Przegląd właściwości CSS

Kaskadowe arkusze stylów

CSS (ang. Cascading Style Sheets) to tak zwane kaskadowe arkusze stylów i jak wiemy służą one do opisania wyglądu elementów witryny, uprzednio zdefiniowanych w HTML. Kodem CSS możemy także wpływać na położenie elementów i wzajemne relacje między nimi. Natomiast w związku z taką definicją, pojawia się nam w głowach naturalne pytanie: W jaki sposób w praktyce dokonać przypisania stylów do konkretnego znacznika HTML? Otóż aby nadać style potrzebujemy ten konkretny element uchwycić (czyli wybrać spośród wszystkich innych) – jako uchwyty stosujemy w CSS tzw. selektory.

Selektory, atrybuty, wartości

Ogólny schemat zapisywania kodu CSS prezentuje się następująco:

```
selector
{
   property: value;
}
```

Selektor (jak sama nazwa wskazuje: selekcja = wybór) określa jednoznacznie do których elementów z kodu HTML zostaną zastosowane właściwości podane w tzw. ciele selektora (czyli pomiędzy nawiasami klamrowymi). Każda właściwość (ang. property = właściwość, atrybut, cecha) określa jakiś aspekt wyglądu elementu, zaś wartość tego atrybutu (ang. value) zapisana jest po dwukropku i zakończona średnikiem. Oczywiście najlepiej jest zobaczyć przykładowy zapis, zamiast ogólnej konwencji:

```
body
{
   background-color: gray;
   font-family: Arial;
}
```

W tym przypadku naszym selektorem uchwyciliśmy sekcję <body> czyli całe ciało witryny, nadając jej szary kolor tła oraz krój czcionki Arial. Znaki białe (spacje, przejścia do nowej linii) są opcjonalne - można je wyciąć, przygotowując tzw. "wersję min" arkusza. Wówczas plik zajmuje mniej miejsca (nieco szybciej się wczytuje internaucie przez internet), ale oczywiście taki arkusz siłą rzeczy jest mało czytelny. Najczęściej więc pracujemy na zwykłym arkuszu, a "wersję min"

przygotowujemy jedynie po zakończeniu pracy nad witryną (tzw. "wersja produkcyjna kodu", która trafia na serwer). Zapis pozbawiony znaków białych wyglądałby tak:

```
body{background-color:gray;font-family:Arial;}
```

Oczywiście "uchwycić" w CSS możemy nie tylko znaczniki – samo posługiwanie się tagami szybko okazałoby się niewystarczające – na przykład taki zapis:

```
div
{
   width: 350px;
}
```

Taki selektor złapie każdego jednego diva na stronie i ustawi mu 350 pikseli szerokości. Nie jest to sytuacja w żaden sposób wygodna, gdyż zazwyczaj struktura divów składa się z bardzo wielu różnych bloków, które siłą rzeczy potrzebują zróżnicowanych szerokości. A zatem musimy mieć w CSS możliwość złapania pojedynczego znacznika div, jednego konkretnego pojemnika (lub ewentualnie kilku, jeżeli zdarzy się tak, że jakieś bloki rzeczywiście mogą, albo nawet powinny mieć tę samą szerokość). I jak zapewne wiemy, do celu uchwycenia konkretnych elementów służy w arkuszach stylów albo klasa (ang. class) albo identyfikator (ang. ID).

Identyfikatory

Identyfikatory ID mogą zostać użyte do uchwycenia tylko jednego elementu, za to w sposób unikalny – nie mogą istnieć w kodzie HTML dwa elementy o takiej samej wartości atrybutu id. Zapis prezentuje się następująco:

Arkusz stylów CSS

```
#container
{

}
```

Kod HTML

```
<div id="container">

</div>
```

Jak widać powyżej, w przypadku zastosowania identyfikatora, selektor w CSS dodatkowo poprzedzamy znakiem "#".

Jeśli chodzi o nazwę identyfikatora (tutaj jest to "container"), to w HTML4 identyfikator musi zaczynać się literą (a-z A-Z), a po niej może nastąpić dowolna ilość liter, cyfr, myślników, podkreśleń, dwukropków, kropek. Zaś w HTML5 identyfikator musi zawierać przynajmniej jeden znak i nie może zawierać żadnej spacji.

Unikalność identyfikatora w HTML jest wymagana i obsługiwana na poziomie tzw. hierarchii DOM (ang. Document Object Model). Zapamiętajmy to dobrze: w języku HTML nie mogą istnieć dwa atrybuty id o takiej samej wartości (nazwie), nawet jeżeli byłyby to atrybuty określone dla różnych znaczników (na przykład dla diva i dla paragrafu):

```
źle!  <div id="pierwszy"></div>   <p id="pierwszy"></p>
```

Atrybut ID musi jednoznacznie (unikalnie) identyfikować jeden konkretny element witryny. Tylko żebyśmy się dobrze rozumieli! Reguła unikalności identyfikatora mówi o tym, że nie mogą w HTML istnieć dwa identyfikatory o tej samej wartości, natomiast w HTML może istnieć wiele identyfikatorów, pod warunkiem, że każdy z nich ma inną wartość:

```
ok!  <div id="pierwszy"></div>   <p id="trzynasty"></p>
```

Klasy

Klasy mogą zostać użyte do uchwycenia dowolnej liczby elementów (w tym także jednego!). Zapis prezentuje się następująco:

Arkusz stylów CSS

```
.container
{

}
```

Kod HTML

```
<div class="container">

</div>
```

Jak widać powyżej w przypadku zastosowania klas, selektor w CSS dodatkowo poprzedzamy znakiem kropki. Częstym błędem osób początkujących jest zakładanie, iż skoro za pomocą identyfikatora możemy uchwycić tylko jeden element, to w takim razie za pomocą klas można uchwycić tylko i wyłącznie kilka obiektów. Nieprawda - elementów posiadających atrybut class może być w HTML dowolna ilość, w tym także tylko jeden!

Kiedy użyć identyfikatora, a kiedy klasy?

Jeżeli łapiemy dwa lub więcej elementów w CSS, to siłą rzeczy pozostaje nam użyć klasy (gdyż nadanie tego samego id dwóm lub więcej elementom łamie regułę unikalności identyfikatora - duży błąd!). Jednak co w przypadku, gdy chcemy uchwycić tylko jeden element – skoro możemy użyć zarówno identyfikatora jak i klasy, to który sposób jest lepszy?

Cóż, to złożony problem. Ustanowienie identyfikatora w HTML wymusza na przeglądarce obsługę mechanizmu unikalności tegoż elementu (jest on wówczas wyróżniony w specjalny sposób w tzw. hierarchii DOM). Jeżeli jedynym naszym celem jest ostylować element, to ustanowienie unikalnego id wydaje się być niepotrzebną, lekką przesadą.

Identyfikator powinniśmy zastosować przede wszystkim w tych elementach, które zamierzamy później uchwycić w skryptach JavaScript, albo które będą służyć jako tzw. punkty nawigacyjne witryny (kotwice nawigacyjne).

Reasumując: stosowanie identyfikatora tylko do celów ostylowania elementu w CSS jest niepotrzebnym "mieszaniem" w hierarchii DOM. Mówiąc najprościej: nie jest to optymalne. Natomiast absolutnie nie jest to błędem! Specyfikacja W3C HTML5 (źródło) mówi wyraźnie: Element posiadający unikalny identyfikator może zostać użyty do różnorodnych celów - przede wszystkim jako sposób linkowania do konkretnych części dokumentu (kotwica nawigacyjna), jako uchwyt dla skryptów JS oraz do ostylowania konkretnego elementu w CSS.

Metody dołączenia stylów CSS do HTML

Istnieją trzy główne sposoby zainkludowania zapisów CSS do dokumentu HTML. W związku z tym istnieje także hierarchia ważności stylów (w przypadku powtarzających się selektorów, to metoda podpięcia zdecyduje o tym, który styl obowiązuje – zasada ogólna jest taka, iż to najpóźniej zdefiniowany w kodzie styl jest uważany za obowiązujący).

Sposoby inkludowania stylów:

1. Zewnętrzny arkusz CSS dołączony do dokumentu HTML znacznikiem <link> znajdującym się w sekcji <head> podstrony (wartość atrybutu href musi wskazywać istniejący plik):

```
<link rel="stylesheet" href="main.css">
```

2. Kod CSS osadzony pomiędzy tagami <style></style> również umieszczonymi w <head>

```
<style>

#container
{
   color: red;
}

</style>
```

3. Stylizowanie inline (ang. "w linii"), czyli wewnątrz atrybutu "style" wybranego tagu HTML, znajdującego się w sekcji <body>

```
<div id="container" style="color: blue;">

</div>
```

Hierarchia ważności poszczególnych sposobów to: 3 > 2 > 1. Style inline przesłaniają te spomiędzy tagów <style>, zaś one przesłaniają właściwości zdefiniowane w zewnętrznym arkuszu (o ile podpięcie pliku tagiem <link> znalazło się przed tagami <style>). Hierarchię ważności można także zmienić stosując po spacji (a przed średnikiem) klauzulę: !important;

```
#container
{
   color: red !important; /* wymuszenie koloru czerwonego */
}
```

Oczywiście żeby uniknąć nawarstwiania się chaosu, nie należy przesadzać z wyróżnianiem niektórych właściwości – czasem się to przydaje – jasne! Jednak należy zachować umiar.

Operator myślnika, subatrybuty właściwości głównych

Znak myślnika "-" oddaje hierarchię właściwości: najpierw następuje właściwość główna (background), następnie operator myślnika, a potem subatrybut właściwości głównej (color):

```
body { background-color: gray; }
```

Za każdym razem, kiedy widzisz w CSS myślnik, zastosuj prostą regułę: idąc zawsze od prawej strony zapisu do lewej, zadawaj sobie pytania o napotykane właściwości, aż do momentu, gdy dotrzesz do selektora. W naszym przypadku mamy: kolor [czego?] tła [tła czego?] sekcji body. Im bardziej coś jest po lewej stronie w CSS, tym wyżej znajduje się w tej kaskadzie. To właśnie z tego powodu zapisy CSS najłatwiej jest czytać "od prawej do lewej".

Wyśrodkowanie elementu, automatyczne marginesy

Element można wyśrodkować na ekranie, z użyciem znanej sztuczki:

```
#container
{
   margin-left: auto;
   margin-right: auto;
}
```

Pamiętajmy o tym – ustawienie automatycznych marginesów z lewej i prawej strony powoduje, iż przeglądarka zawsze wyśrodkuje diva względem szerokości dostępnego płótna (niezależnie od rozmiaru ekranu – nawet przy zmianie wielkości okna pojemnik ułoży się na środku ekranu).

Tło elementu: background

Określamy tło elementu. Zapis z użyciem tylko słowa "background" stanowi tak naprawdę skrócony zapis (główną właściwość) – tak naprawdę istnieje wiele subatrybutów, które można zmienić.

Kolor ustawionego tła – po dwukropku podajemy poprawny zapis koloru (stałą tekstową, notację szesnastkową lub dziesiętną – patrz rozdział o zapisie barw w CSS).

```
background-color: yellow;
```

Obraz tła – wewnątrz nawiasów podajemy prawidłową ścieżkę dostępu do grafiki. Nie można zapomnieć o użyciu url (ang. uniform resource locator).

```
background-image: url("img/dark.jpg");
```

Powtarzanie tła – określamy czy nasz obraz tła ma zostać użyty do wykafelkowania całego dostępnego okna przeglądarki (repeat), czy jednak powtarzanie ma następować tylko w jednej osi (repeat-x, repeat-y), bądź ma zostać wyłączone (no-repeat).

```
background-repeat: no-repeat;
```

Pozycja tła – jako wartość position możemy zapisać kombinację dwóch słów (spośród takiego zestawu: left, right, top, bottom, center) albo użyć wartości procentowych. Pierwsze słowo określa jak tło zachowa się w poziomie, a drugie w pionie (uwaga: nie wstawiamy pomiędzy te słowa myślnika!). Do przetestowania działania tego atrybutu w przeglądarce warto tymczasowo wyłączyć powtarzanie tła (patrz właściwość powyżej).

```
background-position: center top;
background-position: 25% 75%;
```

Rozmiar tła – jak wielki ma być obraz tła? Możemy użyć słów kluczowych: auto (dopasowanie automatyczne realizowane przez przeglądarkę), cover (obraz możliwe największy), contain (przeskaluj obraz do największego rozmiaru, ale tak aby jego szerokość i wysokość zmieściły się na płótnie – chodzi o zachowanie oryginalnej proporcji obrazu), albo użyć wartości procentowych, wyrażonych w pikselach, bądź w jednostce em.

```
background-size: auto auto;
background-size: cover;
background-size: contain;
background-size: 75%;
background-size: 24px;
background-size: 1.5em;
background-size: 1.5em;
```

Przymocowanie tła - jeśli ustawiliśmy grafikę tła, to z pomocą background-attachment możemy zdecydować, jak grafika zachowa się w oknie przeglądarki. Możliwe wartości: scroll (obraz tła przewija się wraz z zawierającym go pojemnikiem), fixed (ustalone przymocowanie – obraz tła nie będzie przewijać się wraz z zawierającym go pojemnikiem).

```
background-attachment: fixed;
background-attachment: scroll;
```

Krój czcionki: font-family

Ustawienie kroju (rodziny) czcionki - jako wartość podajemy nazwę czcionki. Obecnie często korzysta się z tzw. Google Fonts (czcionek Google), jednak ich ustawienie wymaga dołączenia dodatkowego arkusza stylów. Szczegóły w tym odcinku na YouTube. Zwróćmy też uwagę, że w wielu projektach krój czcionki warto ustawić w sekcji body - wówczas jednokrotne wpisanie font-family sprawia, iż ta rodzina czcionek będzie obowiązywała wszędzie w podstronie, dopóki w jakimś selektorze tego nie zmienimy.

```
body
{
   font-family: Arial;
}
```

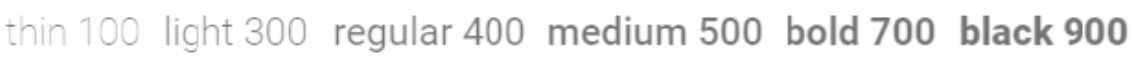
Rozmiar czcionki: font-size

Właściwość pozwalająca określić rozmiar tekstu. Często stosowane, zalecane jednostki wielkości czcionki to: px, %, em, ex. Niezalecane jednostki wartości to z kolei: pt, cm, mm, in, pc. Więcej na temat jednostek przeczytasz tutaj. Można także użyć stałych tekstowych: xx-small, x-small, small, medium, large, x-large, xx-large (small = mały, medium = średni, large = duży, x = "extra", xx = "double extra") lub określić rozmiar relatywnie do elementu nadrzędnego: smaller (mniejszy rozmiar tekstu niż w elemencie nadrzędnym), larger (większy rozmiar).

```
font-size: 22px;
font-size: 2em;
font-size: x-small;
font-size: larger;
```

Waga czcionki: font-weight

Ustawiając wagę czcionki regulujemy "tłustość" tekstu. Możemy użyć stałych tekstowych: normal (standardowa grubość tekstu), bold (czcionka pogrubiona). Oprócz tego możemy użyć wartości liczbowej od 100 do 900, o ile czcionka obsługuje poszczególne grubości. Wartości zmieniamy zawsze tylko o pełne 100, przy czym 400 odpowiada wartości normal, a 700 to czcionka bold. Przykładowe zobrazowanie niektórych wartości liczbowych odnajdziemy poniżej (zrzut ekranu z Google Fonts).



Istnieje także możliwość określenia rozmiaru relatywnie do elementu nadrzędnego: lighter (mniejsza grubość tekstu), bolder (tekst grubszy niż w elemencie nadrzędnym).

```
font-weight: bold;
font-weight: 700;
font-weight: lighter;
```

Stylizacja czcionki: font-style

Nieco rzadziej obecnie używana właściwość - mamy wartości normal, italic albo oblique. Czcionka italic jest (najprościej mówiąc) delikatnie pochylona w prawo, zaś oblique nadaje jeszcze większe pochylenie. Na ilustracji poniżej przedstawiono kolejno ten sam akapit tekstu, jednak zastosowano inny font-style – kolejno: normal, italic, oblique.

I have no special talent. I am only passionately curious. Albert Einstein

I have no special talent. I am only passionately curious. Albert Einstein

I have no special talent. I am only passionately curious. Albert Einstein

```
font-style: normal;
font-style: italic;
font-style: oblique;
```

Wyrównanie tekstu: text-align

Atrybut ten pozwala nam określić w jaki sposób wewnętrzna zawartość elementu blokowego (w tym na przykład tekst) zostanie ułożona w tym pojemniku. Popularne wartości to przede wszystkim: left, center, right, justify. Ta ostatnia to tzw. justowanie tekstu, czyli tekst będzie wówczas tak poukładany, żeby zajmował całą dostępną przestrzeń.

Inne możliwe wartości to: justify-all (to samo co justify, z tym wyjątkiem, że ostatnia linia tekstu także ulegnie takiemu wyrównaniu), start (to samo co left, chyba że tekst ma być czytany od prawej do lewej – w niektórych językach tak jest, decyduje o tym wartość właściwości direction: rtl; albo direction: ltr;), end (wyrównanie w prawo, równie dopasowujące się do wartości direction i w razie potrzeby zmieniające się na lewe), match-parent (wartość dopasowana do elementu nadrzędnego i również reagująca na wartość direction).

```
text-align: center;
```

Ilustracja poniżej przedstawia cztery podstawowe sposoby ułożenia zawartości.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut id nunc ipsum. Etiam dapibus magna ut turpis tincidunt venenatis. In nulla elit, feugiat sed ipsum id, vestibulum ultrices est. Sed id ex feugiat, elementum velit ac, semper elit. Pellentesque viverra eleifend ipsum, vel convallis ante ullamcorper sit amet. Etiam quis urna in velit bibendum iaculis.

text-align: left;

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut id nunc ipsum. Etiam dapibus magna ut turpis tincidunt venenatis. In nulla elit, feugiat sed ipsum id, vestibulum ultrices est. Sed id ex feugiat, elementum velit ac, semper elit. Pellentesque viverra eleifend ipsum, vel convallis ante ullamcorper sit amet. Etiam quis urna in velit bibendum iaculis.

text-align: right;

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut id nunc ipsum. Etiam dapibus magna ut turpis tincidunt venenatis. In nulla elit, feugiat sed ipsum id, vestibulum ultrices est. Sed id ex feugiat, elementum velit ac, semper elit. Pellentesque viverra eleifend ipsum, vel convallis ante ullamcorper sit amet. Etiam quis urna in velit bibendum iaculis.

text-align: center;

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut id nunc ipsum. Etiam dapibus magna ut turpis tincidunt venenatis. In nulla elit, feugiat sed ipsum id, vestibulum ultrices est. Sed id ex feugiat, elementum velit ac, semper elit. Pellentesque viverra eleifend ipsum, vel convallis ante ullamcorper sit amet. Etiam quis urna in velit bibendum iaculis.

text-align: justify;

Zmiana wielkości liter: text-transform

CSS umożliwia zmianę oryginalnej wielkości liter, co szczególnie przydaje się w kontekście pozycjonowania (litery nie muszą być zapisane jako duże w kodzie HTML, aby móc zostać zaprezentowane jako takie w przeglądarce). Możliwe wartości to przede wszystkim: uppercase (zamień wszystkie litery na duże), lowercase (zamień wszystkie litery na małe), capitalize (wielkie litery na początku wyrazów), none (brak zamiany wielkości).

```
text-transform: uppercase;
```

Na ilustracji poniżej przedstawiono kolejno ten sam akapit tekstu, jednak zastosowano inny rodzaj transformacji tekstu – kolejno: uppercase, lowercase, capitalize:

IT DOES NOT MATTER HOW SLOWLY YOU GO AS LONG AS YOU DO NOT STOP. CONFUCIUS
it does not matter how slowly you go as long as you do not stop. confucius
It Does Not Matter How Slowly You Go As Long As You Do Not Stop. Confucius

Dekoracja tekstu: text-decoration

Właściwość przydatna do ustalenia sposobu dekoracji tekstu (z użyciem linii podkreślającej, przekreślającej czy znajdującej się nad tekstem), w praktyce często stosowana dla określania wyglądu linków (z racji tego, że linki często bywają podkreślone, choć absolutnie nie tylko do tego może posłużyć). Możliwe sposoby dekoracji określone są przez subatrybuty:

- text-decoration-color (kolor linii wyróżniającej),
- text-decoration-style (rodzaj linii wyróżniającej: solid, double, dotted, dashed, wavy),
- text-decoration-line (położenie linii wyróżniającej: underline, overline, line-through, blink, underline overline, none).

Bardzo często w kodzie CSS stosujemy jednak zapis skrócony, czyli jedynie z użyciem zapisu text-decoration, odpowiednio dobierając liczbę parametrów.

```
text-decoration: underline;
text-decoration: underline wavy red;
```

Odstęp pomiędzy znakami: letter-spacing

Odstęp taki możemy określić z użyciem od jednej do trzech wartości (kolejno: minimalny, maksymalny i optymalny odstęp między znakami).

```
letter-spacing: 2px;
letter-spacing: 0.5em;
letter-spacing: 0.5em 1em 0.7em;
```

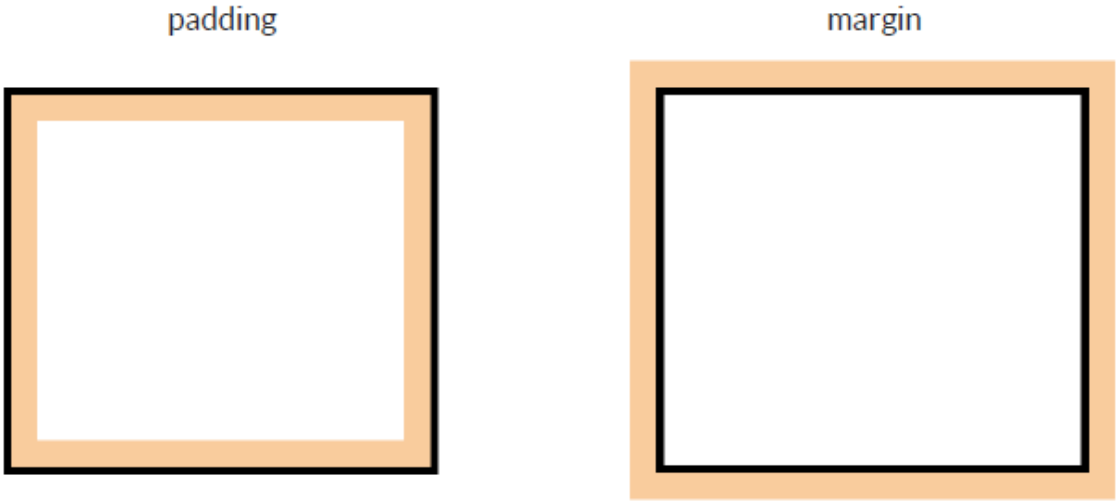
Wysokość linii tekstu: line-height

Ustawiamy wysokość pojedynczej linii tekstu, z użyciem: stałej normal, liczby określającej mnożnik aktualnej wysokości, wysokości podanej w procentach lub pikselach.

```
line-height: 2;
line-height: 25px;
line-height: 140%;
```

Nadawanie odstępów: margin i padding

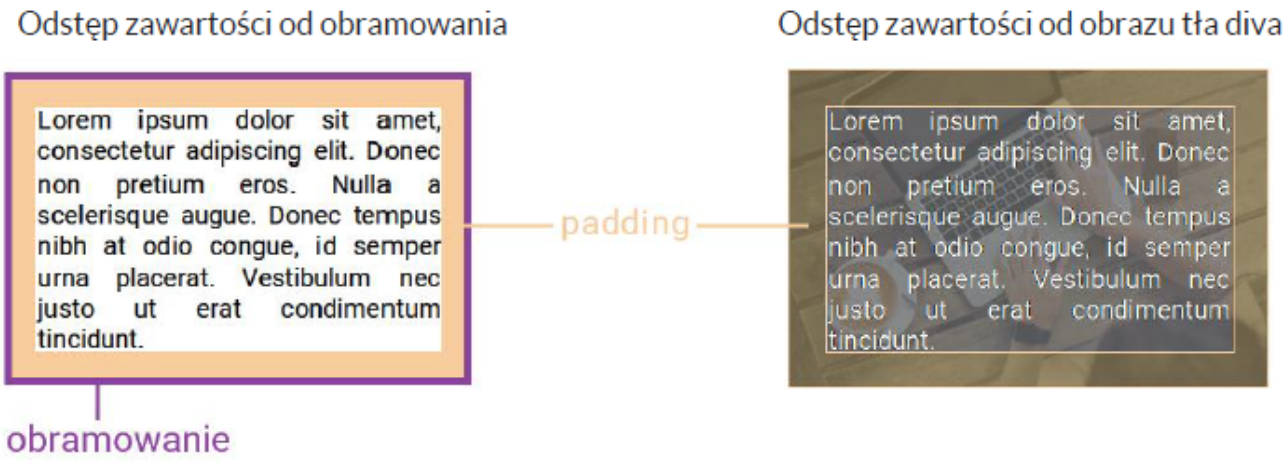
Właściwość padding to odstęp nadawany wewnątrz elementu, zaś margin to jest odstęp nadawany na zewnątrz niego.



W praktyce bardzo często żonglujemy użyciem tak marginesów i paddingów, gdyż nie ma pomiędzy nimi wielu znaczących różnic. No, poza drobnymi niuansami. Najlepszym przykładem takiej różnicy są pionowe marginesy, które w odróżnieniu od paddingu nakładają się na siebie, zmniejszając odstęp pomiędzy zawartościami obu divów. Weźmy jako przykład dwa divy, ułożone jeden pod drugim. Jeżeli oba będą miały ustawione 10 pikseli paddingu, no to oznacza to, że w pionie mamy zawsze taką sytuację, że mamy zawartość górnego diva, 10 pikseli wewnętrznego odstępu, potem drugie tyle i dopiero na końcu zawartość drugiego diva.

A jeżeli byśmy użyli marginesów (również 10-cio pikselowych), to sytuacja troszkę się zmieni - zachodzi tu nakładanie się pionowych odstępów na siebie, a zatem w praktyce odstęp pomiędzy zawartościami obu divów będzie wynosić 10 pikseli, a nie 20. Podkreślam - w przypadku marginesów pionowych, nie poziomych.

Ponadto, jeżeli mielibyśmy ustawione obramowanie jakiegoś elementu, i chcielibyśmy odsunąć zawartość diva od obramowania, to nie mamy wyjścia - odstęp musi być wewnętrzny, czyli trzeba użyć paddingu. Albo kiedy mamy ustawione tło diva jako obraz i chcemy odsunąć tekst od krawędzi - również nie mamy wyjścia – zostaje padding. Natomiast to nie jest tak, że padding jest lepszy - nie, w praktyce to marginesów używa się statystycznie częściej, żeby wstawiać odstępy – aczkolwiek to zależy mocno od osoby i jej osobistych preferencji.



Możliwe zapisy marginesów – najczęściej używamy pikseli lub wartości procentowych

1. Taki sam margines określony z wszystkich stron:

```
margin: 10%;
```

2. Zapis podwójny: najpierw margines górny i dolny, potem lewy i prawy

```
margin: 20px 10px;
```

3. Zapis potrójny: najpierw margines górny, potem lewy i prawy (ten sam), a na końcu podajemy wartość marginesu dolnego

```
margin: 20px 5% 5px;
```

4. Zapis poczwórny: margines górny, prawy, dolny, lewy (zgodnie z ruchem wskazówek zegara)

```
margin: 10px 15px 20px 5px;
```

5. Margines z jednej, wybranej strony (subatrybuty left, right, top, bottom)

```
margin-left: 5px;
margin-right: 15px;
margin-top: 10px;
margin-bottom: 20px;
```

Możliwe zapisy paddingu – najczęściej używamy pikseli lub wartości procentowych

1. Taki sam padding określony z wszystkich stron:

```
padding: 10%;
```

2. Zapis podwójny: najpierw padding górny i dolny, potem lewy i prawy

```
padding: 20px 15px;
```

3. Zapis potrójny: najpierw padding górny, potem lewy i prawy (ten sam), a na końcu podajemy wartość dolnego paddingu

```
padding: 15px 30px 50px;
```

4. Zapis poczwórny: padding górny, prawy, dolny, lewy (zgodnie z ruchem wskazówek zegara)

```
padding: 10px 15px 20px 5px;
```

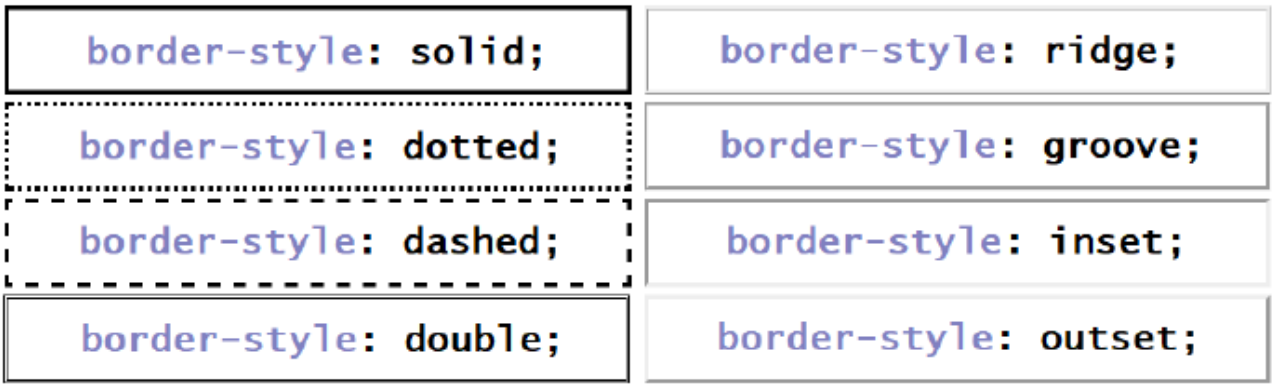
5. Padding z jednej, wybranej strony (subatrybuty left, right, top, bottom)

```
padding-left: 5px;
padding-right: 15px;
padding-top: 10px;
padding-bottom: 20px;
```

Jak widać możliwe zapisy paddingu są analogiczne jak możliwe zapisy marginesów – to dobra wiadomość – zapamiętanie przyjętej konwencji dla odstępów wewnętrznych automatycznie sprawia, iż znamy także zapisy odstępów zewnętrznych.

Obramowanie: border

Do ustawienia obramowania używamy właściwości border (ang. border = obwódka na krawędzi, obszar graniczny elementu). Możemy zastosować trzy podstawowe subatrybuty: border-width (szerokość obramowania w pikselach), border-style (rodzaj linii stanowiącej obwódkę - patrz ilustracja poniżej) oraz border-color (kolor obwódki).



Natomiast w praktyce, często nie piszemy następujących po sobie trzech kolejnych właściwości, tylko stosujemy zapis skrótowy, pisząc jedynie słowo border, a po dwukropku trzy wartości rozdzielone spacją i zakończone oczywiście średnikiem. Najpierw podajemy szerokość w pikselach, potem rodzaj linii i na końcu jej kolor.

```
border: 2px solid green;
```

Można także użyć subatrybutów: left, right, top, bottom. Ponadto istnieje właściwość border-image, pozwalająca użyć obrazu jako obramowania (więcej na ten temat znajdziesz np. tutaj).

```
border-left: 1px solid black;
border-right: 1px dashed blue;
border-top: 1px groove green;
border-bottom: 1px dotted yellow;
border-image: url("obraz.png") 30 round;
```

Stylizowanie list <ul> oraz <ol>: list-style

Wygląd list numerowanych i nienumerowanych ustalamy głównie właściwością list-style. Możliwe subatrybuty to list-style-type (typ listy – patrz ilustracja poniżej: disc, circle, square, decimal, decimal-leading-zero, lower-roman, upper-roman, lower-greek, lower-latin, upper-latin, armenian, georgian, lower-alpha, upper-alpha), list-style-image (obraz markera listy z użyciem url) oraz list-style-position (pozycja markera listy: inside, outside).

armenian	circle	decimal	georgian	decimal-leading-zero	lower-alpha	lower-greek	lower-roman	square	upper-alpha	upper-roman
U.	◦	1.	ა.	01.	a.	α.	i.	■	A.	I.
Բ.	◦	2.	ბ.	02.	b.	β.	ii.	■	B.	II.
Գ.	◦	3.	გ.	03.	c.	γ.	iii.	■	C.	III.
Դ.	◦	4.	დ.	04.	d.	δ.	iv.	■	D.	IV.
Ե.	◦	5.	ე.	05.	e.	ε.	v.	■	E.	V.

W praktyce często zamiast poszczególnych zapisów z subatrybutami stosujemy zapis złożony.

```
list-style-type: square;
list-style-image: url("marker.jpg");
list-style-position: outside;
list-style: square url("marker.jpg") outside;
```

Pozostało nam jeszcze omówić kolejną ważną w praktyce kwestię, a mianowicie: W jaki sposób uchwycić pojedynczy element listy, czyli znacznik ? Możemy użyć dwóch rodzajów selektorów:

Z użyciem spacji

```
ul li
{

}
```

Z użyciem operatora ">"

```
ul > li
{

}
```

Na czym polega różnica? Za pomocą spacji uchwycimy wszystkie elementy wewnątrz listy ul, zaś operatorem ">" tylko elementy leżące bezpośrednio w znacznikach <ul>. Dla lepszego zrozumienia tej różnicy rozważmy następującą strukturę:

```
<div class="pojemnik">
   <span><p>Jeden</p></span>
   <span><p>Dwa</p></span>
   <p>Trzy</p>
   <p>Cztery</p>
</div>
```

Zależnie od zastosowanego selektora otrzymamy inne rezultaty:

Z użyciem spacji

```
.pojemnik  p
{
   color:red;
}
```

Z użyciem operatora ">"

```
.pojemnik > p
{
   color:red;
}
```

Rezultaty w przeglądarce:

Jeden
Dwa
Trzy
Cztery

Pierwsze dwa paragrafy nie są czerwone, ponieważ nie leżą bezpośrednio w divie pojemnik (zostały umieszczone w spanach).

Jeden
Dwa
Trzy
Cztery

Wszystkie paragrafy są czerwone, ponieważ wcale nie muszą leżeć bezpośrednio w divie pojemnik (mogą w spanach, a nawet głębiej).

Stylizowanie linków z użyciem pseudoklas

Domyślny styl linków to podkreślony tekst koloru niebieskiego (jeżeli ten link jeszcze nie był odwiedzony), albo koloru fioletowego (w przypadku adresu, który już odwiedziliśmy). I to jest pierwsza, najważniejsza obserwacja: hiperłącza posiadają swój własny niezależny styl, który jest nakładany na tekstu wewnątrz znaczników <a>. Krój czcionki pozostaje ten sam, ale kolor i podkreślenie zostają nadane, aby uwidocznić w ten prosty sposób, iż dany tekst jest na stronie hiperłączem.

Do stylizowania wylądu linków najlepiej użyć tzw. pseudoklas, które dopisujemy do selektora po dwukropku. W naszym przypadku linkom nadano także klasę o nazwie "link1", co w praktyce umożliwia stworzenie kilku rodzajów hiperłączy na jednej podstronie. Natomiast pseudoklasy to kolejno:

- :link (hiperłącze nieodwiedzone, ang. unvisited link),
- :visited (hiperłącze odwiedzone, ang. visited link),
- :hover (hiperłącze, na którym jest kursor ang. mouse over link),
- :active (hiperłącze, na który właśnie klikamy ang. selected link).

```
a.link1:link
{
   color: green;
   text-decoration: underline;
}

a.link1:visited
{
   color: green;
}

a.link1:hover
{
   color: blue;
}

a.link1:active
{
   color: red;
}
```

Zapis kolorów w CSS

Do tej pory używaliśmy jedynie słów angielskich określających kolory: white, black, red, green, blue etc. Jest to jednak tylko jeden ze sposobów - na pewno zauważyliśmy niejednokrotnie w różnych arkuszach istnienie zapisu szesnastkowego. Wyjaśnijmy go teraz! Taki zapis składa się z trzech składowych, poprzedzonych znakiem "#".

Każda składowa zajmuje maksymalnie dwa znaki - składowa czerwona R (ang. red), zielona G (ang. green) i niebieska B (ang. blue). Wartości każdej składowej są z przedziału od 00 do maksymalnie FF. Wartość "F" to szesnastkowo 15, co w praktyce daje 16 możliwości (od 0 do 15 to razem 16). Oznacza to, że każda składowa daje 16x16 możliwości kolorów czyli jest dziesiętnie z przedziału od 0 do 255. Łącznie mamy zatem 256x256x256 = 16777216 możliwych barw. Istnieje też zapis skrócony, tej 16-stkowej wersji. Jeżeli jakiś kolor składa się z tych samym sześć cyfr, no to możemy zapisać tylko trzy, a przeglądarka też zrozumie o co nam chodzi (#FFFFFF = #FFF albo #555555 = #555).

Jak również istnieje zapis dziesiętny w CSS – notacja z rgb i nawiasami oraz dziesiętnymi wartościami składowych rozdzielonych przecinkami:

Zestawienie przykładowych zapisów koloru białego:

```
background-color: white;
background-color: #FFFFFF;
background-color: #FFF;
background-color: rgb(255,255,255);
```

Nieprzezroczystość: opacity

Ustawienie stopnia przezroczystości elementu – jeżeli wartość opacity jest równa 1 to mamy pełną nieprzezroczystość (element jest całkowicie widoczny), zaś opacity równe 0 oznacza zerową nieprzezroczystość (element jest całkowicie niewidoczny). Oczywiście to tylko wartości skrajne, w praktyce mamy do dyspozycji wszystkie wartości z przedziału od 0 do 1.

W przypadku wartości mniejszych od 1, zamiast pisać np. 0.7 możemy zapisać wartość bez poprzedzającego zera (ale z kropką! – patrz ramka z kodem poniżej), a przeglądarka internetowa i tak zrozumie nasz zapis jako 0.7.

```
opacity: 0;
opacity: 1;
opacity: 0.7;
opacity: .7;
```

Wpływanie na pozycję elementu: position

Możemy wpływać na położenie elementów (czyli ich pozycję na ekranie względem wybranego układu odniesienia), z użyciem właściwości: left, right, top, bottom. Przyjrzyjmy się dostępnym rodzajom pozycjonowania elementów:

```
position: static;
```

Domyślna wartość position to "static" - pozycja statyczna. Przy takim ustawieniu właściwości left, right, top czy bottom są ignorowane, w ogóle nie będą działać. Dopiero jeżeli ustawimy pozycję relative, absolute albo fixed to właściwości left, right, top, bottom zadziałają.

```
position: relative;
```

Stosując pozycjonowanie relatywne elementu pozostawiamy przewidziane dla niego miejsce we flow witryny (miejsce to staje się jednocześnie nowym układem odniesienia dla elementu), po czym aplikujemy nowe położenie (określone właściwościami left, right, top lub bottom). W strukturze witryny zawsze pozostaje "wyrwa" w miejscu sprzed przemieszczenia.

```
position: absolute;
```

Stosując pozycjonowanie absolutne elementu nie pozostawiamy przewidzianego dla niego miejsca we flow witryny (nowym układem odniesienia staje się lewy górny narożnik płótna przeglądarki, no chyba że element pozycjonowany absolutnie znajduje się wewnątrz pojemnika pozycjonowanego relatywnie), po czym aplikujemy nowe położenie (określone właściwościami left, right, top lub bottom). W strukturze witryny nie pozostaje żadna "wyrwa" w miejscu przewidzianym pierwotnie we flow witryny (element zostaje całkowicie z tego flow wyjęty).

```
position: fixed;
```

Podobnie jak w pozycjonowaniu absolutnym nie pozostawiamy we flow witryny miejsca przewidzianego na element, natomiast różnica polega na tym, iż pozycja określona przez współrzędne left, right, top lub bottom utrzymuje się także przy przewijaniu dokumentu (w praktyce położeni – jak to mówimy – "ustala się" permamentnie.