Element pomarańczowy jest drugi, czyli zostanie przyklejony do elementu niebieskiego jeśli zostanie miejsce. Dla elementu żółtego nie starczyło miejsca, tzn. że zostanie on wyświetlony na następnej linii z wyrównaniem również do prawej strony.

Dla poprzednich przykładów nie przedstawialiśmy kodu, żeby wytłumaczyć działanie elementu na float. Natomiast jedyne co się w kodzie zmieniło w poprzednich przykładach to wartość reguły float.

**Float left, right left**

Żeby jeszcze trochę namieszać wam w głowach, powiem, że reguł float można używać z różnymi wartościami. Dla elementu niebieskiego przypiszemy wartość float left, dla pomarańczowego right, a dla żółtego znowu left i zobaczmy co się stanie.

<style>

 .blue {

 float: left;

 }

 .orange {

 float: right;

 }

 .yellow {

 float: left;

 }

</style>



**Pojedynczy float**

Przypuśćmy, że chcemy stworzyć layout, gdzie mamy po lewej stronie jeden długi element (np. menu boczne) oraz tuż obok mamy dwa elementy, które są pod sobą (np. artykuły na blogu). O tak jak na poniższym rysunku.



Przypuśćmy, że chcielibyśmy teraz uzyskać taki layout poprzez zastosowanie tylko jednego float’a. No może to dziwić, ale w zasadzie z tego co przed chwilą było napisane wydaje się to całkiem logiczne. Elementowi niebieskiemu nadajemy float: left, niebieski element pozbywa się rezerwacji, czyli, że może przyjąć elementy na miejsce obok, a że akurat na szerokość te elementy pasują, to w zasadzie wszystko by pasowało. Otóż przekonajmy się sami.

 .blue { width: 200px; height: 400px; float: left; }

 .orange { width: 400px; height: 200px; }

 .yellow { width: 400px; height: 200px; }

<div class="blue"></div>

<div class="orange"></div>

<div class="yellow"></div>



Hmm.., nie tak miało to wyglądać. Nie ma połowy szerokości elementu pomarańczowego i żółtego. Co się stało w takim razie? Jeśli dokładnie przeanalizujemy przypadek okaże się, że elementy pomarańczowy i żółty zostały zakryte elementem niebieskim. Nie o to nam chodziło niestety.

Jeśli dodamy zasadę float tylko do jednego elementu to musimy się spodziewać takiego dość dziwnego zachowania elementów blokowych. Jak w takim razie uzyskać taki layout? W zasadzie w tym przypadku wystarczyłoby dodać float do każdego elementu i wyszłoby na to samo, jednak dobrą praktyką jest zagnieżdżanie tych elementy w kolejne div’y. I tym zewnętrznym div’om nadać własność float.

.blue { width: 200px; height: 400px; float: left; }

.content { width: 400px; height: 400px; float: left; }

 .orange { width: 400px; height: 200px; }

 .yellow { width: 400px; height: 200px; }

<div class="blue"></div>

<div class="content">

 <div class="orange"></div>

 <div class="yellow"></div>

</div>

****

I gotowe osiągnęliśmy to co chcieliśmy dodając jedynie jednego div’a, niektórzy mogą powiedzieć, że aż jednego. Po co tak komplikować sobie życie z zagnieżdżaniem div’ów w kolejnych div’ach? Przejdźmy dalej tam poznamy odpowiedź na to pytanie.

**Domy z klocków**

Do budowy strony internetowej musimy podejść jak do budowy konstrukcji z modułów, bloków tudzież klocków (np. tych takich Lego klocków). Musimy sobie najpierw tą stronę podzielić w głowie, bądź na papierze, czy też w programie graficznym na poszczególne sekcje.

Weźmy przykład prostej strony, która posiada sekcję nagłówka (to nie jest ani head ani h1, tylko taki kontener), gdzie zazwyczaj umieszczamy logo i menu. Następnie po lewej stronie będziemy mieli, menu boczne, kontener do tego menu nazwiemy, np. aside (czyli taka sekcja z boku). Na środku tam gdzie będziemy mieli główną treść postawimy kolejny kontener o nazwie content. Na samym dole na całą szerokość strony dodamy sobie stopkę, po ang. to będzie footer. Taka prosta konstrukcja przykładowej realnej strony.



Można by stwierdzić, że w ten sposób przygotowaliśmy szkielet layout’u, który jeśli będziemy chcieli możemy zacząć wypełniać treścią. Dodatkowo z tak przygotowanym layout’em nie będziemy musieli się przejmować, czy elementy wstawione np. do elementu content będą zachodzić na inne sekcje.

Żeby opanować dokładnie sztukę układania floatów, trzeba niestety sporo ćwiczyć, gdyż jest to rzecz, która zazwyczaj sprawia najwięcej kłopotów z właściwym zrozumieniem. Szczególnie trudne jest na początku przewidzenie co stanie się z danym elementem blokowym jeśli dodamy do niego regułę float.

**Podsumowanie Lekcja 15 Div + Div = Layout**

Tym razem długa lekcja, ale pokryliśmy znaczny obszar. Jeśli przebrnęliście przez nią tzn. że jesteście spory krok bliżej poznania sztuki tworzenia stron internetowych przy użyciu HTML i CSS. Moje Gratulacje. Do zobaczenia w następnej następnej lekcji.

Jeśli podoba Ci się nasz kurs, pomóż nam go rozreklamować. Jeśli chcesz możesz go udostępnić swoim znajomym, albo polubić naszą stronę na fb [how2html](https://www.facebook.com/pages/How2html/1547763982120761). Więcej czytelników to również szybsze aktualizacje… Pozdrawiamy i życzymy miłej nauki…

## Przykład

<html>

<head>

<style>

html {

 background-color: #FFF;

 color: #000;

 margin: 0;

 padding: 0;

}

body {

 margin: 0 auto;

 padding: 0;

 display: block;

 width: 1004px;

 overflow: auto;

 background-color: #FFF;

}

#header {

 width:100%;

 float: left;

 background-color: #999;

}

#lewy {

 width: 200px;

 float: left;

 overflow: hidden;

 background-color: #ccc;

 margin:10px;

}

#prawy {

 width: 200px;

 float: right;

 overflow: hidden;

 background-color: #ccc;

 border-style: solid;

 border-width:5px;

 border-color:blue;

}

#kontener {

 width: 604px;

 float: left;

 overflow: hidden;

 background-color: #fff;

}

#footer {

 clear: both;

 width: 100%;

 background-color: #888;

}

</style>

<body>

</body>

</html>