Практическое занятие №6 Позиционирование и видимость элементов

Задание №1

Проверьте запущен ли у Вас Open Server. Папка с материалами урока (css6.loc) должна быть размещена в директории /domains сервера. Заходим на http://css6.loc и начинаем урок.

Задание №2

Свойство ***position*** устанавливает способ позиционирования элемента относительно себя же (значение ***relative***), окна браузера (значение ***fixed***) или других объектов на веб-странице (значение ***absolute***).

Начнём с position: relative (относительное позиционирование). В этом случае положение элемента устанавливается относительно его исходного места. Добавление свойств ***left, top, right и bottom*** изменяет позицию элемента и сдвигает его в ту или иную сторону от первоначального расположения.

Открываем файлы *index-2.html* и *style.css* данного урока.

У нас есть блок красного цвета, который вложен в <body> с рамкой. Добавим ему позиционирование и сдвинем вниз и направо:

```
.block {
    width: 240px;
    height: 120px;
    background-color: red;
    position: relative;
    top: 50px;
    left: 50px;
}
```

Какие выводы мы можем сделать?

Блок сдвинулся относительно себя на указанную величину (top: 50px;), но сдвинулся он «виртуально» то есть все остальные элементы страницы считают, что блок находиться всё там же. Для того что бы это продемонстрировать увеличим высоту блока на 100px.

```
    height: 220px;
```

Это ВАЖНО! По умолчанию у всех элементов на странице задано ***position: static*** это поведение элемента по умолчанию. Свойства ***left, top, right и bottom*** НЕ РАБОТАЮТ с элементами с *position: static.*

Свойства *left, top, right и bottom* «инвертированы» то есть свойство top задаёт как бы отступ сверху и на самом деле сдвигает элемент вниз. Свойство bottom задаёт отступ снизу и сдвигает элемент вверх.

Задание №3

Свойство ***position: absolute*** очень важное и имеет множество нюансов использования. Чаще всего абсолютное позиционирование используется для изменения размещения элемента внутри блока-родителя. Если в position: relative исходной точкой отсчета было положение самого элемента, то в данном случае позиционирование происходит относительно одного из родительских элементов. В этом задании мы рассмотрим это свойство подробнее.

Открываем файлы index-3.html и style.css в папке урока. Зададим красному блоку абсолютное позиционирование:

```
.block {
    width: 320px;
    height: 180px;
    background-color: red;

    position: absolute;
}
```

Что произошло на странице после применения этих стилей? Наш красный блок полностью «выпал их вёрстки». Синий блок, следовавший за ним занял его места, как будто красного блока не существует вовсе. Действительно элементы с абсолютным позиционированием больше не занимают место в потоке элементов страницы.

Теперь рассмотрим особенности позиционирования этих элементов. Добавим к красному блоку следующее позиционирование:

```
    top: 50px;
    left: 50px;
```

Позиционирование происходит относительно элемента <body>. Почему так происходит? Существует правило, так, если у родителя значение *position* установлено как *static (по умолчанию)* или родителя нет, то отсчёт координат ведётся от края окна браузера. Если у родителя значение position задано как relative, то отсчёт координат ведётся от края родительского элемента. Зададим родителю (элементу .wrapper), как показано ниже:

```
.wrapper {
    width: 1024px;
    margin: 0 auto;
    background-color: greenyellow;
    height: 600px;

    position: relative;
}
```

Теперь мы видим что позиционирование происходим относительно элемента «.wrapper».

Задание №4

В свойствах *left, top, right u bottom* можно указывать значение в процентах. В случае абсолютного позиционирования значение 100% будет соответствовать ширине (left, right) блока относительно которого идёт отсчет или его высоты (для top и bottom).

Открываем файлы index-4.html и style.css в папке нашего урока. В блоке «.wrapper» находятся четыре блока-квадрата, используя абсолютное позиционирование и свойства *left, top, right u bottom* разместите эти квадраты по разным углам блока «.wrapper».

Например, для того что бы разместить квадрат в верхнем правом углу нужно задать ему: top: 0; right: 0;

Задание №5

В этом задании мы наверстаем небольшой блок сообщения и узнаем, как использовать негативные значения свойств *left, top, right u bottom.*

Открываем файлы index-5.html и style.css в папке урока. Сначала настраиваем блок «.message» - добавим ему внешние и внутренние отступы, зафиксируем высоту и установим ему относительное позиционирование как показано ниже:

```
.message {
    width: 320px;
    background-color: burlywood;
    padding: 10px;

    position: relative;
    padding-left: 70px;
    margin-left: 50px;
    height: 150px;

}
```

Теперь зададим стили блоку «.avatar» рамку и позиционирование, как показано на рисунке ниже. Обратите внимание на свойство left: -60px; это значит, что левый край элемента должен будет находиться вне, на расстоянии 60px от левого края родителя.

```
.avatar {
    border-radius: 50%;
    width: 100px;
    height: 100px;

    border: 10px solid white;
    position: absolute;
    top: 25px;
    left: -60px;
}
```

Подправим немного стили текста:

```
.text {
    font-family: Arial;
    font-size: 26px;
    color: white;
}
```

Важный момент: на практике довольно редко мы можем поставить фиксированную высоту у элемента, как и в этом случае. Если текст сообщения будет занимать больше места, чем помещается в элементе, то он просто вылезет за пределы элемента (или появится полоса прокрутки, в зависимости от свойства overflow). Для примера вставьте следующий текст:

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque dictum consequat neque ut fringilla. Nulla facilisi. In est dui, faucibus feugiat viverra non, ornare quis enim. Sed velit tellus, fermentum quis enim ut, elementum bibendum leo. Vestibulum ut malesuada arcu, vitae commodo ante. Nunc lacus lacus, gravida a velit in, rutrum molestie dui. Mauris sed ultricies turpis.
```

Поэтому вместо фиксированной высоты height: 150px лучше использовать min-height: 150px.

Задание №6

Свойство position: fixed; (фиксированное позиционирование) так же, как и absolute исключает элемент из потока web страницы, только теперь «фиксированный» элемент позиционируется непосредственно к окну браузера и не меняет своего положения при прокрутке веб-страницы.

Открываем файлы index-6.html и style.css в папке задания. Зададим элементу «.fixed-block» фиксированное позиционирование.

```
.fixed-block {
    width: 320px;
    height: 120px;
    background-color: red;

    position: fixed;
}
```

Попробуйте прокрутить страницу вниз. Как Вы видите элемент прокручивается вместе со страницей. Позиционирование фиксированного элемента происходит тоже относительно окна браузера, что бы убедиться в этом добавим свойства:

```
bottom: 0;
left: 0;
```

Элемент должен был «прилипнуть» нижнему левому краю окна браузера.

Задание №7

Для управление видимостью элементов существует несколько свойств CSS. Самое простое и распространённое свойство, которое мы уже рассматривали это ***display: none;.*** Если применить это свойство к элементу, то этот элемент перестанет отображаться, выпадет из потока элементов и окружающие элементы будут считать, что такого элемента нет.

Откроем файлы index-7.html и style.css в папке задания. Элементу «.block_1» добавим свойство display: none;

```
.block_1 {
    display: none;
}
```

Как Вы видите «.block_1» полностью исчез со страницы и его место заняли другие блоки.

Для того что бы скрыть элемент, но всё таки оставить место под него нужно воспользоваться свойством ***visibility:hidden***. Добавим третьему блоку это свойство:

```
.block_3 {
    visibility: hidden;
}
```

Аналогичным образом действует свойство opacity: 0; с помощью этого свойства задаётся прозрачность элементов от 1 (непрозрачный) до 0 (полностью прозрачный). Сделаем блок «.block_4» полупрозрачным, а блок «.block_5» полностью прозрачным:

```
.block_4 {
    opacity: 0.5;
}

.block_5 {
    opacity: 0;
}
```

Задание №8

Мы уже сталкивались со свойством overflow. Мы использовали его для того что бы заставить блок учитывать размеры плавающих элементов (float) внутри себя. В этом задании мы познакомимся с этим свойством поближе.

Открываем файлы index-8.html и style.css в папке данного урока. На странице у нас есть блок с текстом, текст настолько большой что не помещается. Так же за границы элемента выведен блок с абсолютным позиционированием с фоновым рисунком. Последовательно примените различные свойства overflow показанные ниже:

```
.main {
    /* скрываем все что вне блока во всех направлениях */
    overflow: hidden;

    /* появляется полосы прокрутки */
    overflow: scroll;

    /* по умолчанию, всё показывается */
    overflow: visible;

    /* скрывать только по оси X */
    overflow-x: hidden;

    /* показываеть по оси X */
    overflow-x: visible;

    /* скрывать только по оси Y */
    overflow-y: hidden;

    /* показывать только по оси Y */
    overflow-y: visible;
}
```

Задание №9

В этом задании мы разберём как элементы перекрывают друг друга. Открываем файлы index-9.html и style.css в папке задания. Создаем каркас из блока обёртки и трёх вложенных элементов:

```
<div class="wrapper" >
    <div class="block block1"></div>
    <div class="block block2"></div>
    <div class="block block3"></div>
</div>
```

Задаём ширину и высоту вложенным блокам, а также абсолютное позиционирование.

```
.wrapper {
    position: relative;
    height: 400px;
    width: 400px;
    background-color: grey;
}

.block {
    position: absolute;
}

.block1 {
    width: 300px;
    height: 300px;
    background-color: yellow;
}

.block2 {
    width: 200px;
    height: 200px;
    background-color: blue;
}

.block3 {
    width: 100px;
    height: 100px;
    background-color: black;
}
```

Как мы видим, по умолчанию блоки с абсолютным позиционированием перекрывают друг друга в той последовательности, в которой они описаны в вёрстке. Чем ниже блок в коде, тем выше у него приоритет. Для того что бы изменить приоритет блоков существует свойство z-index, которое увеличивает «вес» блока по оси Z (перпендикулярна экрану). В качестве значения используются целые числа (положительные, отрицательные и ноль). Чем больше значение, тем выше находится элемент по сравнению с теми элементами, у которых оно меньше. При равном значении z-index, на переднем плане находится тот элемент, который в коде HTML описан ниже. Это свойство работает ТОЛЬКО для элементов, у которых значение position задано как absolute, fixed или relative.

Добавим свойство z-index элементу «.block1»:

```
.block1 {
    width: 300px;
    height: 300px;
    background-color: yellow;
    z-index: 1;
}
```

Как вы видите он закрыл собой соседние два блока. Добавим им свойство z-index тоже:

```
.block2 {
    width: 200px;
    height: 200px;
    background-color: blue;
    z-index: 1;
}

.block3 {
    width: 100px;
    height: 100px;
    background-color: black;
    z-index: 1;
}
```

Как Вы уже поняли, при равных значениях *z-index* наложение блоков определяется последовательностью, в которой эти элементы описаны в вёрстке.

Задание №10

Свойства min-width и min-height ограничивают минимальную ширину и высоту элементов. Свойства max-width и max-height ограничивают максимальную ширину и высоту элементов. Эти свойства очень удобно применять для создания «резинового макета», что мы сейчас и будем делать. Открываем файлы index-10.html и style.css в папке задания. Создаём элемент-обёртку:

```
<div class="wrapper"></div>
```

Отцентрируем его и зададим размеры:

```
.wrapper {
    background-color: gray;
    max-width: 1200px;
    min-width: 640px;
    height: 2000px;
    margin: 0 auto;
}
```

На данный момент у нас уже есть блок, который подстраивается под размеры браузера. Понаблюдайте за его поведением при изменении ширины окна браузера.

Создадим два блока для меню и основного контента:

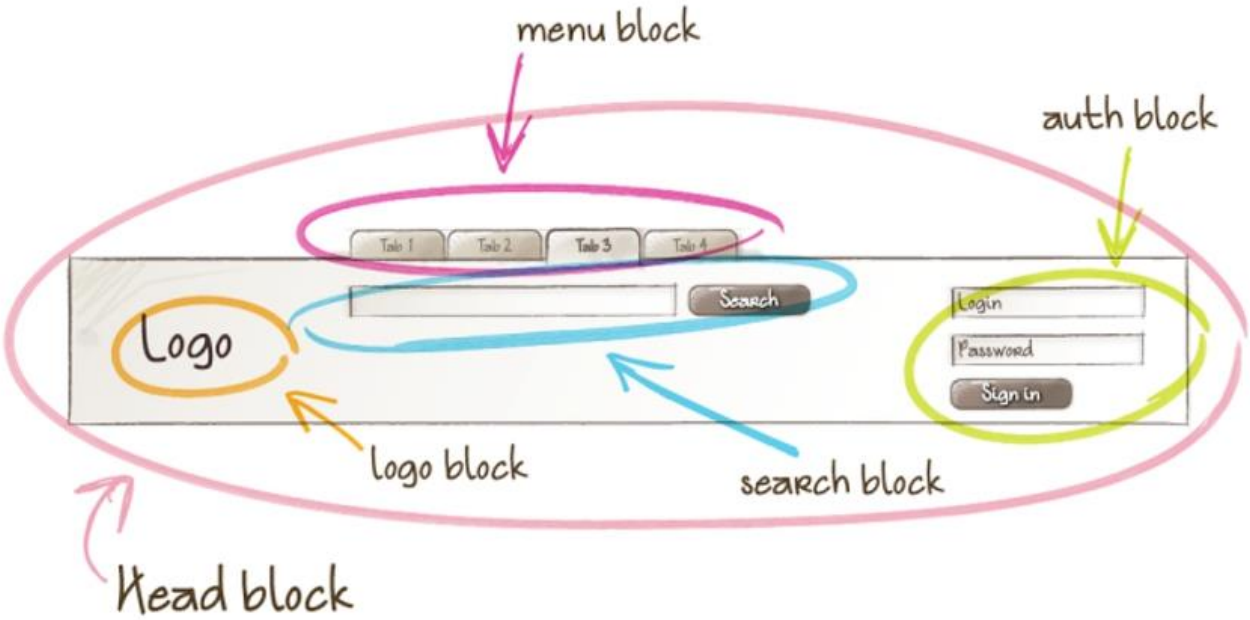
```
<div class="wrapper">
    <div class="menu"></div>
    <div class="content" ></div>
</div>
```

Теперь нам нужно что бы ширина блока «.menu» оставалась всегда одинаковой, а элемент «.content» менял свою ширину. Для этого делаем блок «.menu» плавающим и с фиксированной шириной, а блок «.content» должен иметь margin-left равный ширине блока «.menu». Набрасываем следующие стили:

```
.menu {
    float: left;
    width: 300px;
    background-color: green;
    height: 600px;
}

.content {

    margin-left: 300px;
    background-color: navy;
    height: 1000px;
}
```

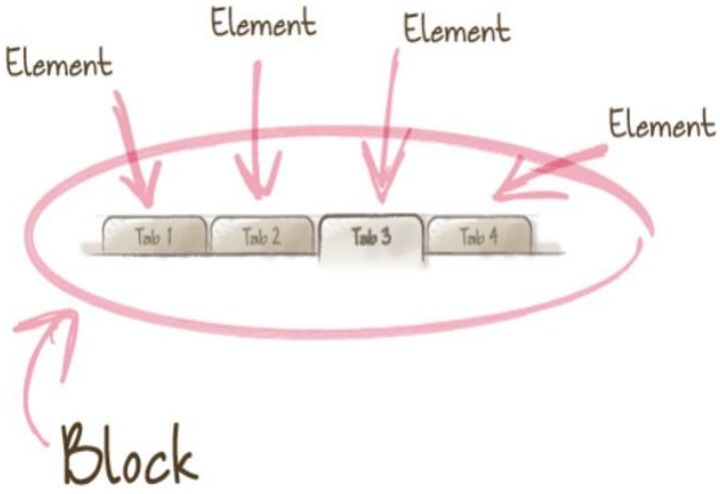
Задание №11

БЭМ – (блок, элемент, модификатор) это одна из множества методологий вёрстки. Суть данной методики состоит в том, что мы страницу разбиваем на ***блоки***. При этом большие блоки могут включать в себя другие блоки поменьше, как на изображении ниже:



У всех блоков определяется класс, название этого класса должно быть односложным, например: «*.menu», «.card», «.personalblock», «.logo»* и так далее.

Элемент – это часть блока. Без блока элементов не существует. Название элемента состоит из названия блока, затем идет разделитель два нижних подчеркивания (__), а затем имя элемента. Например: «*.block__element*», «*.search__input*», «*.card__header*», «*.menu__list*».



Модификатор - это класс, при “навешивании” которого изменяется отображение блока или элемента. Имя модификатора формируется из имени блока, разделителя (__), имени элемента (если применен к элементу), разделителя(_) и имени модификатора, например: *«.block__element_modifier», «.search__input_grey», «cart_shadowed».*

Когда мы верстаем с использованием БЭМ, мы должны соблюдать ряд правил:

- Верстка должна быть модульной (любой блок можно вырвать из одного места страницы и вставить в другое, так что бы он не “развалился” и не поломал ничего вокруг).
- На странице нет “голых” тегов! Все теги имеют свой класс, и стилизуются конкретно по нему.
- Нет селекторов по тегам.

- Не использовать относительных селекторов (*, >, +)
- Зависимость классов допускается только от модификаторов блока!
- Именование классов формируется по правилам синтаксиса БЭМ.

Какие плюсы у данного подхода:

- Прозрачная и понятная система наименования элементов
- Модульность, можно переносить элементы целыми кусками (модулями)
- Гибкость за счет использования модификаторов

Минусы:

- Строгие правила, которые нужно придерживаться, легко сойти с пути
- Если неверно использовать БЕМ, то появляется очень много «дублирования» стилей (многократным повторением одних и тех же стилей). Нужно сохранить баланс между модульностью и дублированием.

Попробуем немного на практике применить БЭМ. Открываем файлы index-11.html и style.css в папке текущего урока. Предположим, что у нас есть заказчик, который хочет, чтобы мы создали четыре блока, каждый из которых состоит из двух красный квадратов, вложенных в друг друга. Создаем их, как показано ниже:

```
<div class="block">
    <div class="block__inner"></div>
</div>

<div class="block">
    <div class="block__inner"></div>
</div>

<div class="block">
    <div class="block__inner"></div>
</div>

<div class="block">
    <div class="block__inner"></div>
</div>
```

Обратите что мы используем классы для каждого элемента. Набрасываем стили:

```css
/* обьявляем блок */
.block {
    width: 200px;
    height: 200px;
    border: 5px solid red;
    margin: 30px;
    display: inline-block;
    vertical-align: top;

    /* стили для размещения внутренего блока по центру */
    text-align: center;
    line-height: 200px;
}

/* элементы блока, сразу после обьявления блока */
.block__inner {
    width: 100px;
    height: 100px;
    border: 5px solid red;
    display: inline-block;
    vertical-align: middle;
}
```

Теперь заказчик хочет, чтобы третий блок был кругом, а у последнего блока круг был внутри. Мы добавляем модификаторы, как показано ниже:

```html
<div class="block">
    <div class="block__inner"></div>
</div>

<div class="block">
    <div class="block__inner"></div>
</div>

<div class="block block_circle">
    <div class="block__inner"></div>
</div>

<div class="block">
    <div class="block__inner block__inner_circle"></div>
</div>
```

Изменяем CSS:

```
/* обьявляем блок */
.block {
    width: 200px;
    height: 200px;
    border: 5px solid red;
    margin: 30px;
    display: inline-block;
    vertical-align: top;

    /* стили для размещения внутренего блока по центру */
    text-align: center;
    line-height: 200px;
}

/* модификатор блока, располагается после обьявления блока */
.block_circle {
    border-radius: 50%;
}

/* элементы блока, сразу после обьявления блока */
.block__inner {
    width: 100px;
    height: 100px;
    border: 5px solid red;
    display: inline-block;
    vertical-align: middle;
}

/* модификато элемента, распологается после обьявления элемента */
.block__inner_circle {
    border-radius: 50%;
}
```

ЭТО ВАЖНО! В БЭМ стили должны располагаться определённым образом: сначала идут объявления блоков и их модификаторов, затем элементов этого блока и их модификаторов.

Теперь заказчик хочет, чтобы у первого и четвёртого блока внутренняя рамка была синего цвета. Создаём модификатор:

```
.block__inner_blue {
    border-color: navy;
}
```

И добавляем его нужным элементам самостоятельно.

Теперь мы можем совмещать модификаторы, например сделайте так что бы у последнего блока внутри был круг (.block__inner_circle) синего цвета (.block__inner_blue).

Задание №12

В этом задании мы познакомимся FlexBox. FlexBox (CSS Flexible Box Layout Module) – это новейший стандарт, который всё еще разрабатывается с 2009 года. Окончательная спецификация еще не принята. По данным сайта caniuse.com, FlexBox поддерживается всеми современными браузерами http://caniuse.com/#search=flexbox .

FlexBox должен заменить множество методов и приёмов, накопившихся за время существования и эволюции CSS и HTML. В начале веб-страницы были похожи на однопоточные текстовые документы, чуть позже разбиение страницы на блоки стали делать таблицами, затем стало модным верстать float-ами, а после добавились еще и приемы с inline-block. В итоге мы получили в наследство гремучую смесь всех этих приемов.

flex-контейнер – это родительский элемент, который содержит в себе flex-элементы. Он может быть блочным (display:flex) или же строчным (display:inline-flex). По умолчанию flex-контейнер содержит только одну очередь flex-блоков (одну строку или колонку в зависимости от расположения главной оси).

flex-блоки или (flex-элементы) – это элементы, которые находятся внутри flex-контейнера.

Сейчас мы рассмотри свойства, которые относятся, только для flex-контейнера. Открываем файлы index-12.html и style.css в папке урока. Так же открываем страницу урока в браузере. В начале урока у нас есть четыре черных блока вложенных в контейнер. Создадим flex-контейнер:

```
.container {
    display: flex;
}
```

Как Вы видите поведение элементов сразу изменилось. По умолчанию flex-контейнер содержит только одну очередь (строку или колонку) flex-элементов.

Свойство ***flex-direction*** задаёт направление основных осей в контейнере и тем самым определяет положение flex-элементов в flex-контейнере.

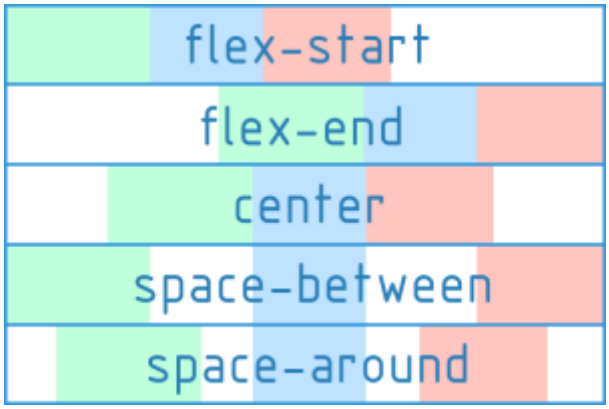
Доступные значения *flex-direction*:

- *row*: строка, прямой порядок элементов (по умолчанию)
- *row-reverse*: строка, обратный порядок
- *column*: колонка, прямой порядок
- *column-reverse*: колонка, обратный порядок

Последовательно применим различные значения flex-direction, как показано ниже и понаблюдайте как изменяется поведение элементов:

```
.container {
    display: flex;

    flex-direction: row-reverse;

    flex-direction: column;

    flex-direction: column-reverse;

}
```

Свойство ***justify-content*** определяет, как браузер распределяет пространство вокруг flex-элементов вдоль главной оси контейнера. Основные значения этого свойства показаны на рисунке ниже:

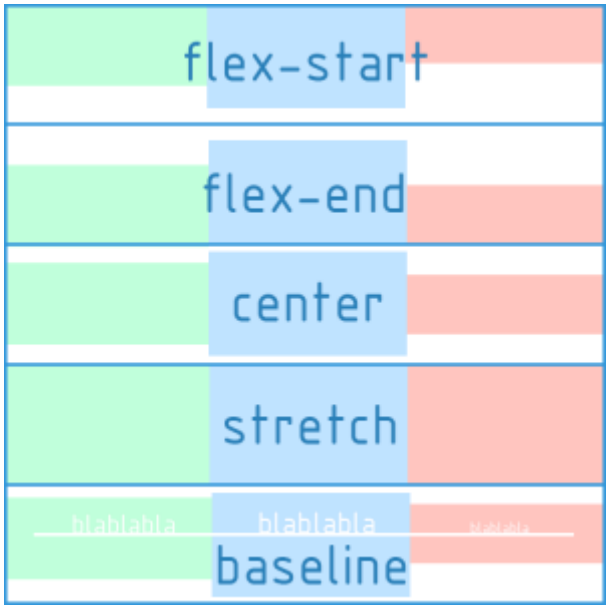


Это ВАЖНО! Свойство ***justify-content*** не работает с элементами выстроенными в колонку, так что нам придется сперва удалить свойство ***flex-direction*** или установить его значение в *flex-direction:row;*

Последовательно применим различные значения *justify-content*, как показано ниже и понаблюдайте за изменением поведения элементов:

```
.container {
    display: flex;

    flex-direction: row;

    justify-content: flex-start;

    justify-content: flex-end;

    justify-content: center;

    justify-content: space-around;

    justify-content: space-between;

}
```

Свойство ***align-items*** определяет то, как будут выровнены элементы вдоль поперечной оси. Фактически это вертикальное выравнивание внутри flex-контейнера. Основные значения этого свойства показаны на рисунке ниже:



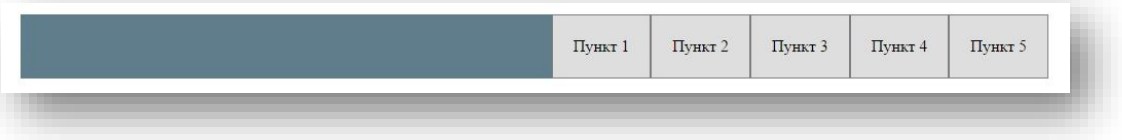
Самостоятельно делаем так, чтобы внутренние (чёрного цвета) блоки имели различную высоту(*height)* и размер текста (*font-size*). Это необходимо для наглядного действия свойства *align-items*. Самостоятельно последовательно примените значения свойства *align-items*, которые показаны на рисунке выше.

Задание №13

С помощью свойств FlexBox изученных в прошлом уроке, разместите элемент «.container__inner» (красный круг) ровно посередине своего родителя, как по горизонтали, так и по вертикали.

Задание №14

В этом задании Вы сделаете простое горизонтальное меню с помощью FlexBox. Образец на рисунке ниже:



Главное концепция, стиль и цвет можно изменять по желанию.

Задание №15

В данном задании Вы создаёт каркас карточек товаров (элементы «.block») с помощью FlexBox. Для этого нам понадобиться свойство ***flex-wrap,*** которое разрешает перенос flex-элементов на новую строку (flex-wrap: wrap;). Создайте flex-контейнер и разместите в нем flex-элементы в многострочном порядке.

Обратите внимание, что в последнем ряду у нас два элемента и пустое место справа, если мы хотим что бы блоки были всегда растянуты на всю ширину flex-контейнера, то мы можем воспользоваться следующими свойствами flex-элементов:

flex-basis – задает «базовый» изначальный размер flex-элемента. Может быть задан в любых единицах измерения длинны (px, em, %, …) или auto(по умолчанию). Если задан как auto – за основу берутся размеры блока (width, height).

flex-grow – свойства определяет степень роста flex-элемента, в отношении к соседним flex-элементам, принимает безразмерное значение (по умолчанию 0).

Устанавливаем данных стили для flex-элементов:

```
.block {
    flex-basis: 320px;
    flex-grow: 1;
}
```

Теперь в последнем ряду два элемента заняли всё доступное пространство.

Задание №16

В этом задании мы создадим простой «резиновый» макет, используя FlexBox. Создаём каркас макета:

```
<header></header>
<main>

    <nav></nav>
    <article></article>

</main>
<footer></footer>
```

Сразу определяем стили для шапки и подвала сайта:

```css
header{
    height: 50px;
    background-color: yellow;
}

footer {
    height: 100px;
    background-color: black;
}
```

Теперь используя уже знакомые свойства flexbox создаем две колонки сайта:

```css
main {
    display: flex; /* создаём контейнер */
    min-width: 640px; /* ограничиваем минимальную шиниру контейнера  */
    flex-direction: row;
    align-items: stretch; /* одинаковая высота колонок */
}

nav {
    flex-basis: 320px; /* базовая ширина */
    flex-grow: 0; /* запрещаем рост */
    background-color: red;
}

article {
    height: 700px; /* высота для наглядности*/
    flex-grow: 1;
    background-color: navy;
}
```

Задание №17

Используя знания, полученные в предыдущем задании, создайте самостоятельно «резиновый» трёх-колоночный макет, используя следующие вводные данные:

- 1 колонка – это меню, ширина 200px.
- 2 колонка – переменная ширина
- 3 колонка – информация и реклама – ширина 300px
- Минимальная доступная ширина сайта 1000px