

# **PRZEDMIOT: Tworzenie stron i aplikacji internetowych**

**KLASA: 5i gr. 2**

## **Tydzien 1 Lekcja 1**

### **Temat: Znaczniki semantyczne w HTML 5**

**W HTML5 znaczniki semantyczne służą do opisywania struktury i znaczenia treści na stronie w sposób bardziej czytelny dla przeglądarek, wyszukiwarek i programistów.**

Lista najczęściej używanych znaczników semantycznych w HTML5:

1. **<header>** – **Definiuje nagłówek strony**, sekcji lub artykułu, zazwyczaj zawierający logo, menu nawigacyjne lub tytuły.
2. **<nav>** – **Określa sekcję nawigacyjną**, zawierającą linki do innych stron lub części dokumentu.
3. **<main>** – **Reprezentuje główną treść dokumentu**, unikalną dla danej strony (powinna występować tylko raz).
4. **<article>** – **Oznacza niezależną, samodzielnią treść**, taką jak wpis na blogu, artykuł czy post.
5. **<section>** – **Grupuje powiązane tematycznie treści**, zwykle z nagłówkiem (np. `<h2>`).
6. **<aside>** – **Zawiera treści poboczne**, takie jak panele boczne, reklamy czy dodatkowe informacje.
7. **<footer>** – **Definiuje stopkę strony** lub sekcji, zawierającą np. informacje kontaktowe, prawa autorskie.
8. **<figure>** – **Służy do grupowania multimedialnych elementów** (np. obrazów, diagramów) z opcjonalnym podpisem.
9. **<figcaption>** – **Podpis dla elementu <figure>, opisujący zawartość multimedialną**.

10. **<details>** – Tworzy interaktywny element, który można rozwinąć/zwinąć, aby pokazać dodatkowe informacje.
11. **<summary>** – Definiuje nagłówek dla elementu **<details>**, widoczny przed rozwinięciem.
12. **<mark>** – Wyróżnia tekst, który jest istotny w danym kontekście (np. wyniki wyszukiwania).
13. **<time>** – Oznacza datę, godzinę lub zakres czasowy, z opcjonalnym atrybutem `datetime`.
14. **<address>** – Służy do oznaczania informacji kontaktowych, np. adresu e-mail, telefonu czy lokalizacji.
15. **<progress>** – Reprezentuje pasek postępu, np. dla ładowania lub wypełnienia formularza.
16. **<meter>** – Wskazuje wartość w określonym zakresie, np. poziom naładowania baterii.
17. **<dialog>** – Definiuje okno dialogowe lub modalne, np. do wyświetlania alertów.
18. **<picture>** – Umożliwia definiowanie różnych źródeł obrazów dla różnych urządzeń lub rozdzielczości.
19. **<template>** – Przechowuje treść, która nie jest wyświetlana od razu, ale może być użyta przez JavaScript.

Znaczniki pomagają w lepszej organizacji kodu, poprawiają dostępność (accessibility) i optymalizację dla wyszukiwarek (SEO).

## Tydzien 1 Lekcja 2

**Temat:** Implementacja połączenia do bazy danych MySQL w PHP (w sposób obiektowy i proceduralny)

- ◆ **1. Obiektowy sposób (OOP - Object-Oriented Programming, czyli programowanie obiektowe)**

```
$mysqli = new  
mysqli("localhost", "my_user", "my_password", "my_db");  
  
if ($mysqli -> connect_errno) {  
    echo "Błąd połączenia: " . $mysqli -> connect_error;  
    exit();  
}
```

- ◆ **2. Proceduralny sposób**

```
$con =  
mysqli_connect("localhost", "my_user", "my_password", "my_db");  
  
if (mysqli_connect_errno()) {  
    echo "Błąd połączenia: " . mysqli_connect_error();  
    exit();  
}
```

# Różnice

## 1. Styl programowania

- OOP (`new mysqli`) – bardziej nowoczesny, wspiera podejście obiektowe, lepiej integruje się np. z frameworkami (Laravel, Symfony).
- Proceduralny (`mysqli_connect`) – starszy styl, przypomina stary `mysql_connect` (już usunięty).

## 2. Czytelność i rozszerzalność

- OOP daje możliwość używania metod (`$mysqli->query()`, `$stmt->bind_param()`), co sprawia, że kod jest bardziej spójny.
- Proceduralny miesza funkcje globalne z innymi elementami, więc w większych projektach kod może być mniej czytelny.

## 3. Wydajność

- **Oba działają tak samo szybko** – pod spodem to ta sama biblioteka `mysqli`.

## 4. Dobre praktyki

- Jeśli implementujesz **mały skrypt** (np. test, coś jednorazowego) – proceduralny jest szybszy do napisania.
- Jeśli implementujesz **większą aplikację** – zdecydowanie lepiej trzymać się OOP (`new mysqli`), bo jest bardziej przejrzysty i łatwiej go łączyć z obiektowym stylem kodu.

### Podsumowanie:

- Oba sposoby są poprawne i tak samo szybkie.

- **Lepszy wybór:** obiektowy (`new mysqli`), bo jest nowocześniejszy, bardziej czytelny i łatwiej rozszerzalny w większych projektach.

## Tydzien 3 Lekcja 1

### Temat: Typy danych w PHP. Konwersja typów.

W PHP typy danych można podzielić na kilka kategorii, w tym typy

#### **skalarne,**

- **Integer (liczba całkowita)** Reprezentuje liczby całkowite (bez części ułamkowej), np. -5, 0, 42. Przykład: `$int = 123;`. Zakres zależy od platformy (zazwyczaj 32- lub 64-bitowy, np. od  $-2^{31}$  do  $2^{31}-1$  na 32-bitowych systemach).
- **Float (liczba zmiennoprzecinkowa, zwana też double)** Reprezentuje liczby z częścią ułamkową, np. 3.14, -0.001. Przykład: `$float = 3.14;`. Uwaga: Precyza jest ograniczona, co może prowadzić do błędów zaokrągleń w operacjach arytmetycznych.
- **String (ciąg znaków)** Przechowuje sekwencję znaków, np. "Witaj", 'Świat!'. Przykład: `$string = "Hello World";`. Może być zapisany w cudzysłowach ("") lub apostrofach ('), przy czym cudzysłowy pozwalają na parsowanie zmiennych wewnętrz ciągu (np. "Witaj, \$name").
- **Boolean (wartość logiczna)** Przechowuje wartości true lub false. Przykład: `$bool = true;`. Używany w wyrażeniach logicznych i warunkach.

#### **złożone,**

- **Array (tablica)** Przechowuje zestaw danych w formie klucz-wartość. Może być indeksowana (liczbowe klucze) lub asocjacyjna (dowolne klucze).

Przykład:

```
$array = [1, 2, 3]; // Indeksowana  
$assoc = ['name' => 'Jan', 'age' => 30]; // Asocjacyjna
```

- **Object (obiekt)** Reprezentuje instancję klasy z właściwościami i metodami.

Przykład:

```
class Person {  
    public $name = "Jan";  
}  
  
$obj = new Person();
```

- **Callable (wywoływalny)** Reprezentuje coś, co może być wywołane jako funkcja, np. funkcje, metody lub domknięcia (closures).

Przykład:

```
$callback = function($x) { return $x * 2; };  
  
echo $callback(5); // Wypisze 10
```

- **Iterable (iterowalny)** Typ wprowadzony w PHP 7.1, reprezentuje dane, które można iterować (np. tablice, obiekty implementujące interfejs Iterator).

Przykład:

```
function process(iterable $items) {  
    foreach ($items as $item) {  
        echo $item;  
    }  
}
```

## □ specjalne

- **NULL** Reprezentuje brak wartości lub zmienną bez przypisanej wartości. Przykład: `$var = null;`. Uwaga: NULL jest nieczułe na wielkość liter (`null = NULL`).

- **Resource (zasób)** Specjalny typ dla uchwytów do zasobów zewnętrznych, np. połączeń z bazą danych, otwartych plików. Przykład: `$file = fopen('example.txt', 'r');`. Uwaga: W nowszych wersjach PHP zasoby są coraz rzadziej używane na rzecz obiektów.

## **pseudotypy**

Pseudotypy to bardziej wskazówki w dokumentacji niż rzeczywiste typy danych, ale są istotne w kontekście typowania.

- **Mixed** Oznacza, że zmienna może mieć dowolny typ. Używane w deklaracjach funkcji, gdy typ nie jest ścisłe określony.

Przykład:

```
function doSomething(mixed $value) {  
    return $value;  
}
```

- **Void** Używane w deklaracjach funkcji, które nic nie zwracają.

Przykład:

```
function logMessage(string $msg): void {  
    echo $msg;  
}
```

- **Never** (od PHP 8.1) Oznacza, że funkcja nigdy nie zwraca wartości (np. rzuca wyjątek lub kończy skrypt).

Przykład:

```
function throwError(): never {  
    throw new Exception("Błąd!");  
}
```

- **Union Types** (od PHP 8.0) Pozwalają określić, że zmienna może mieć jeden z kilku typów, np. int|float.

Przykład:

```
function add(int|float $a, int|float $b): int|float {  
    return $a + $b;  
}
```

- **Intersection Types** (od PHP 8.1) Wymagają, aby wartość była zgodna ze wszystkimi określonymi typami (używane głównie z obiektami).

Przykład:

```
function process(Countable&Traversable $obj) {  
    // ...  
}
```

**PHP jest językiem dynamicznie typowanym, co oznacza, że typy zmiennych są określone w czasie wykonywania i mogą się zmieniać w zależności od przypisanych wartości.**

## Tydzien 3 Lekcja 1

**Temat:** Definiowanie kolorów w CSS

# Sposoby definiowania kolorów w CSS

1. W notacji słownej
2. w notacji RGB (Red Blue Green) i RGBA (Red Blue Green Alpha)
3. w notacji szesnastkowej (heksadecymalna)
4. w notacji HSL/HSLA
5. w notacji CMYK (Cyan Magenta Yellow Key)
6. w formacie dziesiętnym (rzadko używany)
7. Funkcja currentColor
8. Zmienne CSS (Custom Properties)
9. Funkcje kolorystyczne (nowoczesne, np. color())
10. Gradienty (Linearne i Radialne)

## 1. Notacja słowna (nazwy kolorów)

Kolory można określić za pomocą **predefiniowanych nazw w języku angielskim**, np. `red`, `blue`, `green`, `white`, `black`. CSS obsługuje około 140 nazw kolorów (np. tomato, aliceblue).

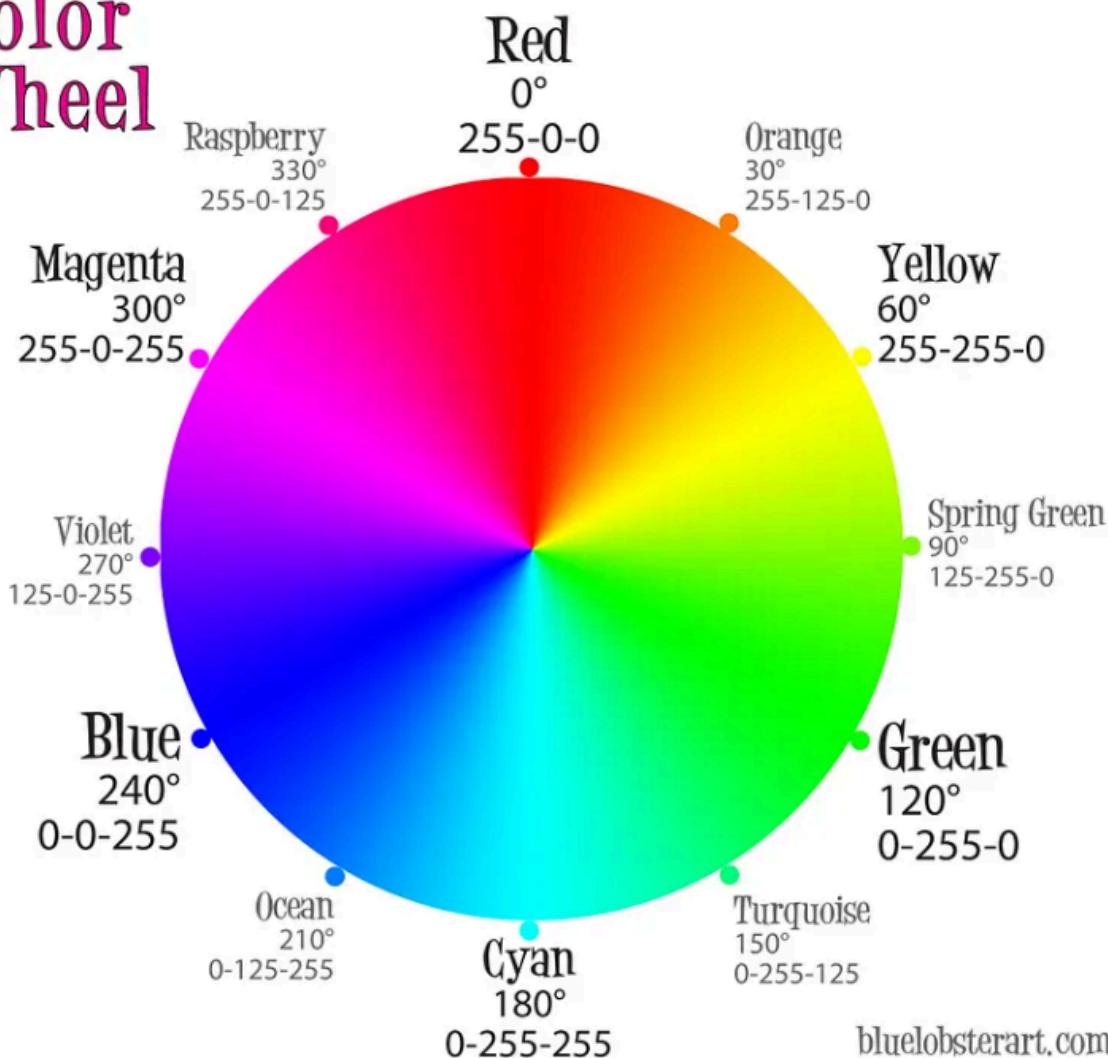
- **Przykład:** `color: red;`
- **Zalety:** Proste, czytelne.
- **Wady:** Ograniczona liczba kolorów, brak precyzji.

## 2. Notacja RGB i RGBA

Kolory definiowane są przez wartości składowych **czerwieni (Red), zieleni (Green) i niebieskiego (Blue)**, każda w zakresie 0–255. RGBA dodaje kanał alfa (**przezroczystość, 0–1**).

- **RGB:** `rgb(255, 0, 0)` – czerwony.
- **RGBA:** `rgba(255, 0, 0, 0.5)` – czerwony z 50% przezroczystością.
- **Przykład:** `rgb(0, 0, 0);` – czarny, `rgb(255, 255, 255);` – biały,
- **Zalety:** Intuicyjne, szeroki zakres kolorów.
- **Wady:** Mniej zwięzłe niż notacja szesnastkowa.

# RGB Color Wheel



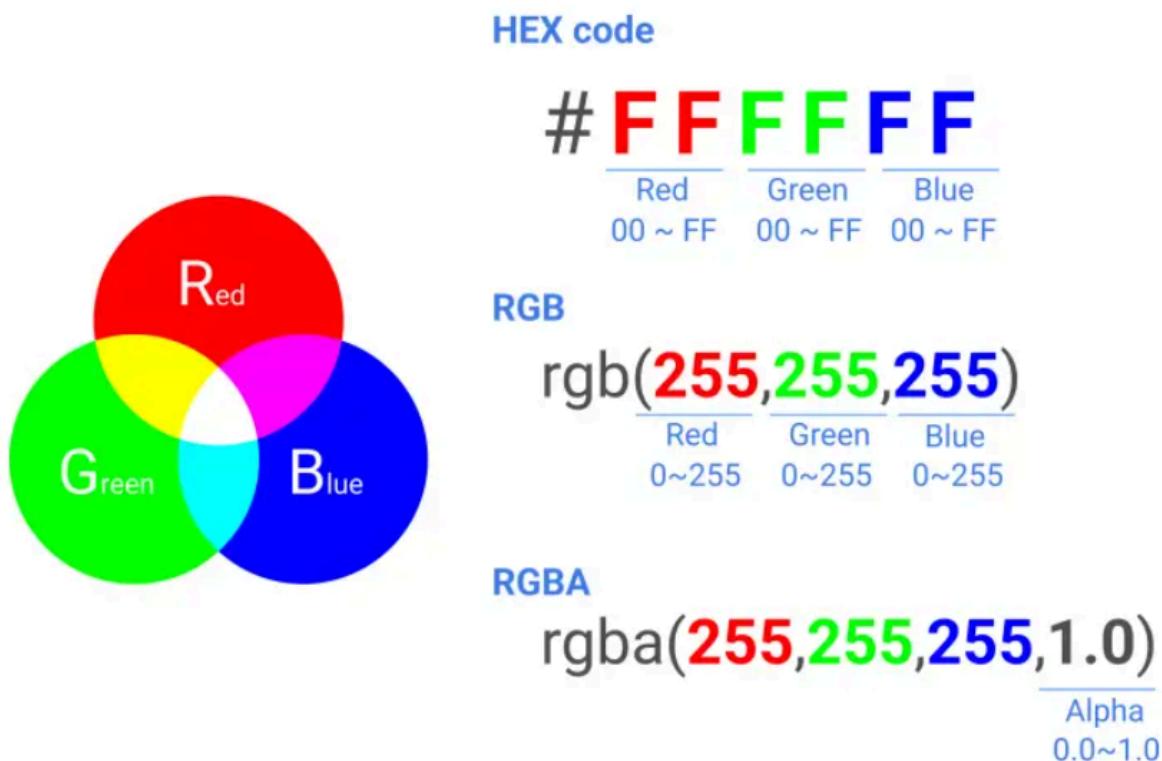
### 3. Notacja szesnastkowa (heksadecymalna)

Kolory zapisuje się w formacie **#RRGGBB**, gdzie **RR, GG, BB** to wartości **heksadecymalne (00–FF) dla czerwieni, zieleni i niebieskiego**. Można dodać dwa znaki dla alfa (**#RRGGBBAA**).

System szesnastkowy używa cyfr 0–9 oraz liter A–F.

- Cyfry:** 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- Litery:** A, B, C, D, E, F

- **Przykład:** #FF0000 (czerwony), #FF000080 (czerwony z 50% przezroczystością).
- **Skrót:** #F00 (odpowiednik #FF0000).
- **Zalety:** Zwięzła, powszechnie używana.
- **Wady:** Mniej czytelna dla początkujących.



#### 4. Notacja HSL i HSLA

Kolory definiowane przez **odcień (Hue, 0–360°)**, **nasycenie (Saturation, 0–100%)** i **jasność (Lightness, 0–100%)**. HSLA dodaje kanał alfa.

- **HSL:** hsl(0, 100%, 50%) – czerwony.
- **HSLA:** hsla(0, 100%, 50%, 0.5) – czerwony z 50% przezroczystością.
- **Przykład:** color: hsl(120, 60%, 70%);
- **Zalety:** Intuicyjne manipulowanie odcieniem i jasnością.
- **Wady:** Mniejsza popularność niż RGB czy heksadecymalna.

## 5. Notacja CMYK

**CMYK (Cyan, Magenta, Yellow, Key/Black)** jest głównie używany w druku i w CSS nie jest standardowo obsługiwany wprost. Niektóre przeglądarki mogą obsługiwać go w eksperymentalnych funkcjach (np. w color()), ale jest rzadko stosowany w systemach webowych.

- **Przykład:** Brak standardowego wsparcia, np. color: device-cmyk(0%, 100%, 100%, 0%) (eksperymentalne).
- **Zalety:** Przydatne w druku.
- **Wady:** Ograniczone wsparcie w CSS.

## 6. Format dziesiętny (rzadko używany)

Kolory można zapisać jako pojedyncza liczba dziesiętna reprezentująca **RGB w formacie binarnym** (rzadko stosowane w praktyce).

- **Przykład:** color: rgb(16711680); (odpowiednik #FF0000).
- **Zalety:** Brak, praktycznie nieużywany.
- **Wady:** Nieczytelny, niepraktyczny.

### Funkcja currentColor

**currentColor** to słowo kluczowe, które **przyjmuje wartość właściwości color elementu lub jego rodzica.**

**Przykład:**

```
div {  
    color: blue;  
    border: 1px solid currentColor; /* Border będzie niebieski */  
}
```

- **Zalety:** Umożliwia spójność kolorów bez powtórek.
- **Wady:** Zależność od wartości color.

## 8. Zmienne CSS (Custom Properties)

Kolory można przechowywać w zmiennych CSS, co ułatwia zarządzanie i ponowne użycie.

**Przykład:**

```
:root {  
    --primary-color: #3498db;  
}  
  
div {  
    background-color: var(--primary-color);  
}
```

- **Zalety:** Łatwa aktualizacja i ponowne użycie.
- **Wady:** Wymaga definicji zmiennych.

## 9. Funkcje kolorystyczne (nowoczesne, np. color())

Nowoczesne funkcje, takie jak **color()**, pozwalają na definiowanie kolorów w różnych modelach (np. sRGB, Display P3). Są częściowo wspierane w nowych przeglądarkach.

- **Przykład:** color(display-p3 1 0 0); (czerwony w przestrzeni Display P3).
- **Zalety:** Wsparcie dla szerokich gamutów kolorów.
- **Wady:** Ograniczone wsparcie w starszych przeglądarkach.

## 10. Gradienty (liniowe i radialne)

Gradienty pozwalają na płynne przejścia między kolorami.

- **Gradient liniowy:**

```
background: linear-gradient(to right, red, blue);  
  
○ Definiuje kierunek (np. to right, 45deg) i kolory.
```

- **Gradient radialny:**

background: radial-gradient([circle](#), yellow, green);

- Definiuje kształt (np. circle) i kolory od centrum.

- **Zalety:** Efekty wizualne, elastyczność.
- **Wady:** Bardziej złożone niż pojedyncze kolory.

## Tydzien 3 Lekcja 2

**Temat:** Model kontenerowy inaczej pudełkowy

### Znaczniki liniowe i blokowe

#### Znaczniki blokowe (`display: block`)

- **Definicja:** Elementy blokowe zajmują całą dostępną szerokość swojego kontenera nadzawanego, tworząc "blok", który zaczyna się od nowej linii i rozciąga się na całą szerokość. Każdy kolejny element blokowy pojawia się poniżej poprzedniego.
- **Cechy:**
  - Zajmują 100% szerokości rodzica (chyba że zmieniono to np. przez `width`).
  - Zawsze zaczynają się od nowej linii.
  - Mogą mieć ustawione właściwości takie jak `width`, `height`, `margin`, `padding` w sposób pełny.

- Przykłady domyślnych elementów blokowych: `<div>`, `<p>`, `<h1>`–`<h6>`, `<ul>`, `<li>`, `<section>`, `<article>`, `<form>`.

Przykład:

```
<div>Blok 1</div>
<div>Blok 2</div>
```

**Efekt:** Oba `<div>` pojawią się jeden pod drugim, każdy zajmując całą szerokość kontenera.

## Znaczniki liniowe (`display: inline`)

- **Definicja:** Elementy liniowe zajmują tylko tyle miejsca, ile jest potrzebne do wyświetlenia ich zawartości, i nie zaczynają się od nowej linii. Są ułożone obok siebie w tej samej linii, o ile pozwala na to przestrzeń.
- **Cechy:**
  - Nie można ustawić dla nich pełnych właściwości `width` i `height` (rozmiar zależy od zawartości).
  - Marginesy (`margin`) i wypełnienia (`padding`) działają tylko w poziomie (lewo/prawo), nie w pionie.
  - Przykłady domyślnych elementów liniowych: `<span>`, `<a>`, `<strong>`, `<em>`, `<img>`, `<b>`, `<i>`.

Przykład:

```
<span>Tekst 1</span>
<span>Tekst 2</span>
```

**Efekt:** Oba `<span>` pojawią się w tej samej linii, obok siebie, z tłem obejmującym tylko ich zawartość.

## Kontener wyśrodkowanie

**dla wszystkich**

```
margin: auto;
```

### dla prawego i lewego

```
margin-right: auto;  
margin-left: auto;  
lub  
margin: 0px auto 0px auto;
```

```
<!DOCTYPE html>  
<html lang="pl">  
<head>  
    <meta charset="utf-8">  
    <meta name="viewport" content="width=device-width, initial-scale=1">  
    <title>Przykładowa struktura</title>  
    <style>  
  
        div#glowny {  
            border: 1px solid #000;  
            background: #00FF00;  
            min-height: 500px;  
            min-width: 500px;  
            margin-right: auto;  
            margin-left: auto;  
        }  
  
        div#zagniezdzony {  
            border: 1px solid #000;  
            background: #0000FF;  
            min-height: 200px;  
            min-width: 200px;  
            margin: 0px auto 0px auto;  
        }  
  
        footer {}  
    </style>  
</head>  
<body>  
    <header>  
        <h1>Nagłówek (header)</h1>  
    </header>
```

```

<div id="glowny">
  <div id="zagniezdzony">
    <p>To jest zagnieżdzony DIV (#zagniezdzony) wewnątrz #glowny.</p>
  </div>
</div>

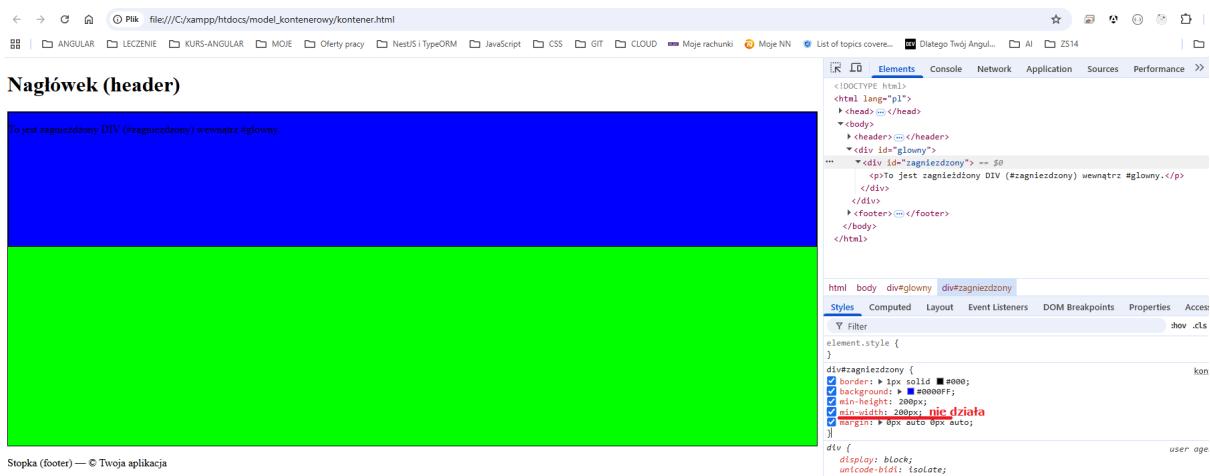
<footer>
  <p>Stopka (footer) — © Twoja aplikacja</p>
</footer>
</body>
</html>

```

## Wymiary:

**min-height:200px - działa**

**min-width: 200px - nie działa**



```

<!DOCTYPE html>
<html lang="pl">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Przykładowa struktura</title>
  <style>

    div#glowny {
      border: 1px solid #000;
      background: #00FF00;
      height: 500px;
    }
  </style>

```

```
        width: 500px;
        margin-right: auto;
        margin-left: auto;
    }

    div#zagniezdzony {
        border: 1px solid #000;
        background: #0000FF;
        height: 200px;
        width: 200px;
        margin: 0px auto 0px auto;
    }
    footer { }

```

</style>

</head>

<body>

<header>

<h1>Nagłówek (header)</h1>

</header>

<div id="glowny">

<div id="zagniezdzony">

<p>To jest zagnieżdzony DIV (#zagniezdzony) wewnątrz #glowny.</p>

</div>

</div>

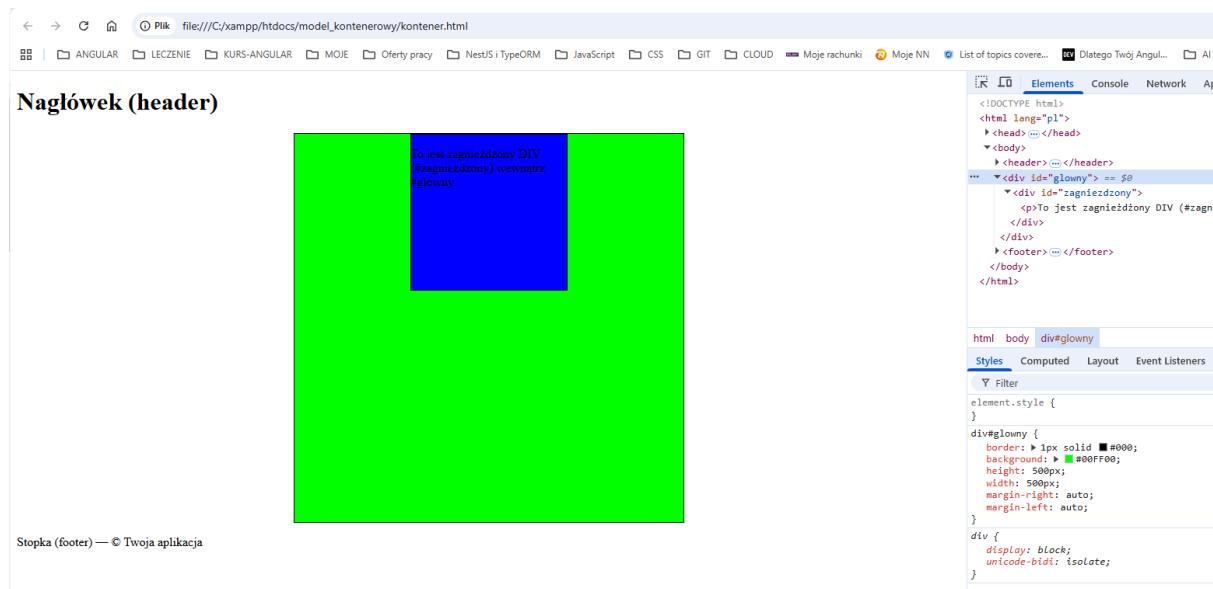
<footer>

<p>Stopka (footer) — © Twoja aplikacja</p>

</footer>

</body>

</html>



Właściwość **overflow** określa zachowanie przeglądarki, gdy zawartość elementu przekracza jego wymiary (tzw. overflow). Może być stosowana do elementów blokowych i liniowo-blokowych, które mają zdefiniowaną wysokość lub szerokość. overflow można ustawić dla **osi X (poziomej)** i **Y (pionowej)** osobno za pomocą **overflow-x** i **overflow-y** lub dla obu osi jednocześnie za pomocą overflow.

### 1. **visible** (domyślna):

- Zawartość, która nie mieści się w elemencie, jest wyświetlana poza jego granicami, bez przycinania.
- Element nie generuje pasków przewijania.

**Przykład:**

```
div {
  width: 100px;
  height: 100px;
  overflow: visible;
}
```

**Efekt:** Nadmiarowa zawartość (np. długie teksty) będzie widoczna poza granicami elementu.

### 2. **hidden**:

- Nadmiarowa zawartość jest przycinana i niewidoczna poza granicami elementu.
- Nie pojawiają się paski przewijania.

Przykład:

```
div {  
    width: 100px;  
    height: 100px;  
    overflow: hidden;  
}
```

**Efekt:** Zawartość poza granicami 100x100 pikseli jest niewidoczna.

### 3. scroll:

- Nadmiarowa zawartość jest przycinana, ale dodawane są paski przewijania (zarówno poziomy, jak i pionowy, niezależnie od tego, czy są potrzebne).

Przykład:

```
div {  
    width: 100px;  
    height: 100px;  
    overflow: scroll;  
}
```

**Efekt:** Pojawiają się paski przewijania, umożliwiające przeglądanie całej zawartości.

### 4. auto:

- Paski przewijania pojawiają się tylko wtedy, gdy zawartość przekracza granice elementu.
- Jest to najczęściej używana wartość, ponieważ dostosowuje się do potrzeb.

Przykład:

```
div {  
    width: 100px;  
    height: 100px;  
    overflow: auto;  
}
```

**Efekt:** Paski przewijania pojawią się tylko, gdy zawartość wykracza poza wymiary elementu.

## 5. **clip** (od CSS Overflow Module Level 3, wsparcie w nowoczesnych przeglądarkach):

- Podobne do hidden, ale bardziej restrykcyjne – zawartość jest przycinana, i nie ma możliwości przewijania (nawet programistycznego).

Przykład:

```
div {  
    width: 100px;  
    height: 100px;  
    overflow: clip;  
}
```

**Efekt:** Zawartość jest przycinana bez możliwości przewijania.

## Właściwości **overflow-x** i **overflow-y**

- **overflow-x:** Kontroluje przepełnienie w osi poziomej (X).
- **overflow-y:** Kontroluje przepełnienie w osi pionowej (Y).
- Można łączyć różne wartości dla każdej osi.

Przykład:

```
div {  
    width: 100px;  
    height: 100px;  
    overflow-x: hidden; /* Przycina w poziomie */  
    overflow-y: auto; /* Pasek przewijania w pionie, jeśli potrzebny */  
}
```

# Tydzien 3 Lekcja 3

**Temat:** Interpreter. ISO position.

**Interpreter - to program komputerowy, który odczytuje, analizuje, i wykonuje kod źródłowy innego programu bezpośrednio, linia po linii.**

**linii, bez wcześniejszego tłumaczenia go na kod maszynowy (tak jak w przypadku kompilatora)**

 **Jak działa?** Analizuje kod źródłowy na bieżąco i wykonuje przeanalizowane fragmenty. To sprawia, że wykonanie jest wolniejsze niż w przypadku skompilowanego kodu (ze względu na "koszt interpretacji"), ale ułatwia szybkie testowanie i debugowanie – nie trzeba kompilować programu za każdym razem.

 **Zalety:** Szybszy cykl rozwoju (edycja → interpretacja → testowanie), przenośność na różne platformy (kod źródłowy działa wszędzie, gdzie jest interpreter).

 **Wady:** Większe zużycie zasobów w czasie wykonywania i wolniejsza praca.

 **Przykłady:** Interpreter Pythona (CPython), Node.js dla JavaScriptu, czy programy jak GNUPlot czy MATLAB.

ISO - Międzynarodowa Organizacja Normalizacyjna (International Organization for Standardization) **to pozarządowa organizacja, która opracowuje i publikuje międzynarodowe standardy (normy) dotyczące jakości, bezpieczeństwa i wydajności produktów, procesów i systemów zarządzania.**

**Celem ISO jest zapewnienie spójności i najwyższej jakości usług i produktów na całym świecie, ułatwiając jednocześnie współpracę między firmami i instytucjami.**

Popularne normy ISO

- ◆ **ISO 9001** (zarządzanie jakością),
- ◆ **ISO 14001** (zarządzanie środowiskowe)
- ◆ **ISO 45001** (bezpieczeństwo i higiena pracy).
- ◆ **ISO/IEC 27001:** Dotyczy systemów zarządzania bezpieczeństwem informacji i cyberbezpieczeństwem.

◆ **Rodzina ISO-8859 (8-bitowe, każdy dla innego regionu)**

- **ISO-8859-1** – Latin-1 (Europa Zachodnia, brak polskich liter)
- **ISO-8859-2** – Latin-2 (Europa Środkowa i Wschodnia – **polski**, czeski, węgierski itd.)

- **ISO-8859-5** – cyrylica (rosyjski, bułgarski itd.)
- **ISO-8859-6** – arabski
- **ISO-8859-7** – grecki
- **ISO-8859-8** – hebrajski  
*(i inne do różnych alfabetów)*

**colspan="wartosc"**

**rowspan="wartosc"**

```
<table border="1">
  <tr>
    <th>Kolumna 1</th> <th>Kolumna 2</th> <th>Kolumna 3</th>
  </tr>
  <tr>
    <td rowspan="2">Komórka scalona na 2 wiersze (rowspan)</td>
    <td>Komórka 2-2</td>
    <td>Komórka 2-3</td>
  </tr>
  <tr>
    <td>Komórka 3-2</td>
    <td>Komórka 3-3</td>
  </tr>
  <tr>
    <td>Komórka 4-1</td>
    <td>Komórka 4-2</td>
    <td>Komórka 4-3</td>
  </tr>
  <tr>
    <td>Komórka 5-1</td>
    <td colspan="2">Komórki scalone z 2 kolumn (colspan)</td>
  </tr>
</table>
```

## Nieuporządkowana lista

```
<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

- Coffee
- Tea
- Milk

## Uporządkowana lista

```
<ol>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

1. Coffee
2. Tea
3. Milk

## Inne listy

```
<dl>
  <dt>Coffee</dt>
  <dd>- black hot drink</dd>
  <dt>Milk</dt>
  <dd>- white cold drink</dd>
</dl>
```

Coffee  
- black hot drink  
Milk  
- white cold drink

Atrybut	Znaczenie
type	Typ numeracji: "1" (domyślnie), "A" (wielkie litery), "a" (małe litery), "I" (wielkie rzymskie), "i" (małe rzymskie).
start	Numer, od którego zaczyna się lista (np. <code>start="5"</code> → pierwszy element ma numer 5).
reversed	Odwaca kolejność numeracji (np. ostatni element będzie 1).

`position: static;`

### Zachowanie

- **Element jest pozycjonowany zgodnie z normalnym układem dokumentu** (czyli tak, jak został umieszczony w kodzie HTML).
- Właściwości **top, right, bottom, left, z-index** **nie działają**
- Jest to wartość domyślna dla wszystkich elementów.

`position: relative;`

### Zachowanie

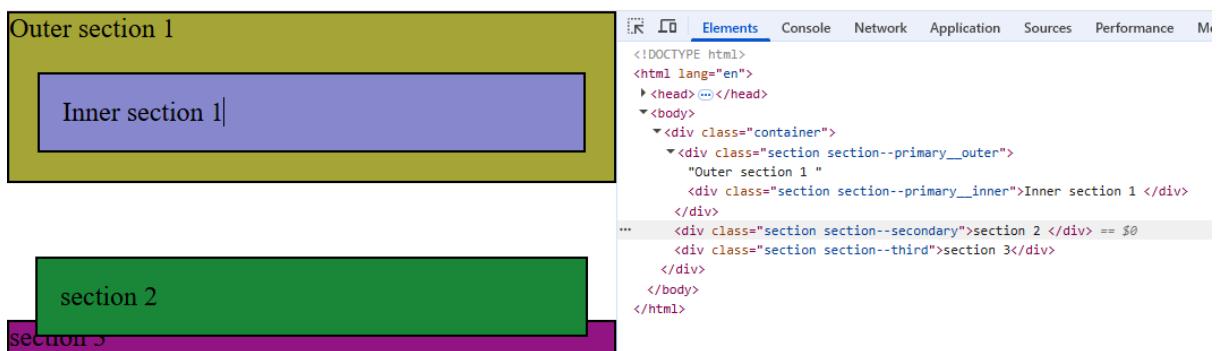
- **pozostaje w normalnym przepływie dokumentu** (tak jak `static`), ale...
- **możesz go przesunąć** względem jego pierwotnej pozycji za pomocą **top, left, right, bottom**. (**może się nałożyć na inny element**)

```

<div class="container">
  <div class="section section--primary__outer">Outer section 1
    <div class="section section--primary__inner">Inner section 1 </div>
  </div>
  <div class="section section--secondary">section 2 </div>
  <div class="section section--third">section 3</div>
</div>

.section--secondary {
  position: relative;
  /*
  // po dodaniu top: 40px; nałoży się na class="section section--third"
  */
  top: 40px;
  padding: 20px;
  margin: 25px;
  background-color: rgb(25, 138, 59);
}

```



- **Nie ma wpływu na układ innych elementów** (inne elementy zachowują się tak, jakby dany element znajdował się w swoim oryginalnym położeniu).
- **relative tworzy kontekst pozycjonowania** – jeśli w środku jest element z **position: absolute**, to będzie pozycjonowany względem tego rodzica, a nie całej strony.

```

<!DOCTYPE html>
<html lang="en">
  <head> ...
  </head>
  <body>
    <div class="container">
      <div class="section section--primary__outer">
        "Outer section 1 "
        <div class="section section--primary__inner">Inner section 1 </div>
      </div>
      <div class="section section--secondary">section 2 </div>
      <div class="section section--third">section 3</div>
    </div>
  </body>
</html>

```

przesunie się top:10px; względem .section primary\_\_outer

**position: absolute;**

## ✓ 1. Zostaje wyjęty z normalnego przepływu dokumentu

- Nie zajmuje już miejsca w układzie strony.
- **Inne elementy zachowują się tak, jakby go nie było.**

```

<!DOCTYPE html>
<html lang="en">
  <head> ...
  </head>
  <body>
    <div class="container">
      <div class="section section--primary__outer">
        "Outer section 1 "
        <div class="section section--primary__inner">Inner section 1 </div>
      </div>
      <div class="section section--secondary">section 2 </div>
      <div class="section section--third">section 3</div>
    </div>
  </body>
</html>

```

section 3 zachowuje się tak jakby section 2 nie było, bo section 2 ma absolute

## ✓ 2. Jest pozycjonowany względem najbliższego przodka z pozycjonowaniem innym niż static

- Szuka w góre drzewa DOM elementu, który ma **position: relative, absolute, fixed lub sticky**.
- Jeśli takiego nie znajdzie, pozycjonuje się względem **okna przeglądarki (elementu <html> / <body>)**.

```

parent {
  position: relative; /* tworzy kontekst pozycjonowania */
  width: 300px;
  height: 300px;
}

```

```
background: lightblue;  
}
```

```
.child {  
position: absolute;  
top: 20px;  
left: 30px;  
background: orange;  
}
```

`.child` ustawi się **20px od góry i 30px od lewej krawędzi `.parent`**, bo `.parent` ma `position: relative`.

### ✓ 3. Możesz używać `top`, `right`, `bottom`, `left`

`position: fixed;`

✓ 1. oznacza, że element jest przyklejony do okna przeglądarki (viewport), a nie do dokumentu.

### ✓ 2. Wyjęcie z normalnego przepływu

- Podobnie jak `absolute`, element **nie zajmuje miejsca w układzie** – inne elementy zachowują się, jakby go nie było.

### ✓ 3. Pozycjonowanie względem okna przeglądarki

- Właściwości `top`, `right`, `bottom`, `left` określają odległość od krawędzi **okna**, a nie od rodzica.
- Nawet jeśli przewijasz stronę, element **pozostaje w tym samym miejscu na ekranie**.

**position: sticky;**

✓ 1. to ciekawa hybryda **relative** i **fixed**. Oto jak działa:

✓ 2. Zachowuje się jak **relative**... dopóki nie przewiniesz

- Element jest w normalnym przepływie dokumentu.
  - Możesz używać `top`, `left`, `right`, `bottom` – ale początkowo działa jak `relative`.

 3. ...a potem „przykleja się” jak **fixed**

- Gdy przewijasz stronę i element osiągnie określoną pozycję (np. `top: 0`), **przykleja się do tej krawędzi** w obrębie swojego kontenera.
  - **Nie wychodzi poza obszar swojego rodzica** – gdy rodzic się kończy, element przestaje być sticky.

## Wartości globalne:

**position: inherit;**

**position**: initial;

**position:** revert;

**position:** revert-layer;

**position**: unset;

## Tydzień 3 Lekcja 4

## **Temat:** Funkcje modelu DOM

**Model DOM (Document Object Model)** to standardowy interfejs programistyczny do reprezentacji i manipulacji strukturą dokumentów HTML lub XML. Pozwala na dynamiczne odczytywanie, modyfikowanie i tworzenie elementów drzewa dokumentu. DOM jest zdefiniowany przez W3C i jest niezależny od języka programowania, ale implementacje różnią się w zależności od środowiska.

W JavaScript DOM jest natywnie dostępny w przeglądarkach do interakcji z stronami internetowymi, podczas gdy w PHP jest to rozszerzenie biblioteki do przetwarzania XML/HTML po stronie serwera

## Funkcje DOM w JavaScript

W JavaScript DOM jest dostępny poprzez obiekt document i umożliwia manipulację elementami HTML w czasie rzeczywistym. Oto wybrane metody, pogrupowane tematycznie:

Kategoria	Metoda	Opis	Przykład użycia
Wybieranie elementów	getElementById(id)	Zwraca element o podanym ID.	document.getElementById('mojElement').innerHTML = 'Nowa treść';
Wybieranie elementów	getElementsByClassName(className)	Zwraca kolekcję elementów o danej klasie.	document.getElementsByClassName('klasa')[0].style.color = 'red';
Wybieranie elementów	getElementsByTagName(tagName)	Zwraca kolekcję elementów o danym tagu.	document.getElementsByTagName('p').length;

Wybieranie elementów	<code>querySelector(selector)</code>	Zwraca pierwszy element pasujący do selektora CSS.	<code>document.querySelector('.klasa').textContent = 'Tekst';</code>
Wybieranie elementów	<code>querySelectorAll(selector)</code>	Zwraca wszystkie elementy pasujące do selektora CSS.	<code>document.querySelectorAll('div').forEach(el =&gt; el.style.background = 'blue');</code>
Tworzenie i modyfikacja	<code>createElement(tagName)</code>	Tworzy nowy element.	<code>let nowyDiv = document.createElement('div');</code>
Tworzenie i modyfikacja	<code>appendChild(child)</code>	Dodaje dziecko do elementu.	<code>document.body.appendChild(nowyDiv);</code>
Tworzenie i modyfikacja	<code>removeChild(child)</code>	Usuwa dziecko z elementu.	<code>parent.removeChild(child);</code>
Tworzenie i modyfikacja	<code>insertBefore(newNode, refNode)</code>	Wstawia nowy węzeł przed referencyjnym.	<code>parent.insertBefore(nowy, istniejacy);</code>
Tworzenie i modyfikacja	<code>replaceChild(newChild, oldChild)</code>	Zastępuje stare dziecko nowym.	<code>parent.replaceChild(nowy, stary);</code>
Atrybuty i właściwości	<code>getAttribute(attr)</code>	Pobiera wartość atrybutu.	<code>element.getAttribute('src');</code>
Atrybuty i właściwości	<code>setAttribute(attr, value)</code>	Ustawia wartość atrybutu.	<code>element.setAttribute('class', 'nowaKlasa');</code>
Wydarzenia	<code>addEventListener(event, callback)</code>	Dodaje nasłuchiwacz zdarzeń.	<code>element.addEventListener('click', () =&gt; alert('Kliknięto!'));</code>

## Funkcje DOM w PHP

W PHP DOM jest dostępny poprzez klasy z rozszerzeniem DOM (wbudowane w PHP od wersji 5). Służy głównie do parsowania i generowania XML/HTML po stronie serwera. Kluczowa klasa to **DOMDocument**. Od PHP 8.4 dodano metody jak **querySelector** i **querySelectorAll** dla łatwiejszego wybierania elementów. Oto wybrane metody:

Kategoria	Klasa/Metoda	Opis	Przykład użycia
Ładowanie dokumentu	DOMDocument::load(filename)	Ładuje XML z pliku.	\$dom = new DOMDocument(); \$dom->load('plik.xml');
Ładowanie dokumentu	DOMDocument::loadHTML(html)	Ładuje HTML jako string.	\$dom->loadHTML('<html><body></body></html>');
Wybieranie elementów	DOMDocument::getElementById(id)	Zwraca element o podanym ID.	\$dom->getElementById('mojId');
Wybieranie elementów	DOMDocument::getElementsByName(tag)	Zwraca kolekcję elementów o danym tagu.	\$dom->getElementsByName('p') ->item(0);
Wybieranie elementów	DOMDocument::querySelector(selector) (od PHP 8.4)	Zwraca pierwszy element pasujący do selektora CSS.	\$dom->querySelector('.klasa');
Wybieranie elementów	DOMDocument::querySelectorAll(selector) (od PHP 8.4)	Zwraca wszystkie elementy pasujące do selektora CSS.	\$dom->querySelectorAll('div');
Tworzenie i modyfikacja	DOMDocument::createElement(tagName [, value])	Tworzy nowy element.	\$nowy = \$dom->createElement('div', 'Treść');
Tworzenie i modyfikacja	DOMNode::appendChild(child)	Dodaje dziecko do węzła.	\$parent->appendChild(\$nowy);
Tworzenie i modyfikacja	DOMNode::removeChild(child)	Usuwa dziecko z węzła.	\$parent->removeChild(\$child);

Tworzenie i modyfikacja	<code>DOMNode::insertBefore(newNode, refNode)</code>	Wstawia nowy węzeł przed referencyjnym.	<code>\$parent-&gt;insertBefore(\$nowy, \$ref);</code>
Tworzenie i modyfikacja	<code>DOMNode::replaceChild(newChild, oldChild)</code>	Zastępuje stare dziecko nowym.	<code>\$parent-&gt;replaceChild(\$nowy, \$stary);</code>
Atrybuty	<code>DOMELEMENT::getAttribute(attr)</code>	Pobiera wartość atrybutu.	<code>\$element-&gt;getAttribute('class');</code>
Atrybuty	<code>DOMELEMENT::setAttribute(attr, value)</code>	Ustawia wartość atrybutu.	<code>\$element-&gt;setAttribute('id', 'nowyId');</code>
Zapisywanie	<code>DOMDocument::save(filename)</code>	Zapisuje XML do pliku.	<code>\$dom-&gt;save('wyjscie.xml');</code>
Zapisywanie	<code>DOMDocument::saveHTML()</code>	Zwraca HTML jako string.	<code>echo \$dom-&gt;saveHTML();</code>

## Tydzień 3 Lekcja 4

### Temat: Pseudoelementy

Link gdzie można potestować:

[https://www.w3schools.com/css/css\\_pseudo\\_elements.asp](https://www.w3schools.com/css/css_pseudo_elements.asp)

**WAŻNE Pseudoelementy mają podwójny dwukropki (::) , a Pseudoklasy pojedynczy dwukropki (:)**

#### 1. ::before

- **Opis:** Tworzy pseudoelement, który jest wstawiany **przed zawartością** wybranego elementu.
- **Zastosowanie:** Dodawanie dekoracyjnych elementów, ikon, tekstu lub innych treści przed główną zawartością elementu. Wymaga właściwości content.

## **Przykład:**

```
p::before {  
    content: "★";  
    color: gold;  
}
```

Wstawia złotą gwiazdkę przed każdym akapitem (<p>).

## **2. ::after**

- **Opis:** Tworzy pseudoelement, który jest wstawiany **po zawartości** wybranego elementu.
- **Zastosowanie:** Podobne do ::before, ale dla treści dodawanej na końcu elementu.

## **Przykład:**

```
p::after {  
    content: "[koniec]";  
    color: red;  
}
```

Dodaje tekst "[koniec]" w kolorze czerwonym po każdym akapicie.

## **3. ::first-line**

- **Opis:** Stylizuje **pierwszą linię** tekstu w elemencie blokowym.
- **Zastosowanie:** Formatowanie pierwszej linii akapitu, np. zmiana czcionki, koloru lub wcięcia.

## **Przykład:**

```
p::first-line {  
    font-weight: bold;  
    color: blue;  
}
```

Pierwsza linia każdego akapitu będzie pogrubiona i niebieska. **Uwaga:** Działa tylko na elementy blokowe (np. div, p) i tylko dla właściwości związanych z tekstem (np. font, color, text-transform).

## **4. ::first-letter**

- **Opis:** Stylizuje **pierwszą literę** pierwszej linii w elemencie blokowym.
- **Zastosowanie:** Tworzenie inicjałów (drop caps) lub wyróżnianie pierwszej litery akapitu.

### **Przykład:**

```
p::first-letter {
    font-size: 2em;
    color: red;
    float: left;
}
```

Pierwsza litera akapitu będzie większa, czerwona i "pływająca" po lewej stronie.

**Uwaga:** Działa tylko na elementy blokowe.

### 5. **::selection**

- **Opis:** Stylizuje tekst zaznaczony przez użytkownika (np. myszą lub klawiaturą).
- **Zastosowanie:** Zmiana wyglądu zaznaczonego tekstu, np. tła lub koloru tekstu.

### **Przykład:**

```
::selection {
    background: yellow;
    color: black;
}
```

Zaznaczony tekst będzie miał żółte tło i czarny kolor liter. **Uwaga:** Obsługuje tylko właściwości color, background, background-color i text-shadow.

### 6. **::marker** (od CSS3)

- **Opis:** Stylizuje znaczniki (bullet points) w listach (<ul>, <ol>).
- **Zastosowanie:** Zmiana wyglądu punktorów lub numerów w listach.

### **Przykład:**

```
li::marker {
    color: green;
    content: "> ";
```

}

Zmienia znacznik listy na zieloną strzałkę. **Uwaga:** Działa tylko na elementy z display: list-item.

## 7. ::backdrop

- **Opis:** Stylizuje tło elementów w trybie pełnoekranowym (np. w trybie pełnego ekranu API lub dialogów <dialog>).
- **Zastosowanie:** Zmiana tła za elementem w trybie pełnoekranowym.

### Przykład:

```
dialog::backdrop {  
    background: rgba(0, 0, 0, 0.5);  
}
```

Tło za oknem dialogowym będzie półprzezroczyste czarne.

## 8. ::placeholder (czasami uznawany za pseudoelement, choć formalnie pseudoklasa w starszych specyfikacjach)

- **Opis:** Stylizuje tekst zastępczy (placeholder) w polach formularzy (<input>, <textarea>).
- **Zastosowanie:** Zmiana wyglądu tekstu w polach formularzy.

### Przykład:

```
input::placeholder {  
    color: gray;  
    font-style: italic;  
}
```

Tekst zastępczy w polach <input> będzie szary i pochylony.

## BRAMKI logiczne

0 = false

1 = true

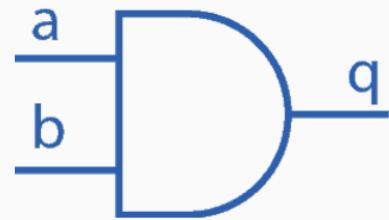
Boolean(0) zwróci false

Boolean(1) zwróci true

Bramki logiczne tabela prawdy dla iloczynu logicznego (AND)

Symbol – AND

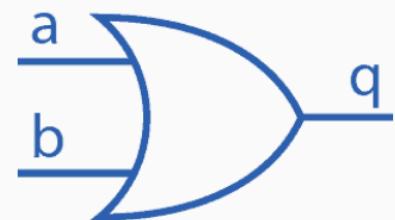
a	b	q
0	0	0
0	1	0
1	0	0
1	1	1



Tablica prawdy dla sumy logicznej (OR)

Bramki logiczne symbole – OR

a	b	q
0	0	0
0	1	1
1	0	1
1	1	1



**AND (&&):** Zwraca true, jeśli oba operandy są true; **w przeciwnym razie zwraca false lub wartość pierwszego "fałszowego" operandu w przypadku wartości niebędących true/false.**

**OR (||):** Zwraca true, jeśli przynajmniej jeden operand jest true; w przeciwnym razie zwraca false lub wartość pierwszego "prawdziwego" operandu w przypadku wartości niebędących true/false.

## Tydzien 3 Lekcja 4

**Temat:** Metody formularza, Operator **nullish coalescing**

W PHP (a tak naprawdę w **HTML**, bo atrybut **method** należy do znacznika `<form>`) mamy głównie **dwie metody**:

#### 1. method="GET"

- dane formularza są **doklejane do adresu URL** w postaci ciągu zapytania (`?klucz=wartość&...`),
- parametry są dostępne w tablicy superglobalnej **`$_GET`**,
- ograniczenie długości danych (adres URL nie może być zbyt długi),
- dobra do prostych formularzy, np. wyszukiwania, filtrowania, bo wynik można zapisać w zakładkach.

#### 2. method="POST"

- dane formularza są przesyłane **w treści zapytania HTTP**,
- parametry są dostępne w **`$_POST`**,
- brak ograniczenia długości,
- bezpieczniejsze (dane nie pojawiają się w adresie), używane do logowania, rejestracji, przesyłania plików.

### **`$_GET[ 'page' ]`**

Służy do **odczytywania danych przesłanych w adresie URL** (parametry GET).

Przykład URL:

`http://localhost/index.php?page=2`

W PHP:

```
echo $_GET[ 'page' ]; // wyświetli: 2
```

Używamy, gdy chcemy np. przechwycić numer strony, filtr, wyszukiwanie – wszystko, co jest w `?parametr=wartość`.

## `$_POST['rodzaj']`

Służy do odczytywania danych przesyłanych formularzem metodą POST.

Przykład formularza:

```
<form method="post">

    <select name="rodzaj">

        <option value="INNE">INNE</option>

        <option value="PIECZYWO">PIECZYWO</option>

    </select>

    <input type="submit">

</form>
```

W PHP:

```
$rodzaj = $_POST['rodzaj']; // pobiera wartość wybraną w formularzu
```

Używamy, gdy chcemy bezpiecznie przesłać dane, które nie powinny być widoczne w URL (np. hasła, formularze).

Inne (rzadziej spotykane w HTML, ale istnieją w specyfikacji HTTP):

- PUT, PATCH, DELETE – metody REST-owe, używane głównie w API.  
Normalny `<form>` w HTML nie wspiera ich bezpośrednio, ale można ich używać np. przez **JavaScript (AJAX/fetch)** albo frameworki (np. Laravel używa ukrytych pól `_method` do symulacji).

Operator **nullish coalescing** (??) to operator w wielu językach programowania (np. w PHP, JavaScript, C#), który sprawdza, czy dana wartość jest **null** albo **undefined (w JS) / null (w PHP)** i w takim przypadku zwraca wartość domyślną.

```
<?php
```

```
echo "null ?? 'default' = " . ($null ?? 'default') . "<br>";           // default
echo "false ?? 'default' = " . (false ?? 'default') . "<br>";             // false
echo "true ?? 'default' = " . (true ?? 'default') . "<br>";              // true
echo "" ?? 'default' = " . (" ?? 'default') . "<br>";                   // (pusty string)
echo "1 ?? 'default' = " . (1 ?? 'default') . "<br>";                     // 1
echo "0 ?? 'default' = " . (0 ?? 'default') . "<br>";                     // 0
echo "'default' ?? " = " . ('default' ?? "") . "<br>";                  // default
echo "'default' ?? 1 = " . ('default' ?? 1) . "<br>";                     // default
echo "null ?? 'default' = " . (null ?? 'default') . "<br>";               // default
echo "'default' ?? null = " . ('default' ?? null) . "<br>";                // default
echo "[] ?? 'default' = "; var_dump([] ?? 'default'); echo "<br>";          // array(0) {}
echo "null ?? [] = "; var_dump(null ?? []); echo "<br>";                  // array(0) {}
echo "[] ?? [['a' => '11']] = "; var_dump([] ?? [['a' => '11']])); echo "<br>"; // array(0) {}
?>
```

✓ Wyjaśnienie:

- Operator ?? zwraca **pierwszą wartość, która nie jest null**.
- **false, 0, '' , [ ] nie są null**, więc zostaną wyświetcone takimi, jakie są.
- **null** → zostaje zastąpione wartością po ??.

```
<!DOCTYPE html>
<html lang="pl">
<head>
  <meta charset="UTF-8">
  <title>Przykład selektorów CSS</title>
  <style>
    /* nth-of-type */
    p:nth-of-type(2) { color: red; }      /* drugi <p> w rodzicu */
    p:last-of-type { color: gray; }       /* ostatni <p> w rodzicu */
    p:nth-of-type(odd) { background-color: lightblue; } /* nieparzyste <p> */
    p:nth-of-type(even) { background-color: lightgreen; } /* parzyste <p> */
```

```
/* sąsiedztwo */
ul + p { color: orange; }      /* bezpośrednio po ul */
ul ~ p { font-weight: bold; }   /* wszystkie p po ul w tym samym rodzicu */

/* bezpośredni potomek */
ul > p { border: 2px solid red; } /* działa tylko jeśli <p> jest bezpośrednio w
<ul> */

/* not */
p:not(#special) { font-style: italic; }

/* ID */
#special { background-color: yellow; }

</style>
</head>
<body>
<div class="content">
    <p>Paragraph 1</p>
    <p>Paragraph 2</p>
    <p id="special">Paragraph 3 (special)</p>
    <p>Paragraph 4</p>

    <ul>
        <p>Paragraph bezpośrednio w ul (niezalecane w HTML, działa z ul >
p)</p>
        <li>Item 1</li>
        <li>
            <p>Paragraph 5</p>
            <p>Paragraph 6</p>
        </li>
    </ul>

    <p>Paragraph 7</p>

    <section>
        <h1>Section H1</h1>
        <p>Paragraph 8</p>
        <p>Paragraph 9</p>
    </section>
</div>
</body>
</html>
```

## Wyjaśnienia:

- `p:nth-of-type(2)` – drugi `<p>` w rodzicu.
- `p:last-of-type` – ostatni `<p>` w rodzicu.
- `p:nth-of-type(odd/even)` – parzyste i nieparzyste `<p>` w rodzicu.
- `ul + p` – `<p>` bezpośrednio po `<ul>` w tym samym rodzicu.
- `ul ~ p` – wszystkie `<p>` po `<ul>` w tym samym rodzicu.
- `ul > p` – **bezpośrednie dzieci `<ul>`**, w HTML poprawne są tylko `<li>`, więc użycie `<p>` w `<ul>` jest niezalecane, ale działa w CSS do demonstracji.
- `p:not(#special)` – wszystkie `<p>` poza tym z `id="special"`.
- `#special` – selektor po `id`

## Lekcja 5

**Temat:** Flexbox. box-sizing: border-box;

```
* {  
    box-sizing: border-box;  
}
```

**box-sizing: border-box;**

**Domyślnie w CSS szerokość (width) i wysokość (height) elementu nie obejmuje paddingu i borderu.**

**content-box** (wartość domyślna): **width dotyczy tylko zawartości. Padding i border są dodawane na zewnątrz**, więc faktyczny rozmiar rośnie.

**border-box: width obejmuje zawartość + padding + border.** Dzięki temu rozmiar elementu jest dokładnie taki, jak ustawisz w width i height, bez powiększania.

Przykład:

```
div {  
    width: 200px;  
    padding: 20px;  
    border: 5px solid black;  
}
```

Przy **content-box (domyślnie)**:

faktyczna szerokość = **200 + 20 + 20 + 5 + 5 = 250px**

Przy **border-box**:

faktyczna szerokość = **200px** (padding i border mieszczą się w środku tej wartości).

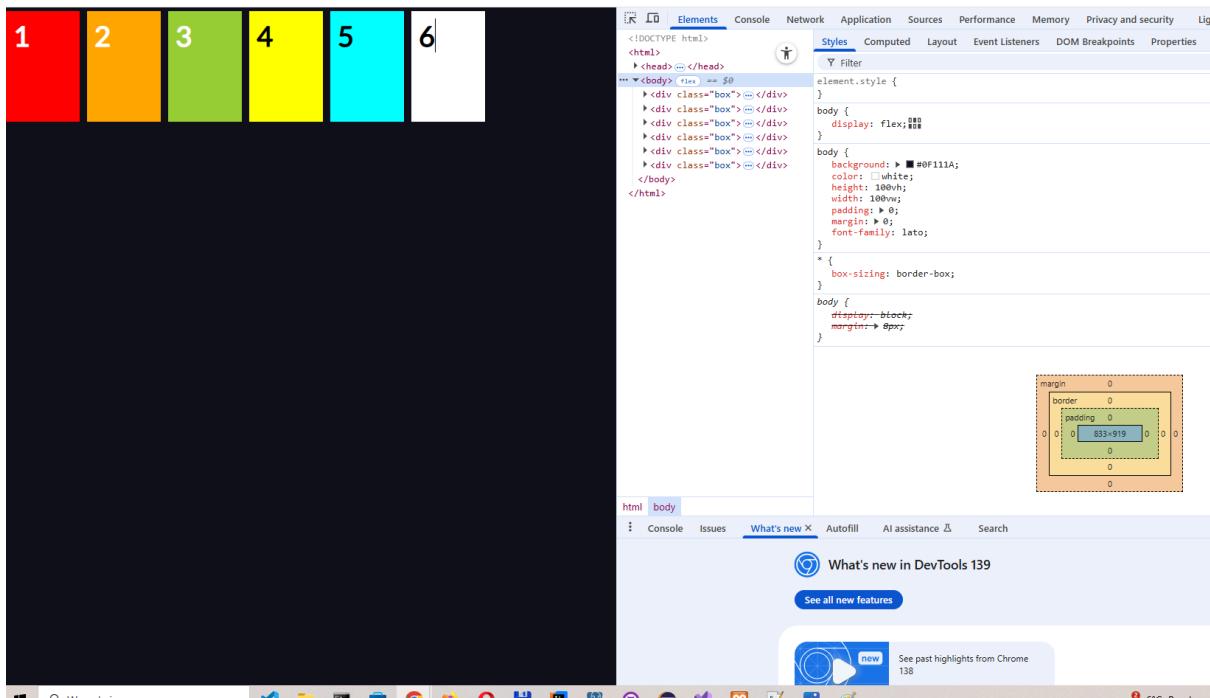
## FLEXBOX

Domyślnie div zajmuje całą szerokość dlatego są pionowo ułożone

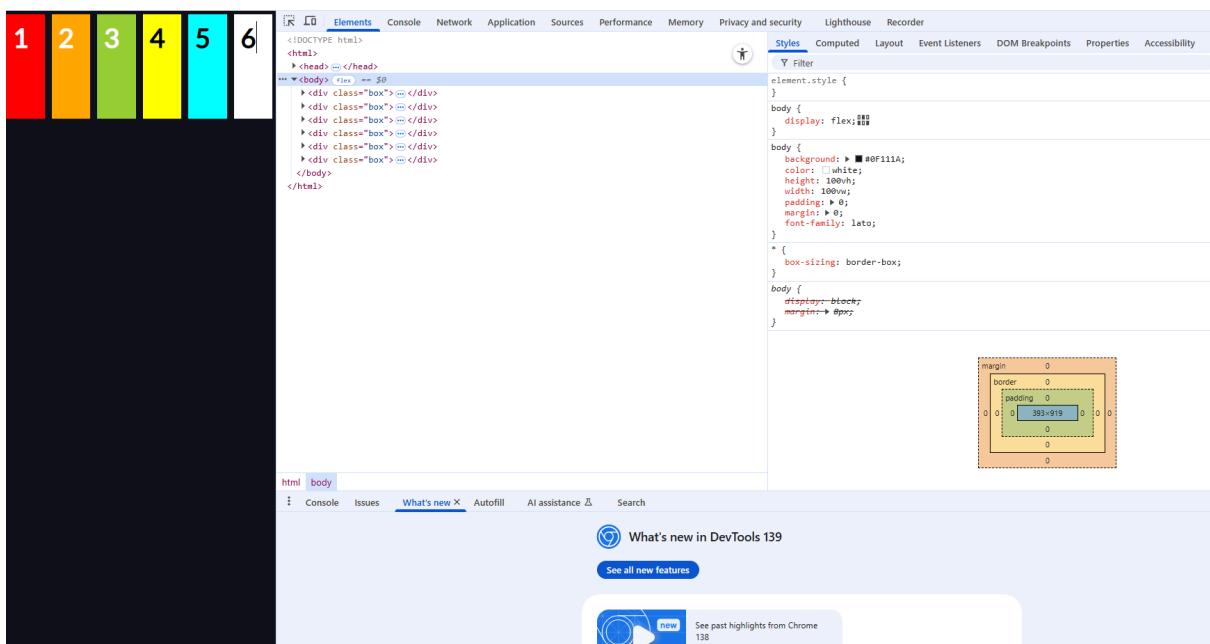
Aby zaaplikować flexbox do tego przykładu i manipulować tymi 6 pudełkami (.box) musimy na rodzicu tych pudełek ustawić display: flex . W naszym przypadku jest nim znacznik body

```
body {  
    display: flex;  
}
```

Ustawienie spowodowało wyrenderowanie się pudełek w poziomie



Dodatkowo kurczą się w szerokości jak zmniejszamy szerokość strony.



Domyślnymi wartościami dla flexbox są właściwości:

```
body {
  display: flex;
  flex-direction: row;
  flex-wrap: nowrap;
}
```

flex-direction: row; - ustawia elementy w wierszu

flex-direction: column; - ustawia elementy w kolumnie

### **flex-wrap**

To na poziomie kontenera (rodzica):

- decyduje, **czy elementy mają prawo przechodzić do nowej linii**, kiedy się nie mieścią.

flex-wrap: nowrap; - elementy **nie zawijają się** (to domyślna wartość). Zostają w jednej linii, i wtedy właśnie wchodzi do gry **flex-shrink** → bo muszą się „ścisnąć”, żeby się zmieścić.

flex-wrap: wrap; - elementy mogą **zawijać się do następnego wiersza** zamiast się kurczyć. Elementy zostają rozstrzelone na 2 linie i mają swoje naturalne rozmiary ustawione przez właściwość width w css

### **flex-shrink**

To na poziomie elementu (dziecka):

Trzecią właściwością której efekt już w tym momencie widzimy, a która jest automatycznie uruchamiana w momencie pozycjonowania na flexboxie jest właściwość:

**flex-shrink** - **UWAGA flex-shrink nie jest ustawiony na rodzicu tylko na każdym dziecku flexboxa.** To oznacza, że my moglibyśmy to ustawienie umieścić dla każdego selektora .box , a prościej:

```
body > * {  
    flex-shrink: 1;  
}
```

Na każdym dziecku znacznika body

**flex-shrink** to właściwość **flexboxa**, która określa, czy i jak bardzo element **może się zmniejszać**, gdy w kontenerze brakuje miejsca.

- **flex-shrink: 0** → element **nie kurczy się**, zostaje przy swojej szerokości/wysokości.

- **flex-shrink: 1 (domyślnie)** → element **może się kurczyć proporcjonalnie** do innych elementów, żeby zmieścić się w kontenerze.
- **flex-shrink: 2** → ten element będzie się kurczył **2x szybciej** niż element z **flex-shrink: 1**.

## ✓ Podsumowanie

- **flex-shrink** → decyduje **czy element może się ścinać**.
- **flex-wrap** → decyduje **czy elementy mogą się przerzucić do nowej linii**.

## Właściwość flex-direction

- **flex-direction: row;** // elementy są renderowane w wierszu
- **flex-direction: column;** // elementy są renderowane w kolumnie, od góry do dołu
- **flex-direction: row-reverse;** // elementy są renderowane w wierszu w przeciwej kolejności, czyli od prawej do lewej
- **flex-direction: column-reverse;** // elementy są renderowane w kolumnie od dołu do góry

## ◆ 1. Główna i poprzeczna os

W **Flexboxie** mamy zawsze **dwie osie**:

### 1. Oś główna (main axis)

👉 to wzdłuż niej układane są elementy flex-itemy.

- Kierunek osi zależy od właściwości **flex-direction**.

### 2. Oś poprzeczna (cross axis)

👉 to oś **prostopadła** do osi głównej.

- Służy m.in. do ustawiania elementów w pionie/horyzoncie zależnie od kierunku.

## ◆ 2. Kierunki osi

Decyduje o tym **flex-direction**:

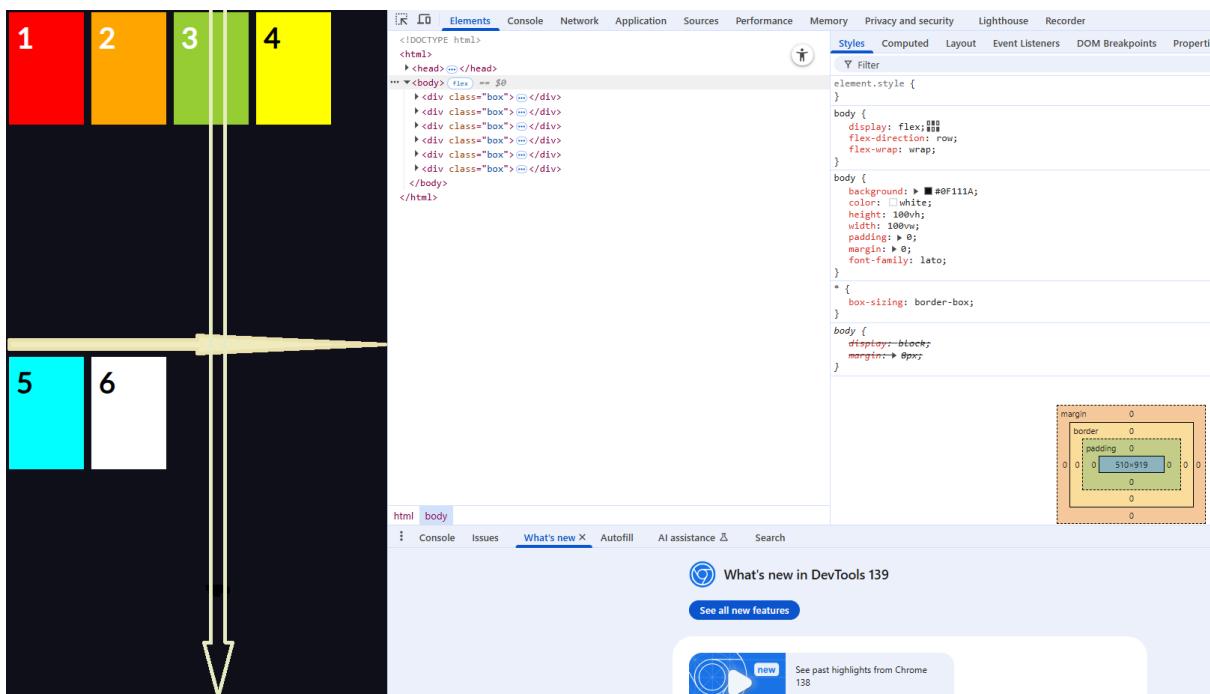
- **row** (domyślne)
  - **oś główna = pozioma** (z lewej do prawej)
  - **oś poprzeczna = pionowa** (z góry na dół)
- **row-reverse**
  - **oś główna = pozioma** (ale z prawej do lewej)
  - **oś poprzeczna = pionowa** (z góry na dół)
- **column**
  - **oś główna = pionowa** (z góry na dół)
  - **oś poprzeczna = pozioma** (z lewej do prawej)
- **column-reverse**
  - **oś główna = pionowa** (z dołu do góry)
  - **oś poprzeczna = pozioma** (z lewej do prawej)

## ◆ 3. Właściwości działające na osie

- **Wzdłuż osi głównej:**  
**justify-content** (ustawia, jak elementy są rozłożone na osi głównej).
- **Wzdłuż osi poprzecznej:**  
**align-items** (dla pojedynczej linii) i **align-content** (dla wielu linii, gdy **flex-wrap: wrap**).

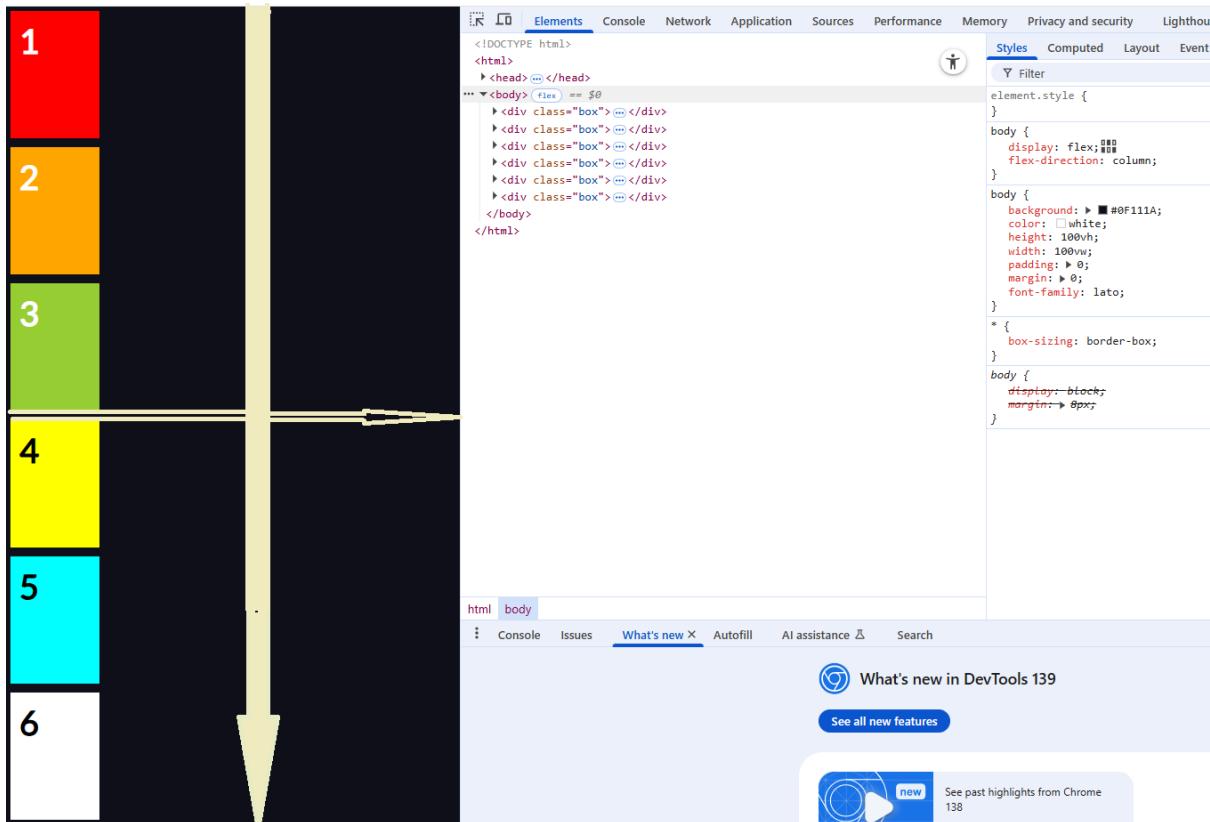
Przykład

```
body {  
  display: flex;  
  flex-direction: row;  
  flex-wrap: wrap;  
}
```



Zmiana osi głównej na oś pionową

```
body {
  display: flex;
  flex-direction: column;
}
```

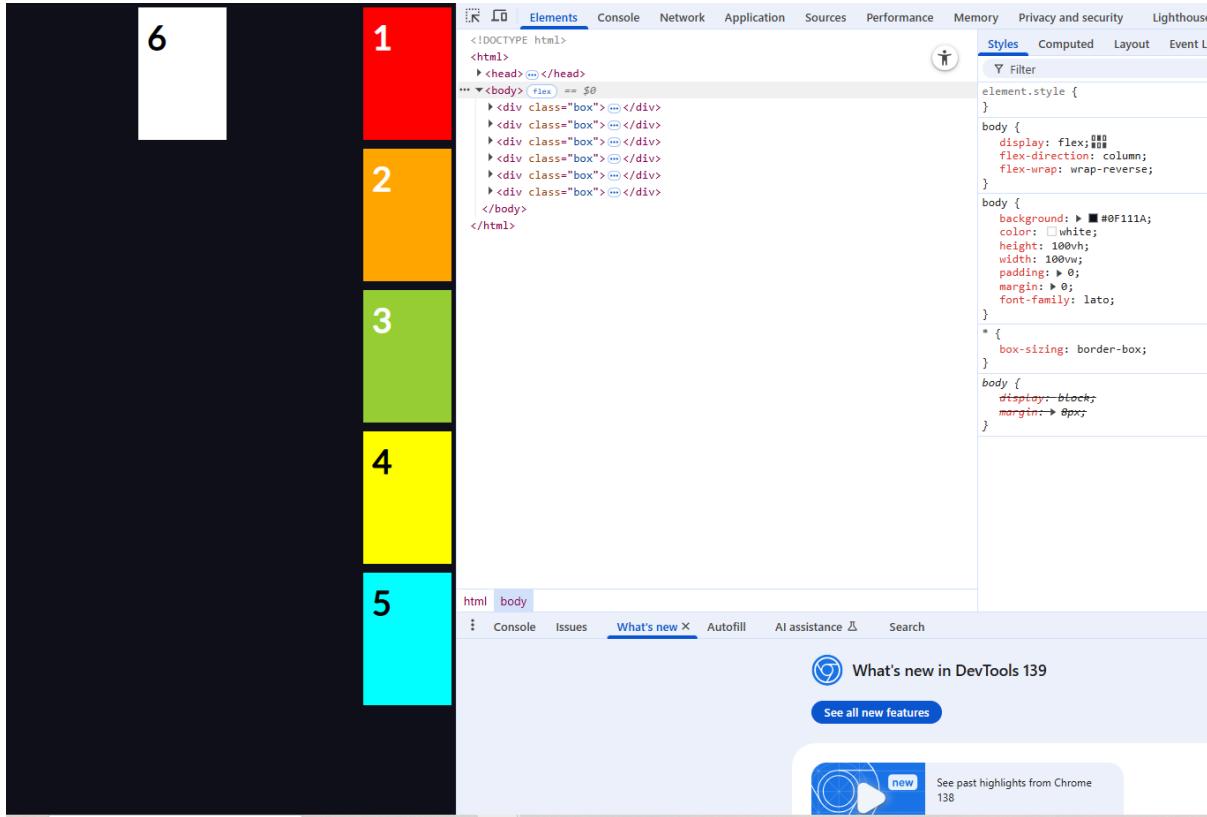


jak dodamy `flex-wrap: wrap` i zmiejszać widok w wysokości to elementy będą się układały wzdłuż osi poziomej

Dla `flex-wrap: wrap-reverse;`

```
body {
  display: flex;
  flex-direction: column;
  flex-wrap: wrap-reverse;
}
```

elementy będą się prezentować w następujący sposób



## 🎯 **justify-content** – odpowiada za rozmieszczenie elementów wzdłuż osi głównej

### 📌 Główne wartości:

1. **flex-start** (*domyślna*)  
👉 wszystkie elementy są „przyklejone” do początku osi głównej.
2. **flex-end**  
👉 wszystkie elementy są „przyklejone” do końca osi głównej.
3. **center**  
👉 elementy są wyśrodkowane na osi głównej.
4. **space-between**  
👉 pierwszy element przy początku, ostatni przy końcu, reszta równo rozłożona między nimi.

## 5. **space-around**

👉 elementy mają równe odstępy *dookoła*, więc na końcach zostaje po pół odstępu.

## 6. **space-evenly**

👉 elementy mają równe odstępy **wszędzie** (także po bokach).

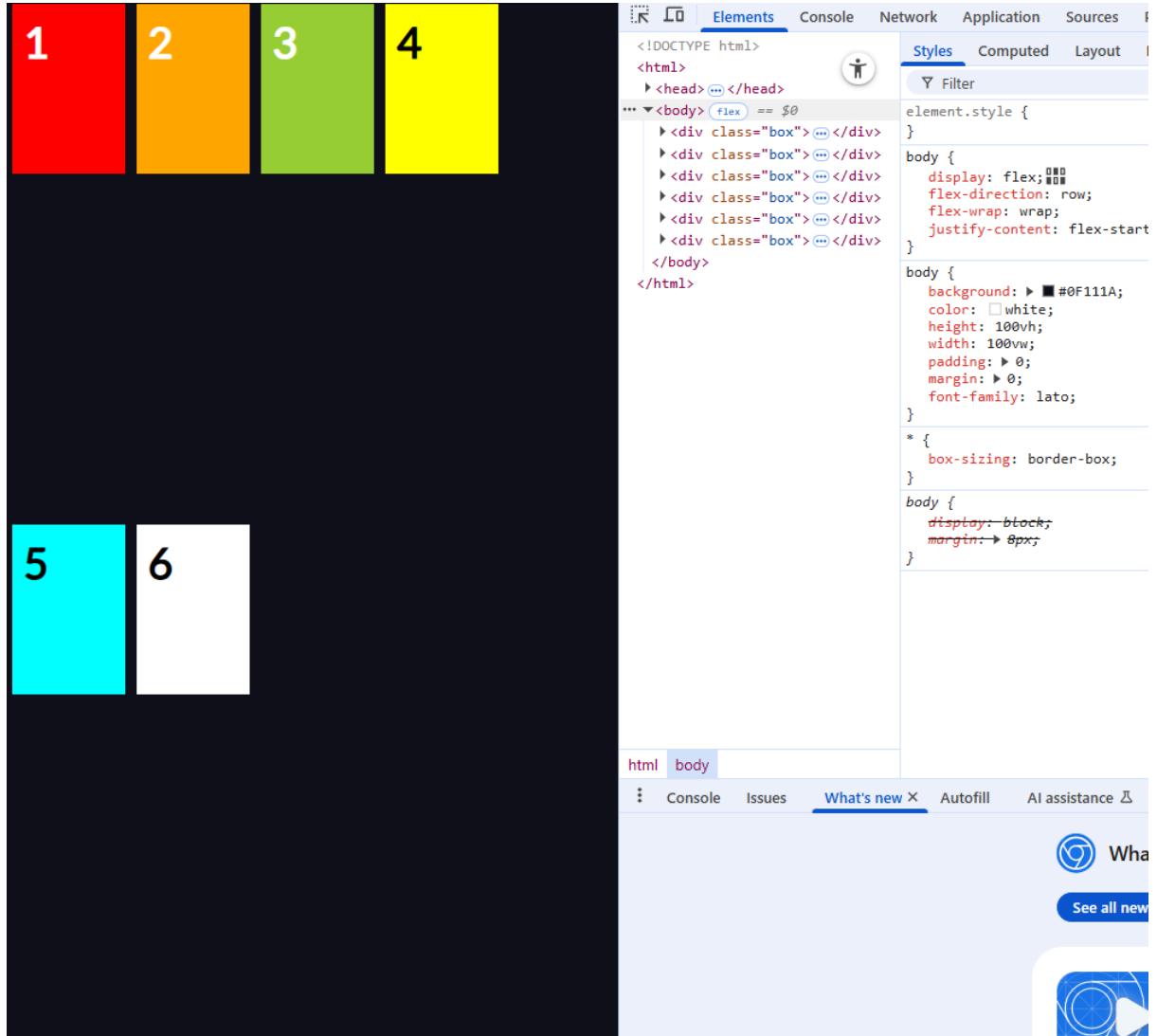


## Mała ściana

Wyobraź sobie, że masz 3 boxy w poziomie (**flex-direction: row**):

- **flex-start** → [ ]
- **flex-end** → [ ]
- **center** → [ ]
- **space-between** → [           ]
- **space-around** → [           ] (pół odstępu po bokach)
- **space-evenly** → [           ] (równe odstępy wszędzie)

```
body {  
  display: flex;  
  flex-direction: row;  
  flex-wrap: wrap;  
  justify-content: flex-start;  
}
```



```
body {
  display: flex;
  flex-direction: row;
  flex-wrap: wrap;
  justify-content: flex-end;
}
```

⌚ align-content?

- Rozmieszcza całe linie elementów (**flex-lines**) na osi poprzecznej (**cross axis**).
  - Działa tylko wtedy, gdy masz wiele linii flexboxa → czyli musisz mieć **flex-wrap: wrap** (albo **wrap-reverse**).
  - Jeśli wszystkie elementy mieszczą się w jednej linii → **align-content** nie ma żadnego efektu (wtedy używasz **align-items**).
- 

## 📌 Wartości **align-content**

### 1. **flex-start**

👉 wszystkie linie „przyklejone” do początku osi poprzecznej (np. do góry).

### 2. **flex-end**

👉 wszystkie linie „przyklejone” do końca osi poprzecznej (np. dół).

### 3. **center**

👉 wszystkie linie wyśrodkowane na osi poprzecznej.

### 4. **stretch (domyslna)**

👉 linie rozciągają się, żeby wypełnić całą przestrzeń na osi poprzecznej.

### 5. **space-between**

👉 pierwsza linia przy początku, ostatnia przy końcu, reszta rozłożona równomiernie między nimi.

### 6. **space-around**

👉 linie mają równe odstępy *dookoła*, więc przy krawędziach zostaje po pół odstępu.

## 7. space-evenly

👉 linie mają równe odstępy wszędzie (także na brzegach).

---



### Różnica między align-items a align-content

- **align-items** → wyrównuje elementy wewnątrz jednej linii.
  - **align-content** → rozkłada całe linie względem siebie.
- 



### Przykład

Wyobraź sobie kontener wysoki na 400px, z **flex-wrap: wrap**, gdzie elementy tworzą 2 linie:

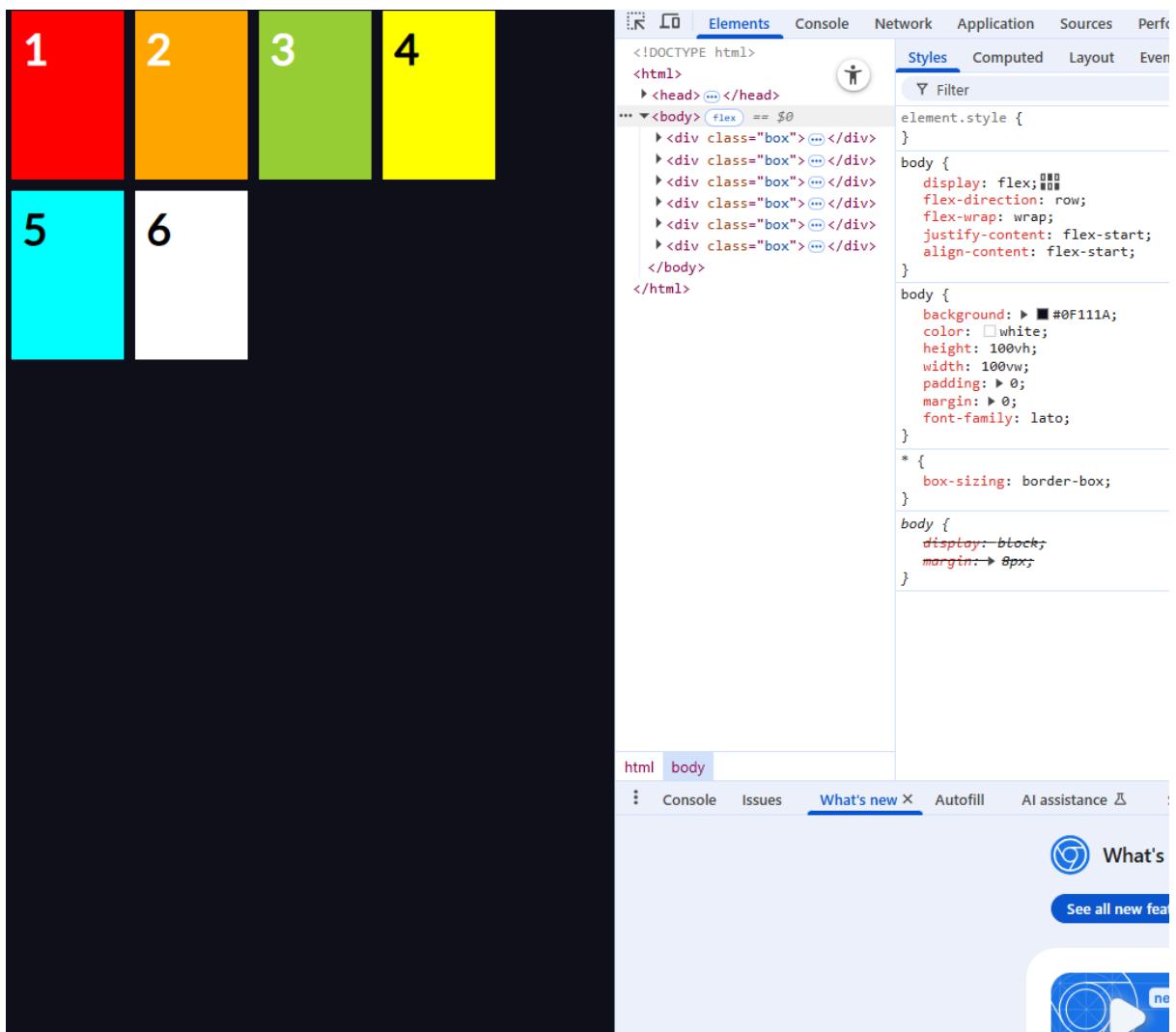
- **align-content: flex-start** → obie linie na górze.
  - **align-content: flex-end** → obie linie na dole.
  - **align-content: center** → obie linie wyśrodkowane.
  - **align-content: space-between** → jedna linia na górze, druga na dole.
  - **align-content: space-around** → odstępy równomierne, trochę wolnej przestrzeni też przy górze i dole.
  - **align-content: space-evenly** → odstępy równe wszędzie, także na krawędziach.
- 



### Podsumowanie

- **align-items** → wyrównanie elementów w jednej linii (cross axis).
- **align-content** → wyrównanie wielu linii jako całości (cross axis).

```
body {
  display: flex;
  flex-direction: row;
  flex-wrap: wrap;
  justify-content: flex-start;
  align-content: flex-start;
}
```



## ⌚ align-items

- Ustawia **jak dzieci (flex items) są wyrównane względem osi poprzecznej (cross axis)**.
- Czyli **prostopadle** do kierunku ułożenia (**flex-direction**).

👉 Jeśli **flex-direction: row** (osią główną poziomą), to **align-items** ustawia elementy **w pionie**.

👉 Jeśli **flex-direction: column** (osią główną pionową), to **align-items** ustawia elementy **w poziomie**.

## 📌 Główne wartości align-items:

1. **stretch** (*domyślna*)  
👉 elementy rozciągają się, aby wypełnić całą wysokość kontenera (dla **row**) albo szerokość (dla **column**).
2. **flex-start**  
👉 elementy „przyklejone” do początku osi poprzecznej (góra, jeśli **row**).
3. **flex-end**  
👉 elementy „przyklejone” do końca osi poprzecznej (dół, jeśli **row**).
4. **center**  
👉 elementy wyśrodkowane na osi poprzecznej.
5. **baseline**  
👉 elementy są wyrównane do **linii bazowej tekstu**. Przydatne, jeśli masz np. różnej wysokości boxy z tekstem.



## Przykład (dla flex-direction: row)

Masz 3 boxy w kontenerze wysokości 200px:

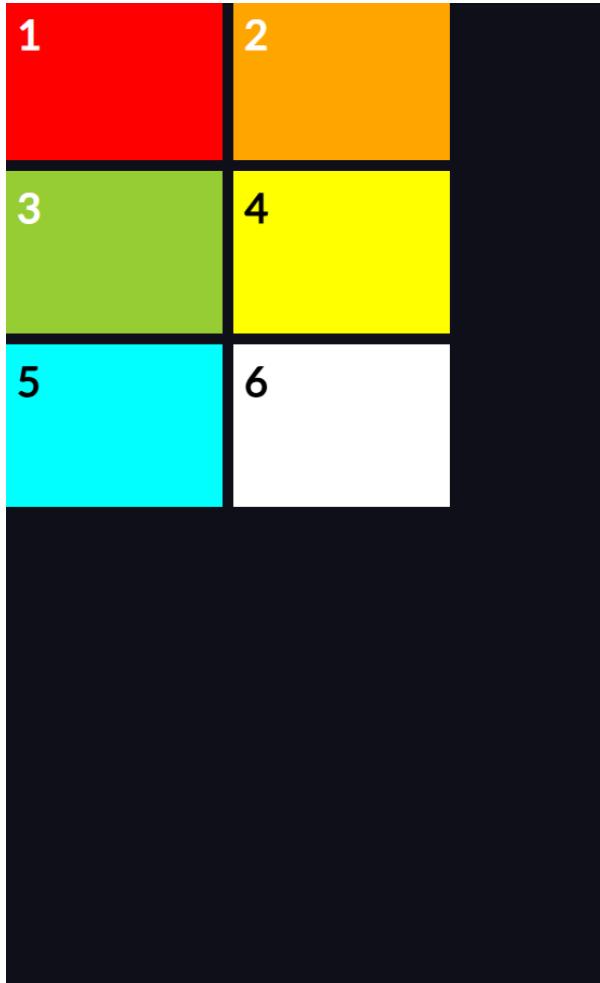
- `align-items: stretch` →  (wszystkie rozciągają się na 200px)
- `align-items: flex-start` →   
  
  
(wszystkie przy górze)
- `align-items: flex-end` →  
  
  
  
(wszystkie przy dole)
- `align-items: center` →  
  
  
  
(wszystkie na środku pionowo)
- `align-items: baseline` →  (ustawione wg linii tekstu, więc np. podpisy w jednej linii)

```
body {
  display: flex;
  flex-direction: row;
  flex-wrap: wrap;
  justify-content: flex-start;
  align-items: flex-start;
}
```

flex-basis ustawiamy na dziecku flexboxa. Czyli w naszym przypadku na `.box` domyślna wartość `flex-basis` to `auto`; Generalnie `flex-basis` to takie width i height dla elementów flexboxa

Ustawione wcześniej szerokość i wysokość przegrywają się napisane przez `flex-basis` (ale `flex-basis` nie jest ustawione na `auto`)

```
.box {
  flex-basis: 200px;
}
```



Elements Console Network Application Sources Performance

Styles Computed Layout Event Listener

Filter

```
<!DOCTYPE html>
<html>
  <head> ... </head>
  <body> (flex)
    > <div class="box"> ... </div>
    > <div class="box"> ... </div>
    > ...
    > <div class="box"> ... </div> == $0
    > <div class="box"> ... </div>
    > <div class="box"> ... </div>
    > <div class="box"> ... </div>
</body>
</html>
```

.box:nth-child(3) {  
 background: ▶ yellowgreen;  
}  
.box {  
 flex-basis: 200px;  
}  
.box {  
 width: 100px;  
 height: 150px;  
 background: ▶ red;  
 padding: ▶ 10px;  
 margin: ▶ 5px;  
}  
body > \* {  
 flex-shrink: 1;  
}  
\* {  
 box-sizing: border-box;  
}  
div {  
 display: block;  
 unicode-bidi: isolate;  
}

Inherited from body

```
body {  
  background: ▶ #0F111A;  
  color: □ white;  
  height: 100vh;  
  width: 100vw;  
  padding: ▶ 0;  
  margin: ▶ 0;  
  font-family: lato;  
}
```

html body div.box

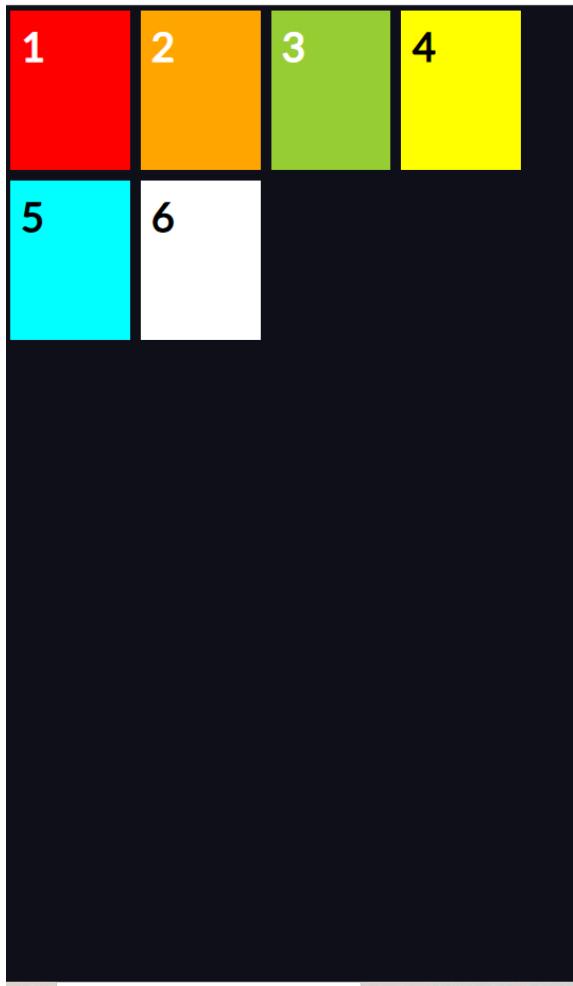
Console Issues What's new Autofill AI assistance Search

What's new in DevTools

See all new features

New See past 138

```
.box {  
  flex-basis: 20%;  
}
```



Elements    Console    Network    Application    Sources    Performance    More

Styles    Computed    Layout    Event List

Filter

```
<!DOCTYPE html>
<html>
  <head> ... </head>
  <body> flex
    <div class="box"> ... </div>
    <div class="box"> ... </div>
    <div class="box"> ... </div>
    <div class="box"> = $0
      <span class="box_span">3</span>
    </div>
    <div class="box"> ... </div>
    <div class="box"> ... </div>
  </body>
</html>
```

Inherited from body

```
body {
  background: #0F111A;
  color: white;
  height: 100vh;
  width: 100vw;
  padding: 0;
  margin: 0;
  font-family: lato;
```

html    body    div.box

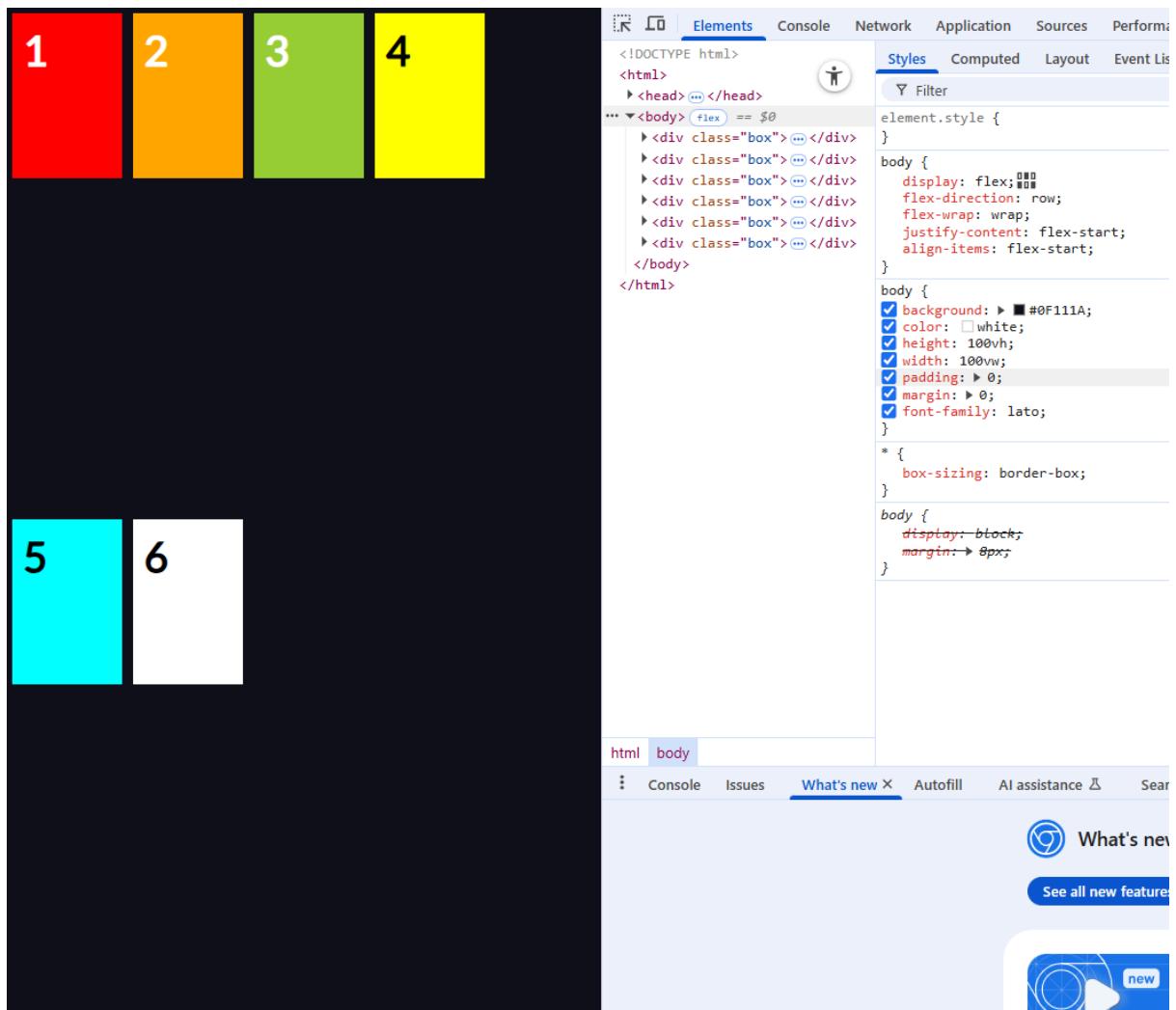
Console    Issues    What's new    Autofill    AI assistance    Search

What's new in Dev

See all new features



See past 138



```
body {  
  display: flex;  
  flex-direction: row;  
  flex-wrap: wrap;  
  justify-content: flex-start;  
  align-content: flex-start;  
}  
  
body > * {  
  flex-shrink: 1;  
}  
  
.box {  
  flex-basis: 20px;  
}
```

szerokość będzie ustawiona na 20px bo flex-direction: row; Jeśli zmienimy na flex-direction: column to wysokość będzie ustawiona na 20px; a szerokość będzie brana z ustawień wcześniejszych

## Lekcja

### Temat: Zadania z testów

#### 🔥 1. Pogrubienie (bold)

 **Semantyczne** — mają znaczenie  
<strong>

Oznacza **ważny tekst**.

<strong>Uwaga! To jest ważne.</strong>

<b>

Pogrubienie **bez znaczenia semantycznego** — tylko wizualne.

<b>Zwykłe pogrubienie bez znaczenia</b>

#### 🔥 2. Pochylenie (italic)

 **Semantyczne**  
<em> (emphasis)

Oznacza **akcentowanie, podkreślenie znaczenia** — np. nacisk w zdaniu.

<em>To jest naprawdę ważne.</em>

#### Niesemantyczne

<i>

Tylko **kursywa** — czysto wizualna (np. cytaty, nazwy obce, ikony).

<i>Cursus vitae</i>

#### 🔥 3. Podkreślenie

 **Niesemantyczne** (tylko wygląd)

<u>

Podkreśla tekst.

<u>Podkreślony tekst</u>

#### Semantyczne

<ins>

Oznacza **tekst dodany**, zwykle podkreślony.

<ins>Nowo dodany fragment.</ins>

#### 🔥 4. Przekreślenie

 **Niesemantyczne**

<s>

Przekreślenie dla tekstu, który **stracił ważność**.

<s>Stara cena: 100 zł</s>

### **Semantyczne**

<del>

Oznacza **tekst usunięty** (delete).

<del>Ten fragment został usunięty.</del>

## **5. Wyróżnienie kolorystyczne**

<mark> **(SEMANTYCZNE)**

Zaznacza tekst jak zakreślaczem.

<mark>Ważny fragment</mark>

## **6. Cytaty i odniesienia**

<q> — **cytat krótki (cudzysłów)**

<q>Być albo nie być</q>

<blockquote> — **cytat długi**

<blockquote>

To jest dłuższy cytat.

</blockquote>

<cite> — **tytuł dzieła**

<cite>Pan Tadeusz</cite>

### ♦ **Różnica między blokowymi a liniowymi (inline)**

- **Blokowe:** <div>, <p>, <h1> → zajmują całą szerokość, nowa linia
- **Inline:** <span>, <a>, <strong> → mieszczą się w jednym wierszu, nie łamią linii

### ♦ **Dostępne style border-style**

<b>Wartość</b>	<b>Opis</b>
----------------	-------------

**solid** ciągła linia

**dashed** linia kreskowana

**dotted** linia kropkowana

**double** podwójna linia

**groove** wcięcie (3D)

**ridge** wypukła (3D)

<b>inset</b>	wcięcie
<b>outset</b>	wypukła
<b>none</b>	brak obramowania
<b>hidden</b>	ukryte obramowanie