

Лабораторная работа № 5

Тема: Объединение JavaScript и CSS. FlexBox. Grid Layout. Движущиеся элементы.

Цель: изучить принцип объединения JavaScript и CSS, разобраться в верстке страницы с помощью FlexBox и Grid, научиться применять полученные знания на практике.

Краткая теория

1.1 СПОСОБЫ ДОБАВЛЕНИЯ СТИЛЕЙ НА СТРАНИЦУ

Для добавления стилей на веб-страницу существует несколько способов, которые различаются своими возможностями и назначением. Далее рассмотрим их подробнее.

Связанные стили

При использовании связанных стилей описание селекторов и их значений располагается в отдельном файле, как правило, с расширением css, а для связывания документа с этим файлом применяется тег `<link>`. Данный тег помещается в контейнер `<head>`, как показано в примере:

```
<!DOCTYPE HTML>
<html>
<head>
  <meta charset="utf-8">
  <title>Стили</title>
  <link rel="stylesheet" href="mysite.css">
  <link rel="stylesheet" href="http://www.htmlbook.ru/main.css">
</head>
<body>
  <h1>Заголовок</h1>
  <p>Текст</p>
</body>
</html>
```

Значение атрибута тега `<link>` — `rel` остаётся неизменным независимо от кода, как приведено в данном примере. Значение `href` задаёт путь к CSS-файлу, он может быть задан как относительно, так и абсолютно. Заметьте, что таким образом можно подключать таблицу стилей, которая находится на другом сайте.

Содержимое файла `mysite.css` подключаемого посредством тега `<link>` приведено в примере:

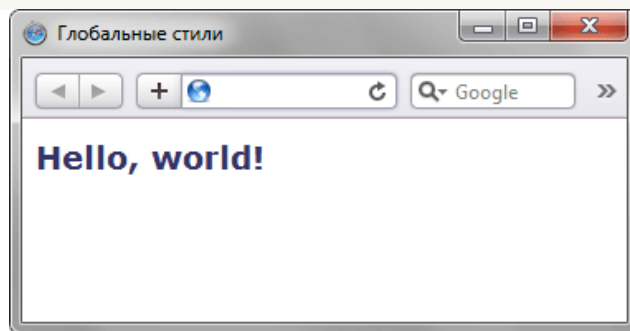
```
H1 {
  color: #000080;
  font-size: 200%;
  font-family: Arial, Verdana, sans-serif;
  text-align: center;
}
P {
  padding-left: 20px;
}
```

Как видно из данного примера, файл со стилем не хранит никаких данных, кроме синтаксиса CSS. В свою очередь и HTML-документ содержит только ссылку на файл со стилем, т. е. таким способом в полной мере реализуется принцип разделения кода и оформления сайта. Поэтому использование связанных стилей является наиболее универсальным и удобным методом добавления стиля на сайт. Ведь стили хранятся в одном файле, а в HTML-документах указывается только ссылка на него.

Глобальные стили

При использовании глобальных стилей свойства CSS описываются в самом документе и располагаются в заголовке веб-страницы. По своей гибкости и возможностям этот способ добавления стиля уступает предыдущему, но также позволяет хранить стили в одном месте, в данном случае прямо на той же странице с помощью контейнера `<style>`, как показано в примере:

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>Глобальные стили</title>
<style>
H1 {
font-size: 120%;
font-family: Verdana, Arial, Helvetica, sans-serif;
color: #333366;
}
</style>
</head>
<body>
<h1>Hello, world!</h1>
</body>
</html>
```



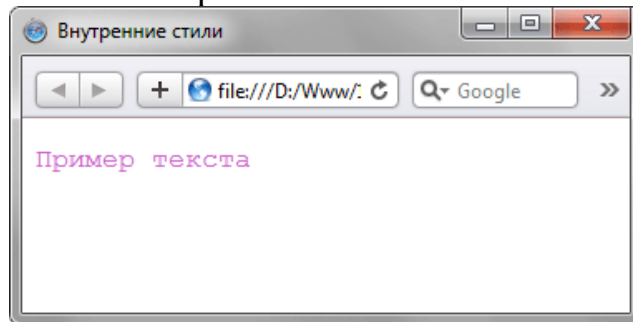
Внутренние стили

Внутренний или встроенный стиль является по существу расширением для одиночного тега используемого на текущей веб-странице. Для определения стиля используется атрибут `style`, а его значением выступает набор стилевых правил:

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
```

```
<title>Внутренние стили</title>
</head>
<body>
  <p style="font-size: 120%; font-family: monospace; color: #cd66cc">Пример текста</p>
</body>
</html>
```

В данном примере стиль тега `<p>` задаётся с помощью атрибута `style`, в котором через точку с запятой перечисляются стилевые свойства:

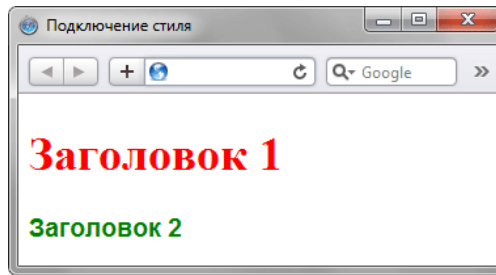


Внутренние стили рекомендуется применять на сайте ограниченно или вообще отказаться от их использования. Дело в том, что добавление таких стилей увеличивает общий объём файлов, что ведет к повышению времени их загрузки в браузере, и усложняет редактирование документов для разработчиков.

Все описанные методы использования CSS могут применяться как самостоятельно, так и в сочетании друг с другом. В этом случае необходимо помнить об их иерархии. Первым имеет приоритет внутренний стиль, затем глобальный стиль и в последнюю очередь связанный стиль. В примере применяется сразу два метода добавления стиля в документ.

```
<!DOCTYPE HTML>
<html>
<head>
  <meta charset="utf-8">
  <title>Подключение стиля</title>
  <style>
    H1 {
      font-size: 120%;
      font-family: Arial, Helvetica, sans-serif;
      color: green;
    }
  </style>
</head>
<body>
  <h1 style="font-size: 36px; font-family: Times, serif; color: red">Заголовок 1</h1>
  <h1>Заголовок 2</h1>
</body>
</html>
```

В данном примере первый заголовок задаётся красным цветом размером 36 пикселей с помощью внутреннего стиля, а следующий — зелёным цветом через таблицу глобальных стилей.



Импорт CSS

В текущую стилевую таблицу можно импортировать содержимое CSS-файла с помощью команды **@import**. Этот метод допускается использовать совместно со связанными или глобальными стилями, но никак не с внутренними стилями. Общий синтаксис следующий.

```
@import url("имя файла") типы носителей;  
@import "имя файла" типы носителей;
```

После ключевого слова **@import** указывается путь к стилевому файлу одним из двух приведенных способов — с помощью **url** или без него. В примере показано, как можно импортировать стиль из внешнего файла в таблицу глобальных стилей.

```
<!DOCTYPE HTML>  
<html>  
  <head>  
    <meta charset="utf-8">  
    <title>Импорт</title>  
    <style>  
      @import url("style/header.css");  
      H1 {  
        font-size: 120%;  
        font-family: Arial, Helvetica, sans-serif;  
        color: green;  
      }  
    </style>  
  </head>  
  <body>  
    <h1>Заголовок 1</h1>  
    <h2>Заголовок 2</h2>  
  </body>  
</html>
```

В данном примере показано подключение файла **header.css**, который расположен в папке **style**.

Аналогично происходит импорт и в файле со стилем, который затем подключается к документу:

```
@import "/style/print.css";  
@import "/style/palm.css";  
BODY {  
  font-family: Arial, Verdana, Helvetica, sans-serif;  
  font-size: 90%;  
  background: white;  
  color: black;
```

```
}
```

В данном примере показано содержимое файла `mysite.css`, который добавляется к нужным документам способом, показанным в примере 3.1, а именно с помощью тега `<link>`.

1.2 ПОДКЛЮЧЕНИЕ JavaScript к HTML

Чтобы ваша первая программа (или сценарий) JavaScript запустилась, ее нужно внедрить в HTML-документ. Сценарии внедряются в HTML-документ различными стандартными способами:

- поместить код непосредственно в атрибут события HTML-элемента;
- поместить код между открывающим и закрывающим тегами `<script>`;
- поместить все ваши скрипты во внешний файл (с расширением `.js`), а затем связать его с документом HTML.

JavaScript в элементе script

Самый простой способ внедрения JavaScript в HTML-документ – использование тега `<script>`. Теги `<script>` часто помещают в элемент `<head>`, и ранее этот способ считался чуть ли не обязательным. Однако в наши дни теги `<script>` используются как в элементе `<head>`, так и в теле веб-страниц.

Таким образом, на одной веб-странице могут располагаться сразу несколько сценариев. В какой последовательности браузер будет выполнять эти сценарии? Как правило, выполнение сценариев браузерами происходит по мере их загрузки. Браузер читает HTML-документ сверху вниз и, когда он встречает тег `<script>`, рассматривает текст программы как сценарий и выполняет его. Остальной контент страницы не загружается и не отображается, пока не будет выполнен весь код в элементе `<script>`.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
</head>
<body>
<p>Это обычный HTML документ</p>
<script language="JavaScript">
    alert("Привет, мир!");
</script>
<p>Выходим обратно в HTML</p>
</body>
</html>
```

Обратите внимание: мы указали атрибут `language` тега `<script>`, указывающий язык программирования, на котором написан сценарий. Значение атрибута `language` по умолчанию – JavaScript, поэтому, если вы используете скрипты на языке JavaScript, то вы можете не указывать атрибут `language`.

JavaScript в атрибутах событий HTML-элементов

Вышеприведенный сценарий был выполнен при открытии страницы и вывел строку: «Привет, мир!». Однако не всегда нужно, чтобы выполнение сценария начиналось сразу при открытии страницы. Чаще всего требуется, чтобы программа запускалась при определенном событии, например при нажатии какой-то кнопки.

В следующем примере функция JavaScript помещается в раздел `<head>` HTML-документа. Вот пример HTML-элемента `<button>` с атрибутом события, обеспечивающим реакцию на щелчки мышью. При нажатии кнопки генерируется событие `onclick`.

```
<!DOCTYPE html>
<html>
<head>
<script>
  function myFunction() {
    document.getElementById("demo").innerHTML = "Привет, javascript!";
  }
</script>
</head>
<body>
<p id="demo">Привет, мир!</p>
<button type="button" onclick="myFunction()">Кликни меня</button>
</body>
</html>
```

Внешний JavaScript

Если JavaScript-кода много – его выносят в отдельный файл, который, как правило, имеет расширение `.js`.

Чтобы включить в HTML-документ JavaScript-код из внешнего файла, нужно использовать атрибут `src` (source) тега `<script>`. Его значением должен быть URL-адрес файла, в котором содержится JS-код:

```
<script src="/scripts/script.js"></script>
```

В этом примере указан абсолютный путь к файлу с именем `script.js`, содержащему скрипт (из корня сайта). Сам файл должен содержать только JavaScript-код, который иначе располагался бы между тегами `<script>` и `</script>`.

По аналогии с элементом `` атрибуту `src` элемента `<script>` можно назначить полный URL-адрес, не относящийся к домену текущей HTML-страницы:

```
<script src="http://www.somesite.com/script.js"></script>
```

Чтобы подключить несколько скриптов, используйте несколько тегов:

```
<script src="/scripts/script1.js"></script>
<script src="/scripts/script2.js"></script>
```

...

Примечание: Элемент `<script>` с атрибутом `src` не может содержать дополнительный JavaScript-код между тегами `<script>` и `</script>`, хотя внешний сценарий выполняется, встроенный код игнорируется.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
</head>
<body>
<script src="script.js">
  alert('Привет, мир!');
</script>
<p>При наличии атрибута src внутренняя часть тега script игнорируется!</p>
</body>
</html>
```

Независимо от того, как JS-код включается в HTML-документ, элементы `<script>` интерпретируются браузером в том порядке, в котором они расположены в HTML-документе. Сначала интерпретируется код первого элемента `<script>`, затем браузер приступает ко второму элементу `<script>` и т. д.

Внешние скрипты практичны, когда один и тот же код используется во многих разных веб-страницах. Браузер скачает js-файл один раз и в дальнейшем будет брать его из своего кеша, благодаря чему один и тот же скрипт, содержащий, к примеру, библиотеку функций, может использоваться на разных страницах без полной перезагрузки с сервера. Кроме этого, благодаря внешним скриптам, упрощается сопровождение кода, поскольку вносить изменения или исправлять ошибки приходится только в одном месте.

Примечание: Во внешние файлы копируется только JavaScript-код без указания открывающего и закрывающего тегов `<script>` и `</script>`.

Расположение тегов `<script>`

Вы уже знаете, что браузер читает HTML-документ сверху вниз и, начинает отображать страницу, показывая часть документа до тега `<script>`. Встретив тег `<script>`, переключается в JavaScript-режим и выполняет сценарий. Закончив выполнение, возвращается обратно в HTML-режим и отображает оставшуюся часть документа.

Это наглядно демонстрирует следующий пример. Метод `alert()` выводит на экран модальное окно с сообщением и приостанавливает выполнение скрипта, пока пользователь не нажмёт «ОК»:

```
<!DOCTYPE html>
<html>
<body>
<p>Начало контента...</p>
<script src="script.js"></script>
<p>...Продолжение контента</p>
</body>
</html>
```


Если на странице используется много скриптов JavaScript, то могут возникнуть длительные задержки при загрузке, в течение которых пользователь видит пустое окно браузера. Поэтому считается хорошей практикой все ссылки на JavaScript-сценарии указывать после контента страницы перед закрывающим тегом `<body>`:

```
<body>
<!-- Контент страницы -->
<script src="script1.js"></script>
<script src="script2.js"></script>
</body>
```

Такое расположение сценариев позволяет браузеру загружать страницу быстрее, так как сначала загрузится контент страницы, а потом будет загружаться код сценария. Для пользователей это предпочтительнее, потому что страница полностью визуализируется в браузере до обработки JavaScript-кода.

Отложенные и асинхронные сценарии

Как отмечалось ранее, по умолчанию файлы JavaScript-кода прерывают синтаксический анализ (парсинг) HTML-документа до тех пор, пока скрипт не будет загружен и выполнен, тем самым увеличивая промежуток времени до первой отрисовки страницы. Возьмём пример, в котором элемент `<script>` расположен где-то в середине страницы:

```
<html>
<body>
  какой-то текст...
  <script src="script.js"></script>
  Этот текст не будет показан, пока браузер не выполнит script.js.
</body>
</html>
```

В этом примере, пока браузер не загрузит и не выполнит `script.js`, он не покажет часть страницы под ним. Такое поведение браузера называется «синхронным» и может доставить проблемы, если мы загружаем несколько JavaScript-файлов на странице, так как это увеличивает время её отрисовки.

А что, если HTML-документ на самом деле не зависит от этих JS-файлов, а разработчик желает контролировать то, как внешние файлы загружаются и выполняются?

Кардинально решить проблему загрузки скриптов помогут атрибуты `async` и `defer` элемента `<script>`.

Атрибут `async`

`Async` используется для того, чтобы указать браузеру, что скрипт может быть выполнен «асинхронно».

При обнаружении тега `<script async src="...">` браузер не останавливает обработку HTML-документа для загрузки и выполнения скрипта, выполнение может произойти после того, как скрипт будет получен параллельно с разбором документа. Когда скрипт будет загружен – он выполнится.

Для сценариев с атрибутом `async` не гарантируется выполнение скриптов в порядке их добавления, например:

```
<html>
<head>
</head>
<body>
  <script async src="script1.js"></script>
  <script async src="script2.js"></script>
  <!-- контент -->
</body>
</html>
```

В примере второй скрипт может быть выполнен перед первым, поэтому важно, чтобы между этими сценариями не было зависимостей.

Примечание: Атрибут `async` используется, если нужно разрешить браузеру продолжить загрузку страницы, не дожидаясь завершения загрузки и выполнения сценария.

Атрибут defer

Атрибут `defer` откладывает выполнение скрипта до тех пор, пока вся HTML-страница не будет загружена полностью.

Как и при асинхронной загрузке скриптов — JS-файл может быть загружен, в то время как HTML-документ ещё грузится. Однако, даже если скрипт будет полностью загружен ещё до того, как браузер закончит обработку страницы, он не будет выполнен до тех пор, пока HTML-документ не обработается до конца.

```
<html>
<head>
  <script defer src="script1.js"></script>
  <script defer src="script2.js"></script>
</head>
<body>
  <!-- контент -->
</body>
</html>
```

Несмотря на то, что в приведенном примере теги `<script defer src="...">` включены в элемент `<head>` HTML-документа, выполнение сценариев не начнется, пока браузер не дойдет до закрывающего тега `</html>`.

Кроме того, в отличие от `async`, относительный порядок выполнения скриптов с атрибутом `defer` будет сохранён.

Применение атрибута `defer` бывает полезным, когда в коде скрипта предусматривается работа с HTML-документом, и разработчик должен быть уверен, что страница полностью получена.

Примечание: Атрибуты `async` и `defer` поддерживаются только для внешних файлов сценариев, т.е. работают только при наличии атрибута `src`.

2.1 FLEXBOX

2.1.1 ЧТО ТАКОЕ FLEXBOX. FLEX CONTAINER

Flexbox - это общее название для модуля **Flexible Box Layout**, который имеется в CSS3. Данный модуль определяет особый режим компоновки/верстки пользовательского интерфейса, который называется **flex layout**. В этом плане Flexbox предоставляет иной подход к созданию пользовательского интерфейса, который отличается от табличной или блочной верстки.

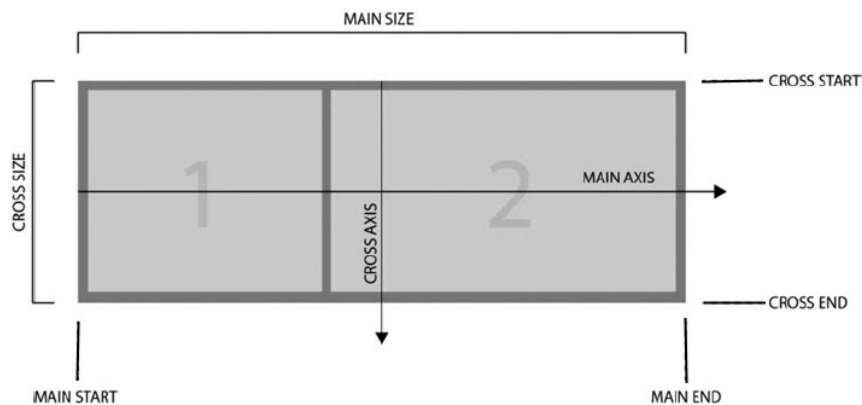
Благодаря Flexbox проще создавать сложные, комплексные интерфейсы, где мы с легкостью можем переопределять направление и выравнивание элементов, создавать адаптивные табличные представления. Кроме того, Flexbox довольно прост в использовании. Единственная проблема, которая может возникнуть при его применении, - это кроссбраузерность. Например, в Internet Explorer поддержка Flexbox и то частичная появилась только в последней версии - IE11. В то же время все современные браузеры, в том числе Microsoft Edge, Opera, Google Chrome, Safari, Firefox, имеют полную поддержку данного модуля.

Основными составляющими компоновки flexbox являются flex-контейнер (flex container) и flex-элементы (flex items). **Flex container** представляет некоторый элемент, внутри которого размещены flex-элементы.

Основные понятия

Прежде чем переходить к изучению верстки flexbox, стоит рассмотреть некоторые основные понятия.

Одно из ключевых понятий представляет **main axis** или центральная ось. Это условная ось во flex-контейнере, вдоль которой позиционируются flex-элементы.



Элементы в контейнере могут располагаться по горизонтали в виде строки и по вертикали в виде столбца. В зависимости от типа расположения будет меняться и центральная ось. Если расположение в виде строки, то центральная ось направлена горизонтально слева направо. Если расположение в виде столбца, то центральная ось направлена вертикально сверху вниз.

Термины **main start** и **main end** описывают соответственно начало и конец центральной оси, а расстояние между ними обозначается как **main size**.

Кроме основной оси существует также поперечная ось или **cross axis**. Она перпендикулярна основной. При расположении элементов в виде строки cross axis направлена сверху вниз, а при расположении в виде столбца она направлена слева

направо. Начало поперечной оси обозначается как **cross start**, а ее конец - как **cross end**. Расстояние между ними описывается термином **cross size**.

То есть, если элементы располагаются в строку, то **main size** будет представлять ширину контейнера или элементов, а **cross size** - их высоту. Если же элементы располагаются в столбик, то, наоборот, **main size** представляет высоту контейнера и элементов, а **cross size** - их ширину.

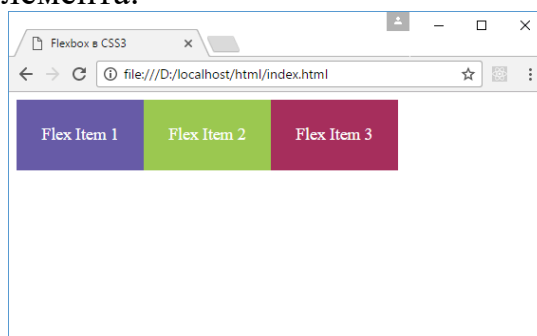
Создание flex-контейнера

Для создания flex-контейнера необходимо присвоить его стилевому свойству **display** одно из двух значений: **flex** или **inline-flex**.

Создадим простейшую веб-страницу, которая применяет flexbox:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Flexbox в CSS3</title>
    <style>
      .flex-container {
        display: flex;
      }
      .flex-item {
        text-align:center;
        font-size: 1.1em;
        padding: 1.5em;
        color: white;
      }
      .color1 {background-color: #675BA7;}
      .color2 {background-color: #9BC850;}
      .color3 {background-color: #A62E5C;}
    </style>
  </head>
  <body>
    <div class="flex-container">
      <div class="flex-item color1">Flex Item 1</div>
      <div class="flex-item color2">Flex Item 2</div>
      <div class="flex-item color3">Flex Item 3</div>
    </div>
  </body>
</html>
```

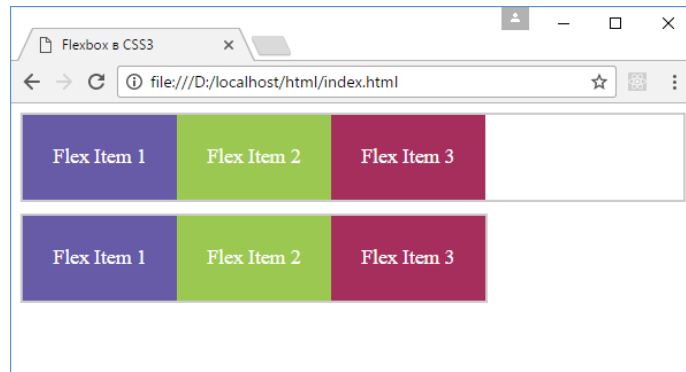
Для контейнера `flex-container` установлено свойство `display:flex`. В нем располагается три flex-элемента.



Если значение flex определяет контейнер как блочный элемент, то значение inline-flex определяет элемент как строчный (inline). Рассмотрим оба способа на примере:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Flexbox в CSS3</title>
    <style>
      .flex-container {
        display: flex;
        border: 2px solid #ccc;
      }
      .inline-flex-container {
        display: inline-flex;
        border: 2px solid #ccc;
        margin-top: 10px;
      }
      .flex-item {
        text-align: center;
        font-size: 1.1em;
        padding: 1.5em;
        color: white;
      }
      .color1 { background-color: #675BA7; }
      .color2 { background-color: #9BC850; }
      .color3 { background-color: #A62E5C; }
    </style>
  </head>
  <body>
    <div class="flex-container">
      <div class="flex-item color1">Flex Item 1</div>
      <div class="flex-item color2">Flex Item 2</div>
      <div class="flex-item color3">Flex Item 3</div>
    </div>

    <div class="inline-flex-container">
      <div class="flex-item color1">Flex Item 1</div>
      <div class="flex-item color2">Flex Item 2</div>
      <div class="flex-item color3">Flex Item 3</div>
    </div>
  </body>
</html>
```



В частности, в первом случае flex-контейнер растягивается по ширине страницы, а во втором случае занимает именно столько места, сколько необходимо для flex-элементов.

2.1.2 НАПРАВЛЕНИЕ FLEX-DIRECTION

Flex-элементы во flex-контейнере могут иметь определенное направление, а именно они могут располагаться в виде строк или в виде столбцов. Для управления направлением элементов CSS3 предоставляет свойство `flex-direction`. Оно определяет направление элементов и может принимать следующие значения:

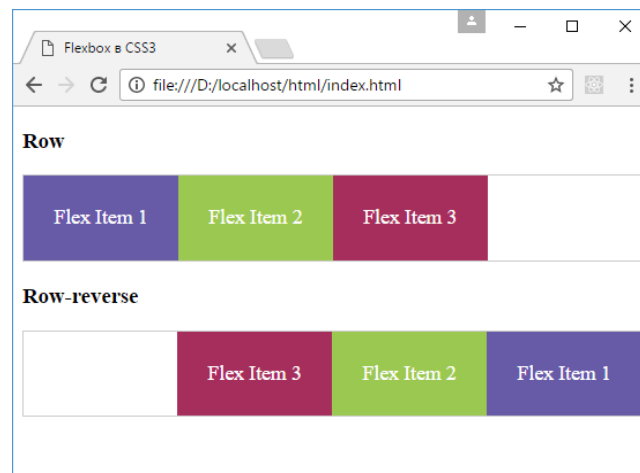
- **row**: значение по умолчанию, при котором элементы располагаются в виде строки слева направо
- **row-reverse**: элементы также располагаются в виде строки только в обратном порядке справа налево
- **column**: элементы располагаются в столбик сверху вниз
- **column-reverse**: элементы располагаются в столбик в обратном порядке снизу вверх

Например, расположение в виде строки:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Flexbox в CSS3</title>
    <style>
      .flex-container {
        display: flex;
        border: 1px solid #ccc;
      }
      .row {
        flex-direction: row;
      }
      .row-reverse {
        flex-direction: row-reverse;
      }
      .flex-item {
        text-align: center;
        font-size: 1.1em;
        padding: 1.5em;
        color: white;
      }
    </style>
  </head>
  <body>
    <div class="flex-container">
      <div class="row">
        <div class="flex-item">Flex Item 1</div>
        <div class="flex-item">Flex Item 2</div>
        <div class="flex-item">Flex Item 3</div>
      </div>
      <div class="row-reverse">
        <div class="flex-item">Flex Item 1</div>
        <div class="flex-item">Flex Item 2</div>
        <div class="flex-item">Flex Item 3</div>
      </div>
    </div>
  </body>
</html>
```

```
.color1 { background-color: #675BA7;}
.color2 { background-color: #9BC850;}
.color3 { background-color: #A62E5C;}
</style>
</head>
<body>
  <h3>Row</h3>
  <div class="flex-container row">
    <div class="flex-item color1">Flex Item 1</div>
    <div class="flex-item color2">Flex Item 2</div>
    <div class="flex-item color3">Flex Item 3</div>
  </div>

  <h3>Row-reverse</h3>
  <div class="flex-container row-reverse">
    <div class="flex-item color1">Flex Item 1</div>
    <div class="flex-item color2">Flex Item 2</div>
    <div class="flex-item color3">Flex Item 3</div>
  </div>
</body>
</html>
```



Аналогично работает расположение в виде столбца:

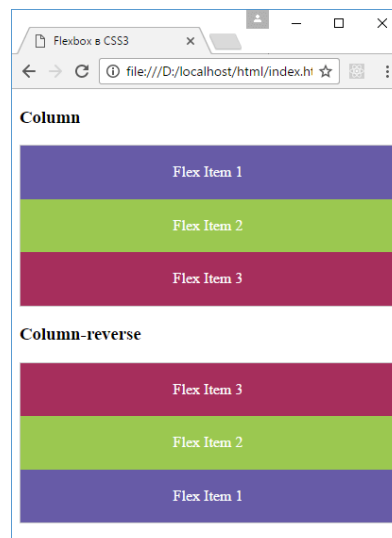
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Flexbox в CSS3</title>
    <style>
      .flex-container {
        display: flex;
        border: 1px solid #ccc;
      }
      .column {
        flex-direction: column;
      }
      .column-reverse {
        flex-direction: column-reverse;
      }
      .flex-item {
```

```

        text-align:center;
        font-size: 1em;
        padding: 1.2em;
        color: white;
    }
    .color1 {background-color: #675BA7;}
    .color2 {background-color: #9BC850;}
    .color3 {background-color: #A62E5C;}
</style>
</head>
<body>
    <h3>Column</h3>
    <div class="flex-container column">
        <div class="flex-item color1">Flex Item 1</div>
        <div class="flex-item color2">Flex Item 2</div>
        <div class="flex-item color3">Flex Item 3</div>
    </div>

    <h3>Column-reverse</h3>
    <div class="flex-container column-reverse">
        <div class="flex-item color1">Flex Item 1</div>
        <div class="flex-item color2">Flex Item 2</div>
        <div class="flex-item color3">Flex Item 3</div>
    </div>
</body>
</html>

```



2.1.3 FLEX-WRAP

Свойство **flex-wrap** определяет, будет ли flex-контейнер несколько рядов элементов (строк или столбцов) в случае если его размеры недостаточны, чтобы вместить в один ряд все элементы. Это свойство может принимать следующие значения:

- **nowrap**: значение по умолчанию, которое определяет flex-контейнер, где все элементы располагаются в одну строку (при расположении в виде строк) или один столбец (при расположении в столбик)

- **wrap**: если элементы не помещаются во flex-контейнер, то создает дополнительные ряды в контейнере для размещения элементов. При расположении в виде строки создаются дополнительные строки, а при расположении в виде столбца добавляются дополнительные столбцы

- **wrap-reverse**: то же самое, что и значение wrap, только элементы располагаются в обратном порядке

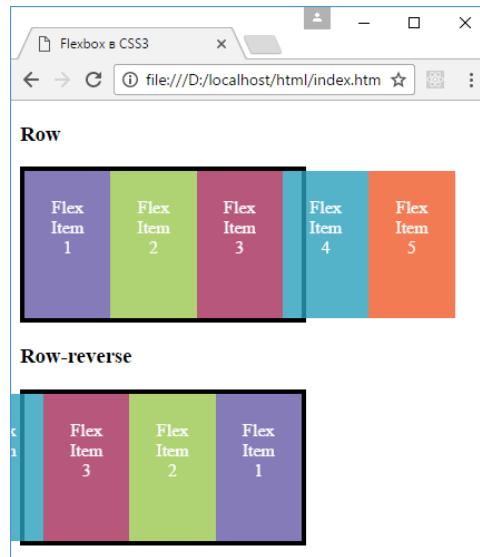
Например, возьмем значение по умолчанию nowrap:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Flexbox в CSS3</title>
    <style>
      .flex-container {
        display: flex;
        border: solid 0.25em #000;
        width: 60%;
        height: 8.25em;
        flex-wrap: nowrap;
      }
      .row {
        flex-direction: row;
      }
      .row-reverse {
        flex-direction: row-reverse;
      }
      .flex-item {
        text-align: center;
        font-size: 1em;
        padding: 1.5em;
        color: white;
        opacity: 0.8;
      }
      .color1 { background-color: #675BA7; }
      .color2 { background-color: #9BC850; }
      .color3 { background-color: #A62E5C; }
      .color4 { background-color: #2A9FBC; }
      .color5 { background-color: #F15B2A; }
    </style>
  </head>
  <body>
    <h3>Row</h3>
    <div class="flex-container row">
      <div class="flex-item color1">Flex Item 1</div>
      <div class="flex-item color2">Flex Item 2</div>
      <div class="flex-item color3">Flex Item 3</div>
      <div class="flex-item color4">Flex Item 4</div>
      <div class="flex-item color5">Flex Item 5</div>
    </div>

    <h3>Row-reverse</h3>
```

```
<div class="flex-container row-reverse">
  <div class="flex-item color1">Flex Item 1</div>
  <div class="flex-item color2">Flex Item 2</div>
  <div class="flex-item color3">Flex Item 3</div>
  <div class="flex-item color4">Flex Item 4</div>
  <div class="flex-item color5">Flex Item 5</div>
</div>
</body>
</html>
```

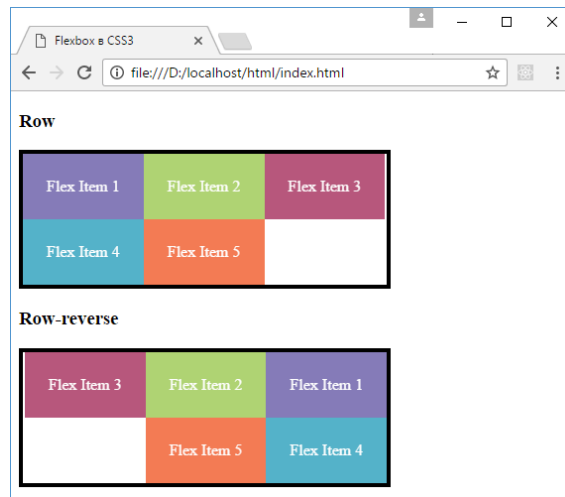
Здесь в каждом из flex-контейнеров по пять элементов, однако ширина контейнера может вмещать не все элементы, тогда они уходят за границу контейнера:



При установке значения `wrap` во flex-контейнере добавляются дополнительные ряды для помещения всех элементов в контейнере. Так, изменим значение свойства `flex-wrap` в контейнере:

```
.flex-container {
  display: flex;
  border: solid 0.25em #000;
  width: 60%;
  height: 8.25em;
  flex-wrap: wrap;}
```

В этом случае появится дополнительная строка:



При расположении в виде столбца контейнер будет создавать дополнительные столбцы:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Flexbox в CSS3</title>
    <style>
      .flex-container {
        display: flex;
        border: solid 0.25em #000;
        width: 60%;
        height: 8.25em;
        flex-wrap: wrap;
      }
      .column {
        flex-direction: column;
      }
      .column-reverse {
        flex-direction: column-reverse;
      }
      .flex-item {
        text-align: center;
        font-size: 1em;
        padding: 1.5em;
        color: white;
        opacity: 0.8;
      }
      .color1 { background-color: #675BA7; }
      .color2 { background-color: #9BC850; }
      .color3 { background-color: #A62E5C; }
      .color4 { background-color: #2A9FBC; }
      .color5 { background-color: #F15B2A; }
    </style>
  </head>
  <body>
    <h3>Column</h3>
    <div class="flex-container column">
```

```

<div class="flex-item color1">Flex Item 1</div>
<div class="flex-item color2">Flex Item 2</div>
<div class="flex-item color3">Flex Item 3</div>
<div class="flex-item color4">Flex Item 4</div>
<div class="flex-item color5">Flex Item 5</div>
</div>
<h3>Column-reverse</h3>
<div class="flex-container column-reverse">
  <div class="flex-item color1">Flex Item 1</div>
  <div class="flex-item color2">Flex Item 2</div>
  <div class="flex-item color3">Flex Item 3</div>
  <div class="flex-item color4">Flex Item 4</div>
  <div class="flex-item color5">Flex Item 5</div>
</div>
</body>
</html>

```



2.1.4 FLEX-FLOW. ПОРЯДОК ЭЛЕМЕНТОВ

Свойство **flex-flow** позволяет установить значения сразу для обоих свойств **flex-direction** и **flex-wrap**. Оно имеет следующий формальный синтаксис:

flex-flow: [flex-direction] [flex-wrap]

Причем второе свойство - **flex-wrap** можно в принципе опустить, тогда для него будет использоваться значение по умолчанию - **nowrap**.

Свойство order

Свойство **order** позволяет установить группу для flex-элемента, позволяя тем самым переопределить его позицию внутри flex-контейнера. В качестве значения свойство принимает числовой порядок группы. К одной группе может принадлежать несколько элементов.

Например, элементы в группе 0 располагаются перед элементами с группой 1, а элементы с группой 1 располагаются перед элементами с группой 2 и так далее.

```

<!DOCTYPE html>
<html>
  <head>

```

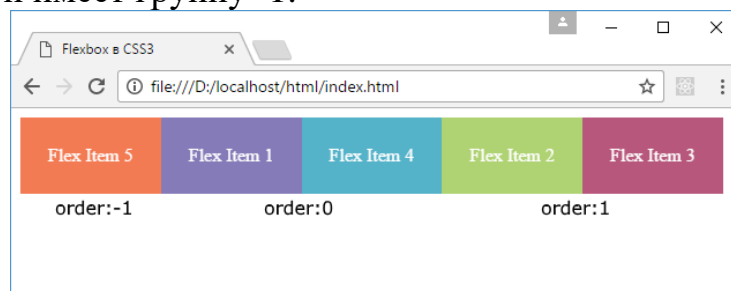
```

<meta charset="utf-8">
<title>Flexbox в CSS3</title>
<style>

.flex-container {
  display: flex;
  flex-flow: row wrap;
}
.flex-item {
  text-align:center;
  font-size: 1em;
  padding: 1.5em;
  color: white;
  opacity: 0.8;
}
.group1 {
  order:-1;
}
.group2 {
  order:1;
}
.color1 {background-color: #675BA7;}
.color2 {background-color: #9BC850;}
.color3 {background-color: #A62E5C;}
.color4 {background-color: #2A9FBC;}
.color5 {background-color: #F15B2A;}
</style>
</head>
<body>
<div class="flex-container">
  <div class="flex-item color1">Flex Item 1</div>
  <div class="flex-item color2 group2">Flex Item 2</div>
  <div class="flex-item color3 group2">Flex Item 3</div>
  <div class="flex-item color4">Flex Item 4</div>
  <div class="flex-item color5 group1">Flex Item 5</div>
</div>
</html>

```

В данном случае определены 3 группы. Первый отображается последний элемент, так как он имеет группу -1:



По умолчанию если у элементов явным образом не указано свойство order, то оно имеет значение 0. И последними в данном случае отображаются второй и третий элемент, так как у них свойство order равно 1.

2.1.5 ВЫРАВНИВАНИЕ ЭЛЕМЕНТОВ. JUSTIFY-CONTENT

Иногда мы можем сталкиваться с тем, что пространство flex-контейнеров по размеру отличается от пространства, необходимого для flex-элементов. Например:

- flex-элементы не используют все пространство flex-контейнера
- flex-элементам требуется большее пространство, чем доступно во flex-контейнере. В этом случае элементы выходят за пределы контейнера.

Для управления этими ситуациями мы можем применять свойство **justify-content**. Оно выравнивает элементы вдоль основной оси - main axis (при расположении в виде строки по горизонтали, при расположении в виде столбца - по вертикали) и принимает следующие значения:

- **flex-start**: значение по умолчанию, при котором первый элемент выравнивается по левому краю контейнера(при расположении в виде строки) или по верху (при расположении в виде столбца), за ним располагается второй элемент и так далее.
- **flex-end**: последний элемент выравнивается по правому краю (при расположении в виде строки) или по низу (при расположении в виде столбца) контейнера, за ним выравнивается предпоследний элемент и так далее
- **center**: элементы выравниваются по центру
- **space-between**: если в строке только один элемент или элементы выходят за границы flex-контейнера, то данное значение аналогично flex-start. В остальных случаях первый элемент выравнивается по левому краю (при расположении в виде строки) или по верху (при расположении в виде столбца), а последний элемент - по правому краю контейнера (при расположении в виде строки) или по низу (при расположении в виде столбца). Все оставшееся пространство между ними равным образом распределяется между остальными элементами
- **space-around**: если в строке только один элемент или элементы выходят за пределы контейнера, то его действие аналогично значению center. В ином случае элементы равным образом распределяют пространство между левым и правым краем контейнера, а расстояние между первым и последним элементом и границами контейнера составляет половину расстояния между элементами.

Выравнивание для расположения элементов в виде строки:

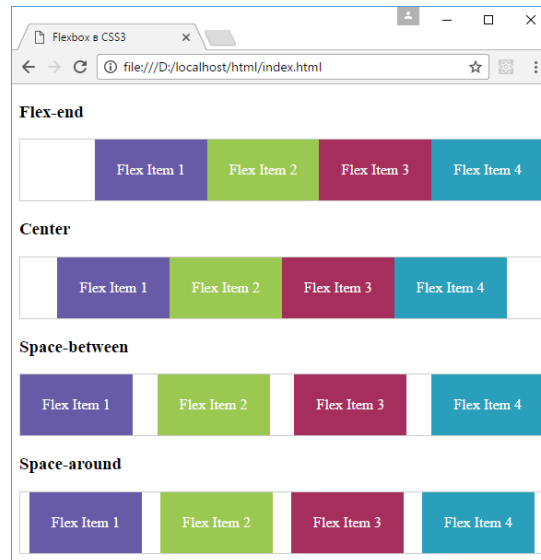
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Flexbox в CSS3</title>
    <style>

      .flex-container {
        display: flex;
        border: 1px #ccc solid;
      }
      .flex-end{
        justify-content: flex-end;
      }
      .center{
```

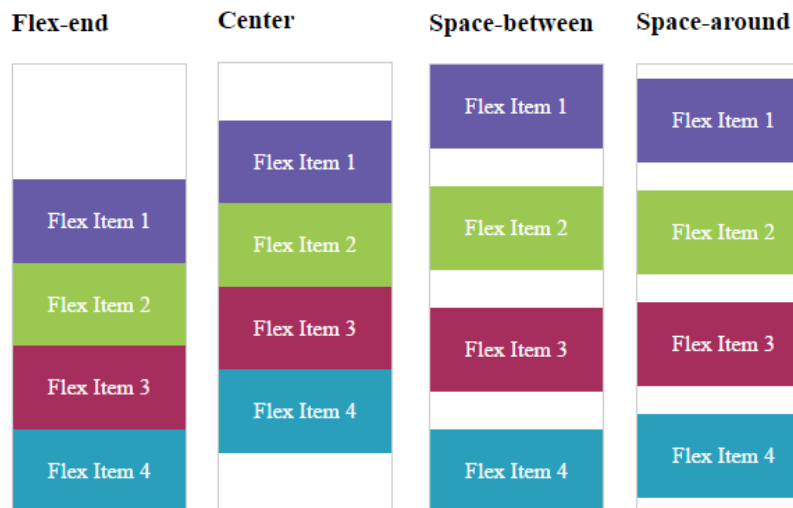
```

        justify-content: center;
    }
    .space-between{
        justify-content: space-between;
    }
    .space-around{
        justify-content: space-around;
    }
    .flex-item {
        text-align:center;
        font-size: 1em;
        padding: 1.5em;
        color: white;
    }
    .color1 {background-color: #675BA7;}
    .color2 {background-color: #9BC850;}
    .color3 {background-color: #A62E5C;}
    .color4 {background-color: #2A9FBC;}
    .color5 {background-color: #F15B2A;}
</style>
</head>
<body>
    <h3>Flex-end</h3>
    <div class="flex-container flex-end">
        <div class="flex-item color1">Flex Item 1</div>
        <div class="flex-item color2">Flex Item 2</div>
        <div class="flex-item color3">Flex Item 3</div>
        <div class="flex-item color4">Flex Item 4</div>
    </div>
    <h3>Center</h3>
    <div class="flex-container center">
        <div class="flex-item color1">Flex Item 1</div>
        <div class="flex-item color2">Flex Item 2</div>
        <div class="flex-item color3">Flex Item 3</div>
        <div class="flex-item color4">Flex Item 4</div>
    </div>
    <h3>Space-between</h3>
    <div class="flex-container space-between">
        <div class="flex-item color1">Flex Item 1</div>
        <div class="flex-item color2">Flex Item 2</div>
        <div class="flex-item color3">Flex Item 3</div>
        <div class="flex-item color4">Flex Item 4</div>
    </div>
    <h3>Space-around</h3>
    <div class="flex-container space-around">
        <div class="flex-item color1">Flex Item 1</div>
        <div class="flex-item color2">Flex Item 2</div>
        <div class="flex-item color3">Flex Item 3</div>
        <div class="flex-item color4">Flex Item 4</div>
    </div>
</body>
</html>

```

Выравнивание при расположении в виде столбцов:



2.1.6 ВЫРАВНИВАНИЕ ЭЛЕМЕНТОВ. ALIGN-ITEMS И ALIGN-SELF

Свойство align-items

Свойство **align-items** также выравнивает элементы, но уже по поперечной оси (cross axis) (при расположении в виде строки по вертикали, при расположении в виде столбца - по горизонтали). Это свойство может принимать следующие значения:

- **stretch**: значение по умолчанию, при котором flex-элементы растягиваются по всей высоте (при расположении в строку) или по всей ширине (при расположении в столбик) flex-контейнера
- **flex-start**: элементы выравниваются по верхнему краю (при расположении в строку) или по левому краю (при расположении в столбик) flex-контейнера
- **flex-end**: элементы выравниваются по нижнему краю (при расположении в строку) или по правому краю (при расположении в столбик) flex-контейнера
- **center**: элементы выравниваются по центру flex-контейнера

- **baseline:** элементы выравниваются в соответствии со своей базовой линией

Выравнивание при расположении в строку:

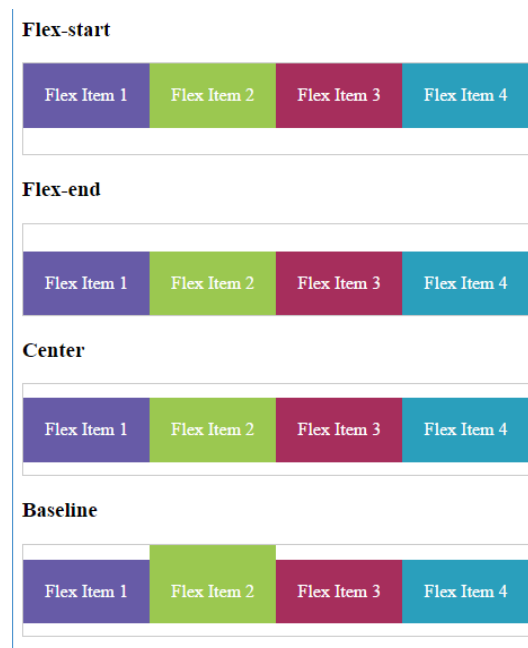
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Flexbox в CSS3</title>
    <style>

      .flex-container {
        display: flex;
        border: 1px #ccc solid;
        height: 5em;
      }
      .flex-start {
        align-items: flex-start;
      }
      .flex-end {
        align-items: flex-end;
      }
      .center {
        align-items: center;
      }
      .baseline {
        align-items: baseline;
      }
      .flex-item {
        text-align: center;
        font-size: 1em;
        padding: 1.2em;
        color: white;
      }
      .largest-item {
        padding-top: 2em;
      }
      .color1 { background-color: #675BA7; }
      .color2 { background-color: #9BC850; }
      .color3 { background-color: #A62E5C; }
      .color4 { background-color: #2A9FBC; }
    </style>
  </head>
  <body>
    <h3>Flex-start</h3>
    <div class="flex-container flex-start">
      <div class="flex-item color1">Flex Item 1</div>
      <div class="flex-item color2">Flex Item 2</div>
      <div class="flex-item color3">Flex Item 3</div>
      <div class="flex-item color4">Flex Item 4</div>
    </div>
```

```

<h3>Flex-end</h3>
<div class="flex-container flex-end">
  <div class="flex-item color1">Flex Item 1</div>
  <div class="flex-item color2">Flex Item 2</div>
  <div class="flex-item color3">Flex Item 3</div>
  <div class="flex-item color4">Flex Item 4</div>
</div>
<h3>Center</h3>
<div class="flex-container center">
  <div class="flex-item color1">Flex Item 1</div>
  <div class="flex-item color2">Flex Item 2</div>
  <div class="flex-item color3">Flex Item 3</div>
  <div class="flex-item color4">Flex Item 4</div>
</div>
<h3>Baseline</h3>
<div class="flex-container baseline">
  <div class="flex-item color1">Flex Item 1</div>
  <div class="flex-item color2 largest-item">Flex Item 2</div>
  <div class="flex-item color3">Flex Item 3</div>
  <div class="flex-item color4">Flex Item 4</div>
</div>
</html>

```

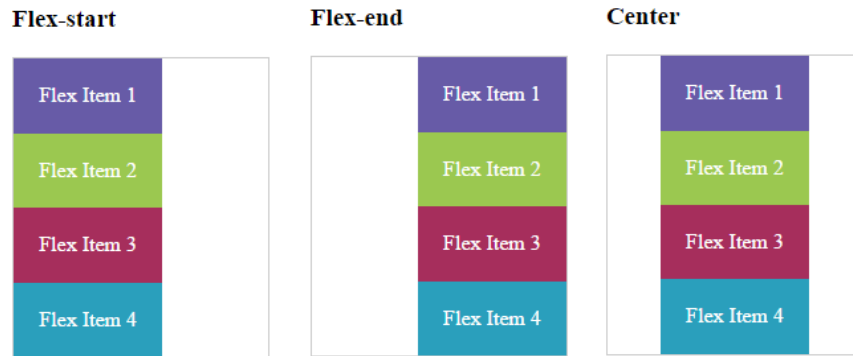


Аналогично свойство работает при расположении в столбик. Например, изменим стили flex-контейнера следующим образом:

```

.flex-container {
  display: flex;
  border: 1px #ccc solid;
  flex-direction: column;
  width: 12em;
}

```



Свойство align-self

Свойство **align-self** позволяет переопределить значение свойства align-items для одного элемента. Оно может принимать все те же значения плюс значение "auto":

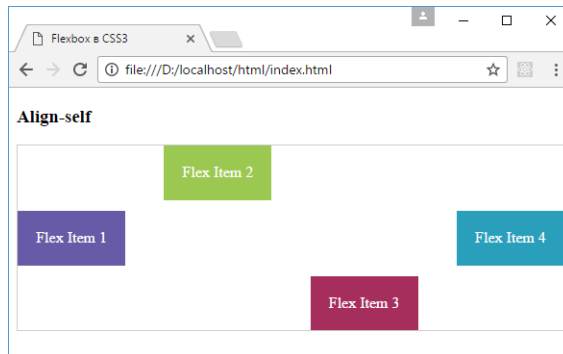
- **auto**: значение по умолчанию, при котором элемент получает значение от свойства align-items, которое определено в flex-контейнере. Если в контейнере такой стиль не определен, то применяется значение stretch.

- **stretch**
- **flex-start**
- **flex-end**
- **center**
- **baseline**

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Flexbox в CSS3</title>
    <style>
      .flex-container {
        display: flex;
        border: 1px #ccc solid;
        justify-content: space-between;
        align-items: stretch;
        height: 12em;
      }
      .flex-item {
        text-align: center;
        font-size: 1em;
        padding: 1.2em;
        color: white;
      }
      .item1 { background-color: #675BA7; align-self: center; }
      .item2 { background-color: #9BC850; align-self: flex-start; }
      .item3 { background-color: #A62E5C; align-self: flex-end; }
      .item4 { background-color: #2A9FBC; align-self: center; }
    </style>
  </head>
  <body>
    <h3>Align-self</h3>
```

```
<div class="flex-container">
  <div class="flex-item item1">Flex Item 1</div>
  <div class="flex-item item2">Flex Item 2</div>
  <div class="flex-item item3">Flex Item 3</div>
  <div class="flex-item item4">Flex Item 4</div>
</div>
</html>
```

Здесь для flex-контейнера задано растяжение по высоте с помощью значения align-items: stretch;. Однако каждый из элементов переопределяет это поведение:



2.1.7 ВЫРАВНИВАНИЕ СТРОК И СТОЛБЦОВ. ALIGN-CONTENT

Свойство **align-content** управляет выравниванием рядов (строк и столбцов) во flex-контейнере и поэтому применяется, если свойство flex-wrap имеет значение wrap или wrap-reverse. Свойство align-content может иметь следующие значения:

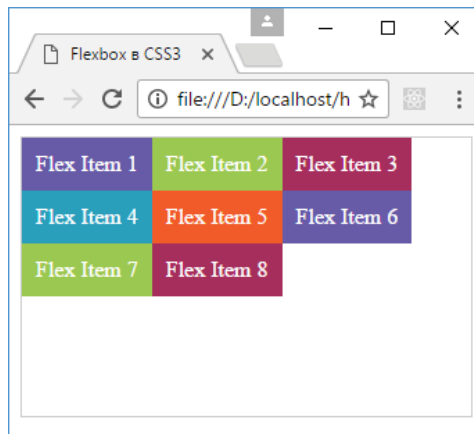
- **stretch**: значение по умолчанию, при котором строки (столбцы) растягиваются, занимая все свободное место
- **flex-start**: строки (столбцы) выравниваются по началу контейнера (для строк - это верхний край, для столбцов - это левый край контейнера)
- **flex-end**: строки (столбцы) выравниваются по концу контейнера (строки - по нижнему краю, столбцы - по правому краю)
- **center**: строки (столбцы) позиционируются по центру контейнера
- **space-between**: строки (столбцы) равномерно распределяются по контейнеру, а между ними образуются одинаковые отступы. Если же имеющегося в контейнере места недостаточно, то действует аналогично значению flex-start
- **space-around**: строки (столбцы) равным образом распределяют пространство контейнера, а расстояние между первой и последней строкой (столбцом) и границами контейнера составляет половину расстояния между соседними строками (столбцами).

Стоит учитывать, что это свойство имеет смысл, если в контейнере две и больше строки (столбца).

Например, расположение строк в начале контейнера:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Flexbox в CSS3</title>
    <style>
```

```
.flex-container {
  display: flex;
  border: 1px #ccc solid;
  flex-wrap: wrap;
  height: 200px;
  align-content: flex-start;
}
.flex-item {
  text-align: center;
  font-size: 16px;
  padding: 10px;
  color: white;
}
.item1 { background-color: #675BA7; }
.item2 { background-color: #9BC850; }
.item3 { background-color: #A62E5C; }
.item4 { background-color: #2A9FBC; }
.item5 { background-color: #F15B2A; }
</style>
</head>
<body>
  <div class="flex-container">
    <div class="flex-item item1">Flex Item 1</div>
    <div class="flex-item item2">Flex Item 2</div>
    <div class="flex-item item3">Flex Item 3</div>
    <div class="flex-item item4">Flex Item 4</div>
    <div class="flex-item item5">Flex Item 5</div>
    <div class="flex-item item1">Flex Item 6</div>
    <div class="flex-item item2">Flex Item 7</div>
    <div class="flex-item item3">Flex Item 8</div>
  </div>
</html>
```

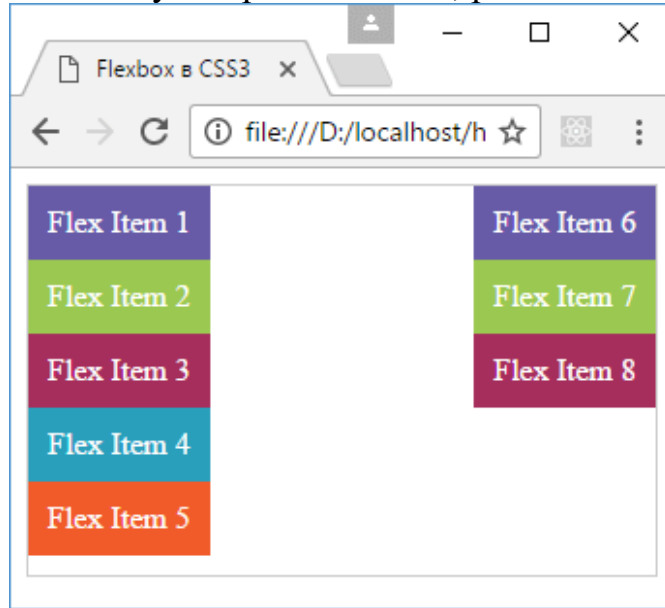


Изменим стиль контейнера:

```
.flex-container {
  display: flex;
  border: 1px #ccc solid;
  flex-wrap: wrap;
  height: 200px;
  align-content: space-between;
```

```
flex-direction: column;
}
```

И в этом случае мы получим ряд столбцов, разделенных отступами:



2.1.8 УПРАВЛЕНИЕ ЭЛЕМЕНТАМИ. FLEX-BASIS, FLEX-SHRINK И FLEX-GROW

Кроме свойств, устанавливающих выравнивание элементов относительно границ flex-контейнера, есть еще три свойства, которые позволяют управлять элементами:

- **flex-basis**: определяет начальный размер flex-элемента
- **flex-shrink**: определяет, как flex-элемент будет уменьшаться относительно других flex-элементов во flex-контейнере
- **flex-grow**: определяет, как flex-элемент будет увеличиваться относительно других flex-элементов во flex-контейнере

flex-basis

Flex-контейнер может увеличиваться или уменьшаться вдоль своей центральной оси, например, при изменении размеров браузера, если контейнер имеет нефиксированные размеры. И вместе с контейнером также могут увеличиваться и уменьшаться его flex-элементы. Свойство **flex-basis** определяет начальный размер flex-элемента до того, как он начнет изменять размер, подстраиваясь под размеры flex-контейнера.

Это свойство может принимать следующие значения:

- **auto**: начальный размер flex-элемента устанавливается автоматически
- **content**: размер flex-элемента определяется по его содержимому, в то же время это значение поддерживается не всеми современными браузерами, поэтому его пока стоит избегать
- **числовое значение**: мы можем установить конкретное числовое значение для размеров элемента

flex-shrink

Если flex-контейнер имеет недостаточно места для размещения элемента, то дальнейшее поведение этого элемента мы можем определить с помощью свойства **flex-shrink**. Оно указывает, как элемент будет усекаться относительно других элементов.

В качестве значения свойство принимает число. По умолчанию его значение 1.

flex-grow

Свойство **flex-grow** управляет расширением элементов, если во flex-контейнере есть дополнительное место. Данное свойство во многом похоже на свойство flex-shrink за тем исключением, что работает в сторону увеличения элементов.

В качестве значения свойство flex-grow принимает положительное число, которое указывает, во сколько раз элемент будет увеличиваться относительно других элементов при увеличении размеров flex-контейнера. По умолчанию свойство flex-grow равно 0.

Свойство flex

Свойство **flex** является объединением свойств flex-basis, flex-shrink и flex-grow и имеет следующий формальный синтаксис:

flex: [flex-grow] [flex-shrink] [flex-basis]

По умолчанию свойство flex имеет значение 0 1 auto.

Кроме конкретных значений для каждого из подсвойств мы можем задать для свойства flex одно из трех общих значений:

- flex: none: эквивалентно значению 0 0 auto, при котором flex-элемент не растягивается и не усекается при увеличении и уменьшении контейнера
- flex: auto: эквивалентно значению 1 1 auto
- flex: initial: эквивалентно значению 0 1 auto

3.1. GRID LAYOUT

3.1.1 ЧТО ТАКОЕ GRID LAYOUT. GRID CONTAINER

Grid Layout представляет специальный модуль CSS3, который позволяет позиционировать элементы в виде сетки или таблицы. Как и Flexbox, Grid Layout представляет гибкий подход к компоновке элементов, только если flexbox размещает вложенные элементы в одном направлении - по горизонтали в виде столбиков или по вертикали в виде строк, то Grid позиционирует элементы сразу в двух направлениях - в виде строк и столбцов, образуя тем самым таблицу.

Поддержка браузерами

При использовании Grid Layout следует учитывать, что только относительно недавно производители браузеров стали внедрять поддержку этого модуля в свои браузеры. Ниже приводится для браузеров список версий, начиная с которых была внедрена полноценная поддержка Grid Layout:

- Google Chrome - с версии 57
- Mozilla Firefox - с версии 52
- Opera - с версии 44
- Safari - с версии 10.1
- iOS Safari - с версии 10.3

Как можно заметить, большинство этих версий браузеров вышли в начале 2017 года. То есть на более старые версии этих браузеров рассчитывать не приходится.

Кроме того, IE (начиная с версии 10) и Microsoft Edge имеет лишь частичную поддержку модуля. А Android Browser, Opera Mini, UC Browser вовсе ее не имеют.

Создание grid-контейнера

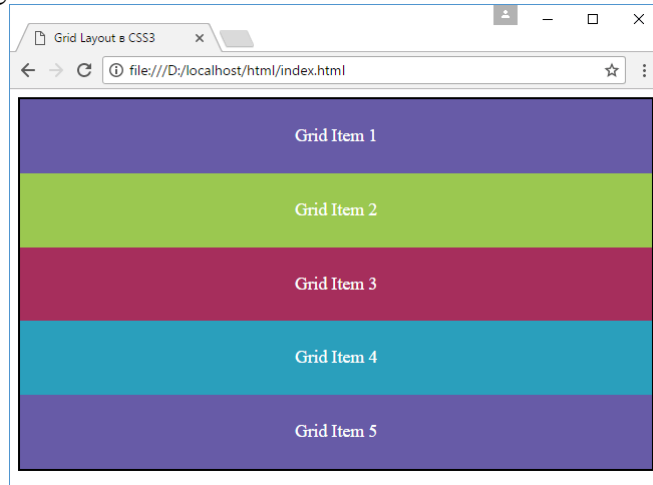
Основой для определения компоновки Grid Layout является grid container, внутри которого размещаются элементы. Для создания grid-контейнера необходимо присвоить его стилю свойству **display** одно из двух значений: **grid** или **inline-grid**.

Создадим простейшую веб-страницу, которая применяет Grid Layout:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width" />
    <title>Grid Layout в CSS3</title>
    <style>
      .grid-container {
        border: solid 2px #000;
        display: grid;
      }
      .grid-item {
        text-align:center;
        font-size: 1.1em;
        padding: 1.5em;
        color: white;
      }
      .color1 {background-color: #675BA7;}
      .color2 {background-color: #9BC850;}
      .color3 {background-color: #A62E5C;}
      .color4 {background-color: #2A9FBC;}
    </style>
  </head>
  <body>
    <div class="grid-container">
      <div class="grid-item color1">Grid Item 1</div>
      <div class="grid-item color2">Grid Item 2</div>
      <div class="grid-item color3">Grid Item 3</div>
      <div class="grid-item color4">Grid Item 4</div>
      <div class="grid-item color1">Grid Item 5</div>
    </div>
```

```
</body>
</html>
```

Для контейнера grid-container установлено свойство display:grid. В нем располагается пять grid-элементов.

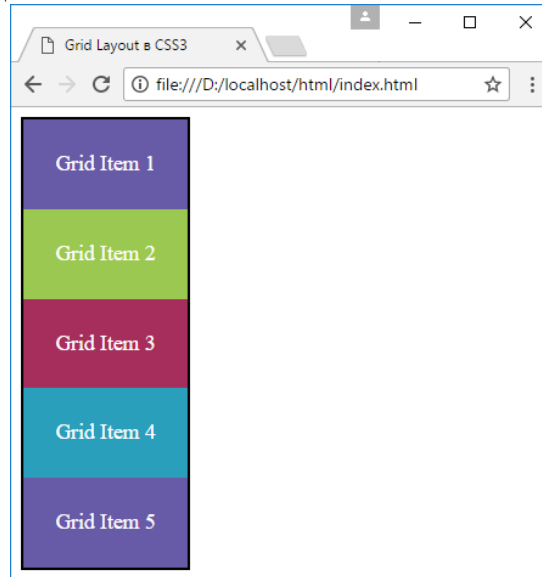


Если значение grid определяет контейнер как блочный элемент, то значение inline-grid определяет элемент как строчный (inline):

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width" />
    <title>Grid Layout в CSS3</title>
    <style>
      .grid-container {
        border: solid 2px #000;
        display: inline-grid;
      }
      .grid-item {
        box-sizing: border-box;
        text-align:center;
        font-size: 1.1em;
        padding: 1.5em;
        color: white;
      }
      .color1 {background-color: #675BA7;}
      .color2 {background-color: #9BC850;}
      .color3 {background-color: #A62E5C;}
      .color4 {background-color: #2A9FBC;}
    </style>
  </head>
  <body>
    <div class="grid-container">
      <div class="grid-item color1">Grid Item 1</div>
      <div class="grid-item color2">Grid Item 2</div>
      <div class="grid-item color3">Grid Item 3</div>
      <div class="grid-item color4">Grid Item 4</div>
      <div class="grid-item color1">Grid Item 5</div>
    </div>
```

```
</body>
</html>
```

В этом случае весь грид занимает только то пространство, которое необходимо для размещения его элементов.



3.1.2 СТРОКИ И СТОЛБЦЫ

Грид образует сетку из строк и столбцов, на пересечении которых образуются ячейки. И для установки строк и столбцов в Grid Layout использовать следующие свойства CSS3:

- **grid-template-columns**: настраивает столбцы
- **grid-template-rows**: настраивает строки

Столбцы

Для определения столбцов используем у grid-контейнера стилевое свойство **grid-template-columns**. В качестве значения свойству **grid-template-columns** передается ширина столбцов. Сколько мы хотим иметь в гриде столбцов, столько и нужно передать значений этому свойству.

Строки

Настройка строк во многом аналогичная настройке столбцов. Для этого у grid-контейнера необходимо установить свойство **grid-template-rows**, которое задает количество и размеры строк

3.1.3 ФУНКЦИЯ REPEAT И СВОЙСТВО GRID

Если у нас столбцов и(или) строк много и они имеют одинаковые размеры, то есть смысл использовать специальную функцию **repeat()**, которая позволит настроить строки и столбцы. Так, в примере выше повторяется определение одинаковых строк и столбцов в grid-контейнере:

```
grid-template-columns: 8em 8em 8em;
grid-template-rows: 5em 5em 5em 5em;
```

Здесь мы видим, что происходит повторение одних и тех же размеров - 8em и 5em для установки ширины столбцов и высоты строк. Поэтому перепишем стили, применив функцию repeat:

```
.grid-container {
  border: solid 2px #000;
  display: grid;
  grid-template-columns: repeat(3, 8em);
  grid-template-rows: repeat(4, 5em);
}
```

Первый параметр функции repeat представляет число повторений, а второй - определение строк или столбцов. Например, свойство grid-template-columns: repeat(3, 8em); говорит, что необходимо определить 3 столбца шириной в 8em.

Соответственно выражение grid-template-rows: repeat(4, 5em) определяет 4 строки высотой по 5em.

Свойство grid

Свойство **grid** объединяет свойства grid-template-rows и grid-template-columns и разом позволяет задать настройки для строк и столбцов в следующем формате:

```
grid: grid-template-rows / grid-template-columns;
```

3.1.4 РАЗМЕРЫ СТРОК И СТОЛБЦОВ

Фиксированные размеры

В примерах, которые были рассмотрены в предыдущих статьях, ширина столбцов и длина строк устанавливались на основании фиксированных значений, которые передаются свойствам **grid-template-columns** и **grid-template-rows**. Для определения размеров мы можем использовать самые различные единицы измерения, которые нам доступны в CSS (px, em, rem, pt, %), например:

```
.grid-container {
  border: solid 2px #000;
  display: grid;
  grid-template-columns: repeat(3, 200px);
  grid-template-rows: repeat(3, 4.5em);
}
```

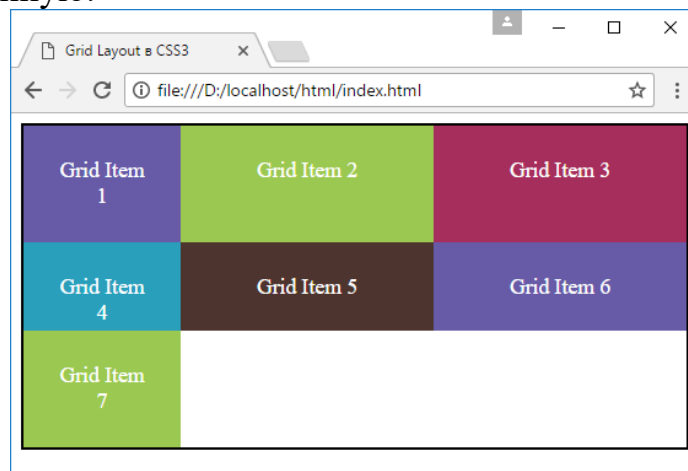
Автоматические размеры

Кроме точных размеров можно задавать автоматические размеры с помощью слова **auto**. В этом случае ширина столбцов и высота строк вычисляются исходя из размеров содержимого:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width" />
    <title>Grid Layout в CSS3</title>
    <style>
      .grid-container {
```

```
border: solid 2px #000;
display: grid;
grid-template-columns: 8em auto auto;
grid-template-rows: auto 4.5em auto;
}
.grid-item {
text-align:center;
font-size: 1.1em;
padding: 1.5em;
color: white;
}
.color1 {background-color: #675BA7;}
.color2 {background-color: #9BC850;}
.color3 {background-color: #A62E5C;}
.color4 {background-color: #2A9FBC;}
.color5 {background-color: #4e342e;}
</style>
</head>
<body>
<div class="grid-container">
<div class="grid-item color1">Grid Item 1</div>
<div class="grid-item color2">Grid Item 2</div>
<div class="grid-item color3">Grid Item 3</div>
<div class="grid-item color4">Grid Item 4</div>
<div class="grid-item color5">Grid Item 5</div>
<div class="grid-item color1">Grid Item 6</div>
<div class="grid-item color2">Grid Item 7</div>
</div>
</body>
</html>
```

Здесь задано три строки и три столбца. Первый столбец имеет фиксированную ширину в 8em, а второй и третий столбцы - автоматическую ширину. И также первая и третья строки имеют автоматическую высоту, а вторая строка - фиксированную.



Пропорциональные размеры

Для установки пропорциональных размеров применяется специальная единица измерения **fr**. Она представляет собой часть пространства (fraction),

которое отводится для данного столбца или строки. Значение `fr` еще называют flex-фактором (flex factor).

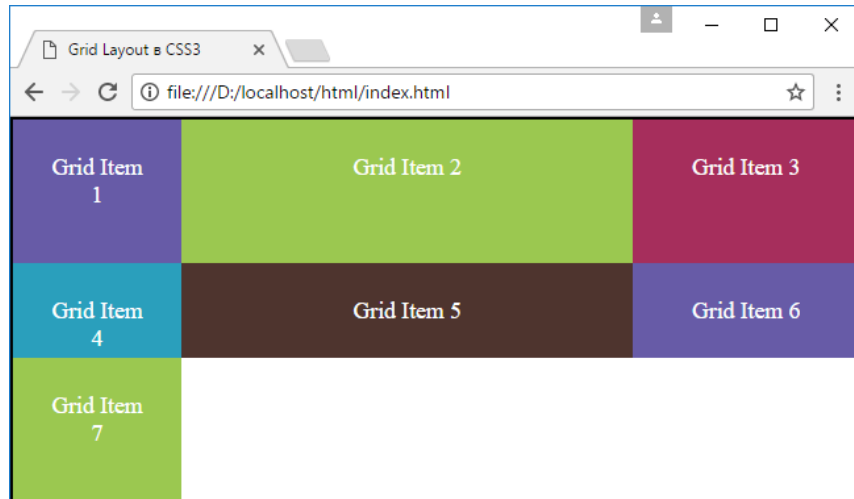
Вычисление пропорциональных размеров производится по формуле:

$$\text{flex-фактор} * \text{доступное_пространство} / \text{сумма всех flex-факторов}$$

При этом под доступным пространством понимается все пространство `grid`-контейнера за исключением фиксированных значений строк и столбцов.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width" />
    <title>Grid Layout в CSS3</title>
    <style>
      *{
        box-sizing: border-box;
      }
      html, body{
        margin:0;
        padding:0;
      }
      .grid-container {
        height: 100vh;
        border: solid 2px #000;
        display: grid;
        grid-template-columns: 8em 2fr 1fr;
        grid-template-rows: 1fr 4.5em 1fr;
      }
      .grid-item {
        text-align:center;
        font-size: 1.1em;
        padding: 1.5em;
        color: white;
      }
      .color1 {background-color: #675BA7;}
      .color2 {background-color: #9BC850;}
      .color3 {background-color: #A62E5C;}
      .color4 {background-color: #2A9FBC;}
      .color5 {background-color: #4e342e;}
    </style>
  </head>
  <body>
    <div class="grid-container">
      <div class="grid-item color1">Grid Item 1</div>
      <div class="grid-item color2">Grid Item 2</div>
      <div class="grid-item color3">Grid Item 3</div>
      <div class="grid-item color4">Grid Item 4</div>
      <div class="grid-item color5">Grid Item 5</div>
      <div class="grid-item color1">Grid Item 6</div>
      <div class="grid-item color2">Grid Item 7</div>
    </div>
  </body>
```


</html>



В данном случае имеются три столбца с шириной 2fr, 8em, 1fr. Поэтому ширина второго столбца будет вычисляться по формуле:

$$2 * (\text{ширина_грида} - 8\text{em}) / (2 + 1)$$

Ширина третьего столбца будет вычисляться по формуле:

$$1 * (\text{ширина_грида} - 8\text{em}) / (2 + 1)$$

И если первый столбец фиксированный с шириной 8em, то ширина второго и третьего столбца будут зависеть от ширины контейнера и будут автоматически масштабироваться при ее изменении.

В отношении строк все аналогично.

3.1.5 ОТСТУПЫ МЕЖДУ СТОЛБЦАМИ И СТРОКАМИ

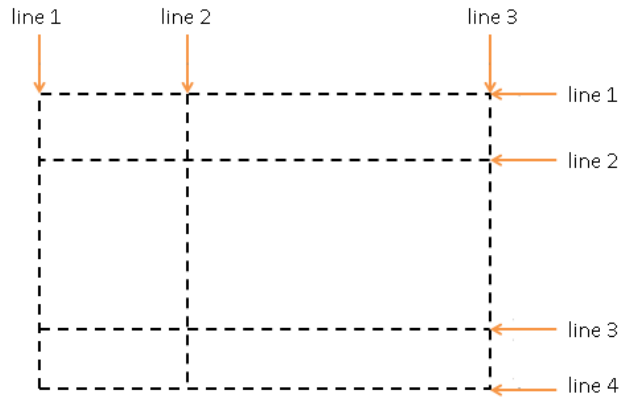
Для создания отступов между столбцами и строками применяются свойства **grid-column-gap** и **grid-row-gap** соответственно.

Если значения свойств **grid-column-gap** и **grid-row-gap** совпадают, то вместо них можно определить одно свойство **gap** (ранее назвалось **grid-gap**), которое установит оба отступа:

```
.grid-container {
  height: 100vh;
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-template-rows: repeat(3, 1fr);
  gap: 10px;
}
```

3.1.6 ПОЗИЦИОНИРОВАНИЕ ЭЛЕМЕНТОВ

Грид представляет собой набор ячеек, которые образуются на пересечении столбцов и строк. Но сами строки и столбцы образуются с помощью grid-линий, которые рассекают грид по вертикали и горизонтали:



И по умолчанию каждый элемент в гриде позиционируется в одну ячейку по порядку. Но мы можем более точно настроить расположение элемента в гриде с помощью ряда свойств:

- **grid-row-start**: задает начальную горизонтальную grid-линию, с которой начинается элемент
- **grid-row-end**: указывает, до какой горизонтальной grid-линии надо растягивать элемент
- **grid-column-start**: задает начальную вертикальную grid-линию, от которой начинается элемент
- **grid-column-end**: указывает, до какой вертикальной grid-линии нужно растягивать элемент.

4.1 ТРАНСФОРМАЦИИ, ПЕРЕХОДЫ И АНИМАЦИИ

4.1.1 ТРАНСФОРМАЦИИ

Одним из нововведений CSS3 по сравнению с предыдущей версией является встроенная возможность трансформаций элемента. К трансформациям относятся такие действия, как вращение элемента, его масштабирование, наклон или перемещение по вертикали или горизонтали. Для создания трансформаций в CSS3 применяется свойство **transform**.

Вращение

Для поворота элемента свойство transform использует функцию **rotate**:

transform: rotate(угол_поворота deg);

После слова **rotate** в скобках идет величина угла поворота в градусах. Например, повернем блок на 30 градусов:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Трансформации в CSS3</title>
    <style>
      div{
        background-color: #ccc;
        width: 120px;
        height: 120px;
        margin: 5px;
```

```
padding: 40px 10px;
box-sizing: border-box;
border: 1px solid #333;
display: inline-block;
}
.rotated{
transform: rotate(30deg);
}
</style>
</head>
<body>
<div></div>
<div class="rotated">rotate(30deg)</div>
<div></div>
</body>
</html>
```



При этом можно отметить, что при повороте вращаемый элемент может накладываться на соседние элементы, так как сначала происходит установка положения элементов и только затем поворот.

Угол поворота может представлять как положительное, так и отрицательное значение. В случае отрицательного значения поворот производится в противоположную сторону.

Масштабирование

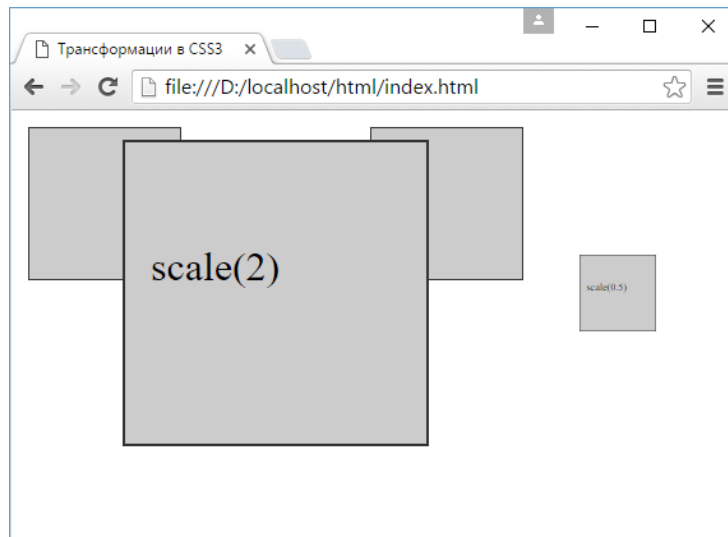
Применение масштабирования имеет следующую форму:

`transform: scale(величина_масштабирования);`

Используем масштабирование:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Трансформации в CSS3</title>
<style>
div{
background-color: #ccc;
width: 120px;
height: 120px;
margin:5px;
padding: 40px 10px;
box-sizing: border-box;
```

```
border: 1px solid #333;
display: inline-block;
}
.halfScale{
transform: scale(0.5);
}
.doubleScale{
transform: scale(2);
}
</style>
</head>
<body>
<div></div>
<div class="doubleScale">scale(2)</div>
<div></div>
<div class="halfScale">scale(0.5)</div>
</body>
</html>
```



Значение больше 1 приводит к растяжению по вертикали и горизонтали, а меньше 1 - к сжатию. То есть значение 0.5 приводит к уменьшению в два раза, а значение 1.5 - к увеличению в полтора раза.

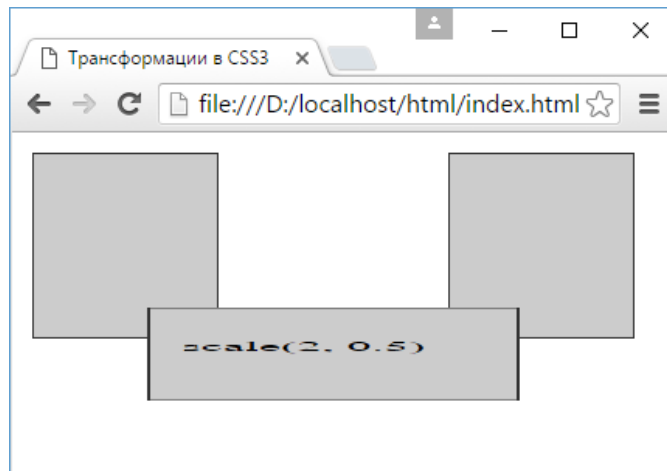
Можно также задать величины масштабирования отдельно для вертикали и горизонтали:

```
<!DOCTYPE html>  
<html>  
  <head>  
    <meta charset="utf-8">  
    <title>Трансформации в CSS3</title>  
    <style>  
      div{  
        background-color: #ccc;  
        width: 120px;  
        height: 120px;  
        margin: 5px;  
        padding: 40px 10px;  
        box-sizing: border-box;
```

```

        border: 1px solid #333;
        display: inline-block;
    }
    .scale1{
        transform: scale(2, 0.5);
    }
</style>
</head>
<body>
    <div></div>
    <div class="scale1">scale(2, 0.5)</div>
    <div></div>
</body>
</html>

```



В данном случае по горизонтали будет идти масштабирование в 2 раза, а по вертикали - в 0.5 раз.

Также можно по отдельности задать параметры масштабирования: функция **scaleX()** задает изменение по горизонтали, а **scaleY()** - по вертикали. Например:

```

.scale1{
    transform: scaleX(2);
}

```

Используя отрицательные значения, можно создать эффект зеркального отражения:

```

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Трансформации в CSS3</title>
    <style>
        div{
            background-color: #ccc;
            width: 120px;
            height: 120px;
            margin: 5px;
            padding: 40px 10px;
            box-sizing: border-box;
            border: 1px solid #333;

```

```

        display: inline-block;
    }
    .scale1{
        transform: scaleX(-1);
    }
</style>
</head>
<body>
    <div></div>
    <div class="scale1">scaleX(-1)</div>
    <div></div>
</body>
</html>

```



Перемещение

Для перемещения элемента используется функция **translate**:

```
transform: translate(offset_X, offset_Y);
```

Значение offset_X указывает, на сколько элемент смещается по горизонтали, а offset_Y - по вертикали.

К примеру, сместим блок на 30 пикселей вниз и на 50 пикселей вправо:

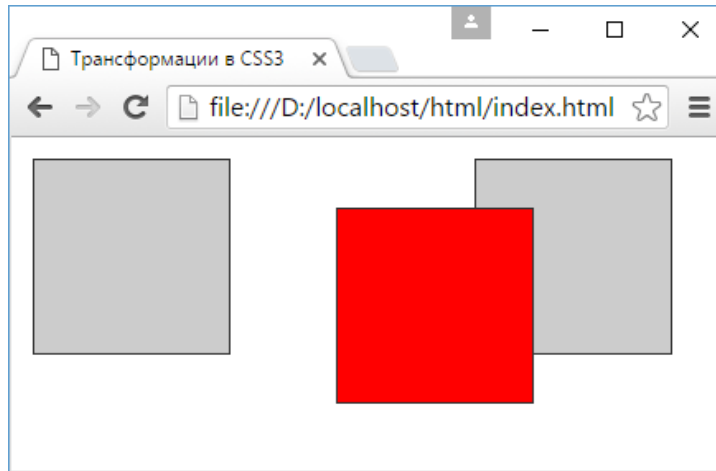
```

<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Трансформации в CSS3</title>
<style>
div{
    background-color: #ccc;
    width: 120px;
    height: 120px;
    margin: 5px;
    padding: 40px 10px;

    box-sizing: border-box;
    border: 1px solid #333;
    display: inline-block;
}

```

```
.translated{
    transform: translate(50px, 30px);
    background-color:red;
}
</style>
</head>
<body>
    <div></div>
    <div class="translated"></div>
    <div></div>
</body>
</html>
```



В качестве единиц измерения смещения можно применять не только пиксели, но и любые другие единицы измерения длины в CSS - em, % и тд.

С помощью дополнительных функций можно отдельно применять смещения к горизонтали или вертикали: **translateX()** (перемещение по горизонтали) и **translateY()** (перемещение по вертикали). Например:

```
transform: translateX(30px);
```

Кроме положительных значений также можно использовать и отрицательные - они перемещают элемент в противоположную сторону:

```
transform: translateY(-2.5em);
```

Наклон

Для наклона элемента применяется функция **skew()**:

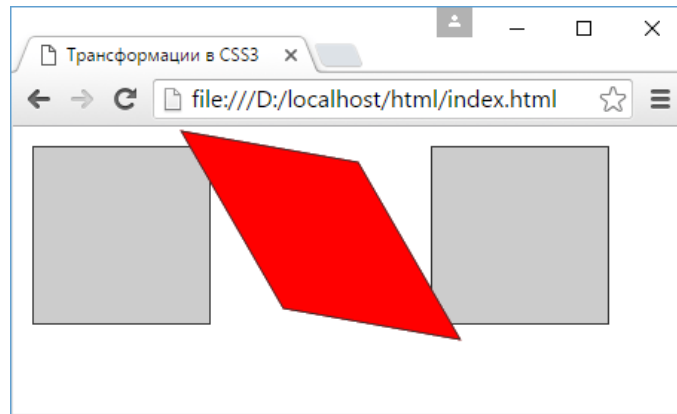
```
transform: skew(X, Y);
```

Первый параметр указывает, на сколько градусов наклонять элемент по оси X, а второй - значение наклона по оси Y.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Трансформации в CSS3</title>
    <style>
        div{
            background-color: #ccc;
            width: 120px;
            height: 120px;
```

```
margin:5px;
padding: 40px 10px;

box-sizing: border-box;
border: 1px solid #333;
display: inline-block;
}
.skewed{
  transform: skew(30deg, 10deg);
  background-color:red;
}
</style>
</head>
<body>
  <div></div>
  <div class="skewed"></div>
  <div></div>
</body>
</html>
```



Для создания наклона только по одной оси для другой оси надо использовать значение 0. Например, наклон на 45 градусов по оси X:

```
transform: skew(45deg, 0);
```

Или наклон на 45 градусов только по оси Y:

```
transform: skew(0,45deg);
```

Для создания наклона отдельно по оси X и по оси Y в CSS есть специальные функции: **skewX()** и **skewY()** соответственно.

```
transform: skewX(45deg);
```

Также можно передавать отрицательные значения. Тогда наклон будет осуществляться в противоположную сторону:

```
transform: skewX(-30deg);
```

Комбинирование преобразований

Если нам надо применить к элементу сразу несколько преобразований, скажем, вращение и перемещение, то мы можем их комбинировать. Например, применение всех четырех преобразований:

```
transform: translate(50px, 100px) skew(30deg, 10deg) scale(1.5)
rotate(90deg);
```

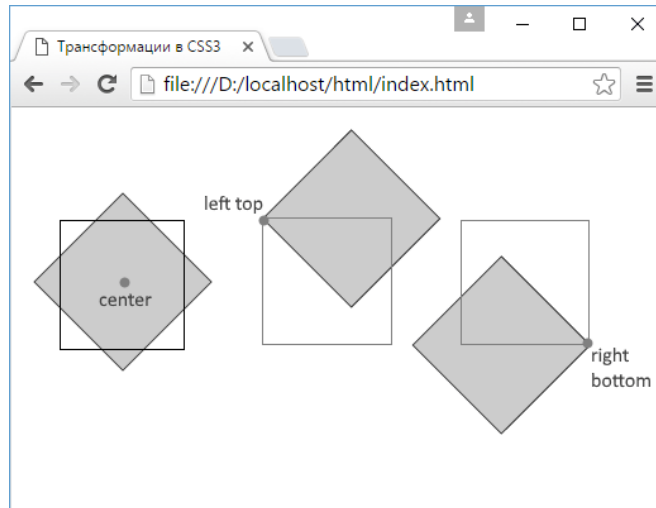

Браузер применяет все эти функции в порядке их следования. То есть в данном случае сначала к элементу применяется перемещение, потом наклон, потом масштабирование и в конце вращение.

Исходная точка трансформации

По умолчанию при применении трансформаций браузер в качестве точки начала преобразования использует центр элемента. Но с помощью свойства **transform-origin** можно изменить исходную точку. Это свойство в качестве значения принимает значения в пикселях, em и процентах. Также для установки точки можно использовать ключевые слова:

- left top: левый верхний угол элемента
- left bottom: левый нижний угол элемента
- right top: правый верхний угол элемента
- right bottom: правый нижний угол элемента

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Трансформации в CSS3</title>
    <style>
      div{
        background-color: #ccc;
        width: 100px;
        height: 100px;
        margin: 80px 30px;
        float: left;
        box-sizing: border-box;
        border: 1px solid #333;
      }
      .transform1 {
        transform: rotate(-45deg);
      }
      .transform2 {
        transform-origin: left top;
        transform: rotate(-45deg);
      }
      .transform3 {
        transform-origin: right bottom;
        transform: rotate(-45deg);
      }
    </style>
  </head>
  <body>
    <div class="transform1"></div>
    <div class="transform2"></div>
    <div class="transform3"></div>
  </body>
</html>
```



4.1.2 ПЕРЕХОДЫ

Переход (transition) представляет анимацию от одного стиля к другому в течение определенного периода времени.

Для создания перехода необходимы прежде всего два набора свойств CSS: начальный стиль, который будет иметь элемент в начале перехода, и конечный стиль - результат перехода. Так, рассмотрим простейший переход:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Переход в CSS3</title>
    <style>
      div{
        width: 100px;
        height: 100px;
        margin: 40px 30px;
        border: 1px solid #333;

        background-color: #ccc;
        transition-property: background-color;
        transition-duration: 2s;
      }
      div:hover{
        background-color: red;
      }
    </style>
  </head>
</body>
```

Анимация набора свойств

При необходимости мы можем анимировать сразу несколько свойств CSS. Так, в выше приведенном примере изменим стили следующим образом:

```
div{
  width: 100px;
  height: 100px;
  margin: 40px 30px;
```

```
border: 1px solid #333;
background-color: #ccc;

transition-property: background-color, width, height, border-color;
transition-duration: 2s;
}
div:hover{
background-color: red;
width: 120px;
height: 120px;
border-color: blue;
}
```

Здесь анимируются сразу четыре свойства. Причем анимация для них всех длится 2 секунды, но мы можем для каждого свойства задать свое время:

```
transition-property: background-color, width, height, border-color;
transition-duration: 2s, 3s, 1s, 2s;
```

Подобно тому как в свойстве `transition-property` через запятую идет перечисление анимируемых свойств, в свойстве `transition-duration` идет перечисление через запятую временных периодов для анимации этих свойств. Причем сопоставление времени определенному свойству идет по позиции, то есть свойство `width` будет анимироваться 3 секунды.

Кроме перечисления через запятую всех анимируемых свойств мы можем просто указать ключевое слово **all**:

```
transition-property: all;
transition-duration: 2s;
```

Теперь будут анимироваться все необходимые свойства, которые меняют значения в стиле для псевдокласса `:hover`.

Функции анимации

Свойства **transition-timing-function** позволяет контролировать скорость хода и выполнение анимации. То есть данное свойство отвечает за то, как и в какие моменты времени анимация будет ускоряться или замедляться.

В качестве значения это свойство может принимать одну из функций:

- **linear**: линейная функция плавности, изменение свойства происходит равномерно по времени
- **ease**: функция плавности, при которой анимация ускоряется к середине и замедляется к концу, предоставляя более естественное изменение
- **ease-in**: функция плавности, при которой происходит только ускорение в начале
- **ease-out**: функция плавности, при которой происходит только ускорение в конце анимации
- **ease-in-out**: функция плавности, при которой анимация ускоряется к середине и замедляется к концу, предоставляя более естественное изменение
- **cubic-bezier**: для анимации применяется кубическая функция Безье

Задержка перехода

Свойство **transition-delay** позволяет определить задержку перед выполнением перехода:

transition-delay: 500ms;

Временной период также указывается в секундах (s) или миллисекундах (ms).

Свойство transition

Свойство **transition** представляет сокращенную запись выше рассмотренных свойств. Например, следующее описание свойств:

transition-property: background-color;

transition-duration: 3s;

transition-timing-function: ease-in-out;

transition-delay: 500ms;

Будет аналогично следующей записи:

transition: background-color 3s ease-in-out 500ms;

4.1.3 АНИМАЦИЯ

Анимация предоставляет большие возможности за изменением стиля элемента. При переходе у нас есть набор свойств с начальными значениями, которые имеет элемент до начала перехода, и конечными значениями, которые устанавливают после завершения перехода. Тогда как при анимации мы можем иметь не только два набора значений - начальные и конечные, но и множество промежуточных наборов значений.

Анимация опирается на последовательную смену ключевых кадров (keyframes). Каждый ключевой кадр определяет один набор значений для анимируемых свойств. И последовательная смена таких ключевых кадров фактически будет представлять анимацию.

По сути переходы применяют ту же модель - так же неявно определяются два ключевых кадра - начальный и конечный, а сам переход представляет переход от начального к конечному ключевому кадру. Единственное отличие анимации в данном случае состоит в том, что для анимации можно определить множество промежуточных ключевых кадров.

В целом объявление ключевого кадра в CSS3 имеет следующую форму:

```
@keyframes название_анимации {
  from {
    /* начальные значения свойств CSS */
  }
  to {
    /* конечные значения свойств CSS */
  }
}
```

После ключевого слова **@keyframes** идет имя анимации. Затем в фигурных скобках определяются как минимум два ключевых кадра. Блок после ключевого слова **from** объявляется начальный ключевой кадр, а после ключевого слова **to** в блоке определяется конечный ключевой кадр. Внутри каждого ключевого кадра определяется одно или несколько свойств CSS, подобно тому, как создается обычный стиль.

Например, определим анимацию для фонового цвета элемента:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Анимация в CSS3</title>
    <style>
      @keyframes backgroundColorAnimation {
        from {
          background-color: red;
        }
        to {
          background-color: blue;
        }
      }
      div{
        width: 100px;
        height: 100px;
        margin: 40px 30px;
        border: 1px solid #333;

        background-color: #ccc;
        animation-name: backgroundColorAnimation;
        animation-duration: 2s;
      }
    </style>
  </head>
  <body>
    <div></div>
  </body>
</html>
```

В данном случае анимация называется `backgroundColorAnimation`. Анимация может иметь произвольное название.

Эта анимация предоставляет переход от красного цвета фона к синему. Причем после завершения анимации будет устанавливаться тот цвет, который определен у элемента `div`.

Чтобы прикрепить анимацию к элементу, у него в стиле применяется свойство **animation-name**. Значение этого свойства - название применяемой анимации.

Также с помощью свойства **animation-duration** необходимо задать время анимации в секундах или миллисекундах. В данном случае время анимации - это 2 секунды.

При подобном определении анимация будет запускаться сразу после загрузки страницы. Однако можно также запускать анимацию по действию пользователя. Например, с помощью определения стиля для псевдокласса :hover можно определить запуск анимации при наведении указателя мыши на элемент:

```
@keyframes backgroundColorAnimation {
  from {
    background-color: red;
  }
  to {
    background-color: blue;
  }
}
div{
  width: 100px;
  height: 100px;
  margin: 40px 30px;
  border: 1px solid #333;
  background-color: #ccc;
}
div:hover{

  animation-name: backgroundColorAnimation;
  animation-duration: 2s;
}
```

Множество ключевых кадров

Как уже выше говорилось выше, анимация кроме двух стандартных ключевых кадров позволяет задействовать множество промежуточных. Для определения промежуточного кадра применяется процентное значение анимации, в котором этот кадр должен использоваться:

```
@keyframes backgroundColorAnimation {
  from {
    background-color: red;
  }
  25% {
    background-color: yellow;
  }
  50% {
    background-color: green;
  }
  75% {
    background-color: blue;
  }
  to {
    background-color: violet;
  }
}
```

В данном случае анимация начинается с красного цвета. Через 25% времени анимации цвет меняется на желтый, еще через 25% - на зеленый и так далее.

Также можно в одном ключевом кадре анимировать сразу несколько свойств:

```
@keyframes backgroundColorAnimation {
  from {
    background-color: red;
    opacity: 0.2;
  }
  to {
    background-color: blue;
    opacity: 0.9;
  }
}
```

Также можно определить несколько отдельных анимаций, но применять их вместе:

```
@keyframes backgroundColorAnimation {
  from {
    background-color: red;
  }
  to {
    background-color: blue;
  }
}
@keyframes opacityAnimation {
  from {
    opacity: 0.2;
  }
  to {
    opacity: 0.9;
  }
}
div{
  width: 100px;
  height: 100px;
  margin: 40px 30px;
  border: 1px solid #333;
  background-color: #ccc;

  animation-name: backgroundColorAnimation, opacityAnimation;
  animation-duration: 2s, 3s;
}
```

В качестве значения свойства `animation-name` через запятую перечисляются анимации, и также через запятую у свойства `animation-duration` задается время этих анимаций. Название анимации и ее время сопоставляются по позиции, то есть анимация `opacityAnimation` будет длиться 3 секунды.

Завершение анимации

В общем случае после завершения временного интервала, указанного у свойства `animation-duration`, завершается и выполнение анимации. Однако с помощью дополнительных свойств мы можем переопределить это поведение.

Так, свойство **animation-iteration-count** определяет, сколько раз будет повторяться анимация. Например, 3 повтора анимации подряд:

```
animation-iteration-count: 3;
```

Если необходимо, чтобы анимация запускалась бесконечное количество раз, то этому свойству присваивается значение **infinite**:

```
animation-iteration-count: infinite;
```

При повторе анимация будет начинаться снова с начального ключевого кадра. Но с помощью свойства `animation-direction: alternate`; противоположное направление анимации при повторе. То есть она будет начинаться с конечного ключевого кадра, а затем будет переход к начальному кадру:

```
animation-name: backgroundColorAnimation, opacityAnimation;  
animation-duration: 2s, 2s;  
animation-iteration-count: 3;  
animation-direction: alternate;
```

При окончании анимации браузер устанавливает для анимированного элемента стиль, который был бы до применения анимации. Но свойство `animation-fill-mode: forwards` позволяет в качестве окончательного значения анимируемого свойства установить именно то, которое было в последнем кадре:

```
animation-name: backgroundColorAnimation;  
animation-duration: 2s;  
animation-iteration-count: 3;  
animation-direction: alternate;  
animation-fill-mode: forwards;
```

Задержка анимации

С помощью свойства **animation-delay** можно определить время задержки анимации:

```
animation-name: backgroundColorAnimation;  
animation-duration: 5s;  
animation-delay: 1s; /* задержка в 1 секунду */
```

Функция плавности анимации

Как и к переходам, к анимации можно применять все те же функции плавности:

- **linear**: линейная функция плавности, изменение свойства происходит равномерно по времени
- **ease**: функция плавности, при которой анимация ускоряется к середине и замедляется к концу, предоставляя более естественное изменение
- **ease-in**: функция плавности, при которой происходит только ускорение в начале
- **ease-out**: функция плавности, при которой происходит только ускорение в начале

- **ease-in-out**: функция плавности, при которой анимация ускоряется к середине и замедляется к концу, предоставляя более естественное изменение
 - **cubic-bezier**: для анимации применяется кубическая функция Безье
- Для установки функции плавности применяется свойство **animation-timing-function**:

```
@keyframes backgroundColorAnimation {
  from {
    background-color: red;
  }
  to {
    background-color: blue;
  }
}
div{
  width: 100px;
  height: 100px;
  margin: 40px 30px;
  border: 1px solid #333;

  animation-name: backgroundColorAnimation;
  animation-duration: 3s;
  animation-timing-function: ease-in-out;
}
```

Свойство animation

Свойство **animation** является сокращенным способом определения выше рассмотренных свойств:

animation: animation-name animation-duration animation-timing-function
animation-iteration-count animation-direction animation-delay animation-fill-mode

Первые два параметра, которые предоставляют название и время анимации, являются обязательными. Остальные значения не обязательны.

Возьмем следующий набор свойств:

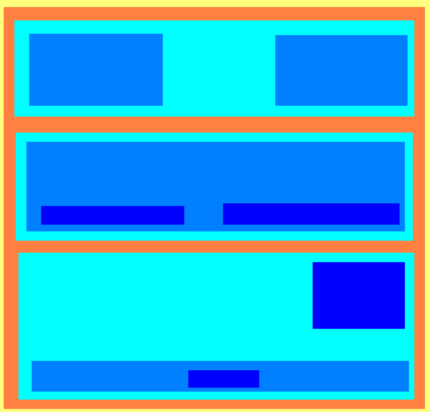
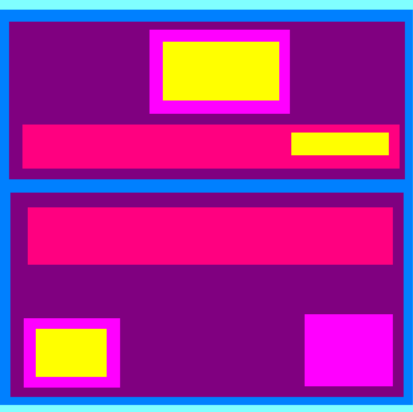
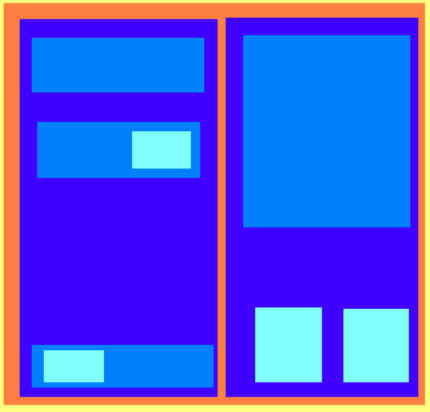
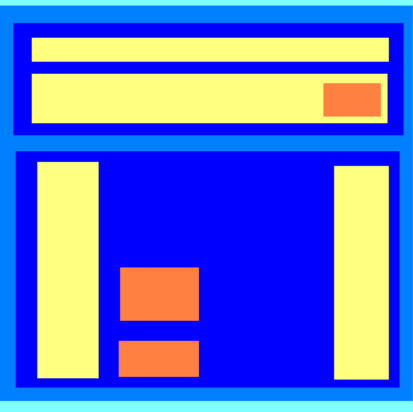
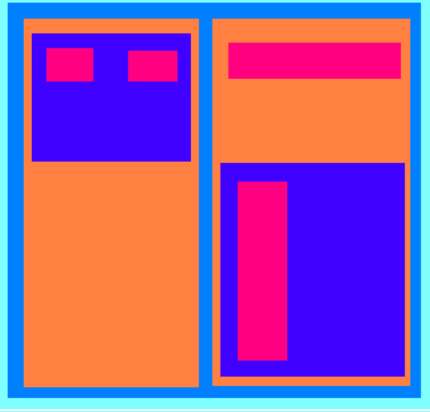
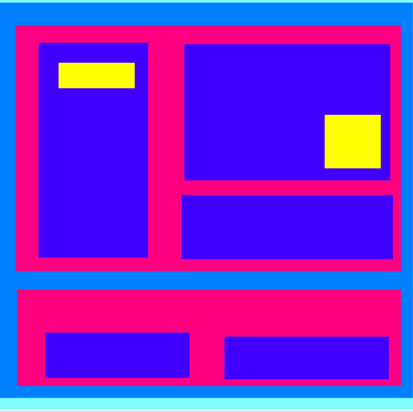
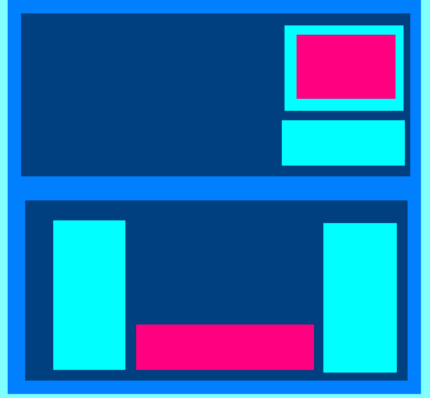
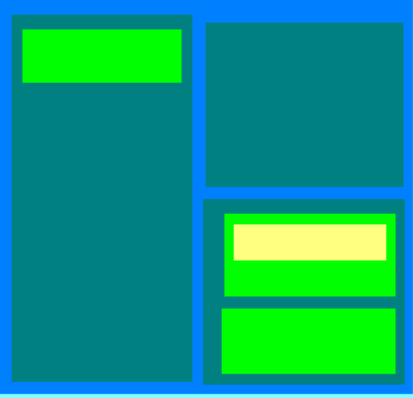
```
animation-name: backgroundColorAnimation;
animation-duration: 5s;
animation-timing-function: ease-in-out;
animation-iteration-count: 3;
animation-direction: alternate;
animation-delay: 1s;
animation-fill-mode: forwards;
```

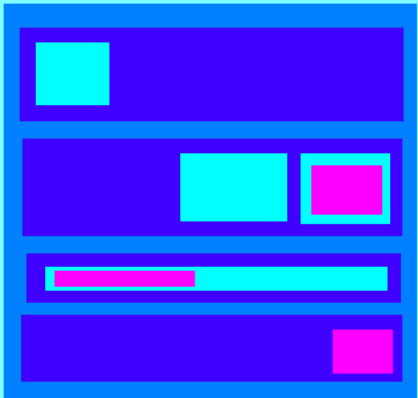
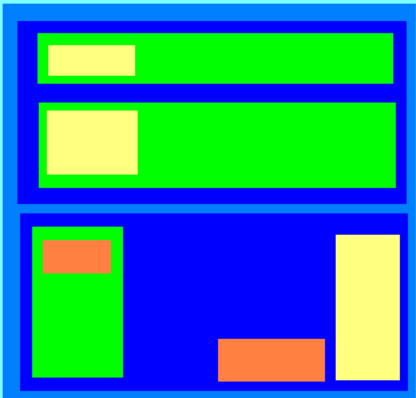
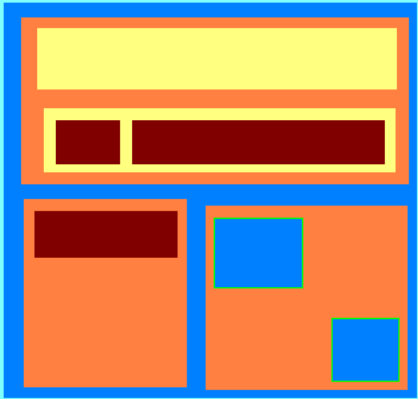
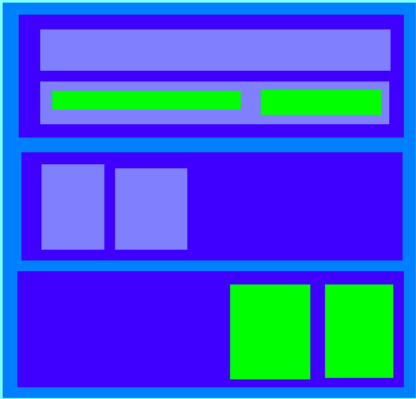
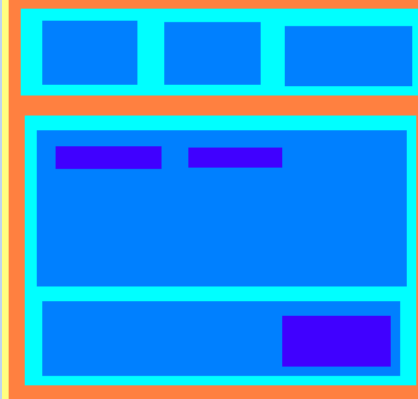
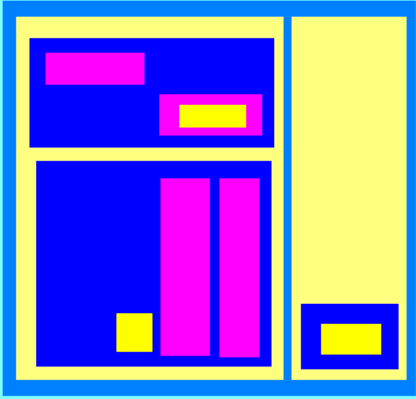
Этот набор будет эквивалентен следующему определению анимации:

```
animation: backgroundColorAnimation 5s ease-in-out 3 alternate 1s
forwards;
```

Задание 1

Расположить блоки, используя *Flexbox*, на странице так, как показано на картинке.


Варианты заданий (по списку подгруппы)			
1		2	
3		4	
5		6	
7		8	



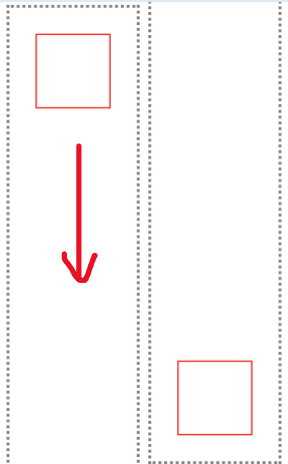

9		10	
11		12	
13		14	

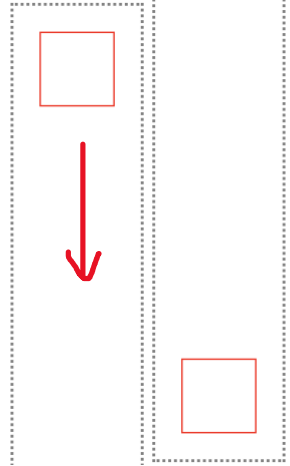
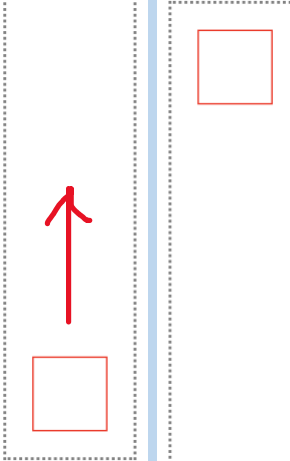

Задание 2

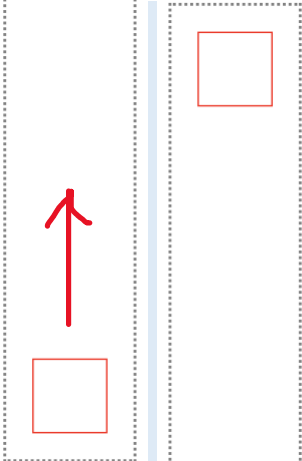



Реализовать CSS анимацию согласно варианту.

Варианты заданий (по списку подгруппы)

1	<p>Реализовать анимацию движения квадрата вертикально влево в течении 3 секунд. После окончания анимация должна повторяться снова.</p> 
---	---

2	<p>Реализовать анимацию смены фона квадрата, анимация должна повторяться бесконечно.</p> 
3	<p>Реализовать анимацию движения квадрата вертикально влево в течении 3 секунд. После окончания анимация должна повториться еще 4 раза и остановиться.</p> 
4	<p>Реализовать анимацию движения квадрата сверху вниз в течении 3 секунд. После окончания анимация должна повторяться снова.</p> 
5	<p>Реализовать анимацию движения квадрата вертикально влево в течении 3 секунд. После окончания анимация должна повториться еще 2 раза и остановиться.</p> 
6	<p>Реализовать анимацию движения квадрата сверху вниз в течении 3 секунд. После окончания анимация должна повториться еще 4 раза и остановиться.</p>

		
7	Реализовать анимацию движения квадрата снизу вверх в течении 3 секунд. После окончания анимация должна повториться еще 4 раза и остановиться.	
		
8		
9	Реализовать анимацию движения квадрата вертикально вправо в течении 3 секунд. После окончания анимация должна повторяться снова.	
		
10	Реализовать анимацию движения квадрата снизу вверх в течении 3 секунд. После окончания анимация должна повторяться снова.	

	
11	<p>Реализовать анимацию движения квадрата вертикально вправо в течении 3 секунд. После окончания анимация должна повториться еще 4 раза и остановиться.</p> 
12	<p>Реализовать анимацию смены фона квадрата, анимация должна повториться 2 раза.</p> 
13	<p>Реализовать анимацию движения квадрата вертикально вправо в течении 3 секунд. После окончания анимация должна повториться еще 2 раза и остановиться.</p> 
14	<p>Реализовать анимацию смены фона квадрата, анимация должна повториться 4 раза.</p>



! Контрольные вопросы !

1. Сайт имеет более ста HTML-документов, имеющих одинаковое стилевое оформление. Какой способ подключения CSS подходит лучше всего?
 - Связанные стили.
 - Глобальные стили.
 - Блочные стили.
 - Внутренние стили.
 - Экспорт стиля.
2. Какой HTML-код применяется для подключения внешнего CSS-файла?
 - `<style>mystyle.css</style>`
 - `<style>@mystyle.css</style>`
 - `<link rel="stylesheet" href="mystyle.css">`
 - `<link>@import url(mystyle.css)</link>`
 - `<stylesheet>mystyle.css</stylesheet>`
3. Какой атрибут используется для определения внутреннего стиля?
 - style
 - class
 - styles
 - font
 - link
4. Перечислите способы внедрения сценариев в HTML-документ.
5. Для чего используется атрибут async?
6. Для чего используется атрибут defer?
7. Назовите основные составляющие компоновки flexbox?
8. Какие значения может принимать свойства display?
9. Для управления направлением элементов CSS3 используют свойство flex-direction. Какие значения оно может принимать?
10. Что определяет свойство flex-wrap?
11. Какое свойство позволяет установить значения сразу для обоих свойств flex-direction и flex-wrap?
12. Опишите принцип работы свойства justify-content. Какие значения принимает это свойство?
13. Опишите принцип работы свойства align-items. Какие значения принимает это свойство?
14. Назовите три свойства, которые позволяют управлять элементами (начальный размер flex-элемента, как flex-элемент будет увеличиваться/уменьшаться относительно других элементов в контейнере)?
15. Для чего используют Grid? Отличие Grid от Flexbox.

16. Какие свойства CSS3 используют для установки строк и столбцов в Grid Layout?
17. Для чего используют функцию repeat()?
18. С помощью каких свойств можно более точно настроить расположение элемента в гриде?
19. Что такое анимация?
20. Перечислите и охарактеризуйте функции плавности анимации.

Содержание отчёта

1. Ф.И.О., группа, название лабораторной работы.
2. Цель работы.
3. Описание проделанной работы.
4. Результаты выполнения лабораторной работы.
5. Выводы.