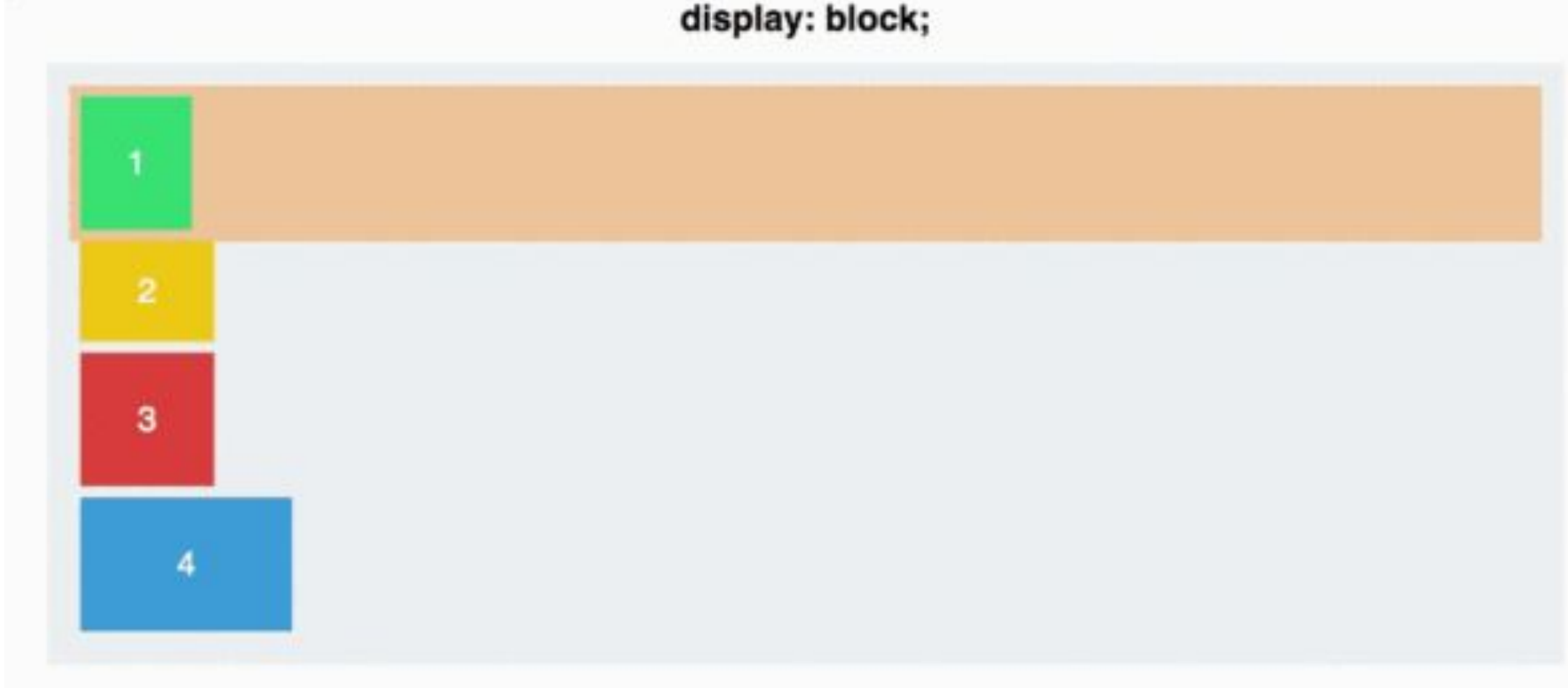
# Flexbox

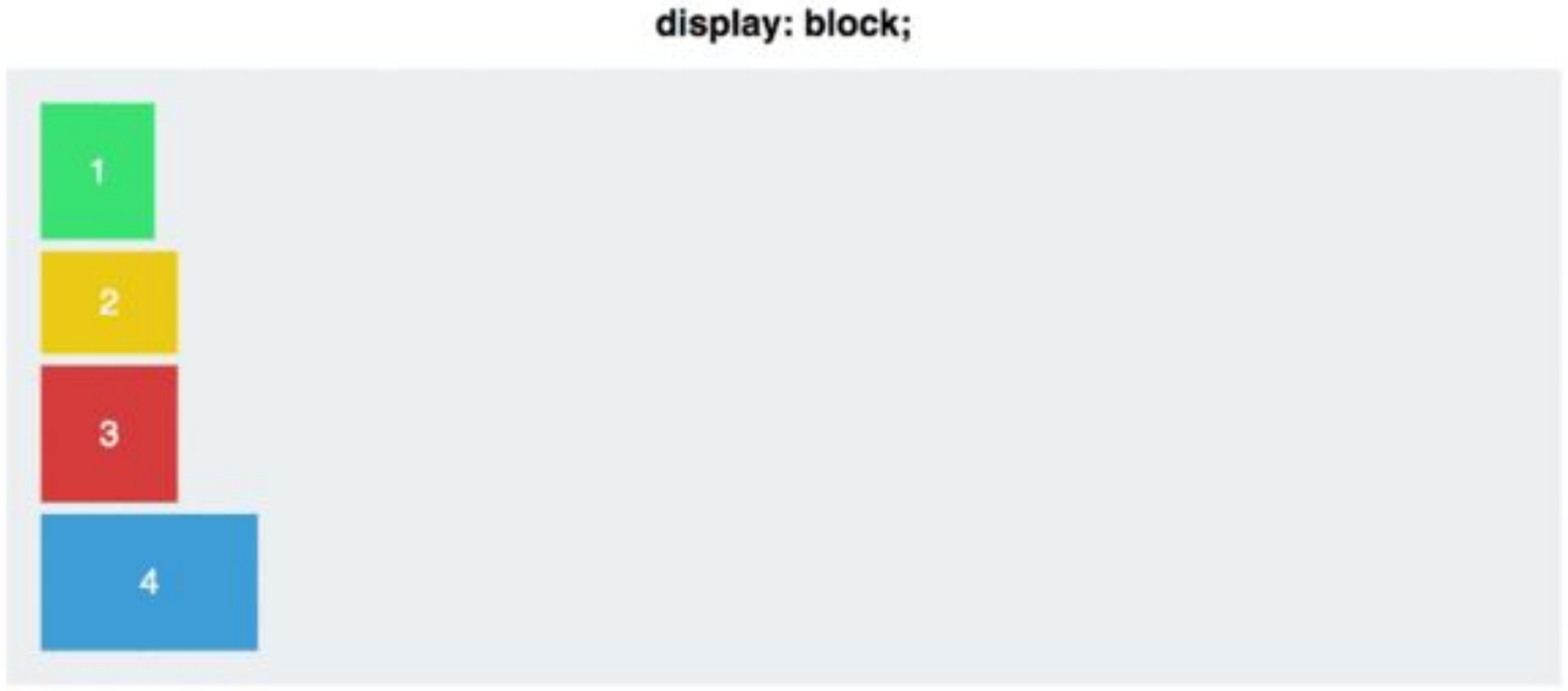
Є сторінка:

На ній розміщено 4 div різних розмірів, які знаходяться всередині сірого div з id=”container”. У кожного div є властивість display: block. Тому кожен блок займає всю ширину рядка.



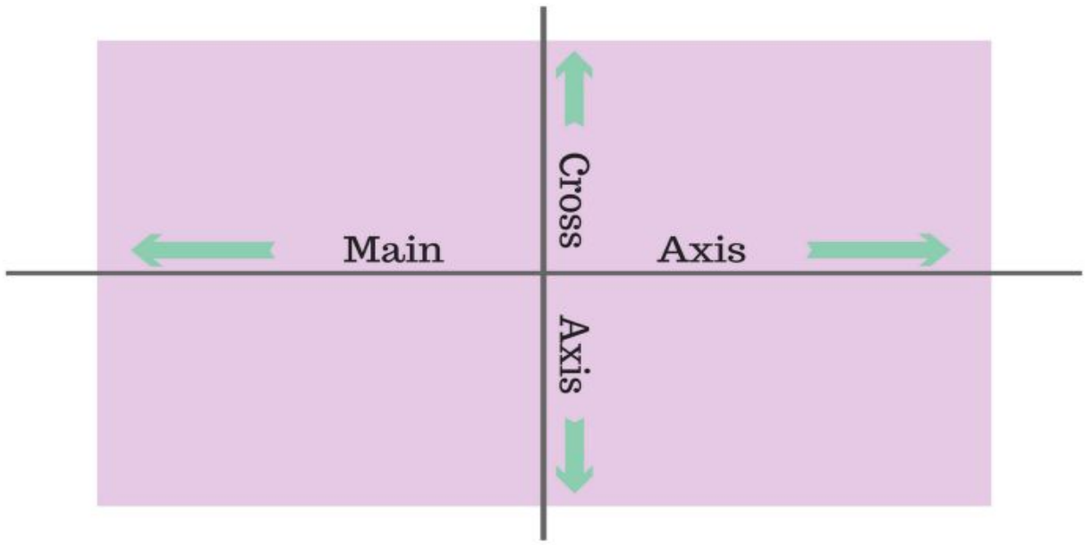
Щоб почати працювати з CSS Flexbox, потрібно зробити контейнер flex-контейнером. Робиться це так:

```
#container {

   display: flex;

}
```



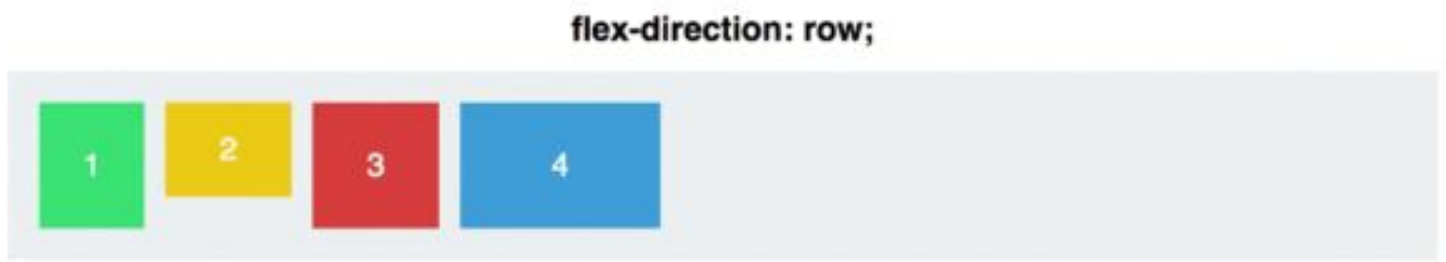
# Flex-direction

У flex-контейнера є дві осі: головна і перпендикулярна їй.



За замовчуванням всі елементи розташовуються вздовж головної осі - зліва направо. Саме тому блоки в попередньому прикладі вишикувалися в лінію, коли ми застосували display: flex. А ось flex-direction дозволяє обертати головну вісь.

```
#container {
    display: flex;
    flex-direction: column;
}
```



Зверніть увагу, що flex-direction: column НЕ вирівнює блоки по осі, перпендикулярній головній. Головна вісь сама змінює своє розташування, і тепер спрямована згори донизу.

Є ще парочка властивостей для flex-direction: row-reverse і column-reverse які впроваджують “дзеркальність”

# justify-content

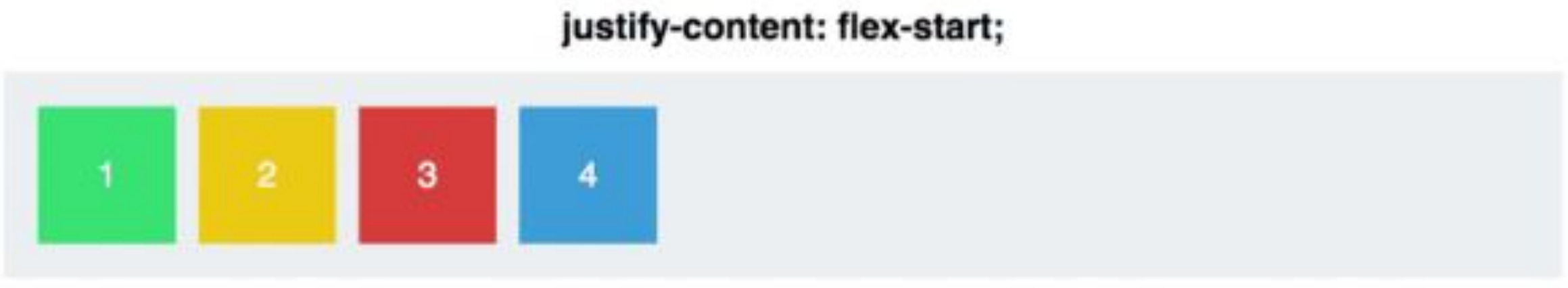
Відповідає за вирівнювання елементів по головній осі:

```
#container {
    display: flex;
    flex-direction: row;
    justify-content: flex-start;
}
```

`Justify-content може приймати 5 значень:`

1. `flex-start`
2. `flex-end`
3. `center`
4. `space-between`
5. `space-around`

Space-between задає однакову відстань між блоками, але не між контейнером і блоками. Space-around також задає однакову відстань між блоками, але тепер відстань між контейнером і блоками дорівнює половині відстані між блоками.

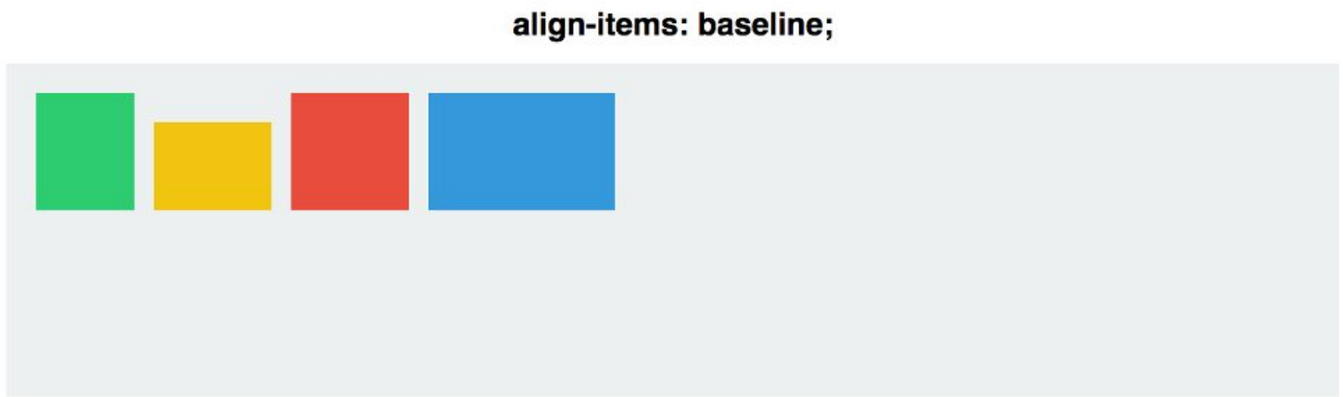


# align-items

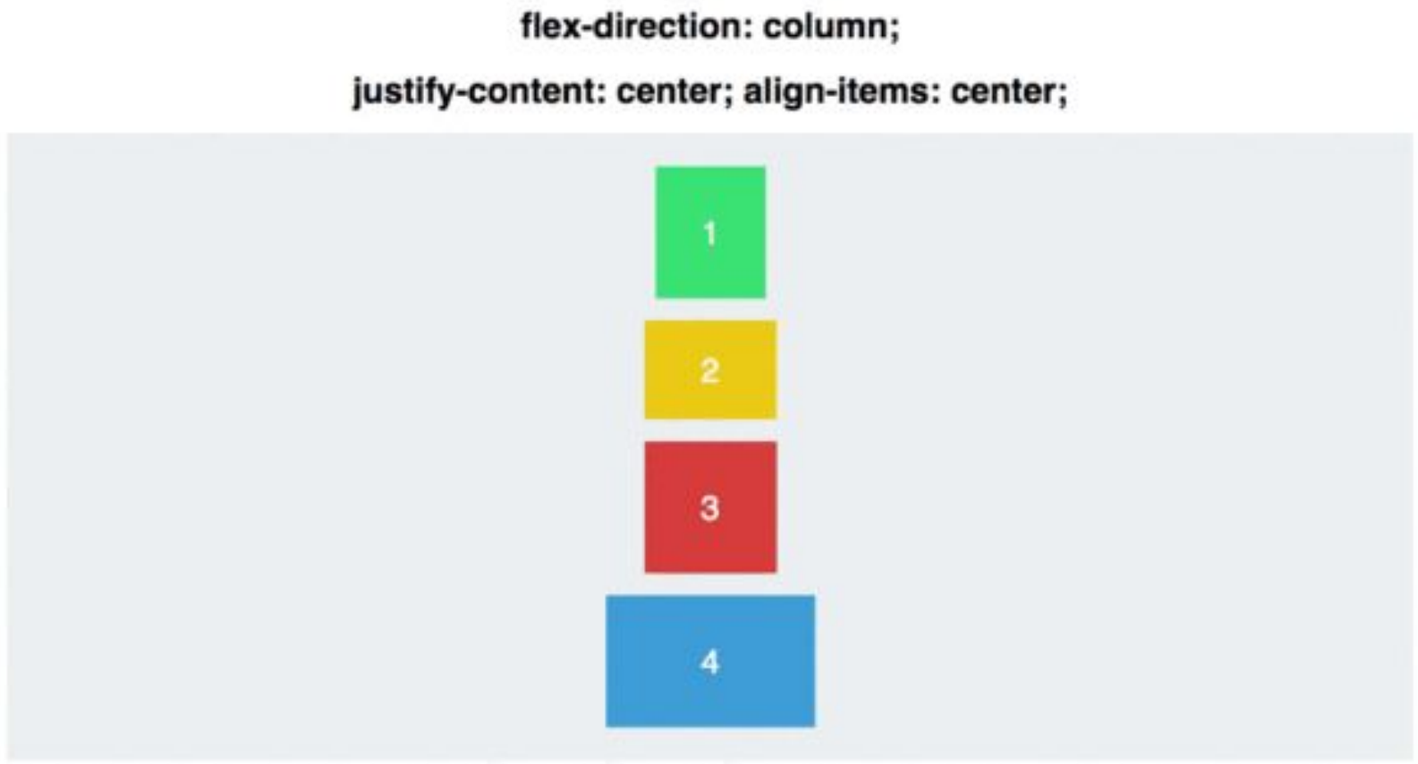
Якщо justify-content працює з головною віссю, то align-items працює з віссю, перпендикулярної головної осі. Повернемося до flex-direction: row і пройдемося по командам align-items: flex-start flex-end center stretch baseline.


Варто зауважити, що для align-items: stretch висота блоків повинна бути auto (або не вказана явно).

Для align-items: baseline теги параграфа прибирати не потрібно, інакше вийде так:




Щоб краще розібратися в тому, як працюють осі, об'єднаємо justify-content з align-items і подивимося, як працює вирівнювання по центру для двох властивостей flex-direction:



# flex-basis

Відповідає за початковий розмір елементів до того, як вони будуть замінені іншими властивостями CSS Flexbox:

# column-gap

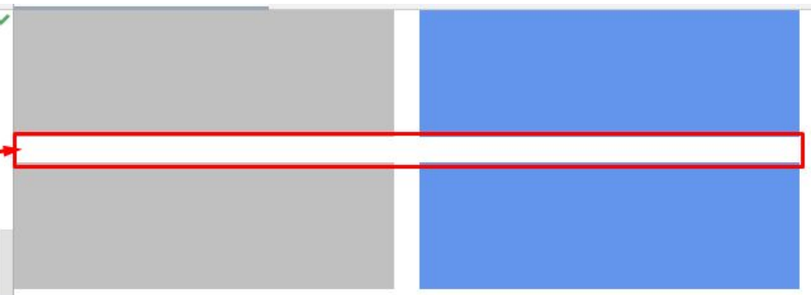
Відповідає за відступи між елементами у flex-контейнері

```
<style>
    body {margin: 0;}

    .wrap {
        display: flex;
        column-gap: 20px;
        row-gap: 20px;
        flex-wrap: wrap;
    }

    .box {
        height: 100px;
        flex-basis: 300px;
    }

    .box_silver {background: silver;}
    .box_cornflowerblue {background: cornflowerblue;}
</style>
</head>
<body>
<div class="wrap">

    <div class="box box_silver"></div>
    <div class="box box_cornflowerblue"></div>
    <div class="box box_silver"></div>

</div>

</body>
</html>
```

# row-gap

Відоповідає за відступи між рядками елементів які знаходяться всередині flex-контейнера

```
<style>
    body {margin: 0;}

    .wrap {
        display: flex;
        column-gap: 20px;
        row-gap: 20px;
        flex-wrap: wrap;
    }

    .box {
        height: 100px;
        flex-basis: 300px;
    }

    .box_silver {background: silver;}
    .box_cornflowerblue {background: cornflowerblue;}
</style>
</head>
<body>
<div class="wrap">

    <div class="box box_silver"></div>
    <div class="box box_cornflowerblue"></div>
    <div class="box box_silver"></div>
    <div class="box box_cornflowerblue"></div>
</div>

</body>
</html>
```



# align-self

Дозволяє маніпулювати окремими дочірніми елементами. Зауважте, що align-self потрібно присвоювати НЕ до батьківського елементу.

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

.square#one { align-self: center; }

// Only this square will be centered.
```

Для двух блоков применим align-self, а для остальных — align-items: center и flex-direction: row.

flex-basis впливає на розмір елементів вздовж головної осі. Подивимось, що станеться, якщо ми змінимо напрямок головної осі:


Зауважте, що нам довелося змінити і висоту елементів: flex-basis може визначати як висоту елементів, так і їх ширину в залежності від напрямку осі.

# flex-grow

Визначає коефіцієнти зростання блоку. Для початку задамо блокам однакову ширину 120px.

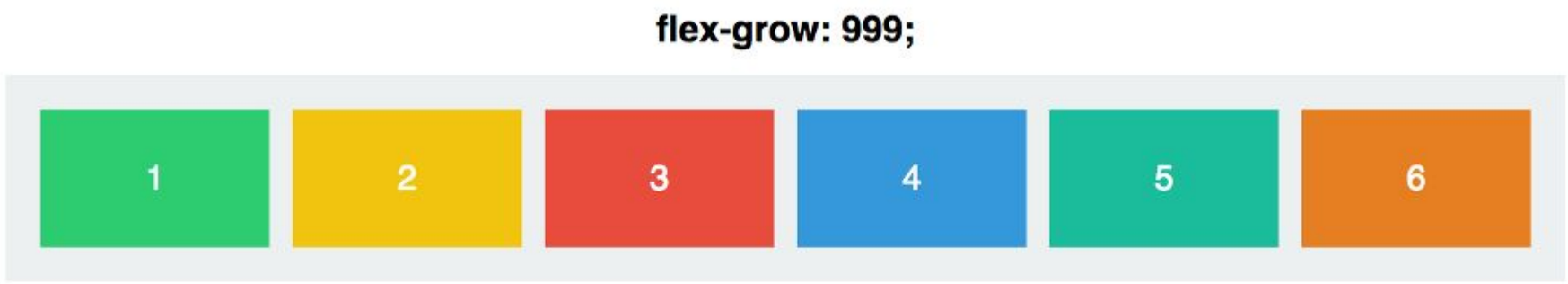
**width: 120px;**


За замовчуванням значення flex-grow дорівнює 0. Це означає, що блокам заборонено збільшуватися в розмірах. Задамо flex-grow рівним 1 для кожного блоку:

**flex-grow: 1; width: 120px;**


Тепер блоки зайняли місце, що залишилося в контейнері

Але що значить flex-grow: 1? Спробуємо зробити flex-grow рівним 999:




І ... нічого не сталося. Так вийшло через те, що **flex-grow приймає не абсолютні значення, а відносні**. Це означає, **що не важливо, яке значення у flex-grow, важливо, яке воно по відношенню до інших блоків:**


Спочатку flex-grow кожного блоку дорівнює 1, в сумі вийде 6. Отже, наш контейнер розділений на 6 частин. Кожен блок буде займати 1/6 частина доступного простору в контейнері. Коли flex-grow третього блоку стає рівним 2, контейнер ділиться на 7 частин: 1 + 1 + 2 + 1 + 1 + 1. Тепер третій блок займає 2/7 простору, інші - по 1/7. І так далі.

flex-grow працює тільки для головної осі, поки ми не змінимо її напрямок.

# flex-shrink

Пряма протилежність flex-grow. Визначає, наскільки блоку можна зменшитися в розмірі. flex-shrink використовується, коли елементи не вміщаються в контейнер. Ви визначаєте, які елементи повинні зменшитися в розмірах, а які - ні. За замовчуванням значення flex-shrink для кожного блоку дорівнює 1. Це означає, що блоки будуть стискуватися, коли контейнер буде зменшуватися.

Задамо flex-grow і flex-shrink рівними 1:

Тепер поміняємо значення flex-shrink для третього блоку на 0. Йому заборонили стискатися, тому його ширина залишиться рівною 120px:

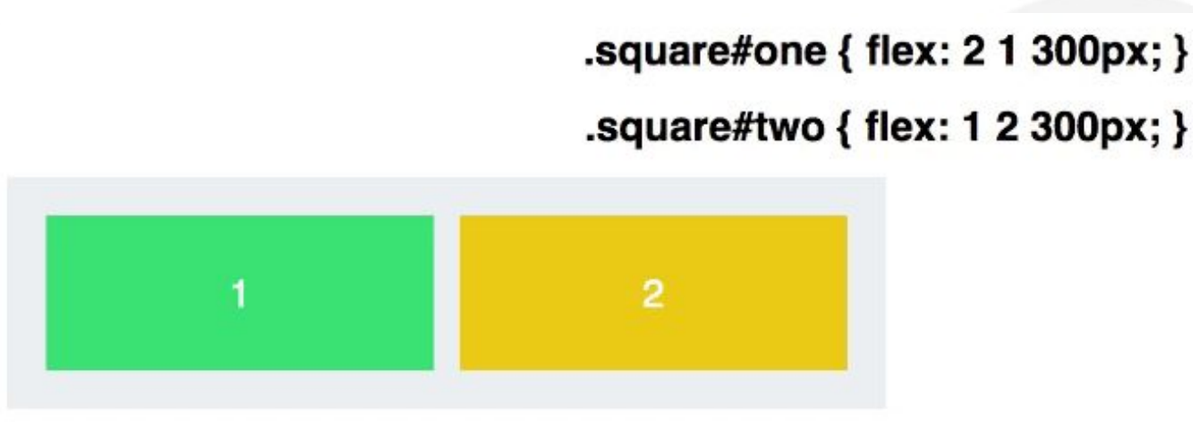

flex-shrink базується на пропорціях. Тобто, якщо у першого блоку flex-shrink дорівнює 6, а в інших він дорівнює 2, то, це означає, що перший блок буде стискатися в три рази швидше, ніж інші.

# flex

Заміняє flex-grow, flex-shrink і flex-basis. Значення за замовчуванням: 0 (grow) 1 (shrink) auto (basis).

```
.square#one { flex: 2 1 300px; }

.square#two { flex: 1 2 300px; }
```

У обох однаковий flex-basis. Це означає, що обидва будуть шириною в 300px (ширина контейнера: 600px плюс margin і padding). Але коли контейнер почне збільшуватися в розмірах, перший блок (з великим flex-grow) буде збільшуватися в два рази швидше, а другий блок (з найбільшим flex-shrink) буде стискатися в два рази швидше:



# flex-wrap

За замовчуванням, якщо батьківський flex блок стискається, всі дочірні елементи так само стискаються. Для того , щоб уникнути такої поведінки, та мати можливість переносити елементи на наступний рядок, у випадку ,якщо вони не вміщаються в поточний, ми використовуємо властивість flex-wrap:wrap. Частіше за все подібний прийом використовують, щоб зробити “сітку товарів” на різних розширеннях екрану