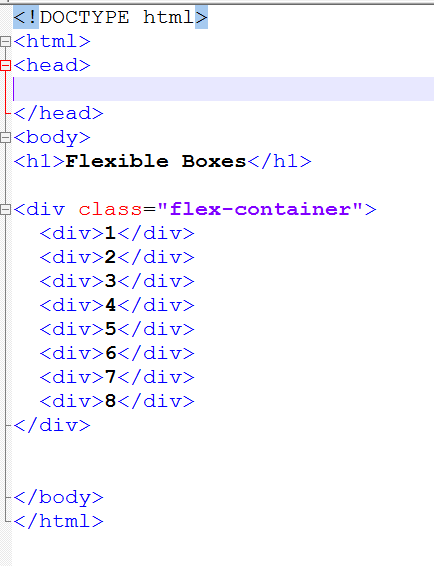
**Блоки Flexbox в CSS**

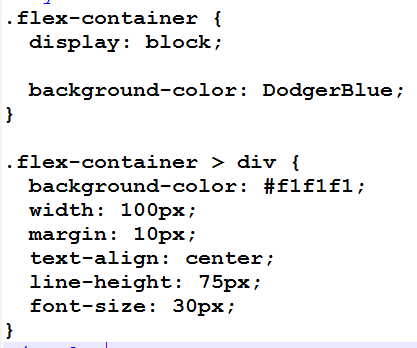
В основу Flexbox положена сетка. Она состоит всего из 2 элементов. Первый элемент – это **flex-контейнер**. Создание flex-контейнера осуществляется посредством добавления к необходимому HTML элементу CSS-свойства display со значением flex или flex-inline. После этого **все непосредственные дочерние элементы flex-контейнера** автоматически становятся **flex-элементами** (2 элемент flexbox сетки).

Разберем на примере с войства flexbox.

Создайте следующий html-документ



И к нему стилевой файл:



Получим

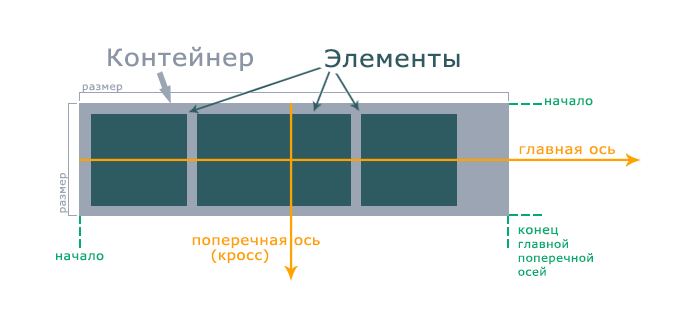


Сделаем div с классом «flex-container» контейнером типа flex, изменив свойство display в значение flex . Получим



Flex-элементы по умолчанию занимают всю высоту flex-контейнера.

Значение flex или flex-inline определяет то, как flex-контейнер будет представлен на странице. Если его необходимо представить как блок, то используйте значение flex. Если элемент необходимо представить как строчный, то используйте значение flex-inline. В этом случае он будет занимать столько места странице, сколько необходимо для отображения его элементов.



Главная ось - главное направление движения элементов внутри контейнера. Направление главной оси можно изменить с помощью свойства [flex-direction](https://wp-kama.ru/id_8045/flex-v-css.html" \l "flex-direction). Обратите внимание, что при смене осей, меняются только направления движения блоков внутри, а начало, конец и размер контейнера остаются прежними.

Начало и конец главной оси - элементы располагаются от начала и до конца контейнера.

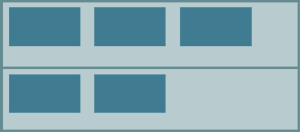
Поперечная ось - направление движения элементов, когда они не умещаются в контейнер по направлению главной оси. Поперечная ось всегда перпендикулярна (⊥) главной.

Начало и конец поперечной оси - по поперечной оси заполняются ряды от начала и до конца контейнера. В каждом таком ряду располагаются элементы.

Размер (главный и поперечный) - базовая величина по которой высчитывается ширина или высота внутренних элементов, если размер указан не точно (указан в процентах или не указан вообще, а элемент должен растянуться или сжаться).

После включения flex свойства, внутри контейнера создаются две оси: главная и поперечная (перпендикулярная (⊥), кросс ось). Все вложенные элементы (первого уровня) выстраиваются по главной оси. По умолчанию главная ось горизонтальная и имеет направление слева направо (→), а кросс ось соответственно вертикальная и направлена сверху вниз (↓).

Еще одной важной особенностью flexbox является наличие рядов в поперечном направлении. Чтобы понять о чем речь, давайте представим что есть главная горизонтальная ось, много элементов и они не «лезут» в контейнер, поэтому переходят на другой ряд. Т.е. контейнер выглядит так: контейнер, внутри него два ряда, в каждом ряду по несколько элементов.



Запомните, что выравнивать по вертикали мы можем не только элементы, но и ряды

**Свойство Flex-Direction**

Изменяет направление главной оси контейнера. Поперечная ось меняется соответственно.

row (default) - направление элементов слева направо (→)

column - направление элементов сверху вниз (↓)

row-reverse - направление элементов справа налево (←)

column-reverse - направление элементов снизу вверх (↑)

Задание

Измените свойство flex-direction  в стилях примера, просмотрите результат.

**Свойство flex-wrap:**

Управляет переносом непомещающихся в контейнер элементов.

nowrap (default) - вложенные элементы располагаются в один ряд (при direction=row) или в одну колонку (при direction=column) независимо от того помещаются они в контейнер или нет.

* wrap - включает перенос элементов на следующий ряд, если они не помещаются в контейнер. Так включается движение элементов по поперечной оси.
* wrap-reverse - тоже что wrap только перенос будет не вниз, а вверх (в обратном направлении).

**Задание**

Добавьте в основной контейнер еще 9 блоков, пронумеруйте их, измените свойство flex-wrap с различными значениями. Как должно быть задано свойство flex-direction?

Свойство flex-flow: direction wrap

Объединяет оба свойства flex-direction и flex-wrap. Они часто используются вместе, поэтому чтобы писать меньше кода было создано свойство flex-flow.

flex-flow принимает значения двух этих свойств, разделенные пробелом. Или можно указать одно значение любого свойства.

Пример

/\* только flex-direction \*/

**flex-flow**: row;

**flex-flow**: row-reverse;

**flex-flow**: column;

**flex-flow**: column-reverse;

/\* только flex-wrap \*/

**flex-flow**: nowrap;

**flex-flow**: wrap;

**flex-flow**: wrap-reverse;

/\* сразу оба значения: flex-direction и flex-wrap \*/

**flex-flow**: row nowrap;

**flex-flow**: column wrap;

**flex-flow**: column-reverse wrap-reverse;

Свойство justify-content:

Выравнивание элементов вдоль основной оси :

Выравнивает элементы по направлению основной (главной) оси: если direction=row, то по горизонтали, а если direction=column, то по вертикали.

* flex-start (default) - элементы будут идти с начала (в конце может остаться место).
* flex-end - элементы выравниваются по концу (место останется в начале)
* center - по центру (место останется слева и права)
* space-between - крайние элементы прижимаются к краям (место между элементами распределяется равномерно)
* space-around - свободное пространство равномерно распределяется между элементами (крайние элементы не прижимаются к краям). Пространство между краем контейнера и крайними элементами будет в два раза меньше чем пространство между элементами в середине ряда.
* space-evenly - тоже что space-around, только расстояние у крайних элементов до краев контейнера такое же как и между элементами.

Задание

Удалите 7 блоков из html-файла, просмотрите применение сврйства justify-content с различными изначениями.

Свойство align-content:

Выравнивает ряды, в которых находятся элементы по поперечной оси. То же что justify-content только для противоположной оси.

Заметка: Работает когда есть как минимум 2 ряда, т.е. при наличии только 1 ряда ничего не произойдет.

Т.е. если flex-direction: row, то это свойство будет выравнивать невидимые ряды по вертикали *¦*. Тут важно заметить, что высота блока должна быть задана жестко и должна быть больше высоты рядов иначе сами ряды будут растягивать контейнер и любое их выравнивание теряет смысл, потому что между ними нет свободного места... А вот когда flex-direction: column, то ряды движутся по горизонтали *→* и ширина контейнера почти всегда больше ширины рядов и выравнивание рядов сразу приобретает смысл.

Это свойство мало где нужно и вместо него чаще используется align-items .

* stretch (default) - ряды растягиваются заполняя строку полностью
* flex-start - ряды группируются в верхней части контейнера (в конце может остаться место).
* flex-end - ряды группируются в нижней части контейнера (место останется в начале)
* center - ряды группируются по центру контейнера (место останется по краям)
* space-between - крайние ряды прижимаются к краям (место между рядами распределяется равномерно)
* space-around - свободное пространство равномерно распределяется между рядами (крайние элементы не прижимаются к краям). Пространство между краем контейнера и крайними элементами будет в два раза меньше чем пространство между элементами в середине ряда.
* space-evenly - тоже что space-around, только расстояние у крайних элементов до краев контейнера такое же как и между элементами.

Задание

Установите для класса «flex-container» основного контейнера высоту в 200 рх, просмотрите результаты, изменяя свойство justify-content

Свойтсво align-items:

Выравнивает элементы по поперечной оси внутри ряда (невидимой строки). Т.е. сами ряды выравниваются через align-content, а элементы внутри этих рядов (строк) через align-items и все это по поперечной оси. По главной оси такого разделения нет, там нет понятия рядов и элементы выравниваются через justify-content.

* stretch (default) - элементы растягиваются заполняя строку полностью
* flex-start - элементы прижимаются к началу ряда
* flex-end - элементы прижимаются к концу ряда
* center - элементы выравниваются по центру ряда
* baseline - элементы выравниваются по базовой линии текста

Задание

Добавьте 9 блоков, высоту основного контейнера установите 600 рх, изучите значения свойства align-items.

### Свойства для элементов контейнера

flex-grow:

Задает коэффициент увеличения элемента при наличии свободного места в контейнере. По умолчанию flex-grow: 0 т.е. никакой из элементов не должен увеличиваться и заполнять свободное место в контейнере.

**По умолчанию flex-grow: 0**

Примеры:

* Если всем элементам указать flex-grow:1, то все они растянуться одинаково и заполнять все свободное место в контейнере.
* Если одному из элементов указать flex-grow:1, то он заполнит все свободное место в контейнере и выравнивания через justify-content работать уже не будут: свободного места нет выравнивать нечего...
* При flex-grow:1. Если один из них имеет flex-grow:2, то он будет в 2 раза больше, чем все остальные
* Если все flex-блоки внутри flex-контейнера имеют flex-grow:3, то они будут одинакового размера
* При flex-grow:3. Если один из них имеет flex-grow:12, то он будет в 4 раза больше, чем все остальные

Как это работает? Допустим, что контейнер имеет ширину 500px и содержит два элемента, каждый из которых имеет базовую ширину 100px. Значит в контейнере остается 300 свободных пикселей. Теперь, если первому элементу укажем flex-grow:2;, а второму flex-grow: 1;, то блоки займут всю доступную ширину контейнера и ширина первого блока будет 300px, а второго 200px. Объясняется это тем, что доступные 300px свободного места в контейнере распределились между элементами в соотношении 2:1, +200px первому и +100px второму.

Заметка: в значении можно указывать дробные числа, например: 0.5 - flex-grow:0.5

Задание

Добавьте в divы контейненра свойства



и просмотрите результат

Свойство flex-shrink:

Задает коэффициент уменьшения элемента. Свойство противоположное flex-grow и определяет как элемент должен сжиматься, если в контейнере не остается свободного места. Т.е. свойство начинает работать, когда сумма размеров всех элементов больше чем размер контейнера.

По умолчанию **flex-shrink:1**

Допустим, что контейнер имеет ширину 600px и содержит два элемента, каждый из которых имеет ширину 300px - flex-basis:300px;. Т.е. два элемента полностью заполняют контейнер. Первому элементу укажем flex-shrink: 2;, а второму flex-shrink: 1;. Теперь уменьшим ширину контейнера на 300px, т.е. элементы должны сжаться на 300px чтобы находится внутри контейнера. Сжиматься они будут в соотношении 2:1, т.е. первый блок сожмется на 200px, а второй на 100px и новые размеры элементов станут 100px и 200px.

Задание

Добавьте к третьему блоку свойство flex-shrink: 0

**Свойство flex-basis:**

Устанавливает базовую ширину элемента - ширину до того как будут высчитаны остальные условия влияющие на ширину элемента. Значение можно указать в *px, em, rem, %, vw, vh* и т.д. Итоговая ширина будет зависеть от базовой ширины и значений flex-grow, flex-shrink и контента внутри блока. При *auto* элемент получает базовую ширину относительно контента внутри него.

По умолчанию: auto

Иногда лучше установить ширину элемента жестко через привычное свойство width. Например, width: 50%; будет означать, что элемент внутри контейнера будет ровно 50%, однако при этом все также будут работать свойства flex-grow и flex-shrink. Такое может быть нужно, когда элемент растягивается контентом внутри него, больше указанного во flex-basis.

Задание

Добавьте к третьему блоку свойство flex-basis:200px

Свойство lex: {grow shrink basis}

Короткая запись трех свойств: *flex-grow flex-shrink flex-basis*.

По умолчанию: **flex: 0 1 auto**

Свойство align-self:

Позволяет изменить свойство align-items, только для отдельного элемента.

По умолчанию: от align-items контейнера

* stretch - элемент растягиваются заполняя строку полностью
* flex-start - элемент прижимаются к началу строки
* flex-end - элемент прижимаются к концу строки
* center - элемент выравниваются по центру строки
* baseline - элемент выравниваются по базовой линии текста

**Задание**

Просмотрите установку данного свойства для блоков

**Свойство order:**

Позволяет менять порядок (позицию, положение) элемента в общем ряду.

По умолчанию: **order: 0**

По умолчанию элементы имеют order: 0 и ставятся в порядке их появления в HTML коде и направления ряда. Но если изменить значение свойства order, то элементы будут выстраиваться в порядке значений: -1 0 1 2 3 .... Например если одному из элементов указать order: 1, то сначала будут идти все нулевые, а потом элемент с 1.

Так можно, например, первый элемент перекинуть в конец, при этом не меняя направление движения остальных элементов или HTML код.