Концерн Беллегпром

УО «Барановичский государственный колледж легкой промышленности

имени В.Е. Чернышева»

Учебная дисциплина «Программные средства создания Internet-приложений»

**Инструкция**

по выполнению лабораторной работы №7

«Разработка адаптивной версии веб-страницы под различные разрешения экранов и типы устройств»

Составитель: Самохвал Тамара Васильевна

Обсуждено и одобрено на заседании цикловой комиссии

дисциплин программного обеспечения

Протокол от \_\_\_\_\_\_\_\_\_\_\_\_ №\_\_\_\_\_\_\_\_\_\_

Барановичи

2020

**Цель: Сформировать умения по разработке адаптивной версии веб-страницы**

**Оснащение: ПК, HTML-редактор, браузер**

**Ход работы**

1. **Изучите необходимые теоретические сведения:**

Медиа-запросы – это инструмент для создания отзывчивых сайтов (адаптивных сайтов) с помощью CSS.

В общем случае медиа-запрос состоит из ключевого слова, описывающего тип устройства (необязательный параметр) и выражения, проверяющего характеристики данного устройства. Из всех характеристик чаще всего проверяется ширина устройства width. Медиа-запрос является логическим выражением, которое возвращает истину или ложь.

Медиа-запрос начинается с правила **@media**, после которого следует условие применения стилей, состоящее из типа носителя (например, **all**), логического оператора (**and**) и медиа-функции (**max-width: 360px**). Медиа-запрос включает блок свойств CSS только в том случае, если медиа-функция принимает значение true.

Рассмотрим медиа-запрос у которого **screen** (экран), шириной **max-width:650px**, при котором будут использоваться определённые стили, которые мы занесем в этот медиа запрос.

@media screen and (max-width:650px){

        p{color: red; }

    }

Что выполняет данный медиа запрос? Если ширина экрана будет равна 650px или меньше, то текст закрасится в красный цвет. Если вы пожелаете чтобы текст становился красным не при 650px или меньшей ширине, а на оборот от ширины 650px и больше то вы уже должны использовать не max-width, а min-width.

@media screen and (min-width:650px){

        p{color: red; }

    }

Можно также задавать два отдельных запроса, используя max-width и min-width.

@media screen and (max-width:750px) and (min-width:550px){

        p{color: red; }

    }

Такой медиа запрос определит стили для ширины экрана находящейся в диапазоне между 550px и 750px (включительно) и текст будет окрашиваться в красный цвет только тогда, когда значение ширины экрана будет лежать в этом диапазоне.

Когда Вы хотите задать какие-либо стили за пределами диапазона значений ширины экрана, например, если ширина экрана будет меньше 550px или больше 750px, то медиа-запрос будет иметь следующий вид.

@media screen and (max-width:550px), (min-width:750px){

        p{color: red; }

    }

Работу адаптивной верстки можно сразу проверять в браузере. Достаточно просто уменьшить браузер или воспользоваться инструментами разработчика.

**Как писать медиа-запросы в CSS (2 варианта)**

|  |  |
| --- | --- |
| **Пример 1**  @media (max-width: 1199px){  .container{  width: 960px;  }  .catalog\_\_subcatalog{  width: 670px;  }  } | Можно для одного экрана все стили поместить в одну директиву. |
| **Пример 2**  .container{  width: 1140px;  }  @media (max-width: 1199px){  .container{  width: 960px;  }  }  .catalog\_\_subcatalog{  width: 867px;  }  @media (max-width: 1199px){  .catalog\_\_subcatalog{  width: 670px;  }  } | А можно каждый раз указывать директиву. |

В первом примере все стили группируются по медиа-запросам. Во втором примере для каждого селектора сразу пишется медиа-запрос. И так одинаковых медиа-запросов может быть много.

При помощи медиа-запросов можно создавать так называемые контрольные точки (англ. *breakpoints*) и привязывать к ним CSS-стили. Существуют стандарты, какими должны брейкпойнты в медиа-запросах: 575px, 767px, 991px и 1199px. В коде это выглядит так:

@media (max-width: 1199px){ }

@media (max-width: 991px){ }

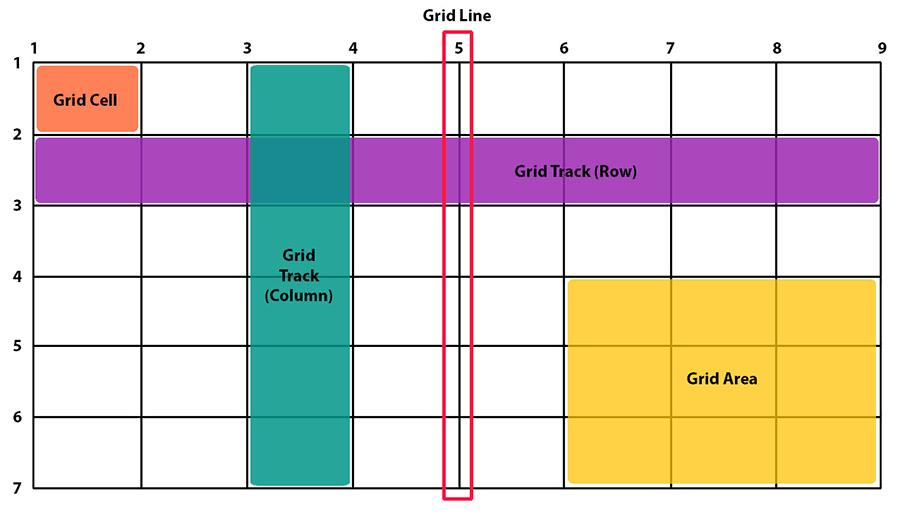
@media (max-width: 767px){ }

@media (max-width: 575px){ }

И в данном случае они так и должны располагаться – в порядке убывания. То есть, если вы сперва написали стили для больших экранов, то потом пишете стили для экранов поменьше, потом еще меньше и так до смартфонов.

CSS Grid Layout – это новое многообещающее явление в мире верстки. Используя Grid, можно построить макеты, которые ранее невозможно было построить в CSS.

Grid – это набор горизонтальных и вертикальных «линий», которые пересекаются между собой и создают сетку из рядов и колонок. Элементы могут быть помещены в сетку, опираясь на номер линии или номер ряда/колонки.

[](https://wp-kama.ru/wp-content/uploads/2018/05/struktura-grid-setki.png)

##### **Описание Грид сетки**

* **Контейнер** – содержит Grid сетку, в которой находятся элементы.
* **Элементы** – HTML элементы внутри сетки. Ими станут HTML элементы первого уровня (прямые дети контейнера). Чтобы элемент появился в сетке, в нем (внутри) элемента должно быть хоть что-то (текст, другие HTML теги). Пустой элемент - это всего лишь ячейка для расположения в нем чего-либо.
* **Линии** – это образные линии (на самом деле никаких линий нет), разделяющие сетку на колонки и ряды, они создают структуру сетки. Линии автоматически нумеруются. Также линиям можно указывать имена, чтобы потом прикреплять к ним элементы по номеру или по имени линии. По сути, линия – это номер или имя колонки/ряда. Расстояние между линиями (колонками/рядами) можно указать через grid-gap:, grid-row-gap:, grid-column-gap:.
* **Ряд/колонка (row/column, track)** – все, что находится между соседними линиями, т.е. линии разделяют сетку на ряды и колонки.
* **Ячейка (cell)** – место, куда будет расположен элемент. Ячейка это пересечение колонки и ряда.
* **Область (area, поле)** – объединение одной или нескольких ячеек в общую ячейку (поле). Это такая большая ячейка, также ограниченная линиями. Области можно задать имя, чтобы удобно было помещать туда элементы.
* **Разрыв (gap)** – расстояние между рядами и колонками. Разрывает линию на две. Так между линиями, а как следствие и колонками/рядами, ячейками появляется пустое место. Это своего рода margin, border-spacing между ячейками. По умолчанию линия между ячейками всего одна (ячейки слеплены), но если указать разрыв, то мы разорвем линию, и появится расстояние между колонками/рядами, при этом номер или имя линии (колонки/ряда) остается одно.

Для включения Grid, любому HTML элементу достаточно присвоить css свойство display:grid;. После включения grid свойства, внутри контейнера создаются grid сетка, а все вложенные элементы (первого уровня) станут ячейками сетки.

Пример создания грид-блока с двумя столбцами и тремя рядами разных размеров:

.grid{

**display**: grid;

**grid**: 1fr 25% 30px / 40% 1fr; /\* ряды / колонки \*/

**grid-gap**: 1em;

**height**: 200px;

}

##### **Особенности Grid**

Элементы Grid сетки можно расставлять сразу на несколько полей сетки. Можно менять направление или расположения элементов в сетке. Колонкам и рядам можно давать имена. Можно создавать шаблон сетки и расставлять элементы по шаблону.

* **Размеры колонок/рядов**. Сетку можно создавать с точными или гибкими размерами колонок/рядов (шириной/высотой). Точные это px, em, %, а гибкие новая единица измерения в grid fr (фракция - свободное место в сетке).
* **Расположение элемента.** Элементы можно размещать в указанном месте сетки, указав номер колонки/ряда или их имя (если оно есть). Или путем привязки элемента к области Grid (область нужно создать). Если не указать конкретное расположение элемента в сетке, то элемент размещается по умолчанию в первую свободную ячейку: как во flex: по горизонтали (→) или по вертикали (↓). Поведение по умолчанию можно изменить через свойство grid-auto-flow:.
* **Выравнивание элементов**. Элементы внутри ячейки можно выравнивать по горизонтали/вертикали. Выравнивается вложенный в ячейку элемент, а не сама ячейка. Например, в контейнере есть вложенный элемент первого уровня (это ячейка), внутри него есть «текст» или какой-то «div» (текст или div - это реальный элемент) выравнивание элемента выровняет вложенный в ячейку элемент внутри ячейки (размеры ячейки при этом не изменятся).
* **Несколько элементов в одной ячейке**. В одной ячейке или области можно разместить несколько элементов. Чтобы указать кто «выше» (важнее) и кто «ниже» (неважный), нужно использовать css свойство z-index:.
* **Расширение сетки**. Сколько колонок/рядов имеет сетка обычно указывается сразу, но если разместить элемент за пределами сетки (указать ему номер ряда/ячейки, который выходит за пределы сетки), тогда сетка автоматически расширяется, и создаются дополнительные линии (колонки/ряды).

## CSS свойства Grid

### Для контейнера

*grid-template-rows:*

*grid-template-columns:*

Указывают из скольких рядов (строк) и скольких колонок состоит сетка и какие у них размеры. Т.е. указывается сразу два понятия: сколько и какой размер.

В значении через пробелы указываются размеры: высота ряда (rows) или ширина колонки (columns). Сколько раз будет указан размер, столько будет рядов/колонок.

// синтаксис:

**grid-template-rows**: размер размер ...;

**grid-template-columns**: размер размер ...;

**размер** – это высота ряда или ширина колонки, может быть:

* + **auto** – размер ряда/колонки подстраивается под размеры элементов, так, чтобы уместился самый большой из них. Не дает сжиматься меньше min-width или min-height самого широкого или высокого элемента соответственно. Не дает растягиваться больше, чем max-content. Если в контейнере есть свободное место, то размер может растянуться до конца контейнера.
  + **px, em, %, vh, vw** – размер [абсолютный](https://drafts.csswg.org/css-values/#absolute-lengths) (px, pt), [относительный](https://drafts.csswg.org/css-values/#relative-lengths) (em, vw, vh) или в % от ширины/высоты контейнера.
  + **fr** (фракция - свободное место в сетке) – специальная единица измерения в grid. Свободное место в контейнере делится на фракции, так если одной колонке указать 1fr, а другой 2fr, то вторая будет больше первой в 2 раза и обе они заполнят все свободное пространство. Можно указывать дробные значения: 0.5fr, 2.3fr.
  + **min-content** – наименьший размер контента. Для текста это ширина самого длинного слова или неразрывного фрагмента.
  + **max-content** – наибольший размер контента. Для текста это длина самой большой строки без переносов.
  + **fit-content( max )** – функция которой передается макс. размер. Если контент меньше этого размера, ведет себя как auto, если больше, то ограничивает размер ряда/колонки до указанного в параметре max.
  + [**minmax( min, max )**](https://wp-kama.ru/id_8945/grid-v-css.html#minmax-funktsiya) – функция, позволяет разом указать минимальный и максимальный размер.

У двух этих свойств есть сокращенные записи:

* [grid-template: grid-template-rows / grid-template-columns](https://wp-kama.ru/id_8945/grid-v-css.html" \l "grid-template)
* [grid: grid-template-rows / grid-template-columns](https://wp-kama.ru/id_8945/grid-v-css.html" \l "grid)

*grid-template-areas:*

Позволяет создать визуальный шаблон сетки. В этом свойстве задаются имена ячейкам, а затем элементы привязываются к этим именам через свойство [grid-area:](https://wp-kama.ru/id_8945/grid-v-css.html" \l "grid-area) указываемое для отдельного элемента.

Синтаксис визуально показывает, как выглядит сетка:

**grid-template-areas**: "имя имя2 имя3"

"имя имя4 имя5"

"имя none .";

// или можно так

**grid-template-areas**: "имя имя2 имя3" "имя имя4 имя5" "имя6 none .";

// или одинарные кавычки

**grid-template-areas**: 'имя имя2 имя3' 'имя имя4 имя5' 'имя6 none .';

* "имя имя2 имя3" – в значении внутри кавычек нужно через пробелы указывать имена. Каждые кавычки с именами будут представлять собой ряд сетки, а имена внутри кавычек задают имена ячейкам внутри этого ряда.
* "имя имя имя2" – если указать одно и тоже имя несколько раз подряд, то имя объединит ячейки и мы получим область (большую ячейку). Объединять ячейки таким способом можно не только внутри ряда, но и между рядами.
* . (точка) – указывается вместо имени и обозначает ячейку, которую нужно пропустить (пустую ячейку). Можно использовать несколько точек подряд, пока между ними нет пробела они будут считаться за одну.
* none – область не определена

*grid-template:*

Позволяет разом указать три свойства: grid-template-rows, grid-template-columns и grid-template-areas.

// синтаксис:

**grid-template**: none;

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

repeat() – функция

Позволяет повторять что-либо N раз. Используется при создании колонок/рядов в свойствах: grid-template-rows:, grid-template-columns:, grid-template:.

repeat( сколько\_раз, что\_повторять )

Варианты записи repeat():

1. repeat(2, 50px) // 50px 50px
2. repeat( 2, 50px 70px ) // 50px 70px 50px 70px

Количество повторов задается через:

* + **auto-fill** – повторяет ряд/колонку пока есть место в контейнере. Хотя бы одно повторение будет всегда.
    - если контейнер ограничен максимальным размером, ряд/колонка повторяется пока есть место вставить очередной ряд/колонку.
    - если у контейнера задан минимальный размер, ряд/колонка повторяется пока не превысят этот минимум.
  + **auto-fit** – то же самое, но после размещения элементов оставшиеся пустыми ряд/колонка сжимаются и исчезают, в результате контейнер всегда будет выглядеть заполненным (без пустых мест на краях) (это видно визуально только если в контейнере есть резиновый блок).

minmax() – функция

Позволяет задавать минимальное и максимальное значения для ряда/колонки. Используется в свойствах, где нужно задавать размер: grid-template-rows:, grid-template-columns:, grid-template:.

minmax( мин\_значение, макс\_значение )

Значение может быть:

* **px, em, rem, ...** – единицы длины (100px)
* **%** – проценты (10%)
* **fr** – гибкие размеры. Может использоваться только для макс. значения.
* **max-content** – наименьший возможный размер ячейки, при котором содержимое свободно в ней умещается. Например, если в ячейке текст, идеальной шириной ячейки будет вся длина текста (в одну строку без переносов).
* **min-content** – наименьший размер, при котором ячейка не переполняется. Например, если в ячейке текст, то ширина будет равна самому длинному слову в тексте.
* **auto** – зависит от того, используется оно как максимальное или минимальное значение в функции minmax():
  + если в качестве максимума, то тоже самое что max-content.
  + если в качестве минимума, то значение будет минимальным возможным размером для ячейки. Такой размер отличается от min-content и задается свойствами min-width или min-height.

grid-row-gap: задает разрыв между рядами.

grid-column-gap: задает разрыв между колонками.

grid-gap: задает разрыв для рядов и колонок разом. Это сокращение для двух предыдущих свойств. Если указать один параметр он будет выставлен для обоих значений.

// синтаксис:

**grid-row-gap**: размер;

**grid-column-gap**: размер;

**grid-gap**: размер размер; /\* row column \*/

**grid-gap**: размер;

Размер может быть [абсолютным](https://drafts.csswg.org/css-values/#absolute-lengths) (px, pt), [относительным](https://drafts.csswg.org/css-values/#relative-lengths) (%, em).

*align-content:*

*justify-content:*

*place-content:*

Выравнивает ряды/колонки. Выравниваются ячейки сетки, не элементы внутри ячеек. Чтобы выровнять элементы, используйте: justify-items, align-items, place-items.

// синтаксис:

**align-content**: значение; // выравнивает ряды по вертикали ↓↑

**justify-content**: значение; // выравнивает колонки по горизонтали ←→

**place-content**: значение; // сокращенная запись: установит оба значения

**place-content**: align-content justify-content; // сокращенная запись

Значение может быть:

* **stretch (default)** – растягивает ряды/колонки (ячейки) полностью. Все пространство контейнера заполняется. Имеет смысл только если у рядов/колонок размер не задан жестко (они резиновые). Если они не резиновые, то работает как start.
* **start** – ряды/колонки упакованы вплотную друг к другу к начальному краю контейнера.
* **end** – ряды/колонки упакованы вплотную друг к другу к конечному краю контейнера.
* **center** – ряды/колонки упакованы вплотную друг к другу и находятся по середине контейнера.
* **space-around** – свободное пространство равномерно распределяется между рядами/колонками и добавляется по краям. Получается, что крайние ряд/колонка не прижимаются к краям контейнера, но расстояние до края в два раза меньше, чем между рядами/колонками.
* **space-evenly** – то же что и space-around, только расстояние до краев контейнера такое же, как и между рядами/колонками.
* **space-between** – крайние ряд/колонка прижимаются к краям контейнера, и свободное место равномерно распределяется между рядами/колонками.

*align-items:*

*justify-items:*

*place-items:*

Выравнивает элементы сетки - то, что находится внутри ячеек сетки. Срабатывает для всех элементов grid. Чтобы выровнять сами ячейки (ряды колонки), используйте: justify-content, align-content, place-content.

// синтаксис:

**align-items**: значение; // выравнивает элементы по вертикали ↓↑

**justify-items**: значение; // выравнивает элементы по горизонтали ←→

**place-items**: значение; // сокращенная запись: установит оба значения

**place-items**: align-items justify-items; // сокращенная запись

Значение может быть:

* **auto (default)** – указывает использовать значение justify-items:, которое задается для контейнера. Если элемент позиционирован через absolute, то применяется normal.
* **stretch** – растягивает все элементы на всю ширину/высоту ячеек.
* **start** – размещает все элементы в начале ячеек (слева или сверху).
* **end** – размещает все элементы в конце ячеек (справа или внизу).
* **center** – размещает все элементы по центру ячеек.

### Для элементов

*grid-row-start:*

*grid-row-end:*

*grid-column-start:*

*grid-column-end:*

*grid-row:*

*grid-column:*

Указывает положение элемента в сетке. Т.е. размещает элемент в указанную ячейку. Указывать нужно имя или номер линии, к которой относится ячейка и к которой нужно прикрепить элемент.

grid-column и grid-row – это сокращение для свойств: grid-column-start/grid-column-end и grid-row-start / grid-row-end. Можно указать только первое (одно) значение, оно будет относиться к начальной линии, и элемент будет растягиваться на 1 ряд/колонку (т.е. будет помещен в одну ячейку, начальная линия которой указана).

// синтаксис:

**grid-row-start**: значение; // где начинается линия ряда

**grid-row-end**: значение; // где кончается линия ряда

**grid-column-start**: значение; // где начинается линия колонки

**grid-column-end**: значение; // где кончается линия колонки

**grid-row**: grid-row-start / grid-row-end;

**grid-column**: grid-column-start / grid-column-end;

// можно указать одно значение, второе значение будет span 1

**grid-row**: grid-row-start;

**grid-column**: grid-column-start;

Значение может быть:

* **число/имя** – порядковый номер или имя линии, к которой нужно прикрепить текущий элемент.
* **span число** – слово span значит растянуть. Текущий элемент будет растягиваться на указанное число рядов/колонок. Если указано слово span, то относится всегда ко второму значению.
* **span имя** – слово span значит растянуть. Текущий элемент будет растягиваться до указанного названия линии ряда/колонки. Если указано слово span, то относится всегда ко второму значению.
* **auto** – элемент размещается по указанному алгоритму в свойстве контейнера [grid-auto-flow:](https://wp-kama.ru/id_8945/grid-v-css.html" \l "grid-auto-flow).

Если не установлены свойства grid-column-end/grid-row-end элемент по умолчанию будет растягиваться на 1 ряд/колонку.

Элементы могут перекрывать друг друга, в таких случаях приоритет можно выставить через z-index:.

*grid-area:*

Дает элементу имя. По имени элемент будет относиться к области, указанной в свойстве [grid-template-areas:](https://wp-kama.ru/id_8945/grid-v-css.html" \l "grid-template-areas). Или в значении можно указать номер/имя линий.

// синтаксис:

**grid-area**: имя области;

**grid-area**: row-start / column-start / row-end / column-end;

* **имя области** – название области сетки.
* **row-start / column-start / row-end / column-end** – может быть числом или именем линии.

*align-self:*

*justify-self:*

*place-self:*

Выравнивает текущий элемент внутри ячейки. Применяется к отдельному элементу контейнера.

// синтаксис:

**align-self**: значение; // выравнивает элемент внутри ячейки по вертикали ↓↑

**justify-self**: значение; // выравнивает элемент внутри ячейки по горизонтали ←→

**place-self**: значение; // сокращенная запись: установит оба значения

**place-self**: align-self justify-self; // сокращенная запись

Значение может быть:

* **stretch (defailt)** – растягивает текущий элемент на всю ширину/высоту ячейки.
* **start** – размещает текущий элемент в начале ячейки (слева или сверху).
* **end** – размещает текущий элемент в конце ячейки (справа или внизу).
* **center** – размещает текущий элемент по центру ячейки.

## Сравнение Flex и Grid

В отличие от Flex, которая ориентирована на одну ось, Grid оптимизирована для двумерных компоновок: когда требуется расположить (выровнять) содержимое в обоих измерениях (по вертикали и горизонтали).

Кроме того, благодаря возможности явного позиционирования элементов в сетке, Grid позволяет выполнять кардинальные преобразования в структуре, не требуя никаких изменений HTML-разметки. Комбинируя медиа-запросы со свойствами CSS, управляющими компоновкой контейнера grid и его дочерних элементов, можно адаптировать верстку под любые форм-факторы устройств.

Grid и Flexbox, имеют свои особенности и нельзя сказать, что одно заменяет другое.

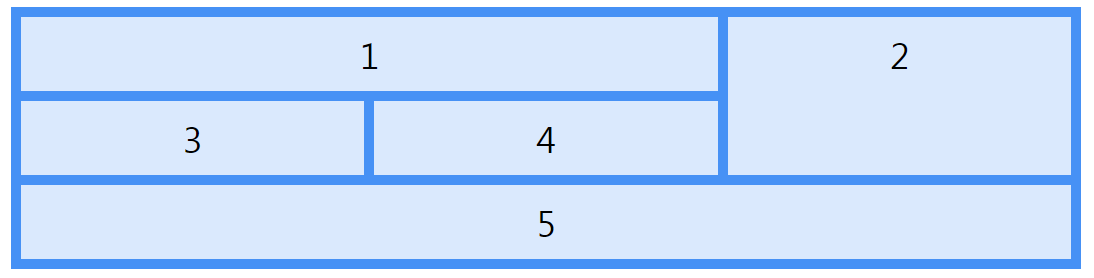
Flexbox фокусируется на распределении пространства внутри одной оси, использует более простой подход к компоновке, может использовать систему упаковки строк на основе размера содержимого для управления своей вторичной осью и полагается на иерархию разметки. Тогда как Grid больше подходит для создания каркасов, потому что имеет более мощный и комплексный подход и в целом не зависит от иерархии разметки. В отдельных случаях Grid позволяет создать адаптивный макет, который невозможно создать с помощью Flex или как-то еще.

**2. Выполните самостоятельные задания**:

**1.** Создайте веб-страницу, в которой текст будет менять цвет в зависимости от размера (ширины) экрана. Контрольные точки – 480рх, 800рх. Используйте медиа-запросы. Цвета выберите сами. Файл сохраните под именем **LR6-1.html**.

Дополните веб-страницу адаптацией размера текста по отношению к размеру экрана. Контрольные точки – те же. После прохождения контрольной точки размер шрифта изменяется в 2 раза (при уменьшении экрана – уменьшается, при увеличении – увеличивается). Файл сохраните под именем **LR6-1-1.html**.

1. Создайте веб-страницу **LR6-2.html** с использованием Grid-сетки по образцу. Цветовое оформление и содержимое элементов может быть другим.



1. Создайте веб-страницу с использованием Grid-сетки на основе макета, соответствующего номеру вашего варианта. Цветовое оформление и содержимое элементов продумайте сами. Файл сохраните под именем **LR6-3.html**.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

1. **Список контрольных вопросов**
   * + 1. Что такое медиа-запрос?
       2. Как записывается медиа-запрос?
       3. Что такое контрольная точка?
       4. Что такое grid?
       5. Из чего состоит grid-сетка?
       6. Как задать размеры grid**-контейнера?**
       7. **Какие единицы измерения используются при создании** grid**-контейнера?**
       8. **Назначение и синтаксис функции**repeat()?
       9. **Чем отличаются свойства** grid**-контейнера и** grid-элемента?
       10. Что такое именованная область? Как задать имя области?