

# Technologie tworzenia stron i aplikacji WWW

## wykład 3 – CSS 3

dr inż. Grzegorz Rogus



CSS Kaskadowe Arkusze Styli

# CSS (Kaskadowe Arkusze Stylu)

Kaskadowe arkusze stylu dzielimy na trzy grupy ze względu na miejsce wystąpienia:

1. **Wbudowane (inline-style)** - są zapisane w miejscu ich działania, tzn. w znaczniku, któremu mają nadawać specyficzne cechy.
2. **Osadzone (embedded-style)** - są zapisane za pomocą zacznika `<style>` w sekcji head dokumentu hipertekstowego.
3. **Dołączone (linked-style)** - są zapisane w osobnych plikach o rozszerzeniu .css.

Która z metod zapisywania CSS ma największy priorytet ?

**KASKADOWOŚĆ**

obowiązujący jest styl położony najbliżej elementu

# Kaskadowość – metoda rozwiązywania problem redundancji reguł

- Proces łączenia różnych arkuszy stylów i rozwiązywania konflikty między różnymi regułami CSS i deklaracjami, kiedy więcej niż jedna reguła ma zastosowanie do określonego elementu.

ISTOTNOŚĆ

priorytet  
według źródeł  
stylów

>

SPECYFICZNOŚĆ

>

KOLEJNOŚĆ

Kaskadowość w obrębie dokumentu

# CSS (Kaskadowe Akusze Stylu)

priorytet według źródeł stylów

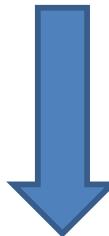


Uwaga na **!important** - łamie zasady kaskadowości

a {background-color: white !important;}

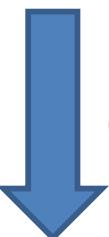
# Reguły Kaskady

ISTOTNOŚĆ



Ten sam dokument

SPECYFICZNOŚĆ



Ten sam typ selektora

KOLEJNOŚĆ

1. deklaracja **!important**
2. Lokalne style inline
3. Wewnętrzny arkusz stylów
4. Importowane style do wewnętrznego arkusza CSS
5. Deklaracja domyślna przeglądarki

A	id selectors	#foo
B	class selectors	bar
	attribute selectors	[title], [colspan="2"]
	pseudo-classes	:hover, :nth-child(2)
C	type selectors	div, li
	pseudo-elements	::before, ::first-letter

Ostatnia deklaracja w kodzie będzie przesłaniała wszystkie inne deklaracje i to ona zostanie zastosowany.

# Kaskadowość w obrębie dokumentu

1

```
.button {  
    font-size: 20px;  
    color: white;  
    background-color: blue;  
}
```

2

```
nav#nav div.pull-right .button {  
    background-color: green;  
}
```

3

```
a {  
    background-color: purple;  
}
```

4

```
#nav a.button:hover {  
    background-color: yellow;  
}
```

Inline  
IDs  
Classes  
Elements

~~(0, 0, 1, 0)~~

(0, 1, 2, 2)

~~(0, 0, 0, 1)~~

~~(0, 1, 2, 1)~~

→ Wynik selekcji

Selektor zawierający 1 identyfikator jest bardziej szczegółowy niż selektor zawierający 1000 klas;  
Selektor zawierający 1 klasę jest bardziej szczegółowy niż selektor zawierający 1000 elementów;  
Selektor uniwersalny \* nie ma wartości specyficzności (0, 0, 0, 0);

Własności	
<b>Fonty</b>	font-family font-size font-weight font-style font-variant text-decoration @font-face
<b>Wielkość</b>	width height
<b>Układ</b>	position left, right float, clear
<b>Grafika</b>	background-image background-repeat background-position
<b>Barwy</b>	color background-color
<b>Przestrzeń i odstępy</b>	margin padding margin-left, margin-right, margin-top, margin-bottom padding-left, padding-right, padding-top, padding-bottom
<b>Obramowanie</b>	border-width border-style border-color border (ustawia grubość, styl i barwę za jednym zamachem)
<b>Wyrównanie tekstu</b>	text-align text-indent word-spacing letter-spacing line-height white-space

# SELEKTORY CSS – IDENTYFIKACJA

**\*** - spełniony przez każdy element

**E** - spełniony przez każdy element E

```
body { font-family:Verdana; font-size:11px; }
```

**E.value** - spełniony przez te elementy E, które posiadają atrybut CLASS o wartości value

```
p.title1 { font-size:18px; }
```

**E#value** - spełniony przez te elementy E, które posiadają atrybut ID o wartości value

```
img#news { border:2px double #FFFFFF; }
```

# SELEKTORY kontekstowe CSS – ZAGNIEŻDŻANIE (selektory potomków)

**E F** - spełniony przez każdy element F zagnieżdżony wewnątrz elementu E

```
p b { color:#FFFFFF; }
```

**E > F** - spełniony przez każdy element F zagnieżdżony bezpośrednio w E

```
ul > li { font-family:Verdana; font-size:11px; }
```

**E + F** - spełniony przez każdy element F, który następuje bezpośrednio za

elementem E

```
i + b { color:#000000; }
```

# Wykorzystanie selektorów potomków (selektory kontekstowe)

```
div ul { color: blue }

<div>
  <p>Moje ulubione sporty to:</p>
  <ul>
    <li>narciarstwo,</li>
    <li>kolarstwo,</li>
    <li>pływanie.</li>
  </ul>
</div>

<p>Moje ulubione sporty to:</p>
<ul>
  <li>narciarstwo,</li>
  <li>kolarstwo,</li>
  <li>pływanie.</li>
</ul>
```

Moje ulubione sporty to:

- narciarstwo,
- kolarstwo,
- pływanie.

Moje ulubione sporty to:

- narciarstwo,
- kolarstwo,
- pływanie.

```
div.figure p
{
  text-align: center;
  font-size: smaller;
  text-indent: 0;
}

<div class="figure">
  <p></p>
  <p>Kask Breeze firmy Mango</p>
</div>
```



Kask Breeze firmy Mango

# Wykorzystanie selektorów potomków (selektory kontekstowe)

## Wykorzystanie selektorów „dzieci”

```
body > p { color: blue }

<body>
  <p>To jest dziecko body</p>
  <div>
    <p>To jest potomek body</p>
  </div>
</body>
```

To jest dziecko body

To jest potomek body

## Wykorzystanie selektorów „braci”

```
p + p
{
  text-indent: 0.7em;
  margin-top : 0
}
```

<p>Pierwszy paragraf w tym tekście nie musi posiadać wcięcia akapitowego</p>

<p>Drugi paragraf w tym tekście juz powinien takie wcięcie posiadać</p>

</p>

<p>Za wyjątkiem tych paragrafów które następują po rysunkach</p>

Pierwszy paragraf w tym tekście nie musi posiadać wcięcia akapitowego

Drugi paragraf w tym tekście juz powinien takie wcięcie posiadać



Za wyjątkiem tych paragrafów które następują po rysunkach

Tytuł

# SELEKTORY CSS – PSEUDOKLASY

**A:link** - spełniony przez każdy link, który nie został jeszcze odwiedzony

`a:link { text-decoration:none; }`

**A:visited** - spełniony przez każdy link, który został odwiedzony

`a:visited { text-decoration:underline; }`

**E:hover** - spełniony przez każdy element E, nad którym właśnie znajduje się wskaźnik

`tr:hover { text-decoration:underline; }`

**E:focus** - spełniony przez każdy element E, na którym właśnie znajduje się focus dokumentu

`input:focus { border-radius:3px; }`

DEMO -> linki.html

# SELEKTORY CSS – KOLEJNOŚĆ

**E:first-child** – spełniony przez pierwszy element E

```
ul li:first-child {color:#f46c00;}
```

**E:last-child** – spełniony przez ostatni element E

```
ul li:last-child {color:#000000;}
```

**E:nth-child(n)** – spełniony przez n-ty element E

```
ul li:nth-child(2) {text-decoration:underline;}
```

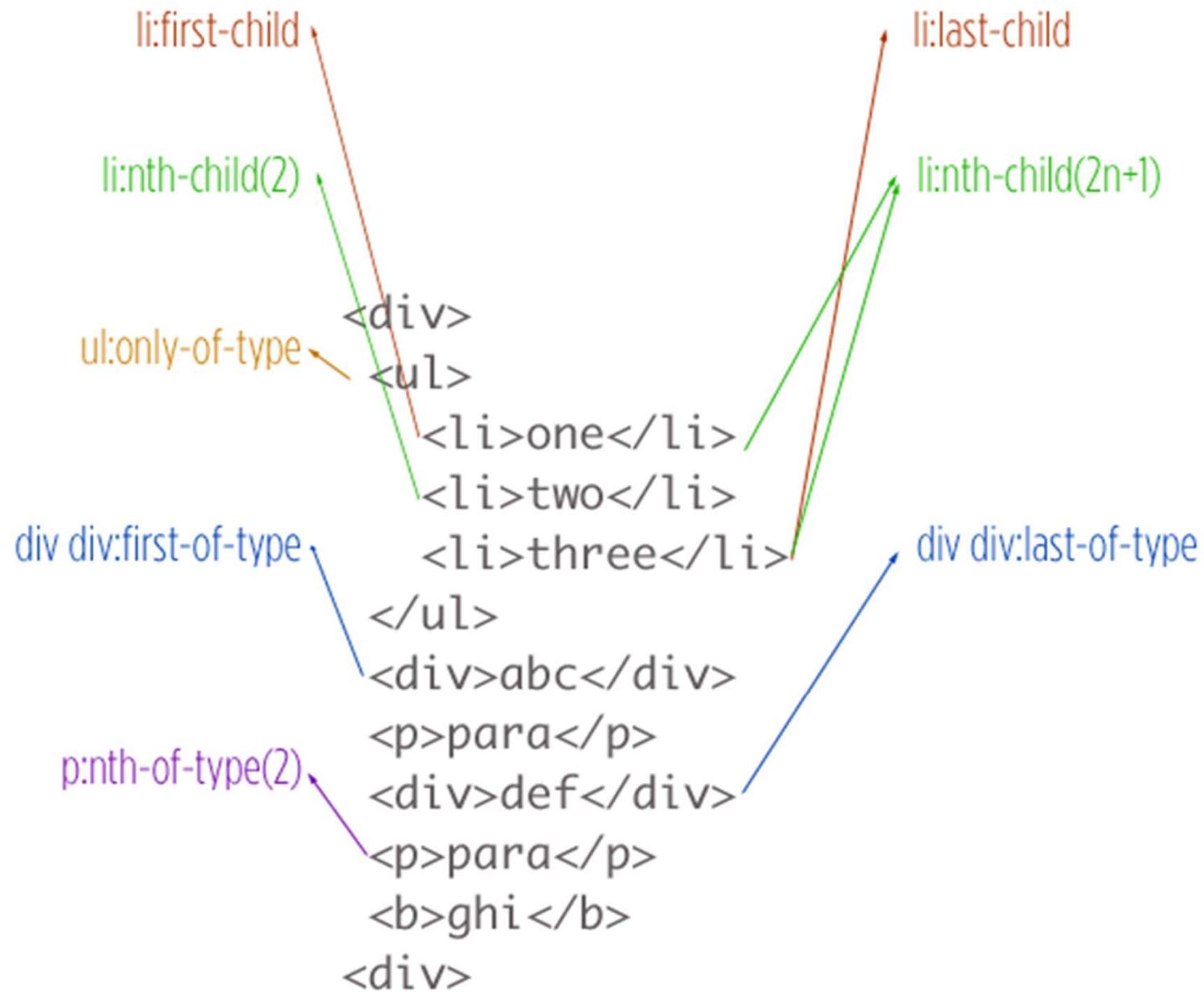
**E:nth-child(even)** – spełniony przez wszystkie nieparzyste elementy E

```
ul li:nth-child(even) {text-decoration:underline;}
```

**E:nth-child(odd)** – spełniony przez wszystkie parzyste elementy E

```
ul li:nth-child(odd) {text-decoration:none;}
```

# SELEKTORY CSS - PSEUDOELEMENTY



# SELEKTORY CSS - PSEUDOELEMENTY

**E::after** - wstawia coś po zawartości elementu E

```
p::after {content: url(smiley.gif) }
```

**::before** - wstawia coś przed zawartością elementu E

```
p::before {content: url(smiley.gif) }
```

**E::first-letter** - spełniony przez pierwszą literę elementu E

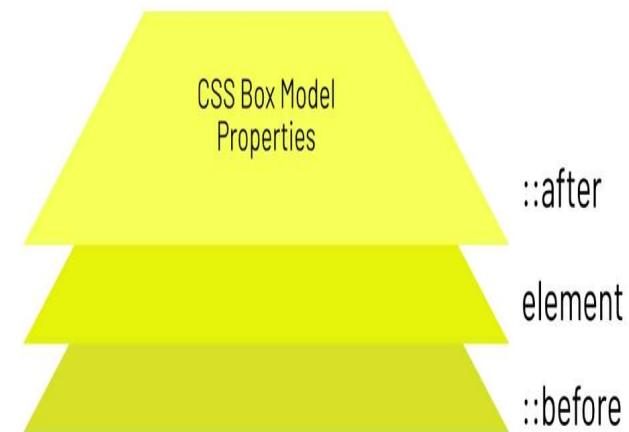
```
p::first-letter {font-size:2em; }
```

**E::first-line** - spełniony przez pierwszą linię elementu E

```
p::first-line {font-style:italic; }
```

**E::selection** - spełniony przez część obiektu E wybranego przez użytkownika

```
p::selection {background-color:#d4d4d4; }
```



# Zmienne w CSS – bez użycia preprocesora

Zmienne w CSS są definiowane za pomocą specjalnej notacji:

```
:root {  
    --primary-color: yellow;  
}
```

Dostęp do zmiennych jest możliwy za pomocą funkcji var() :

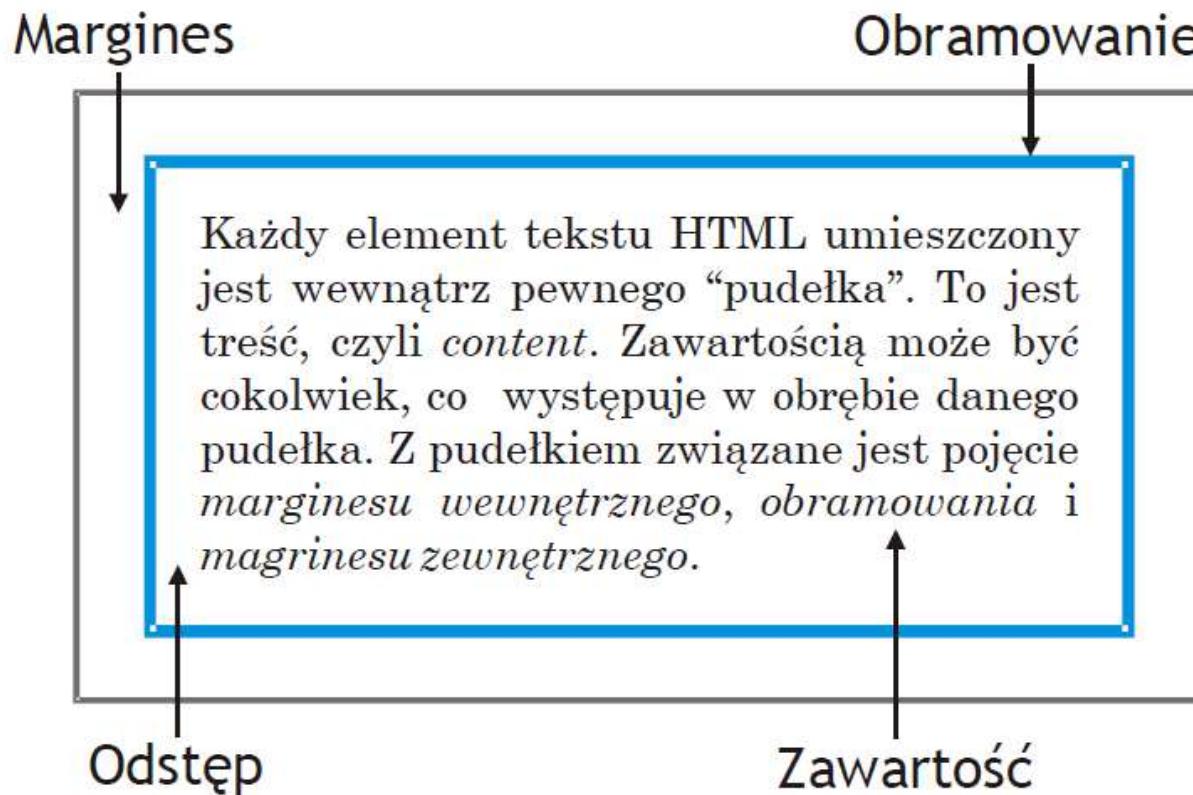
```
p {  
    color: var(--primary-color)  
}
```

Wartością zmiennych może być dowolna wartość CSS deklarowana w dowolnym selektorze:

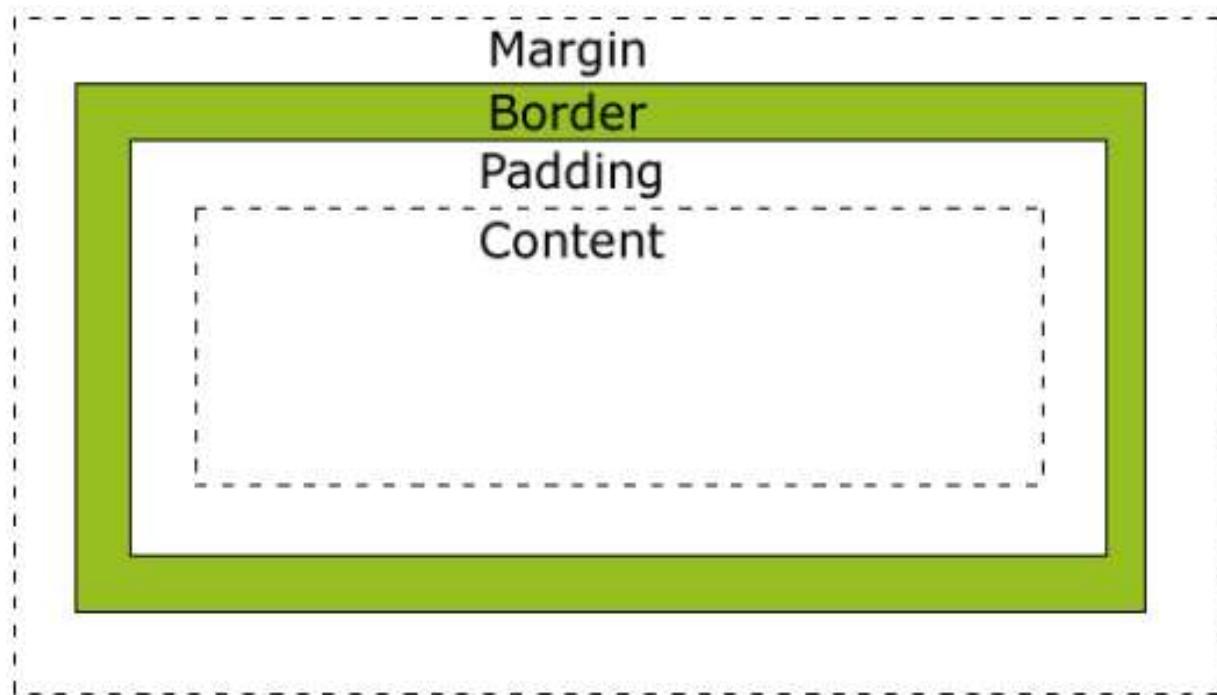
```
:header {  
    --default-padding: 30px 30px 20px 20px;  
    --default-color: red;  
    --default-background: #fff;  
}
```

# BOX MODEL

Koncepcja modelu pojemnika zakłada, że każdy element dokumentu HTML może być traktowany jako prostokątny obszar, którego zawartość otoczona jest *marginesem wewnętrzny (odstępem), obramowaniem i marginesem zewnętrznym*.



# BOX MODEL – całkowita szerokość



[http://www.w3schools.com/css/css\\_boxmodel.asp](http://www.w3schools.com/css/css_boxmodel.asp)

Oznacza to, że rzeczywista wielkość danego obiektu jest sumą wielkości zawartości, paddingu, ramki oraz marginesu.

**Szerokość** = szerokość elementu + lewy odstęp + prawy odstęp + lewa ramka + prawa ramka + lewy margines + prawy margines

# BOX MODEL

## Przykład wykorzystania modelu pojemnika

```
h1  
{  
    background-color: skyblue;  
}  
...
```

**Model pojemnika — box model**

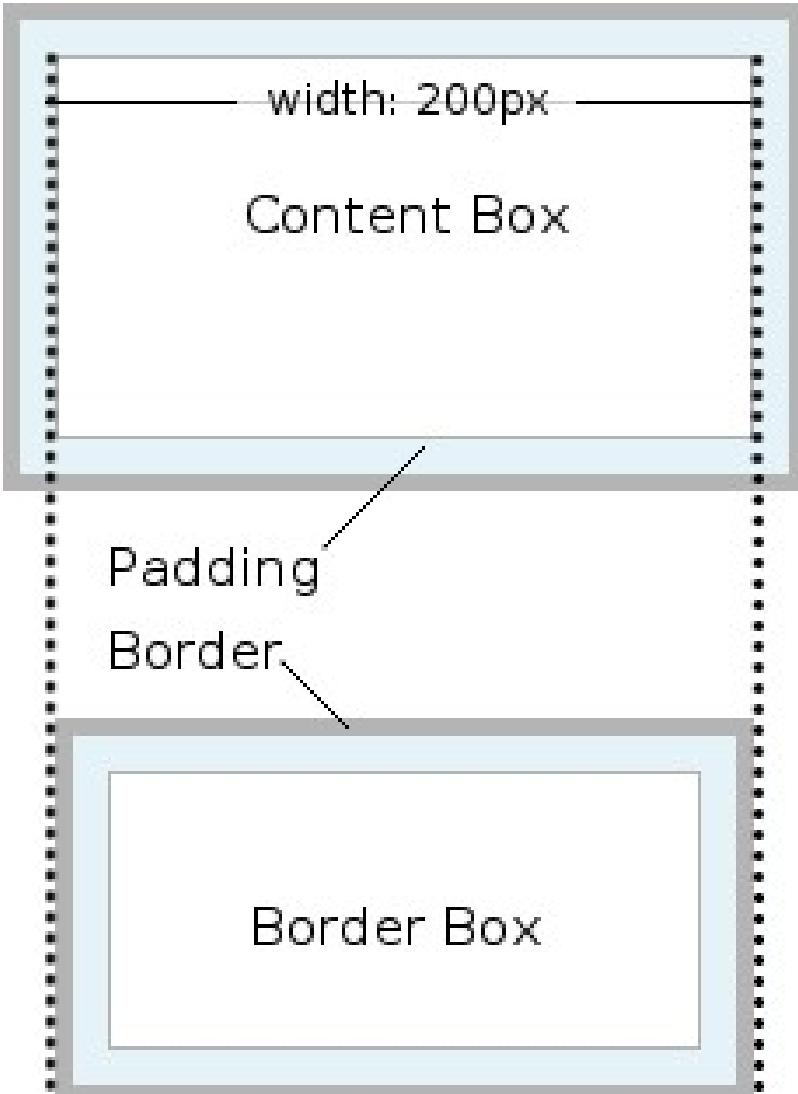
```
<h1>Model pojemnika &mdash; box model</h1>
```

```
h1  
{  
    background-color: skyblue;  
  
    padding-left: 20px;  
    padding-top: 15px;  
    padding-bottom: 15px;  
    padding-right: 20px;  
}  
...
```

**Model pojemnika — box model**

```
<h1>Model pojemnika &mdash; box model</h1>
```

# box-sizing – inny model szerokości



Czym jest podana szerokość: ????

Całkowita (pułka) = 234px;  
 $200+2\times5+2\times10+2\times2$

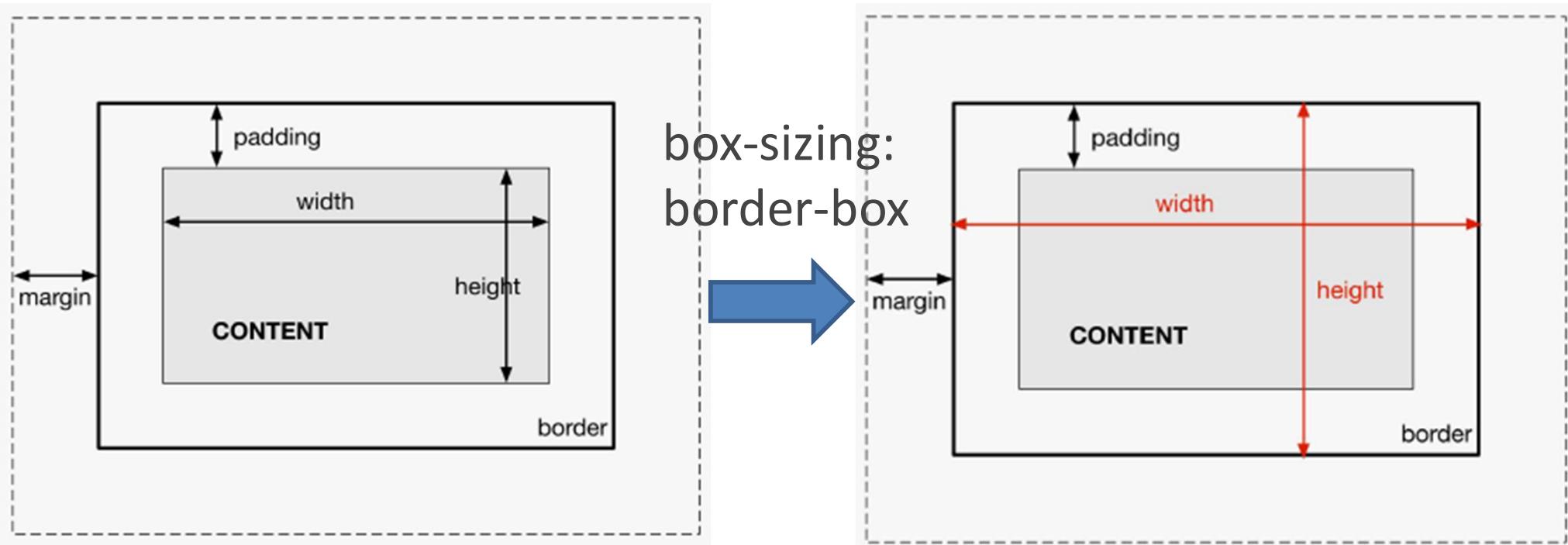
```
p {  
    margin: 10px;  
    padding: 5px;  
    width: 200px;  
    border-width: 2px;  
}
```

CSS3 wprowadza box-sizing

- content-box (szerokość = zawartość)
- border-box (szerokość = zawartość + obramowanie + odstęp)

Całkowita (pułka) = 220px;  
 $200+2\times10$

# box-sizing – inny model szerokości cd.



**width** = right border + right padding +  
content width + left padding + left  
border

**height** = top border + top padding +  
content height + bottom padding +  
bottom border

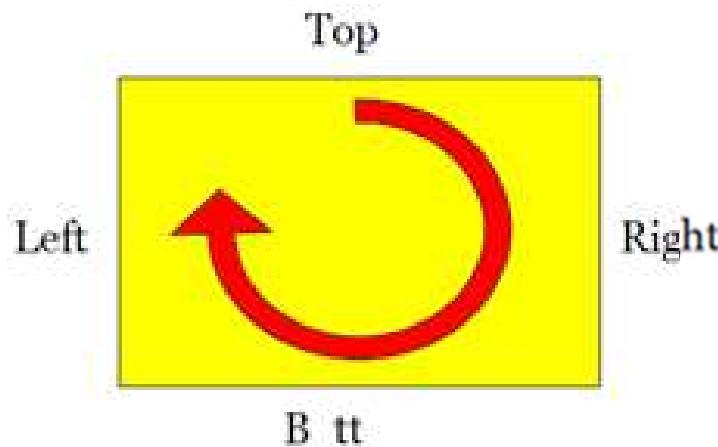
**width** = ~~right border + right padding +~~  
~~content width + left padding + left~~  
~~border~~

**height** = ~~top border + top padding +~~  
~~content height + bottom padding +~~  
~~bottom border~~

# ZASADA WSKAZÓWEK ZEGARA

Jak zapamiętać przypisanie wartości do określonych boków pojemnika?

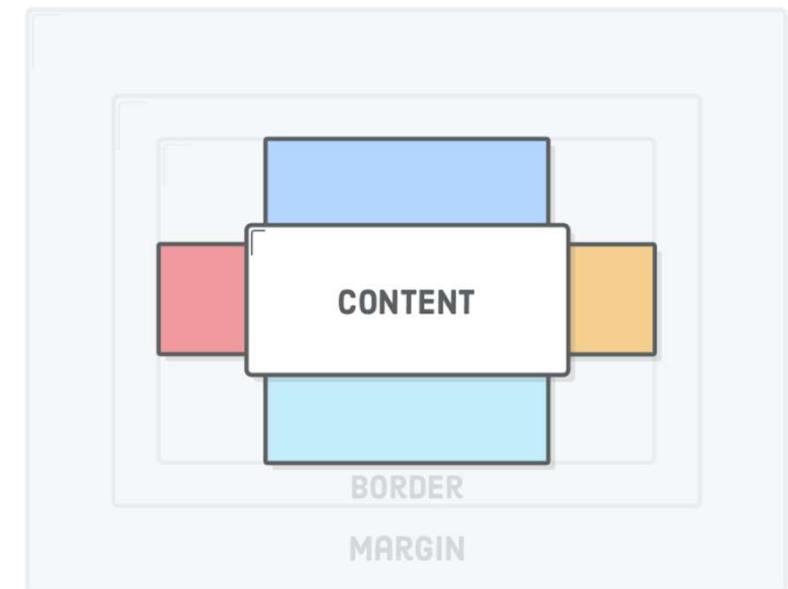
- ▶ **Reguła stopera lub zegara wskazującego 12-stkę,**
- ▶ **Reguła trouble – TopRightBottomLeft.**



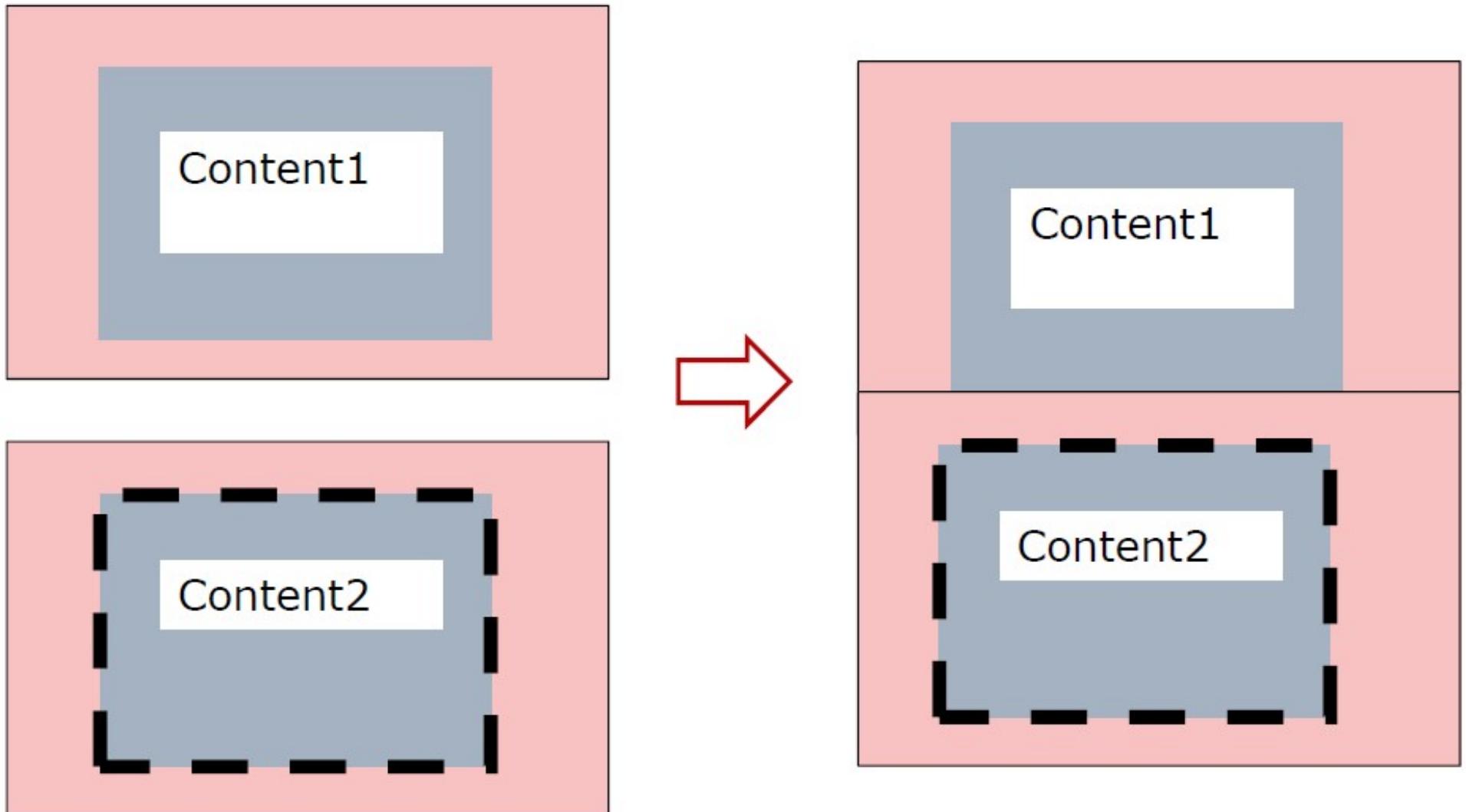
border-width: 6px;

border-width: 0px 3px 2px 10px;

border-width: 3px 0px;



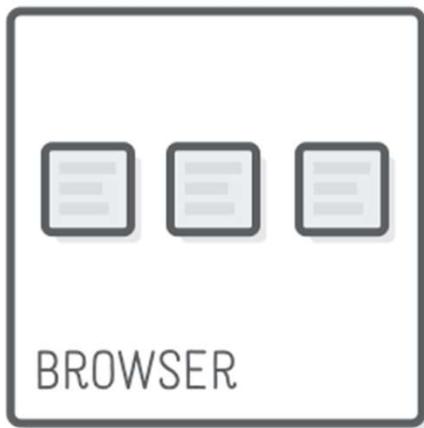
# Zapadanie marginesów



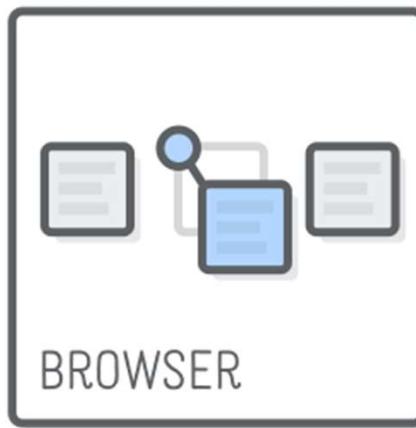
- ▶ Jeżeli dwa, lub więcej, pojemniki sąsiadują obok siebie pionowo to *niezależnie* od ustalonych dla nich marginesów pionowych, odstęp pomiędzy nimi będzie równy *większej z ustalonych wartości*.

# Pozycjonowanie elementów na stronie

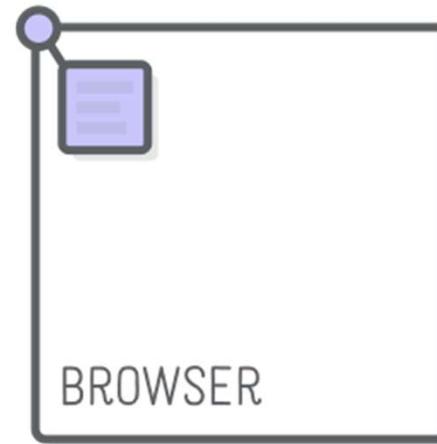
**STATIC**



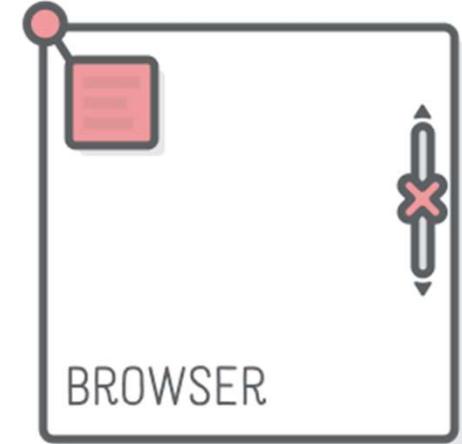
**RELATIVE**



**ABSOLUTE**



**FIXED**



# Pozycjonowanie elementów na stronie

Pozycjonowanie służy w CSS do ustalania, względem czego układają się elementy. Występują przeważnie z właściwościami *top*, *bottom*, *left* i *right*.

Istnieją 4 rodzaje pozycjonowania:

- **position: static** – elementy układają się w kolejności ich umieszczania, nie nakładają się na siebie;
- **position: fixed** - elementy układają się względem okna przeglądarki, np.:

```
div { position:fixed; top:20px; right:30px; }
```

ustawi element 20px od góry i 30px od prawej strony okna.

# Pozycjonowanie elementów na stronie

- **position: relative** – element układa się względem elementu poprzedniego, np.:

```
div { position:relative; left:30px; }
```

ustawi element w odległości 30px od elementu poprzedniego.

- **position: absolute** – element układa się względem elementu, w którym jest zamknięty i który nie jest opisany jako position:static.

```
div { position:absolute; left:30px; }
```

ustawi element w odległości 30px od krawędzi elementu nadziednego.

# POSITION I PRZEPŁYW DOKUMENTU

Obiekty **static** i **relative** uczestniczą w tworzeniu przepływu dokumentu.

Obiekty **fixed** i **absolute** są wyrwane z przepływu dokumentu i przez to nie wpływają na wysokość elementów, w których się znajdują.

Aby to obejść bardzo często korzysta się z par obiektów o pozycjonowaniu **relative** i **absolute**.

Obiekty „relative” tworzą kontenery dla obiektów „absolute”, dzięki czemu można precyzyjnie rozmieścić elementy na stronie, a jednocześnie zachować prawidłowy przepływ.

# PRZEPŁYW DOKUMENTU

Elementy HTML w dokumencie układają się na kilka sposobów:

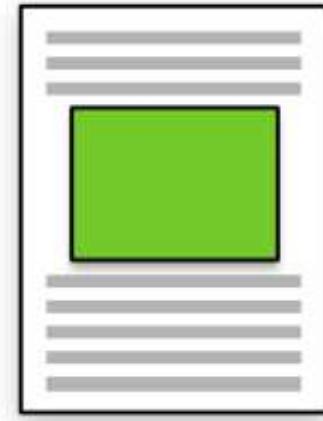
## Normalny przepływ treści (ang. normal flow)

Elementy układają się kolejno jeden pod drugim.

Obecność elementu w dokumencie odsuwa inne elementy, tak że żadne na siebie się nie nakładają.

To jest domyślne zachowanie w CSS i określone jest pozycją statyczną:

**position: static;**



# PRZEPŁYW DOKUMENTU

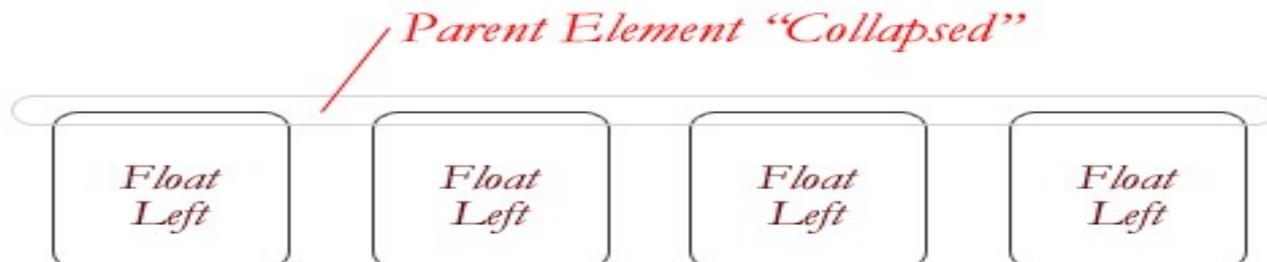
## Wyjęcie z przepływu treści

Obiekt przestaje istnieć w swoim oryginalnym miejscu w dokumencie i pozostałe elementy są rozstawiane, jakby tego obiektu nie było.

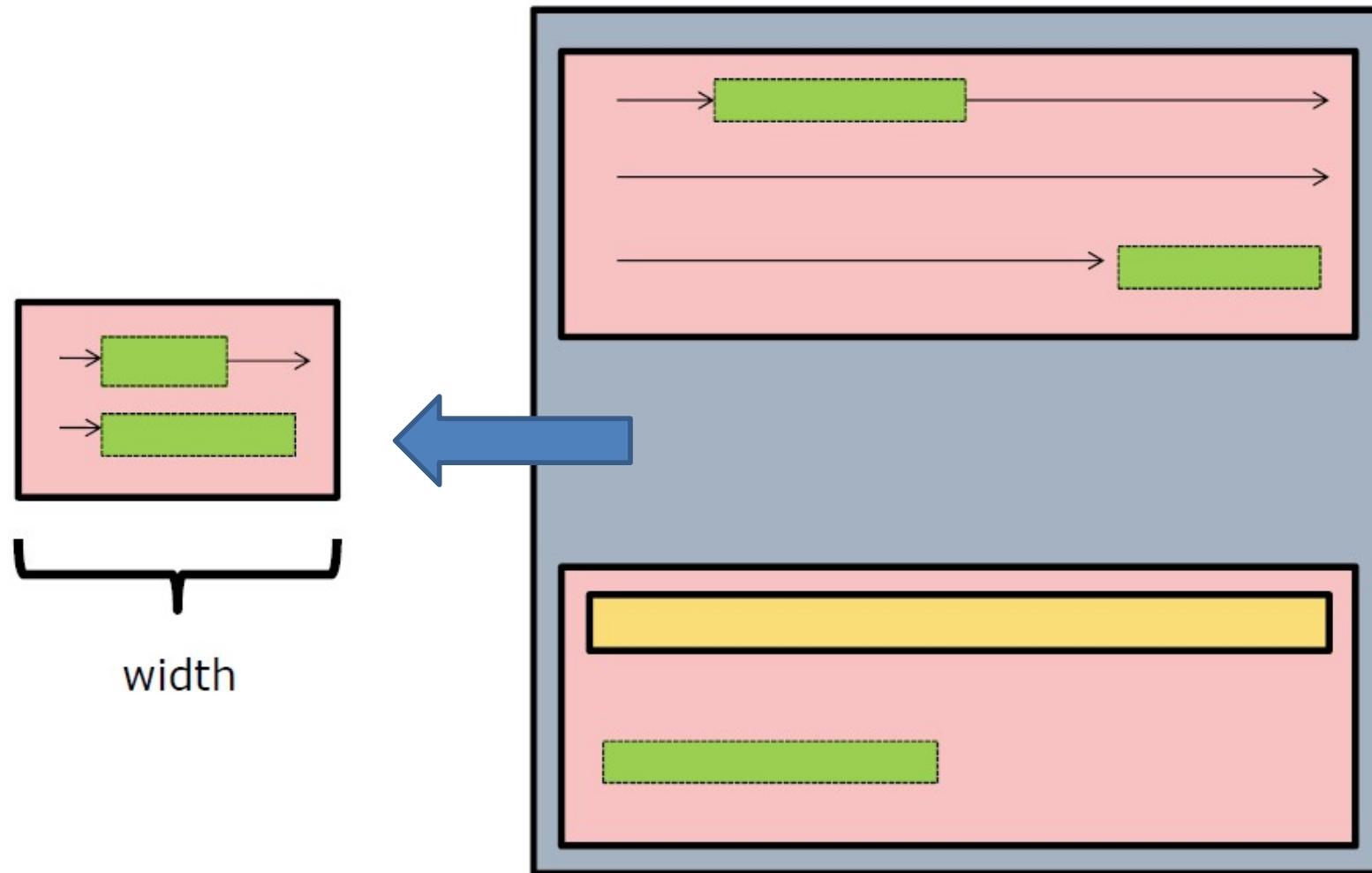
Obiekty wyjęte z biegu dokumentu najczęściej umieszczane są w innym miejscu strony przez **pozycjonowanie absolutne** lub **float**.



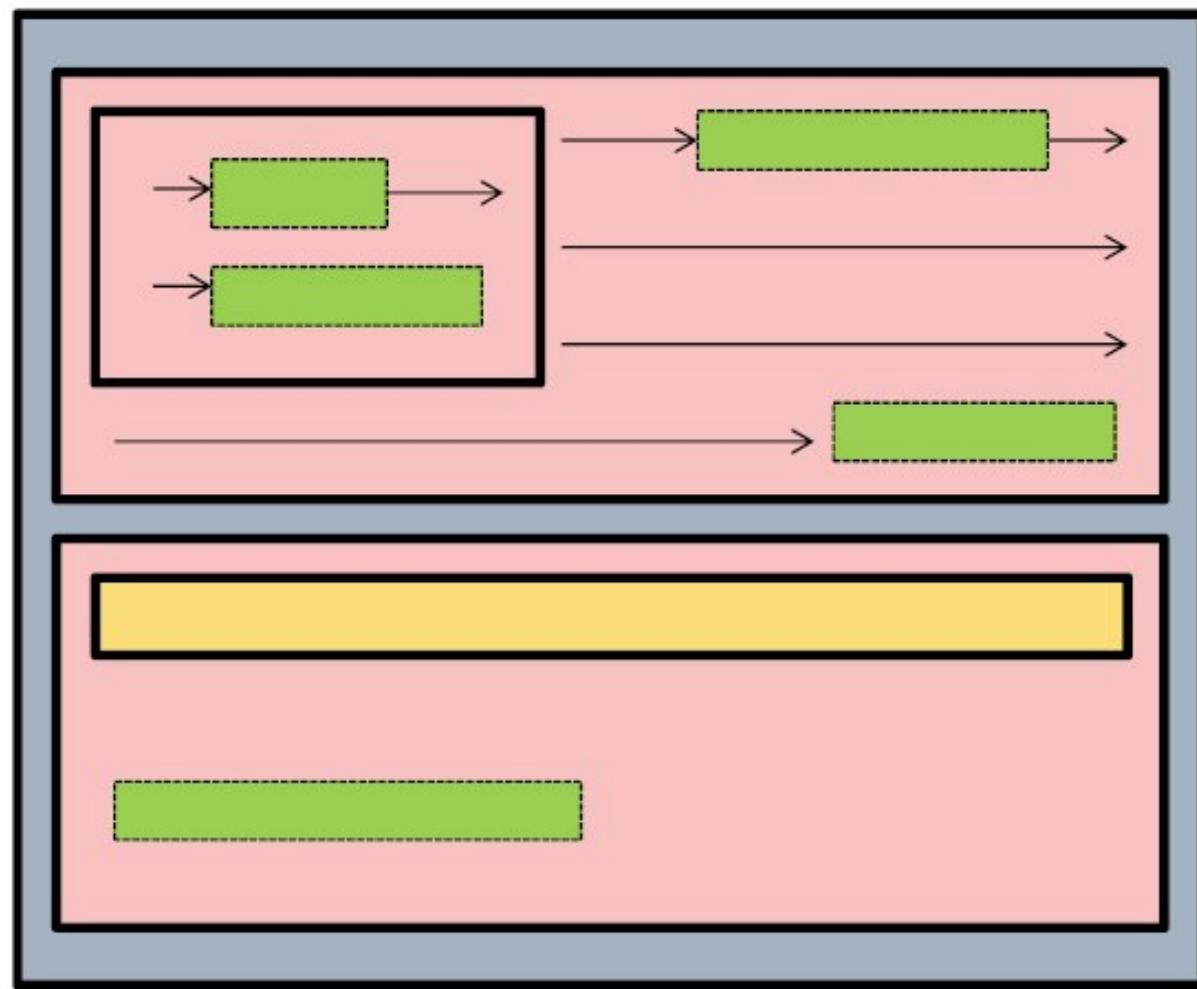
Jeżeli wszystkie dzieci danego elementu są wyjęte z biegu dokumentu, to element będzie miał zerową wysokość, ponieważ nie będzie rezerwował miejsca na żaden element w nim.



# Floating – wyjście z normalnego przepływu

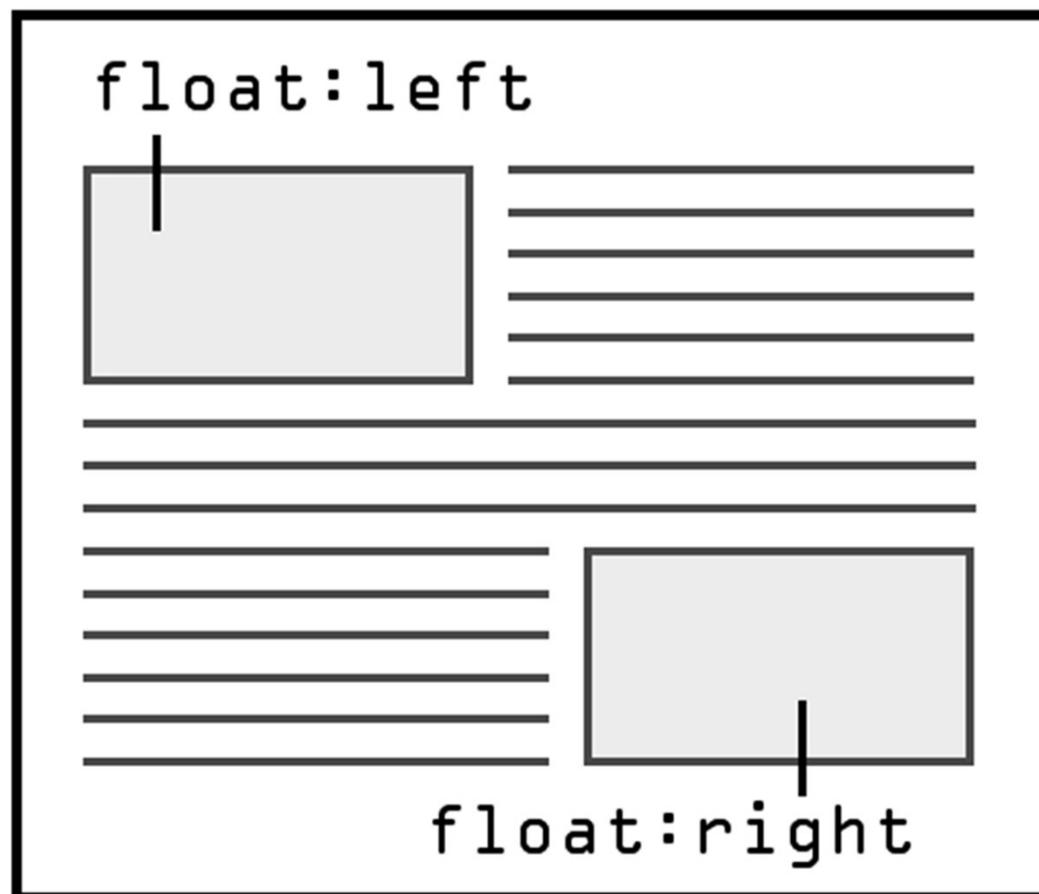


# Przepływ dokumentu



# FLOAT

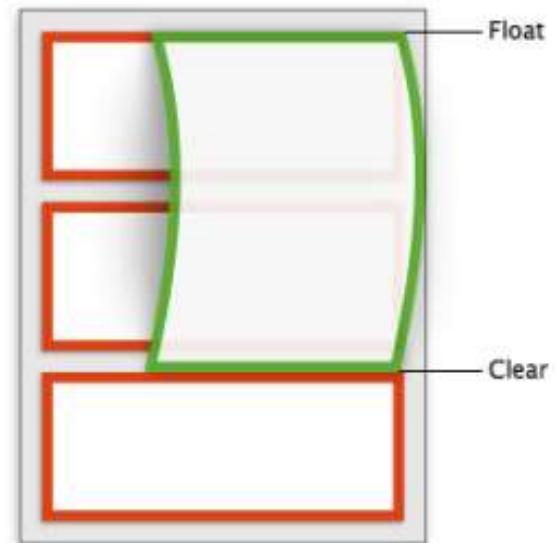
- **left** - dla umieszczenia elementu z lewej strony,
- **right** - dla umieszczenia elementu z prawej strony,
- **none** - oznacza, że element nie będzie pływający. To jest domyślna wartość dla wszystkich elementów i zazwyczaj nie trzeba jej nadawać.



# ZAPOBIEGANIE OPŁYWANIU (CLEAR)

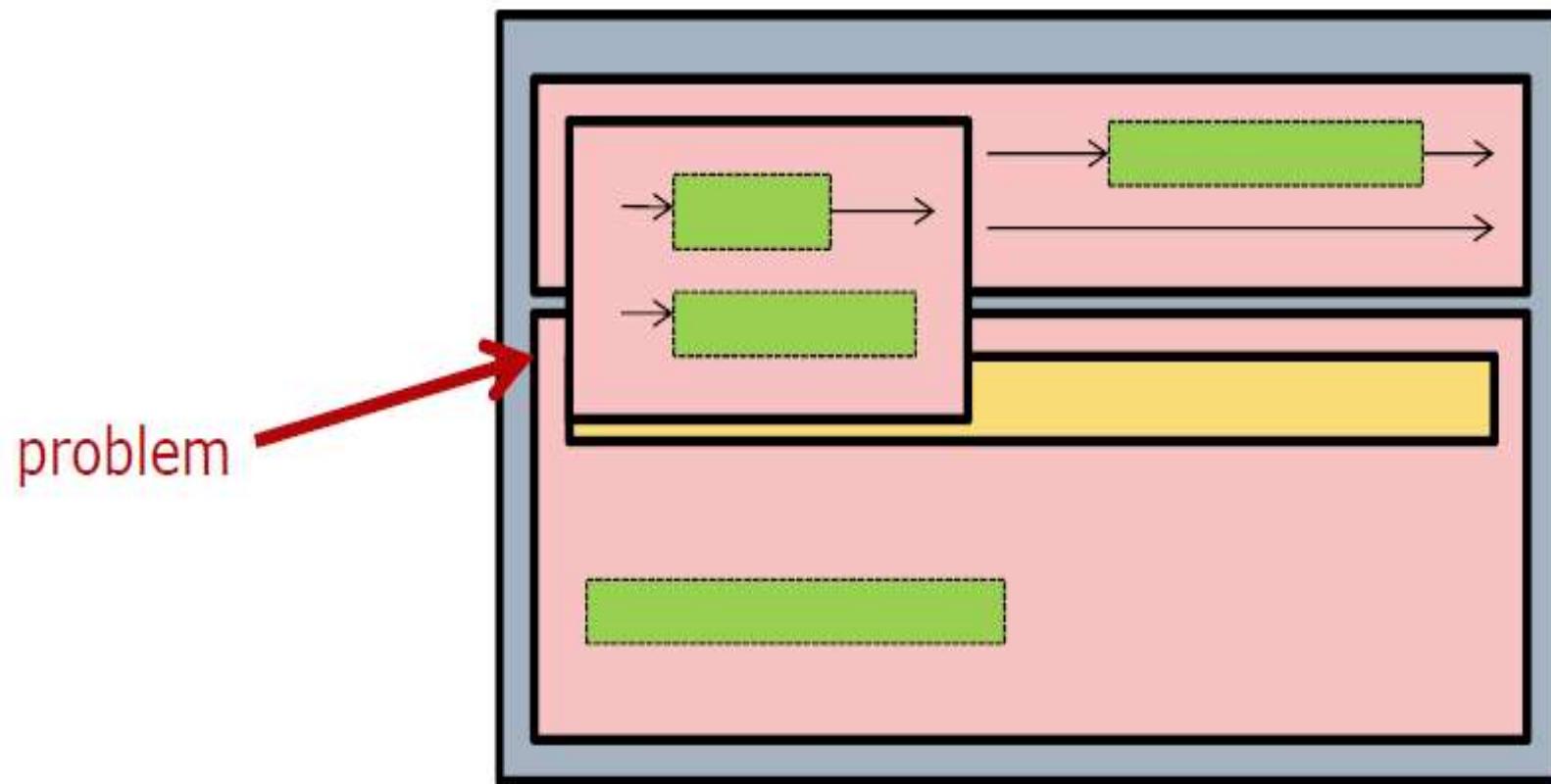
## Przerywanie przez clear

Ponieważ obiekty z float są wyjęte z normalnego biegu dokumentu, mogą rozciągać się nad kilkoma innymi — np. ilustracja może rozciągać się z boku kilku akapitów.

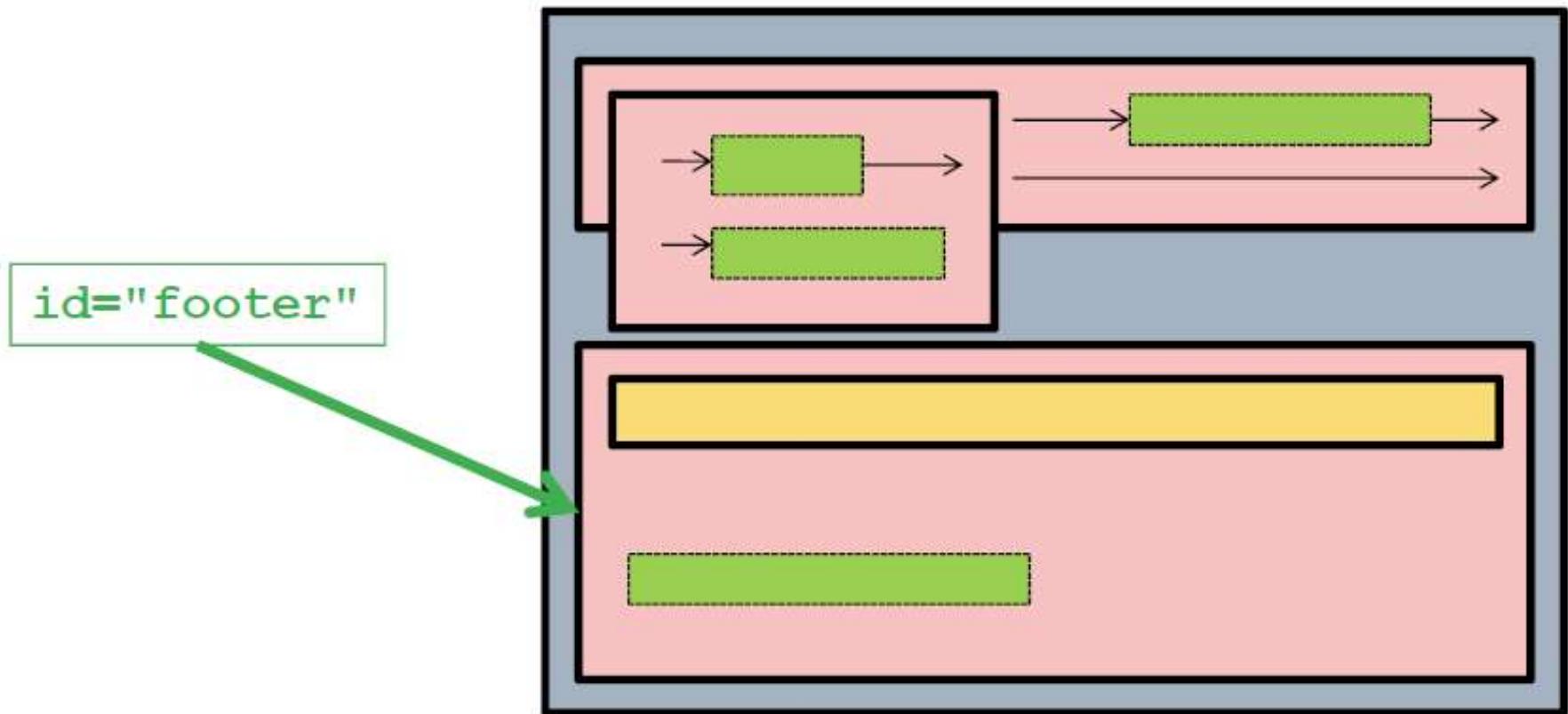


Aby uniknąć tego efektu należy jakiemuś elementowi za elementem z float nadać właściwość clear — powoduje ona ponowne „złączenie” float z biegiem dokumentu.

# Przepływ dokumentu - opływanie



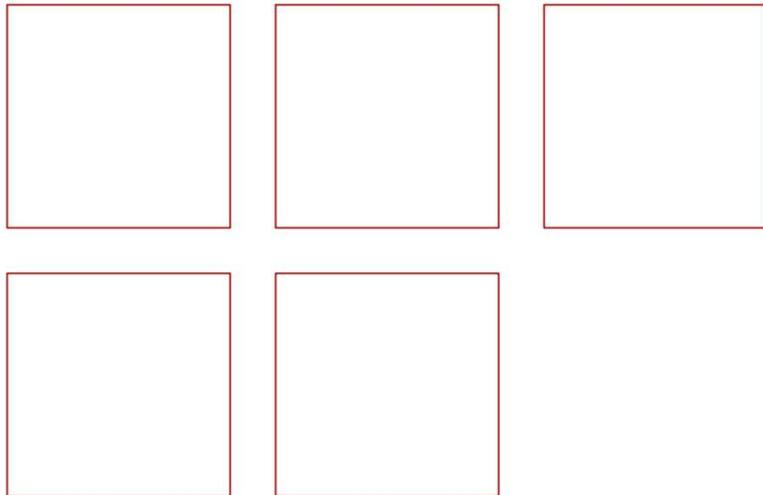
# Wykorzystanie właściwości clear



```
#footer { clear: left; }
```

# Float – problem z wyświetlaniem w kontenerze

GR - Zawartość kontenera



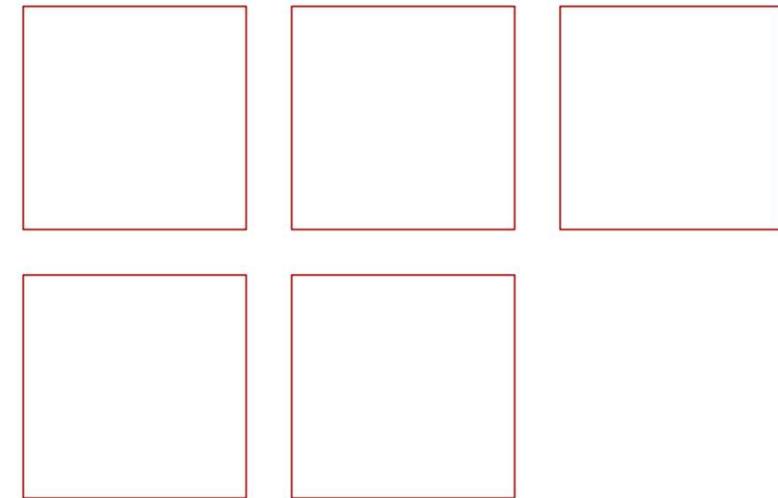
Brak  
wysokości

```
<div id="main_container">  
  <p>GR – zawartość kontenera</p>  
  <div class="floated_box"></div>  
  <div class="floated_box"></div>  
  <div class="floated_box"></div>  
  <div class="floated_box"></div>  
  <div class="floated_box"></div>  
  <div style="clear: both;"></div>  
</div>
```

```
.floated_box {  
  float: left;  
  width: 100px;  
  height: 100px;  
  border: 1px solid #990000;  
  margin: 10px; }
```

```
#main_container {  
  width: 400px;  
  margin: 20px auto;  
  border: 2px solid #cccccc;  
  padding: 5px; }
```

GR - Zawartość kontenera



# Overflow

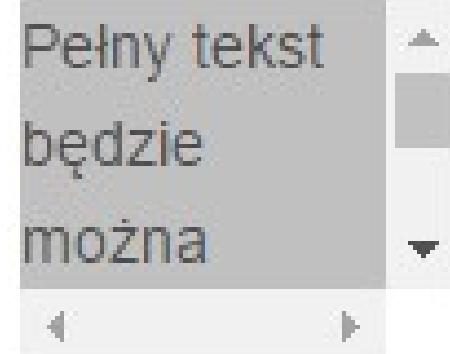
Za pomocą właściwości overflow możemy ustalić jak ma zachować się element HTML w momencie gdy jego zawartość nie będzie mieściła się w jego rozmiarach.

- visible - domyślne
- hidden - zakrywanie nie mieszczącej się zawartości w zadeklarowanych ramach
- scroll - przewijanie
- auto - w razie konieczności pojawią się przewijanie

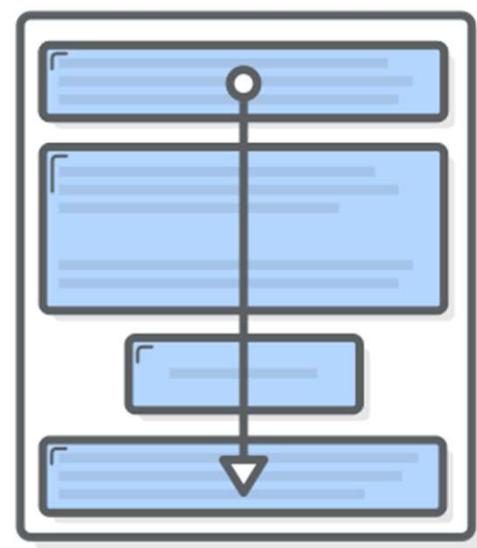
Nadwiar tekstu  
w tym bloku  
zostanie  
przycięty.

Cały tekst jest  
widoczny, bez  
przycinania i  
konieczności  
pojawienia się  
przewijania.

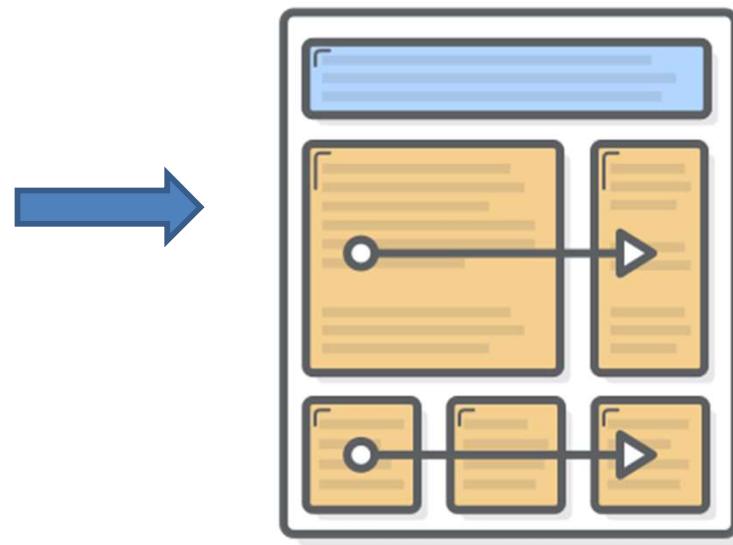
Pełny tekst  
będzie  
można



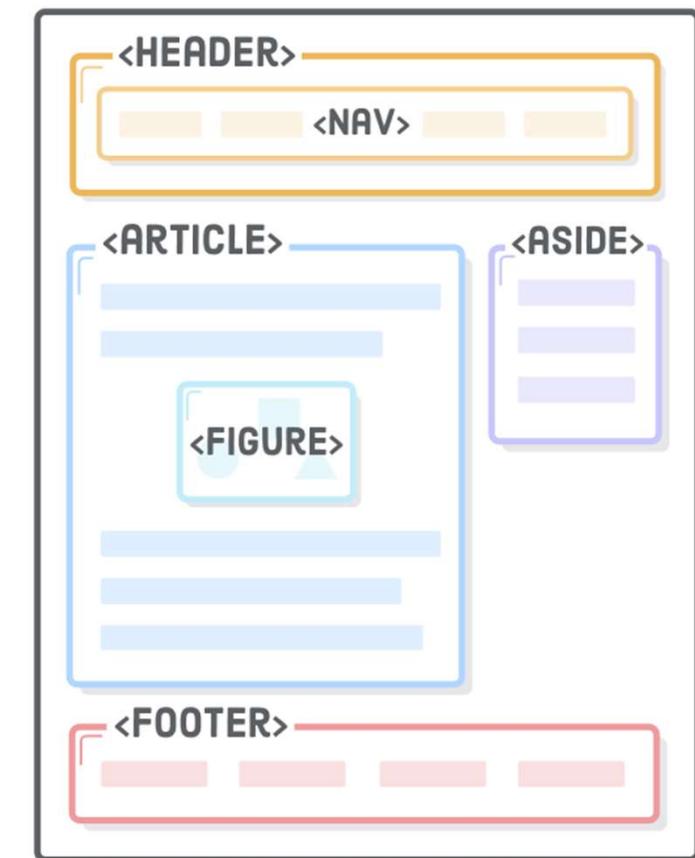
# Tworzenie Layout (Układ strony)



VERTICAL FLOW

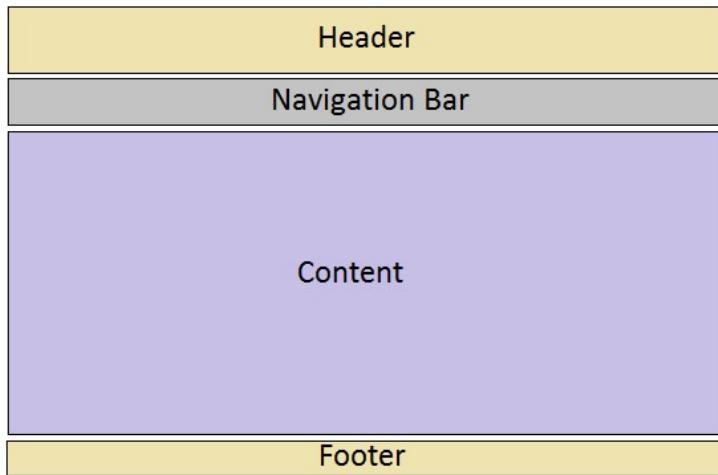


HORIZONTAL FLOW

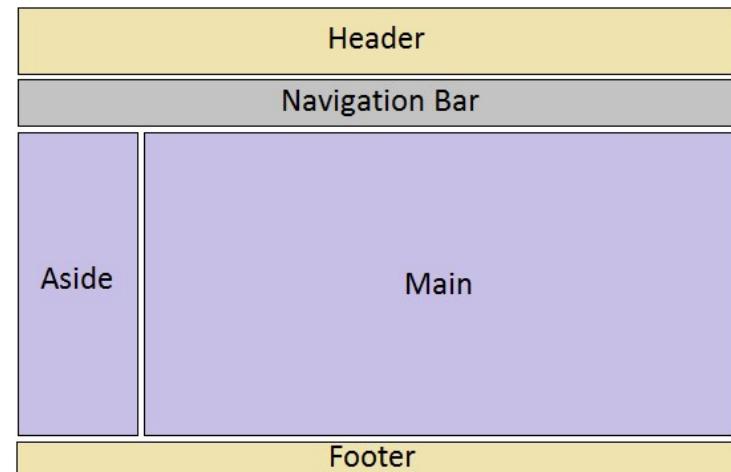


# Najpopularniejsze szablony

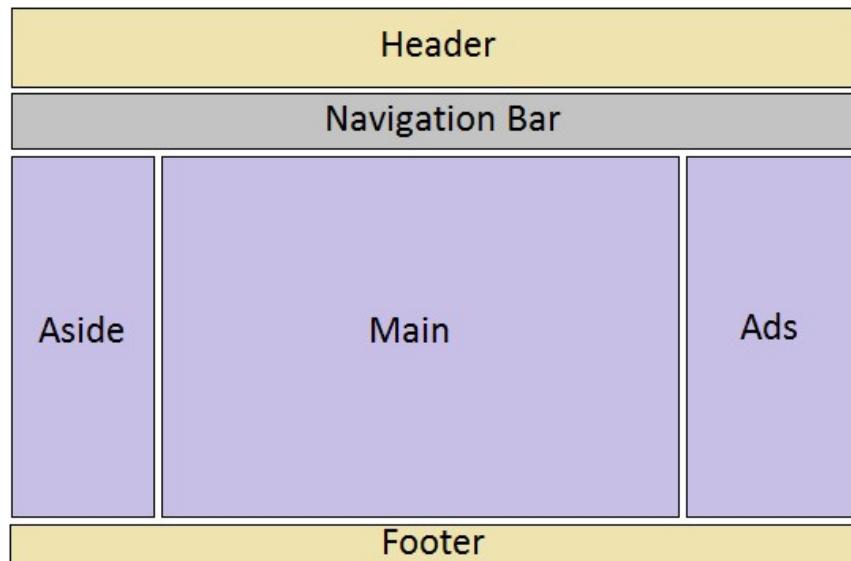
## ➤ One-column layout



## ➤ Two-column layout



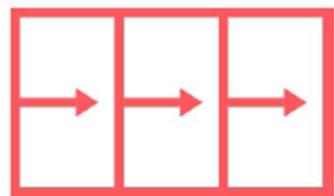
## ➤ Three-column layout



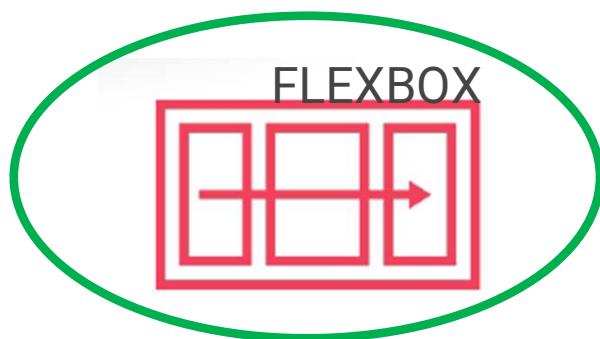
# Tworzenie layout

- Floats
- Inline-block
- ~~display: table~~
- Absolute & Relative positioning

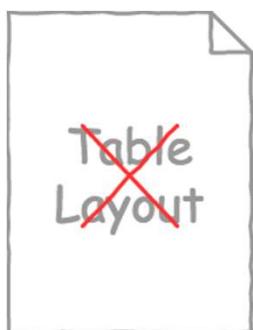
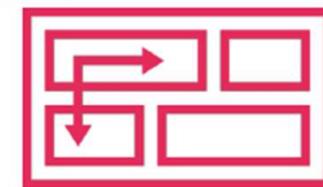
FLOAT LAYOUTS



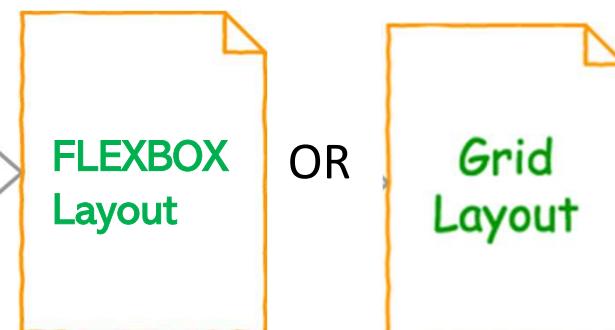
FLEXBOX



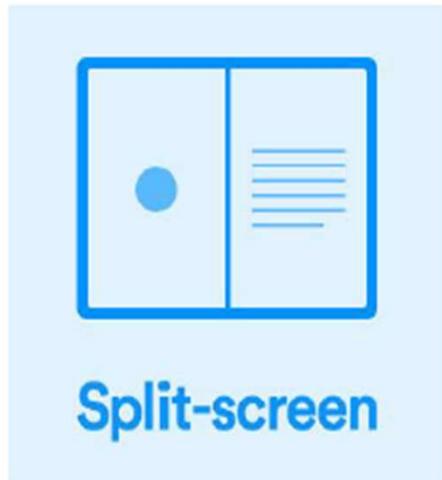
CSS GRID



OR



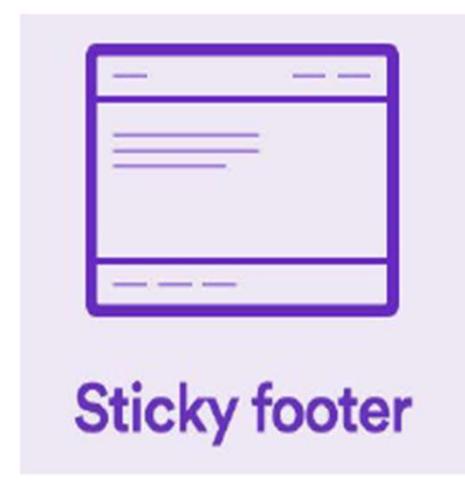
# Flexbox - zastosowanie



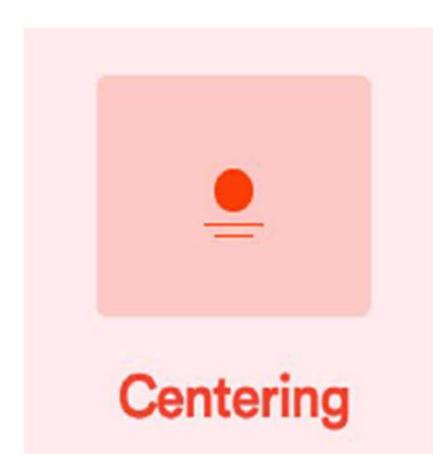
Split-screen



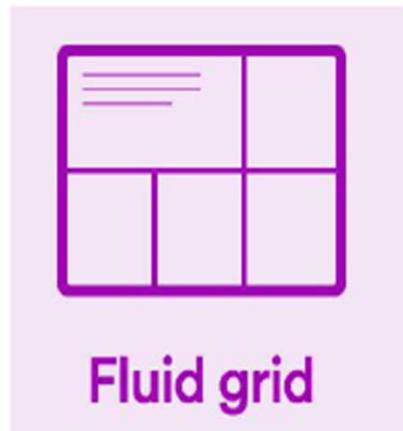
Sidebar



Sticky footer



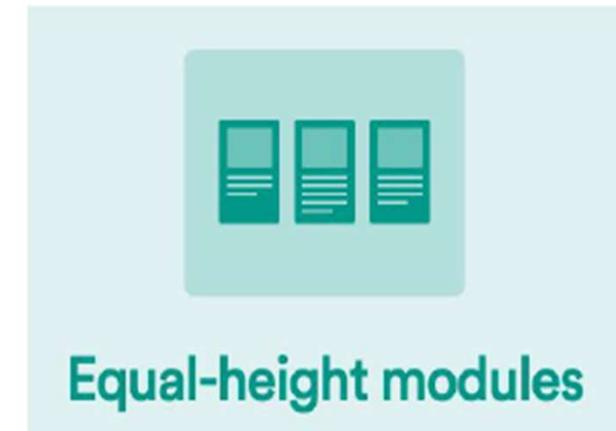
Centering



Fluid grid



Collection grid



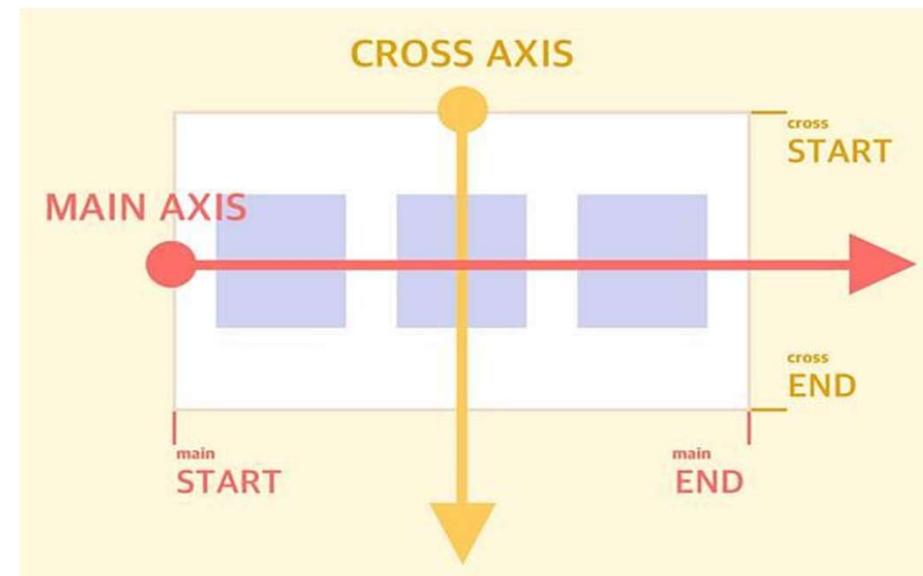
Equal-height modules

# FLEXBOX – najważniejsze fakty

**`id=flex-container`**

Umieść w kontenerze: **`display: flex;`**

**`class=`  
`flex-`  
`item`**



# Flexbox



Parent

**flex-direction**  
**flex-wrap**  
**flex-flow**  
**justify-content**  
**align-items**  
**align-content**



Child

**order**  
**flex-grow**  
**flex-shrink**  
**flex-basis**  
**flex**  
**align-self**

# FLEXBOX – co kontrolujemy?



ALIGNMENT



DIRECTION

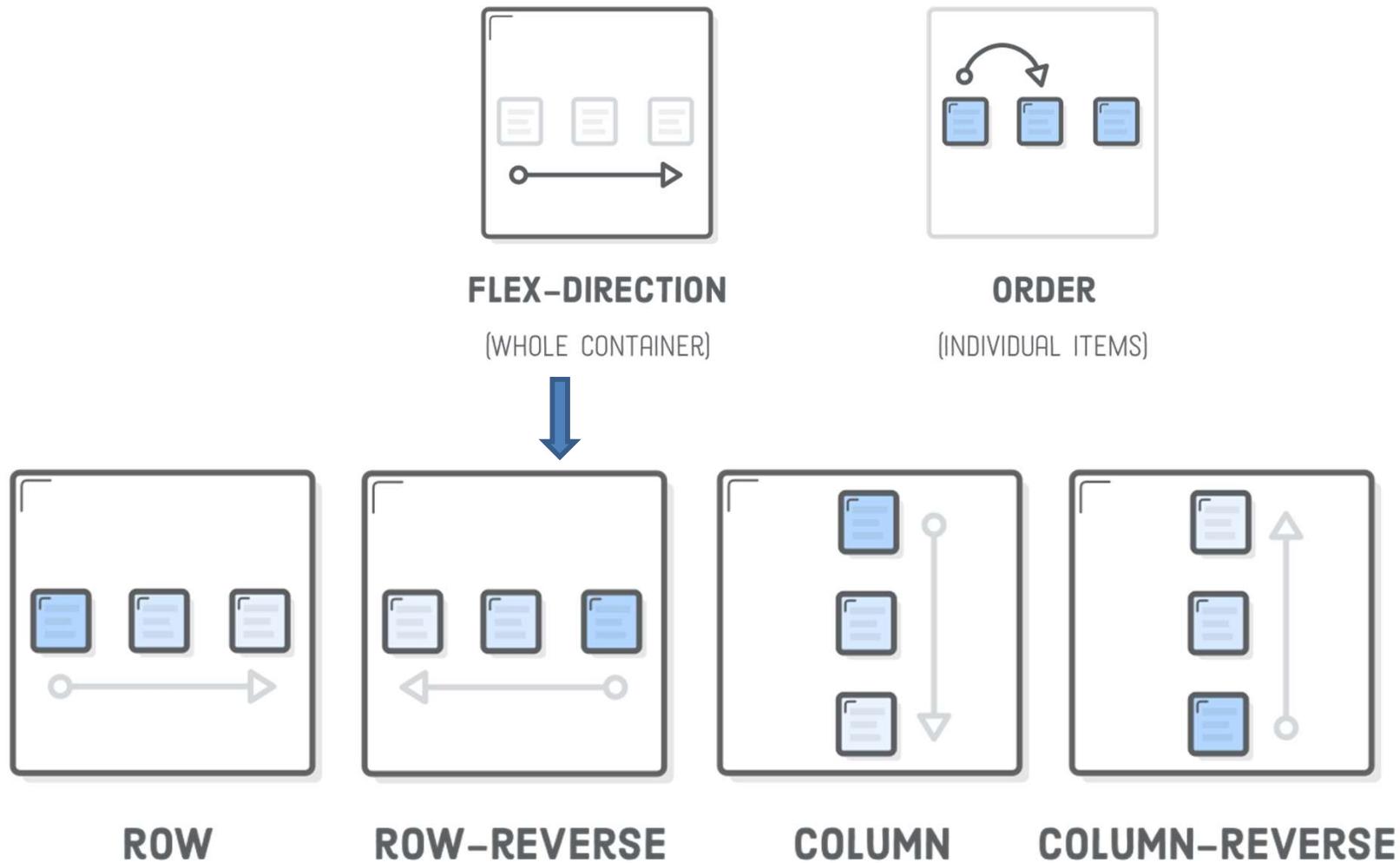


ORDER



SIZE

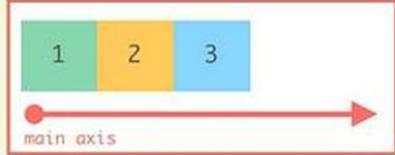
# flex container order



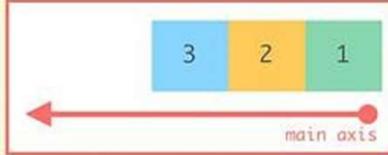
# FLEXBOX – właściwości

## flex-direction

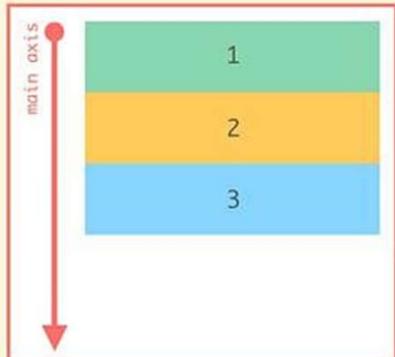
row



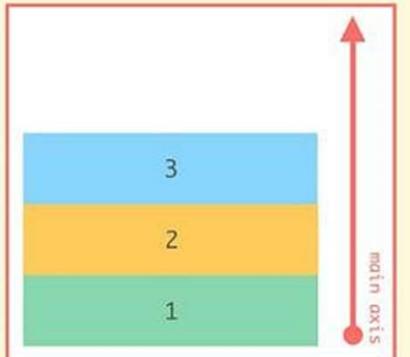
row-reverse



column

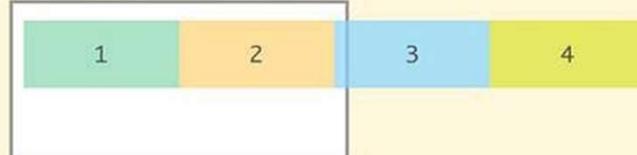


column-reverse

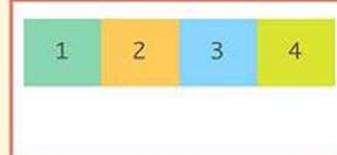


## flex-wrap

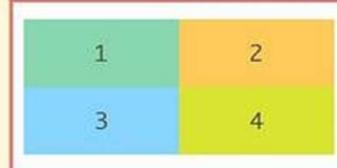
original size



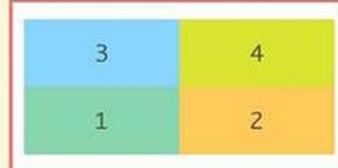
nowrap



wrap



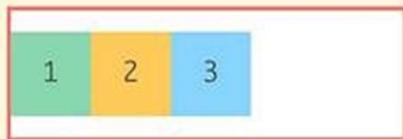
wrap-reverse



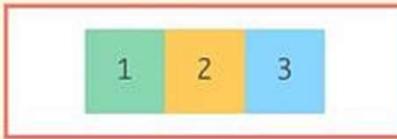
# FLEXBOX – właściwości

## justify-content

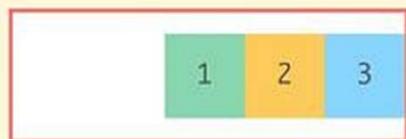
flex-start



center



flex-end



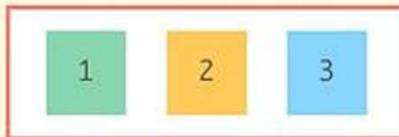
space-around



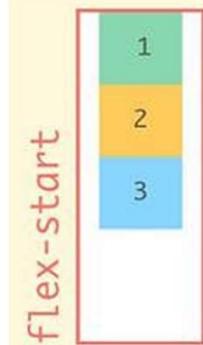
space-between



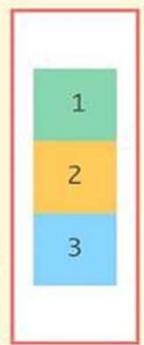
space-evenly



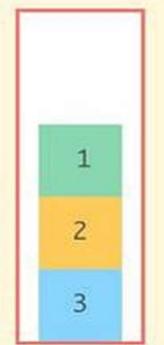
flex-start



center



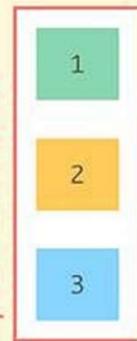
flex-end



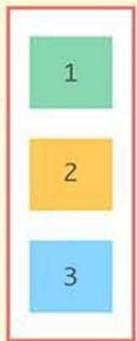
space-between



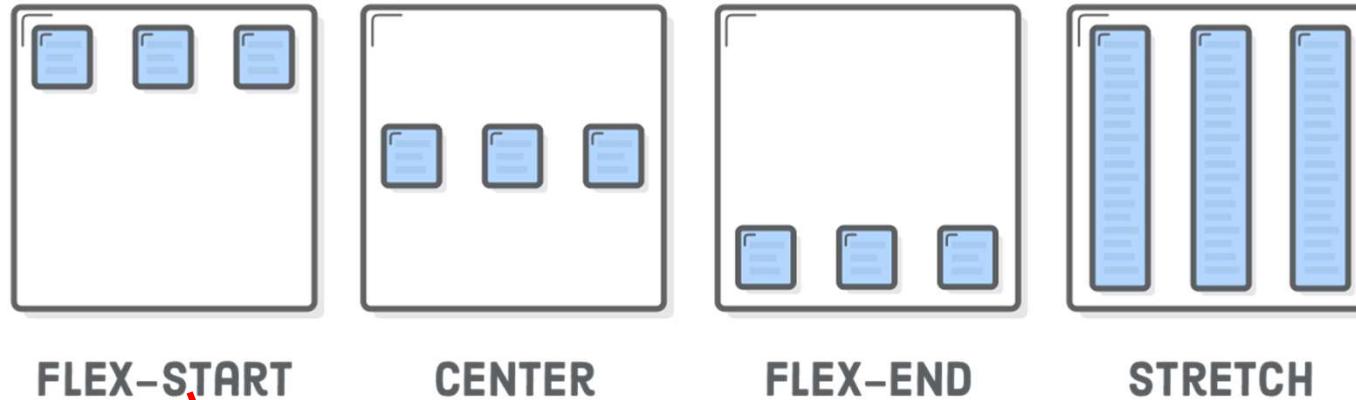
space-around



space-evenly



# aligning (vertical) a flex item



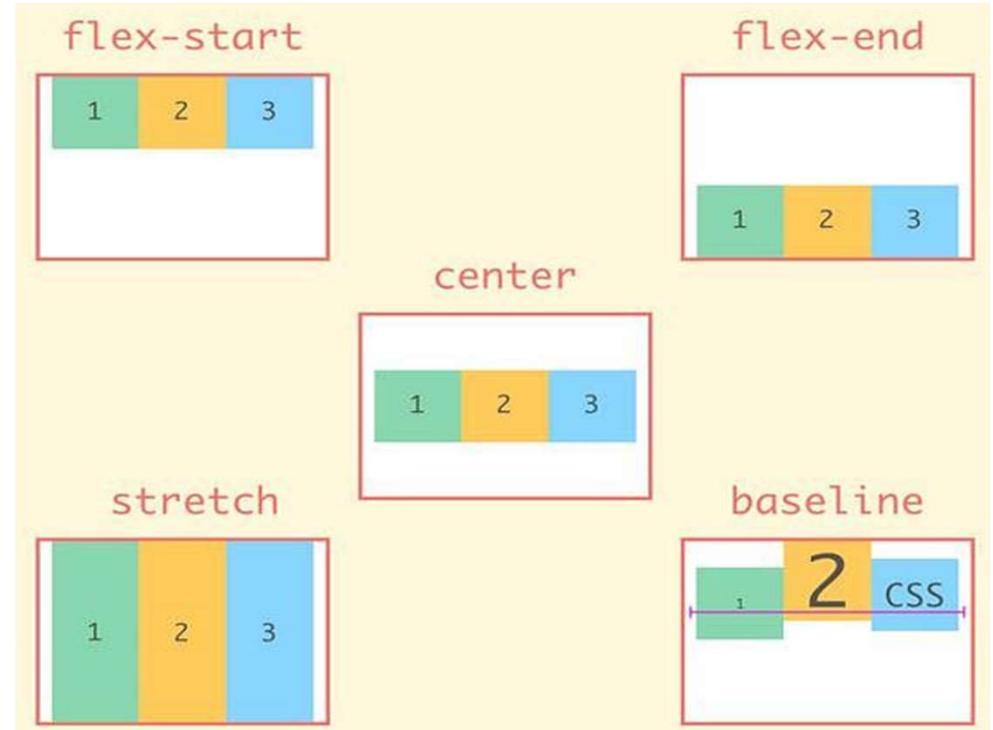
**FLEX-START**

**CENTER**

**FLEX-END**

**STRETCH**

```
#flex-container {  
display: flex;  
align-items: flex-start;  
}
```

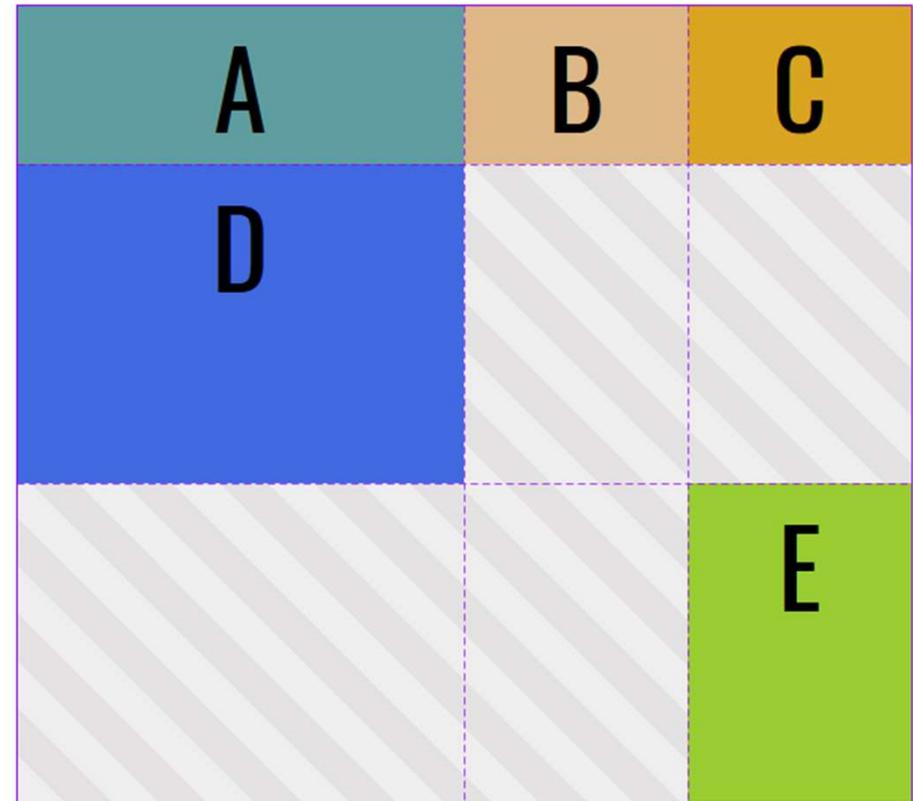


# O co chodzi z CSS Grid

Idea, która stoi za CSS Grid, to oprzeć layout o strukturę [siatki](#).

Siatka składa się z kolumn i wierszy (jest dwuwymiarowa).

Najmniejszą funkcjonalną częścią siatki jest [komórka](#).



# Grid – koncepcja i właściwości

## CONTAINER

1 grid-template-rows  
grid-template-columns  
grid-template-areas

2 grid-row-gap  
grid-column-gap

justify-items

3 justify-content

grid-auto-rows

grid-auto-columns

grid-auto-flow

grid-template

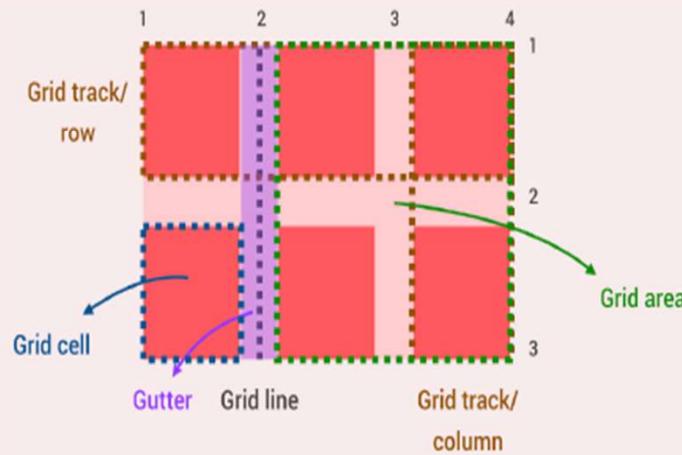
grid-gap

## ITEM

1 grid-row-start  
grid-row-end  
grid-column-start  
grid-column-end

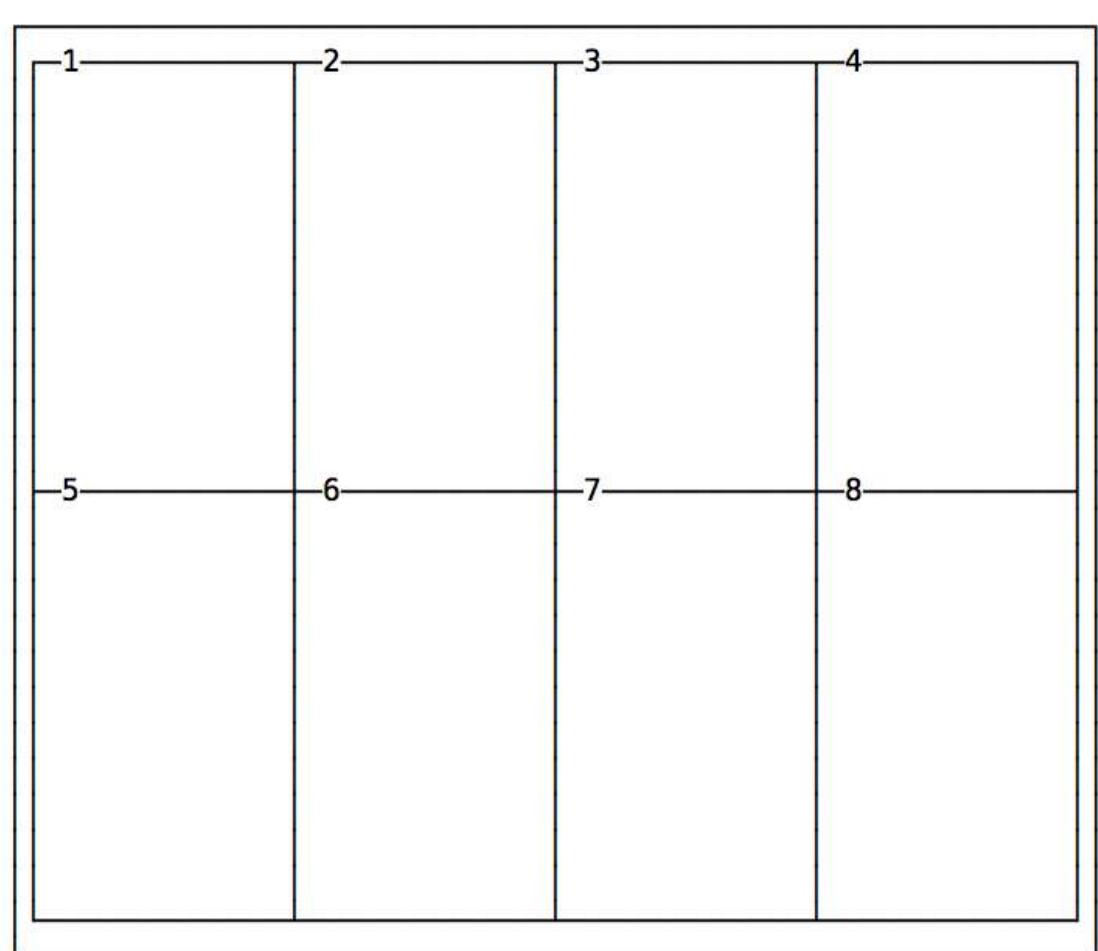
2 justify-self  
align-self

3 order



# CSS grid

```
.container {  
    display: grid;  
    grid-template-columns: 200px 200px 200px 200px;  
    grid-template-rows: 300px 300px;  
}
```

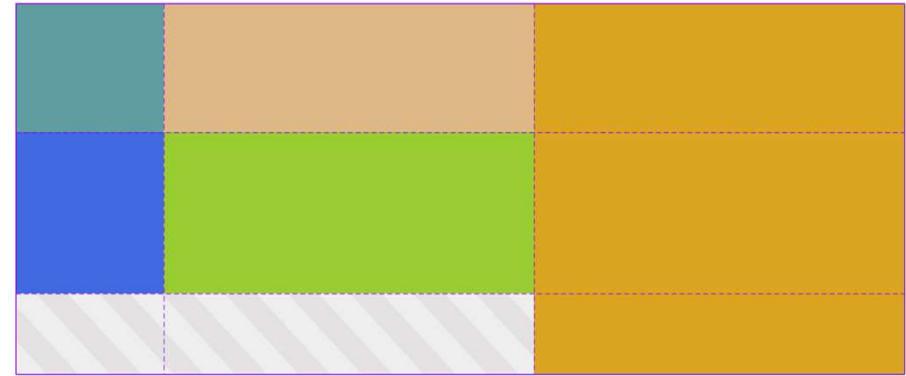


Generowanie struktury siatki  
i umieszczanie w niej elementów.

# Struktura siatki i rozmieszczenie elementów

**Sposób 1** - Jawne tworzenie struktury siatki (rozmiar i liczba wierszy i kolumn) oraz jawne umieszczanie elementów w siatce (wskazanie powierzchni siatki, która ma zajmować dany element).

**Sposób 2** - niejawne tj. automatyczne wg. reguł, które można dopasowywać. Elementy siatki są rozkładane automatycznie. W procesie rozmieszczania elementów powstaje też siatka (niejawnny sposób powstawania siatki).

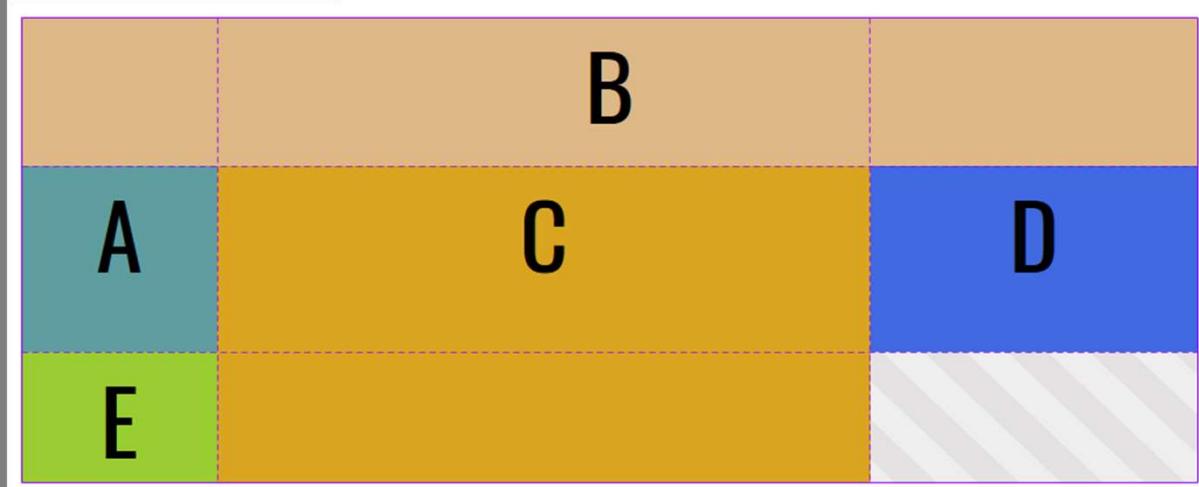


# Jawne umieszczanie części elementów

```
.grid {  
  display: grid;  
  grid-template-columns: 100px 2fr 1fr;  
  grid-template-rows: 80px 100px 70px;  
}
```

```
.e2 {  
  grid-row: 1;  
  grid-column: 1/-1  
}
```

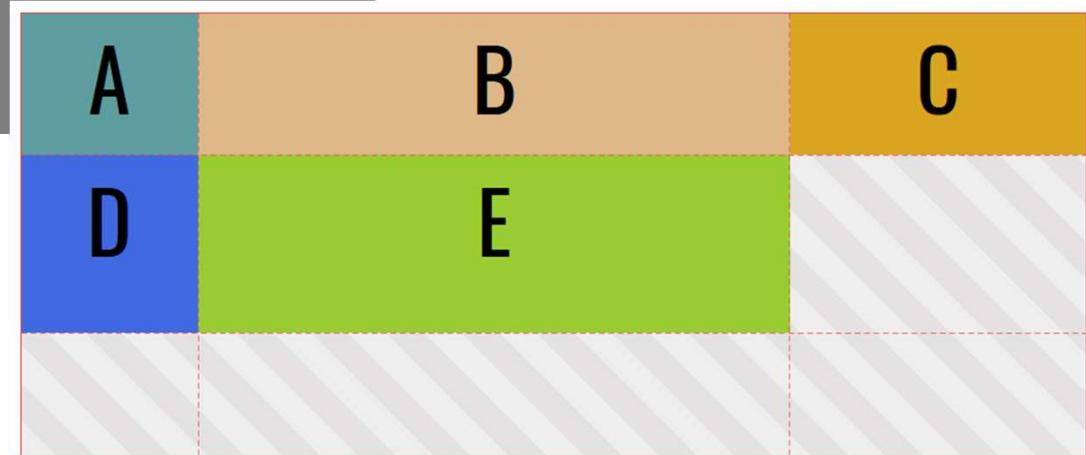
```
.e3 {  
  grid-column: 2;  
  grid-row-start: 2;  
  grid-row-end: span 2;  
}
```



Elementy z jawną deklaracją pozycji zajmują swoje miejsce. Dopiero wtedy elementy bez jawnego określenia pozycji

# Automatyczne umieszczanie elementów w istniejącej siatce

```
.grid {  
    display: grid;  
    grid-template-columns: 100px 2fr 1fr;  
    grid-template-rows: 80px 100px 70px;  
}
```



Umiejscowienie elementów nastąpiło w kolejności jak w strukturze HTML i po jeden element na komórkę - to podstawowe zasady automatycznego umieszczania elementów w

# Automatyczne umieszczanie elementów w istniejącej siatce

```
.grid {  
    display: grid;  
    grid-template-columns: 100px 200px;  
    grid-template-rows: 80px 100px;  
}
```



Brakuje miejsca w istniejącej siatce na element(y) siatki? Zostanie utworzony (domyślnie) kolejny wiersz.

**Sposób 3** - Wymuszenie struktury siatki poprzez wskazanie pozycji elementu. Wskazanie pozycja elementu (np. od 3 do 4 linii wiersza i od 2 do 4 linii kolumn) wymusi powstanie trzech wierszy i trzech kolumn (jeśli ich wcześniej nie było).



1

```
.cards {  
  display: grid;  
}  
}
```



2

```
.cards {  
  display: grid;  
  grid-template-columns: 250px 250px 250px;  
  grid-template-rows: 200px 200px 200px;  
}
```



```
.cards {  
  display: grid;  
  grid-template-columns: 250px 250px 250px;  
  grid-template-rows: 200px 200px 200px;  
  grid-gap: 20px;  
}
```

3



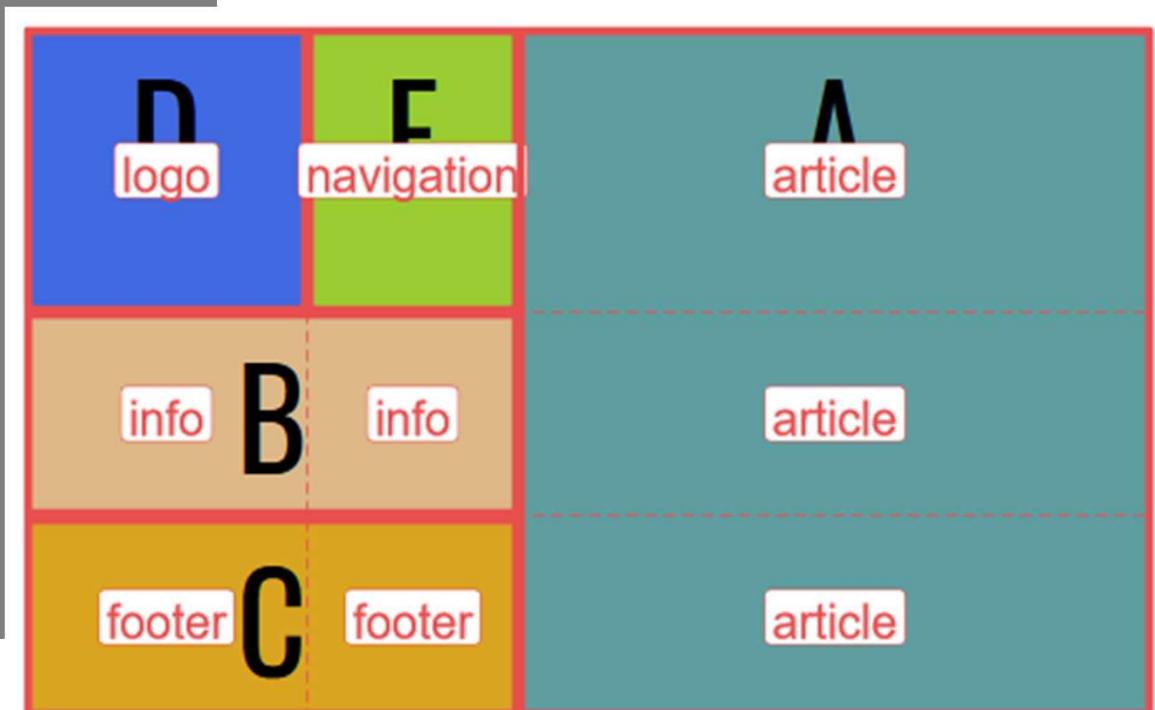
```
.cards {  
  display: grid;  
  grid-template-columns: 1fr 1fr 2fr;  
  grid-template-rows: 200px 200px 200px;  
  grid-gap: 20px;  
}
```

4



# grid-template-areas - nazywanie komórek

```
.grid {  
  display: grid;  
  grid-template-columns: 100px 1fr 3fr;  
  grid-template-rows: 100px auto auto;  
  grid-template-areas:  
    "logo navigation article"  
    "info info article"  
    "footer footer article";  
}  
  
.e1 { grid-area: article; }  
.e2 { grid-area: info; }  
.e3 { grid-area: footer; }
```



# GRID – budowa układu strony

```
.item-a {  
  grid-area: header;  
}  
  
.item-b {  
  grid-area: main;  
}  
  
.item-c {  
  grid-area: sidebar;  
}  
  
.item-d {  
  grid-area: footer;  
}  
  
.container {  
  grid-template-columns: 50px 50px 50px 50px;  
  grid-template-rows: auto;  
  grid-template-areas:  
    "header header header header"  
    "main main . sidebar"  
    "footer footer footer footer";  
}
```



```
.wrapper {  
    display:grid;  
    grid-template-columns: 1fr 3fr;  
    grid-template-rows: auto;  
    grid-gap: 20px;  
    grid-template-areas:  
        "aa aa"  
        "sidebar content"  
        "stopka stopka";  
}  
  
header {  
    background-color: red;  
    grid-area: aa;  
    padding: 20px;}  
  
article {  
    background-color: blue;  
    grid-area: content;  
    padding: 20px;}  
  
aside {  
    background-color: yellow;  
    grid-area: sidebar; }  
  
footer {  
    background-color: green;  
    grid-area: stopka; }
```

```
<div class="wrapper">  
    <header>  
        To jest naglowek  
    </header>  
    <article>  
        <h1> Przyklad szablonu typu grid </h1>  
        <p> Zawartosc strony glownej</p>  
    </article>  
    <aside>  
        <ul> Lista reklam  
            <li> reklama 1 </li> <li> reklama 2 </li> <li> reklama 3 </li>  
        </ul>  
    </aside>  
    <footer> To jest stopka create by GR - WdAI</footer>  
</div>
```



# CSS3 – co nowego

- **CSS3 Borders - Cienie i Zaokrąglenia**
- **CSS3 Text Effects**
- **CSS3 Backgrounds**
- **CSS3 Fonts**
- **CSS3 2D Transforms**
- **CSS3 3D Transforms**
- **CSS3 Transitions**
- **CSS3 Animations**
- **CSS3 Układ elastyczny**
- **CSS3 Zaawansowane filtry**

# CSS3 Transformacje 2D i 3D

## Transformacje 2D

translate()

rotate()

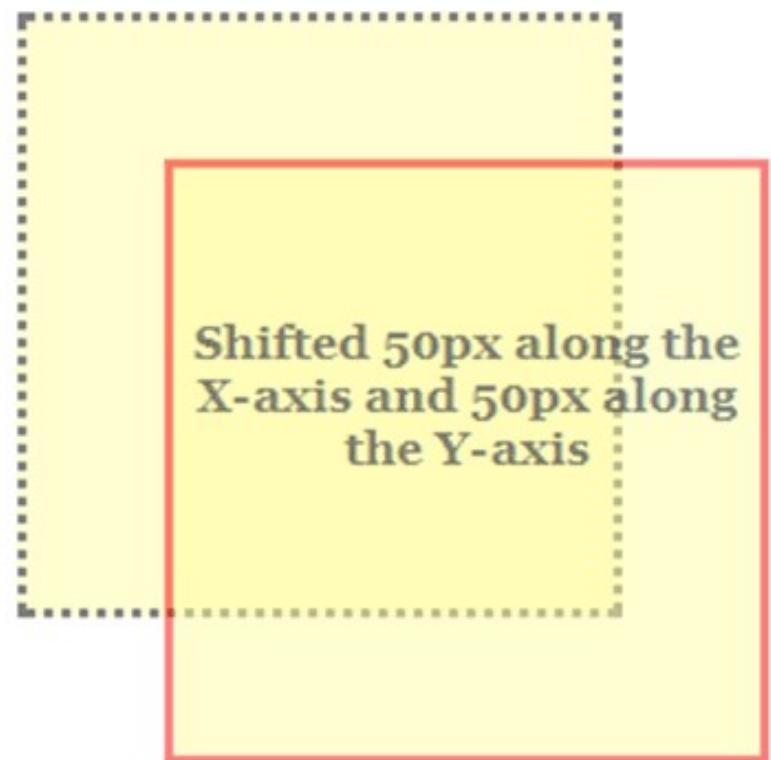
scale()

skew()

matrix()

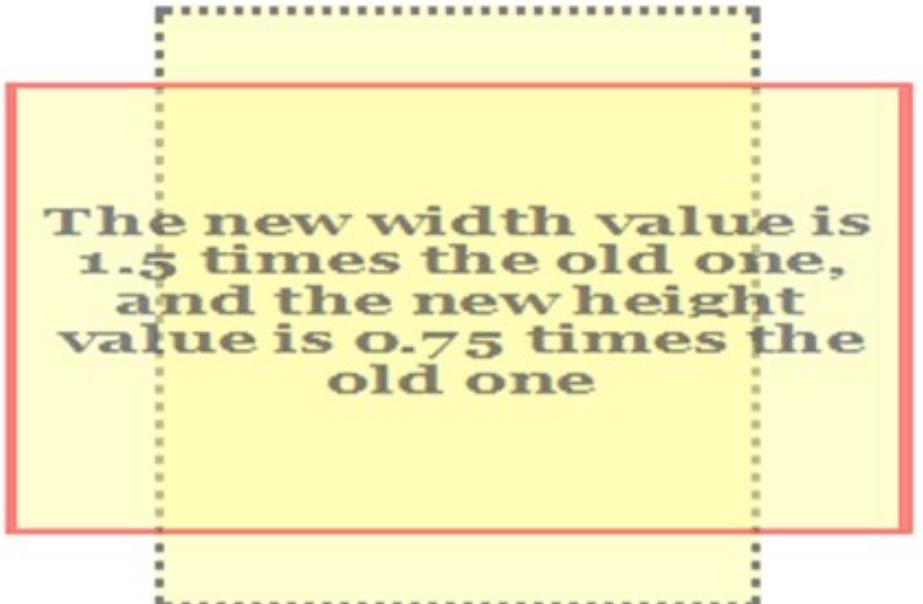
# Translate

```
#box3  
{  
transform:translate(50px,50px);  
-ms-transform:translate(50px,50px); /* IE */  
-moz-transform:translate(50px,50px); /* Firefox */  
-webkit-transform:translate(50px,50px); /* Safari & Chrome */  
-o-transform:translate(50px,50px); /* Opera */  
}
```



# Scale

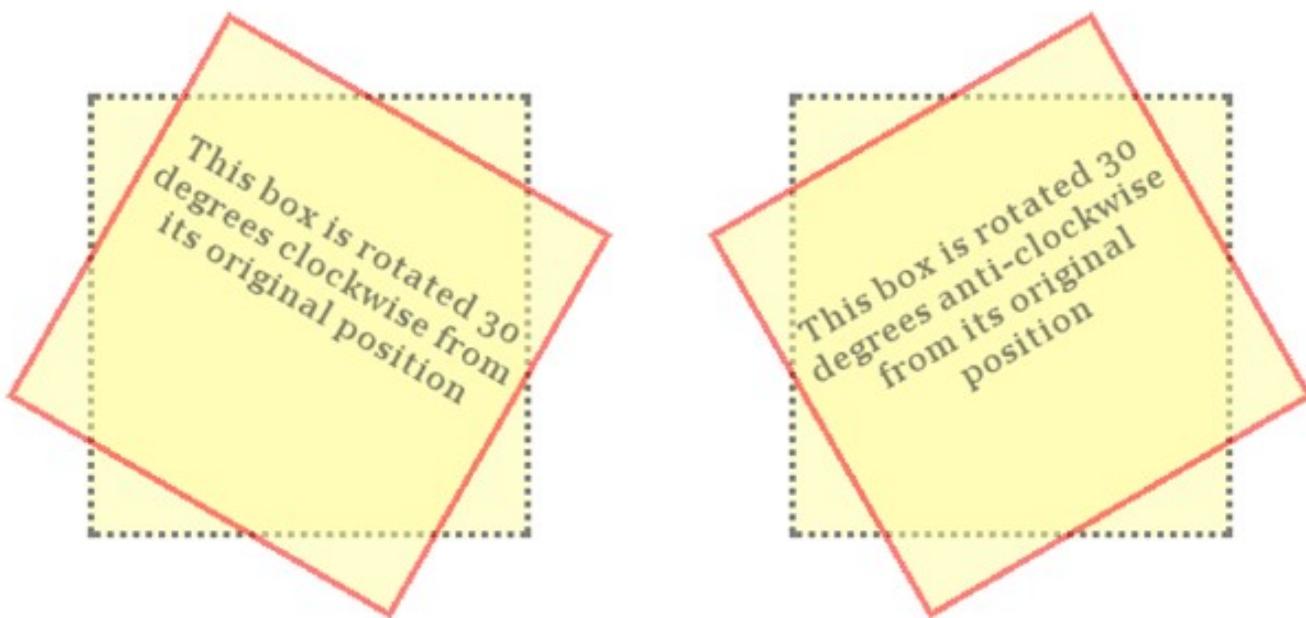
```
#box3
{
    transform:scale(1.5,0.75);
    -ms-transform:scale(1.5,0.75); /* IE */
    -moz-transform:scale(1.5,0.75); /* Firefox */
    -webkit-transform:scale(1.5,0.75); /* Safari and Chrome */
    -o-transform:scale(1.5,0.75); /* Opera */
}
```



The new width value is  
1.5 times the old one,  
and the new height  
value is 0.75 times the  
old one

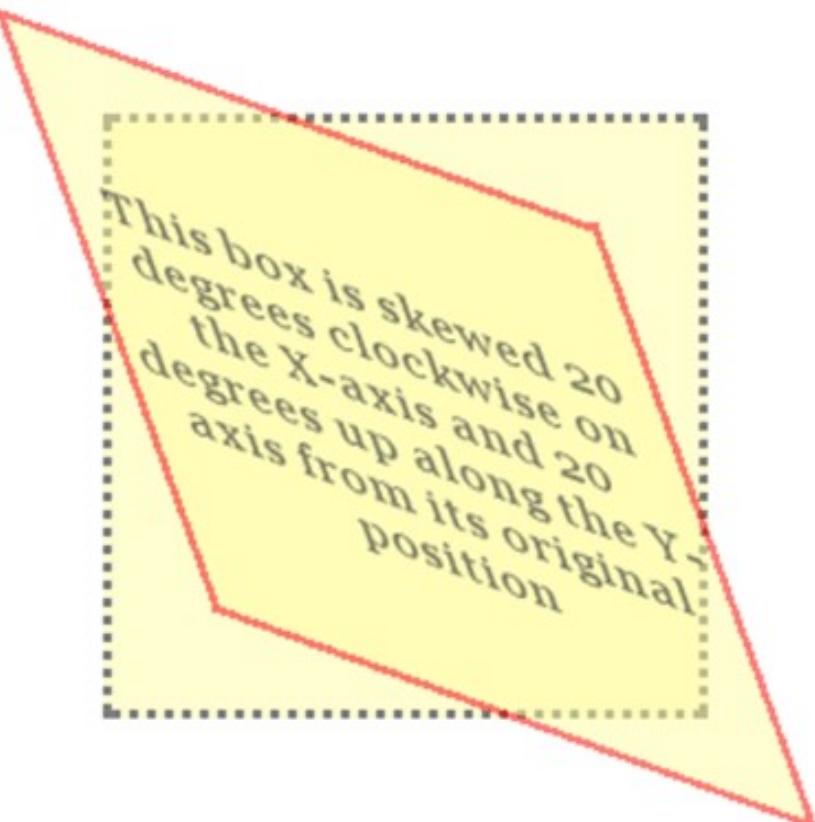
# Rotation

```
#box1
{
transform:rotate(30deg);
-ms-transform:rotate(30deg); /* IE */
-moz-transform:rotate(30deg); /* Firefox */
-webkit-transform:rotate(30deg); /* Safari and Chrome */
-o-transform:rotate(30deg);/* Opera */
}
#box2
{
transform:rotate(-30deg);
-ms-transform:rotate(-30deg); /* IE */
-moz-transform:rotate(-30deg); /* Firefox */
-webkit-transform:rotate(-30deg); /* Safari and Chrome */
-o-transform:rotate(-30deg);/* Opera */
}
```



# Skew

```
#box
{
    transform:skew(20deg,20deg);
    -ms-transform:skew(20deg,20deg); /* IE */
    -moz-transform:skew(20deg,20deg); /* Firefox */
    -webkit-transform:skew(20deg,20deg); /* Safari and Chrome */
    -o-transform:skew(20deg,20deg);/* Opera */
}
```



# CSS3 3D Transforms

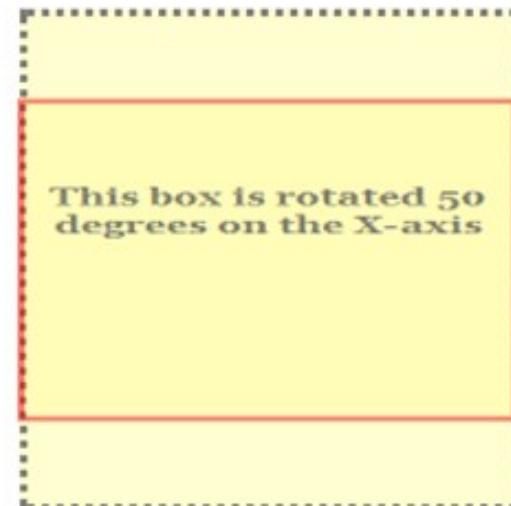
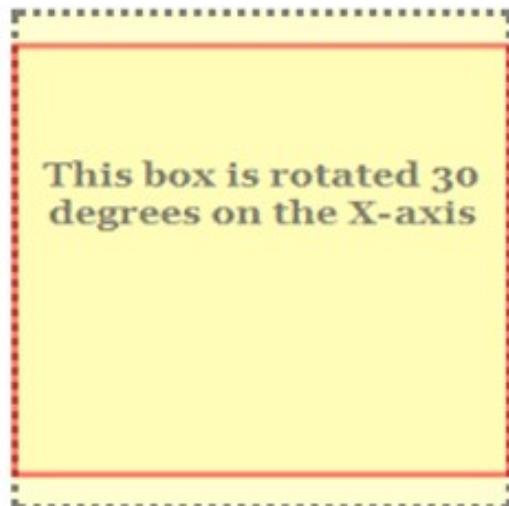
## TranslateZ

```
#box1  
{ transform:translateZ(10px);  
-ms-transform:translateZ(10px); /* IE */  
-moz-transform:translateZ(10px); /* Firefox */  
-webkit-transform:translateZ(10px); /* Safari and Chrome */  
-o-transform:translateZ(10px);/* Opera */  
}  
  
#box2  
{ transform:translateZ(-10px);  
-ms-transform:translateZ(-10px); /* IE */  
-moz-transform:translateZ(-10px); /* Firefox */  
-webkit-transform:translateZ(-10px); /* Safari and Chrome */  
-o-transform:translateZ(-10px);/* Opera */  
}
```



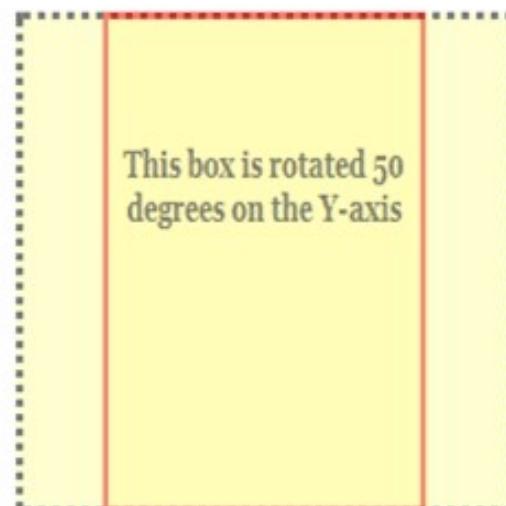
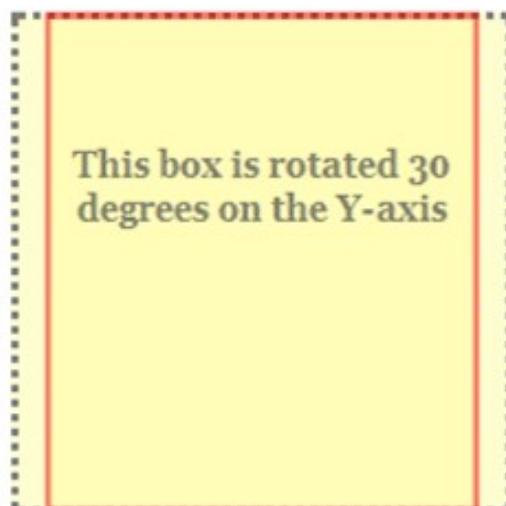
## RotateX

```
#box1
{ transform:rotateX(30deg);
-ms-transform:rotateX(30deg); /* IE */
-moz-transform:rotateX(30deg); /* Firefox */
-webkit-transform:rotateX(30deg); /* Safari and Chrome */
-o-transform:rotateX(30deg); /* Opera */
}
#box2
{ transform:rotateX(50deg);
-ms-transform:rotateX(50deg); /* IE */
-moz-transform:rotateX(50deg); /* Firefox */
-webkit-transform:rotateX(50deg); /* Safari and Chrome */
-o-transform:rotateX(50deg); /* Opera */
}
```



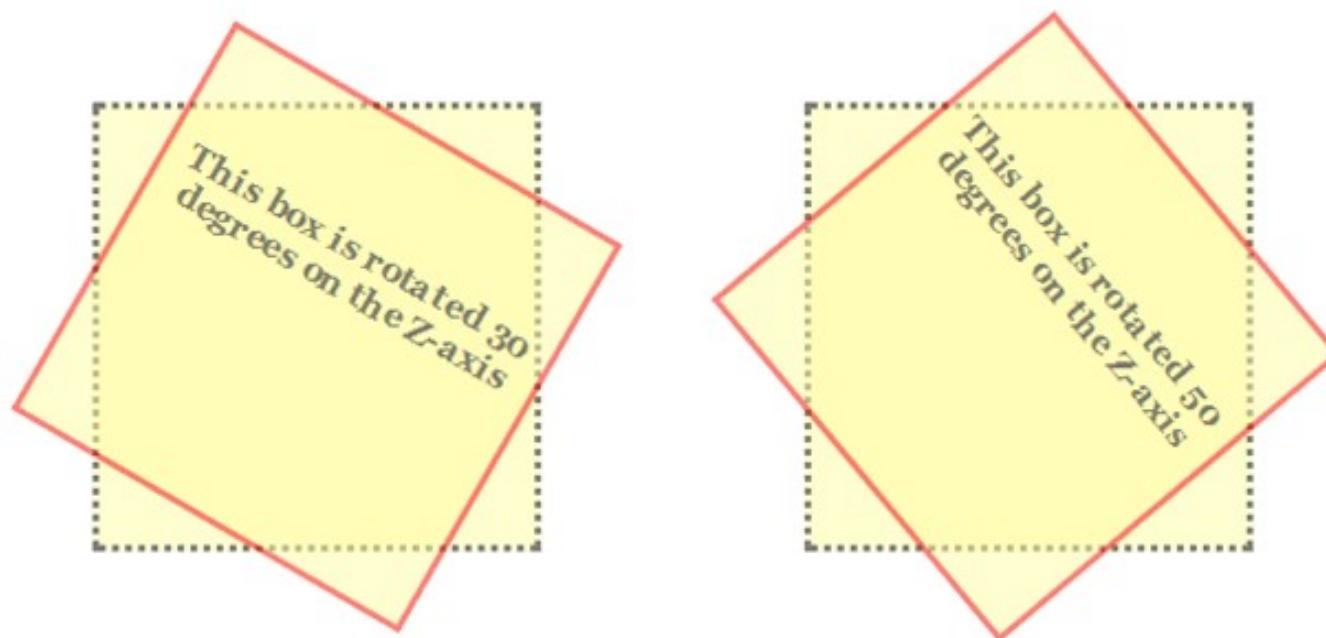
## RotateY

```
#box1  
{ transform:rotateY(30deg);  
-ms-transform:rotateY(30deg); /* IE */  
-moz-transform:rotateY(30deg); /* Firefox */  
-webkit-transform:rotateY(30deg); /* Safari and Chrome */  
-o-transform:rotateY(30deg); /* Opera */  
}  
  
#box2  
{ transform:rotateY(50deg);  
-ms-transform:rotateY(50deg); /* IE */  
-moz-transform:rotateY(50deg); /* Firefox */  
-webkit-transform:rotateY(50deg); /* Safari and Chrome */  
-o-transform:rotateY(50deg); /* Opera */  
}
```



## RotateZ

```
#box1
{
  transform:rotateZ(30deg);
  -ms-transform:rotateZ(30deg); /* IE */
  -moz-transform:rotateZ(30deg); /* Firefox */
  -webkit-transform:rotateZ(30deg); /* Safari and Chrome */
  -o-transform:rotateZ(30deg); /* Opera */
}
#box2
{
  transform:rotateZ(50deg);
  -ms-transform:rotateZ(50deg); /* IE */
  -moz-transform:rotateZ(50deg); /* Firefox */
  -webkit-transform:rotateZ(50deg); /* Safari and Chrome */
  -o-transform:rotateZ(50deg); /* Opera */
}
```



# Multiple Transforms dla jednego elementu

```
<style type="text/css">

#submenu {
    background-color: #eeee;
    -webkit-transition: all 1s ease-in-out;
    -moz-transition: all 1s ease-in-out;
    -o-transition: all 1s ease-in-out;
    -ms-transition: all 1s ease-in-out;
    transition: all 1s ease-in-out;
}

#submenu:hover {
    background-color: #fc3;
    -webkit-transform: rotate(360deg) scale(2);
    -moz-transform: rotate(360deg) scale(2);
    -o-transform: rotate(360deg) scale(2);
    -ms-transform: rotate(360deg) scale(2);
    transform: rotate(360deg) scale(2);
}

</style>
```

# CSS3 Transitions

## Transitions (Przemiana)

Definiuje przemianę (przejście) obiektu

Użycie: **transition: własność czas-trwania rozkład-w-czasie opóźnienie.**

Algorytm użycia transition w CSS:

1. Zdefiniować dla obiektu styl zwykły
2. Zdeklarować końcowy stan dla obiektu np., dla hover
3. Do stanu zwykłego dodać funkcję transition przy użyciu własności : transition-property, transition-duration, transition-timing-function, and transition-delay.

lista właściwości jakie mogą być zmieniane

- background-color and background-position ■ border-color, border-spacing, and border-width
- bottom, top, left, and right ■ clip ■ color ■ crop ■ font-size and font-weight ■ height and width
- letter-spacing ■ line-height ■ margin ■ max-height, max-width, min-height, and min-width
- opacity ■ outline-color, outline-offset, and outline-width ■ padding ■ text-indent ■ text-shadow
- vertical-align ■ visibility ■ word-spacing ■ z-index

Dla leniwych : <http://www.css3-generator.de/transform.html>

↓ co ma być animowane,  
można wpisać "all" dla wszystkich właściwości

**transition: 5s height ease-out**

czas trwania animacji



czas opóźnienia animacji  
sposób animowania,  
można pominięć wtedy  
przyjmowany jest "linear"

**transition: 1s 2s opacity**

↑ czas opóźnienia animacji

- do jednego elementu można zastosować kilka animacji  
(wartości kolejnych dajemy po przecinku)
- brakuje najpopularniejszej animacji - z `display:none` do `display:block`  
przy zmianie wysokości lub przezroczystości

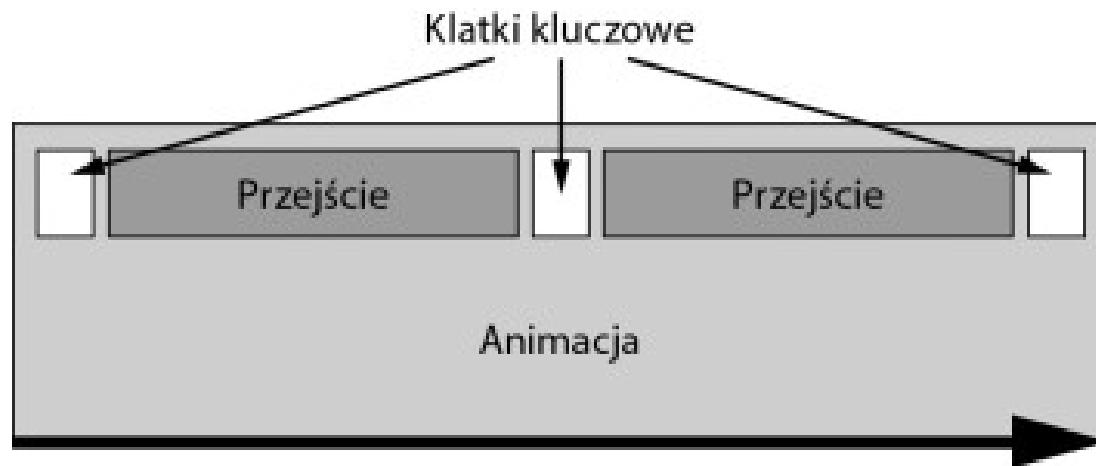
<http://www.css3.info/preview/css3-transitions/> - przykłady i omówienie

# CSS3 Animation

Animacja składa się z dwóch bloków kodu:

Pierwszy blok zawiera definicję samej animacji przypisanej do obiektu, określając jej:

**nazwę, czas trwania, rozkład czasowy, opóźnienie startu, liczbę powtórzeń i kierunek.**



Drugi blok, czyli @keyframes, zawiera selektory poszczególnych klatek animacji obiektu oraz przypisane poszczególnym klatkom style.