Cascading Style Sheets

Dlaczego stosować style?

Specyfikacja

The World Wide Web Consortium (www.w3c.org)

Style wewnętrzne lokalne

W sekcji BODY:
<p **style**=”font-size: *xx-large*; text-align: *center*;”>*To jest jakiś tekst*</p>

Style wewnętrzne centralne

W sekcji HEAD:
<style>
p
{font-size: *xx-large*;
text-align: *center*;}
</style>

Style zewnętrzne

Ładowanie stylów (sekcja HEAD):
<**link** rel="*stylesheet*" media="*screen*" href="*style.css*" />

Zawartość zewnętrznego pliku ***style.css***:
p
{font-size: *xx-large*;
text-align: *center*;}

Pojęcie klasy stylu

Definicje stylu

p

{font-size: *xx-large*;

text-align: *center*;

color: *blue*;}

p.przypisy

{font-size: *small*;

text-align: *justify*;}

p.autor

{color: *black*;}

/ Poniżej: element <a> będzie zawsze zielony, gdy wystąpi wewnątrz elementu */*

p.autor > a

{color: green;}

Zastosowanie stylu

W sekcji BODY:

Tutaj znajduje się jakiś akapit tekstu

Tutaj znajduje się inny akapit tekstu

<p **class**=*”autor”*>*Jan Kowalski*

<p **class**=*”przypisy”*>*To jest jakiś przypis*

Przykład CSS

h2

{text-align: *left*;

background-color: *#b0c4de*;

font-weight: *bold*;

font-size: *small*;

color: *#ffffff*;

margin-top: *55px*;

text-indent: *1.0cm*;

padding-top: *1px*;

padding-bottom: *2px*;}

img.ramka

{border-style: *solid*;

border-width: *1px*;

margin: *3px*;}

Czcionki

W sekcji CSS (na początku pliku) oraz w definicji <body>:

*// Prawdopodobnie najlepszym wyborem dla języka polskiego będzie **Roboto**:*
@import url(https://fonts.googleapis.com/css?family=Roboto:400,100,100italic,300,300italic,400italic,500,500italic,700,700italic,900,900italic&subset=latin,latin-ext);
body {font-family: 'Roboto', sans-serif;}

*// Druga w kolejce, całkiem zgrabna **Source Sans Pro**:*
@import url(https://fonts.googleapis.com/css?family=Source+Sans+Pro:400,200,200italic,300,300italic,400italic,600,600italic,700,700italic,900,900italic&subset=latin,latin-ext);
body {font-family: 'Source Sans Pro', sans-serif;}

*// Całkiem niezła **Libre Franklin**:*
@import url(https://fonts.googleapis.com/css?family=Libre+Franklin:400,100,100italic,200,200italic,300,300italic,400italic,500,500italic,600,600italic,700,700italic,800,800italic,900,900italic&subset=latin,latin-ext);
body {font-family: 'Libre Franklin', sans-serif;}

*// Domyślna **Open Sans**:*
@import url(https://fonts.googleapis.com/css?family=Open+Sans:400,400italic,300,300italic,600,600italic,700,700italic,800,800italic&subset=latin,latin-ext);
body {font-family: 'Open Sans', sans-serif;}

*// Wyrazista, stylizowana odrobinę na gotycki / łaciński styl **Proza Libre**:*
@import url(https://fonts.googleapis.com/css?family=Proza+Libre:400,400italic,500,500italic,600,600italic,700,700italic,800,800italic&subset=latin,latin-ext);
body {font-family: 'Proza Libre', sans-serif;}

*// Elegancka, wytworna, delikatna **Josefin Sans**:*
@import url(https://fonts.googleapis.com/css?family=Josefin+Sans:400,100,100italic,300,300italic,400italic,600,600italic,700,700italic&subset=latin,latin-ext);
body {font-family: 'Josefin Sans', sans-serif;}

Style responsywne

...czyli rozwiązanie, które umożliwia dostosowanie się strony do ekranu różnych urządzeń (zastosowane są tzw. „*Media queries*” oraz „*Flexible Box Layout*”). Zastąpiło przestarzały standard WAP oraz dedykowane wersje stron „*m.domena.pl*”.

W sekcji <head>

/ Stary sposób */*
~~<link href="mobile.css" rel="stylesheet" type="text/css" media="handheld" />~~
/ Nowy sposób */*
<meta name="viewport" content="width=device-width, initial-scale=1.0">
/ Chcemy, aby zawartość strony była przeskalowana do szerokości urządzenia w skali 1:1 bez względu czy strona jest w widoku portrait czy landscape. Opcja ta powiększa także czcionki, aby tekst był wygodny w czytaniu na smartfonie.* Jeśli nie chcemy pozwolić na skalowanie strony, dodajemy jeszcze atrybut „*user-scalable=no*”, choć nie jest to opcja zalecana przez Google. */*

W pliku CCS

@media screen and (min-width: 800px) and (max-width: 1280px) {
/ Tu znajdą się opcje, które będą ładowane tylko na ekranach o wymiarach między 800px a 1280px szerokości, przy czym szerokość oznacza tu zawsze wymiar poziomy, tj. jeśli smartfon ma szerokość rzędu 360px w trybie portrait, to w trybie landscape jego szerokość będzie wynosiła 640px. Należy raczej unikać atrybutów orientacji ekranu, gdyż aktywacja klawiatury na niektórych*

```
smartfonach w położeniu portrait powoduje, że przeglądarka wykrywa je jako landscape (bo szerokość jest wtedy większa niż wysokość. */
.duzyEkran{ color: #cccccc; }
.mobile-text{ display: none; }
.non-mobile-text{ display: block; }
}
```

Uwaga 1: Aby określić tzw. „punkty graniczne" (czyli wartość *min-width* i *max-width*), należy najpierw zaprojektować treść na najmniejszy ekran, a potem stopniowo go powiększać i subiektywnie zadecydować, w którym momencie powinna pojawić się druga kolumna, powiększona czcionka, itp. Specjaliści zalecają, aby nie brać pod uwagę rozdzielczości poszczególnych smartfonów i tabletów, lecz jedynie wygląd strony na poszczególnych rozdzielczościach: jeśli w wierszu jest więcej niż 10 wyrazów (70-80 liter wraz ze spacjami), to jest to sygnał, że należy utworzyć nowy breakpoint (choć i tak decydująca jest subiektywna ocena estetyczna, bo na nią ma wpływ także język strony [np. niemieckie słowa są dłuższe], odległość urządzenia od oczu i parę innych czynników). Postępować należy według powiedzenia: „*Start with the small screen first, then expand until it looks like shit. Time to insert a breakpoint!*".

Uwaga 2: Parametr „*width*" podajemy zawsze w procentach i dopiero dla największego ekranu podajemy go w pikselach, jako „*max-width*" (aby nie poszerzał się w nieskończoność).

Uwaga 3: Należy zauważyć, iż właściwość „*device-width*" jest wycofywana z *Media Queries 4* (obecnie jeszcze jako *Draft*) i deweloperzy zalecają rezygnację z niej.

Uwaga 4: Realna rozdzielczość ekranu smartfona / tableta nie jest tym samym, co wymiar fizyczny ekranu podawany w CSS (to zależy od tzw. „*pixel ratio*" [gęstości pikseli na ekranie], przez które dzielona jest rozdzielczość urządzenia). I tak dla przykładu:

- *Samsung Galaxy S5* = 1080x1920px (w CSS: 360x640px);
- *Samsung Galaxy Note 2* = 720x1280px (w CSS: 360x640px);
- *Samsung Galaxy Note 4* = 1440x2560px (w CSS: 360x640px);
- *Samsung Galaxy Tab 3* = 800x1280px (w CSS: 800x1280px);
- *LG G4* = 1440x2560px (w CSS: 360x640px);
- *Sony Xperia Z3* = 1080x1920px (w CSS: 360x598px);

Przykład ustawienia breakpointów:

- 320 – 768px;
- 769 – 1023px;
- 1024 – w zwyż.

Stosowanie

1. Szerokość div-ów (i innych elementów blokowych) należy określać w procentach, a nie w pikselach.
2. Warto pamiętać o opcji „*display: flex;*" (wewnątrz tego elementu nie należy stosować „*vertical-align*" ani „*float*").

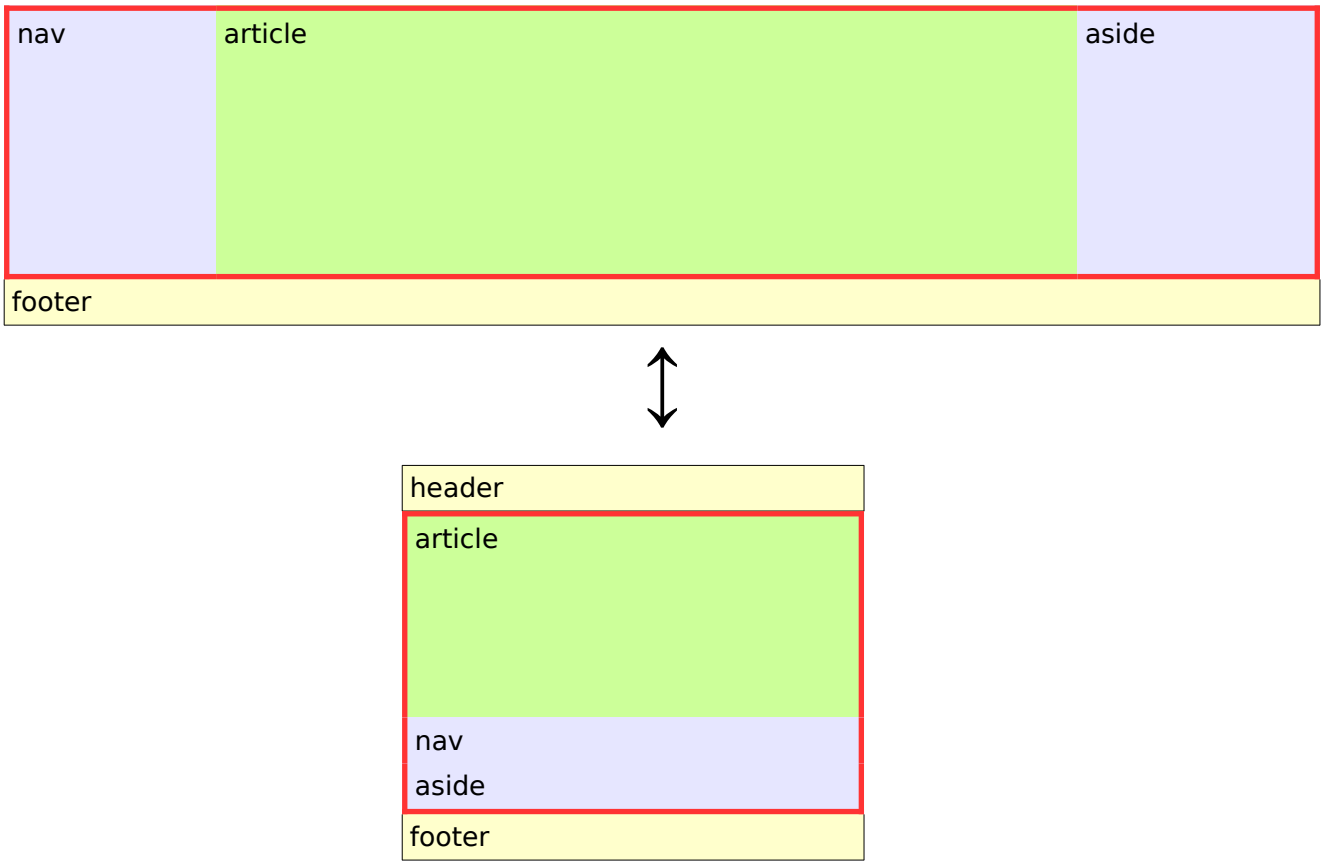
W CSS:

```
@media all and (max-width: 640px) {
 #main, #page {
 flex-direction: column;}
}
```

Flexible Box Layout Module (level 1)

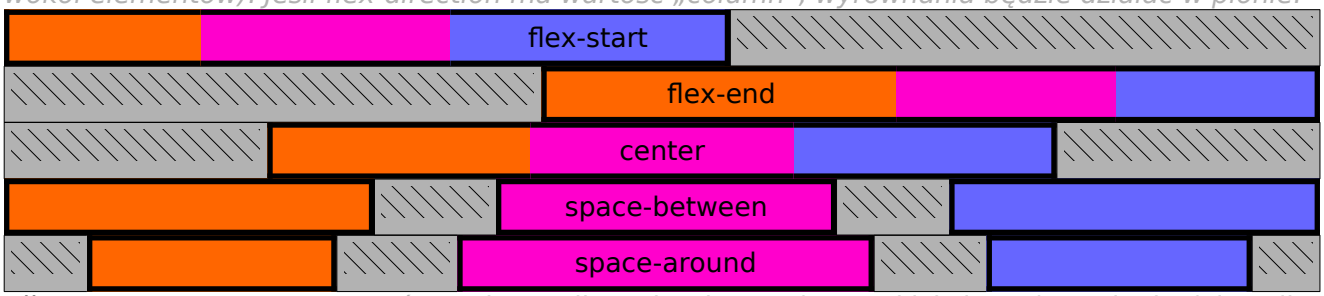
Specyfikacja: *https://www.w3.org/TR/css-flexbox-1/*

header



Flex container

Uwaga 1: Wewnątrz konteneru nie działają atrybuty „*column-**” oraz pseudoelementy „*::first-line*”, „*::first-letter*”.
Uwaga 2: Definicje mają zastosowanie tylko wobec bezpośrednich „dzieci” elementu *div.main*, już nie wobec „wnuków”. Oczywiście można zagnieżdżać „*flex container-y*”, tzn. „dziecko” takiego elementu samo może być „*flex containerem*”. Dzieci „*flex container*” zwane są „*flex items*”. Dla przykładu, dla menu bazującym na elementach *<li>*, kontenerem (*flex-container*) będzie element *<ul>* (a nie *<nav>*!), zaś *flex-items* będą elementy *<li>*.
Aby dokonać takiej transformacji, należy w CSS:
div.main { *// Tzw. **flex container**.*
display: flex;
flex-direction: row; // Wewnętrzne bloki/elementy są ułożone obok siebie w wierszu, w poziomie. Możliwa opcja „column”.
flex-wrap: nowrap; // Wartość „wrap” pozwala elementom na zawijanie się w kolejnych wierszach. Jeśli zawijanie ma dotyczyć kolumn, musi być określona konkretna wysokość kontenera.
justify-content: center; // Wyrównanie dla wszystkich items wewnątrz kontenera. Inne wartości: flex-start (wyrównanie do lewej), flex-end (wyrównanie do prawej), space-between (równomierne rozłożenie elementów w wierszu z odstępami między nimi), space-around (równomierne odstępy wokół elementów). Jeśli flex-direction ma wartość „column”, wyrównania będzie działać w pionie.



align-content: center; // Wyrównanie zawijanych wierszy (wszystkich items) w pionie (ale tylko jeśli kontener jest w trybie „column”). Inne wartości: stretch, flex-start, flex-end, space-between, space-around.

align-items: center;} *// Inne wartości: stretch, flex-start, flex-end, baseline. Wartość „stretch” rozciąga elementy w pionie na całą wysokość. Działa tylko jeśli kontener jest w trybie „column”.*

Uwaga: Jeśli flex-direction ma wartość „row”, wtedy wyrównanie elementów do dołu nie będzie działać. Można to rozwiązać na dwa sposoby:
a) ustawić odpowiedni margines elementu flex-container od góry;
b) uczynić z nadrzędnego kontenera również flex-container, którego wartość flex-direction będzie wynosiła „column”, a wartość jego „justify-content” będzie równa „flex-end”.

```
@media all and (max-width: 640px) {
div.main {
flex-direction: column;} // Elementy ustawiane są w kolumnę

article, nav, aside {
order: 0;}
}
```

Flex items

Uwaga: Wobec tego typu elementów nie działają opcje: *float*, *vertical-align*. Kontener flex-item może zostać rozciągnięty jeśli znajduje się w nim zbyt duży obrazek.

```
nav { // Tzw. flex items.
flex-grow: 1; // Jak bardzo element może rozciągnąć się względem innych elementów, gdy zostanie wolna niewypełniona przestrzeń (liczba jest wagą; 0 = brak rozciągnięcia). Np. jeśli wolna przestrzeń wynosi 200px, a wagi trzech elementów to 1:1:2 – to pierwszy i drugi rozciągną się po 50px, a trzeci o 100px.
flex-shrink: 6; // Jak bardzo element może się skurczyć względem innych elementów, gdy zabraknie miejsca. Liczba jest wagą, wartość 1 w każdym elemencie oznacza równomierne kurczenie się wszystkich elementów.
flex-basis: 20%; // Ile procent wiersza ma zajmować jedna komórka/element. Możliwa jest także wartość: auto (dopasowuje się do zawartości elementu).
align-self: flex-start; // Wyrównanie w pionie pojedynczego item względem pozostałych items (gdy kierunek jest „row”). Inne wartości: stretch, center, flex-start, flex-end, baseline. Uwaga: parametr „stretch” wyrównuje do środka w poziomie, gdy kierunek jest „row”.
order: 1;}

nav > ul {
display: flex;} // Tylko jeśli elementy menu mają być osobnymi elementem typu flex

article {
flex-grow: 3;
flex-shrink: 1;
flex-basis: 60%;
order: 2;}

aside {
flex-grow: 1;
flex-shrink: 2;
flex-basis: 20%;
order: 3;}
```

Uwaga: W stosunku do *items* zawijanych jako kolumny (kierunek elementów: z góry na dół), można w CSS użyć parametru „*break-before: always;*” (np. w przypadku piątego *item*), co spowoduje, że następne *items* będą umieszczane już w drugiej kolumnie.

Stosowanie <img>

Poniższa definicja gwarantuje, że szerokość obrazka zawsze dostosuje się do szerokości ekranu (lub nadrzędnego elementu), bez względu czy smartfon będzie w pozycji pionowej czy poziomej. Atrybut „*height: auto*” gwarantuje przy tym zachowanie odpowienich proporcji obrazka.

```
img
{width: auto; // Lub width: 100%;
```

max-width: 100%;
height: auto;
margin: 0 auto 1em;}

Warunkowe stosowanie <img>

Zależne od powiększenia:
*<**img** src="fotka-domyslna.jpg" srcset="fotka-lepsza.jpg 1.5x, fotka-najlepsza.jpg 2x" alt="Fotka responsywna" width="100" height="150">*
Zależne od wielkości ekranu pobierany jest tylko jeden z obrazków, przeglądarka wybiera sobie obrazek zależnie od gęstości pikseli ekranu, ale także od aktualnego powiększenia i szybkości połączenia z Internetem (*100vw = 100% viewport width*):
*<**img** sizes="100vw" srcset="fotka-400.jpg 400w, fotka-800.jpg 800w, fotka-1600.jpg 1600w" src="fotka-400.jpg" alt="Fotka responsywna">*
lub grafika warunkowa (tzw. „*art direction*”):

```
<picture>
  <source media="(min-width: 1024px)" srcset="large.jpg">
  <source media="(min-width: 320px)" srcset="med.jpg">
// Jeśli dwie powyższe reguły okażą się fałszywe, to stosowana jest poniższa domyślna:
  <img src="small.jpg" alt="System responsywny">
</picture>
```

Dla wersji CSS:

```
img
{width: 300px;
height: 300px;}
@media (min-width: 32em) { img { width: 500px; height:300px } }
@media (min-width: 45em) { img { width: 700px; height:400px } }
```

Wersja kombinowana:

```
<picture>
   <source media="(max-width: 500px)" srcset="banner-smartphone.jpeg, banner-smartphone-HD.jpeg 2x">
   // Jeśli powyższa reguła okaże się fałszywa, stosowana jest poniższa domyślna:
   <img src="banner.jpeg" srcset="banner-smartphone-HD.jpeg 2x" alt="Banner">
</picture>
```

Inne przykłady:

```
<picture>
   <source srcset="maly-landscape.jpg" media="(max-width: 40em) and (orientation: landscape)">
   <source srcset="maly-portrait.jpg" media="(max-width: 40em) and (orientation: portrait)">
   <source srcset="domyslny-landscape.jpg" media="(min-width: 40em) and (orientation: landscape)">
   <source srcset="domyslny-portrait.jpg" media="(min-width: 40em) and (orientation: portrait)">
   <img srcset="domyslny-landscape.jpg" alt="Domyślny obrazek">
</picture>
```

lub

```
<picture>
  <source srcset="mala-fotka.jpg, mala-fotka-powiekszona.jpg 2x" media="(max-width: 768px)">
  <source srcset="domyslna-fotka.jpg, domyslna-powiekszona-fotka.jpg 2x">
  <img srcset="domyslna-fotka.jpg, domyslna-powiekszona-fotka.jpg 2x" alt="Domyślna fotka">
</picture>
```

Testowanie strony responsywnej:
http://ready.mobi
https://www.google.com/webmasters/tools/mobile-friendly/