6.1 6.1 Teoria

Step 1

Teraz nauczymy się dwóch ważnych narzędzi układu strony `CSS` : - `Flexbox` dla większej kontroli nad układaniem elementów wzdłuż jednej osi - `Grid` dla prawdziwych układów opartych na siatce, jak te, których projektanci druku używali od dziesięcioleci. Każde z tych narzędzi ma swoje specjalne przeznaczenie, ale możemy ich używać razem aby osiągnąć układy, o których do tej pory tylko marzyliśmy. Na przykład, możemy stworzyć ogólną strukturę strony za pomocą siatki i użyć `flexboxa` do okiełznania nagłówka i elementów nawigacyjnych. Używajmy każdej z technik do tego, do czego najlepiej się nadaje - nie musimy wybierać tylko jednej. nie musimy wybierać tylko jednej. Teraz, gdy przeglądarki zaczęły wspierać te techniki, projektanci i programiści programiści mają prawdziwe możliwości osiągnięcia wyrafinowanych układów z wbudowaną elastycznością potrzebną do radzenia sobie z szeroką gamą rozmiarów ekranu. Gdy stare przeglądarki znikną z użycia, możemy pożegnać się z naszymi starymi hackami do układów typu `float` .

Specyfikacja jest przepełniona opcjami i nowymi koncepcjami, które wymagają wyjaśnienia i przykładów. To dużo, aby spakować w umyśle wszystko na raz, więc polecam spędzić trochę czasu, aby uzyskać zrozumienie tych technik indywidualnie.

Moduł `CSS` **Flexible Box Layout** (znany również jako po prostu `Flexbox`) daje projektantom i programistom poręczne narzędzie do układania elementów stron internetowych takich jak paski menu, listy produktów, galerie i wiele innych.

Zgodnie ze specyfikacją, definiującym aspektem układu `flex` jest możliwość uczynienia elementów `flex` "elastycznymi", zmieniającymi swoją szerokość/wysokość, aby wypełnić dostępną przestrzeń w głównym wymiarze.

Oznacza to, że umożliwia rozciąganie lub kurczenie się elementów wewnątrz ich pojemników, zapobiegając marnowaniu miejsca i przepełnieniu - to prawdziwa zaleta przy dopasowywaniu układów do różnych rozmiarów rzutni. Inne zalety to między innymi:

- Możliwość nadania wszystkim sąsiadującym elementom tej samej wysokości
- Łatwe centrowanie w poziomie i w pionie (co było nieuchwytne przy starych metodach `CSS`)
- Możliwość zmiany kolejności wyświetlania elementów, niezależnie od źródła

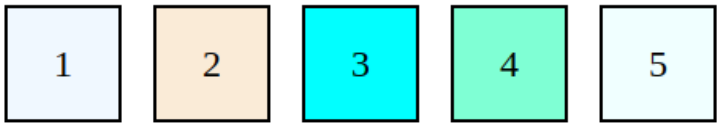
Model układu `Flexbox` jest niezwykle solidny, ale ponieważ został zaprojektowany z myślą o maksymalnej elastyczności, potrzeba trochę czasu, aby go zrozumieć. Kiedy mówimy elementowi, aby stał się `flexboxem` , **wszystkie jego elementy dziecięce ustawiają się w jednej osi, jak koraliki na sznurku**. Sznurek może być poziomy, może wisieć w pionie lub może nawet zawijać się na wiele linii, ale koraliki są zawsze na jednym sznurku (lub, używając właściwego terminu, na jednej osi). Jeśli chcemy ustawić rzeczy zarówno w poziomie, jak i w pionie, to jest to zadanie `CSS Grid` , które przedstawimy później.

Zanim się za to zabierzemy, należy krótko powiedzieć o wsparciu dla przeglądarek. Wszystkie obecne wersje przeglądarek obsługują najnowszą specyfikację **W3C Flexible Box Layout Module**; jednak starsze przeglądarki wymagają przedrostków, a nawet innych, przestarzałych właściwości i wartości. Będziemy trzymać się obecnych standardowych właściwości, aby wszystko było proste, gdy będziemy się tego uczył po raz pierwszy, ale należy wiedzieć, że gotowe do produkcji arkusze stylów mogą wymagać więcej kodu.

Poznaliśmy już tryb układu blokowego (`block`) do układania elementów w normalny przepływ oraz tryb `inline` do wyświetlania treści w jego obrębie w poziomie. `Flexbox` to kolejny tryb układu z własnymi zachowaniami. Aby włączyć tryb `flexbox` dla elementu, należy ustawić jego właściwość `display` na `flex` lub `inline- flex` . Element ten jest teraz kontenerem `flex` , a wszystkie jego bezpośrednie elementy potomne (niezależnie od tego, czy są to `div` , elementy listy, akapity itp.) automatycznie stają się elementami `flex` w tym kontenerze. Elementy `flex` są układane i wyrównywane wzdłuż linii `flex` .

Poniższy kod pokazuje efekt prostego dodania `display: flex` do `div` , a więc włączenia przełącznika `Flexbox` . Do kontenera dodano style, aby jego granice były wyraźne.

```
<!--flexbox_1_example.html-->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Flexbox one example</title>
</head>
<style>
    #container{
        display: flex;
    }
    .box {
        border: 2px solid;
        font-size: 18pt;
        text-align: center;
        line-height: 50px;
        padding: 10px 10px 10px 10px;
        margin: 10px;
        width: 50px;
        height: 50px;
    }
    .box1 {
        background-color: aliceblue;
    }
    .box2 {
        background-color: antiquewhite;
    }
    .box3 {
        background-color: aqua;
    }
    .box4 {
        background-color: aquamarine;
    }
    .box5 {
        background-color: azure;
    }
</style>
<body>
<div id="container">
    <div class="box box1">1</div>
    <div class="box box2">2</div>
    <div class="box box3">3</div>
    <div class="box box4">4</div>
    <div class="box box5">5</div>
</div>
</body>
</html>
```



Widać, że elementy ustawiły się w rzędzie od lewej do prawej, co jest domyślnym zachowaniem `Flexbox`, jeśli strona jest w języku angielskim lub innym języku napisanym w rzędach od lewej do prawej. Dzieje się tak dlatego, że domyślny kierunek `Flexbox` odpowiada kierunkowi języka, w którym napisana jest strona. W przypadku języka hebrajskiego lub arabskiego lub kolumn, jeśli strona jest ustawiona w pionowym kierunku pisania, `flexbox` byłby domyślnie ustawiony od prawej do lewej. Ponieważ nie jest to związane z jednym domyślnym kierunkiem, terminologia określająca kierunki ma tendencję do bycia nieco abstrakcyjną. Warto zauważyć, że można przekształcić dowolny element `flex` w kontener `flex`, ustawiając właściwość `display` na `flex`, co spowoduje powstanie zagnieżdżonego `flexboxa`.

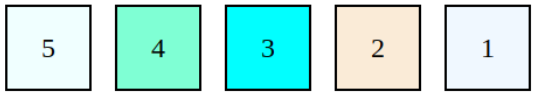
Po przekształceniu elementu w kontener `flex`, istnieje kilka właściwości, które można które można ustawić na tym kontenerze, aby kontrolować przepływ elementów w nim. Przepływ odnosi się do kierunku, w którym układane są elementy `flex`, a także do tego, czy one mogą być zawijane na dodatkowe linie.

Możemy być zadowoleni z elementów ustawionych w rzędzie, jak pokazano na rysunku powyżej, ale istnieje kilka innych opcji, które są kontrolowane za pomocą właściwości `flex-direction`.

```
flex-direction

Wartości: row | column | row-reverse | column-reverse
Domyślnie: row
Dotyczy: flex containers
Dziedziczy: nie
```

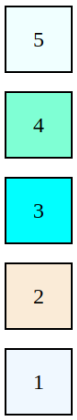
Domyślną wartością jest `row`, jak widzieliśmy w poprzednim przykładzie. Możemy również określić, że elementy mają być wyrównane pionowo w kolumnie. Pozostałe opcje, `row-reverse` i `column-reverse`, układają elementy w kierunku, w jakim byśmy się spodziewali, ale zaczynają się one na końcu i są wypełniane w przeciwnym kierunku. Przykład poniżej przedstawia efekty działania każdego w naszym prostym przykładzie.

`flex-direction: row-reverse;`



`flex-direction: column;`

`flex-direction: column-reverse;`

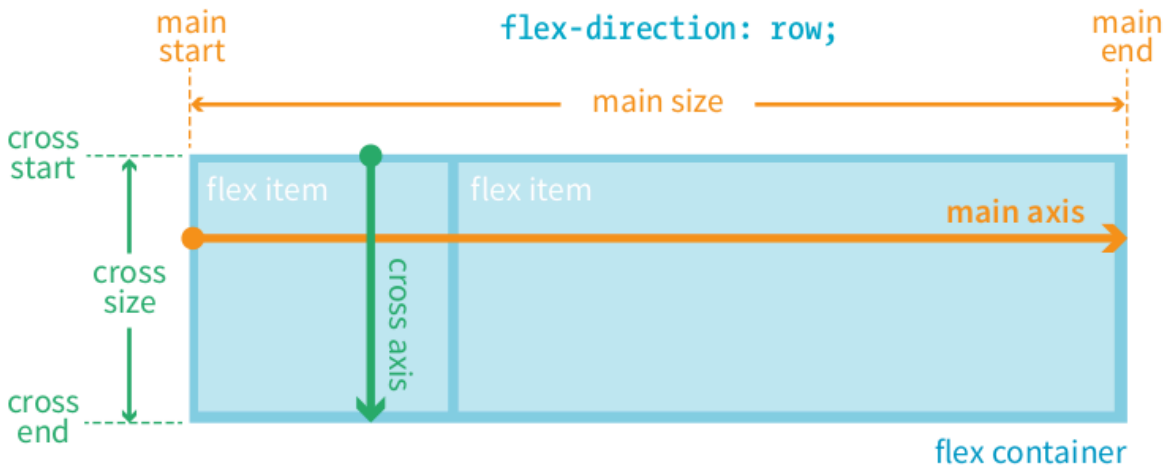


Teraz, gdy widzieliśmy `Flexbox` w akcji, nadszedł dobry moment, aby zapoznać się z formalną terminologią `Flexbox`. Ponieważ system ten jest bezkierunkowy, w wartościach właściwości nie ma odniesień do `"left"`, `"right"`, `"top"` czy `"bottom"`. Zamiast tego mówimy o **osi głównej** (ang. `main axis`) i **osi poprzecznej** (ang. `cross axis`). **Oś główna** to kierunek przepływu, który został określony dla kontenera `flex`. Dla języków głównie poziomych, gdy ustawiony jest na `roe`, główna oś jest pozioma; dla `column`, główna oś jest pionowa. **Oś poprzeczna** to kierunek prostopadły do osi głównej (pionowy dla rzędu, poziomy dla kolumny).

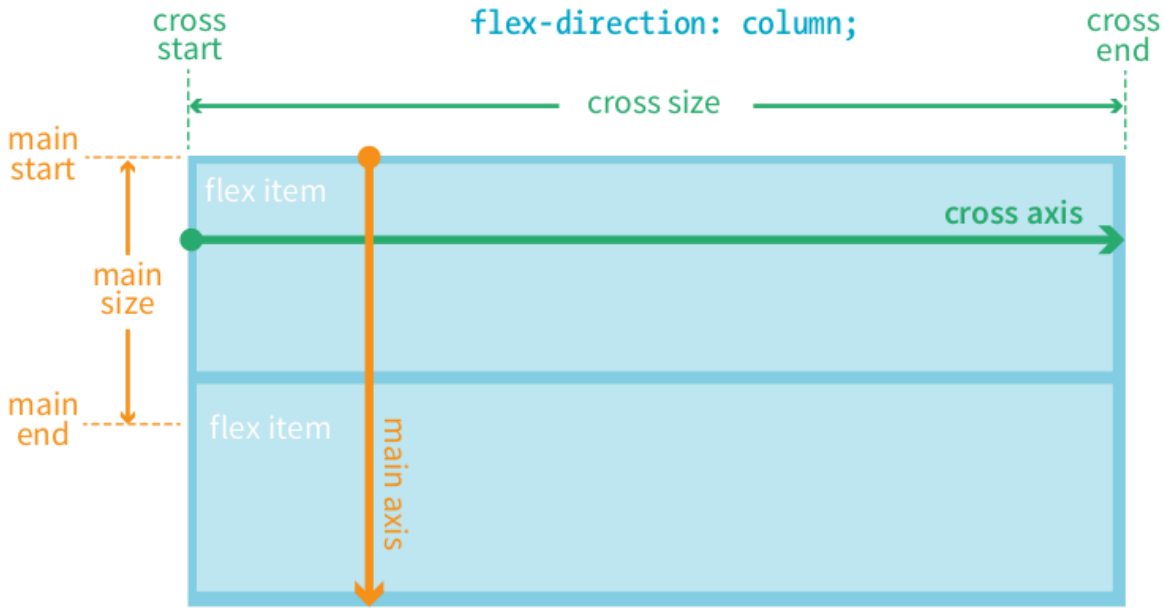
Oprócz osi, zrozumienie innych części systemu `Flexbox` ułatwia naukę właściwości. Zarówno oś główna, jak i oś poprzeczna mają początek i koniec, w oparciu o kierunek, w którym przepływają elementy. **Rozmiar główny** (ang. `main size`) to szerokość (lub wysokość, jeśli jest to kolumna) kontenera wzdłuż osi głównej, a **rozmiar poprzeczny** (ang. `cross size`) to wysokość (lub szerokość, jeśli jest to kolumna) wzdłuż osi poprzecznej.

FOR LANGUAGES THAT READ HORIZONTALLY FROM LEFT TO RIGHT:

When **flex-direction** is set to **row**, the main axis is horizontal and the cross axis is vertical.



When **flex-direction** is set to **column**, the main axis is vertical and the cross axis is horizontal.



Jeśli mamy dużą lub nieznaną liczbę elementów `flex` w kontenerze i nie chcemy, aby wszystkie zostały zgniecione w dostępnej przestrzeni, możemy pozwolić im przełamać się na dodatkowe linie za pomocą właściwości `flex-wrap`.

```
flex-wrap

Wartości: nowrap | wrap | wrap-reverse
Domyślnie: nowrap
Dotyczy: flex containers
Dziedziczy: nie
```

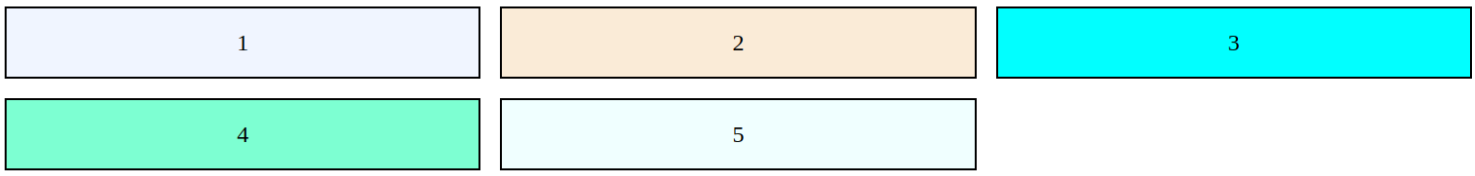
Domyślnie, elementy robią to co do nich należy i nie zawijają się na dodatkowe linie (`nowrap`). Słowo kluczowe `wrap` włącza możliwość zawijania na wiele linii w kierunku od początku do końca krzyża. Na przykład, jeśli kierunkiem jest rząd, to linie są pozycjonowane od góry w dół.

`wrap-reverse` rozbija elementy na wiele linii, ale przepływa je w przeciwnym kierunku, od `cross end` do `cross start` (w tym przypadku od dołu do góry).

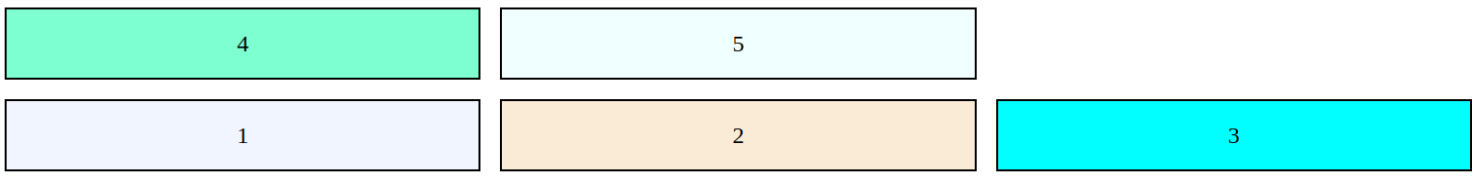
Do naszego numerowanego przykładu `flexbox` dodajmy więcej divów i nadajmy elementom `flex` szerokość `25%`, tak aby tylko cztery zmieściły się na całej szerokości kontenera. Poniżej zobacyzmy porównanie różnych opcji zawijania, gdy `flex-direction` jest domyślnie ustawiony na `row`.

```
<!--flexbox_2_example.html-->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Flexbox one example</title>
</head>
<style>
  #container{
    display: flex;
    flex-direction: row;
    flex-wrap: wrap;
  }
  .box {
    border: 2px solid;
    font-size: 18pt;
    text-align: center;
    line-height: 50px;
    padding: 10px 10px 10px 10px;
    margin: 10px;
    width: 25%;
  }
  .box1 {
    background-color: aliceblue;
  }
  .box2 {
    background-color: antiquewhite;
  }
  .box3 {
    background-color: aqua;
  }
  .box4 {
    background-color: aquamarine;
  }
  .box5 {
    background-color: azure;
  }
</style>
<body>
<div id="container">
  <div class="box box1">1</div>
  <div class="box box2">2</div>
  <div class="box box3">3</div>
  <div class="box box4">4</div>
  <div class="box box5">5</div>
</div>
</body>
</html>
```

`flex-wrap: wrap;`



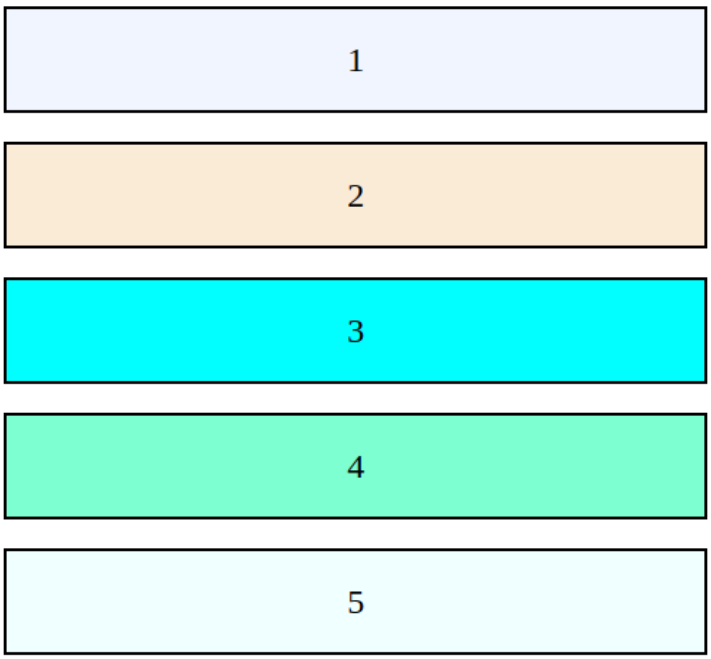
`flex-wrap: wrap-reverse;`



`flex-wrap: nowrap;`

Domyślnie, gdy `flex-direction` jest ustawiony na kolumnę, kontener rozszerza się tak, aby zawierał wysokość elementów. Aby zobaczyć działanie zawijania, należy ustawić wysokość kontenera jak zrobiono poniżej Na rysunku pokazano, jak działa zawijanie dla każdego ze słów kluczowych `flex-wrap`. Zauważmy, że elementy nadal mają `25%` szerokości swojego kontenera nadrzędnego, więc pomiędzy kolumnami pozostało miejsce.

```
#container {
  display: flex;
  height: 350px;
  flex-direction: column;
  flex-wrap: nowrap;
}
```




```
#container{
    display: flex;
    height: 350px;
    flex-direction: column;
    flex-wrap: wrap;
  }
```


```
#container{
    display: flex;
    height: 350px;
    flex-direction: column;
    flex-wrap: wrap-reverse;
  }
```

Skrócona właściwość `flex-flow` sprawia, że określanie `flex-direction` i `flex-wrap` jest krótkie. Pominięcie jednej wartości skutkuje wartością domyślną dla odpowiedniej właściwości, co oznacza, że możemy użyć `flex-flow` dla obu kierunków i zawijania elementów.

```
flex-flow

Wartości: flex-direction flex-wrap
Domyślnie: row nowrap
Dotyczy: flex containers
Dziedziczy: nie
```

Używając `flex-flow`, możemy skrócić poprzedni przykład w taki sposób:

```
<!--flexbox_3_example.html-->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Flexbox one example</title>
</head>
<style>
  #container{
    display: flex;
    height: 350px;
    flex-flow: column wrap
  }
  .box {
    border: 2px solid;
    font-size: 18pt;
    text-align: center;
    line-height: 50px;
    padding: 10px 10px 10px 10px;
    margin: 10px;
    width: 25%;
  }
  .box1 {
    background-color: aliceblue;
  }
  .box2 {
    background-color: antiquewhite;
  }
  .box3 {
    background-color: aqua;
  }
  .box4 {
    background-color: aquamarine;
  }
  .box5 {
    background-color: azure;
  }
</style>
<body>
<div id="container">
  <div class="box box1">1</div>
  <div class="box box2">2</div>
  <div class="box box3">3</div>
  <div class="box box4">4</div>
  <div class="box box5">5</div>
</div>
</body>
</html>
```

Step 2

Do tej pory widzieliśmy, jak włączyć tryb `flexbox`, zamieniając element w kontener `flex`, a jego dzieci w elementy `flex`. Dowiedzieliśmy się również, jak zmienić kierunek przepływu elementów i pozwolić im zawijać się na wiele linii. Pozostały zestaw właściwości kontenera wpływa na wyrównanie elementów wzdłuż osi głównej (`justify-content`) i osi poprzecznej (`align-items` i `align-content`).

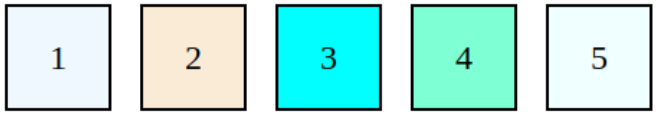
Domyślnie elementy `flex` mają taką szerokość, jaka jest potrzebna do zawarcia zawartości elementu, co oznacza, że kontener może potencjalnie mieć wolne miejsce na linii `flex`. Widzieliśmy to na poprzednich przykładach. Również domyślnie elementy przepływają tuż obok siebie od "głównego początku" (w zależności od kierunku języka i kierunku linii `flex`).

Właściwość `justify-content` określa, w jaki sposób powinno być rozmieszczone dodatkowe miejsce wokół lub pomiędzy elementami, które są nieelastyczne lub osiągnęły swój maksymalny rozmiar.

```
justify-content

Wartości: flex-start | flex-end | center | space-between | space-around
Domyślnie: flex-start
Dotyczy: flex containers
Dziedziczy: nie
```

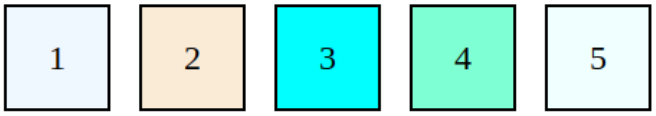
Zastosujmy `justify-content` do elementu `flex`, ponieważ kontroluje on odstępy wewnątrz samego kontenera:

```
<!--flexbox_4_example.html-->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Flexbox one example</title>
</head>
<style>
  #container {
    display: flex;
    justify-content: flex-start;
  }
  .box {
    border: 2px solid;
    font-size: 18pt;
    text-align: center;
    line-height: 50px;
    padding: 10px 10px 10px 10px;
    margin: 10px;
    width: 50px;
  }
  .box1 {
    background-color: aliceblue;
  }
  .box2 {
    background-color: antiquewhite;
  }
  .box3 {
    background-color: aqua;
  }
  .box4 {
    background-color: aquamarine;
  }
  .box5 {
    background-color: azure;
  }
</style>
<body>
<div id="container">
  <div class="box box1">1</div>
  <div class="box box2">2</div>
  <div class="box box3">3</div>
  <div class="box box4">4</div>
  <div class="box box5">5</div>
</div>
</body>
</html>
```

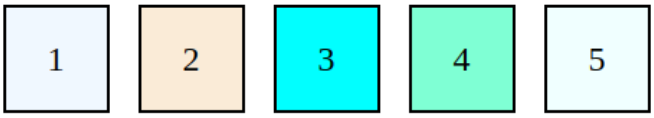


Jak można się spodziewać, `flex-start` i `flex-end` pozycjonują linię elementów odpowiednio w kierunku początku i końca osi głównej, a `center` je wyśrodkowuje.

`justify-content: flex-end;`

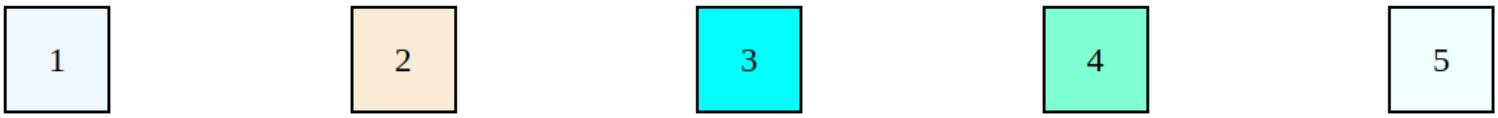


`justify-content: center;`

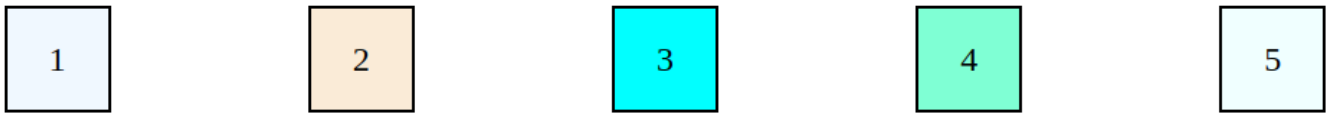


`space-between` i `space-around` zasługują na nieco więcej wyjaśnień. Gdy `justify-content` jest ustawione na `space-between`, pierwszy element jest umieszczany w punkcie początkowym, ostatni element w punkcie końcowym, a pozostała przestrzeń jest rozdzielana równomiernie pomiędzy pozostałe elementy. Właściwość `space-around` dodaje równą ilość miejsca po lewej i prawej stronie każdego elementu, co powoduje podwojenie przestrzeni między sąsiednimi elementami.

`justify-content: space-between;`



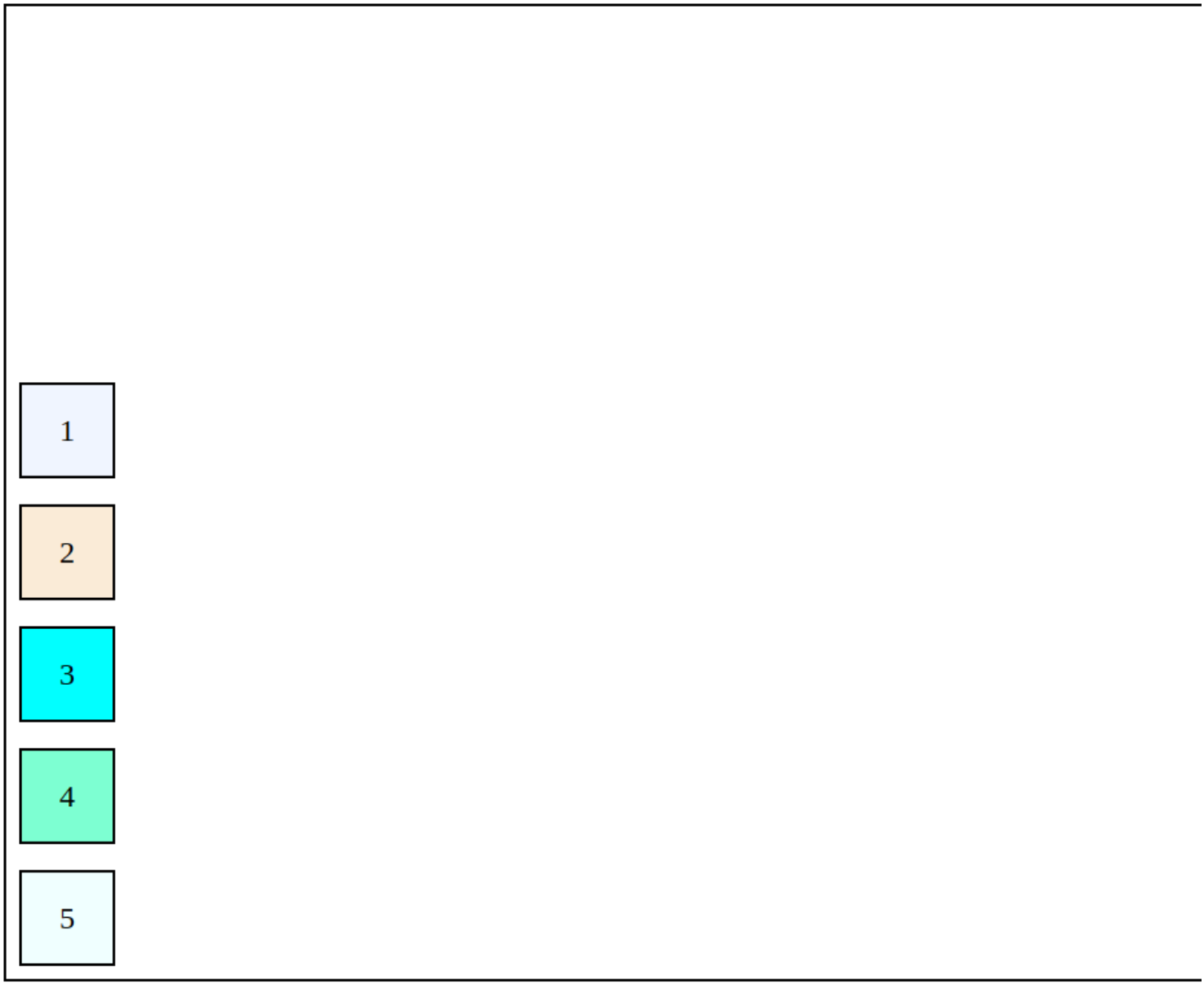
`justify-content: space-around;`



Gdy kierunek jest ustawiony na kolumnę z pionową osią główną, słowa kluczowe działają w ten sam sposób, jednak aby zobaczyć efekt, musi istnieć wyraźna wysokość kontenera z pozostawionym miejscem.

```
<!--flexbox_5_example.html-->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Flexbox five example</title>
</head>
<style>
  #container {
    display: flex;
    flex-direction: column;
    justify-content: flex-start;
    height: 750px;
    border: 2px solid;
  }
  .box {
    border: 2px solid;
    font-size: 18pt;
    text-align: center;
    line-height: 50px;
    padding: 10px 10px 10px 10px;
    margin: 10px;
    width: 50px;
  }
  .box1 {
    background-color: aliceblue;
  }
  .box2 {
    background-color: antiquewhite;
  }
  .box3 {
    background-color: aqua;
  }
  .box4 {
    background-color: aquamarine;
  }
  .box5 {
    background-color: azure;
  }
</style>
<body>
<div id="container">
  <div class="box box1">1</div>
  <div class="box box2">2</div>
  <div class="box box3">3</div>
  <div class="box box4">4</div>
  <div class="box box5">5</div>
</div>
</body>
</html>
```


```
justify-content: flex-end;
```




```
justify-content: center;
```


```
justify-content: space-between;
```

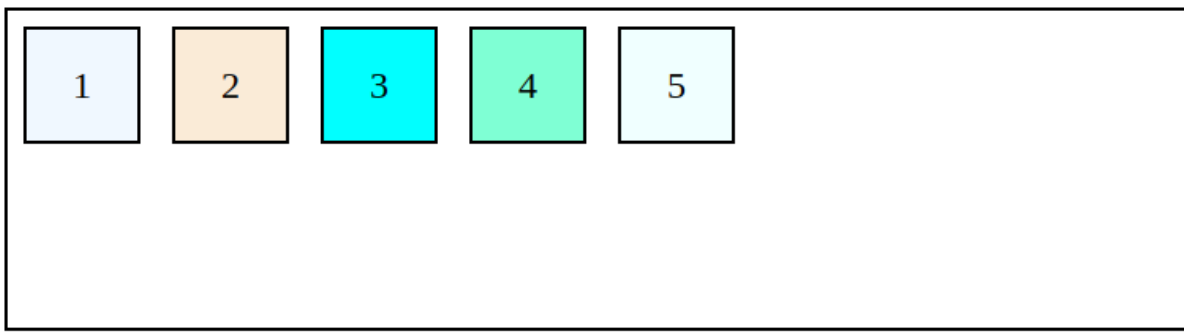

```
justify-content: space-around;
```

To załatwia sprawę ułożenia rzeczy na osi głównej, ale możemy również chcieć pobawić się wyrównywaniem na osi poprzecznej (w górę i w dół, gdy kierunkiem jest `row`, w lewo i w prawo, gdy kierunkiem jest `column`). Wyrównanie i rozciąganie w osi poprzecznej jest zadaniem właściwości `align-items`.

```
align-items

Wartości: flex-start | flex-end | center | baseline | stretch
Domyślnie: stretch
Dotyczy: flex containers
Dziedziczy: nie
```

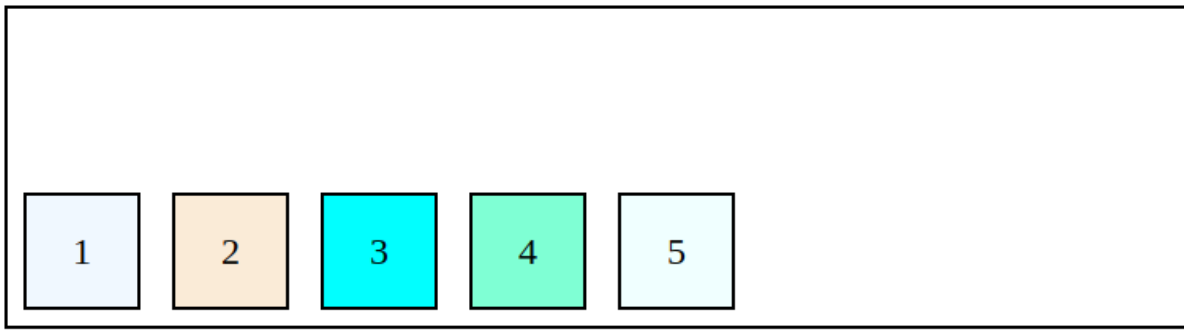
Różne wartości słowa kluczowego `align-items` w odniesieniu do wierszy pokazano poniżej. Aby zobaczyć efekt, musimy określić wysokość kontenera; w przeciwnym razie rozszerza się on tylko na tyle, by pomieścić zawartość bez dodatkowego miejsca. Podobnie jak `justify-content`, właściwość `align-items` odnosi się do kontenera `flex` (może to być trochę mylące, ponieważ "elementy" są w nazwie).

```
<!--flexbox_6_example.html-->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Flexbox six example</title>
</head>
<style>
  #container {
    display: flex;
    flex-direction: row;
    height: 200px;
    align-items: flex-start;
    border: 2px solid;
  }
  .box {
    border: 2px solid;
    font-size: 18pt;
    text-align: center;
    line-height: 50px;
    padding: 10px 10px 10px 10px;
    margin: 10px;
    width: 50px;
  }
  .box1 {
    background-color: aliceblue;
  }
  .box2 {
    background-color: antiquewhite;
  }
  .box3 {
    background-color: aqua;
  }
  .box4 {
    background-color: aquamarine;
  }
  .box5 {
    background-color: azure;
  }
</style>
<body>
<div id="container">
  <div class="box box1">1</div>
  <div class="box box2">2</div>
  <div class="box box3">3</div>
  <div class="box box4">4</div>
  <div class="box box5">5</div>
</div>
</body>
</html>
```

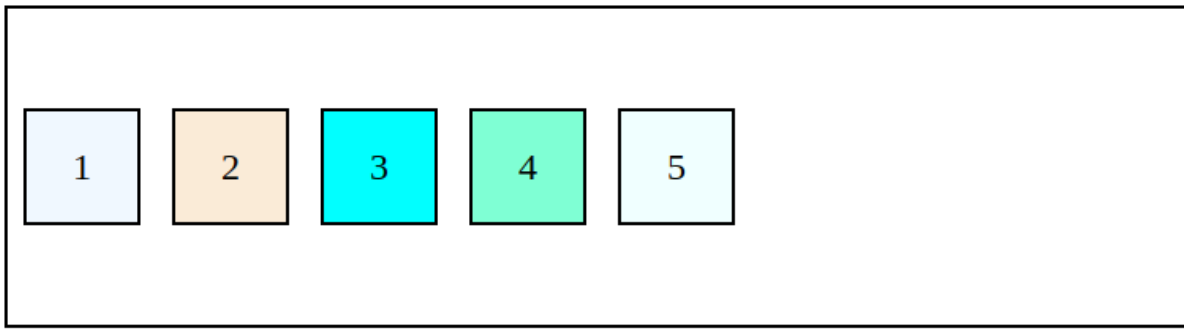


Wartości `flex-start`, `flex-end` i `center` powinny być znane, tyle że tym razem odnoszą się do początku, końca i środka osi poprzecznej. Wartość `baseline` wyrównuje linie bazowe pierwszych linii tekstu, niezależnie od ich rozmiaru. Może to być dobra opcja do wyłożenia elementów o różnych rozmiarach tekstu, takich jak nagłówki i akapity w wielu elementach. Wreszcie, `stretch`, które jest domyślne, powoduje, że elementy są rozciągane, dopóki nie wypełnią osi poprzecznej.

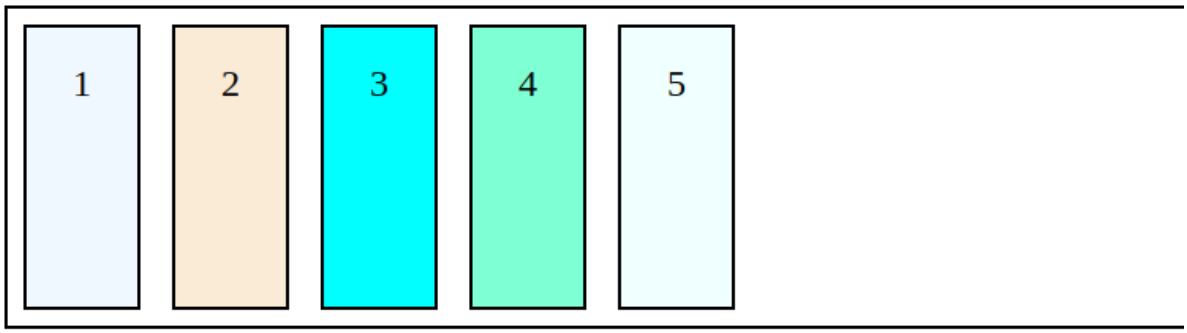
`align-items: flex-end;`



`align-items: center;`



`align-items: stretch;`



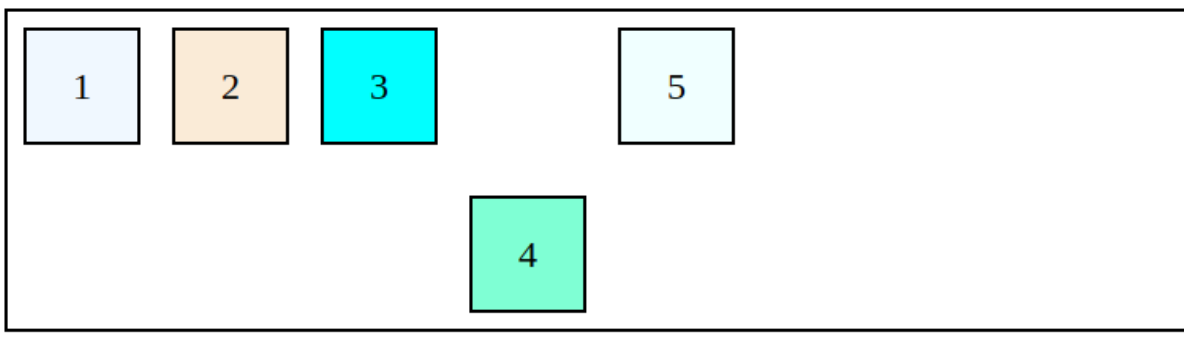
Gdy kierunek kontenera `flex` jest ustawiony na `column`, `align-items` wyrównuje elementy w lewo i w prawo. Zauważmy, że kiedy ustawiliśmy elementy w kolumnie i nie podaliśmy żadnych informacji o wyrównaniu, każdy element rozciągnął się na całą szerokość osi poprzecznej, ponieważ `stretch` jest wartością domyślną.

Jeśli chcemy, aby jeden lub więcej elementów zastąpiło ustawienie `cross-axis`, użyjmy właściwości `align-self` na poszczególnych elementach. Jest to pierwsza właściwość, jaką widzieliśmy, która dotyczy elementu, a nie samego kontenera. `align-self` używa tych samych wartości, co `align-items;` działa tylko na jednym elemencie na raz.

```
align-self

Wartości: flex-start | flex-end | center | baseline | stretch
Domyślnie: stretch
Dotyczy: flex items
Dziedziczy: no
```

W poniższym kodzie i na rysunku czwarte pole jest ustawione tak, aby wyrównać je na końcu osi poprzecznej. na końcu osi poprzecznej, podczas gdy pozostałe mają domyślne zachowanie rozciągania.

```
<!--flexbox_7_example.html-->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Flexbox seven example</title>
</head>
<style>
  #container {
    display: flex;
    flex-direction: row;
    height: 200px;
    align-items: baseline;
    border: 2px solid;
  }
  .box {
    border: 2px solid;
    font-size: 18pt;
    text-align: center;
    line-height: 50px;
    padding: 10px 10px 10px 10px;
    margin: 10px;
    width: 50px;
  }
  .box1 {
    background-color: aliceblue;
  }
  .box2 {
    background-color: antiquewhite;
  }
  .box3 {
    background-color: aqua;
  }
  .box4 {
    background-color: aquamarine;
    align-self: flex-end;
  }
  .box5 {
    background-color: azure;
  }
</style>
<body>
<div id="container">
  <div class="box box1">1</div>
  <div class="box box2">2</div>
  <div class="box box3">3</div>
  <div class="box box4">4</div>
  <div class="box box5">5</div>
</div>
</body>
</html>
```

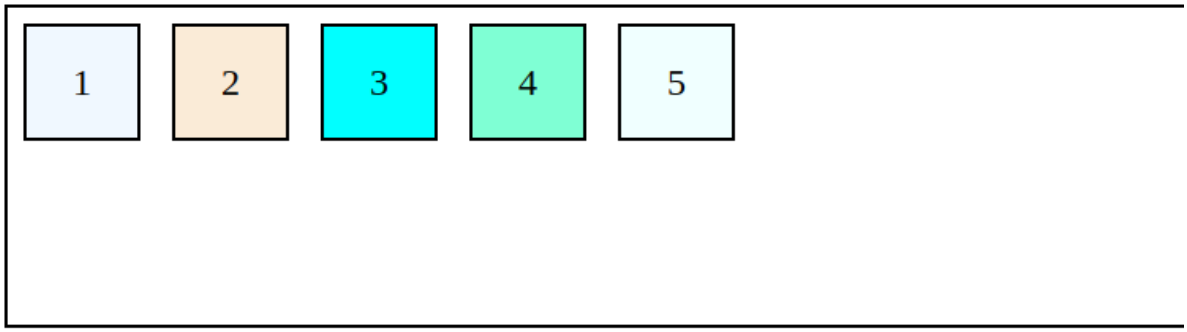


Ostatnia opcja wyrównania, `align-content`, wpływa na sposób rozłożenia wielu linii `flex` na osi poprzecznej. Ta właściwość ma zastosowanie tylko wtedy, gdy `flex-wrap` jest ustawiony na `wrap` lub `wrap-reverse` i istnieje wiele linii do wyrównania. Jeśli elementy znajdują się w jednej linii, nie robi to nic.

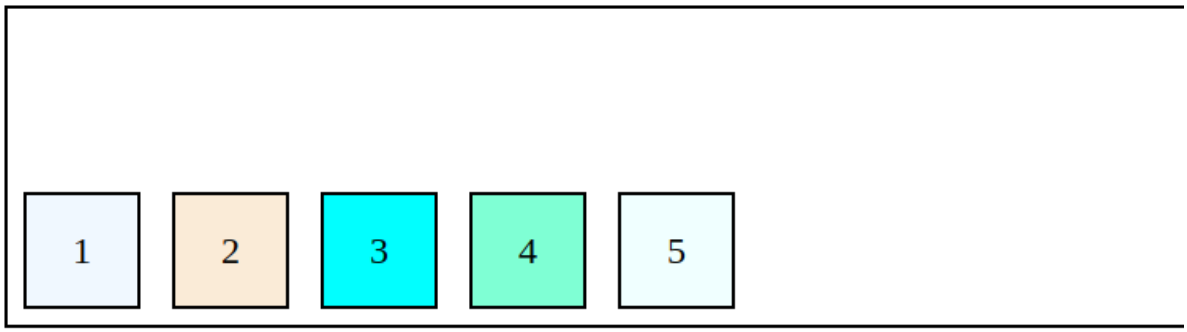
```
align-content

Wartości: flex-start | flex-end | center | space-around | space-between | stretch
Domyślnie: stretch
Dotyczy: multi-line flex containers
Dziedziczy: nie
```

Wszystkie wartości, które widzimy na liście właściwości, powinny wyglądać znajomo i działają tak, jak byśmy się spodziewali. Tym razem jednak dotyczą one tego, jak dodatkowa przestrzeń jest rozmieszczona wokół wielu linii na osi poprzecznej, jak pokazano poniżej. Ponownie właściwość `align-content` ma zastosowanie do elementu `flex` container. Dla kontenera wymagana jest również wysokość, ponieważ bez niej kontener byłby tylko wystarczająco wysoki, aby pomieścić zawartość i nie pozostałoby żadne miejsce.

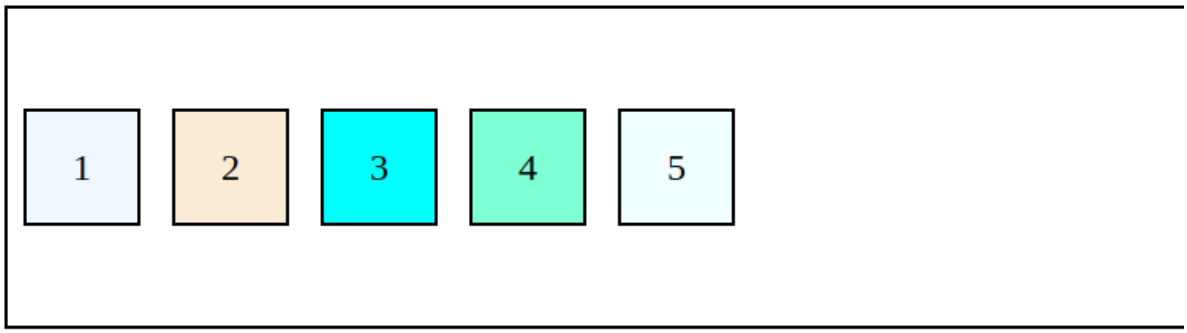
```
<!--flexbox_8_example.html-->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Flexbox eight example</title>
</head>
<style>
  #container {
    display: flex;
    flex-direction: row;
    flex-wrap: wrap;
    height: 200px;
    align-content: flex-start;
    border: 2px solid;
  }
  .box {
    border: 2px solid;
    font-size: 18pt;
    text-align: center;
    line-height: 50px;
    padding: 10px 10px 10px 10px;
    margin: 10px;
    width: 50px;
  }
  .box1 {
    background-color: aliceblue;
  }
  .box2 {
    background-color: antiquewhite;
  }
  .box3 {
    background-color: aqua;
  }
  .box4 {
    background-color: aquamarine;

  }
  .box5 {
    background-color: azure;
  }
</style>
<body>
<div id="container">
  <div class="box box1">1</div>
  <div class="box box2">2</div>
  <div class="box box3">3</div>
  <div class="box box4">4</div>
  <div class="box box5">5</div>
</div>
</body>
</html>
```



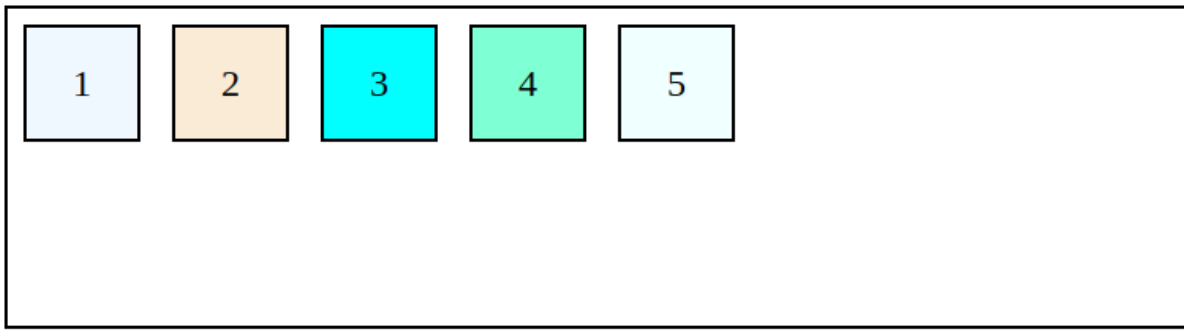

```
align-content: flex-end;
```



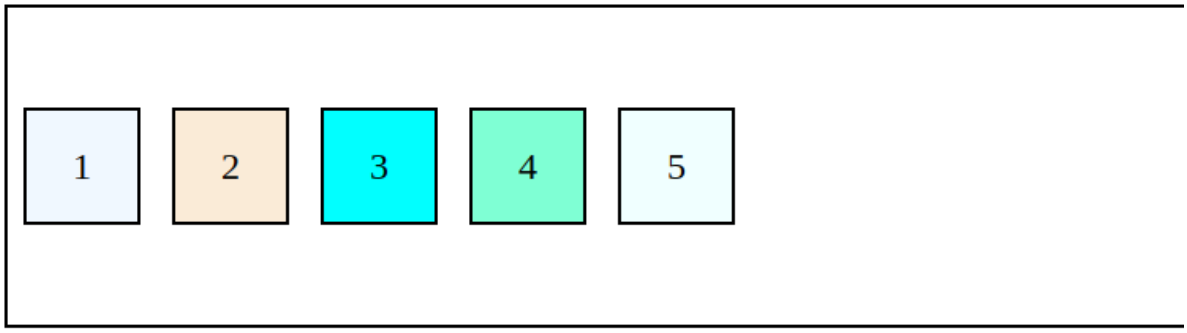

```
align-content: center;
```



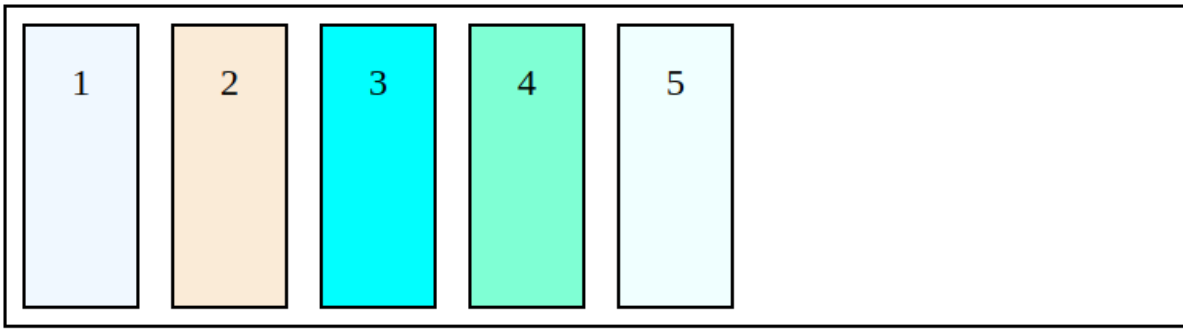

```
align-content: space-between;
```




```
align-content: space-around;
```




```
align-content: stretch;
```



Step 3

Skoro już mówimy o wyrównywaniu, jest jedna dobra sztuczka, którą należy pokazać, a która przyda się, gdy zaczniemy układać komponenty z `Flexbox`.

Paski menu są wszechobecne w sieci i często zdarza się, że jeden element paska, taki jak logo lub pole wyszukiwania, jest wizualnie wyróżniony od pozostałych. Możemy użyć marginesu, aby umieścić dodatkową przestrzeń kontenera po określonej stronie lub stronach elementu `flex`, wyróżniając w ten sposób jeden element. Powinno to być bardziej jasne dzięki przykładowi.

Menu na rysunku poniżej ma logo i cztery opcje menu. Chcielibyśmy, aby logo pozostało w lewym rogu, a opcje - po prawej stronie, niezależnie od szerokości rzutni.

```
<!--menu_example.html-->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Menu example</title>
</head>
<style>
  ul {
    display: flex;
    align-items: center;
    background-color: #00af8f;
    list-style: none; /* removes bullets */
    padding: .5em;
    margin: 0;
  }
  li {
    margin: 0 1em;
  }
  li.logo {
    margin-right: auto;
  }
</style>
<body>
  <ul>
    <li class="logo"><img src="img/logo.png" alt="Logo"></li>
    <li>About</li>
    <li>Blog</li>
    <li>Shop</li>
    <li>Contact</li>
  </ul>
</body>
</html>
```

Zmieniliśmy nieuporządkowaną listę (`ul`) w kontener `flex`, więc jej elementy listy (`li`) są teraz elementami `flex`. Domyślnie elementy pozostałyby razem na początku głównej osi (po lewej stronie) z dodatkową przestrzenią po prawej stronie. Ustawienie prawego marginesu elementu logo na auto powoduje przesunięcie dodatkowej przestrzeni na prawo od logo i przesunięcie pozostałych elementów na prawo.

Ta technika ma zastosowanie do wielu scenariuszy. Jeśli chcemy, aby tylko ostatni element pojawił się po prawej stronie, ustawmy jego lewy margines na `auto`. Chcemy mieć równą przestrzeń wokół środkowego elementu listy? Ustawmy zarówno lewy jak i prawy margines na `auto`. Chcemy przesunąć przycisk na dół kolumny? Ustawmy górny margines ostatniego elementu na `auto`. Dodatkowa przestrzeń w kontenerze przechodzi do marginesu i odsuwa sąsiednie elementy.

Jednym z największych cudów modelu `flexbox` jest to, że elementy zmieniają rozmiar, lub, używając formalnego określenia, `flex`, dopasowują się do dostępnej przestrzeni. To właśnie ta elastyczność sprawia, że `Flexbox` jest tak potężnym narzędziem do projektowania dla szerokiej gamy rozmiarów ekranu i okien przeglądarki, z którymi stykamy się jako projektanci stron internetowych. Piękne jest to, że przeglądarka sama ustala rozmiary w locie, a to oznacza mniej matematyki dla nas! W tej części poznamy właściwości `flex`.

Wcześniej dowiedzieliśmy się o właściwości `justify-content`, która rozprowadza dodatkową przestrzeń w kontenerze pomiędzy i wokół elementów wzdłuż osi głównej. Koncepcja `flex` dotyczy tego, jak przestrzeń jest dystrybuowana wewnątrz elementów, zwiększając lub zmniejszając elementy w zależności od potrzeb, aby je dopasować.

Elastyczność jest kontrolowana za pomocą właściwości `flex`, która określa, jak bardzo element może rosnąć i kurczyć się oraz określa jego rozmiar początkowy. Pełna historia jest taka, że `flex` jest skrótem właściwości `flex-grow`, `flex-shrink` i `flex-basis`, ale specyfikacja zdecydowanie zaleca, aby autorzy używali skrótu `flex` zamiast poszczególnych właściwości w celu uniknięcia sprzecznych wartości domyślnych i zapewnienia, że autorzy uwzględniają wszystkie trzy aspekty `flex` dla każdej instancji.

```
flex

Wartości: none | 'flex-grow flex-shrink flex-basis'
Domyślnie:0 1 auto
Dotyczy: flex items
Dziedziczy: nie
```

Wartość dla właściwości `flex` to zazwyczaj trzy właściwości `flex` wymienione w tej kolejności:

```
flex: flex-grow flex-shrink flex-basis;
```

Dla właściwości `flex-grow` i `flex-shrink` wartości `1` i `0` działają jak przełączniki `on/off`, gdzie `1` "włącza" lub pozwala elementowi rosnąć lub kurczyć się, a `0` zapobiega temu. Właściwość `flex-basis` ustawia wymiary początkowe, albo na określony rozmiar, albo na rozmiar oparty na zawartości.

W tym szybkim przykładzie, element listy zaczyna się od szerokości 200 pikseli, jest dozwolony do rozszerzenia, aby wypełnić dodatkową przestrzeń (`1`), ale nie jest dozwolone kurczenie się (`0`) węższe niż oryginalne 200 pikseli.

```
li {
  flex: 1 0 200px;
}
```

To powinno dać nam ogólny pogląd. Teraz przyjrzymy się bliżej wzrostowi, kurczeniu i rozmiarowi bazy, w tej kolejności.

Ale najpierw należy zauważyć, że `flex` i jego właściwości składowe **odnoszą się do elementów** `flex`, a nie do kontenera. Śledzenie, które właściwości są przypisane do kontenera, a które do elementów, jest jedną ze sztuczek w używaniu `Flexbox`.

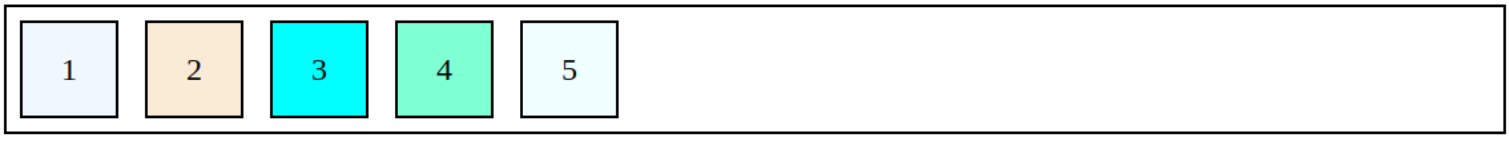
Pierwsza wartość we właściwości `flex` określa, czy (i w jakim stosunku) element może się rozciągać na większą skalę - innymi słowy, jest to wartość `flex-grow` (patrz uwaga). Domyślnie jest ona ustawiona na `0`, co oznacza, że element nie może rosnąć szerzej niż rozmiar jego zawartości lub określona szerokość. Ponieważ elementy nie rozszerzają się domyślnie, właściwości wyrównania mają szansę wejść w życie. Gdyby dodatkowe miejsce było zajęte wewnątrz elementów, wyrównanie nie działałoby.

```
flex-grow

Wartości: number
Domyslnie: 0
Dotyczy: flex items
Dziedziczy: nie
```

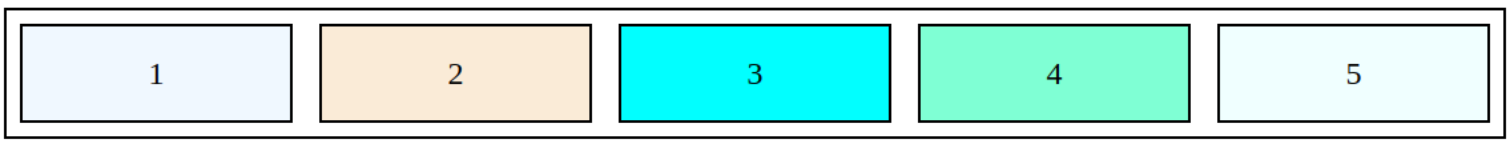
Jeśli ustawimy wartość `flex-grow` dla wszystkich elementów w kontenerze na `1`, przeglądarka bierze każdą dodatkową przestrzeń dostępną wzdłuż osi głównej i stosuje ją równo do każdego elementu, pozwalając im wszystkim rozciągnąć się w tym samym stopniu.

Weźmy przykład prostego pudełka i zobaczmy, jak zachowuje się ono przy zastosowaniu różnych ustawień `flex`. Rysunek poniżej pokazuje, co się dzieje, gdy `flex-grow` jest ustawiony na `1` dla wszystkich elementów pudełka (`flex-shrink` i `flex-basis` mają wartości domyślne). Porównajmy to z tym samym przykładem z `flex-grow` ustawionym na domyślne `0`.

```
<!--flex_example_1.html-->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Flex example 1</title>
</head>
<style>
  #container {
    display: flex;
    border: 2px solid;
  }
  .box {
    border: 2px solid;
    font-size: 18pt;
    text-align: center;
    line-height: 50px;
    padding: 10px 10px 10px 10px;
    margin: 10px;
    width: 50px;
    flex: 0 1 auto;
  }
  .box1 {
    background-color: aliceblue;
  }
  .box2 {
    background-color: antiquewhite;
  }
  .box3 {
    background-color: aqua;
  }
  .box4 {
    background-color: aquamarine;

  }
  .box5 {
    background-color: azure;
  }
</style>
<body>
<div id="container">
  <div class="box box1">1</div>
  <div class="box box2">2</div>
  <div class="box box3">3</div>
  <div class="box box4">4</div>
  <div class="box box5">5</div>
</div>
</body>
</html>
```

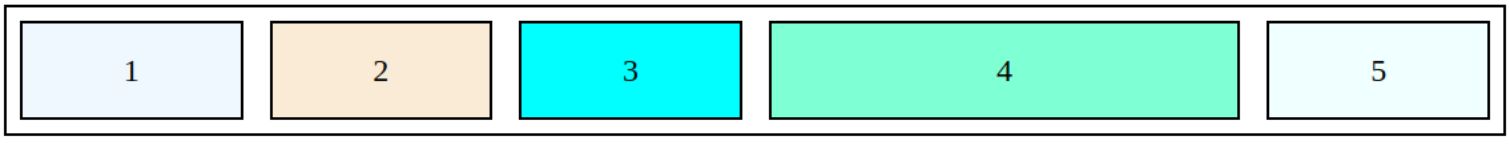


`flex: 1 1 auto;`



Jeśli określimy wyższą liczbę całkowitą `flex-grow` dla elementu, działa ona jak współczynnik, który stosuje więcej miejsca wewnątrz tego elementu. Na przykład, nadanie elementowi `box4` wartości `flex-grow: 3` oznacza, że otrzymuje on trzy razy więcej miejsca niż pozostałe elementy ustawione na `flex-grow: 1`

```
<!--flex_example_2.html-->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Flex example 1</title>
</head>
<style>
  #container {
    display: flex;
    border: 2px solid;
  }
  .box {
    border: 2px solid;
    font-size: 18pt;
    text-align: center;
    line-height: 50px;
    padding: 10px 10px 10px 10px;
    margin: 10px;
    width: 50px;
    flex: 1 1 auto;
  }
  .box1 {
    background-color: aliceblue;
  }
  .box2 {
    background-color: antiquewhite;
  }
  .box3 {
    background-color: aqua;
  }
  .box4 {
    background-color: aquamarine;
    flex: 3 1 auto;
  }
  .box5 {
    background-color: azure;
  }
</style>
<body>
<div id="container">
  <div class="box box1">1</div>
  <div class="box box2">2</div>
  <div class="box box3">3</div>
  <div class="box box4">4</div>
  <div class="box box5">5</div>
</div>
</body>
</html>
```



Zauważmy, że wynikowy element nie jest trzy razy szerszy od pozostałych; po prostu dodano do niego trzy razy więcej miejsca.

Jeśli na linii nie pozostało zbyt wiele miejsca, istnieje szansa, że każda porcja przestrzeni może być na tyle mała, że nie doda się do dużej różnicy. Być może trzeba będzie po prostu pobawić się wartościami `flex-grow` i dostosować szerokość przeglądarki, aż uzyskamy pożądany efekt.

Teraz, kiedy mamy już tę koncepcję, kurczenie się powinno być proste, ponieważ opiera się na tej samej zasadzie.

Step 4

Druga wartość właściwości `flex`, `flex-shrink`, włącza się, gdy pojemnik nie jest wystarczająco szeroki, aby pomieścić elementy, co powoduje deficyt miejsca. Zasadniczo zabiera ona trochę miejsca z wnętrza elementów, zmniejszając je tak, aby się zmieściły, zgodnie z określonym współczynnikiem.

```
flex-shrink

Wartości: number
Domyślnie: 1
Dotyczy: flex items
Dziedziczy: nie
```

Domyślnie wartość `flex-shrink` jest ustawiona na `1`, co oznacza, że jeśli nic nie zrobimy, elementy będą się kurczyć w tym samym tempie. Gdy `flex-shrink` ma wartość `0`, elementy nie są dopuszczane do kurczenia się i mogą zwisać ze swojego kontenera i być poza widokiem rzutni. Wreszcie, podobnie jak w przypadku `flex-grow`, większa liczba całkowita działa jako współczynnik. Element z `flex-shrink` równym `2` będzie się kurczył dwa razy szybciej niż gdyby miał ustawioną wartość `1`. Zazwyczaj nie ma potrzeby określania wartości współczynnika kurczenia. Wystarczy, że włączymy (`1`) lub wyłączymy (`0`) kurczenie.

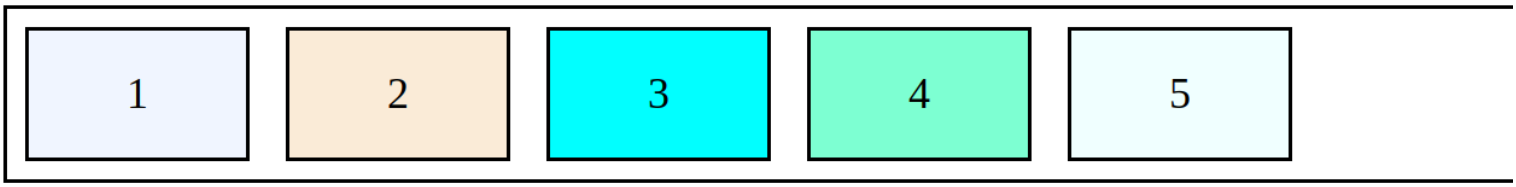
Elementy `Flex` przestają się zmniejszać, gdy osiągną swój minimalny rozmiar (określony przez `min-width` / `min-height`). Domyślnie (gdy `min-width` / `min-height` jest `auto`), to minimum jest oparte na jego `min-content`. Ale może być łatwo ustawione na zero, lub `12em`, lub dowolną inną długość, która wydaje się przydatna. Zwróćmy uwagę na ten efekt, gdy głęboko zagnieżdżone elementy wymuszają, by element `flex` był szerszy niż oczekiwany.

Trzecia wartość `flex` określa rozmiar początkowy elementu przed jego zawinięciem, zwiększeniem lub zmniejszeniem (`flex-basis`). Można ją zastosować zamiast właściwości `width` (lub `height` dla kolumn) dla elementów `flex`.

```
flex-basis

Wartości: length | percentage | content | auto
Domyślnie: auto
Dotyczy: flex items
Dziedziczy: nie
```

W tym przykładzie `flex-basis` pudełek jest ustawiony na 100 pikseli. Elementy mogą się zmniejszać, aby zmieścić się w dostępnej przestrzeni (`flex-shrink: 1`), ale nie mogą rosnąć szerzej (`flex-grow: 0`) niż 100 pikseli, pozostawiając dodatkowe miejsce w kontenerze.

```
<!--flex_example_3.html-->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Flex example 1</title>
</head>
<style>
  #container {
    display: flex;
    border: 2px solid;
  }
  .box {
    border: 2px solid;
    font-size: 18pt;
    text-align: center;
    line-height: 50px;
    padding: 10px 10px 10px 10px;
    margin: 10px;
    width: 50px;
    flex: 0 1 100px;
  }
  .box1 {
    background-color: aliceblue;
  }
  .box2 {
    background-color: antiquewhite;
  }
  .box3 {
    background-color: aqua;
  }
  .box4 {
    background-color: aquamarine;
  }
  .box5 {
    background-color: azure;
  }
</style>
<body>
<div id="container">
  <div class="box box1">1</div>
  <div class="box box2">2</div>
  <div class="box box3">3</div>
  <div class="box box4">4</div>
  <div class="box box5">5</div>
</div>
</body>
</html>
```



Domyślnie `flex-basis` jest ustawiony na `auto`, który używa określonych wartości właściwości `width` / `height` dla rozmiaru elementu. Jeśli główna właściwość rozmiaru elementu (`width` lub `height`) nie jest ustawiona lub jest `auto` (domyślnie), `flex-basis` używa szerokości zawartości. Można również jawnie ustawić `flex-basis` jako szerokość zawartości za pomocą słowa kluczowego `content`; jednak ta wartość jest słabo obsługiwana w tym momencie.

W tym przykładzie podstawa `flex` dla pudełek jest ustawiona na 100 pikseli, ponieważ wartość `auto` używa wartości ustawionej przez szerokość. Elementy mogą rosnąć, zajmując każdą dodatkową przestrzeń w kontenerze, ale nie mogą się kurczyć.

```
box {
  width: 100px;
  flex: 1 0 auto;
}
```

Kiedy przeglądarka przechodzi do wymiarowania elementu `flex`, sprawdza wartość `flex-basis`, porównuje ją z dostępnym miejscem wzdłuż osi, a następnie dodaje lub usuwa miejsce z elementów zgodnie z ich ustawieniami `grow` i `shrink`. Ważne jest, aby pamiętać, że jeśli pozwolisz elementowi rosnąć lub kurczyć się, może on być węższy lub szerszy niż długość przewidziana przez `flex-basis` lub `width`.

Zaletą używania właściwości `flex` jest to, że istnieją pewne poręczne wartości skrótów, które obejmują typowe scenariusze `Flexbox`. Co ciekawe, niektóre z tych skrótów zastępują wartości domyślne poszczególnych właściwości, co na początku może być mylące, ale w końcu skutkuje bardziej przewidywalnymi zachowaniami. Tworząc komponent `Flexbox`, sprawdźmy, czy jedno z tych łatwych ustawień załatwi sprawę:

- `flex: initial;` - To jest to samo co `flex: 0 1 auto`. Zapobiega to powiększaniu się elementu `flex`, nawet gdy jest dodatkowe miejsce, ale pozwala mu się kurczyć, aby zmieścić się w kontenerze. Rozmiar jest oparty na podanych właściwościach `width/height`, domyślnie jest to rozmiar zawartości. Mając wartość początkową, możemy użyć `justify-content` do wyrównania poziomego.
- `flex: auto;` - To jest to samo co `flex: 1 1 auto`. Pozwala elementom być w pełni elastycznym, rosnącym lub kurczącym się w zależności od potrzeb. Rozmiar jest oparty na określonych właściwościach `width/height`.
- `flex: none;` - Jest to odpowiednik `flex: 0 0 auto`. Tworzy całkowicie nieelastyczny element `flex`, jednocześnie dopasowując go do właściwości szerokości i wysokości. Możemy także użyć `justify-content` do wyrównania, gdy `flex` jest ustawiony na `none`.
- `flex: integer;` - To jest to samo co `flex: integer 1 0px`. Wynikiem jest element elastyczny z podstawą `flex` równą `0`, co oznacza, że ma absolutny `flex`, a wolne miejsce jest przydzielane zgodnie z numerem `flex` zastosowanym do elementów.

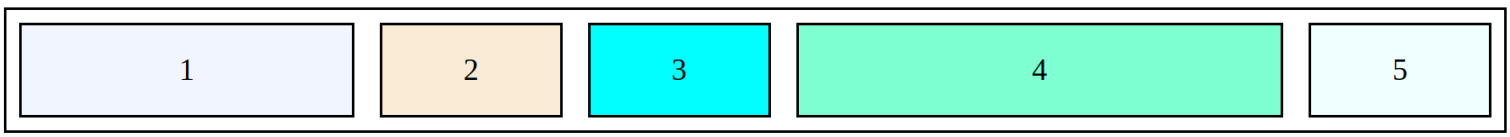
Widzieliśmy jak dodatkowe miejsce jest przypisywane do elementów na podstawie ich współczynników `flex`. Nazywa się to **względnym** `flex` i w ten sposób dodatkowe miejsce jest obsługiwane, gdy `flex-basis` jest ustawiony na dowolny rozmiar inny niż zero (`0`), taki jak określona wartość `width` / `height` lub `auto`.

Jeśli jednak zmniejszymy wartość `flex-basis` do `0`, stanie się coś interesującego. Przy podstawie `0`, elementy otrzymują rozmiar proporcjonalny zgodnie ze współczynnikami `flex`, co jest znane jako **absolutny** `flex`. Tak więc przy `flex-basis: 0`, element o wartości `flex-grow` równej `2` byłby dwa razy szerszy niż elementy ustawione na `1`. Ponownie, to rozwiązanie jest stosowane tylko wtedy, gdy `flex-basis` wynosi `0`.

W praktyce zaleca się, aby zawsze dołączać jednostkę po `0`, taką jak `0px` lub preferowane `0%`.

W tym przykładzie absolutnego `flex`, pierwsze pole ma wartość `flex-grow` równą `2`, a czwarte pole ma wartość `flex-grow` równą 3 poprzez wspomniany skrót `flex: integer.` Na rysunku widać, że wynikowy całkowity rozmiar pól jest proporcjonalny do wartości `flex-grow`, ponieważ `flex-basis` jest ustawiony na `0`.

```
<!--flex_example_4.html-->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Flex example 1</title>
</head>
<style>
  #container {
    display: flex;
    border: 2px solid;
  }
  .box {
    border: 2px solid;
    font-size: 18pt;
    text-align: center;
    line-height: 50px;
    padding: 10px 10px 10px 10px;
    margin: 10px;
    width: 50px;
    flex: 1 0 0%;
  }
  .box1 {
    background-color: aliceblue;
    flex: 2;
  }
  .box2 {
    background-color: antiquewhite;
  }
  .box3 {
    background-color: aqua;
  }
  .box4 {
    background-color: aquamarine;
    flex: 3;
  }
  .box5 {
    background-color: azure;
  }
</style>
<body>
<div id="container">
  <div class="box box1">1</div>
  <div class="box box2">2</div>
  <div class="box box3">3</div>
  <div class="box box4">4</div>
  <div class="box box5">5</div>
</div>
</body>
</html>
```



Step 5

Jedną z zabójczych cech `Flexboxa` jest możliwość wyświetlania elementów w kolejności różniącej się od ich kolejności w źródle Oznacza to, że możemy zmienić kolejność elementów w układzie używając tylko `CSS`. Jest to potężne narzędzie dla responsywnego projektowania, pozwalające na przesunięcie treści z dalszej części dokumentu na mniejsze ekrany.

Aby zmienić kolejność elementów, należy zastosować właściwość `order` do konkretnego elementu (elementów), który chcemy przenieść.

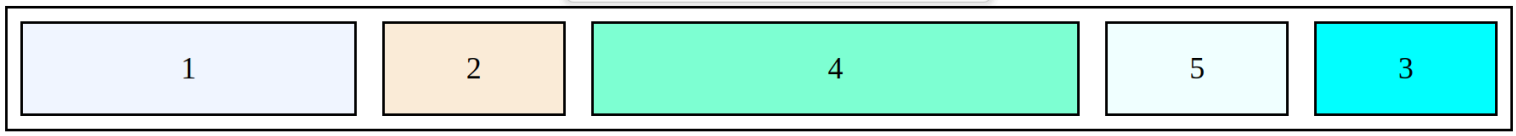
```
order

Wartości: integer
Domyślnie: 0
Dotyczy: flex items and absolutely positioned children of flex containers
Dziedziczy: nie
```

Wartość właściwości `order` jest dodatnią lub ujemną liczbą, która wpływa na umieszczenie elementu wzdłuż linii `flex`. Jest to podobne do właściwości `z-index` w tym, że nie ma znaczenia konkretna wartość liczbowa, tylko to, jak odnosi się do innych wartości.

Domyślnie wszystkie elementy mają wartość `order` równą zero (`0`). Gdy elementy mają tę samą wartość porządkową, są układane w kolejności, w jakiej pojawiają się w źródle HTML. Jeśli mają różne wartości `order`, są układane od najniższej wartości kolejności do najwyższej.

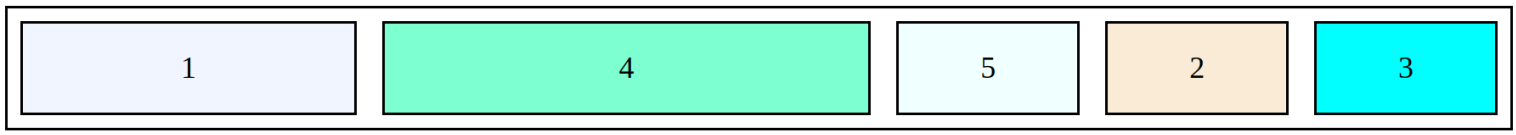
Wracając do naszego przykładu kolorowego, numerowanego pudełka, nadajmy pudełkowi 3 wartość order `1`. Z wyższą wartością porządkową, pojawia się ono po wszystkich elementach ustawionych na `0` (domyślnie):

```
<!--flex_example_4.html-->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Flex example 5</title>
</head>
<style>
    #container {
        display: flex;
        border: 2px solid;
    }
    .box {
        border: 2px solid;
        font-size: 18pt;
        text-align: center;
        line-height: 50px;
        padding: 10px 10px 10px 10px;
        margin: 10px;
        width: 50px;
        flex: 1 0 0%;
    }
    .box1 {
        background-color: aliceblue;
        flex: 2;
    }
    .box2 {
        background-color: antiquewhite;
    }
    .box3 {
        background-color: aqua;
        order: 1;
    }
    .box4 {
        background-color: aquamarine;
        flex: 3;
    }
    .box5 {
        background-color: azure;
    }
</style>
<body>
<div id="container">
    <div class="box box1">1</div>
    <div class="box box2">2</div>
    <div class="box box3">3</div>
    <div class="box box4">4</div>
    <div class="box box5">5</div>
</div>
</body>
</html>
```



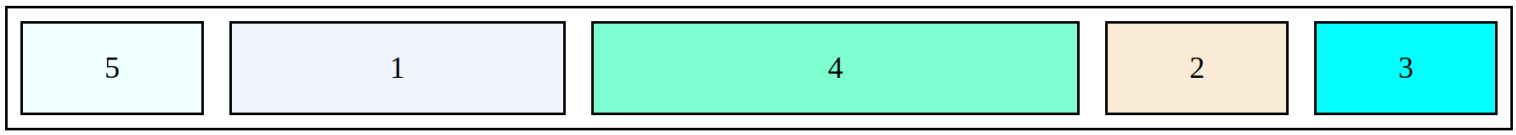
Gdy wiele elementów ma tę samą wartość `order`, ta grupa elementów dzielących wartość (zwana grupą porządkową) przylega do siebie i wyświetla się w kolejności źródłowej. Co się stanie, jeśli nadamy `box2` również wartość porządkową `1`? Teraz zarówno `box2` jak i `box3` mają wartość order `1` (co czyni je grupą porządkową) i są wyświetlane w kolejności źródłowej po wszystkich elementach o niższej wartości zamówienia `0`.

```
.box2 {
  background-color: antiquewhite;
  order: 1;
}
.box3 {
  background-color: aqua;
  order: 1;
}
```



Możemy również użyć ujemnych wartości dla zamówienia. Kontynuując nasz przykład, nadajmy polu 5 wartość order równą `-1`. Zauważmy że nie przesuwa się on tylko o jedno miejsce do tyłu; przesuwa się przed wszystkie elementy, które nadal mają ustawiony porządek na `0`, czyli wartość większą niż `-1`.

```
.box5 {
   background-color: azure;
   order: -1;
}
```



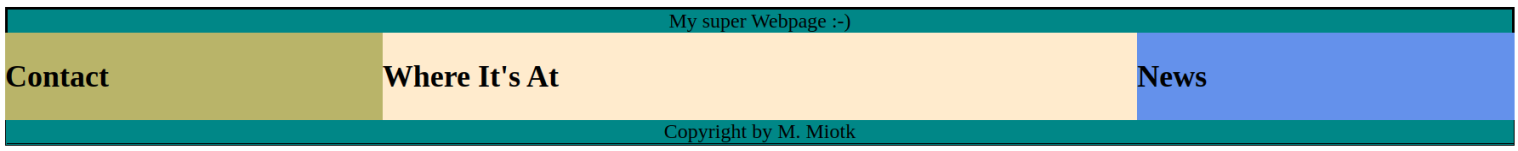
Użyliśmy prostych wartości `1` i `-1`, ale mogliśmy użyć 10008 lub -649, a wynik byłby taki sam; kolejność idzie od najmniejszej wartości do największej. Wartości liczbowe nie muszą być w kolejności sekwencyjnej.

Przyjrzyjmy się teraz, jak możemy wykorzystać order do czegoś bardziej użytecznego niż przesuwanie małych pól w linii. Oto prosty dokument z nagłówkiem, główną sekcją składającą się z artykułu i dwóch elementów bocznych oraz stopką:

```
<!--order_example.html-->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Order example</title>
</head>
<body>
  <header>…</header>
  <main>
    <article><h2>Where It's At</h2></article>
    <aside id="news"><h2>News></h2></aside>
    <aside id="contact"><h2>Contact</h2></aside>
  </main>
  <footer>…<footer>
</body>
</html>
```

W poniższym `CSS` z elementu main robimy element główny kontenera `flexbox`, dzięki czemu elementy `article` i `aside` ustawiają się w rzędzie, tworząc trzy kolumn. Ustawiony współczynnik `flex` dla każdego elementu, pozwala im rosnąć i kurczyć się, oraz ustawić ich szerokości za pomocą `flex-basis`. Na koniec używamy właściwości `order`, aby określić kolejność, w jakiej mają się pojawiać. Zauważmy, że sekcja `Contact` jest teraz pierwsza w rzędzie, chociaż w kolejności źródłowej pojawia się jako ostatnia. A jako dodatkowy bonus, wszystkie kolumny wypełniają wysokość głównego kontenera, pomimo ilości zawartej w nich zawartości.

```
<!--order_example.html-->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Order example</title>
  <style>
    main {
      display: flex;
    }
    header {
      border: 2px solid;
      text-align: center;
      border-bottom: none;
      background: darkcyan;
    }
    footer {
      border: 1px solid;
      text-align: center;
      border-top: none;
      background: darkcyan;
    }
    article {
      flex: 1 1 50%;
      order: 2;
      background: blanchedalmond;
    }
    #news {
      flex: 1 1 25%;
      order: 3;
      background: cornflowerblue;
    }
    #contact {
      flex: 1 1 25%;
      order: 1;
      background: darkkhaki;
    }
  </style>
</head>
<body>
  <header>My super Webpage :-)</header>
  <main>
    <article><h2>Where It's At</h2></article>
    <aside id="news"><h2>News</h2></aside>
    <aside id="contact"><h2>Contact</h2></aside>
  </main>
  <footer>Copyright by M. Miotk<footer>
</body>
</html>
```



Step 6

Projektanci stron internetowych i deweloperzy mamy moduł `CSS` do używania **siatki bazowej** (ang. `grid layout`) do osiągnięcia prawdziwego układu strony - i musieliśmy czekać tylko 25 lat, żeby go dostać! Moduł układu siatki `CSS` zapewnia system układania elementów w wierszach i kolumnach (pamiętajmy, że `Flexbox` układa elementy tylko na jednej osi) w sposób, który może pozostać

całkowicie elastyczny, aby dopasować się do różnych rozmiarów ekranu lub naśladować układ strony drukowanej. Siatki można wykorzystać do tworzenia układów stron internetowych, które są dziś znane, lub do bardziej wyrafinowanego wykorzystania typografii i białych przestrzeni. Siatki można też użyć do sformatowania tylko części strony, na przykład galerii zdjęć lub produktów.

Moduł układu siatki jest jedną z bardziej złożonych specyfikacji w `CSS`, której szczegóły mogłyby wypełnić książkę.

Proces używania modułu `CSS Grid Layout` jest zasadniczo prosty:

1. Użyj właściwości `display`, aby przekształcić element w kontener siatki (ang. `grid`). Dzieci elementu automatycznie stają się elementami siatki.
2. Ustaw kolumny i wiersze dla siatki. Możemy ustawić je jawnie i/lub podać wskazówki, jak wiersze i kolumny powinny być tworzone w locie.
3. Przypiszmy każdy element siatki do obszaru na siatce. Jeśli nie przypiszemy ich wyraźnie, wpływają one do komórek sekwencyjnie.

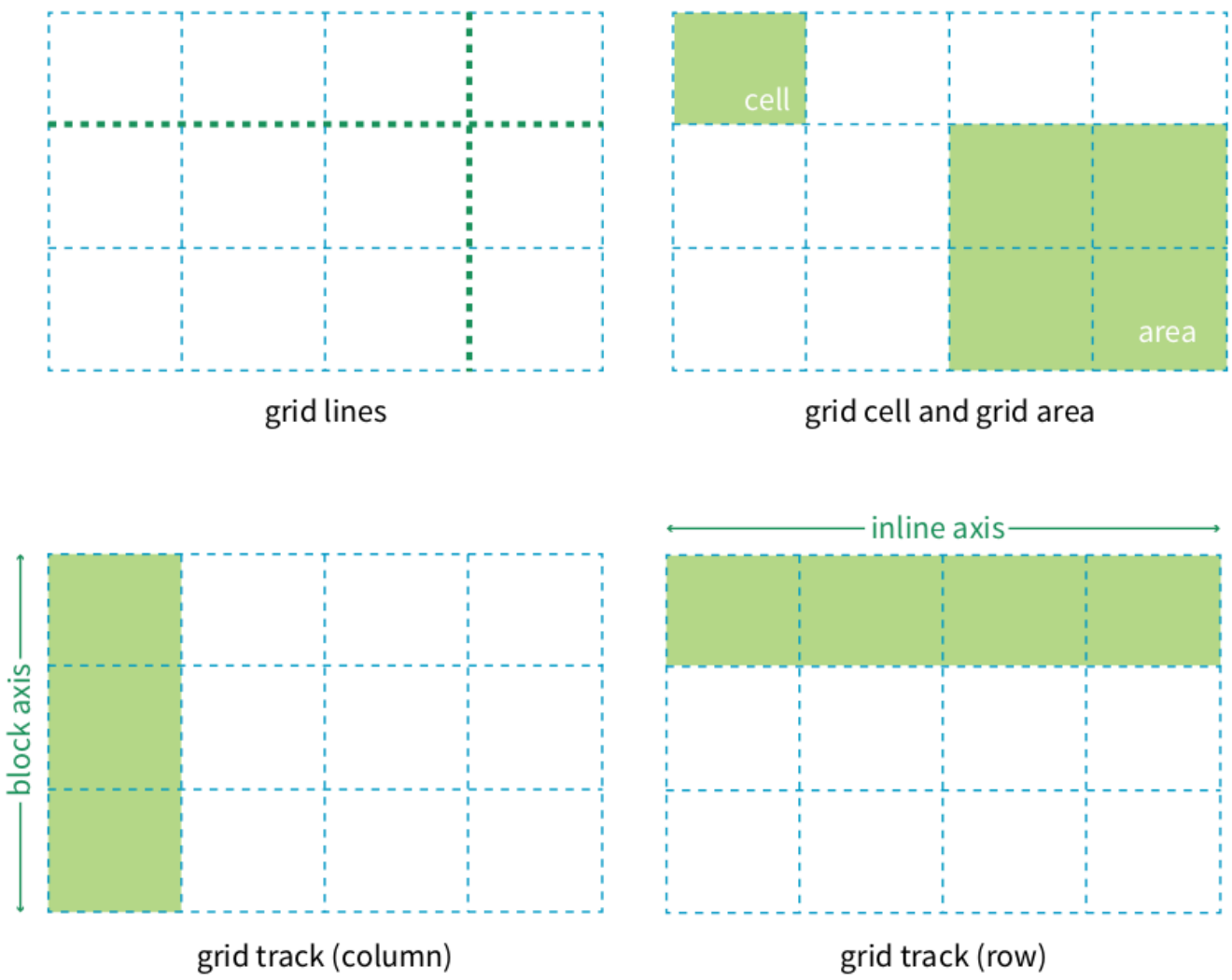
To, co czyni Układ siatki skomplikowanym, to fakt, że specyfikacja zapewnia tak wiele opcji do określenia każdej małej rzeczy. Wszystkie te opcje są świetne do dostosowywania pracy produkcyjnej, ale mogą być kłopotliwe, gdy uczysz się siatek po raz pierwszy. W tej sekcji przedstawimy solidny zestaw narzędzi do tworzenia siatki, który możemy rozbudowywać samodzielnie w miarę potrzeb.

Zanim zanurkujemy w konkretne właściwości, będziemy musiał zapoznać się z podstawowymi częściami i słownictwem systemu `Grid`.

Rozpoczynając pracę ze znacznikiem, element, który ma zastosowaną właściwość `display: grid`, staje się **kontenerem siatki** i definiuje kontekst dla siatki. Wszystkie jego bezpośrednie elementy potomne automatycznie stają się **elementami siatki**, które są pozycjonowane w siatce.

Kluczowe słowa w poprzednim akapicie to "bezpośrednie dziecko", ponieważ tylko te elementy stają się elementami siatki. Elementy zawarte w tych elementach nie są, więc nie można ich umieścić na siatce. Możemy jednak zagnieździć siatkę wewnątrz innej siatki, jeśli musimy zastosować siatkę na głębszym poziomie.

Sama siatka składa się z wielu elementów, na co wskazuje poniższy rysunek:



Linia siatki (ang. `Grid line`) -. Poziome i pionowe linie podziału siatki nazywane są liniami siatki.

Komórka siatki (ang. `Grid Cell`) - Najmniejszą jednostką siatki jest komórka siatki, która jest ograniczona czterema sąsiednimi liniami siatki, przez którą nie przebiegają żadne linie siatki.

Obszar siatki (ang. `Grid area`) - Obszar siatki to prostokątny obszar składający się z jednej lub więcej przylegających do siebie komórek siatki.

Ścieżka siatki (ang. `Grid track`) - Przestrzeń pomiędzy dwoma sąsiadującymi liniami siatki to ścieżka siatki, która jest ogólną nazwą kolumny lub wiersza siatki. Mówi się, że kolumny siatki idą wzdłuż osi blokowej, która jest pionowa (ponieważ elementy blokowe są ułożone w stos) dla języków pisanych poziomo. Rzędy siatki idą wzdłuż osi inline (poziomej).

Warto zwrócić uwagę, że struktura ustalona dla siatki jest niezależna od liczby elementów siatki w pojemniku. Możemy umieścić 4 elementy siatki w siatce z 12 komórkami, pozostawiając 8 komórek jako "białą przestrzeń". Na tym polega piękno siatek. Możemy również skonfigurować siatkę z mniejszą liczbą komórek niż elementy siatki, a przeglądarka dodaje komórki do siatki, aby je pomieścić. Jest to cudowny - w pełni elastyczny system.

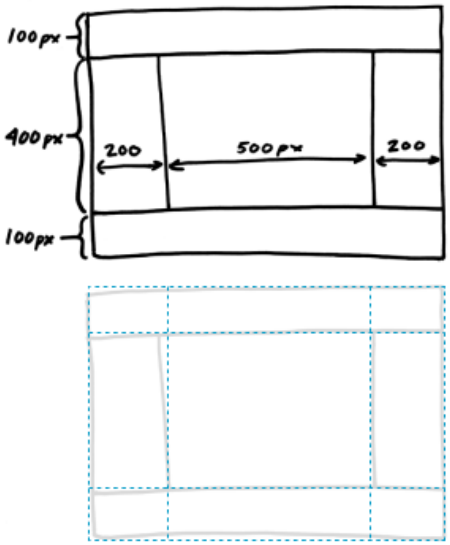
Aby zmienić element w kontener siatkowy, należy ustawić jego właściwość `display` na `grid` lub `inline-grid`. W tym prostym przykładzie, `div #layout` staje się kontenerem siatki, a każde z jego dzieci (`#one`, `#two`, `#three`, `#four`, i `#five`), jest zatem elementem siatki.

```
<!--grid_example_2.html-->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Grid example 2</title>
  <style>
      #layout {
        display: grid;
      }
        #layout > div {
            background-color: #ccc;
            border: 1px solid #000;
            padding: 10px;
        }
  </style>
</head>
<body>
  <div id="layout">
    <div id="one">One</div>
    <div id="two">Two</div>
    <div id="three">Three</div>
    <div id="four">Four</div>
    <div id="five">Five</div>
  </div>
</body>
</html>
```

One
Two
Three
Four
Five

To ustawia scenę (lub, używając bardziej dokładnego terminu, kontekst) dla siatki. Teraz możemy określić ile chcemy mieć wierszy i kolumn oraz jak szerokie powinny być.

Ponieważ nie chcemy mieć w głowie komórek i rozpiętości, zrobimy szybki szkic tego, jak chcemy, aby wyglądała nasza ostateczna siatka. Szkic to dobry pierwszy krok w pracy z siatkami. Ze szkicu wynika, że układ wymaga trzech ścieżek wierszy i trzech ścieżek kolumn, mimo że niektóre obszary zawartości zajmują więcej niż jedną komórkę. Jest to dość standardowy układ dla strony internetowej i chociaż trzymamy się jednowyrazowej treści, aby skupić się na strukturze, możemy sobie wyobrazić dłuższe treści tekstowe wypełniające każdy obszar.



Aby ustawić siatkę w `CSS`, określmy wysokość każdego wiersza i szerokość każdej kolumny za pomocą właściwości, `grid-template-rows` i `grid-template-columns`, które są stosowane do elementu kontenera.

```
grid-template-rows
grid-template-columns

Wartości: none | list of track sizes and optional line names
Domyślnie: none
Dotyczy: grid containers
Dziedziczy: nie
```

Wartość właściwości `grid-template-rows` jest listą wysokości dla każdego wiersza w siatce. Wartość właściwości `grid-template-columns` jest listą szerokości dla każdej ścieżki kolumny. Liczba rozmiarów ścieżek określa liczbę wierszy lub kolumn. Na przykład, jeśli podamy cztery długości dla `grid-template-columns`, otrzymamy siatkę, która jest początkowo podzielona na cztery kolumny.

Możemy również umieścić nazwy dla linii siatki między torami, do czego przejdziemy za chwilę, ale na razie zacznijmy najprościej jak to możliwe.

W poniższym przykładzie dodano właściwości szablonu, aby podzielić kontener `#layout` na trzy kolumny i trzy wiersze o rozmiarach, które wyznaczono w oryginalnym szkicu.

```
<!--grid_example_2.html-->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Grid example 2</title>
  <style>
    #layout {
      display: grid;
      grid-template-rows: 100px 400px 100px;
      grid-template-columns: 200px 500px 200px;
    }
    #layout > div {
      background-color: #ccc;
      border: 1px solid #000;
      padding: 10px;
    }
  </style>
</head>
<body>
<div id="layout">
  <div id="one">One</div>
  <div id="two">Two</div>
  <div id="three">Three</div>
  <div id="four">Four</div>
  <div id="five">Five</div>
</div>
</body>
</html>
```

One	Two	Three
Four	Five	

Zobaczmy, co się stanie, jeśli wykonamy szybkie sprawdzenie dotychczasowej siatki w przeglądarce. Rysunek powyżej pokazuje, że domyślnie elementy siatki wpływają w kolejności do dostępnych komórek siatki.

Ponieważ w `div` `#layout` znajduje się tylko pięć elementów podrzędnych, tylko pierwsze pięć komórek jest wypełnionych. To automatyczne zachowanie przepływu nie jest tym, o co może chodzić w tej siatce, ale jest przydatne w przypadkach, w których zawartość może być wrzucana do siatki sekwencyjnie, jak na przykład galeria zdjęć. Wkrótce umieścimy każdy z naszych elementów na tej siatce celowo, ale najpierw przyjrzyjmy się dokładniej wartościom właściwości szablonu.

Gdy przeglądarka tworzy siatkę, automatycznie przypisuje każdej linii siatki numer, do którego można się odwołać podczas pozycjonowania elementów. Linia siatki na początku ścieżki siatki to `1`, a od niej linie są numerowane kolejno.

Wiersze są numerowane również od końca ścieżek, zaczynając od `-1`, a numery liczone są od tego miejsca (`-2`, `-3`, itd.), co pokazują szare numery na rysunku powyżej. Możliwość zlokalizowania końca wiersza lub kolumny bez konieczności liczenia wierszy (lub nawet wiedzy o tym, ile jest wierszy lub kolumn) to przydatna funkcja.

Jeśli jednak nie lubimy śledzić liczb, możemy również przypisać liniom nazwy, które mogą być bardziej intuicyjne. W poniższym przykładzie przypisano nazwy, które odpowiadają temu, jak będziemy używać siatki na stronie końcowej. Nazwy linii są dodawane w nawiasach kwadratowych w pozycji, w jakiej pojawiają się w stosunku do ścieżek.

```
#layout {
  display: grid;
  grid-template-rows: [header-start] 100px [content-start] 400px [footer-start] 100px;
  grid-template-columns: [ads] 200px [main] 500px [links] 200px;
}
```

W oparciu o ten przykład, linia siatki na górze siatki może być teraz określona jako `"header-start"`, `"1"` lub `"-4"`. Możemy również nazwać linię, która przychodzi po pierwszej ścieżce wiersza `"header-end"`, nawet jeśli już została nazwana `"content-start"`. Aby nadać wierszowi więcej niż jedną nazwę, wystarczy zawrzeć wszystkie nazwy w nawiasach, oddzielając je spacjami:

```
grid-template-rows: [header-start] 100px [header-end content-start] 400px [footer-start] 100px;
```

Często zdarza się, że każdy wiersz siatki ma wiele nazw i numerów i można wybrać ten, który jest najłatwiejszy w użyciu. Za chwilę będziemy używać tych numerów i nazw do umieszczania elementów na siatce.

Step 7

Podaliśmy wszystkie rozmiary ścieżek w moim przykładzie w określonych długościach pikseli, aby ułatwić ich wizualizację, ale stałe rozmiary są jedną z wielu opcji. Nie oferują one również elastyczności wymaganej w naszym świecie wielu urządzeń. Moduł Układu Siatki zapewnia całą masę sposobów na określenie rozmiarów ścieżek, w tym stare standardy, takie jak długości (np. piksele lub ems) i wartości procentowe, ale także kilka nowszych i specyficznych dla siatki wartości. Przedstawimy szybko kilka przydatnych wartości specyficznych dla siatki: jednostkę `fr`, funkcję `minmax()`, `auto` oraz wartości oparte na zawartości `min-content/max-content`. Przyjrzymy się również funkcjom, które pozwalają na ustawienie powtarzającego się wzorca szerokości ścieżek: funkcja `repeat()` z opcjonalnymi wartościami `auto-fill` i `auto-fit`.

Specyficzna dla siatki jednostka ułamkowa (`fr`) pozwala programistom tworzyć szerokości ścieżek, które rozszerzają się i kurczą w zależności od dostępnej przestrzeni. Wracając do przykładu, jeśli zmienimy środkową kolumnę z `500px` na `1fr`, przeglądarka przypisze całą pozostałą przestrzeń (po tym jak `200-pikselowe` ścieżki kolumnowe zostaną dołączone) do tej ścieżki kolumnowej.

```
<!--grid_example_2.html-->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Grid example 3</title>
    <style>
        #layout {
            display: grid;
            grid-template-rows: 100px 400px 100px;
            grid-template-columns: 200px 1fr 200px;
        }
        #layout > div {
            background-color: #ccc;
            border: 1px solid #000;
            padding: 10px;
        }
    </style>
</head>
<body>
<div id="layout">
    <div id="one">One</div>
    <div id="two">Two</div>
    <div id="three">Three</div>
    <div id="four">Four</div>
    <div id="five">Five</div>
</div>
</body>
</html>
```

One	Two	Three
Four	Five	

Jednostka `fr` jest świetna do łączenia stałych i elastycznych szerokości ścieżek, ale moglibyśmy również użyć wszystkich jednostek `fr`, aby nadać wszystkim kolumnom proporcjonalne szerokości. W tym przykładzie wszystkie szerokości kolumn zginają się zgodnie z dostępną szerokością przeglądarki, ale środkowa kolumna będzie zawsze dwa razy szersza od kolumn bocznych.

```
grid-template-columns: 1fr 2fr 1fr;
```

One	Two	Three
Four	Five	

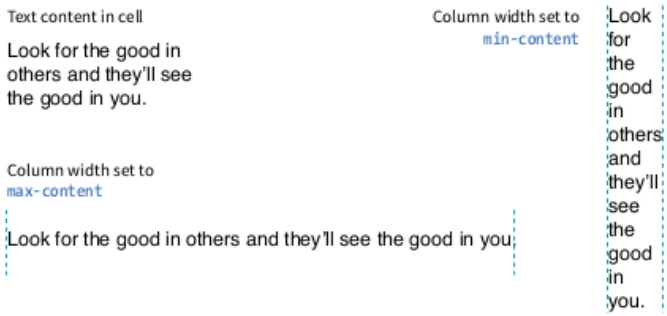
Można zwęzić zakres rozmiarów ścieżki, ustawiając jej minimalną i maksymalną szerokość za pomocą funkcji `minmax()` w miejsce określonego rozmiaru ścieżki.

```
grid-template-columns: 200px minmax(15em, 45em) 200px;
```

One	Two	Three
Four	Five	

Ta reguła ustawia środkową kolumnę na szerokość, która wynosi co najmniej `15em`, ale nigdy nie jest szersza niż `45em`. Ta metoda pozwala na elastyczność, ale pozwala autorowi na ustalenie limitów.

Wartości `min-content`, `max-content` i `auto` powodują, że ścieżka jest zwymiarowana w zależności od rozmiaru zawartości w niej zawartej.



Wartość `min-content` jest najmniejszą, jaką może uzyskać ścieżka bez przepełnienia (domyślnie, chyba że zostanie nadpisana przez jawne `min-width`). Innymi słowy, szerokość najdłuższego słowa lub najszerszego obrazu. Może nie być przydatny dla elementów zawierających normalne akapity, ale może być przydatny w niektórych przypadkach, gdy nie chcemy, aby ścieżka była większa niż musi być. Ten przykład ustanawia trzy kolumny, z prawą kolumną o rozmiarze wystarczająco szerokim, aby pomieścić najdłuższe słowo lub obraz:

```
grid-template-columns: 50px 1fr min-content;
```

One	Two	Three
Four	Five	

Właściwość `max-content` przydziela maksymalną ilość miejsca potrzebną dla zawartości, nawet jeśli oznacza to rozszerzenie ścieżki poza granice kontenera siatki. Kiedy jest używany jako szerokość kolumny, ścieżka kolumny będzie tak szeroka, jak najszersza zawartość w tej ścieżce bez zawijania linii. Oznacza to, że jeśli mamy akapit, ścieżka będzie wystarczająco szeroka, aby zawrzeć tekst ustawiony w jednym wierszu. To sprawia, że `max-content` jest bardziej odpowiedni dla krótkich fraz lub elementów nawigacyjnych, gdy nie chcemy, aby ich tekst był zawijany (`auto` może działać lepiej, ponieważ pozwala na zawijanie, jeśli nie ma wystarczająco dużo miejsca).

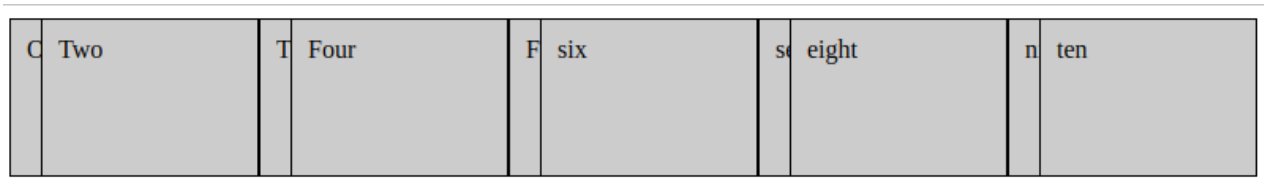
Użycie słowa kluczowego `auto` dla rozmiaru ścieżki jest w zasadzie przekazaniem kluczy przeglądarce. Ogólnie rzecz biorąc, powoduje ono, że ścieżka jest zwymiarowana na tyle, aby pomieścić jej zawartość, przy jednoczesnym uwzględnieniu innych ograniczeń.

W funkcji `minmax()`, słowo kluczowe `auto` zachowuje się bardzo podobnie do `min-content` lub `max-content`, w zależności od tego, czy umieścimy je w slocie `minmax` czy `max`. Jako samodzielne słowo kluczowe zachowuje się podobnie do `minmax(min-content, max-content)`, pozwalając ścieżce ścisnąć się tak wąsko, jak to tylko możliwe bez przepełnienia czymkolwiek, ale rosnąć tak, aby zmieścić swoją zawartość bez zawijania, jeśli jest wystarczająco dużo miejsca.

W przeciwieństwie do `max-content`, `auto` pozwala `align-content` i `justify-content` rozciągnąć ścieżkę poza rozmiar zawartości. Jako minimum, ma kilka sprytniejszych rozwiązań niż `min-content` - na przykład, używając określonej `min-width` lub `min-height` na elemencie (jeśli istnieje) zamiast jego `min-content size`, i ignorując zawartość wszelkich elementów siatki z paskami przewijania.

Powiedzmy, że mamy siatkę, która ma 10 kolumn z naprzemiennymi szerokościami kolumn, tak jak w tym przypadku:

```
<!--grid_example_4.html-->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Grid example 4</title>
  <style>
    #layout {
      display: grid;
      grid-template-rows: 100px 400px 100px;
      grid-template-columns: 20px 1fr 20px 1fr 20px 1fr 20px 1fr 20px 1fr 20px 1fr;
    }
    #layout > div {
      background-color: #ccc;
      border: 1px solid #000;
      padding: 10px;
    }
  </style>
</head>
<body>
<div id="layout">
  <div id="one">One</div>
  <div id="two">Two</div>
  <div id="three">Three</div>
  <div id="four">Four</div>
  <div id="five">Five</div>
  <div id="six">six</div>
  <div id="seven">seven</div>
  <div id="eigth">eight</div>
  <div id="nine">nine</div>
  <div id="ten">ten</div>
</div>
</body>
</html>
```

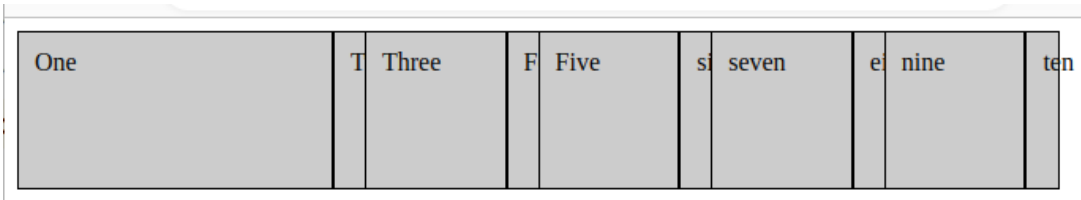


Jest to trochę kłopotliwe, gdy trzeba to wpisać, więc wspaniali ludzie z `W3C` zapewnili miły skrót w postaci funkcji `repeat()`. W poprzednim przykładzie, wzór `"20px 1fr"` powtarza się pięć razy, co można zapisać w następujący sposób:

```
grid-template-columns: repeat(5, 20px 1fr);
```

Pierwsza liczba wskazuje liczbę powtórzeń, a rozmiary ścieżek po przecinku stanowią wzór. Możemy użyć notacji `repeat()` w dłuższej sekwencji wielkości ścieżek - na przykład, jeśli te 10 kolumn znajduje się pomiędzy dwoma kolumnami o szerokości `200 pikseli` na początku i na końcu:

```
grid-template-columns: 200px repeat(5, 20px 1fr) 200px;
```



Można również podać nazwy linii siatki przed i/lub po każdym rozmiarze ścieżki, a nazwy te będą powtarzane we wzorcu:

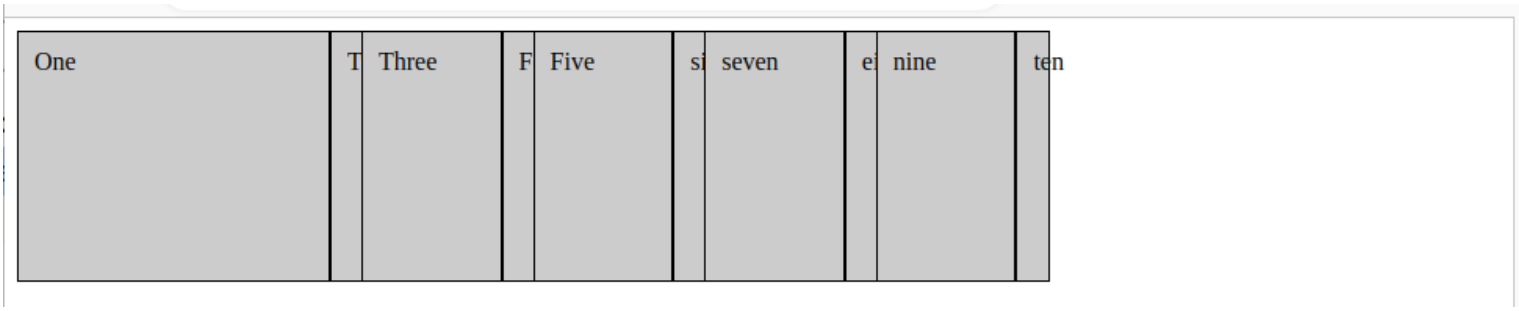
```
grid-template-rows: repeat(4, [date] 5em [event] 1fr);
```

W poprzednich przykładach `repeat()`, mówiliśmy przeglądarce ile razy ma powtórzyć podany wzór. Możemy również pozwolić przeglądarce na samodzielne obliczenie tego na podstawie dostępnego miejsca poprzez użycie wartości `auto-fill` i `auto-fit` zamiast liczby całkowitej w `repeat()`.

Na przykład, jeśli podamy:

```
grid-template-rows: repeat(auto-fill, 10em);
height: 35em;
```

a kontener siatki ma wysokość `35em`, to przeglądarka tworzy rząd co 10 emów, aż do wyczerpania miejsca, co skutkuje trzema rzędami. emNawet jeśli jest tylko tyle treści, aby wypełnić pierwszy rząd, wszystkie trzy rzędy są tworzone, a przestrzeń jest przechowywana w układzie



Wartość `auto-fit` działa podobnie, z tą różnicą, że wszystkie ścieżki, które nie są dopasowane, są usuwane z układu. Jeśli pozostało miejsce, jest ono rozdzielane zgodnie z podanymi wartościami wyrównania w pionie (`align-content`) i w poziomie (`justify-content`).

Step 8

Do tej pory badaliśmy, jak podzielić kontener siatki na ścieżki wierszy i kolumn za pomocą właściwości `grid-template-columns` i `grid-template-rows`, a także przyjrzeliśmy się wielu możliwym wartościom wymiarów ścieżek. Dowiedzieliśmy się, że można przypisać nazwy poszczególnym wierszom siatki, aby ułatwić sobie nawiązywanie do nich podczas umieszczania elementów na siatce.

Możemy również przypisać nazwy do obszarów siatki, co dla niektórych programistów jest jeszcze bardziej intuicyjną metodą niż wywoływanie konkretnych linii. Pamiętajmy, że obszar siatki składa się z jednej lub więcej komórek w prostokącie (nie ma L-kształtnych lub innych nieprostokątnych kolekcji komórek). Nazwanie obszarów siatki jest trochę zabawne do wdrożenia, ale zapewnia miłe skróty, gdy ich potrzebujemy.

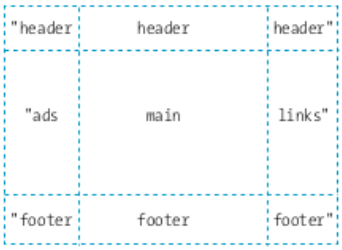
Aby przypisać nazwy do obszarów siatki, użyjmy właściwości `grid-template-areas`.

```
grid-template-areas

Wartości: none | series of area names
Domyślnie: none
Dotyczy: grid containers
Dziedziczy: nie
```

Wartością właściwości jest lista nazw podanych dla każdej komórki w siatce, wymienionych wiersz po wierszu, przy czym każdy wiersz jest ujęty w cudzysłów. Gdy sąsiednie komórki mają wspólną nazwę, tworzą obszar siatki o tej nazwie.

W poniższym przykładzie nadano nazwy obszarom w przykładowej siatce, nad którą pracowaliśmy do tej pory. Zauważmy, że dla każdej z dziewięciu komórek występuje nazwa komórki, tak jak w każdym wierszu. Listy komórek w rzędach nie muszą być ułożone w stos, ale wielu programistów uważa, że pomocne jest ułożenie komórek w linii według nazw z użyciem spacji znakowych, aby lepiej zwizualizować strukturę siatki.

```
#layout {
      display: grid;
      grid-template-rows: [header-start] 100px [content-start] 400px [footer-start] 100px;
      grid-template-columns: [ads] 200px [main] 1fr [links] 200px;
      grid-template-areas:
        "header header header"
        "ads main links"
        "footer footer footer";
}
```



Jeżeli w siatce są trzy kolumny, to dla każdego wiersza muszą być podane trzy nazwy. Jeśli chcesz pozostawić komórkę bez nazwy, wpiszmy w jej miejsce jedną lub więcej kropek (`.`) jako uchwyt spacji, aby każda komórka była nadal uwzględniona. Ponownie, szkic siatki z określonymi obszarami ułatwi zaplanowanie wartości `grid-template-areas` .

Należy pamiętać, że rozmiary ścieżek nadal pochodzą z właściwości `grid-template-columns` i `grid-template-rows` . Właściwość `grid-template-areas` po prostu przypisuje nazwy obszarom, co ułatwia późniejsze umieszczanie w nich elementów.

Użyj właściwości skrótu `grid` , aby ustawić wartości dla `grid-template-rows` , `grid-template-columns` i `grid-template-areas` z jedną regułą stylu. Pamiętajmy, że wszystkie właściwości, których nie użyjemy, zostaną zresetowane do wartości domyślnych, tak jak w przypadku wszystkich skrótów.

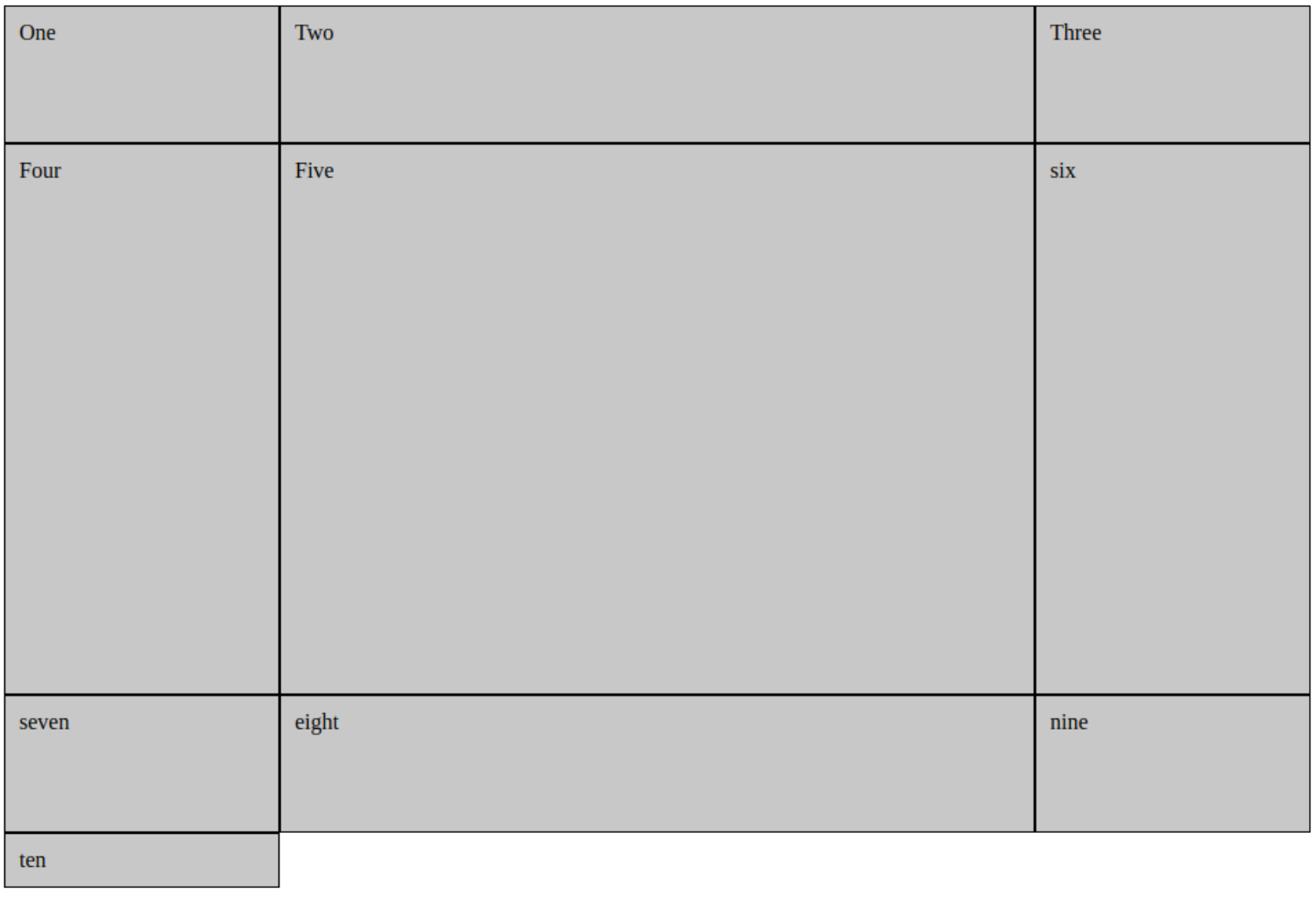
```
grid

Wartości: none | row info / column info
Domyślnie: none
Dotyczy: grid containers
Dziedziczy: nie
```

W siatce wartości wierszy i kolumn są oddzielone ukośnikiem, przy czym wartości wierszy pojawiają się jako pierwsze:

```
grid: rows / columns
```

Łatwiej jest to ogarnąć bez plątaniny nazw linii i obszarów, więc oto skrócona deklaracja dla naszej przykładowej siatki z informacjami o ścieżce wiersza i kolumny:

```
<!--grid_example_5.html-->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Grid example 5</title>
  <style>
    #layout {
      display: grid;
      grid: 100px 400px 100px / 200px 1fr 200px;
    }
    #layout > div {
      background-color: #ccc;
      border: 1px solid #000;
      padding: 10px;
    }
  </style>
</head>
<body>
<div id="layout">
  <div id="one">One</div>
  <div id="two">Two</div>
  <div id="three">Three</div>
  <div id="four">Four</div>
  <div id="five">Five</div>
  <div id="six">six</div>
  <div id="seven">seven</div>
  <div id="eigth">eight</div>
  <div id="nine">nine</div>
  <div id="ten">ten</div>
</div>
</body>
</html>
```



Aby dołączyć niestandardowe nazwy linii, należy dodać nazwy w nawiasach wokół odpowiednich ścieżek, jak widzieliśmy we wcześniejszym przykładzie.

Włączanie nazw obszarów wygląda na początku trochę zawile, ale jeśli pamiętamy, że wymieniamy nazwy komórek rząd po rzędzie, to ma sens, że pojawiają się one wraz z innymi informacjami o rzędzie, przed ukośnikiem. Pełna kolejność wygląda następująco:

```
[start line name] "area names" <track size> [end line name]
```

Nazwy linii i obszarów są opcjonalne. Powtórz to dla każdego wiersza w siatce, po prostu wymieniając je jeden po drugim, bez żadnych specjalnych znaków oddzielających wiersze. Pomocne może okazać się ułożenie ich w stos, tak jak to zrobiłem w poniższym przykładzie, aby każdy rząd był odrębny. Gdy wiersze są gotowe, dodajemy ukośnik, a po nim wypisujemy informacje o ścieżce kolumny. Oto kompletny przykład naszej siatki napisanej za pomocą skrótu `grid`:

```
#layout {
      display: grid;
      grid:
        [header-start] "header header header" 100px
        [content-start] "ads main links" 400px
        [footer-start] "footer footer footer" 100px /
        [ads] 200px
        [main] 1fr
        [links] 200px;
}
```

Istnieje również właściwość `grid-template`, która działa dokładnie jak `grid`, ale może być używana tylko z jawnie zdefiniowanymi siatkami. Specyfikacja `Grid Layout` zdecydowanie zaleca, aby używać skrótu `grid` zamiast `grid-template`, chyba że chcemy kaskadowego zachowania `grid-template`.

Step 9

You can add text and also images, math formulas, code examples and much more in this theory step.