# Chapter 1

## CSS

Oto początek mojego rozdziału książki i wogóle tego co chce napisać.

## Rozmazany tekst

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

```css .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.

```css .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.

```css .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.

```css .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.

```css .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.

```css 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.

``css 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.

``css 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.

``css 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.

``css .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

## Warunkowe logowanie

Dobrze wiemy, że JavaScript to bardzo elastyczny język. Nieraz potrzebujemy warunkowego wykonania instrukcji. Dla łatwiejszego zrozumienia przykładu tworzymy prosty obiekt

```javascript var user = { name: 'Luke', showName: function() { return 'Name: ' + this.name; } }; ```

Obiekt użytkownika zawiera nazwę i funkcję. Nasz cel to wyświetlenie `user.name`, jeśli istnieje obiekt `user`.

```javascript console.log(user && user.name); ```

W ten sposób wyświetlisz w konsoli właśność `name`, jeśli istnieje obiekt `user`. Możliwości tego triku nie kończą się na przekazaniu właśności obiektu. A może funkcja?

```javascript console.log(user && user.showName());

Zastowowanie tej sztuczki w naszych projektach nie sprawia wiele trudności.

## 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?

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

## Wzorzec modułu

Brak klas w JavaScript stanowi spory problem dla początkujących programistów. Często

Tworzymy

```javascript var aya = aya || {};

aya.framework = function() {

// private property var version = '0.0.2';

// private method var getVersion = function() { return version; }

// all returned is a public return {

// initialization init: function() { console.log('init successful...'); },

printModuleVersion: function() { console.log('Version: ' + getVersion()); } } }(); ```

Przygotowany w ten sposób obiekt przechowuje wewnątrz pewne zmienne jako prywatne. Natomiast kod zwracany jest traktowany jako publiczny. Ten prosty sposób pozwala na ukrycie części implementacji i dostęp spoza samego obiektu.

```javascript aya.framework.init();

Jak się spodziewamy w konsoli wyświetli się informacja o inicjalizacji obiektu.

```javascript aya.framework.printModuleVersion(); ```

Podobnie będzie, kiedy sprawdzimy wersję naszego obiektu, ale możliwe będzie to jedynie poprzez publiczną metodę.

```javascript aya.framework.printModuleVersion(); ```

Próba uzyzkania wersji przy użyciu metody prywatnej poza obiektem, kończy się błędem, czyli dokładnie jak powinno.

```console > TypeError: Object #<Object> has no method 'getVersion' ```
```

## Domyślne wartości zmiennych

Nie zawsze wiesz czy szukana zmienna istnieje lub została zainicjalizowana. Ponowne przypisanie wartości zatrze informacje o dotychczasowej zmiennej. Istnieje łatwe sprzwdzenie czy istnieje dana zmienna lub przypisanie jej wartości domyślnej.

``javascript var app = app || {}; ``

Przykładowe użycie...

## Tak lub nie

???

``javascript if (Math.round(Math.random())) { // do if true } else { // do if false } `` Bardzo prosty sposób na symulowanie działania naszego skryptu.