WYKAZY

Typ stylu

```
selektor { list-style-type: typ }
```

Selektorem *mogą być znaczniki dotyczące wykazów:* `ul` *- wypunktowanie,* `ol` *- wykaz numerowany oraz* `li` *- pojedynczy punkt wykazu [zobacz: Wstawianie stylów].*

*Natomiast "**typ**" odpowiada za wygląd wyróżnika wykazu (markera) i należy zamiast niego wpisać:*

1. `disc` - koło

   PRZYKŁAD:

   list-style-type: disc

   - Punkt pierwszy
   - Punkt drugi
   - Punkt trzeci

2. `circle` - okrąg

   PRZYKŁAD:

   list-style-type: circle

   - Punkt pierwszy
   - Punkt drugi
   - Punkt trzeci

3. `square` - kwadrat

   PRZYKŁAD:

   list-style-type: square

   - Punkt pierwszy
   - Punkt drugi
   - Punkt trzeci

4. `decimal` - liczby arabskie

   PRZYKŁAD:

   list-style-type: decimal

   0. Punkt pierwszy
   1. Punkt drugi
   2. Punkt trzeci

5. `lower-alpha` - małe litery

   PRZYKŁAD:

   list-style-type: lower-alpha

   . Punkt pierwszy
   a. Punkt drugi
   b. Punkt trzeci

6. `upper-alpha` - duże litery

   PRZYKŁAD:

list-style-type: upper-alpha

. Punkt pierwszy
A. Punkt drugi
B. Punkt trzeci

7. `upper-roman` - duże liczby rzymskie

PRZYKŁAD:

list-style-type: upper-roman

. Punkt pierwszy
I. Punkt drugi
II. Punkt trzeci

8. `lower-roman` - małe liczby rzymskie

PRZYKŁAD:

list-style-type: lower-roman

. Punkt pierwszy
i. Punkt drugi
ii. Punkt trzeci

9. `none` - brak wyróżnika (markera)

PRZYKŁAD:

list-style-type: none

- Punkt pierwszy
- Punkt drugi
- Punkt trzeci

Oprócz tego Netscape 6, Firefox oraz częściowo Opera 6, Chrome i MSIE 8.0 (wartości dla CSS 2.1) obsługują dodatkowo następujące typy:

- `decimal-leading-zero` ("prowadzące zero" np.: 01, 02, 03,...)
- `lower-latin` (małe łacińskie: a, b, c,... z) - to samo co lower-alpha
- `upper-latin` (duże łacińskie: A, B, C,... Z) - to samo co upper-alpha
- `lower-greek` (małe greckie: ἑ, ἡ, ἱ,...)
- `georgian` (an, ban, gan,..., he, tan, in, in-an,...)
- `armenian`
- `hebrew` (hebrajskie) (CSS 2.0)
- `cjk-ideographic` (CSS 2.0)
- `hiragana` (a, i, u, e, o, ka, ki,...) (CSS 2.0)
- `katakana` (A, I, U, E, O, KA, KI,...) (CSS 2.0)
- `hiragana-iroha` (i, ro, ha, ni, ho, he, to,...) (CSS 2.0)
- `katakana-iroha` (I, RO, HA, NI, HO, HE, TO,...) (CSS 2.0)

Następujące wartości `list-style-type` wchodzą w skład CSS 2.0, ale nie CSS 2.1: `hebrew`, `cjk-ideographic`, `hiragana`, `katakana`, `hiragana-iroha`, `katakana-iroha`.

UWAGA! Niestety Internet Explorer 7.0 nie interpretuje dodatkowych typów, natomiast MSIE 8.0 obsługuje pierwszą część (CSS 2.1) dodatkowych typów, ale tylko w trybie Standards Compliance.

WYKAZY

Zawijanie tekstu

```
selektor { list-style-position: pozycja }
```

***Selektorem** mogą być znaczniki dotyczące wykazów: `ul` - wypunktowanie, `ol` - wykaz numerowany oraz `li` - pojedynczy punkt wykazu [zobacz: Wstawianie stylów].*

*Natomiast "**pozycja**" określa, jak będą zawijane wiersze wykazu, które nie zmieszczą się w jednej linii. Możliwe są tutaj dwa przypadki:*

1. `outside` - wyróżniki na zewnątrz wykazu (domyślnie).

PRZYKŁAD:

list-style-position: outside

- Punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy.
- Punkt drugi... punkt drugi... punkt drugi... punkt drugi... punkt drugi... punkt drugi... punkt drugi... punkt drugi... punkt drugi... punkt drugi... punkt drugi... punkt drugi... punkt drugi... punkt drugi... punkt drugi.
- Punkt trzeci... punkt trzeci... punkt trzeci... punkt trzeci... punkt trzeci... punkt trzeci... punkt trzeci... punkt trzeci... punkt trzeci... punkt trzeci... punkt trzeci... punkt trzeci... punkt trzeci... punkt trzeci.

2. `inside` - wyróżniki wewnątrz wykazu.

PRZYKŁAD:

list-style-position: inside

- Punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy.
- Punkt drugi... punkt drugi... punkt drugi... punkt drugi... punkt drugi... punkt drugi... punkt drugi... punkt drugi... punkt drugi... punkt drugi... punkt drugi... punkt drugi... punkt drugi... punkt drugi... punkt drugi... punkt drugi.
- Punkt trzeci... punkt trzeci... punkt trzeci... punkt trzeci... punkt trzeci... punkt trzeci... punkt trzeci... punkt trzeci... punkt trzeci... punkt trzeci... punkt trzeci... punkt trzeci... punkt trzeci... punkt trzeci... punkt trzeci.

Wyróżnik obrazkowy

```
selektor { list-style-image: url(ścieżka dostępu) }
```

***Selektorem** mogą być znaczniki dotyczące wykazów: `ul` - wypunktowanie, `ol` - wykaz numerowany oraz `li` - pojedynczy punkt wykazu [zobacz: Wstawianie stylów].*

*Natomiast jako "**ścieżka dostępu**" należy wpisać względną ścieżkę do obrazka, który ma się pojawić jako wyróżnik wykazu (marker). Wpisanie `none` usunie obrazek.*

Ścieżkę dostępu należy konstruować względem arkusza CSS, a nie względem dokumentu HTML!

Polecenie pozwala zastosować dowolny obrazek jako wyróżnik wykazu (czyli marker). Dzięki temu nie musimy korzystać już tylko z podstawowych typów wykazu, ale możemy stworzyć swoje własne. Sprawi to, że nasza strona będzie wyglądała bardziej oryginalnie i estetycznie.

Wspólnie z tą własnością można również podać dodatkowo typ stylu. W takim przypadku, jeśli obrazek będzie niedostępny lub użytkownik wyłączy wyświetlanie obrazów, wykaz przyjmie podany typ.

PRZYKŁAD:

list-style-image: url(minus.gif)

- Punkt pierwszy
- Punkt drugi
- Punkt trzeci

WYKAZY

Atrybuty mieszane

```
selektor { list-style: wartości atrybutów }
```

***Selektorem** mogą być znaczniki dotyczące wykazów: `ul` - wypunktowanie, `ol` - wykaz numerowany oraz `li` - pojedynczy punkt wykazu [zobacz: Wstawianie stylów].*

*Natomiast jako "**wartości atrybutów**" należy wpisać kolejne wartości (oddzielone spacjami), jakie mają przyjąć poszczególne atrybuty wykazu. Są to:*

- *typ stylu*
- *zawijanie tekstu*

- *wyróżnik obrazkowy*

Tak jak przy wielu innych elementach, także i tutaj możliwe jest użycie atrybutów mieszanych. Jak zawsze zaoszczędzają one nam pisania. Zamiast wpisywać: `list-style-type: disc; list-style-position: inside`, wystarczy podać: `list-style: disc inside`. Nie ma wymogu, aby podawać w tej deklaracji pełną listę cech składowych, jednak jeśli jakąś opuścimy, zostanie jej przypisana wartość domyślna. Dlatego poniższa reguła stylów nie ustawi wyróżnika wewnątrz wykazu (`inside`), ponieważ nie zostało to określone w deklaracji `list-style`, która tutaj unieważnia `list-style-position`:

Zaznacz kod Wypróbuj kod

```
li { list-style-position: inside; list-style: disc }
```

Zgodnie z zasadą kaskadowości, oczekiwany skutek (zawijanie tekstu wokół wyróżnika) odniesiemy, podając deklaracje w odwrotnej kolejności:

Zaznacz kod Wypróbuj kod

```
li { list-style: disc; list-style-position: inside }
```

albo przenosząc wartość `list-style-position` (`inside`) do zbiorczej deklaracji `list-style`:

Zaznacz kod Wypróbuj kod

```
li { list-style: disc inside }
```

PRZYKŁAD:

list-style: disc inside url(minus.gif)

- Punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy...punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy...punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy.
- Punkt drugi... punkt drugi... punkt drugi... punkt drugi... punkt drugi... punkt drugi... punkt drugi... punkt drugi... punkt drugi... punkt drugi... punkt drugi.

Odstęp wyróżnika

(nieinterpretowane)

```
selektor { marker-offset: odstęp }
```

***Selektorem** może być element z ustalonym wyświetlaniem typu `marker` [zobacz: Wstawianie stylów].*

*Natomiast "**odstęp**" oznacza odległość wyróżnika wykazu (markera) od tekstu, znajdującego się w poszczególny punktach. Wartość odstępu należy podać używając jednostek długości.*

Polecenie to pozwala nam w sposób jawny zdefiniować odstępy markerów od treści wykazu.

UWAGA! Polecenie wchodzi w skład CSS 2.0, ale nie CSS 2.1. Nie jest interpretowane!

PRZYKŁAD:

Zaznacz kod Wypróbuj kod

```
li:before {

        display: marker;

        marker-offset: 3em;

}
```

Istnieje jednak dość prosty, lecz być może nie tak oczywisty, sposób zastąpienia nieobsługiwanej własności `marker-offset`przez zespół reguł CSS interpretowanych praktycznie wszędzie:

Zaznacz kod Wypróbuj kod

```
ul, li {

		list-style-type: none;

		display: block;

		margin: 0;

		padding: 0;

}



ul li {

		background: url(marker.gif) no-repeat left top;

		padding-left: 20px;

}
```

Pierwsza z reguł stylów powyżej usuwa domyślne formatowanie wykazu oraz jego punktów. Trzeba tutaj nadmienić, że z uwagi na różnice w interpretacji nie wszystkie z wymienionych poleceń formatujących w tej deklaracji jest wymagana we wszystkich przeglądarkach, jednak tylko taki zestaw daje prawie pewność, że otrzymamy to, o co nam chodziło. Dlatego nie radzę niczego pomijać!

Druga reguła formatuje elementy `li` zawierające się wewnątrz znacznika `ul` w taki sposób, aby wyglądały jak zwyczajny wykaz. Jednak dodatkowo dostajemy możliwość wpływu na pozycję *markera*. Sztuczka polega na zastąpienia tradycyjnego wyróżnika wykazu specjalnie przygotowanych obrazkiem, który wstawiamy w tle elementów `li`. CSS daje możliwość określenia tła właściwie dla każdego elementu, a dodatkowo można ustalić, aby nie było ono powtarzane oraz określić dokładną jego pozycję. Na koniec pozostaje tylko wykonanie wolnej przestrzeni z lewej strony punktów wykazu, tak aby zmieściły się tam obrazki markerów - tutaj właśnie można określić dowolny interesujący nas odstęp pomiędzy wyróżnikami wykazu a treścią poszczególnych punktów.

Jeżeli chcemy dodatkowo ustalić margines z lewej strony wykazu, możemy to zrobić dodając deklarację `margin-left` do elementu `ul` (lub ewentualnie `li`). Analogicznie można dodać margines przed/po całym takim wykazem, oddzielający go od sąsiadującej treści, czy nawet odstępy pomiędzy poszczególnymi punktami wykazu.

Warto zauważyć, że ten sposób pozwala również dokładnie określić pionową pozycję markera, co przy wykorzystaniu tradycyjnych metod nie jest możliwe, a właśnie wtedy sprawia to największe problemy, gdyż różne przeglądarki ustawiają obrazkowe wyróżniki wykazów w odmienny sposób - kilka pikseli wyżej lub niżej niż w innych.

PRZYKŁAD:

- Punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy.
- Punkt drugi
- Punkt trzeci

- Punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy.
- Punkt drugi
- Punkt trzeci

- Punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy... punkt pierwszy.
- Punkt drugi
- Punkt trzeci

Automatyczna numeracja

(interpretuje Internet Explorer 8.0, Firefox, Opera, Chrome, Konqueror)

Jeżeli zaznajomiłeś się już czytelniku bardziej szczegółowo z zagadnieniem wykazów w języku HTML, to zapewne wiesz już, że nie oferują one żadnych bardziej skomplikowanych sposobów numeracji poszczególnych punktów i podpunktów. To znaczy możliwe jest określenie np. następującego sposobu numeracji:

- Punkty główne - liczby rzymskie: I, II, III,...
- Podpunkty - liczby arabskie: 1, 2, 3,...
- Podpunkty drugiego rzędu - litery: a, b, c,...

Co jednak zrobić, jeżeli nie odpowiada nam taki sposób numeracji punktów albo nie możemy z góry określić ile będziemy mieli poziomów zagnieżdżenia? CSS daje na to prostą receptę - *automatyczną numerację*, czyli możliwość dowolnego określania sposobu numerowania elementów, np.:

- Punkty główne: 1., 2., 3.,...
- Podpunkty: 1.1., 1.2., 1.3.,... 2.1., 2.2., 2.3.,... 3.1., 3.2., 3.3.,...
- Podpunkty drugiego rzędu: 1.1.1., 1.1.2., 1.1.3.,... 1.2.1., 1.2.2., 1.2.3.,... 2.1.1., 2.1.2., 2.1.3.,... 2.2.1., 2.2.2., 2.2.3.,...
- i tak dalej...

Automatyczna numeracja w CSS korzysta z tzw. *liczników*, które są podobne do *zmiennych* w językach programowania. Każdy licznik posiada swój unikalny *identyfikator* (nazwę), za pomocą którego można się do niego odnosić w arkuszu CSS. Dodatkowo każdy licznik przechowuje wartość, która jest liczbą określającą wartość licznika dla aktualnego punktu wykazu (1, 2, 3,...).

Zasada tworzenia automatycznej numeracji jest prosta:

1. Najpierw określany nazwę licznika dla danego wykazu, od razu zerując go:

Zaznacz kod | Wypróbuj kod

```
ol { counter-reset: nazwa_licznika }
```

2. Usuwamy domyślne numerowanie wykazu, aby nie kolidowało z automatycznym:

Zaznacz kod | Wypróbuj kod

```
ol li { list-style-type: none }
```

3. Następnie dla kolejnych punktów i podpunktów wykazu wyświetlamy aktualną wartość licznika na początku treści, a następnie zwiększamy tą wartość o jeden - dla następnego punktu lub podpunktu:

Zaznacz kod | Wypróbuj kod

```
ol li:before { content: counters(nazwa_licznika, ".") ". "; counter-increment: nazwa_licznika }
```

Oto efekt:

1. Punkt pierwszy (1.)
    1. Podpunkt pierwszy (1.1.)
        1. Podpunkt drugiego rzędu (1.1.1.)
        2. Podpunkt drugiego rzędu (1.1.2.)
    2. Podpunkt drugi (1.2.)
        1. Podpunkt drugiego rzędu (1.2.1.)
        2. Podpunkt drugiego rzędu (1.2.2.)
2. Punkt drugi (2.)
    1. Podpunkt pierwszy (2.1.)
        1. Podpunkt drugiego rzędu (2.1.1.)
        2. Podpunkt drugiego rzędu (2.1.2.)
    2. Podpunkt drugi (2.2.)
        1. Podpunkt drugiego rzędu (2.2.1.)
        2. Podpunkt drugiego rzędu (2.2.2.)

UWAGA! Nie obsługuje MSIE 7.0! W MSIE 8.0 wszystko jest w porządku, ale tylko w trybie Standards Compliance.
W Operze 10 mogą wystąpić błędy, jeśli ta sama nazwa licznika zostanie użyta do numerowania więcej niż jednego wykazu na stronie.

Ponieważ przedstawiony powyżej wykaz jest wielokrotnie zagnieżdżony, więc znajduje się w nim kilka znaczników `ol` umieszczonych jeden wewnątrz drugiego. Dlatego właśnie polecenie `ol { counter-reset: nazwa_licznika }` tak naprawdę przy każdym znaczniku `ol` tworzy nowy *egzemplarz* tego licznika. Egzemplarze można oznaczać dla potrzeb analizy kolejnymi numerami (przyjmijmy numerację od zera): `nazwa_licznika[0]`, `nazwa_licznika[1]`, `nazwa_licznika[2]` itd. Każdy z egzemplarzy przechowuje osobną numerację (działają niezależnie od siebie), ale jednocześnie dla każdego z egzemplarzy tego samego licznika (czyli egzemplarzy z tą samą nazwą) można uzyskać aktualny stan wszystkich liczników nadrzędnych innych egzemplarzy. Realizuje się to za pomocą *funkcji* `counters()`, która przyjmuje dwa argumenty rozdzielone przecinkiem:

1. identyfikator licznika (wspólna nazwa egzemplarzy),
2. łańcuch znakowy separatora, który ma rozdzielać kolejne wartości egzemplarzy licznika.

Wartość odczytaną przez tę funkcję możemy następnie podstawić w deklaracji stylu do własności `content`, co spowoduje wyświetlenie jej w treści elementu:

Zaznacz kod | Wypróbuj kod

```
ol li:before { content: counters(nazwa_licznika, ".") }
```

W świetle tego przebieg generowania automatycznej numeracji dla przedstawionego wcześniej wykazu odbywa się następująco:

Zaznacz kod | Wypróbuj kod

```
<ol>            <!-- nazwa_licznika[0] = 0 -->

<li>1 (nazwa_licznika[0] + 1 = 1)

                <ol>            <!-- nazwa_licznika[1] = 0 -->

                <li>1.1 (nazwa_licznika[1] + 1 = 1)

                                <ol>            <!-- nazwa_licznika[2] = 0 -->

                                <li>1.1.1 (nazwa_licznika[2] + 1 = 1)</li>

                                <li>1.1.2 (nazwa_licznika[2] + 1 = 2)</li>

                                </ol>

                </li>

                <li>1.2 (nazwa_licznika[1] + 1 = 2)

                                <ol>            <!-- nazwa_licznika[3] = 0 -->

                                <li>1.2.1 (nazwa_licznika[3] + 1 = 1)</li>

                                <li>1.2.2 (nazwa_licznika[3] + 1 = 2)</li>

                                </ol>

                </li>

                </ol>

</li>

<li>2 (nazwa_licznika[0] + 1 = 2)

                <ol>            <!-- nazwa_licznika[4] = 0 -->

                <li>2.1 (nazwa_licznika[4] + 1 = 1)

                                <ol>            <!-- nazwa_licznika[5] = 0 -->

                                <li>2.1.1 (nazwa_licznika[5] + 1 = 1)</li>

                                <li>2.1.2 (nazwa_licznika[5] + 1 = 2)</li>

                                </ol>

                </li>

                <li>2.2 (nazwa_licznika[4] + 1 = 2)

                                <ol>            <!-- nazwa_licznika[6] = 0 -->
```

```
                        <li>2.2.1 (nazwa_licznika[6] + 1 = 1)</li>

                        <li>2.2.2 (nazwa_licznika[6] + 1 = 2)</li>

                        </ol>

                </li>

                </ol>

</li>

</ol>
```

Oprócz funkcji `counters()` istnieje również funkcja `counter()`, która działa bardzo podobnie, jednak przyjmuje jako argument jedynie nazwę identyfikatora licznika, a odczytuje tylko wartość aktualnego egzemplarza licznika, czyli pojedynczą liczbę. Użycie w powyższym przykładzie funkcji `counter()` zamiast `counters()`, spowodowałoby wyświetlenie wykazu numerowanego tradycyjnie:

1. Punkt pierwszy (1.)
    1. Podpunkt pierwszy (1.)
        1. Podpunkt drugiego rzędu (1.)
        2. Podpunkt drugiego rzędu (2.)
    2. Podpunkt drugi (2.)
        1. Podpunkt drugiego rzędu (1.)
        2. Podpunkt drugiego rzędu (2.)
2. Punkt drugi (2.)
    1. Podpunkt pierwszy (1.)
        1. Podpunkt drugiego rzędu (1.)
        2. Podpunkt drugiego rzędu (2.)
    2. Podpunkt drugi (2.)
        1. Podpunkt drugiego rzędu (1.)
        2. Podpunkt drugiego rzędu (2.)

Jeżeli w jednej deklaracji chcemy ustawić lub zwiększyć więcej niż jeden licznik, nie należy wstawiać kilka razy tej samej własności `counter-reset` czy `counter-increment`, lecz umieścić kilka identyfikatorów liczników w tej samej wartości, rozdzielonych spacjami:

Zaznacz kod | Wypróbuj kod

```
ol { counter-reset: licznik1 licznik2 }

ol li { counter-increment: licznik1 licznik2 }
```

Ponadto warto odnotować, że `counter-reset` wcale nie musi ustawiać licznika na zero, a `counter-increment` wcale nie musi zwiększać licznika tylko o jeden. Licznik może być nawet zmniejszany, poprzez podanie wartości ujemnej (**numeracja w tył**):

Zaznacz kod | Wypróbuj kod

```
ol { counter-reset: licznik 3 }

ol li { list-style-type: none }

ol li:before { content: counters(licznik, ".") ". "; counter-increment: licznik -1 }
```

1. Punkt drugi (2.)
    1. Podpunkt drugi (2.2.)
        1. Podpunkt drugiego rzędu (2.2.2.)
        2. Podpunkt drugiego rzędu (2.2.1.)
    2. Podpunkt pierwszy (2.1.)
        1. Podpunkt drugiego rzędu (2.1.2.)
        2. Podpunkt drugiego rzędu (2.1.1.)
2. Punkt pierwszy (1.)
    1. Podpunkt drugi (1.2.)
        1. Podpunkt drugiego rzędu (1.2.2.)
        2. Podpunkt drugiego rzędu (1.2.1.)
    2. Podpunkt pierwszy (1.1.)
        1. Podpunkt drugiego rzędu (1.1.2.)
        2. Podpunkt drugiego rzędu (1.1.1.)

Zatem ogólna postać wartości `counter-reset` i `counter-increment` jest następująca: lista identyfikatorów liczników rozdzielonych spacjami, przy których po każdym z nich może opcjonalnie wystąpić liczba oddzielona od swojego (poprzedzającego) identyfikatora licznika również spacją:

Zaznacz kod | Wypróbuj kod

```
ol { counter-reset: licznik1 licznik2 3 }

ol li { counter-increment: licznik1 licznik2 -1 }
```

Automatyczną numerację można wykorzystać nie tylko do wykazów, ale również np. do tytułów:

Zaznacz kod | Wypróbuj kod

```
body { counter-reset: h1 }

h1 { counter-reset: h2 }

h1:before { content: counter(h1) ". "; counter-increment: h1 }

h2 { counter-reset: h3 }

h2:before { content: counter(h1) "." counter(h2) ". "; counter-increment: h2 }

h3 { counter-reset: h4 }

h3:before { content: counter(h1) "." counter(h2) "." counter(h3) ". "; counter-increment: h3 }

h4 { counter-reset: h5 }

h4:before { content: counter(h1) "." counter(h2) "." counter(h3) "." counter(h4) ". "; counter-increment: h4 }

h5 { counter-reset: h6 }

h5:before { content: counter(h1) "." counter(h2) "." counter(h3) "." counter(h4) "." counter(h5) ". "; counter-increment: h5 }

h6:before { content: counter(h1) "." counter(h2) "." counter(h3) "." counter(h4) "." counter(h5) "." counter(h6) ". "; counter-increment: h6 }
```

Powyższe reguły stylów będą funkcjonowały prawidłowo tylko w przypadku poprawnego *semantycznie* kodu, tzn. każdy element `h2` powinien być poprzedzony przez `h1`, `h3` - przez `h2`, `h4` - przez `h3`, `h5` - przez `h4`, `h6` - przez `h5`.

Style licznika

Domyślnie liczniki automatycznej numeracji mają postać liczb arabskich. Możliwe jest jednak przypisanie dowolnego innego stylustosowanego dla wykazów:

Zaznacz kod | Wypróbuj kod

```
ol { counter-reset: upper_alpha }

ol li { list-style-type: none }

ol li:before { content: counters(upper_alpha, ".", upper-alpha) ") "; counter-increment: upper_alpha }
```

1. Punkt pierwszy (A)
    1. Podpunkt pierwszy (A.A)
        1. Podpunkt drugiego rzędu (A.A.A)
        2. Podpunkt drugiego rzędu (A.A.B)
    2. Podpunkt drugi (A.B)
        1. Podpunkt drugiego rzędu (A.B.A)
        2. Podpunkt drugiego rzędu (A.B.B)
2. Punkt drugi (B)
    1. Podpunkt pierwszy (B.A)
        1. Podpunkt drugiego rzędu (B.A.A)
        2. Podpunkt drugiego rzędu (B.A.B)
    2. Podpunkt drugi (B.B)

1. Podpunkt drugiego rzędu (B.B.A)
2. Podpunkt drugiego rzędu (B.B.B)

Numeracja typu: I.i.1.A.a.α., I.i.1.a.β.,... I.i.1.A.b.α., I.i.1.A.b.β.,... I.i.1.B.b.α., I.i.1.B.b.β.,... I.i.2.B.b.α., I.i.2.B.b.β.,... I.ii.2.B.b.α., I.ii.2.B.b.β.,... II.ii.2.B.b.α., II.ii.2.B.b.β.,...

Zaznacz kod | Wypróbuj kod

```
body { counter-reset: h1 }

h1 { counter-reset: h2 }

h1:before { content: counter(h1, upper-roman) ". "; counter-increment: h1 }

h2 { counter-reset: h3 }

h2:before { content: counter(h1, upper-roman) "." counter(h2, lower-roman) ". "; counter-increment: h2 }

h3 { counter-reset: h4 }

h3:before { content: counter(h1, upper-roman) "." counter(h2, lower-roman) "." counter(h3, decimal) ". "; counter-increment: h3 }

h4 { counter-reset: h5 }

h4:before { content: counter(h1, upper-roman) "." counter(h2, lower-roman) "." counter(h3, decimal) "." counter(h4, upper-alpha) ". "; counter-
increment: h4 }

h5 { counter-reset: h6 }

h5:before { content: counter(h1, upper-roman) "." counter(h2, lower-roman) "." counter(h3, decimal) "." counter(h4, upper-alpha) "." counter(h5, lower-
alpha) ". "; counter-increment: h5 }

h6:before { content: counter(h1, upper-roman) "." counter(h2, lower-roman) "." counter(h3, decimal) "." counter(h4, upper-alpha) "." counter(h5, lower-
alpha) "." counter(h6, lower-greek) ". "; counter-increment: h6 }
```

Firefox, Chrome i Konqueror obsługują również style typu: `disc` (koło), `circle` (okrąg) i `square` (kwadrat).