Technologie informacyjne
przed 10 wykładem

Andrzej Giniewicz

8.05.2024

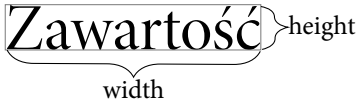
Podczas dzisiejszych zajęć nauczymy się o tym, w jaki sposób dodać styl do dokumentu HTML korzystając z języka CSS. CSS jest jednocześnie prosty, ponieważ ma niewiele zasad, ale i trudny, ponieważ ma ogrom możliwości ich wykorzystania. Najpierw nauczymy się podstawowych 10 cech oraz elementów języka CSS, które z osobna wydają się błahe, ale w sumie pokrywają znaczną większość języka. Następnie omówimy kilka nowych technologii CSS, które wchodzą do powszechnego użytku.

1 10 najważniejszych cech CSS

CSS jest skrótem od Cascading Stylesheet, czyli kaskadowe arkusze stylu. Jest to język, który pozwala opisać cechy elementów strony, między innymi takie jak ich położenie, kolor oraz rozmiar. Najczęściej kod CSS umieszczamy w osobnym pliku i odwołujemy się do konkretnych znaczników za pomocą tak zwanych selektorów, informując przeglądarkę, że na przykład wszystkie akapity mają być złożone krojem pisma URW Garamond. Najważniejsze aspekty języka CSS możemy podzielić na 10 elementów, które pozwalają nam na korzystanie ze znacznej jego części.

1.1 Model pudełkowy

Model pudełkowy jest podstawą działania CSS. Według modelu pudełkowego każda zawartość, czyli na przykład tekst, obrazek lub inne elementy, są zamknięte w pudełkach. Pudełko to najmniejszy możliwy prostokątny obszar o bokach równoległych do krawędzi strony, otaczający daną treść. Oznacza to, że pudełka są idealnie dopasowane. Domyślnie wysokość pudełka znajduje się w parametrze `height`, a szerokość w parametrze `width`.



Poza pudełkiem określony jest obszar nazywany marginesem wewnętrznym (`padding`). Za marginesem wewnętrznym znajduje się obramowanie (`border`), natomiast za obramowaniem margines zewnętrzny (`margin`). Obszar ograniczony przez zawartość, margines wewnętrzny i obramowanie nazywany jest tłem pudełka (`background`). Domyślnie wysokość i szerokość dotyczy tylko zawartości, natomiast marginesy i obramowanie mają swoje własne atrybuty, które można ustawić niezależnie.

Marginesy zewnętrzne pomiędzy sąsiadującymi pudełkami mogą na siebie nachodzić, ponieważ margines zewnętrzny oznacza „zarezerwuj tyle miejsca". Jeśli obok siebie są dwa obiekty, jeden żądający 10 pikseli odstępu i drugi 20 pikseli odstępu, wystarczy, że będą oddalone od siebie o 20 pikseli, aby spełnić oba te wymagania.

Parametry `margin` i `padding` możemy ustawić na kilka sposobów:

- używając osobnych własności, na przykład dla marginesów wewnętrznych będzie to `margin-top`, `margin-right`, `margin-bottom`, `margin-left`, przy czym każdej własności podajemy jeden rozmiar;
- używając skrótu `margin: top right bottom left;` w którym podajemy 4 rozmiary jeden po drugim;
- używając skrótu `margin: top right bottom;` w którym lewy będzie równy prawemu;
- używając skrótu `margin: top right;` w którym lewy będzie równy prawemu a dolny górnemu;
- używając skrótu `margin: top;` w którym wszystkie marginesy są równe górnemu.

1.2 Rodzaj wyświetlania

Rodzaj wyświetlania `display` jest jedną z ważniejszych własności. Pozwala on odróżnić elementy liniowe (`display: inline;`) od elementów blokowych (`display: block;`). Dzięki niemu możemy zmienić domyślne zachowanie elementów HTML[1]. Elementy liniowe podobnie jak tekst nie rozpoczynają nowej linii i zajmują tylko tyle miejsca, ile potrzebują. Elementy blokowe pojawiają się w nowej linii i zajmują tyle miejsca, ile tylko mogą. Oprócz standardowych wartości `inline` oraz `block` można wskazać jeszcze `none` powodujące, że obiektu nie ma i nie zajmuje miejsca oraz `inline-block`, czyli wyświetlanie liniowo-blokowe. W wyświetlaniu liniowo-blokowym, pudełka zachowują się jak elementy liniowe na zewnątrz zawartości (czyli z punktu widzenia innych elementów), ale jak blokowe we wnętrzu zawartości. Istnieje więcej rodzajów wyświetlania, których zachowanie można sprawdzić w dokumentacji.

Co ważne, spośród opisanych trybów tylko wyświetlanie blokowe oraz liniowo-blokowe pozwala na ustawianie wysokości oraz szerokości. Jeśli w pliku CSS podajesz wysokość i szerokość, ale zdaje się ona nie działać, sprawdź, czy masz ustawiony właściwy rodzaj wyświetlania.

[1] W ramach przypomnienia, na przykład `div`, `p`, `ul` są elementami blokowymi, natomiast `span`, `em`, `a` są elementami liniowymi.

1.3 Pozycjonowanie

Są cztery najważniejsze sposoby pozycjonowania (`position`):

- statyczne (`static`) — domyślne i ignorujące wszelkie próby ręcznego pozycjonowania;
- względne (`relative`) — pozwala przesunąć element względem standardowej pozycji elementu. Pozostałe elementy wciąż myślą, że element jest tam, gdzie był przed przesunięciem;
- ustalone (`fixed`) — pozwala ustawić element względem okna, niezależnie od tego, w którym miejscu strony jesteśmy. Pozostałe elementy myślą, że element zniknął;
- bezwzględne (`absolute`) — pozwala ustawić element względem najbliższego rodzica posiadającego pozycję inną niż statyczną albo tagu `<body>`.

Po wybraniu sposobu pozycjonowania możemy przesuwać elementy za pomocą właściwości `left`, `right`, `top` lub `bottom`, którym w najprostszej wersji podajemy odległość. Jeśli próbujesz przesunąć element w CSS i atrybuty zdają się nie ruszać, upewnij się, czy przypadkiem nie jest ustawione pozycjonowanie statyczne, które ignoruje ręczne pozycjonowanie i każe ustawić elementy przeglądarce.

1.4 Jednostki

Do ustawiania rozmiarów i przesunięć w CSS możemy użyć wielkości z różnymi jednostkami. Najpopularniejszymi jednostkami są:

- `px` (piksele) — w przybliżeniu odpowiada pikselom ekranu (jeśli rozdzielczość to 96 punktów na cal i odległość od ekranu to długość ramienia);
- `%` (wartość procentowa) — miara liczona względem rozmiaru pudełka, które jest rodzicem elementu;
- `em` — jednostka odpowiadająca stopniowi pisma użytemu w danym elemencie. Jeśli element ma stopień pisma równy 12 pikseli, 2 em to 24 piksele. Niestety użycie 1 em daje różne rozmiary w różnych miejscach strony;
- `rem` — jednostka odpowiadająca stopniowi pisma elementu `<html>`. Użycie 1 rem daje ten sam rezultat na całej stronie, ale nie ma wsparcia we wszystkich przeglądarkach.

Jest to dobry moment, aby wspomnieć o sposobie sprawdzania wsparcia różnych własności CSS (i nie tylko) w przeglądarkach. W przypadku jednostki rem, na stronie `http://caniuse.com/#feat=rem` możemy sprawdzić, że ponad 99% przeglądarek wspiera już tę jednostkę. Niewspierana jest przez przeglądarki Internet Explorer 10 i starsze oraz Safari 5 na systemie iOS. Ponieważ wszystkie przeglądarki wydane w 2013 roku lub później wspierają jednostkę rem, możemy swobodnie z niej korzystać.

1.5 Warstwy

Niekiedy elementy strony nachodzą na siebie. Aby ustalić kolejność rysowania, posługujemy się warstwami, dodając „współrzędną z". Zakładamy, że to co jest bliżej oglądającego (na wierzchu), ma wyższą współrzędną z. Ustalamy ją własnością `z-index` na liczbę całkowitą.

Wraz z warstwami często chcemy sterować przezroczystością (ang. transparency). W CSS odpowiednim parametrem jest „nieprzezroczystość" (ang. opacity), której wartość to liczba rzeczywista od 0 (całkowicie przezroczysty, ang. transparent) do 1 (całkowicie nieprzezroczysty, ang. opaque). Odpowiedni parametr w CSS do sterowania nieprzezroczystością to `opacity`.

Przezroczystość działa dobrze wszędzie oprócz IE < 9. Możemy to sprawdzić pod adresem `https://caniuse.com/#feat=css-opacity`.

1.6 Elementy pływające

Elementy pływające w CSS służą pierwotnie do wstawiania ilustracji obłamanych tekstem. Ponieważ dla CSS rodzaj pudełka nie gra roli, można za ich pomocą obłamywać tekstem dowolne inne elementy. Obłamanie odbywa się przez ustawienie ściągania ilustracji do lewej (`float: left;`) lub do prawej (`float: right;`). Po ustawieniu elementu jako pływający jest on wydzielany ze swojego standardowego miejsca i przesuwany na do lewej lub prawej strony rodzica. Pozostałe elementy liniowe opływają go. Informacja ta jest ignorowana, jeśli pozycja elementu jest bezwzględna.

Mamy również możliwość wskazać, że dany element nie powinien znajdować się obok elementu pływającego. Robimy to atrybutem `clear`, podając jako jego wartość, czy zabraniamy opływania z lewej (`left`), z prawej (`right`) lub z obu (`both`) stron.

Niestety, rozmiar rodzica nie rozciąga się tak bardzo, aby objąć elementy pływające. W rodzicu możemy sterować tym, co się dzieje, gdy dziecko „wyjdzie poza pudełko rodzica" (atrybut `overflow`). Dziecko może wystawać (`visible`) lub być ucięte (`hidden`), jeśli jest przesuwalne (`scroll`), w rodzicu pojawią się paski przesuwania. Ostatecznie zachowanie automatyczne (`auto`) powoduje, że jeśli jest miejsce, to rodzic się rozciągnie, a gdy nie ma, pojawi się pasek przesuwania.

1.7 Selektory

Selektory służą do nadawania reguł elementom. Najbardziej podstawowymi selektorami są:

- joker (*) — dotyczy dowolnego elementu.
- tag (np.: `p`) — dotyczy wszystkich elementów o danym tagu.
- .klasa (np.: `.menu`) — dotyczy wszystkich elementów o danej klasie.
- #id (np.: `#active`) — dotyczy elementu o danym id.
- tag.klasa (np.: `li.menu`) — dotyczy elementów o danym tagu i klasie.

- tag#id (np.: `a#active`) — dotyczy elementów o danym tagu i id.
- reguła1, reguła2 (np.: `h1, h2, h3, h4, h5, h6`) — dotyczy elementów spełniających dowolną z reguł.

Oprócz podstawowych selektorów możemy mieć tak zwane selektory kontekstowe, które składają się z innych selektorów:

- reguła1 reguła2 (np.: `ul li`) — dotyczy wszystkich elementów spełniających regułę 2, znajdujących się **wewnątrz** elementu spełniającego regułę 1.
- reguła1 > reguła2 (np.: `ul > li`) — dotyczy wszystkich elementów spełniających regułę 2, znajdujących się **bezpośrednio wewnątrz** elementu spełniającego regułę 1.
- reguła1 + reguła2 (np.: `li + p`) — dotyczy wszystkich elementów spełniających regułę 2, znajdujących się **tuż za** zamknięciem elementu spełniającego regułę 1.
- reguła1 ~ reguła2 (np.: `li ~ p`) — dotyczy wszystkich elementów spełniających regułę 2, znajdujących się **tuż przed** otwarciem elementu spełniającego regułę 1.

Należy zwrócić szczególną uwagę, gdy piszemy selektory. Selektor `p em` i `p, em` to nie jest ten sam ani taki sam selektor. Pierwszy wybiera tagi `em` wewnętrz tagów `p`, drugi wybiera wszystkie tagi `p` i `em`.

Kolejnym rodzajem zaawansowanych selektorów są selektory atrybutów:

- [atrybut=wartość] (np.: `[target="_blank"]`) — wybiera elementy o atrybucie równym wartości.
- [atrybut*=wartość] (np.: `[href*="google.com"]`) — wybiera elementy o atrybucie zawierającym wartość.
- [atrybut^=wartość] (np.: `[href^="https://"]`) — wybiera elementy o atrybucie zaczynającym się od wartości.
- [atrybut$=wartość] (np.: `[href$=".pdf"]`) — wybiera elementy kończące się wartością.
- [atrybut~=wartość] (np.: `[data~="test"]`) — wybiera elementy posiadające wartość jako słowo (gdy oddzielone spacjami).
- [atrybut|=wartość] (np.: `[data|="test"]`) — jak wyżej, ale gdy oddzielone myślnikami.

Istnieje też całe mnóstwo bardziej zaawansowanych selektorów[2]. Przykładowo reguła

```
article > section:first-of-type > p:first-of-type::first-letter {
  color: red;
  font-size: 500%;
  float: left;
}
```

[2] Pełny opis w można znaleźć w specyfikacji lub na `http://www.w3schools.com/cssref/css_selectors.asp`.

mówi, że pierwsza litera pierwszego akapitu pierwszej sekcji każdego artykułu będzie czerwona, większa 5 razy niż otaczający tekst oraz oblana otaczającym tekstem.

1.8 Selektory medium

Selektory medium (ang. media query) pozwalają na określenie różnych wariantów stylu dla różnych rodzajów odbiorników. Ogólny schemat to:

```
@media warunki {...}
```

Warunkiem może być wyrażenie w nawiasie lub rodzaj urządzenia: wydruk (ang. print), ekran (ang. screen), czytnik ekranowy (ang. speech) lub dowolne urządzenie (ang. all, wartość domyślna). Przed warunkami można dodać przedrostek oznaczający oprócz albo tylko (dla starszych przeglądarek), odpowiednio z angielskiego „not" albo „only". Warunki oddzielamy słowem kluczowym `and` oznaczającym i lub przecinkiem oznaczającym lub. Do najpopularniejszych wyrażeń należą związane z własnością `min-width`, `max-width` oraz `orientation`.

Selektory medium pozwalają na reaktywność, czyli dostosowywanie wyglądu strony do warunków odbiorcy. Przyjrzyjmy się kodowi, który ma trzy wersje strony zależne od szerokości ekranu.

```
@media all and (min-width: 1000px) {
  .wrapper, .paper, .navigation, .footer {
    width: 1000px;
    margin: 0 auto;
  }
}
@media all and (min-width: 768px) {
  .main-content {
    width: 70%;
    float: left;
  }
  .aside-content {
    width: 30%;
    float: right;
  }
}
```

Mamy tutaj dwa warianty strony. Jeśli przeglądarka osiągnie minimum 768 pikseli szerokości, cechy obiektów wskazywanych przez klasy `main-content` oraz `aside-content` się zmienią. Szerokość pierwszego z nich będzie wynosiła 70%, szerokość drugiego 30%. Pierwszy będzie przyciągnięty do lewej, drugi do prawej strony. Efektem będzie ustawienie dwukolumnowego układu w proporcji 70:30, gdy szerokość to przynajmniej 768 pikseli oraz umieszczenie

treści jedna pod drugą (domyślnie) dla węższych ekranów. Oznacza to, że na większości telefonów strona wyświetli się w układzie jednokolumnowym, natomiast na tabletach w dwukolumnowym. Dodatkowo, gdy strona przekroczy 1000 pikseli, co jest często spotykane na komputerach, kilka innych elementów strony zostanie wyśrodkowanych a ich rozmiar zostanie ustawiony na 1000 pikseli. Oznacza to, że strona będzie zajmowała całą szerokość na małych ekranach, natomiast na dużych będzie wyśrodkowana i o stałej szerokości.

Tego typu projektowanie nazywa się „mobile first", ponieważ pierwsza wersja strony powstaje dla najbardziej ograniczonych urządzeń, jakimi są telefony — a następnie dla coraz większych ekranów dodajemy nowe możliwości, które powodują, że strona dostosowuje się do ich możliwości.

1.9 Dołączanie stylu

Styl możemy dołączyć do elementu HTML na trzy sposoby:

- z zewnętrznego pliku (tag `<link rel="stylesheet" href="plik.css">` wewnątrz tagu `<head>`),
- z pliku HTML (tag `<style> ... </style>` wewnątrz tagu `<head>`),
- w tagu (atrybut `style="..."` tagu, np.: `<p style="color:blue;">`).

Ostatnia forma nie jest zalecana, choć działa. Zdecydowanie najlepszą opcją jest pierwsza, dzięki której przeglądarka może zapamiętać plik CSS i nie pobierać go ponownie, jeśli nie uległ zmianie.

1.10 Kaskadowanie

Kaskadowanie jest na tyle centralną funkcjonalnością, że występuje nawet w nazwie języka. Jest to zbiór reguł mówiących, co się dzieje, gdy jakiś element nie ma stylu lub jest kilka definicji stylu dotyczących tego samego elementu.

Rozważamy trzy rodzaje stylu: styl **autora** (czyli ustalony przez autora strony), styl **użytkownika** (czyli ustalony przez użytkownika strony) oraz **aplikacji klienckiej** (ang. user agent, czyli na przykład przez twórców przeglądarki).

Pierwszym krokiem podejmowania decyzji w kaskadowaniu jest wybranie deklaracji, które opisują dany element dla ustalonego typu medium. Jako kolejne następuje sortowanie elementów. Jeśli na przykład kolor danego elementu ustalony jest w kilku regułach, wygrywa ta, która w posortowanej liście występuje najpóźniej, ponieważ jej wartość nadpisuje wcześniejsze. Sortowanie odbywa się najpierw według **ważności** reguły od najmniej ważnych do najważniejszych:

1. styl aplikacji klienckiej,
2. styl użytkownika,
3. styl autora,
4. styl autora oznaczony jako ważny,

5. styl użytkownika oznaczony jako ważny.

W obrębie każdego z tych poziomów następuje sortowanie według **szczegółowości** reguły. Sortowanie według szczegółowości wymaga wyliczenia czterech wartości:

- $a = 1$, jeśli element jest atrybutem `style="..."` jakiegoś tagu, $a = 0$ w przeciwnym wypadku,
- b jest liczbą atrybutów id w selektorze (np.: `#active`),
- c jest liczbą innych atrybutów (w tym klas) i pseudo-klas (selektorów elementów istniejących w dokumencie, takich jak `:first-of-type`),
- d jest liczbą tagów i pseudo-elementów (selektorów części elementów dokumentu, takich jak `::first-letter`, które nie wskazuje na element).

Sortowanie w obrębie wspólnego poziomu ważności odbywa się w porządku leksykograficznym rosnącym dla wektora (a, b, c, d). Wciąż jednak mogą powstać remisy. Jeśli danego elementu dotyczy kilka reguł o tej samej ważności i specyficzności, kolejność pomiędzy nimi określana jest poprzez kolejność ich deklaracji. Zakładamy, że najpierw zdefiniowane zostały wszystkie style w atrybucie `style="..."`, następnie wszystkie style w tagu `<style>` a następnie wszystkie style pochodzące z innych plików, podłączonych tagiem `<link>`. W obrębie reguł w każdej z tych grup kolejność definiowana jest przez kolejność występowania w kodzie. Liczy się również kolejność załączania plików oraz elementów stylu, choć rzadko obserwujemy, aby miało to jakiś wpływ — pozostałe reguły sortowania zwykle są rozstrzygające.

Popatrzmy na przykład selektora z wcześniejszego podpunktu.

```
article > section:first-of-type > p:first-of-type::first-letter
```

Zakładając, że reguła ta występowała w pliku `.css`, to wartość $a = 0$, $b = 0$ (brak id), $c = 2$ (dwukrotnie `first-of-type`, które wskazuje element dokumentu, brak klas i innych atrybutów), $d = 4$ (`article`, `section`, `p` oraz `first-letter`). Oznacza to, że kluczem sortowania reguły będzie $(0, 0, 2, 4)$, zatem będzie rozważana przed $(0, 0, 2, 3)$, ale po $(0, 1, 1, 1)$.

Niektóre elementy stylu możemy określić jako ważne. Robimy to w sytuacji, gdy przykładowo biblioteka, z której korzystamy, ma regułę o wyższej specyficzności, ale mimo to chcemy zmienić wybraną cechę. Dzięki dodaniu tekstu `!important` na końcu reguły, zwiększamy ważność reguły, tym samym umieszczając ją przed wszystkimi mniej ważnymi regułami dotyczącymi tego tagu, niezależnie od ich specyficzności.

Informacji `!important` powinniśmy używać w ostateczności, ponieważ nie można wskazać, że coś jest jeszcze ważniejsze. Oznacza to, że nie sposób przeciążyć reguły inną o mniejszej specyficzności.

Aby ustalić wartość atrybutu, na przykład kolor, jeśli procedura kaskadowania zwróci choć jedną wartość, będzie ona przypisana do elementu. W przeciwnym razie, jeśli atrybut może być dziedziczony (należy sprawdzić w dokumentacji atrybutu), wybrana jest wartość wyliczona dla rodzica, o ile rodzic istnieje. Jeśli wciąż nie można ustalić wartości, wybierana jest wartość domyślna.

W CSS istnieją dwie specjalne własności `inherit` oraz `initial`, które pozwalają obejść kaskadowanie. Przykładowo `inherit !important` spowoduje, że nawet jeśli dla elementu ustalilibyśmy jakiś styl (na przykład w bibliotece), chcemy, aby styl był identyczny, jak styl rodzica elementu.

2 Nowe możliwości CSS i przydatne triki

Język CSS cały czas się rozwija, dodawane są do niego nowe atrybuty i wartości. Powstają nowe metody. W niniejszej sekcji przestawimy kilka nowych tricków, które warto znać projektując styl stron.

2.1 Wymiarowanie pudełek

Wymiarowanie pudełek tylko przez zawartość nie zawsze pozwala na łatwe rozmieszczanie elementów. Jeśli chcemy, możemy skorzystać z fragmentu kodu

```
:root {
  box-sizing: border-box;
}
*, ::before, ::after {
  box-sizing: inherit;
}
```

który przełączy CSS na mierzenie pudełek wraz z ramką i marginesem wewnętrznym. Niestety, nie wszędzie skorzystamy z atrybutu `box-sizing`. Dostępność można sprawdzić na `https://caniuse.com/#feat=css3-boxsizing`.

2.2 Obliczenia w CSS

Niekiedy wykonywanie obliczeń, szczególnie z różnymi jednostkami, bywa uciążliwe. Przykładowo możemy ustalić szerokość kolumn na 70% i 30%, przy czym mniejszą z nich skrócić o 1.5em, aby dodać przewidywalny odstęp.

```
.main {
  width: 70%;
}
.sidebar {
  width: calc(30% - 1.5em);
  margin-left: 1.5em;
}
```

To, czy możemy użyć funkcji `calc`, można sprawdzić na `https://caniuse.com/#feat=calc`.

2.3 Sowa po lobotomii

Sowa po lobotomii, nazywana tak[3] ze względu na zapis selektora `* + *`, pozwala odwołać się do elementów, które nie są pierwsze w swoim rodzicu. Na przykład, aby ustalić margines pomiędzy elementami, ale nie nad pierwszym z nich, możemy ustawić.

```
body * + * {
 margin-top: 1.5em;
}
```

W podobny sposób można przełączyć akapity na bardziej książkowe, ustawiając justowanie tekstu, a następnie dla wszystkich akapitów, które nie są pierwsze, ustawić wcięcie akapitowe i skasować odstęp pomiędzy akapitami.

```
p {
  text-align: justify;
}
p + p {
  text-indent: 2em;
  margin-top: 0em;
}
```

2.4 Clearfix

Czasem, jeśli korzystamy z elementów pływających, musimy sztucznie dodawać tag `div` tylko po to, żeby ustalić mu opcję `clear: both;`, powodującą, że kolejne elementy nie zachowują się źle w layoucie, wskakując obok ilustracji lub wjeżdżając pod nią. Dobrym zwyczajem może być dodanie klasy `clearfix` lub `cf` do każdego elementu zawierającego elementy opływane:

```
.clearfix::after, .cf::after {
  display: block;
  content: " ";
  clear: both;
}
```

3 Nowe sposoby rozmieszczania elementów

Dwie nowe i ważne technologie, to:

- Flexbox — `https://caniuse.com/#feat=flexbox` oraz

[3] Patrz na przykład `https://alistapart.com/article/axiomatic-css-and-lobotomized-owls`.

- system siatek — `https://caniuse.com/#feat=css-grid`.

Jeśli mamy dostęp do obu technologii, używajmy:

- **elementów pływających** do ilustracji i tabelek opływanych tekstem,
- **Flexbox** do galerii, rozmieszczania elementów interfejsu takich jak menu, ciągi paneli,
- **systemu siatek** do tworzenia głównego układu strony.

Jeśli projektujemy stronę bez systemów siatek, do układu strony najlepiej użyć Flexboxa. Jeśli nie mamy ani Flexboxa ani systemu siatek, musimy stosować elementy pływające. Wcześniej przy okazji reaktywności zademonstrowany był przykład, jak zbudować układ dwukolumnowy za pomocą elementów pływających. W niniejszej sekcji zajmiemy się nowszymi technologiami. Będzie to jedynie wstęp do obu tematów.

3.1 Flexbox

Flexbox sam rozmieszcza elementy w pudełku. Włączamy go, ustalając `display` w kontenerze na `flex` lub `inline-flex`. Flexbox rozmieszcza elementy wzdłuż głównej osi (domyślnie OX).

```
.container {
  display: flex;
}
```

Elementy wewnątrz mogą mieć ustalany rozmiar parametrami `flex-basis`, `flex-grow` oraz `flex-shrink`. Flexbox ustala rozmiary następująco:

- Jeśli nie podano `flex-basis`, wynosi tyle, co zawartość pudełka.
- Jeśli suma `flex-basis` jest mniejsza niż szerokość pudełka i podano `flex-grow`, każdy element otrzyma proporcjonalnie pozostałe wolne miejsce.
- Jeśli suma `flex-basis` jest większa niż szerokość pudełka i podano `flex-shrink`, każdy element będzie zmniejszony proporcjonalnie, aby wszystko się zmieściło.

Komenda `flex` jest skrótem dla trzech powyższych, kolejno dla `grow`, `shrink` i `basis`. Standardowo `shrink` to `0` i basis to `0%`.

Domyślnie, nic się z pudełkami nie dzieje, ale jeśli mamy pudełka A i B, możemy sterować wypełnieniem. Przykładowo:

- `flex: 0 0 300px`, `flex: 1` — pierwsza kolumna ma 300px, druga zajmuje całe pozostałe miejsce,
- `flex: 1`, `flex: 2` — kolumny zajmują całe miejsce, druga jest 2 razy większa niż pierwsza.

Aby stworzyć układ dwukolumnowy w proporcji 70 – 30, mając `div` o klasie `main` i `aside` wewnątrz `container`, moglibyśmy napisać

```
.container {
  display: flex;
}
.main {
  flex: 7;
}
.aside {
  flex: 3;
}
```

co jest o wiele prostszym rozwiązaniem niż triki z ustawianiem opływania dla kolumn. Unikamy też problemów z tłem rodzica, któremu nie pozostały żadne niepływające elementy wewnętrzne.

Flexbox pozwala na ustawienie szeregu dodatkowych opcji, między innymi (wartości domyślne zostały pogrubione):

- `flex-direction` — **row**, row-reverse, column, column-reverse — kolejność umieszczania pudełek w kontenerze,
- `flex-wrap` — **nowrap**, wrap, wrap-reverse — co gdy wystąpi overflow,
- `justify-content` — **flex-start**, flex-end, center, space-between, space-around — wyrównywanie wzdłuż osi głównej,
- `align-items` — **strech**, flex-start, flex-end, baseline, center — rozmieszczenie wzdłuż osi dodatkowej,
- `align-content` — flex-start, flex-end, center, space-between, space-around, stretch — wyrównywanie wzdłuż osi dodatkowej gdy nastąpi zawijanie.

W sieci można znaleźć dużo materiałów. Przykładowe źródło to `https://css-tricks.com/snippets/css/a-guide-to-flexbox/`.

3.2 Systemy siatek

System siatek, czyli Grid, działa jak dwuwymiarowy Flexbox. Pozwala na wyznaczenie kolumn i wierszy na stronie i umieszczanie elementów układu w odpowiednich komórkach lub zakresach siatki.

Grida włączamy, ustalając `display` w kontenerze na `grid` lub `inline-grid`.

```
.container {
  display: grid;
}
```

W systemie siatek tworzymy kolumny i wiersze. Jeśli pominiemy specyfikację wierszy, tworzone są automatycznie w miarę potrzeby. Stworzy się tyle wierszy lub kolumn, ile szerokości podamy. Możemy podać szerokość jako wymiar lub słowo `auto`, co proporcjonalnie rozdysponuje całe miejsce pomiędzy kolumny lub wiersze z tym słowem kluczowym. Opcjonalnie przed szerokością możemy podać nazwy w nawiasach kwadratowych. Domyślne nazwy to numery kolumn.

```
.container {
  display: grid;
  grid-template-columns: [nazwa] szerokość ...;
  grid-template-rows:  [nazwa] szerokość ...;
}
```

Elementy w siatce możemy ustawić, wykorzystując nazwy lub numery kolumn, podając początkową i końcową kolumnę oraz początkowy i końcowy wiersz. Przykładowo, jeśli istnieje wiersz o nazwie `header`, poniższy kod umieści element o klasie `element`, wykorzystując wszystkie kolumny.

```
.element {
  grid-column-start: 1;
  grid-column-end: -1;
  grid-row-start: header;
  grid-row-end: header;
}
```

Jeśli element nie zajmuje całego rozmiaru wyznaczonych dla niego elementów siatki, możemy sterować jego położeniem w poziomie za pomocą `justify-self` oraz pionie za pomocą `align-self`. Możliwe wartości dla obu to:

- `start` — umieść po lewej/na górze;
- `center` — wyśrodkuj;
- `end` — umieść po prawej/na dole;
- `strech` — rozciągnij.

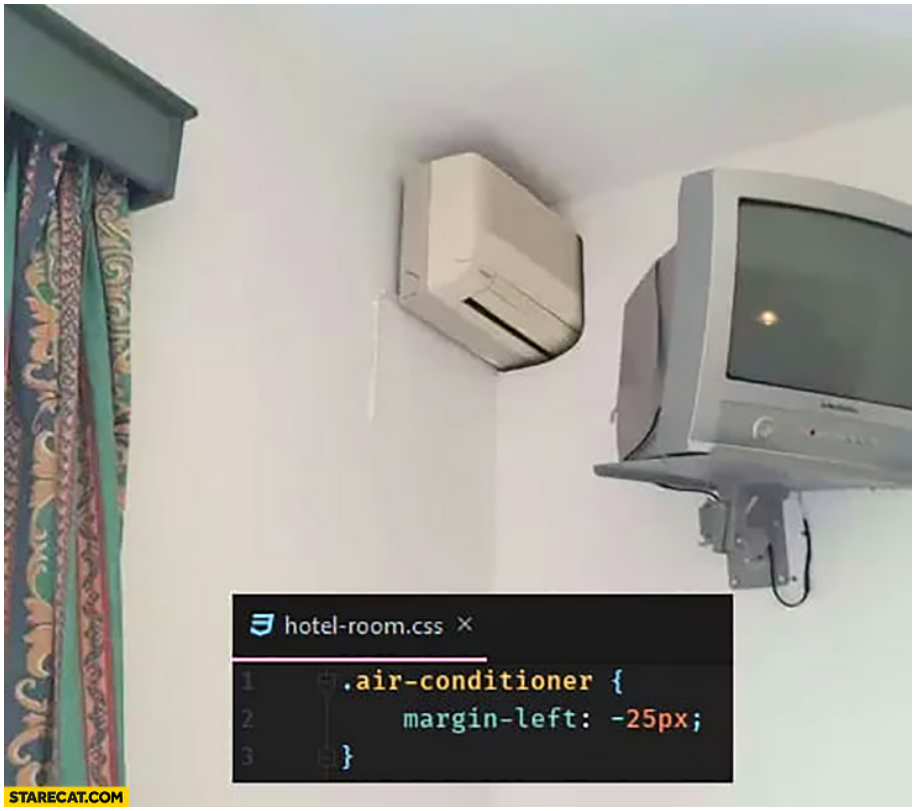
W sieci można znaleźć dużo materiałów. Przykładowe źródło to `https://css-tricks.com/snippets/css/complete-guide-grid/`.

4 Podsumowanie

CSS zmienił oblicze Internetu. Do tego stopnia, że w 2005 roku twórca standardu postanowił zrobić doktorat, opisując właśnie CSS. Jest on ogólnodostępny pod adresem `https://www.wiumlie.no/2006/phd/` i zawiera wiele przydatnych informacji — w tym porównanie z innymi (dziś czasem nawet zapomnianymi) językami specyfikacji stylu dokumentów.

Na koniec kilka rad:

1. zawsze sprawdzaj swój kod walidatorem `https://jigsaw.w3.org/css-validator/`;
2. upewniaj się, czy wykorzystane atrybuty CSS są powszechnie wspierane `https://caniuse.com/`;
3. korzystaj z narzędzi pomagających wybierać kolor lub gotowych palet `https://color.adobe.com/`;
4. oglądaj, co da się zrobić `https://duckduckgo.com/?q=best+css+design`;
5. podglądaj, jak robią to inni `https://www.thoughtco.com/get-inspect-element-tool-for-browser-756549`;
6. szukaj informacji w sieci `https://www.w3schools.com/css/`;
7. zaglądaj do porządnych książek `https://learning.oreilly.com/library/view/css-in-depth/9781617293450/`, `http://book.webtypography.net/`;
8. eksperymentuj;
9. bądź ostrożny…



Źródło:
`https://starecat.com/hotel-room-css-air-conditioner-between-rooms-negative-css-margin/`.

10. …, ale gdy coś nie wyjdzie, nie panikuj — wróć do punktu 1.