Contents

Rozmazany tekst

Hej, dziś krótko. Poniżej macie prosty tip na to jak przy pomocy CSS zrobić rozmazany tekst.

```
.class {
   color: transparent;
   text-shadow: 0 0 5px rgba(0,0,0,0.5);
}
```

Skalę rozmazania możecie zmienić manipulując wartością opactity kodu rgba, a także wartością rozmycia właściwości text-shadow. Proste. Dodatkowo mam dla was bardzo ciekawy tester prezentacji fontów dla urządzeń mobilnych. Tutaj w wygodny sposób możecie sprawdzić jak wybrany przez was font prezentuje się na ekranach telefonów i tabletów. Opcjonalnie można zmienić rozmiar testowanego kroju a także jego rozmieszczenie względem ekranu.

Clearfix

Prawdopodobnie najpolularniejszy trik jaki stosowano przy tworzeniu stron internetowych.

```
.clearfix:after {
     visibility: hidden;
     display: block;
     font-size: 0;
     content: " ";
     clear: both;
     height: 0;
     }
```

```
.clearfix { display: inline-block; }
/* start commented backslash hack \*/
* html .clearfix { height: 1%; }
.clearfix { display: block; }
/* close commented backslash hack */
```

Nicolas Gallagher opisał dokładnie (“micro” clearfix)[http://nicolasgallagher.com/micro-clearfix-hack/], który jest nowszym podejściem do tematu.

```
.group:before,
.group:after {
    content: "";
    display: table;
}
.group:after {
    clear: both;
}
.group {
    zoom: 1; /* For IE 6/7 (trigger hasLayout) */
}
```

Aktaulna wersja, jeśli potrzebujesz wsparcia w IE8 lub nowszych przeglądarkach.

```
.group:after {
  content: "";
  display: table;
  clear: both;
}
```

Krótsze kody kolorów

Używanie skróconych kodów kolorów nie jest niczym nowym, ale wiele osób zwyczajnie, o tym zapomina. Niektórzy użytkownicy pamiętają jeszcze 216 bezpiecznych kolorów w przeglądarkach, choć dostępne spektrum dla 3 składowych to 256^3, czyli 16777216 kolorów.

Tutaj pojawia się pewien dylemat. Ile kolorów odróżnia ludzkie oko? To zależy indywidualnie od każdego z nas, ale w przypadku 16^3, otrzymujemy dokładne 4096 różnych kolorów. Czy to wystarczy? Oceńcie sami.

```
.white {
    color: #ffffff; /* use #fff */
}
```

```
.black {
    color: #000000; /* use #000 */
}
.test {
    color: #abc; /* use #aabbcc */
}
```

Powyższe przykłady są bardzo prozaiczne i zasadniczo możliwe jest zapisanie każdego z kolorów w formie 3 cyfrowego kodu. Kwestia jak duża będzie odległość pomiędzy odcieniami każdego koloru.

```
div {
    background: #c54218; /* use #c41 */
}
```

Potrzebowaliśmy koloru `#c54218`, a otrzymaliśmy `#cc4411`, czy całkiem bliski naszym oczekiwaniom.

'Uwaga:' Nie wszystkie kolory tak równie łatwo interpoluje się do wartości skróconych.

Dla wyjaśnienia ciemniejszy i najbliższy białemu kolor to `#eee`, który jest dość ciemny.

Połącz razem wiele skrótów i wartości definicji

Nieustanna optymalizacja naszych stylów, prowadzi czasami do trudnych w odnalezieniu błędów. Nadpisywanie właściwości skrótowych jest dobrym rozwiązanie, bo często zaoszczędzamy kilka znaków w naszych stylach.

Bardzo rzadko zdarza się, aby style wymagały czterech różnych kolorów obramowania, ale jeśli tak jest pozostaje tylko właściwe kodowanie.

```
div {
    border-top: 1px solid #f00;
    border-right: 1px solid #0f0;
    border-bottom: 1px solid #00f;
    border-left: 1px solid #aaa;
}
```

W tym wypadku mamy również szerokie możliwości optymalizacji. Przykład o tyle niefortunny, bo domyślne wartości obramowania to szerokości 1 piksela i linia ciągła.

```
div {
    border-color: #f00 #0f0 #00f #aaa;
}
```

Zatem wystarczyłoby samo zdefiniowanie samych kolorów. O ile mniej kodu. Szybko i czytelnie.

Jeśli nadpisujemy pojedyncze właściwości lub style, przeważnie lepsze jest zdefiniowanie tego samego raz i odpowiednie nadpisanie reguły.

Krótsze definicje właściwości

Arkusze stylów kaskadowych zostały pomyślane w ten sposób, aby programiści tworzyli kod jak najszybciej. Poszczególne właściwości zawierają szereg skrótów, które w jednej linii definiują wiele właściwości.

```
div {
    border: 1px solid #aaa;
}
```

Prosta definicja `border` umożliwia ustawienie zarówno wielkości obramowania, jego stylu jak i koloru. Tworzenie czterech reguł jest nieoptymalne z wielu powodów.

```
.global {
    border-top: 1px solid #aaa;
    border-right: 1px solid #aaa;
    border-bottom: 1px solid #aaa;
    border-left: 1px solid #aaa;
}
```

Po pierwsze waga pliku CSS jest większa. Dodatkowo analiza takich stylów bywa kłopotliwa, bo zmiana w jednym miejscu, bywa nadpisywana kolejną regułą w dalszej części.

Skrótowe wartości właściwości

Używanie skróconych definicji niesie wiele korzyści. Jeśli w jednym momencie definiujemy zbiór wspólnych wartości zmniejszamy wielkość stylów. Dodatkowo dajemy sobie łatwą możliwość zmiany wyglądu w jednej linii, bez dopisywania kolejnych reguł.

```css
p {
    margin: 10px;
}
```

Nasza strona jest bardzo prosta i prezentuje paragrafy z marginesem wielkości `10px`. Jedna linia ustawia od razu wszystkie wartości

Jednak nie zawsze spotykamy równie banalne przykłady. Choć CSS pozwala na jednoczesne ustawienie wartości skrajnych.

```css
p {
    margin: 0 10px; /* padding: 0 10px 0 10px; */
}
```

Taki zapis odnosi się do zerowego marginesu w pionie i `10px` po bokach.

```css
p {
    margin: 0 10px 25px; /* margin: 0 10px 25px 10px */
}
```

Powyższa reguła ustawia dolny margines paragrafu na `25px`, co jest krótsze niż zdefiniowanie lewego marginesu przez kolejną liczbę.

Istnieją przypadki, kiedy konieczne jest podanie wszystkich czterech wartości. Jednak nawet wtedy bedzie to znacznie krótszy zapis niż 4 osobne reguły.

```css
p {
    margin: 10px 10px 25px 50px;
}
```

Analogicznie poza marginesem, ustawimy dopełnienie elementu, obramowanie.

VERIFY

Liczenie elementów DOM

Optymalizacja kodu lub poszukiwanie błędów wymaga od programistów sprawnego poruszania się w modelu obiektowym. Odnalezienie elementu od danym identyfikatorze czy klasie nie jest wcale trudne, ale tak prozaiczne czynności często nie wystarczają do rozwiązania problemu.

Ile elementów zawiera nasza strona?

```
var all = document.getElementsByTagName("*");
var ids = 0, classes = 0;
for (var i = all.length; i--;) { if (all[i].id !== '') ids++; if (all[i].className !== '') c
'all: ' + all.length + ', ids: ' + ids + ', classes: ' + classes;
```