Лабораторная работа №4.

Тема: Создание адаптивного макета с CSS Flexbox.

Цель работы: освоить основы Flexbox для создания адаптивных макетов, совместимых с разными разрешениями экрана.

Основные термины и понятия:

CSS Flexbox (от *flexible box*, что переводится как «гибкий контейнер») – это мощный инструмент для создания адаптивной и гибкой верстки веб-страниц. В отличие от традиционных способов позиционирования элементов, таких как float или inline-block, Flexbox предоставляет разработчику простые, интуитивные и более мощные средства для управления выравниванием, распределением пространства и порядком элементов внутри контейнера.

Одной из ключевых особенностей Flexbox является его способность автоматически подстраиваться под доступное пространство, обеспечивая удобное создание адаптивных интерфейсов. Это делает его особенно полезным для современных веб-приложений, где важно корректное отображение элементов на экранах разных размеров.

Основные преимущества Flexbox:

– Простое выравнивание элементов: Flexbox упрощает процесс центрирования, выравнивания и распределения элементов как по горизонтали, так и по вертикали.

– Адаптивность: элементы автоматически подстраиваются под размеры контейнера, что позволяет легко реализовывать сложные макеты без необходимости задавать точные размеры для каждого элемента.

– Удобство управления пространством: Flexbox позволяет равномерно распределять свободное место или сжимать элементы, чтобы они помещались в ограниченном пространстве.

– Поддержка большинства браузеров: Flexbox стабильно работает во всех современных браузерах, включая мобильные версии.

Flexbox основан на двух основных сущностях:

– **Flex-контейнер (flex-container)** – это элемент, к которому применяется свойство display: flex или display: inline-flex. После применения одного из этих свойств к элементу, он становится «гибким контейнером» и получает возможность управлять расположением и поведением своих дочерних элементов.

– **Flex-элементы (flex-items)** – это прямые дочерние элементы flex-контейнера. Только они подчиняются правилам Flexbox. Элементы второго и более глубокого уровня вложенности не затрагиваются, если не задать для них отдельный flex-контейнер.

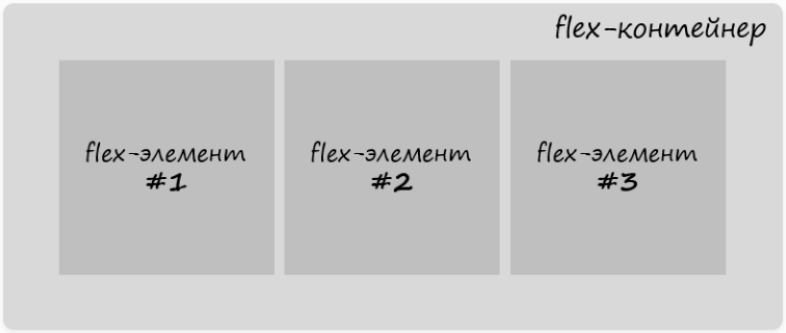
.css

```
.container {
  display: flex; /* inline-flex */
}
```

.html

```
<div class="container">
  <div class="item">Элемент 1</div>
  <div class="item">Элемент 2</div>
  <div class="item">Элемент 3</div>
</div>
```

В этом примере является flex-контейнером, а вложенные – flex-элементами.



Расположение Flex-контейнера на странице. Значение свойства display определяет, как flex-контейнер будет отображаться на странице:

– flex – контейнер ведет себя как блочный элемент, занимая всю ширину родителя.

– inline-flex – контейнер ведет себя как строчный элемент, занимая ровно столько места, сколько нужно для отображения его содержимого.

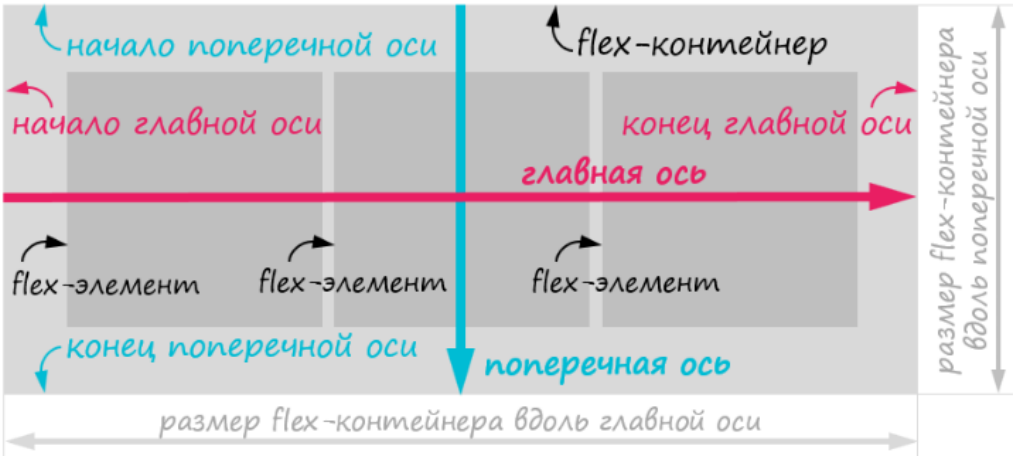
В результате:

– Используйте display: flex, если вам нужен гибкий макет, который растягивается на всю ширину.

– Используйте display: inline-flex, если flex-контейнер должен вписываться в строчный поток, занимая минимально необходимое пространство.

Оси Flexbox.

Flexbox работает на основе двух взаимосвязанных осей, которые определяют расположение и выравнивание flex-элементов внутри flex-контейнера.



Главная ось (main axis) – определяет основное направление, по которому располагаются flex-элементы. По умолчанию главная ось направлена горизонтально, слева направо. Её ориентация задается с помощью свойства flex-direction.

Поперечная ось (cross axis) – перпендикулярна главной оси. Если главная ось горизонтальная, то поперечная будет вертикальной, и наоборот. Элементы можно выравнивать по этой оси с помощью свойств, таких как align-items и align-self.

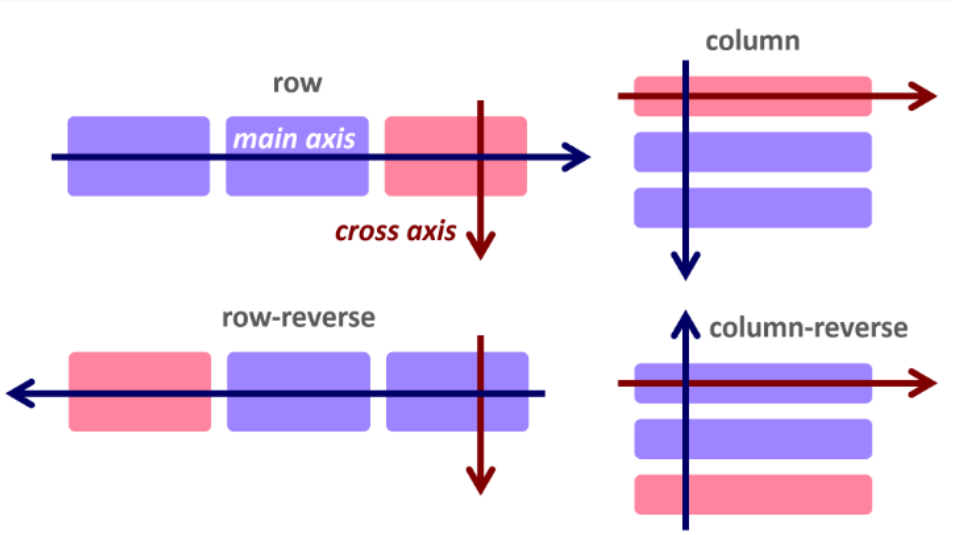
Направление главной и поперечной осей *настраивается* с помощью свойства flex-direction. Оно управляет порядком и направлением расположения flex-элементов:

– **row** – элементы располагаются по горизонтали слева направо (значение по умолчанию).

– **row-reverse** – элементы располагаются по горизонтали справа налево.

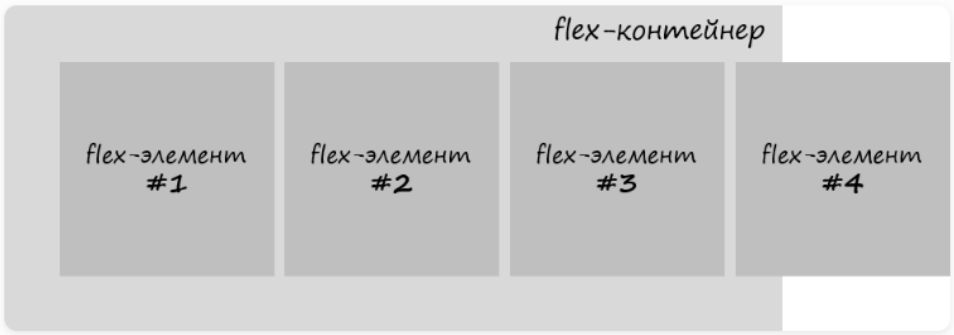
– **column** – элементы располагаются по вертикали сверху вниз.

– **column-reverse** – элементы располагаются по вертикали снизу вверх.



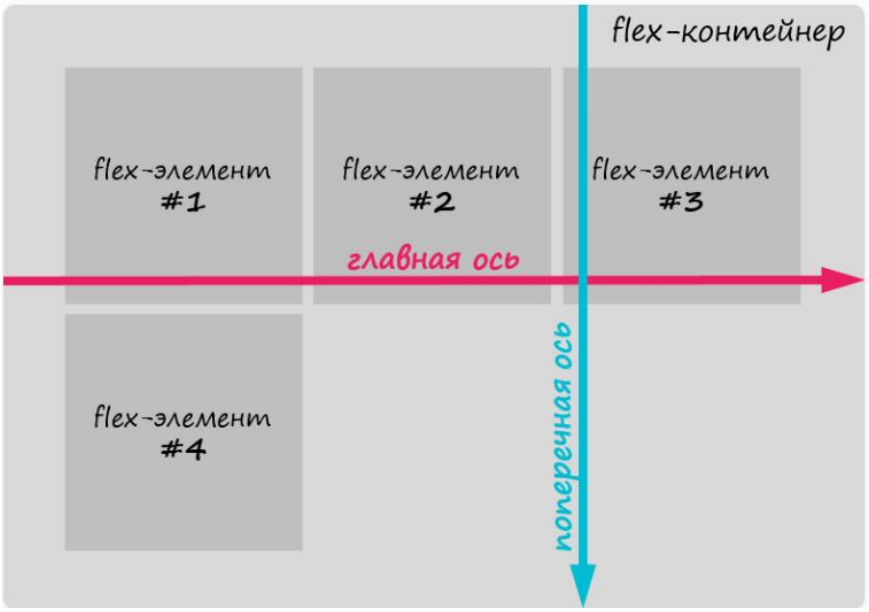
С помощью этого свойства можно сделать так, чтобы flex-элементы располагались не рядами (rows), а колонками (columns).

По умолчанию flex-элементы не переносятся на новую линию, даже когда им не хватает места в текущей линии. Они просто выходят за её пределы.



Но это можно изменить. Разрешить перенос flex-элементов на новые линии осуществляется с помощью установки flex-контейнеру свойства flex-wrap со значением wrap или wrap-reverse.

```
flex-wrap: wrap;
/* nowrap (только на одной линии - по умолчанию)
    wrap (разрешить перенос flex-элементов на новые линии)
    wrap-reverse (осуществлять перенос flex-элементов в обратном порядке) */
```



Значения wrap и wrap-reverse свойства flex-wrap определяют **направление поперечной оси**.

Свойства flex-direction и flex-wrap можно указать с помощью универсального свойства flex-flow:

```
flex-flow: row nowrap; /* 1 значение - flex-direction, 2 значение - flex-wrap */
```

Одной из главных функций Flexbox является управление пространством и выравниванием элементов:

Элементы могут растягиваться, сжиматься или сохранять свои размеры в зависимости от доступного пространства.

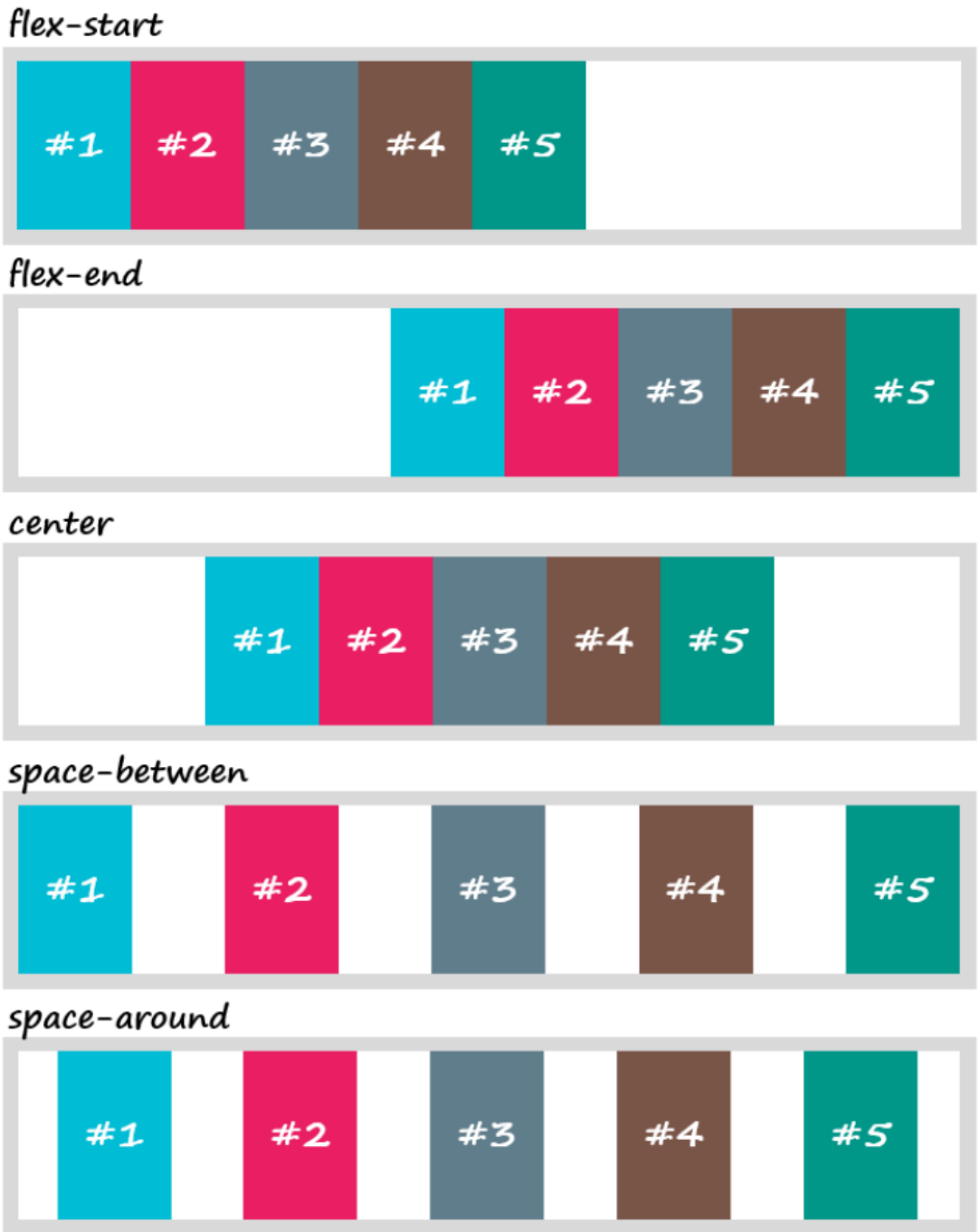
Свойства Flexbox позволяют равномерно распределять элементы, центрировать их или выравнивать по краям, как по главной, так и по поперечной оси.

Во Flexbox выравнивание элементов внутри контейнера осуществляется по двум направлениям (осям).

Выравнивание flex-элементов по направлению главной оси

Выравнивание элементов вдоль основной оси осуществляется с помощью CSS свойства justify-content:

```
justify-content: flex-start;
/* flex-start (flex-элементы выравниваются относительно начала оси) – по умолчанию
    flex-end (flex-элементы выравниваются относительно конца оси)
    center (по центру flex-контейнера)
    space-between (равномерно, т.е. с одинаковым расстоянием между flex-элементами)
    space-around (равномерно, но с добавлением половины пространства перед первым flex-
элементом и после последнего) */
```



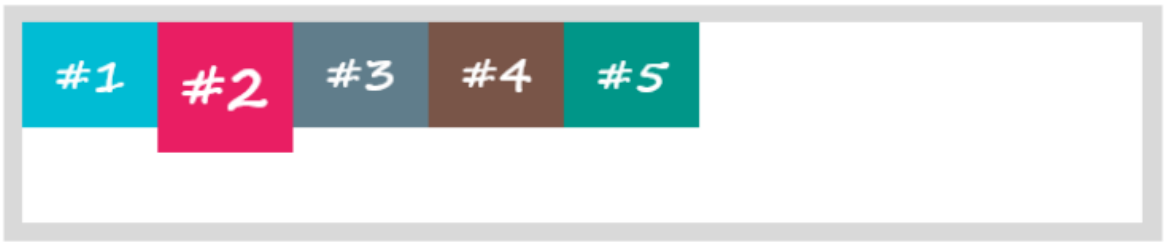
Выравнивание flex-элементов вдоль поперечной оси

Выравнивание flex-элементов во flex-контейнере по направлению поперечной оси осуществляется с помощью свойства align-items:

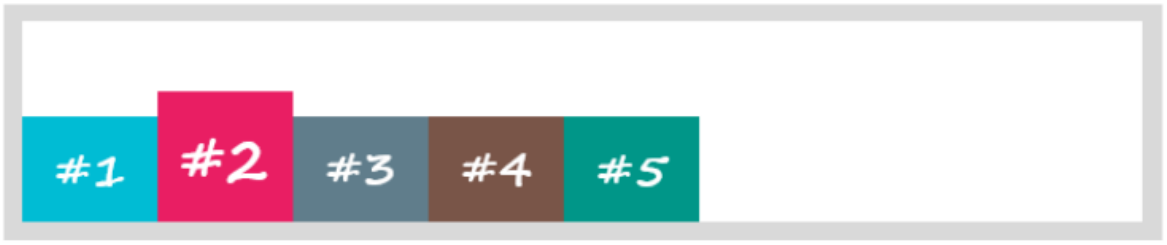
```
align-items: stretch;
/* stretch (растягиваются по всей длине линии вдоль направления поперечной оси) – по умолчанию
    flex-start (располагаются относительно начала поперечной оси)
    flex-end (относительно конца поперечной оси)
    baseline (относительно базовой линии)
    center (по центру) */
```

stretch

flex-start



flex-end



baseline

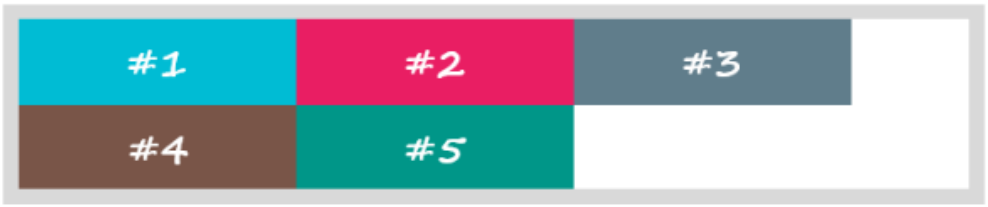
center

Выравнивание линий flex-контейнера

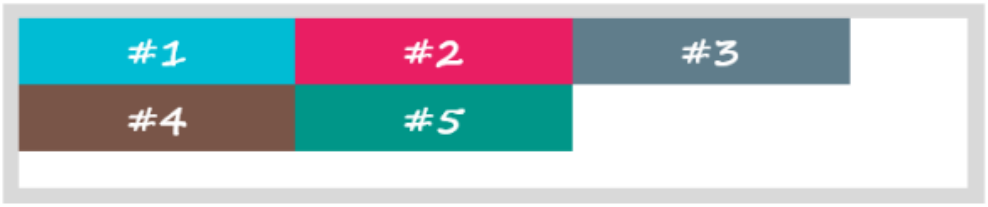
Flexbox позволяет выравнивать не только сами flex-элементы, но и линии, на которых они расположены.

```
align-content: stretch
/* stretch (растягиваются по всей длине поперечной оси) – по умолчанию
    flex-start (относительно начала поперечной оси)
    flex-end (относительно конца поперечной оси)
    center (по центру)
    space-between (равномерно, т.е. с одинаковым расстоянием между линиями)
    space-around (равномерно, но с добавлением половины пространства перед первой линией
и после последней) */
```

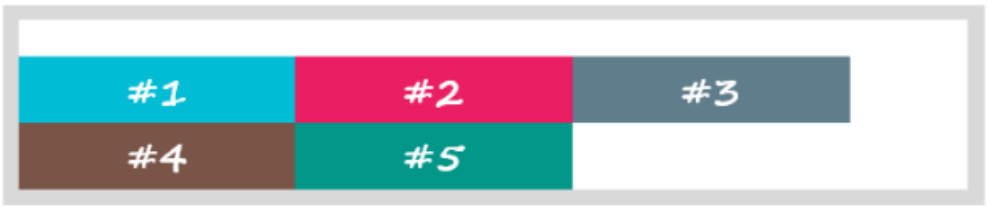
stretch



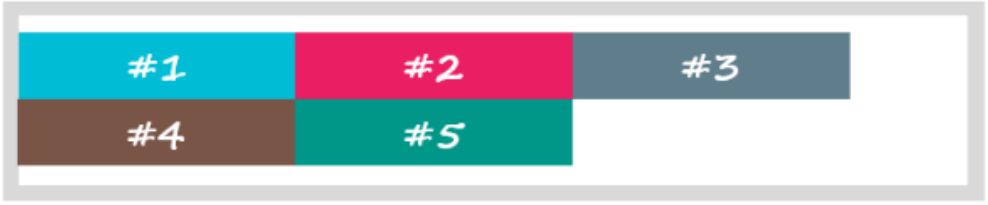
flex-start



flex-end

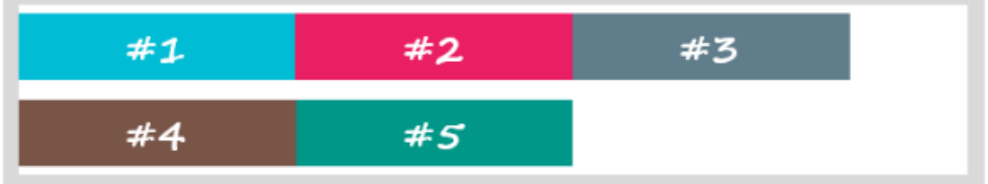


center



space-between

space-around



Свойство align-content имеет смысл использовать только тогда, когда flex-элементы во flex-контейнере располагаются на нескольких линиях. Чтобы это произошло, необходимо, во-первых, чтобы ширина всех flex-элементов была больше ширины flex-контейнера, во-вторых, flex-контейнер должен иметь в качестве CSS-свойства flex-wrap значение wrap или wrap-reverse.

CSS-свойство align-self

Свойство align-self в отличие от предыдущих (justify-content, align-items и align-content) предназначено для flex-элементов. Оно позволяет изменить **выравнивание flex-элемента** вдоль направления поперечной оси. Свойство align-self может принимать такие же значения, как и align-items.

```
align-self: stretch; /* auto (по умолчанию) || stretch || flex-start || flex-end || baseline || center */
```

Изменение порядка следования flex-элементов

По умолчанию flex-элементы отображаются во flex-контейнере в том порядке, в котором они расположены в HTML коде. Для изменения порядка следования одних flex-элементов относительно других в CSS Flexbox можно использовать свойство order. Данное CSS свойство выстраивает flex-элементы во flex-контейнере в порядке возрастания их номеров.

.css

```
order: 0;
/*
  0 (по умолчанию)
  целое положительное или отрицательное
число
*/
```

.html

```
<div class="flex-container">
  <div class="flex-container-element-1">...</div>
  <div class="flex-container-element-2">...</div>
  <div class="flex-container-element-3">...</div>
  <div class="flex-container-element-4">...</div>
</div>
```

.css

```
.flex-container {
  display: flex;
}
/* переместим 2 flex-элемент в конец */
.flex-container-element-2 {
  order: 2;
}
/* передвинем 3 элемент на место второго */
.flex-container-element-3 {
  order: 1;
}
/* расположим 4 flex-элемент в начало */
.flex-container-element-4 {
  order: -1;
}
```

Основные свойства flex-контейнера:

– flex-direction: определяет направление главной оси (row, column).

– flex-wrap: указывает, должны ли flex-элементы переноситься на новую строку (nowrap, wrap, wrap-reverse).

– justify-content: выравнивает flex-элементы вдоль главной оси (flex-start, flex-end, center, space-between, space-around, space-evenly).

– align-items: выравнивает flex-элементы вдоль поперечной оси (flex-start, flex-end, center, baseline, stretch).

– align-content: выравнивает строки с flex-элементами вдоль поперечной оси, когда есть свободное пространство (flex-start, flex-end, center, space-between, space-around, stretch).

Основные свойства flex-элементов:

– order: определяет порядок расположения элемента среди других flex-элементов.

– flex-grow: указывает, насколько элемент может расти, если есть свободное пространство.

– flex-shrink: определяет, насколько элемент может уменьшаться, если не хватает пространства.

– flex-basis: задает базовый размер элемента перед распределением свободного пространства.

– align-self: позволяет задать выравнивание для отдельного flex-элемента вдоль поперечной оси.

Медиазапросы

Одним из самых важных инструментов при создании адаптивной вёрстки является использование медиазапросов. Медиазапросы – специальные условные конструкции, которые позволяют применять стили только для определённых устройств.

Медиазапросы записываются следующим образом:

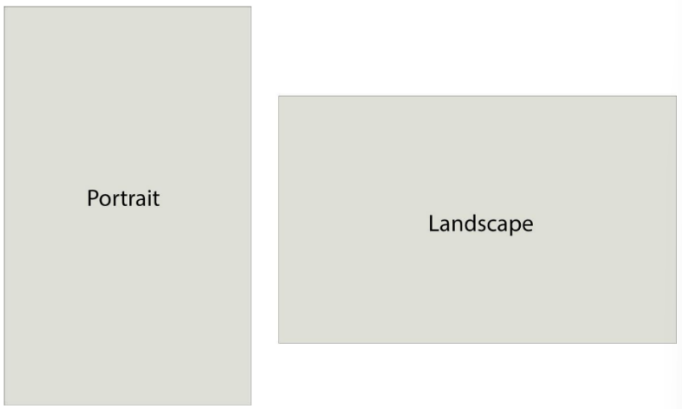
```
@media (условия) {
 /* CSS-код, который будет выполнен для данного условия */
}
```

В качестве *условия* могут выступать различные значения и константы.

Ориентация экрана

Для определения ориентации экрана используется ключ orientation, значением которого может выступать одно из двух значений:

1. landscape. Условие выполнится для устройств с *горизонтальной* ориентацией экрана. Горизонтальная – ориентация, при которой ширина *viewport* больше его высоты.
2. portrait. Условие выполнится для устройств с *вертикальной* ориентацией экрана. Вертикальная – ориентация, при которой высота *viewport* больше его ширины.



```
@media (orientation: landscape) {
  /* При горизонтальной ориентации фоновым цветом сайта будет белый */
  body {
    background: #FFF;
  }
}

@media (orientation: portrait) {
  /* При вертикальной ориентации фоновым цветом сайта будет чёрный */
  body {
    background: #000;
  }
}
```

Разрешение экрана

При использовании медиазапросов также можно исходить из ширины или высоты *viewport*. Для этого используются знакомые по обычным CSS-правилам условия *width, min-width, max-width* для ширины и *height, min-height, max-height* для высоты.

С помощью таких условий создаются *breakpoint* – контрольные точки. Это границы значений, по которым видоизменяется макет. Такие точки остановки позволяют иметь правила для мониторов, планшетов, телефонов.

```
/* Здесь будут все стили для устройств с viewport больше 1400 пикселей. */

@media (max-width: 1400px) {
  /*
  Стили для устройств, у которых ширина viewport меньше или равно 1400 пикселей,
  но больше 990 пикселей.
  Эти стили будут использованы для планшетов и ноутбуков с низким разрешением
  */
}

@media (max-width: 990px) {
  /*
  Стили для устройств, у которых ширина viewport меньше или равно 990 пикселей,
  но больше 770 пикселей.
  Эти стили подойдут для некоторых мобильных устройств и планшетов
  */
}

@media (max-width: 770px) {
  /*
  Стили для устройств, у которых ширина viewport меньше или равно 770 пикселей.
  Это множество мобильных устройств
  */
}
```

Обратите внимание на порядок написания свойств. Помните, что CSS является каскадной таблицей, поэтому порядок стилей необходимо контролировать. В данном случае к элементу вначале будет применён стиль по умолчанию, который не находится в медиазапросе, затем поочерёдно будут применяться стили в зависимости от значений в условии медиазапроса.

Например, при ширине *viewport* 770 пикселей для элемента стили применятся в следующем порядке:

– Стили по умолчанию.

– Стили для условия медиазапроса max-width: 1400px.

– Стили для условия медиазапроса max-width: 990px.

– Стили для условия медиазапроса max-width: 770px.

Подход, описанный выше называется *Desktop First*. В начале пишутся стили для больших мониторов, а в последствии, используя медиазапросы, – стили для меньших значений *viewport*. Его характерная черта в медиазапросах – использование конструкции *max-width* в качестве условия.

В противовес *Desktop First* существует подход *Mobile First*. Его особенностью является то, что вначале пишутся стили под мобильные устройства, а затем, используя медиазапросы, пишутся стили для больших размеров *viewport*. Если в *Desktop First* основной конструкцией являлось использование *max-width*, то в *Mobile First* используется *min-width*.

Стили, написанные с использованием подхода *Mobile First* выглядят следующим образом:

```
/* Здесь будут все стили для мобильных устройств с viewport меньше 770 пикселей. */

  @media (min-width: 770px) {
    /*
    Стили для устройств, у которых ширина viewport больше или равно 770 пикселей
    */
  }

  @media (min-width: 990px) {
    /*
    Стили для устройств, у которых ширина viewport
    больше или равно 990 пикселей, но меньше 1400 пикселей.
    */
  }

  @media (min-width: 1400px) {
    /*
    Стили для устройств, у которых ширина viewport
    больше или равно 1400 пикселей
    */
  }
```

Логические операторы

Условия внутри медиазапросов можно комбинировать. Для этого существует три логических оператора:

– Логическое *«И»*. Означает, что несколько условий должны быть выполнены для того, чтобы CSS-стили применились к элементу. Для использования логического «И» используется ключевое слово and. Например, условие, которое проверяет, что экран устройства находится в портретной (вертикальной) ориентации и имеет ширину *viewport* не меньше 600 пикселей:

```
@media (orientation: portrait) and (min-width: 600px) {
 .container {
  /* Для устройств с портретной ориентацией И шириной viewport не менее 600 пикселей
сделать элементы с классом container шириной в 100 процентов */
  width: 100%;
 }
}
```

– Логическое *«ИЛИ»*. Свойства применятся в том случае, если хотя бы *одно из* условий будет выполнено. Условия для этого отделяются запятыми. Прошлый пример с использованием «ИЛИ»:

```
@media (orientation: portrait), (min-width: 600px) {
  .container {
    /* Для устройств с портретной ориентацией ИЛИ шириной viewport не менее 600 пикселей
   сделать элементы с классом container шириной в 100 процентов */
    width: 100%;
  }
 }
```

– Логическое *«НЕ»*. Свойства применятся в том случае, если условие *не выполняется*. Используется ключевое слово not:

```
@media not all and (orientation: landscape) {
  .container {
    /* Для устройств с портретной ориентацией (условие выглядит как «НЕ горизонтальная»)
   сделать элементы с классом container шириной в 100 процентов */
    width: 100%;
  }
 }
```

Использование медиазапросов при подключении CSS

Медиазапросы возможно писать не только внутри CSS-файла, но и указывать их в HTML при подключении файла стилей. В этом случае медиазапрос указывается в атрибуте media.

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Подключение CSS файлов</title>

    <!-- Общие стили для проекта -->
    <link rel="stylesheet" href="style.css">

    <!-- Стили для экранов с viewport не менее 750px -->
    <link rel="stylesheet" media="screen and (min-width: 750px)" href="style750px.css">

</head>
<body>
```

Задание:

Сконструируйте адаптивный макет, состоящий из:

– Верхнего горизонтального меню, выровненным с использованием Flexbox.

– Основного содержимого, включающего две колонки (главный контент и боковую панель).

– Нижнего колонтитула с выравниванием текста и элементов.

– Настройте поведение элементов при изменении ширины экрана:

– При ширине экрана менее 768px все элементы должны располагаться вертикально.

Дополнительно:

– Используйте медиазапросы для адаптации шрифтов и отступов.

Ход выполнения лабораторной работы должен быть отражен в *отчете*. Отчет должен содержать титульный лист, цель работы, задание, листинг исходного кода, описание проделанной работы, скриншоты результата, вывод, ответы на контрольные вопросы.

Контрольные вопросы:

1. Определение CSS Flexbox.
2. Какие основные преимущества Flexbox?
3. На каких сущностях основан Flexbox?
4. Какие CSS-свойства для расположения Flex-контейнера?
5. Определение осей Flexbox.
6. Какие CSS-свойства для настраивания направления осей?
7. Опишите свойство flex-direction и его значения. Опишите свойство flex-wrap и его значения.
8. Опишите свойство flex-direction justify-content и его значения.
9. Опишите свойство flex-direction align-items и его значения.
10. Опишите свойство flex-direction align-content и его значения.
11. Опишите свойство flex-direction align-self и его значения.
12. Опишите порядок следования flex-элементов и можно ли его изменить?
13. Перечислите основные свойства флекс-контейнера.
14. Перечислите основные свойства флекс-элементов.
15. Что такое медиазапросы и для чего они нужны?
16. Опишите orientation и его значения.
17. Опишите viewport и его условия.
18. Какие отличия Desktop First и Mobile First?
19. Каким образом можно комбинировать медиазапросы?